



# OpenShift Container Platform 4.18

## Networking

Configuring and managing cluster networking



# OpenShift Container Platform 4.18 Networking

---

Configuring and managing cluster networking

## Legal Notice

Copyright © 2025 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This document provides instructions for configuring and managing your OpenShift Container Platform cluster network, including DNS, ingress, and the Pod network.

## Table of Contents

<b>CHAPTER 1. UNDERSTANDING NETWORKING .....</b>	<b>23</b>
1.1. NETWORKING IN OPENSPLIT CONTAINER PLATFORM	23
1.1.1. Common practices for networking services	23
1.1.2. Networking features	23
1.2. NETWORKING WITH NODES, CLIENTS, AND CLUSTERS	24
1.2.1. What is a node?	24
1.2.2. Understanding clusters	25
1.2.3. Understanding external clients	25
1.3. NETWORKING CONCEPTS AND COMPONENTS	25
1.4. HOW PODS COMMUNICATE	26
1.4.1. Pod-to-pod communication	26
1.4.1.1. Example: Controlling pod-to-pod communication	26
1.4.2. Service-to-pod communication	27
1.4.2.1. Example: Controlling service-to-pod communication	28
1.5. SUPPORTED LOAD BALANCERS	29
1.5.1. Configuring Load balancers	29
1.5.1.1. Define the default load balancer type	29
1.5.1.2. Specify load balancer behavior for an Ingress Controller	30
1.6. THE DOMAIN NAME SYSTEM (DNS)	30
1.6.1. Key DNS terms	30
1.6.2. Example: DNS use case	31
1.7. NETWORK CONTROLS	32
1.8. ROUTES AND INGRESS	33
1.8.1. Routes	33
1.8.2. Ingress	33
1.8.3. Comparing Routes and ingress	33
1.8.4. Example: Configuring routes and ingress to expose a web application	34
1.8.4.1. Configuring Routes	34
1.8.4.2. Configuring ingress	34
1.9. SECURITY AND TRAFFIC MANAGEMENT	36
1.9.1. Exposing applications	36
1.9.2. Securing connections	36
1.9.3. Example: Exposing applications and securing connections	37
1.9.4. Choosing between service types and API resources	38
<b>CHAPTER 2. ACCESSING HOSTS .....</b>	<b>40</b>
2.1. ACCESSING HOSTS ON AMAZON WEB SERVICES IN AN INSTALLER-PROVISIONED INFRASTRUCTURE CLUSTER	40
<b>CHAPTER 3. NETWORKING DASHBOARDS .....</b>	<b>41</b>
3.1. NETWORK OBSERVABILITY OPERATOR	41
3.2. NETWORKING AND OVN-KUBERNETES DASHBOARD	41
3.3. INGRESS OPERATOR DASHBOARD	41
<b>CHAPTER 4. NETWORKING OPERATORS .....</b>	<b>42</b>
4.1. KUBERNETES NMSTATE OPERATOR	42
4.1.1. Installing the Kubernetes NMState Operator	43
4.1.1.1. Installing the Kubernetes NMState Operator by using the web console	43
4.1.1.2. Installing the Kubernetes NMState Operator by using the CLI	43
4.1.1.3. Viewing metrics collected by the Kubernetes NMState Operator	45
4.1.2. Uninstalling the Kubernetes NMState Operator	47
4.1.3. Additional resources	49

4.1.4. Next steps	49
<b>4.2. AWS LOAD BALANCER OPERATOR</b>	49
4.2.1. AWS Load Balancer Operator release notes	49
4.2.1.1. AWS Load Balancer Operator 1.2.0	49
4.2.1.1.1. Notable changes	49
4.2.1.1.2. AWS Load Balancer Operator 1.1.1	49
4.2.1.1.3. AWS Load Balancer Operator 1.1.0	49
4.2.1.1.3.1. Notable changes	50
4.2.1.1.3.2. New features	50
4.2.1.1.3.3. Bug fixes	50
4.2.1.4. AWS Load Balancer Operator 1.0.1	50
4.2.1.5. AWS Load Balancer Operator 1.0.0	50
4.2.1.5.1. Notable changes	50
4.2.1.5.2. Bug fixes	50
4.2.1.6. Earlier versions	51
4.2.2. AWS Load Balancer Operator in OpenShift Container Platform	51
4.2.2.1. AWS Load Balancer Operator considerations	51
4.2.2.2. AWS Load Balancer Operator	51
4.2.2.3. Using the AWS Load Balancer Operator in an AWS VPC cluster extended into an Outpost	52
4.2.3. Preparing an AWS STS cluster for the AWS Load Balancer Operator	53
4.2.3.1. Prerequisites	53
4.2.3.2. Creating an IAM role for the AWS Load Balancer Operator	54
4.2.3.2.1. Creating an AWS IAM role by using the Cloud Credential Operator utility	54
4.2.3.2.2. Creating an AWS IAM role by using the AWS CLI	55
4.2.3.3. Configuring the ARN role for the AWS Load Balancer Operator	56
4.2.3.4. Creating an IAM role for the AWS Load Balancer Controller	57
4.2.3.4.1. Creating an AWS IAM role for the controller by using the Cloud Credential Operator utility	58
4.2.3.4.2. Creating an AWS IAM role for the controller by using the AWS CLI	58
4.2.3.5. Additional resources	60
4.2.4. Installing the AWS Load Balancer Operator	60
4.2.4.1. Installing the AWS Load Balancer Operator by using the web console	60
4.2.4.2. Installing the AWS Load Balancer Operator by using the CLI	61
4.2.4.3. Creating the AWS Load Balancer Controller	63
4.2.5. Configuring the AWS Load Balancer Operator	66
4.2.5.1. Trusting the certificate authority of the cluster-wide proxy	66
4.2.5.2. Adding TLS termination on the AWS Load Balancer	67
4.2.5.3. Creating multiple ingress resources through a single AWS Load Balancer	68
4.2.5.4. AWS Load Balancer Operator logs	71
<b>4.3. eBPF MANAGER OPERATOR</b>	72
4.3.1. About the eBPF Manager Operator	72
4.3.1.1. About Extended Berkeley Packet Filter (eBPF)	72
4.3.1.2. About the eBPF Manager Operator	72
4.3.1.3. Additional resources	73
4.3.1.4. Next steps	73
4.3.2. Installing the eBPF Manager Operator	73
4.3.2.1. Installing the eBPF Manager Operator using the CLI	73
4.3.2.2. Installing the eBPF Manager Operator using the web console	75
4.3.2.3. Next steps	76
4.3.3. Deploying an eBPF program	76
4.3.3.1. Deploying a containerized eBPF program	76
<b>4.4. EXTERNAL DNS OPERATOR</b>	78
4.4.1. External DNS Operator release notes	78
4.4.1.1. External DNS Operator 1.3.0	78

4.4.1.1. Bug fixes	78
4.4.1.2. External DNS Operator 1.2.0	78
4.4.1.2.1. New features	79
4.4.1.2.2. Bug fixes	79
4.4.1.3. External DNS Operator 1.1.1	79
4.4.1.4. External DNS Operator 1.1.0	79
4.4.1.4.1. Bug fixes	79
4.4.1.5. External DNS Operator 1.0.1	79
4.4.1.6. External DNS Operator 1.0.0	79
4.4.1.6.1. Bug fixes	79
4.4.2. Understanding the External DNS Operator	79
4.4.2.1. External DNS Operator	80
4.4.2.2. Viewing External DNS Operator logs	80
4.4.2.2.1. External DNS Operator domain name limitations	80
4.4.3. Installing the External DNS Operator	81
4.4.3.1. Installing the External DNS Operator with OperatorHub	81
4.4.3.2. Installing the External DNS Operator by using the CLI	82
4.4.4. External DNS Operator configuration parameters	84
4.4.4.1. External DNS Operator configuration parameters	84
4.4.5. Creating DNS records on AWS	86
4.4.5.1. Creating DNS records on an public hosted zone for AWS by using Red Hat External DNS Operator	86
4.4.5.2. Creating DNS records in a different AWS Account using a shared VPC	88
4.4.6. Creating DNS records on Azure	90
4.4.6.1. Creating DNS records on an Azure public DNS zone	90
4.4.7. Creating DNS records on GCP	92
4.4.7.1. Creating DNS records on a public managed zone for GCP	92
4.4.8. Creating DNS records on Infoblox	94
4.4.8.1. Creating DNS records on a public DNS zone on Infoblox	94
4.4.9. Configuring the cluster-wide proxy on the External DNS Operator	95
4.4.9.1. Trusting the certificate authority of the cluster-wide proxy	95
4.5. METALLB OPERATOR	96
4.5.1. About MetalLB and the MetalLB Operator	96
4.5.1.1. When to use MetalLB	96
4.5.1.2. MetalLB Operator custom resources	97
4.5.1.3. MetalLB software components	97
4.5.1.4. MetalLB and external traffic policy	98
4.5.1.5. MetalLB concepts for layer 2 mode	99
4.5.1.6. MetalLB concepts for BGP mode	101
4.5.1.7. Limitations and restrictions	102
4.5.1.7.1. Infrastructure considerations for MetalLB	102
4.5.1.7.2. Limitations for layer 2 mode	103
4.5.1.7.2.1. Single-node bottleneck	103
4.5.1.7.2.2. Slow failover performance	103
4.5.1.7.2.3. Additional Network and MetalLB cannot use same network	103
4.5.1.7.3. Limitations for BGP mode	103
4.5.1.7.3.1. Node failure can break all active connections	104
4.5.1.7.3.2. Support for a single ASN and a single router ID only	104
4.5.1.8. Additional resources	104
4.5.2. Installing the MetalLB Operator	104
4.5.2.1. Installing the MetalLB Operator from the OperatorHub using the web console	104
4.5.2.2. Installing from OperatorHub using the CLI	105
4.5.2.3. Starting MetalLB on your cluster	107

4.5.2.4. Deployment specifications for MetalLB	108
4.5.2.4.1. Limit speaker pods to specific nodes	108
4.5.2.4.2. Configuring pod priority and pod affinity in a MetalLB deployment	109
4.5.2.4.3. Configuring pod CPU limits in a MetalLB deployment	111
4.5.2.5. Additional resources	112
4.5.2.6. Next steps	112
4.5.3. Upgrading the MetalLB Operator	112
4.5.3.1. Manually upgrading the MetalLB Operator	112
4.5.3.2. Additional resources	114
4.6. CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM	114
4.6.1. Cluster Network Operator	114
4.6.2. Viewing the cluster network configuration	115
4.6.3. Viewing Cluster Network Operator status	116
4.6.4. Enabling IP forwarding globally	116
4.6.5. Viewing Cluster Network Operator logs	117
4.6.6. Cluster Network Operator configuration	117
4.6.6.1. Cluster Network Operator configuration object	118
defaultNetwork object configuration	119
Configuration for the OVN-Kubernetes network plugin	119
4.6.6.2. Cluster Network Operator example configuration	124
4.6.7. Additional resources	124
4.7. DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM	125
4.7.1. Checking the status of the DNS Operator	125
4.7.2. View the default DNS	125
4.7.3. Using DNS forwarding	126
4.7.4. Checking DNS Operator status	128
4.7.5. Viewing DNS Operator logs	129
4.7.6. Setting the CoreDNS log level	129
4.7.7. Viewing the CoreDNS logs	130
4.7.8. Setting the CoreDNS Operator log level	130
4.7.9. Tuning the CoreDNS cache	131
4.7.10. Advanced tasks	132
4.7.10.1. Changing the DNS Operator managementState	132
4.7.10.2. Controlling DNS pod placement	133
4.7.10.3. Configuring DNS forwarding with TLS	134
4.8. INGRESS OPERATOR IN OPENSHIFT CONTAINER PLATFORM	137
4.8.1. OpenShift Container Platform Ingress Operator	137
4.8.2. The Ingress configuration asset	137
4.8.3. Ingress Controller configuration parameters	137
4.8.3.1. Ingress Controller TLS security profiles	148
4.8.3.1.1. Understanding TLS security profiles	148
4.8.3.1.2. Configuring the TLS security profile for the Ingress Controller	149
4.8.3.1.3. Configuring mutual TLS authentication	151
4.8.3.4. View the default Ingress Controller	153
4.8.3.5. View Ingress Operator status	153
4.8.3.6. View Ingress Controller logs	153
4.8.3.7. View Ingress Controller status	153
4.8.3.8. Creating a custom Ingress Controller	154
4.8.3.9. Configuring the Ingress Controller	154
4.8.3.9.1. Setting a custom default certificate	155
4.8.3.9.2. Removing a custom default certificate	156
4.8.3.9.3. Autoscaling an Ingress Controller	157
4.8.3.9.4. Scaling an Ingress Controller	161

4.8.9.5. Configuring Ingress access logging	162
4.8.9.6. Setting Ingress Controller thread count	165
4.8.9.7. Configuring an Ingress Controller to use an internal load balancer	166
4.8.9.8. Configuring global access for an Ingress Controller on GCP	168
4.8.9.9. Setting the Ingress Controller health check interval	169
4.8.9.10. Configuring the default Ingress Controller for your cluster to be internal	169
4.8.9.11. Configuring the route admission policy	170
4.8.9.12. Using wildcard routes	171
4.8.9.13. HTTP header configuration	171
4.8.9.13.1. Order of precedence	171
4.8.9.13.2. Special case headers	173
4.8.9.14. Setting or deleting HTTP request and response headers in an Ingress Controller	174
4.8.9.15. Using X-Forwarded headers	176
Example use cases	176
4.8.9.16. Enable or disable HTTP/2 on Ingress Controllers	177
4.8.9.16.1. Enabling HTTP/2	178
4.8.9.16.2. Disabling HTTP/2	178
4.8.9.17. Configuring the PROXY protocol for an Ingress Controller	179
4.8.9.18. Specifying an alternative cluster domain using the appsDomain option	181
4.8.9.19. Converting HTTP header case	182
4.8.9.20. Using router compression	184
4.8.9.21. Exposing router metrics	185
4.8.9.22. Customizing HAProxy error code response pages	187
4.8.9.23. Setting the Ingress Controller maximum connections	189
4.8.10. Additional resources	189
<b>4.9. INGRESS NODE FIREWALL OPERATOR IN OPENSHIFT CONTAINER PLATFORM</b>	189
4.9.1. Ingress Node Firewall Operator	189
4.9.2. Installing the Ingress Node Firewall Operator	190
4.9.2.1. Installing the Ingress Node Firewall Operator using the CLI	190
4.9.2.2. Installing the Ingress Node Firewall Operator using the web console	191
4.9.3. Deploying Ingress Node Firewall Operator	192
4.9.3.1. Ingress Node Firewall configuration object	193
Ingress Node Firewall Operator example configuration	193
4.9.3.2. Ingress Node Firewall rules object	194
Ingress object configuration	194
Ingress Node Firewall rules object example	195
Zero trust Ingress Node Firewall rules object example	196
4.9.4. Ingress Node Firewall Operator integration	197
4.9.5. Configuring Ingress Node Firewall Operator to use the eBPF Manager Operator	198
4.9.6. Viewing Ingress Node Firewall Operator rules	199
4.9.7. Troubleshooting the Ingress Node Firewall Operator	199
4.9.8. Additional resources	199
<b>4.10. SR-IOV OPERATOR</b>	200
4.10.1. Installing the SR-IOV Network Operator	200
4.10.1.1. Installing the SR-IOV Network Operator	200
4.10.1.1.1. CLI: Installing the SR-IOV Network Operator	200
4.10.1.1.2. Web console: Installing the SR-IOV Network Operator	201
4.10.1.2. Next steps	202
4.10.2. Configuring the SR-IOV Network Operator	202
4.10.2.1. Configuring the SR-IOV Network Operator	203
4.10.2.1.1. SR-IOV Network Operator config custom resource	203
4.10.2.1.2. About the Network Resources Injector	205
4.10.2.1.3. Disabling or enabling the Network Resources Injector	205

4.10.2.1.4. About the SR-IOV Network Operator admission controller webhook	206
4.10.2.1.5. Disabling or enabling the SR-IOV Network Operator admission controller webhook	207
4.10.2.1.6. About custom node selectors	207
4.10.2.1.7. Configuring a custom NodeSelector for the SR-IOV Network Config daemon	207
4.10.2.1.8. Configuring the SR-IOV Network Operator for single node installations	208
4.10.2.1.9. Deploying the SR-IOV Operator for hosted control planes	209
4.10.2.2. About the SR-IOV network metrics exporter	210
4.10.2.2.1. Enabling the SR-IOV network metrics exporter	212
4.10.2.3. Next steps	213
4.10.3. Uninstalling the SR-IOV Network Operator	213
4.10.3.1. Uninstalling the SR-IOV Network Operator	213
<b>CHAPTER 5. NETWORK SECURITY .....</b>	<b>215</b>
5.1. UNDERSTANDING NETWORK POLICY APIs	215
5.1.1. Key differences between AdminNetworkPolicy and NetworkPolicy custom resources	215
5.2. ADMIN NETWORK POLICY	216
5.2.1. OVN-Kubernetes AdminNetworkPolicy	216
5.2.1.1. AdminNetworkPolicy	217
AdminNetworkPolicy example	217
5.2.1.1.1. AdminNetworkPolicy actions for rules	218
AdminNetworkPolicy Allow example	218
AdminNetworkPolicy Deny example	219
AdminNetworkPolicy Pass example	219
5.2.2. OVN-Kubernetes BaselineAdminNetworkPolicy	220
5.2.2.1. BaselineAdminNetworkPolicy	220
BaselineAdminNetworkPolicy example	220
BaselineAdminNetworkPolicy Deny example	221
5.2.3. Monitoring ANP and BANP	222
5.2.3.1. Metrics for AdminNetworkPolicy	223
5.2.4. Egress nodes and networks peer for AdminNetworkPolicy	223
5.2.4.1. Northbound traffic controls for AdminNetworkPolicy and BaselineAdminNetworkPolicy	223
5.2.4.1.1. Using nodes peer to control egress traffic to cluster nodes	224
5.2.4.1.2. Using networks peer to control egress traffic towards external destinations	225
5.2.4.1.3. Using nodes peer and networks peer together	227
5.2.5. Troubleshooting AdminNetworkPolicy	228
5.2.5.1. Checking creation of ANP	228
5.2.5.1.1. Using nbctl commands for ANP and BANP	229
5.2.5.2. Additional resources	241
5.2.6. Best practices for AdminNetworkPolicy	241
5.2.6.1. Designing AdminNetworkPolicy	241
5.2.6.1.1. Considerations for using BaselineAdminNetworkPolicy	242
5.2.6.1.2. Differences to consider between AdminNetworkPolicy and NetworkPolicy	242
5.3. NETWORK POLICY	243
5.3.1. About network policy	243
5.3.1.1. About network policy	243
5.3.1.1.1. Using the allow-from-router network policy	245
5.3.1.1.2. Using the allow-from-hostnetwork network policy	246
5.3.1.2. Optimizations for network policy with OVN-Kubernetes network plugin	246
5.3.1.2.1. NetworkPolicy CR and external IPs in OVN-Kubernetes	248
5.3.1.3. Next steps	248
5.3.1.4. Additional resources	249
5.3.2. Creating a network policy	249
5.3.2.1. Example NetworkPolicy object	249

5.3.2.2. Creating a network policy using the CLI	249
5.3.2.3. Creating a default deny all network policy	251
5.3.2.4. Creating a network policy to allow traffic from external clients	253
5.3.2.5. Creating a network policy allowing traffic to an application from all namespaces	254
5.3.2.6. Creating a network policy allowing traffic to an application from a namespace	256
5.3.2.7. Additional resources	258
5.3.3. Viewing a network policy	258
5.3.3.1. Example NetworkPolicy object	258
5.3.3.2. Viewing network policies using the CLI	259
5.3.4. Editing a network policy	260
5.3.4.1. Editing a network policy	260
5.3.4.2. Example NetworkPolicy object	262
5.3.4.3. Additional resources	262
5.3.5. Deleting a network policy	262
5.3.5.1. Deleting a network policy using the CLI	262
5.3.6. Defining a default network policy for projects	263
5.3.6.1. Modifying the template for new projects	263
5.3.6.2. Adding network policies to the new project template	265
5.3.7. Configuring multitenant isolation with network policy	266
5.3.7.1. Configuring multitenant isolation by using network policy	266
5.3.7.2. Next steps	269
5.4. AUDIT LOGGING FOR NETWORK SECURITY	269
5.4.1. Audit configuration	269
5.4.2. Audit logging	270
5.4.3. AdminNetworkPolicy audit logging	273
5.4.4. BaselineAdminNetworkPolicy audit logging	276
5.4.5. Configuring egress firewall and network policy auditing for a cluster	279
5.4.6. Enabling egress firewall and network policy audit logging for a namespace	283
5.4.7. Disabling egress firewall and network policy audit logging for a namespace	284
5.4.8. Additional resources	285
5.5. EGRESS FIREWALL	285
5.5.1. Viewing an egress firewall for a project	285
5.5.1.1. Viewing an EgressFirewall object	285
5.5.2. Editing an egress firewall for a project	286
5.5.2.1. Editing an EgressFirewall object	286
5.5.3. Removing an egress firewall from a project	286
5.5.3.1. Removing an EgressFirewall object	287
5.5.4. Configuring an egress firewall for a project	287
5.5.4.1. How an egress firewall works in a project	287
5.5.4.1.1. Limitations of an egress firewall	289
5.5.4.1.2. Matching order for egress firewall policy rules	289
5.5.4.1.3. How Domain Name Server (DNS) resolution works	289
5.5.4.1.3.1. Improved DNS resolution and resolving wildcard domain names	289
5.5.4.1.4. EgressFirewall custom resource (CR) object	291
5.5.4.1.4.1. EgressFirewall rules	291
5.5.4.1.4.2. Example EgressFirewall CR objects	292
5.5.4.1.4.3. Example nodeSelector for EgressFirewall	293
5.5.4.1.5. Creating an egress firewall policy object	293
5.6. CONFIGURING IPSEC ENCRYPTION	294
5.6.1. Modes of operation	295
5.6.2. Prerequisites	295
5.6.3. Network connectivity requirements when IPsec is enabled	295
5.6.4. IPsec encryption for pod-to-pod traffic	296

5.6.4.1. Types of network traffic flows encrypted by pod-to-pod IPsec	296
5.6.4.2. Encryption protocol and IPsec mode	297
5.6.4.3. Security certificate generation and rotation	297
5.6.5. IPsec encryption for external traffic	297
5.6.5.1. Supported platforms	297
5.6.5.2. Limitations	298
5.6.6. Enabling IPsec encryption	298
5.6.7. Configuring IPsec encryption for external traffic	299
5.6.8. Disabling IPsec encryption for an external IPsec endpoint	304
5.6.9. Disabling IPsec encryption	305
5.6.10. Additional resources	306
5.7. ZERO TRUST NETWORKING	306
5.7.1. Root of trust	306
5.7.2. Traffic authentication and encryption	306
5.7.3. Identification and authentication	307
5.7.4. Inter-service authorization	307
5.7.5. Transaction-level verification	307
5.7.6. Risk assessment	308
5.7.7. Site-wide policy enforcement and distribution	308
5.7.8. Observability for constant, and retrospective, evaluation	308
5.7.9. Endpoint security	309
5.7.10. Extending trust outside of the cluster	309
<b>CHAPTER 6. CONFIGURING AN INGRESS CONTROLLER FOR MANUAL DNS MANAGEMENT .....</b>	<b>310</b>
6.1. MANAGED DNS MANAGEMENT POLICY	310
6.2. UNMANAGED DNS MANAGEMENT POLICY	310
6.3. CREATING A CUSTOM INGRESS CONTROLLER WITH THE UNMANAGED DNS MANAGEMENT POLICY	310
6.4. MODIFYING AN EXISTING INGRESS CONTROLLER	311
6.5. ADDITIONAL RESOURCES	312
<b>CHAPTER 7. VERIFYING CONNECTIVITY TO AN ENDPOINT .....</b>	<b>313</b>
7.1. CONNECTION HEALTH CHECKS PERFORMED	313
7.2. IMPLEMENTATION OF CONNECTION HEALTH CHECKS	313
7.3. CONFIGURING POD CONNECTIVITY CHECK PLACEMENT	314
7.4. PODNETWORKCONNECTIVITYCHECK OBJECT FIELDS	315
Connection log fields	317
7.5. VERIFYING NETWORK CONNECTIVITY FOR AN ENDPOINT	318
<b>CHAPTER 8. CHANGING THE MTU FOR THE CLUSTER NETWORK .....</b>	<b>323</b>
8.1. ABOUT THE CLUSTER MTU	323
8.1.1. Service interruption considerations	323
8.1.2. MTU value selection	323
8.1.3. How the migration process works	323
8.2. CHANGING THE CLUSTER NETWORK MTU	325
8.3. ADDITIONAL RESOURCES	331
<b>CHAPTER 9. CONFIGURING THE NODE PORT SERVICE RANGE .....</b>	<b>332</b>
9.1. PREREQUISITES	332
9.2. EXPANDING THE NODE PORT RANGE	332
9.3. ADDITIONAL RESOURCES	333
<b>CHAPTER 10. CONFIGURING THE CLUSTER NETWORK RANGE .....</b>	<b>334</b>
10.1. EXPANDING THE CLUSTER NETWORK IP ADDRESS RANGE	334

10.2. ADDITIONAL RESOURCES	335
<b>CHAPTER 11. CONFIGURING IP FAILOVER .....</b>	<b>336</b>
11.1. IP FAILOVER ENVIRONMENT VARIABLES	337
11.2. CONFIGURING IP FAILOVER IN YOUR CLUSTER	338
11.3. CONFIGURING CHECK AND NOTIFY SCRIPTS	342
11.4. CONFIGURING VRRP PREEMPTION	344
11.5. DEPLOYING MULTIPLE IP FAILOVER INSTANCES	345
11.6. CONFIGURING IP FAILOVER FOR MORE THAN 254 ADDRESSES	345
11.7. HIGH AVAILABILITY FOR EXTERNALIP	346
11.8. REMOVING IP FAILOVER	347
<b>CHAPTER 12. CONFIGURING SYSTEM CONTROLS AND INTERFACE ATTRIBUTES USING THE TUNING PLUGIN .....</b>	<b>350</b>
12.1. CONFIGURING SYSTEM CONTROLS BY USING THE TUNING CNI	350
12.2. ENABLING ALL-MULTICAST MODE BY USING THE TUNING CNI	353
12.3. ADDITIONAL RESOURCES	356
<b>CHAPTER 13. USING THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) .....</b>	<b>357</b>
13.1. SUPPORT FOR SCTP ON OPENSHIFT CONTAINER PLATFORM	357
13.1.1. Example configurations using SCTP protocol	357
13.2. ENABLING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)	358
13.3. VERIFYING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) IS ENABLED	359
<b>CHAPTER 14. USING PTP HARDWARE .....</b>	<b>362</b>
14.1. ABOUT PRECISION TIME PROTOCOL IN OPENSHIFT CLUSTER NODES	362
14.1.1. Elements of a PTP domain	362
Advantages of PTP over NTP	363
14.1.2. Overview of linuxptp and gpsd in OpenShift Container Platform nodes	364
14.1.3. Overview of GNSS timing for PTP grandmaster clocks	365
Handling leap second events in GNSS-synced PTP grandmaster clocks	366
14.1.4. About PTP and clock synchronization error events	366
14.1.5. 2-card E810 NIC configuration reference	366
14.1.6. 3-card Intel E810 PTP grandmaster clock	368
14.2. CONFIGURING PTP DEVICES	369
14.2.1. Installing the PTP Operator using the CLI	370
14.2.2. Installing the PTP Operator by using the web console	371
14.2.3. Discovering PTP-capable network devices in your cluster	372
14.2.4. Configuring linuxptp services as a grandmaster clock	373
14.2.4.1. Configuring linuxptp services as a grandmaster clock for dual E810 NICs	378
14.2.4.2. Configuring linuxptp services as a grandmaster clock for 3 E810 NICs	384
14.2.5. Grandmaster clock PtpConfig configuration reference	391
14.2.5.1. Grandmaster clock class sync state reference	393
14.2.5.2. Intel E810 NIC hardware configuration reference	393
14.2.5.3. Dual E810 NIC configuration reference	395
14.2.5.4. 3-card E810 NIC configuration reference	396
14.2.6. Holdover in a grandmaster clock with GNSS as the source	397
14.2.7. Configuring dynamic leap seconds handling for PTP grandmaster clocks	399
14.2.8. Configuring linuxptp services as a boundary clock	401
14.2.8.1. Configuring linuxptp services as boundary clocks for dual-NIC hardware	407
14.2.8.2. Configuring linuxptp as a highly available system clock for dual-NIC Intel E810 PTP boundary clocks	408
14.2.9. Configuring linuxptp services as an ordinary clock	412
14.2.9.1. Intel Columbiaville E800 series NIC as PTP ordinary clock reference	417

14.2.10. Configuring FIFO priority scheduling for PTP hardware	418
14.2.11. Configuring log filtering for linuxptp services	419
14.2.12. Troubleshooting common PTP Operator issues	420
14.2.13. Getting the DPLL firmware version for the CGU in an Intel 800 series NIC	423
14.2.14. Collecting PTP Operator data	425
<b>14.3. DEVELOPING PTP EVENTS CONSUMER APPLICATIONS WITH THE REST API V2</b>	<b>425</b>
14.3.1. About the PTP fast event notifications framework	425
14.3.2. Retrieving PTP events with the PTP events REST API v2	425
14.3.3. Configuring the PTP fast event notifications publisher	426
14.3.4. PTP events REST API v2 consumer application reference	428
14.3.5. Reference event consumer deployment and service CRs using PTP events REST API v2	430
14.3.6. Subscribing to PTP events with the REST API v2	432
14.3.7. Verifying that the PTP events REST API v2 consumer application is receiving events	432
14.3.8. Monitoring PTP fast event metrics	434
14.3.9. PTP fast event metrics reference	435
PTP fast event metrics only when T-GM is enabled	436
<b>14.4. PTP EVENTS REST API V2 REFERENCE</b>	<b>437</b>
14.4.1. PTP events REST API v2 endpoints	438
14.4.1.1. api/ocloudNotifications/v2/subscriptions	438
HTTP method	438
Description	438
HTTP method	439
Description	439
HTTP method	440
Description	440
14.4.1.2. api/ocloudNotifications/v2/subscriptions/{subscription_id}	441
HTTP method	441
Description	441
HTTP method	441
Description	441
14.4.1.3. api/ocloudNotifications/v2/health	442
HTTP method	442
Description	442
14.4.1.4. api/ocloudNotifications/v2/publishers	442
HTTP method	442
Description	442
14.4.1.5. api/ocloudNotifications/v2/{resource_address}/CurrentState	443
HTTP method	443
Description	443
<b>14.5. DEVELOPING PTP EVENTS CONSUMER APPLICATIONS WITH THE REST API V1</b>	<b>446</b>
14.5.1. About the PTP fast event notifications framework	446
14.5.2. Retrieving PTP events with the PTP events REST API v1	447
14.5.3. Configuring the PTP fast event notifications publisher	448
14.5.4. PTP events consumer application reference	450
14.5.5. Reference cloud-event-proxy deployment and service CRs	451
14.5.6. Subscribing to PTP events with the REST API v1	453
14.5.7. Verifying that the PTP events REST API v1 consumer application is receiving events	453
14.5.8. Monitoring PTP fast event metrics	455
14.5.9. PTP fast event metrics reference	456
PTP fast event metrics only when T-GM is enabled	458
<b>14.6. PTP EVENTS REST API V1 REFERENCE</b>	<b>459</b>
14.6.1. PTP events REST API v1 endpoints	460
14.6.1.1. api/ocloudNotifications/v1/subscriptions	460

HTTP method	460
Description	460
HTTP method	460
Description	460
HTTP method	461
Description	461
14.6.1.2. api/ocloudNotifications/v1/subscriptions/{subscription_id}	461
HTTP method	461
Description	462
HTTP method	462
Description	462
14.6.1.3. api/ocloudNotifications/v1/health	462
HTTP method	462
Description	462
14.6.1.4. api/ocloudNotifications/v1/publishers	463
HTTP method	463
Description	463
14.6.1.5. api/ocloudNotifications/v1/{resource_address}/CurrentState	463
HTTP method	463
Description	464
<b>CHAPTER 15. CIDR RANGE DEFINITIONS .....</b>	<b>467</b>
15.1. MACHINE CIDR	468
15.2. SERVICE CIDR	468
15.3. POD CIDR	468
15.4. HOST PREFIX	468
15.5. CIDR RANGES FOR HOSTED CONTROL PLANES	468
<b>CHAPTER 16. MULTIPLE NETWORKS .....</b>	<b>470</b>
16.1. UNDERSTANDING MULTIPLE NETWORKS	470
16.1.1. Usage scenarios for a secondary network	470
16.1.2. Secondary networks in OpenShift Container Platform	471
16.1.3. UserDefinedNetwork and NetworkAttachmentDefinition support matrix	471
16.2. PRIMARY NETWORKS	474
16.2.1. About user-defined networks	474
16.2.1.1. Benefits of a user-defined network	475
16.2.1.2. Limitations of a user-defined network	475
16.2.1.3. Layer 2 and layer 3 topologies	476
16.2.1.4. About the ClusterUserDefinedNetwork CR	478
16.2.1.4.1. Best practices for ClusterUserDefinedNetwork CRs	479
16.2.1.4.2. Creating a ClusterUserDefinedNetwork CR by using the CLI	480
16.2.1.4.3. Creating a ClusterUserDefinedNetwork CR by using the web console	483
16.2.1.5. About the UserDefinedNetwork CR	484
16.2.1.5.1. Best practices for UserDefinedNetwork CRs	485
16.2.1.5.2. Creating a UserDefinedNetwork CR by using the CLI	486
16.2.1.5.3. Creating a UserDefinedNetwork CR by using the web console	489
16.2.1.6. Additional configuration details for user-defined networks	489
16.2.1.7. User-defined network status condition types	493
16.2.1.8. Opening default network ports on user-defined network pods	494
16.2.2. Creating primary networks using a NetworkAttachmentDefinition	495
16.2.2.1. Approaches to managing a primary network	495
16.2.2.2. Creating a primary network attachment with the Cluster Network Operator	495
16.2.2.2.1. Configuration for a primary network attachment	497

16.2.2.3. Creating a primary network attachment by applying a YAML manifest	497
<b>16.3. SECONDARY NETWORKS</b>	498
16.3.1. Creating secondary networks on OVN-Kubernetes	498
16.3.1.1. Configuration for an OVN-Kubernetes secondary network	498
16.3.1.1.1. Supported platforms for OVN-Kubernetes secondary network	498
16.3.1.1.2. OVN-Kubernetes network plugin JSON configuration table	499
16.3.1.1.3. Compatibility with multi-network policy	500
16.3.1.1.4. Configuration for a localnet switched topology	501
16.3.1.1.4.1. Configuration for a layer 2 switched topology	504
16.3.1.1.5. Configuring pods for secondary networks	504
16.3.1.1.6. Configuring pods with a static IP address	505
16.3.1.2. Creating secondary networks with other CNI plugins	505
16.3.1.2.1. Configuration for a bridge secondary network	505
16.3.1.2.1.1. Bridge CNI plugin configuration example	507
16.3.1.2.2. Configuration for a host device secondary network	507
16.3.1.2.2.1. host-device configuration example	508
16.3.1.2.3. Configuration for a VLAN secondary network	508
16.3.1.2.3.1. VLAN configuration example	509
16.3.1.2.4. Configuration for an IPVLAN secondary network	510
16.3.1.2.4.1. IPVLAN CNI plugin configuration example	511
16.3.1.2.5. Configuration for a MACVLAN secondary network	511
16.3.1.2.5.1. MACVLAN CNI plugin configuration example	512
16.3.1.2.6. Configuration for a TAP secondary network	512
16.3.1.2.6.1. Tap configuration example	513
16.3.1.2.6.2. Setting SELinux boolean for the TAP CNI plugin	513
16.3.1.2.7. Configuring routes using the route-override plugin on a secondary network	515
16.3.1.2.7.1. Route-override plugin configuration example	515
16.3.1.3. Attaching a pod to a secondary network	516
16.3.1.3.1. Adding a pod to a secondary network	516
16.3.1.3.1.1. Specifying pod-specific addressing and routing options	518
16.3.1.4. Configuring multi-network policy	522
16.3.1.4.1. Differences between multi-network policy and network policy	522
16.3.1.4.2. Enabling multi-network policy for the cluster	523
16.3.1.4.3. Supporting multi-network policies in IPv6 networks	523
16.3.1.4.4. Working with multi-network policy	524
16.3.1.4.4.1. Prerequisites	524
16.3.1.4.4.2. Creating a multi-network policy using the CLI	524
16.3.1.4.4.3. Editing a multi-network policy	527
16.3.1.4.4.4. Viewing multi-network policies using the CLI	529
16.3.1.4.4.5. Deleting a multi-network policy using the CLI	529
16.3.1.4.4.6. Creating a default deny all multi-network policy	530
16.3.1.4.4.7. Creating a multi-network policy to allow traffic from external clients	531
16.3.1.4.4.8. Creating a multi-network policy allowing traffic to an application from all namespaces	533
16.3.1.4.4.9. Creating a multi-network policy allowing traffic to an application from a namespace	535
16.3.1.4.5. Additional resources	537
16.3.1.5. Removing a pod from a secondary network	537
16.3.1.5.1. Removing a pod from a secondary network	538
16.3.1.6. Editing a secondary network	538
16.3.1.6.1. Modifying a secondary network attachment definition	538
16.3.1.7. Configuring IP address assignment on secondary networks	539
16.3.1.7.1. Configuration of IP address assignment for a network attachment	539
16.3.1.7.1.1. Static IP address assignment configuration	540
16.3.1.7.1.2. Dynamic IP address (DHCP) assignment configuration	541

16.3.7.1.3. Dynamic IP address assignment configuration with Whereabouts	542
16.3.7.1.3.1. Dynamic IP address configuration objects	542
16.3.7.1.3.2. Dynamic IP address assignment configuration that uses Whereabouts	543
16.3.7.1.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges	543
16.3.7.1.4. Creating a whereabouts-reconciler daemon set	544
16.3.7.1.5. Configuring the Whereabouts IP reconciler schedule	545
16.3.7.1.6. Creating a configuration for assignment of dual-stack IP addresses dynamically	546
16.3.8. Configuring the master interface in the container network namespace	547
16.3.8.1. About configuring the master interface in the container network namespace	547
16.3.8.1.1. Creating multiple VLANs on SR-IOV VFs	547
16.3.8.1.2. Creating a subinterface based on a bridge master interface in a container namespace	552
16.3.9. Removing an additional network	556
16.3.9.1. Removing a secondary network attachment definition	556
16.4. VIRTUAL ROUTING AND FORWARDING	556
16.4.1. About virtual routing and forwarding	557
16.4.1.1. Benefits of secondary networks for pods for telecommunications operators	557
16.5. ASSIGNING A SECONDARY NETWORK TO A VRF	557
16.5.1. Creating a secondary network attachment with the CNI VRF plugin	558
<b>CHAPTER 17. HARDWARE NETWORKS .....</b>	<b>561</b>
17.1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS	561
Additional resources	561
17.1.1. Components that manage SR-IOV network devices	562
17.1.1.1. Supported platforms	562
17.1.1.2. Supported devices	563
17.1.2. Additional resources	564
17.1.3. Next steps	564
17.2. CONFIGURING AN SR-IOV NETWORK DEVICE	565
17.2.1. SR-IOV network node configuration object	565
17.2.1.1. SR-IOV network node configuration examples	568
17.2.1.2. Automated discovery of SR-IOV network devices	569
17.2.1.2.1. Example SriovNetworkNodeState object	569
17.2.1.3. Configuring the SR-IOV Network Operator on Mellanox cards when Secure Boot is enabled	570
17.2.1.4. Virtual function (VF) partitioning for SR-IOV devices	572
17.2.1.5. A test pod template for clusters that use SR-IOV on OpenStack	573
17.2.1.6. A test pod template for clusters that use OVS hardware offloading on OpenStack	574
17.2.1.7. Huge pages resource injection for Downward API	575
17.2.2. Configuring SR-IOV network devices	576
17.2.3. Creating a non-uniform memory access (NUMA) aligned SR-IOV pod	577
17.2.4. Exclude the SR-IOV network topology for NUMA-aware scheduling	578
17.2.5. Troubleshooting SR-IOV configuration	579
17.2.6. Next steps	579
17.3. CONFIGURING AN SR-IOV ETHERNET NETWORK ATTACHMENT	579
17.3.1. Ethernet device configuration object	580
17.3.1.1. Creating a configuration for assignment of dual-stack IP addresses dynamically	581
17.3.1.2. Configuration of IP address assignment for a network attachment	582
17.3.1.2.1. Static IP address assignment configuration	582
17.3.1.2.2. Dynamic IP address (DHCP) assignment configuration	584
17.3.1.2.3. Dynamic IP address assignment configuration with Whereabouts	585
17.3.1.2.3.1. Dynamic IP address configuration objects	585
17.3.1.2.3.2. Dynamic IP address assignment configuration that uses Whereabouts	585
17.3.1.2.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges	585

	586
17.3.2. Configuring SR-IOV additional network	586
17.3.3. Assigning an SR-IOV network to a VRF	587
17.3.3.1. Creating an additional SR-IOV network attachment with the CNI VRF plugin	588
17.3.4. Runtime configuration for an Ethernet-based SR-IOV attachment	590
17.3.5. Adding a pod to a secondary network	591
17.3.5.1. Exposing MTU for vfio-pci SR-IOV devices to pod	593
17.3.6. Configuring parallel node draining during SR-IOV network policy updates	594
17.3.7. Excluding the SR-IOV network topology for NUMA-aware scheduling	597
17.3.8. Additional resources	601
17.4. CONFIGURING AN SR-IOV INFINIBAND NETWORK ATTACHMENT	601
17.4.1. InfiniBand device configuration object	601
17.4.1.1. Creating a configuration for assignment of dual-stack IP addresses dynamically	602
17.4.1.2. Configuration of IP address assignment for a network attachment	602
17.4.1.2.1. Static IP address assignment configuration	603
17.4.1.2.2. Dynamic IP address (DHCP) assignment configuration	604
17.4.1.2.3. Dynamic IP address assignment configuration with Whereabouts	605
17.4.1.2.3.1. Dynamic IP address configuration objects	606
17.4.1.2.3.2. Dynamic IP address assignment configuration that uses Whereabouts	606
17.4.1.2.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges	606
17.4.1.2.4. Configuring SR-IOV additional network	607
17.4.1.2.5. Runtime configuration for an InfiniBand-based SR-IOV attachment	608
17.4.1.2.6. Adding a pod to a secondary network	609
17.4.1.2.6.1. Exposing MTU for vfio-pci SR-IOV devices to pod	611
17.4.1.2.7. Additional resources	612
17.5. CONFIGURING AN RDMA SUBSYSTEM FOR SR-IOV	612
17.5.1. Configuring SR-IOV RDMA CNI	612
17.6. CONFIGURING INTERFACE-LEVEL NETWORK SYSCTL SETTINGS AND ALL-MULTICAST MODE FOR SR-IOV NETWORKS	615
17.6.1. Labeling nodes with an SR-IOV enabled NIC	615
17.6.2. Setting one sysctl flag	615
17.6.2.1. Setting one sysctl flag on nodes with SR-IOV network devices	616
17.6.2.2. Configuring sysctl on a SR-IOV network	617
17.6.2.3. Configuring sysctl settings for pods associated with bonded SR-IOV interface flag	621
17.6.2.3.1. Setting all sysctl flag on nodes with bonded SR-IOV network devices	621
17.6.2.3.2. Configuring sysctl on a bonded SR-IOV network	623
17.6.2.4. About all-multicast mode	627
17.6.2.4.1. Enabling the all-multicast mode on an SR-IOV network	627
17.7. CONFIGURING QINQ SUPPORT FOR SR-IOV ENABLED WORKLOADS	632
17.7.1. About 802.1Q-in-802.1Q support	632
17.7.2. Configuring QinQ support for SR-IOV enabled workloads	633
17.8. USING HIGH PERFORMANCE MULTICAST	636
17.8.1. High performance multicast	636
17.8.2. Configuring an SR-IOV interface for multicast	637
17.9. USING DPDK AND RDMA	638
17.9.1. Example use of a virtual function in a pod	638
17.9.2. Using a virtual function in DPDK mode with an Intel NIC	639
17.9.3. Using a virtual function in DPDK mode with a Mellanox NIC	642
17.9.4. Using the TAP CNI to run a rootless DPDK workload with kernel access	645
17.9.5. Overview of achieving a specific DPDK line rate	650
17.9.6. Using SR-IOV and the Node Tuning Operator to achieve a DPDK line rate	651
17.9.6.1. DPDK library for use with container applications	653

17.9.6.2. Example SR-IOV Network Operator for virtual functions	653
17.9.6.3. Example SR-IOV network operator	655
17.9.6.4. Example DPDK base workload	656
17.9.6.5. Example testpmd script	657
17.9.7. Using a virtual function in RDMA mode with a Mellanox NIC	657
17.9.8. A test pod template for clusters that use OVS-DPDK on OpenStack	661
17.9.9. Additional resources	661
<b>17.10. USING POD-LEVEL BONDING</b>	<b>662</b>
17.10.1. Configuring a bond interface from two SR-IOV interfaces	662
17.10.1.1. Creating a bond network attachment definition	663
17.10.1.2. Creating a pod using a bond interface	664
<b>17.11. CONFIGURING HARDWARE OFFLOADING</b>	<b>665</b>
17.11.1. About hardware offloading	665
17.11.2. Supported devices	666
17.11.3. Prerequisites	666
17.11.4. Setting the SR-IOV Network Operator into systemd mode	666
17.11.5. Configuring a machine config pool for hardware offloading	667
17.11.6. Configuring the SR-IOV network node policy	669
17.11.6.1. An example SR-IOV network node policy for OpenStack	670
17.11.7. Improving network traffic performance using a virtual function	670
17.11.8. Creating a network attachment definition	672
17.11.9. Adding the network attachment definition to your pods	673
<b>17.12. SWITCHING BLUEFIELD-2 FROM DPU TO NIC</b>	<b>674</b>
17.12.1. Switching Bluefield-2 from DPU mode to NIC mode	674
<b>CHAPTER 18. OVN-KUBERNETES NETWORK PLUGIN .....</b>	<b>677</b>
<b>18.1. ABOUT THE OVN-KUBERNETES NETWORK PLUGIN</b>	<b>677</b>
18.1.1. OVN-Kubernetes purpose	677
18.1.2. OVN-Kubernetes IPv6 and dual-stack limitations	678
18.1.3. Session affinity	679
Stickiness timeout for session affinity	679
<b>18.2. OVN-KUBERNETES ARCHITECTURE</b>	<b>679</b>
18.2.1. Introduction to OVN-Kubernetes architecture	679
18.2.2. Listing all resources in the OVN-Kubernetes project	681
18.2.3. Listing the OVN-Kubernetes northbound database contents	683
18.2.4. Command-line arguments for ovn-nbctl to examine northbound database contents	687
18.2.5. Listing the OVN-Kubernetes southbound database contents	688
18.2.6. Command-line arguments for ovn-sbctl to examine southbound database contents	690
18.2.7. OVN-Kubernetes logical architecture	691
18.2.7.1. Installing network-tools on local host	692
18.2.7.2. Running network-tools	692
18.2.8. Additional resources	696
<b>18.3. TROUBLESHOOTING OVN-KUBERNETES</b>	<b>696</b>
18.3.1. Monitoring OVN-Kubernetes health by using readiness probes	696
18.3.2. Viewing OVN-Kubernetes alerts in the console	697
18.3.3. Viewing OVN-Kubernetes alerts in the CLI	697
18.3.4. Viewing the OVN-Kubernetes logs using the CLI	698
18.3.5. Viewing the OVN-Kubernetes logs using the web console	699
18.3.5.1. Changing the OVN-Kubernetes log levels	699
18.3.6. Checking the OVN-Kubernetes pod network connectivity	703
18.3.7. Checking OVN-Kubernetes network traffic with OVS sampling using the CLI	703
18.3.7.1. OVN-Kubernetes network traffic with OVS sampling flags	706
18.3.8. Additional resources	707

18.4. TRACING OPENFLOW WITH OVNKUBE-TRACE	707
18.4.1. Installing the ovnkube-trace on local host	707
18.4.2. Running ovnkube-trace	709
18.4.3. Additional resources	714
18.5. CONVERTING TO IPV4/IPV6 DUAL-STACK NETWORKING	714
18.5.1. Converting to a dual-stack cluster network	715
18.5.2. Converting to a single-stack cluster network	719
18.6. CONFIGURING OVN-KUBERNETES INTERNAL IP ADDRESS SUBNETS	719
18.6.1. Configuring the OVN-Kubernetes join subnet	719
18.6.2. Configuring the OVN-Kubernetes masquerade subnet as a post-installation operation	720
18.6.3. Configuring the OVN-Kubernetes transit subnet	721
18.7. CONFIGURING GATEWAY MODE	722
18.7.1. Setting local and shared gateway modes	723
18.8. CONFIGURE AN EXTERNAL GATEWAY ON THE DEFAULT NETWORK	724
18.8.1. Prerequisites	724
18.8.2. How OpenShift Container Platform determines the external gateway IP address	724
18.8.3. AdminPolicyBasedExternalRoute object configuration	725
18.8.3.1. Example secondary external gateway configurations	727
18.8.4. Configure a secondary external gateway	728
18.8.5. Additional resources	729
18.9. CONFIGURING AN EGRESS IP ADDRESS	729
18.9.1. Egress IP address architectural design and implementation	729
18.9.1.1. Platform support	730
18.9.1.2. Public cloud platform considerations	731
18.9.1.2.1. Amazon Web Services (AWS) IP address capacity limits	732
18.9.1.2.2. Google Cloud Platform (GCP) IP address capacity limits	732
18.9.1.2.3. Microsoft Azure IP address capacity limits	732
18.9.1.3. Considerations for using an egress IP on additional network interfaces	733
Requirements for assigning an egress IP to a network interface that is not the primary network interface	733
18.9.1.4. Assignment of egress IPs to pods	734
18.9.1.5. Assignment of egress IPs to nodes	734
18.9.1.6. Architectural diagram of an egress IP address configuration	735
18.9.2. EgressIP object	736
18.9.3. The egressIPConfig object	738
18.9.4. Labeling a node to host egress IP addresses	739
18.9.5. Next steps	739
18.9.6. Additional resources	739
18.10. ASSIGNING AN EGRESS IP ADDRESS	740
18.10.1. Assigning an egress IP address to a namespace	740
18.10.2. Additional resources	741
18.11. CONFIGURING AN EGRESS SERVICE	741
18.11.1. Egress service custom resource	742
18.11.2. Deploying an egress service	743
18.12. CONSIDERATIONS FOR THE USE OF AN EGRESS ROUTER POD	745
18.12.1. About an egress router pod	745
18.12.1.1. Egress router modes	746
18.12.1.2. Egress router pod implementation	746
18.12.1.3. Deployment considerations	747
18.12.1.4. Failover configuration	747
18.12.2. Additional resources	748
18.13. DEPLOYING AN EGRESS ROUTER POD IN REDIRECT MODE	748
18.13.1. Egress router custom resource	748

18.13.2. Deploying an egress router in redirect mode	750
<b>18.14. ENABLING MULTICAST FOR A PROJECT</b>	753
18.14.1. About multicast	753
18.14.2. Enabling multicast between pods	753
<b>18.15. DISABLING MULTICAST FOR A PROJECT</b>	755
18.15.1. Disabling multicast between pods	755
<b>18.16. TRACKING NETWORK FLOWS</b>	756
18.16.1. Network object configuration for tracking network flows	757
18.16.2. Adding destinations for network flows collectors	757
18.16.3. Deleting all destinations for network flows collectors	759
18.16.4. Additional resources	759
<b>18.17. CONFIGURING HYBRID NETWORKING</b>	759
18.17.1. Configuring hybrid networking with OVN-Kubernetes	759
18.17.2. Additional resources	761
<b>CHAPTER 19. CONFIGURING ROUTES .....</b>	<b>762</b>
<b>19.1. ROUTE CONFIGURATION</b>	762
19.1.1. Creating an HTTP-based route	762
19.1.2. Creating a route for Ingress Controller sharding	763
19.1.3. Configuring route timeouts	765
19.1.4. HTTP Strict Transport Security	766
19.1.4.1. Enabling HTTP Strict Transport Security per-route	766
19.1.4.2. Disabling HTTP Strict Transport Security per-route	767
19.1.4.3. Enforcing HTTP Strict Transport Security per-domain	768
19.1.5. Throughput issue troubleshooting methods	771
19.1.6. Using cookies to keep route statefulness	772
19.1.6.1. Annotating a route with a cookie	772
19.1.7. Path-based routes	773
19.1.8. HTTP header configuration	774
19.1.8.1. Order of precedence	774
19.1.8.2. Special case headers	775
19.1.9. Setting or deleting HTTP request and response headers in a route	777
19.1.10. Route-specific annotations	778
19.1.11. Configuring the route admission policy	786
19.1.12. Creating a route through an Ingress object	787
19.1.13. Creating a route using the default certificate through an Ingress object	789
19.1.14. Creating a route using the destination CA certificate in the Ingress annotation	790
19.1.15. Configuring the OpenShift Container Platform Ingress Controller for dual-stack networking	792
<b>19.2. SECURED ROUTES</b>	793
19.2.1. Creating a re-encrypt route with a custom certificate	793
19.2.2. Creating an edge route with a custom certificate	795
19.2.3. Creating a passthrough route	796
19.2.4. Creating a route with externally managed certificate	797
<b>CHAPTER 20. CONFIGURING INGRESS CLUSTER TRAFFIC .....</b>	<b>800</b>
<b>20.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW</b>	800
20.1.1. Comparison: Fault tolerant access to external IP addresses	800
<b>20.2. CONFIGURING EXTERNALIPS FOR SERVICES</b>	801
20.2.1. Prerequisites	801
20.2.2. About ExternalIP	801
20.2.3. Additional resources	802
20.2.4. Configuration for ExternalIP	802
20.2.5. Restrictions on the assignment of an external IP address	803

20.2.6. Example policy objects	804
20.2.7. ExternalIP address block configuration	805
Example external IP configurations	806
20.2.8. Configure external IP address blocks for your cluster	806
20.2.9. Next steps	807
<b>20.3. CONFIGURING INGRESS CLUSTER TRAFFIC USING AN INGRESS CONTROLLER</b>	<b>807</b>
20.3.1. Using Ingress Controllers and routes	807
20.3.2. Prerequisites	808
20.3.3. Creating a project and service	808
20.3.4. Exposing the service by creating a route	809
20.3.5. Ingress sharding in OpenShift Container Platform	810
20.3.6. Ingress Controller sharding	810
20.3.6.1. Traditional sharding example	811
20.3.6.2. Overlapped sharding example	812
20.3.6.3. Sharding the default Ingress Controller	813
20.3.6.4. Ingress sharding and DNS	814
20.3.6.5. Configuring Ingress Controller sharding by using route labels	814
20.3.6.6. Configuring Ingress Controller sharding by using namespace labels	815
20.3.6.7. Creating a route for Ingress Controller sharding	817
Additional resources	819
<b>20.4. CONFIGURING THE INGRESS CONTROLLER ENDPOINT PUBLISHING STRATEGY</b>	<b>819</b>
20.4.1. Ingress Controller endpoint publishing strategy	819
20.4.1.1. Configuring the Ingress Controller endpoint publishing scope to Internal	821
20.4.1.2. Configuring the Ingress Controller endpoint publishing scope to External	822
20.4.1.3. Adding a single NodePort service to an Ingress Controller	823
20.4.2. Additional resources	826
<b>20.5. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER</b>	<b>826</b>
20.5.1. Using a load balancer to get traffic into the cluster	826
20.5.2. Prerequisites	827
20.5.3. Creating a project and service	827
20.5.4. Exposing the service by creating a route	828
20.5.5. Creating a load balancer service	828
<b>20.6. CONFIGURING INGRESS CLUSTER TRAFFIC ON AWS</b>	<b>831</b>
20.6.1. Configuring Classic Load Balancer timeouts on AWS	831
20.6.1.1. Configuring route timeouts	831
20.6.1.2. Configuring Classic Load Balancer timeouts	832
20.6.2. Configuring ingress cluster traffic on AWS using a Network Load Balancer	832
20.6.2.1. Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer	832
20.6.2.2. Switching the Ingress Controller from using a Network Load Balancer to a Classic Load Balancer	834
20.6.2.3. Replacing Ingress Controller Classic Load Balancer with Network Load Balancer	835
20.6.2.4. Configuring an Ingress Controller Network Load Balancer on an existing AWS cluster	836
20.6.2.5. Configuring an Ingress Controller Network Load Balancer on a new AWS cluster	837
20.6.2.6. Choosing subnets while creating a LoadBalancerService Ingress Controller	838
20.6.2.7. Updating the subnets on an existing Ingress Controller	840
20.6.2.8. Configuring AWS Elastic IP (EIP) addresses for a Network Load Balancer (NLB)	842
20.6.3. Additional resources	843
<b>20.7. CONFIGURING INGRESS CLUSTER TRAFFIC FOR A SERVICE EXTERNAL IP</b>	<b>843</b>
20.7.1. Prerequisites	843
20.7.2. Attaching an ExternalIP to a service	844
20.7.3. Additional resources	845
<b>20.8. CONFIGURING INGRESS CLUSTER TRAFFIC BY USING A NODEPORT</b>	<b>845</b>

20.8.1. Using a NodePort to get traffic into the cluster	845
20.8.2. Prerequisites	846
20.8.3. Creating a project and service	846
20.8.4. Exposing the service by creating a route	847
20.8.5. Additional resources	848
<b>20.9. CONFIGURING INGRESS CLUSTER TRAFFIC USING LOAD BALANCER ALLOWED SOURCE RANGES</b>	<b>848</b>
20.9.1. Configuring load balancer allowed source ranges	848
20.9.2. Migrating to load balancer allowed source ranges	849
20.9.3. Additional resources	850
<b>20.10. PATCHING EXISTING INGRESS OBJECTS</b>	<b>850</b>
20.10.1. Patching Ingress objects to resolve an ingressWithoutClassName alert	850
<b>CHAPTER 21. KUBERNETES NMSTATE .....</b>	<b>852</b>
21.1. OBSERVING AND UPDATING THE NODE NETWORK STATE AND CONFIGURATION	852
21.1.1. Viewing the network state of a node by using the CLI	852
21.1.2. Viewing the network state of a node (NNS) from the web console	853
21.1.2.1. Viewing a graphical representation of the NNS topology	853
21.1.3. The NodeNetworkConfigurationPolicy manifest file	854
Additional resources	854
21.1.4. Managing policy from the web console	854
21.1.4.1. Monitoring the policy status	854
21.1.4.2. Creating a policy	855
21.1.5. Updating the policy	856
21.1.5.1. Updating the policy by using form	856
21.1.5.2. Updating the policy by using YAML	857
21.1.5.3. Deleting the policy	857
21.1.6. Managing policy by using the CLI	857
21.1.6.1. Creating an interface on nodes	857
Additional resources	859
21.1.6.2. Confirming node network policy updates on nodes	859
21.1.6.3. Removing an interface from nodes	860
21.1.7. Example policy configurations for different interfaces	861
21.1.7.1. Example: Ethernet interface node network configuration policy	862
21.1.7.2. Example: Linux bridge interface node network configuration policy	862
21.1.7.3. Example: VLAN interface node network configuration policy	863
21.1.7.4. Example: Bond interface node network configuration policy	865
21.1.7.5. Example: Multiple interfaces in the same node network configuration policy	866
21.1.7.6. Example: Node network configuration policy for virtual functions	867
21.1.7.7. Example: Network interface with a VRF instance node network configuration policy	870
21.1.8. Creating an IP over InfiniBand interface on nodes	871
21.1.9. Capturing the static IP of a NIC attached to a bridge	873
21.1.9.1. Example: Linux bridge interface node network configuration policy to inherit static IP address from the NIC attached to the bridge	873
21.1.10. Examples: IP management	874
21.1.10.1. Static	874
21.1.10.2. No IP address	874
21.1.10.3. Dynamic host configuration	875
21.1.10.4. Media Access Control (MAC) address	875
21.1.10.5. DNS	876
21.1.10.6. Static routing	879
21.2. TROUBLESHOOTING NODE NETWORK CONFIGURATION	880
21.2.1. Troubleshooting an incorrect node network configuration policy configuration	880

21.2.2. Troubleshooting DNS connectivity issues in a disconnected environment	883
21.2.2.1. Configuring the bind9 DNS named server	883
21.2.2.2. Configuring the dnsmasq DNS server	884
<b>CHAPTER 22. CONFIGURING THE CLUSTER-WIDE PROXY .....</b>	<b>885</b>
22.1. PREREQUISITES	886
22.2. ENABLING THE CLUSTER-WIDE PROXY	886
22.3. REMOVING THE CLUSTER-WIDE PROXY	889
22.4. VERIFYING THE CLUSTER-WIDE PROXY CONFIGURATION	889
Additional resources	890
<b>CHAPTER 23. CONFIGURING A CUSTOM PKI .....</b>	<b>891</b>
23.1. CONFIGURING THE CLUSTER-WIDE PROXY DURING INSTALLATION	891
23.2. ENABLING THE CLUSTER-WIDE PROXY	893
23.3. CERTIFICATE INJECTION USING OPERATORS	896
<b>CHAPTER 24. LOAD BALANCING ON RHOSP .....</b>	<b>898</b>
24.1. LIMITATIONS OF LOAD BALANCER SERVICES	898
24.1.1. Local external traffic policies	898
24.2. SCALING CLUSTERS FOR APPLICATION TRAFFIC BY USING OCTAVIA	898
24.2.1. Scaling clusters by using Octavia	898
24.3. SERVICES FOR A USER-MANAGED LOAD BALANCER	899
24.3.1. Configuring a user-managed load balancer	902
24.4. SPECIFYING A FLOATING IP ADDRESS IN THE INGRESS CONTROLLER	909
<b>CHAPTER 25. LOAD BALANCING WITH METALLB .....</b>	<b>912</b>
25.1. CONFIGURING METALLB ADDRESS POOLS	912
25.1.1. About the IPAddressPool custom resource	912
25.1.2. Configuring an address pool	913
25.1.3. Configure MetalLB address pool for VLAN	914
25.1.4. Example address pool configurations	915
25.1.4.1. Example: IPv4 and CIDR ranges	915
25.1.4.2. Example: Assign IP addresses	916
25.1.4.3. Example: IPv4 and IPv6 addresses	916
25.1.4.4. Example: Assign IP address pools to services or namespaces	917
25.1.5. Next steps	918
25.2. ABOUT ADVERTISING FOR THE IP ADDRESS POOLS	918
25.2.1. About the BGPAutomation custom resource	918
25.2.2. Configuring MetalLB with a BGP advertisement and a basic use case	919
25.2.2.1. Example: Advertise a basic address pool configuration with BGP	920
25.2.3. Configuring MetalLB with a BGP advertisement and an advanced use case	920
25.2.3.1. Example: Advertise an advanced address pool configuration with BGP	921
25.2.4. Advertising an IP address pool from a subset of nodes	922
25.2.5. About the L2Automation custom resource	923
25.2.6. Configuring MetalLB with an L2 advertisement	924
25.2.7. Configuring MetalLB with a L2 advertisement and label	925
25.2.8. Configuring MetalLB with an L2 advertisement for selected interfaces	926
25.2.9. Configuring MetalLB with secondary networks	927
25.2.9.1. Enabling IP forwarding for a specific interface	927
25.2.9.2. Enabling IP forwarding globally	929
25.2.10. Additional resources	929
25.3. CONFIGURING METALLB BGP PEERS	929
25.3.1. About the BGP peer custom resource	930
25.3.2. Configuring a BGP peer	931

25.3.3. Configure a specific set of BGP peers for a given address pool	932
25.3.4. Exposing a service through a network VRF	934
25.3.5. Example BGP peer configurations	938
25.3.5.1. Example: Limit which nodes connect to a BGP peer	939
25.3.5.2. Example: Specify a BFD profile for a BGP peer	939
25.3.5.3. Example: Specify BGP peers for dual-stack networking	939
25.3.6. Next steps	940
25.4. CONFIGURING COMMUNITY ALIAS	940
25.4.1. About the community custom resource	940
25.4.2. Configuring MetalLB with a BGP advertisement and community alias	941
25.5. CONFIGURING METALLB BFD PROFILES	942
25.5.1. About the BFD profile custom resource	942
25.5.2. Configuring a BFD profile	944
25.5.3. Next steps	945
25.6. CONFIGURING SERVICES TO USE METALLB	945
25.6.1. Request a specific IP address	945
25.6.2. Request an IP address from a specific pool	945
25.6.3. Accept any IP address	946
25.6.4. Share a specific IP address	946
25.6.5. Configuring a service with MetalLB	948
25.7. MANAGING SYMMETRIC ROUTING WITH METALLB	949
25.7.1. Challenges of managing symmetric routing with MetalLB	949
25.7.2. Overview of managing symmetric routing by using VRFs with MetalLB	950
25.7.3. Configuring symmetric routing by using VRFs with MetalLB	951
25.8. CONFIGURING THE INTEGRATION OF METALLB AND FRR-K8S	955
25.8.1. FRR configurations	956
25.8.2. Configuring the FRRConfiguration CRD	957
25.8.2.1. The routers field	957
25.8.2.2. The toAdvertise field	957
25.8.2.3. The toReceive field	958
25.8.2.4. The bgp field	959
25.8.2.5. The nodeSelector field	960
25.8.3. How FRR-K8s merges multiple configurations	964
25.8.3.1. Configuration conflicts	964
25.8.3.2. Merging	964
25.9. METALLB LOGGING, TROUBLESHOOTING, AND SUPPORT	965
25.9.1. Setting the MetalLB logging levels	965
25.9.1.1. FRRouting (FRR) log levels	969
25.9.2. Troubleshooting BGP issues	969
25.9.3. Troubleshooting BFD issues	972
25.9.4. MetalLB metrics for BGP and BFD	973
25.9.5. About collecting MetalLB data	975
<b>CHAPTER 26. ASSOCIATING SECONDARY INTERFACES METRICS TO NETWORK ATTACHMENTS ...</b>	<b>977</b>
26.1. EXTENDING SECONDARY NETWORK METRICS FOR MONITORING	977
26.1.1. Network Metrics Daemon	977
26.1.2. Metrics with network name	978



# CHAPTER 1. UNDERSTANDING NETWORKING

Understanding the fundamentals of networking in OpenShift Container Platform ensures efficient and secure communication within your clusters and is essential for effective network administration. Key elements of networking in your environment include understanding how pods and services communicate, the role of IP addresses, and the use of DNS for service discovery.

## 1.1. NETWORKING IN OPENSHIFT CONTAINER PLATFORM

OpenShift Container Platform ensures seamless communication between various components within the cluster and between external clients and the cluster. Networking relies on the following core concepts and components:

- Pod-to-pod communication
- Services
- DNS
- Ingress
- Network controls
- Load balancing

### 1.1.1. Common practices for networking services

In OpenShift Container Platform, services create a single IP address for clients to use, even if multiple pods are providing that service. This abstraction enables seamless scaling, fault tolerance, and rolling upgrades without affecting clients.

Network security policies manage traffic within the cluster. Network controls empower namespace administrators to define ingress and egress rules for their pods. By using network administration policies, cluster administrators can establish namespace policies, override namespace policies, or set default policies when none are defined.

Egress firewall configurations control outbound traffic from pods. These configuration settings ensure that only authorized communication occurs. The ingress node firewall protects nodes by controlling incoming traffic. Additionally, the Universal Data Network manages data traffic across the cluster.

### 1.1.2. Networking features

OpenShift Container Platform offers several networking features and enhancements. These features and enhancements are listed as follows:

- **Ingress Operator and Route API:** OpenShift Container Platform includes an Ingress Operator that implements the Ingress Controller API. This component enables external access to cluster services by deploying and managing HAProxy-based Ingress Controllers that support advanced routing configurations and load balancing. OpenShift Container Platform uses the Route API to translate upstream Ingress objects to route objects. Routes are specific to networking in OpenShift Container Platform, but you can also use third-party Ingress Controllers.
- **Enhanced security:** OpenShift Container Platform provides advanced network security features, such as the egress firewall and the ingress node firewall.

- Egress firewall: The egress firewall controls and restricts outbound traffic from pods within the cluster. You can set rules to limit which external hosts or IP ranges with which pods can communicate.
- Ingress node firewall: The ingress node firewall is managed by the Ingress Firewall Operator and provides firewall rules at the node level. You can protect your nodes from threats by configuring this firewall on specific nodes within the cluster to filter incoming traffic before it reaches these nodes.



### NOTE

OpenShift Container Platform also implements services, such as Network Policy, Admin Network Policy, and Security Context Constraints (SCC) to secure communication between pods and enforce access controls.

- Role-based access control (RBAC): OpenShift Container Platform extends Kubernetes RBAC to provide more granular control over who can access and manage network resources. RBAC helps maintain security and compliance within the cluster.
- Multi-tenancy support: OpenShift Container Platform offers multi-tenancy support to enable multiple users and teams to share the same cluster while keeping their resources isolated and secure.
- Hybrid and multi-cloud capabilities: OpenShift Container Platform is designed to work seamlessly across on-premise, cloud, and multi-cloud environments. This flexibility allows organizations to deploy and manage containerized applications across different infrastructures.
- Observability and monitoring: OpenShift Container Platform provides integrated observability and monitoring tools that help manage and troubleshoot network issues. These tools include role-based access to network metrics and logs.
- User-defined networks (UDN): UDNs allow administrators to customize network configurations. UDNs provide enhanced network isolation and IP address management.
- Egress IP: Egress IP allows you to assign a fixed source IP address for all egress traffic originating from pods within a namespace. Egress IP can improve security and access control by ensuring consistent source IP addresses for external services. For example, if a pod needs to access an external database that only allows traffic from specific IP addresses, you can configure an egress IP for that pod to meet the access requirements.
- Egress router: An egress router is a pod that acts as a bridge between the cluster and external systems. Egress routers allow traffic from pods to be routed through a specific IP address that is not used for any other purpose. With egress routers, you can enforce access controls or route traffic through a specific gateway.

## 1.2. NETWORKING WITH NODES, CLIENTS, AND CLUSTERS

A node is a machine in the cluster that can run either control-plane components, workload components, or both. A node is either a physical server or a virtual machine. A cluster is a collection of nodes that run containerized applications. Clients are the tools and users that interact with the cluster.

### 1.2.1. What is a node?

Nodes are the physical or virtual machines that run containerized applications. Nodes host the pods and provide resources, such as memory and storage for running the applications. Nodes enable

communication between pods. Each pod is assigned an IP address. Pods within the same node can communicate with each other using these IP addresses. Nodes facilitate service discovery by allowing pods to discover and communicate with services within the cluster. Nodes help distribute network traffic among pods to ensure efficient load balancing and high availability of applications. Nodes provide a bridge between the internal cluster network and external networks to allow external clients to access services running on the cluster.

### 1.2.2. Understanding clusters

A cluster is a collection of nodes that work together to run containerized applications. These nodes include control plane nodes and compute nodes.

### 1.2.3. Understanding external clients

An external client is any entity outside the cluster that interacts with the services and applications running within the cluster. External can include end users, external services, and external devices. End users are people who access a web application hosted in the cluster through their browsers or mobile devices. External services are other software systems or applications that interact with the services in the cluster, often through APIs. External devices are any hardware outside the cluster network that needs to communicate with the cluster services, such as the Internet of Things (IoT) devices.

## 1.3. NETWORKING CONCEPTS AND COMPONENTS

Networking in OpenShift Container Platform uses several key components and concepts.

- Pods and services are the smallest deployable units in Kubernetes, and services provide stable IP addresses and DNS names for sets of pods. Each pod in a cluster is assigned a unique IP address. Pods use IP addresses to communicate directly with other pods, regardless of which node they are on. The pod IP addresses will change when pods are destroyed and created. Services are also assigned unique IP addresses. A service is associated with the pods that can provide the service. When accessed, the service IP address provides a stable way to access pods by sending traffic to one of the pods that backs the service.
- Route and Ingress APIs define rules that route HTTP, HTTPS, and TLS traffic to services within the cluster. OpenShift Container Platform provides both Route and Ingress APIs as part of the default installation, but you can add third-party Ingress Controllers to the cluster.
- The Container Network Interface (CNI) plugin manages the pod network to enable pod-to-pod communication.
- The Cluster Network Operator (CNO) CNO manages the networking plugin components of a cluster. Using the CNO, you can set the network configuration, such as the pod network CIDR and service network CIDR.
- DNS operators manage DNS services within the cluster to ensure that services are reachable by their DNS names.
- Network controls define how pods are allowed to communicate with each other and with other network endpoints. These policies help secure the cluster by controlling traffic flow and enforcing rules for pod communication.
- Load balancing distributes network traffic across multiple servers to ensure reliability and performance.

- Service discovery is a mechanism for services to find and communicate with each other within the cluster.
- The Ingress Operator uses OpenShift Container Platform Route to manage the router and enable external access to cluster services.

## Additional resources

- [About network policy](#)

# 1.4. HOW PODS COMMUNICATE

Pods use IP addresses to communicate and a Dynamic Name System (DNS) to discover IP addresses for pods or services. Clusters use various policy types that control what communication is allowed. Pods communicate in two ways: pod-to-pod and service-to-pod.

## 1.4.1. Pod-to-pod communication

Pod-to-pod communication is the ability of pods to communicate with each other within the cluster. This is crucial for the functioning of microservices and distributed applications.

Each pod in a cluster is assigned a unique IP address that they use to communicate directly with other pods. Pod-to-pod communication is useful for intra-cluster communication where pods need to exchange data or perform tasks collaboratively. For example, Pod A can send requests directly to Pod B using Pod B's IP address. Pods can communicate over a flat network without Network Address Translation (NAT). This allows for seamless communication between pods across different nodes.

### 1.4.1.1. Example: Controlling pod-to-pod communication

In a microservices-based application with multiple pods, a frontend pod needs to communicate with the a backend pod to retrieve data. By using pod-to-pod communication, either directly or through services, these pods can efficiently exchange information.

To control and secure pod-to-pod communication, you can define network controls. These controls enforce security and compliance requirements by specifying how pods interact with each other based on labels and selectors.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-some-pods
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: app
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: backend
  ports:
    - protocol: TCP
      port: 80
```

### 1.4.2. Service-to-pod communication

Service-to-pod communication ensures that services can reliably route traffic to the appropriate pods. Services are objects that define a logical set of pods and provide a stable endpoint, such as IP addresses and DNS names. Pod IP addresses can change. Services abstract pod IP addresses to provide a consistent way to access the application components even as IP addresses change.

Key concepts of service-to-pod communication include:

- Endpoints: Endpoints define the IP addresses and ports of the pods that are associated with a service.
- Selectors: Selectors use labels, such as key-value pairs, to define the criteria for selecting a set of objects that a service should target.
- Services: Services provide a stable IP address and DNS name for a set of pods. This abstraction allows other components to communicate with the service rather than individual pods.
- Service discovery: DNS makes services discoverable. When a service is created, it is assigned a DNS name. Other pods discover this DNS name and use it to communicate with the service.
- Service Types: Service types control how services are exposed within or outside the cluster.
  - ClusterIP exposes the service on an internal cluster IP. It is the default service type and makes the service only reachable from within the cluster.
  - NodePort allows external traffic to access the service by exposing the service on each node's IP at a static port.
  - LoadBalancer uses a cloud provider's load balancer to expose the service externally.

Services use selectors to identify the pods that should receive the traffic. The selectors match labels on the pods to determine which pods are part of the service. Example: A service with the selector **app: myapp** will route traffic to all pods with the label **app: myapp**.

Endpoints are dynamically updated to reflect the current IP addresses of the pods that match the service selector. {product-name} maintains these endpoints and ensures that the service routes traffic to the correct pods.

The communication flow refers to the sequence of steps and interactions that occur when a service in Kubernetes routes traffic to the appropriate pods. The typical communication flow for service-to-pod communication is as follows:

- Service creation: When you create a service, you define the service type, the port on which the service listens, and the selector labels.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

- DNS resolution: Each pod has a DNS name that other pods can use to communicate with the service. For example, if the service is named **my-service** in the **my-app** namespace, its DNS name is **my-service.my-app.svc.cluster.local**.
- Traffic routing: When a pod sends a request to the service's DNS name, OpenShift Container Platform resolves the name to the service's ClusterIP. The service then uses the endpoints to route the traffic to one of the pods that match its selector.
- Load balancing: Services also provide basic load balancing. They distribute incoming traffic across all the pods that match the selector. This ensures that no single pod is overwhelmed with too much traffic.

#### 1.4.2.1. Example: Controlling service-to-pod communication

A cluster is running a microservices-based application with two components: a front-end and a backend. The front-end needs to communicate with the backend to fetch data.

##### Procedure

1. Create a backend service.

```
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

2. Configure backend pods.

```
apiVersion: v1
kind: Pod
metadata:
  name: backend-pod
  labels:
    app: backend
spec:
  containers:
    - name: backend-container
      image: my-backend-image
      ports:
        - containerPort: 8080
```

3. Establish front-end communication.

The front-end pods can now use the DNS name **backend.default.svc.cluster.local** to communicate with the backend service. The service ensures that the traffic is routed to one of the backend pods.

Service-to-pod communication abstracts the complexity of managing pod IPs and ensures reliable and efficient communication within the cluster.

## 1.5. SUPPORTED LOAD BALANCERS

Load balancing distributes incoming network traffic across multiple servers to maintain the health and efficiency of your clusters by ensuring that no single server bears too much load. Load balancers are devices that perform load balancing. They act as intermediaries between clients and servers to manage and direct traffic based on predefined rules.

OpenShift Container Platform supports the following types of load balancers:

- Classic Load Balancer (CLB)
- Elastic Load Balancing (ELB)
- Network Load Balancer (NLB)
- Application Load Balancer (ALB)

ELB is the default load-balancer type for AWS routers. CLB is the default for self-managed environments. NLB is the default for Red Hat OpenShift Service on AWS (ROSA).



### IMPORTANT

Use ALB in front of an application but not in front of a router. Using an ALB requires the AWS Load Balancer Operator add-on. This operator is not supported for all Amazon Web Services (AWS) regions or for all OpenShift Container Platform profiles.

### 1.5.1. Configuring Load balancers

You can define your default load-balancer type during cluster installation. After installation, you can configure your ingress controller to behave in a specific way that is not covered by the global platform configuration that you defined at cluster installation.

#### 1.5.1.1. Define the default load balancer type

When installing the cluster, you can specify the type of load balancer that you want to use. The type of load balancer you choose at cluster installation gets applied to the entire cluster.

This example shows how to define the default load-balancer type for a cluster deployed on AWS. You can apply the procedure on other supported platforms.

```
apiVersion: v1
kind: Network
metadata:
  name: cluster
platform:
  aws: ①
    lbType: classic ②
```

① The **platform** key represents the platform on which you have deployed your cluster. This example uses **aws**.

② The **lbType** key represents the load balancer type. This example uses the Classic Load Balancer, **classic**.

### 1.5.1.2. Specify load balancer behavior for an Ingress Controller

After you install a cluster, you can configure your Ingress Controller to specify how services are exposed to external networks, so that you can better control the settings and behavior of a load balancer.



#### NOTE

Changing the load balancer settings on an Ingress Controller might override the load balancer settings you specified at installation.

```
apiVersion: v1
kind: Network
metadata:
  name: cluster
endpointPublishingStrategy:
  loadBalancer: ①
    dnsManagementPolicy: Managed
    providerParameters:
      aws:
        classicLoadBalancer: ②
        connectionIdleTimeout: 0s
        type: Classic
        type: AWS
        scope: External
    type: LoadBalancerService
```

① The `loadBalancer` field specifies the load balancer configuration settings.

② The **classicLoadBalancer** field sets the load balancer to **classic** and includes settings specific to the CLB on AWS.

## 1.6. THE DOMAIN NAME SYSTEM (DNS)

The Domain Name System (DNS) is a hierarchical and decentralized naming system used to translate human-friendly domain names, such as `www.example.com`, into IP addresses that identify computers on a network. DNS plays a crucial role in service discovery and name resolution.

OpenShift Container Platform provides a built-in DNS to ensure that services can be reached by their DNS names. This helps maintain stable communication even if the underlying IP addresses change. When you start a pod, environment variables for service names, IP addresses, and ports are created automatically to enable the pod to communicate with other services.

### 1.6.1. Key DNS terms

- CoreDNS: CoreDNS is the DNS server and provides name resolution for services and pods.
- DNS names: Services are assigned DNS names based on their namespace and name. For example, a service named **my-service** in the **default** namespace would have the DNS name **my-service.default.svc.cluster.local**.
- Domain names: Domain names are the human-friendly names used to access websites and services, such as **example.com**.

- IP addresses: IP addresses are numerical labels assigned to each device connected to a computer network that uses IP for communication. An example of an IPv4 address is **192.0.2.1**. An example of an IPv6 address is **2001:0db8:85a3:0000:0000:8a2e:0370:7334**.
- DNS servers: DNS servers are specialized servers that store DNS records. These records map domain names to IP addresses. When you type a domain name into your browser, your computer contacts a DNS server to find the corresponding IP address.
- Resolution process: A DNS query is sent to a DNS resolver. The DNS resolver then contacts a series of DNS servers to find the IP address associated with the domain name. The resolver will try using the name with a series of domains, such as **<namespace>.svc.cluster.local**, **svc.cluster.local**, and **cluster.local**. This process stops at the first match. The IP address is returned to your browser and then connects to the web server using the IP address.

### 1.6.2. Example: DNS use case

For this example, a front-end application is running in one set of pods and a back-end service is running in another set of pods. The front-end application needs to communicate with the back-end service. You create a service for the back-end pods that gives it a stable IP address and DNS name. The front-end pods use this DNS name to access the back-end service regardless of changes to individual pod IP addresses.

By creating a service for the back-end pods, you provide a stable IP and DNS name, **backend-service.default.svc.cluster.local**, that the front-end pods can use to communicate with the back-end service. This setup would ensure that even if individual pod IP addresses change, the communication remains consistent and reliable.

The following steps demonstrate an example of how to configure front-end pods to communicate with a back-end service using DNS.

- Create the back-end service.

- Deploy the back-end pods.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  labels:
    app: backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend-container
          image: your-backend-image
          ports:
            - containerPort: 8080
```

- b. Define a service to expose the back-end pods.

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

2. Create the front-end pods.

- a. Define the front-end pods.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
labels:
  app: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend-container
          image: your-frontend-image
          ports:
            - containerPort: 80
```

- b. Apply the pod definition to your cluster.

```
$ oc apply -f frontend-deployment.yaml
```

3. Configure the front-end to communicate with the back-end.

In your front-end application code, use the DNS name of the back-end service to send requests. For example, if your front-end application needs to fetch data from the back-end pod, your application might include the following code:

```
fetch('http://backend-service.default.svc.cluster.local/api/data')
  .then(response => response.json())
  .then(data => console.log(data));
```

## 1.7. NETWORK CONTROLS

Network controls define rules for how pods are allowed to communicate with each other and with other network endpoints. Network controls are implemented at the network level to ensure that only allowed traffic can flow between pods. This helps secure the cluster by restricting traffic flow and preventing unauthorized access.

- Admin network policies (ANP): ANPs are cluster-scoped custom resource definitions (CRDs). As a cluster administrator, you can use an ANP to define network policies at a cluster level. You cannot override these policies by using regular network policy objects. These policies enforce strict network security rules across the entire cluster. ANPs can specify ingress and egress rules to allow administrators to control the traffic that enters and leaves the cluster.
- Egress firewall: The egress firewall restricts egress traffic leaving the cluster. With this firewall, administrators can limit the external hosts that pods can access from within the cluster. You can configure egress firewall policies to allow or deny traffic to specific IP ranges, DNS names, or external services. This helps prevent unauthorized access to external resources and ensures that only allowed traffic can leave the cluster.
- Ingress node firewall: The ingress node firewall controls ingress traffic to the nodes in a cluster. With this firewall, administrators define rules that restrict which external hosts can initiate connections to the nodes. This helps protect the nodes from unauthorized access and ensures that only trusted traffic can reach the cluster.

## 1.8. ROUTES AND INGRESS

Routes and ingress are both used to expose applications to external traffic. However, they serve slightly different purposes and have different capabilities.

### 1.8.1. Routes

Routes are specific to OpenShift Container Platform resources that expose a service at a host name so that external clients can reach the service by name.

Routes map a host name to a service. Route name mapping allows external clients to access the service using the host name. Routes provide load balancing for the traffic directed to the service. The host name used in a route is resolved to the IP address of the router. Routes then forward the traffic to the appropriate service. Routes can also be secured using SSL/TLS to encrypt traffic between the client and the service.

### 1.8.2. Ingress

Ingress is a resource that provides advanced routing capabilities, including load balancing, SSL/TLS termination, and name-based virtual hosting. Here are some key points about Ingress:

- HTTP/HTTPS routing: You can use Ingress to define rules for routing HTTP and HTTPS traffic to services within the cluster.
- Load balancing: Ingress Controllers, such as NGINX or HAProxy, manage traffic routing and load balancing based on user-defined defined rules.
- SSL/TLS termination: SSL/TLS termination is the process of decrypting incoming SSL/TLS traffic before passing it to the backend services.
- Multiple domains and paths: Ingress supports routing traffic for multiple domains and paths.

### 1.8.3. Comparing Routes and ingress

Routes provide more flexibility and advanced features compared to ingress. This makes routes suitable for complex routing scenarios. Routes are simpler to set up and use, especially for basic external access needs. Ingress is often used for simpler, straightforward external access. Routes are used for more complex scenarios that require advanced routing and SSL/TLS termination.

#### 1.8.4. Example: Configuring routes and ingress to expose a web application

A web application is running on your OpenShift Container Platform cluster. You want to make the application accessible to external users. The application should be accessible through a specific domain name, and the traffic should be securely encrypted using TLS. The following example shows how to configure both routes and ingress to expose your web application to external traffic securely.

##### 1.8.4.1. Configuring Routes

1. Create a new project.

```
$ oc new-project webapp-project
```

2. Deploy the web application.

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git --name=webapp
```

3. Expose the service with a Route.

```
$ oc expose svc/webapp --hostname=webapp.example.com
```

4. Secure the Route with TLS.

- a. Create a TLS secret with your certificate and key.

```
$ oc create secret tls webapp-tls --cert=path/to/tls.crt --key=path/to/tls.key
```

- b. Update the route to use the TLS secret.

```
$ oc patch route/webapp -p '{"spec":{"tls":{"termination":"edge","certificate":"path/to/tls.crt","key":"path/to/tls.key"}}}'
```

##### 1.8.4.2. Configuring ingress

1. Create an ingress resource.

Ensure your ingress Controller is installed and running in the cluster.

2. Create a service for the web application. If not already created, expose the application as a service.

```
apiVersion: v1
kind: Service
metadata:
  name: webapp-service
  namespace: webapp-project
spec:
  selector:
    app: webapp
```

```

ports:
- protocol: TCP
  port: 80
  targetPort: 8080

```

3. Create the ingress resource.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  namespace: webapp-project
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
    - host: webapp.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: webapp-service
                port:
                  number: 80

```

4. Secure the ingress with TLS.

a. Create a TLS secret with your certificate and key.

```
$ oc create secret tls webapp-tls --cert=path/to/tls.crt --key=path/to/tls.key -n webapp-project
```

b. Update the ingress resource to use the TLS secret.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: webapp-ingress
  namespace: webapp-project
spec:
  tls: ①
    - hosts:
        - webapp.example.com
      secretName: webapp-tls ②
  rules:
    - host: webapp.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:

```

```

  name: webapp-service
  port:
    number: 80

```

- 1 The **TLS** section specifies TLS settings.
- 2 The **secretName** field is the name of Kubernetes secret that contains the TLS certificate and key.

## 1.9. SECURITY AND TRAFFIC MANAGEMENT

Administrators can expose applications to external traffic and secure network connections using service types, such as **ClusterIP**, **NodePort**, and **LoadBalancer** and API resources such as **Ingress** and **Route**. The Ingress Operator and Cluster Network Operator (CNO) help configure and manage these services and resources. The Ingress Operator deploys and manages one or more Ingress Controllers. These controllers route external HTTP and HTTPS traffic to services within the cluster. A CNO deploys and manages the cluster network components, including pod networks, service networks, and DNS.

### 1.9.1. Exposing applications

ClusterIP exposes services on an internal IP within the cluster to make the cluster accessible only to other services within the cluster. The NodePort service type exposes the service on a static port on each node's IP. This service type allows external traffic to access the service. Load balancers are typically used in cloud or bare-metal environments that use MetalLB. This service type provisions an external load balancer that routes external traffic to the service. On bare-metal environments, MetalLB uses VIPs and ARP announcements or BGP announcements.

Ingress is an API object that manages external access to services, such as load balancing, SSL/TLS termination, and name-based virtual hosting. An Ingress Controller, such as NGINX or HAProxy, implements the Ingress API and handles traffic routing based on user-defined rules.

### 1.9.2. Securing connections

Ingress Controllers manage SSL/TLS termination to decrypt incoming SSL/TLS traffic before passing it to the backend services. SSL/TLS termination offloads the encryption/decryption process from the application pods. You can use TLS certificates to encrypt traffic between clients and your services. You can manage certificates with tools, such as **cert-manager**, to automate certificate distribution and renewal.

Routes pass TLS traffic to a pod if it has the SNI field. This process allows services that run TCP to be exposed using TLS and not only HTTP/HTTPS. A site administrator can manage the certificates centrally and allow application developers to read private keys even without permission.

The Route API enables encryption of router-to-pod traffic with cluster-managed certificates. This ensures external certificates are centrally managed while the internal leg remains encrypted. Application developers receive unique private keys for their applications. These keys can be mounted as a secret in the pod.

Network controls define rules for how pods can communicate with each other and other network endpoints. This enhances security by controlling traffic flow within the cluster. These controls are implemented at the network plugin level to ensure that only allowed traffic flows between pods.

Role-based access control (RBAC) manages permissions and control who can access resources within the cluster. Service accounts provide identity for pods that access the API. RBAC allows granular control over what each pod can do.

### 1.9.3. Example: Exposing applications and securing connections

In this example, a web application running in your cluster needs to be accessed by external users.

1. Create a service and expose the application as a service using a service type that suits your needs.

```
apiVersion: v1
kind: Service
metadata:
  name: my-web-app
spec:
  type: LoadBalancer
  selector:
    app: my-web-app
  ports:
    - port: 80
      targetPort: 8080
```

2. Define an **Ingress** resource to manage HTTP/HTTPS traffic and route it to your service.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-web-app-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
    - host: mywebapp.example.com
      http:
        paths:
          - path: /
            pathType: Prefix
        backend:
          service:
            name: my-web-app
            port:
              number: 80
```

3. Configure TLS for your ingress to ensure secured, encrypted connections.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-web-app-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  tls:
    - hosts:
```

```

- mywebapp.example.com
secretName: my-tls-secret
rules:
- host: mywebapp.example.com
  http:
    paths:
    - path: /
      pathType: Prefix
    backend:
      service:
        name: my-web-app
        port:
          number: 80

```

#### 1.9.4. Choosing between service types and API resources

Service types and API resources offer different benefits for exposing applications and securing network connections. By leveraging the appropriate service type or API resource, you can effectively manage how your applications are exposed and ensure secure, reliable access for both internal and external clients.

OpenShift Container Platform supports the following service types and API resources:

- Service Types
  - **ClusterIP** is intended for internal-only exposure. It is easy to set up and provides a stable internal IP address for accessing services within the cluster. **ClusterIP** is suitable for communication between services within the cluster.
  - **NodePort** allows external access by exposing the service on each node's IP at a static port. It is straightforward to set up and useful for development and testing. **NodePort** is good for simple external access without the need for a load balancer from the cloud provider.
  - **LoadBalancer** automatically provisions an external load balancer to distribute traffic across multiple nodes. It is ideal for production environments where reliable, high-availability access is needed.
  - **ExternalName** maps a service to an external DNS name to allow services outside the cluster to be accessed using the service's DNS name. It is good for integrating external services or legacy systems with the cluster.
  - Headless service is a DNS name that returns the list of pod IPs without providing a stable **ClusterIP**. This is ideal for stateful applications or scenarios where direct access to individual pod IPs is needed.
- API Resources
  - **Ingress** provides control over routing HTTP and HTTPS traffic, including support for load balancing, SSL/TLS termination, and name-based virtual hosting. It is more flexible than services alone and supports multiple domains and paths. **Ingress** is ideal when complex routing is required.
  - **Route** is similar to **Ingress** but provides additional features, including TLS re-encryption and passthrough. It simplifies the process of exposing services externally. **Route** is best for when you need advanced features, such as integrated certificate management.

If you need a simple way to expose a service to external traffic, **Route** or **Ingress** might be the best

choice. These resources can be managed by a namespace admin or developer. The easiest approach is to create a route, check its external DNS name, and configure your DNS to have a CNAME that points to the external DNS name.

For HTTP/HTTPS/TLS, **Route** or **Ingress** should suffice. Anything else is more complex and requires a cluster admin to ensure ports are accessible or MetalLB is configured. **LoadBalancer** services are also an option in cloud environments or appropriately configured bare-metal environments.

## CHAPTER 2. ACCESSING HOSTS

Learn how to create a bastion host to access OpenShift Container Platform instances and access the control plane nodes with secure shell (SSH) access.

### 2.1. ACCESSING HOSTS ON AMAZON WEB SERVICES IN AN INSTALLER-PROVISIONED INFRASTRUCTURE CLUSTER

The OpenShift Container Platform installer does not create any public IP addresses for any of the Amazon Elastic Compute Cloud (Amazon EC2) instances that it provisions for your OpenShift Container Platform cluster. To be able to SSH to your OpenShift Container Platform hosts, you must follow this procedure.

#### Procedure

1. Create a security group that allows SSH access into the virtual private cloud (VPC) created by the **openshift-install** command.
2. Create an Amazon EC2 instance on one of the public subnets the installer created.
3. Associate a public IP address with the Amazon EC2 instance that you created.  
Unlike with the OpenShift Container Platform installation, you should associate the Amazon EC2 instance you created with an SSH keypair. It does not matter what operating system you choose for this instance, as it will simply serve as an SSH bastion to bridge the internet into your OpenShift Container Platform cluster's VPC. The Amazon Machine Image (AMI) you use does matter. With Red Hat Enterprise Linux CoreOS (RHCOS), for example, you can provide keys via Ignition, like the installer does.
4. After you provisioned your Amazon EC2 instance and can SSH into it, you must add the SSH key that you associated with your OpenShift Container Platform installation. This key can be different from the key for the bastion instance, but does not have to be.



#### NOTE

Direct SSH access is only recommended for disaster recovery. When the Kubernetes API is responsive, run privileged pods instead.

5. Run **oc get nodes**, inspect the output, and choose one of the nodes that is a master. The hostname looks similar to **ip-10-0-1-163.ec2.internal**.
6. From the bastion SSH host you manually deployed into Amazon EC2, SSH into that control plane host. Ensure that you use the same SSH key you specified during the installation:

```
$ ssh -i <ssh-key-path> core@<master-hostname>
```

# CHAPTER 3. NETWORKING DASHBOARDS

Networking metrics are viewable in dashboards within the OpenShift Container Platform web console, under **Observe** → **Dashboards**.

## 3.1. NETWORK OBSERVABILITY OPERATOR

If you have the Network Observability Operator installed, you can view network traffic metrics dashboards by selecting the **Netobserv** dashboard from the **Dashboards** drop-down list. For more information about metrics available in this **Dashboard**, see [Network Observability metrics dashboards](#).

## 3.2. NETWORKING AND OVN-KUBERNETES DASHBOARD

You can view both general networking metrics as well as OVN-Kubernetes metrics from the dashboard.

To view general networking metrics, select **Networking/Linux Subsystem Stats** from the **Dashboards** drop-down list. You can view the following networking metrics from the dashboard: **Network Utilisation**, **Network Saturation**, and **Network Errors**.

To view OVN-Kubernetes metrics select **Networking/Infrastructure** from the **Dashboards** drop-down list. You can view the following OVN-Kubernetes metrics: **Networking Configuration**, **TCP Latency Probes**, **Control Plane Resources**, and **Worker Resources**.

## 3.3. INGRESS OPERATOR DASHBOARD

You can view networking metrics handled by the Ingress Operator from the dashboard. This includes metrics like the following:

- Incoming and outgoing bandwidth
- HTTP error rates
- HTTP server response latency

To view these Ingress metrics, select **Networking/Ingress** from the **Dashboards** drop-down list. You can view Ingress metrics for the following categories: **Top 10 Per Route**, **Top 10 Per Namespace**, and **Top 10 Per Shard**

# CHAPTER 4. NETWORKING OPERATORS

## 4.1. KUBERNETES NMSTATE OPERATOR

The Kubernetes NMState Operator provides a Kubernetes API for performing state-driven network configuration across the OpenShift Container Platform cluster’s nodes with NMState. The Kubernetes NMState Operator provides users with functionality to configure various network interface types, DNS, and routing on cluster nodes. Additionally, the daemons on the cluster nodes periodically report on the state of each node’s network interfaces to the API server.



### IMPORTANT

Red Hat supports the Kubernetes NMState Operator in production environments on bare-metal, IBM Power®, IBM Z®, IBM® LinuxONE, VMware vSphere, and Red Hat OpenStack Platform (RHOSP) installations.

Red Hat support exists for using the Kubernetes NMState Operator on Microsoft Azure but in a limited capacity. Support is limited to configuring DNS servers on your system as a postinstallation task.

Before you can use NMState with OpenShift Container Platform, you must install the Kubernetes NMState Operator.



### NOTE

The Kubernetes NMState Operator updates the network configuration of a secondary NIC. The Operator cannot update the network configuration of the primary NIC, or update the **br-ex** bridge on most on-premise networks.

On a bare-metal platform, using the Kubernetes NMState Operator to update the **br-ex** bridge network configuration is only supported if you set the **br-ex** bridge as the interface in a machine config manifest file. To update the **br-ex** bridge as a postinstallation task, you must set the **br-ex** bridge as the interface in the NMState configuration of the **NodeNetworkConfigurationPolicy** custom resource (CR) for your cluster. For more information, see [Creating a manifest object that includes a customized br-ex bridge](#) in *Postinstallation configuration*.

OpenShift Container Platform uses **nmstate** to report on and configure the state of the node network. This makes it possible to modify the network policy configuration, such as by creating a Linux bridge on all nodes, by applying a single configuration manifest to the cluster.

Node networking is monitored and updated by the following objects:

#### **NodeNetworkState**

Reports the state of the network on that node.

#### **NodeNetworkConfigurationPolicy**

Describes the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** CR to the cluster.

#### **NodeNetworkConfigurationEnactment**

Reports the network policies enacted upon each node.

## 4.1.1. Installing the Kubernetes NMState Operator

You can install the Kubernetes NMState Operator by using the web console or the CLI.

### 4.1.1.1. Installing the Kubernetes NMState Operator by using the web console

You can install the Kubernetes NMState Operator by using the web console. After it is installed, the Operator can deploy the NMState State Controller as a daemon set across all of the cluster nodes.

#### Prerequisites

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

#### Procedure

1. Select **Operators** → **OperatorHub**.
2. In the search field below **All Items**, enter **nmstate** and click **Enter** to search for the Kubernetes NMState Operator.
3. Click on the Kubernetes NMState Operator search result.
4. Click on **Install** to open the **Install Operator** window.
5. Click **Install** to install the Operator.
6. After the Operator finishes installing, click **View Operator**.
7. Under **Provided APIs**, click **Create Instance** to open the dialog box for creating an instance of **kubernetes-nmstate**.
8. In the **Name** field of the dialog box, ensure the name of the instance is **nmstate**.



#### NOTE

The name restriction is a known issue. The instance is a singleton for the entire cluster.

9. Accept the default settings and click **Create** to create the instance.

#### Summary

After you install the Kubernetes NMState Operator, the Operator has deployed the NMState State Controller as a daemon set across all of the cluster nodes.

### 4.1.1.2. Installing the Kubernetes NMState Operator by using the CLI

You can install the Kubernetes NMState Operator by using the OpenShift CLI (**oc**). After it is installed, the Operator can deploy the NMState State Controller as a daemon set across all of the cluster nodes.

#### Prerequisites

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

## Procedure

1. Create the **nmstate** Operator namespace:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-nmstate
spec:
  finalizers:
  - kubernetes
EOF
```

2. Create the **OperatorGroup**:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-nmstate
  namespace: openshift-nmstate
spec:
  targetNamespaces:
  - openshift-nmstate
EOF
```

3. Subscribe to the **nmstate** Operator:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: kubernetes-nmstate-operator
  namespace: openshift-nmstate
spec:
  channel: stable
  installPlanApproval: Automatic
  name: kubernetes-nmstate-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Confirm the **ClusterServiceVersion** (CSV) status for the **nmstate** Operator deployment equals **Succeeded**:

```
$ oc get clusterserviceversion -n openshift-nmstate \
-o custom-columns=Name:.metadata.name,Phase:.status.phase
```

### Example output

Name	Phase
kubernetes-nmstate-operator.4.18.0-202210210157	Succeeded

5. Create an instance of the **nmstate** Operator:

```
$ cat << EOF | oc apply -f -
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
EOF
```

6. Verify that all pods for the NMState Operator are in a **Running** state:

```
$ oc get pod -n openshift-nmstate
```

#### Example output

Name	Ready	Status	Restarts	Age
pod/nmstate-handler-wn55p	1/1	Running	0	77s
pod/nmstate-operator-f6bb869b6-v5m92	1/1	Running	0	4m51s
...				

### 4.1.1.3. Viewing metrics collected by the Kubernetes NMState Operator

The Kubernetes NMState Operator, **kubernetes-nmstate-operator**, can collect metrics from the **kubernetes\_nmstate\_features\_applied** component and expose them as ready-to-use metrics. As a use case for viewing metrics, consider a situation where you created a **NodeNetworkConfigurationPolicy** custom resource and you want to confirm that the policy is active.



#### NOTE

The **kubernetes\_nmstate\_features\_applied** metrics are not an API and might change between OpenShift Container Platform versions.

In the **Developer** and **Administrator** perspectives, the Metrics UI includes some predefined CPU, memory, bandwidth, and network packet queries for the selected project. You can run custom Prometheus Query Language (PromQL) queries for CPU, memory, bandwidth, network packet and application metrics for the project.

The following example demonstrates a **NodeNetworkConfigurationPolicy** manifest example that is applied to an OpenShift Container Platform cluster:

```
# ...
interfaces:
- name: br1
  type: linux-bridge
  state: up
  ipv4:
    enabled: true
    dhcp: true
    dhcp-custom-hostname: foo
  bridge:
    options:
      stp:
```

```

    enabled: false
  port: []
# ...

```

The **NodeNetworkConfigurationPolicy** manifest exposes metrics and makes them available to the Cluster Monitoring Operator (CMO). The following example shows some exposed metrics:

```

controller_runtime_reconcile_time_seconds_bucket{controller="nodenetworkconfigurationenactment",le="0.005"} 16
controller_runtime_reconcile_time_seconds_bucket{controller="nodenetworkconfigurationenactment",le="0.01"} 16
controller_runtime_reconcile_time_seconds_bucket{controller="nodenetworkconfigurationenactment",le="0.025"} 16
...
# HELP kubernetes_nmstate_features_applied Number of nmstate features applied labeled by its name
# TYPE kubernetes_nmstate_features_applied gauge
kubernetes_nmstate_features_applied{name="dhcipv4-custom-hostname"} 1

```

## Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in to the web console as the administrator and installed the Kubernetes NMState Operator.
- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.
- You have enabled monitoring for user-defined projects.
- You have deployed a service in a user-defined project.
- You have created a **NodeNetworkConfigurationPolicy** manifest and applied it to your cluster.

## Procedure

1. If you want to view the metrics from the **Developer** perspective in the OpenShift Container Platform web console, complete the following tasks:
  - a. Click **Observe**.
  - b. To view the metrics of a specific project, select the project in the **Project:** list. For example, **openshift-nmstate**.
  - c. Click the **Metrics** tab.
  - d. To visualize the metrics on the plot, select a query from the **Select query** list or create a custom PromQL query based on the selected query by selecting **Show PromQL**.



### NOTE

In the **Developer** perspective, you can only run one query at a time.

2. If you want to view the metrics from the **Administrator** perspective in the OpenShift Container Platform web console, complete the following tasks:
  - a. Click **Observe** → **Metrics**.
  - b. Enter **kubernetes\_nmstate\_features\_applied** in the **Expression** field.
  - c. Click **Add query** and then **Run queries**.
3. To explore the visualized metrics, do any of the following tasks:
  - a. To zoom into the plot and change the time range, do any of the following tasks:
    - To visually select the time range, click and drag on the plot horizontally.
    - To select the time range, use the menu which is in the upper left of the console.
  - b. To reset the time range, select **Reset zoom**.
  - c. To display the output for all the queries at a specific point in time, hold the mouse cursor on the plot at that point. The query output displays in a pop-up box.

#### 4.1.2. Uninstalling the Kubernetes NMState Operator

You can use the Operator Lifecycle Manager (OLM) to uninstall the Kubernetes NMState Operator, but by design OLM does not delete any associated custom resource definitions (CRDs), custom resources (CRs), or API Services.

Before you uninstall the Kubernetes NMState Operator from the **Subscription** resource used by OLM, identify what Kubernetes NMState Operator resources to delete. This identification ensures that you can delete resources without impacting your running cluster.

If you need to reinstall the Kubernetes NMState Operator, see "Installing the Kubernetes NMState Operator by using the CLI" or "Installing the Kubernetes NMState Operator by using the web console".

#### Prerequisites

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

#### Procedure

1. Unsubscribe the Kubernetes NMState Operator from the **Subscription** resource by running the following command:
 

```
$ oc delete --namespace openshift-nmstate subscription kubernetes-nmstate-operator
```
2. Find the **ClusterServiceVersion** (CSV) resource that associates with the Kubernetes NMState Operator:
 

```
$ oc get --namespace openshift-nmstate clusterserviceversion
```

#### Example output that lists a CSV resource

NAME	DISPLAY	VERSION	REPLACES	PHASE
kubernetes-nmstate-operator.v4.18.0	Kubernetes NMState Operator	4.18.0		
Succeeded				

3. Delete the CSV resource. After you delete the file, OLM deletes certain resources, such as **RBAC**, that it created for the Operator.

```
$ oc delete --namespace openshift-nmstate clusterserviceversion kubernetes-nmstate-operator.v4.18.0
```

4. Delete the **nmstate** CR and any associated **Deployment** resources by running the following commands:

```
$ oc -n openshift-nmstate delete nmstate nmstate
```

```
$ oc delete --all deployments --namespace=openshift-nmstate
```

5. After you deleted the **nmstate** CR, remove the **nmstate-console-plugin** console plugin name from the **console.operator.openshift.io/cluster** CR.

- a. Store the position of the **nmstate-console-plugin** entry that exists among the list of enable plugins by running the following command. The following command uses the **jq** CLI tool to store the index of the entry in an environment variable named **INDEX**:

```
INDEX=$(oc get console.operator.openshift.io cluster -o json | jq -r '.spec.plugins | to_entries[] | select(.value == "nmstate-console-plugin") | .key')
```

- b. Remove the **nmstate-console-plugin** entry from the **console.operator.openshift.io/cluster** CR by running the following patch command:

```
$ oc patch console.operator.openshift.io cluster --type=json -p "[{\\"op\\": \"remove\", \\"path\\": \"/spec/plugins/$INDEX\"]}" ①
```

① **INDEX** is an auxiliary variable. You can specify a different name for this variable.

6. Delete all the custom resource definitions (CRDs), such as **nmstates.nmstate.io**, by running the following commands:

```
$ oc delete crd nmstates.nmstate.io
```

```
$ oc delete crd nodenetworkconfigurationenactments.nmstate.io
```

```
$ oc delete crd nodenetworkstates.nmstate.io
```

```
$ oc delete crd nodenetworkconfigurationpolicies.nmstate.io
```

7. Delete the namespace:

```
$ oc delete namespace kubernetes-nmstate
```

### 4.1.3. Additional resources

- [Creating an interface on nodes](#)

### 4.1.4. Next steps

- [Observing and updating the node network state and configuration](#)

## 4.2. AWS LOAD BALANCER OPERATOR

### 4.2.1. AWS Load Balancer Operator release notes

The AWS Load Balancer (ALB) Operator deploys and manages an instance of the **AWSLoadBalancerController** resource.



#### IMPORTANT

The AWS Load Balancer (ALB) Operator is only supported on the **x86\_64** architecture.

These release notes track the development of the AWS Load Balancer Operator in OpenShift Container Platform.

For an overview of the AWS Load Balancer Operator, see [AWS Load Balancer Operator in OpenShift Container Platform](#).



#### NOTE

AWS Load Balancer Operator currently does not support AWS GovCloud.

#### 4.2.1.1. AWS Load Balancer Operator 1.2.0

The following advisory is available for the AWS Load Balancer Operator version 1.2.0:

- [RHEA-2025:0034 Release of AWS Load Balancer Operator 1.2.z on OperatorHub](#)

##### 4.2.1.1.1. Notable changes

- This release supports the AWS Load Balancer Controller version 2.8.2.
- With this release, the platform tags defined in the **Infrastructure** resource will now be added to all AWS objects created by the controller.

#### 4.2.1.2. AWS Load Balancer Operator 1.1.1

The following advisory is available for the AWS Load Balancer Operator version 1.1.1:

- [RHEA-2024:0555 Release of AWS Load Balancer Operator 1.1.z on OperatorHub](#)

#### 4.2.1.3. AWS Load Balancer Operator 1.1.0

The AWS Load Balancer Operator version 1.1.0 supports the AWS Load Balancer Controller version 2.4.4.

The following advisory is available for the AWS Load Balancer Operator version 1.1.0:

- [RHEA-2023:6218 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update](#)

#### 4.2.1.3.1. Notable changes

- This release uses the Kubernetes API version 0.27.2.

#### 4.2.1.3.2. New features

- The AWS Load Balancer Operator now supports a standardized Security Token Service (STS) flow by using the Cloud Credential Operator.

#### 4.2.1.3.3. Bug fixes

- A FIPS-compliant cluster must use TLS version 1.2. Previously, webhooks for the AWS Load Balancer Controller only accepted TLS 1.3 as the minimum version, resulting in an error such as the following on a FIPS-compliant cluster:

remote error: tls: protocol version not supported

Now, the AWS Load Balancer Controller accepts TLS 1.2 as the minimum TLS version, resolving this issue. ([OCPBUGS-14846](#))

#### 4.2.1.4. AWS Load Balancer Operator 1.0.1

The following advisory is available for the AWS Load Balancer Operator version 1.0.1:

- [Release of AWS Load Balancer Operator 1.0.1 on OperatorHub](#)

#### 4.2.1.5. AWS Load Balancer Operator 1.0.0

The AWS Load Balancer Operator is now generally available with this release. The AWS Load Balancer Operator version 1.0.0 supports the AWS Load Balancer Controller version 2.4.4.

The following advisory is available for the AWS Load Balancer Operator version 1.0.0:

- [RHEA-2023:1954 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update](#)



#### IMPORTANT

The AWS Load Balancer (ALB) Operator version 1.x.x cannot upgrade automatically from the Technology Preview version 0.x.x. To upgrade from an earlier version, you must uninstall the ALB operands and delete the **aws-load-balancer-operator** namespace.

#### 4.2.1.5.1. Notable changes

- This release uses the new **v1** API version.

#### 4.2.1.5.2. Bug fixes

- Previously, the controller provisioned by the AWS Load Balancer Operator did not properly use the configuration for the cluster-wide proxy. These settings are now applied appropriately to the controller. ([OCPBUGS-4052](#), [OCPBUGS-5295](#))

#### 4.2.1.6. Earlier versions

The two earliest versions of the AWS Load Balancer Operator are available as a Technology Preview. These versions should not be used in a production cluster. For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

The following advisory is available for the AWS Load Balancer Operator version 0.2.0:

- [RHEA-2022:9084 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update](#)

The following advisory is available for the AWS Load Balancer Operator version 0.0.1:

- [RHEA-2022:5780 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update](#)

#### 4.2.2. AWS Load Balancer Operator in OpenShift Container Platform

The AWS Load Balancer Operator deploys and manages the AWS Load Balancer Controller. You can install the AWS Load Balancer Operator from OperatorHub by using OpenShift Container Platform web console or CLI.

##### 4.2.2.1. AWS Load Balancer Operator considerations

Review the following limitations before installing and using the AWS Load Balancer Operator:

- The IP traffic mode only works on AWS Elastic Kubernetes Service (EKS). The AWS Load Balancer Operator disables the IP traffic mode for the AWS Load Balancer Controller. As a result of disabling the IP traffic mode, the AWS Load Balancer Controller cannot use the pod readiness gate.
- The AWS Load Balancer Operator adds command-line flags such as **--disable-ingress-class-annotation** and **--disable-ingress-group-name-annotation** to the AWS Load Balancer Controller. Therefore, the AWS Load Balancer Operator does not allow using the **kubernetes.io/ingress.class** and **alb.ingress.kubernetes.io/group.name** annotations in the **Ingress** resource.
- You have configured the AWS Load Balancer Operator so that the SVC type is **NodePort** (not **LoadBalancer** or **ClusterIP**).

##### 4.2.2.2. AWS Load Balancer Operator

The AWS Load Balancer Operator can tag the public subnets if the **kubernetes.io/role/elb** tag is missing. Also, the AWS Load Balancer Operator detects the following information from the underlying AWS cloud:

- The ID of the virtual private cloud (VPC) on which the cluster hosting the Operator is deployed in.
- Public and private subnets of the discovered VPC.

The AWS Load Balancer Operator supports the Kubernetes service resource of type **LoadBalancer** by using Network Load Balancer (NLB) with the **instance** target type only.

## Procedure

1. You can deploy the AWS Load Balancer Operator on demand from OperatorHub, by creating a **Subscription** object by running the following command:

```
$ oc -n aws-load-balancer-operator get sub aws-load-balancer-operator --template='{{.status.installplan.name}}\n'
```

### Example output

```
install-zlfbt
```

2. Check if the status of an install plan is **Complete** by running the following command:

```
$ oc -n aws-load-balancer-operator get ip <install_plan_name> --template='{{.status.phase}}\n'
```

### Example output

```
Complete
```

3. View the status of the **aws-load-balancer-operator-controller-manager** deployment by running the following command:

```
$ oc get -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-manager
```

### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
aws-load-balancer-operator-controller-manager	1/1	1	1	23h

### 4.2.2.3. Using the AWS Load Balancer Operator in an AWS VPC cluster extended into an Outpost

You can configure the AWS Load Balancer Operator to provision an AWS Application Load Balancer in an AWS VPC cluster extended into an Outpost. AWS Outposts does not support AWS Network Load Balancers. As a result, the AWS Load Balancer Operator cannot provision Network Load Balancers in an Outpost.

You can create an AWS Application Load Balancer either in the cloud subnet or in the Outpost subnet. An Application Load Balancer in the cloud can attach to cloud-based compute nodes and an Application Load Balancer in the Outpost can attach to edge compute nodes. You must annotate Ingress resources with the Outpost subnet or the VPC subnet, but not both.

## Prerequisites

- You have extended an AWS VPC cluster into an Outpost.

- You have installed the OpenShift CLI (**oc**).
- You have installed the AWS Load Balancer Operator and created the AWS Load Balancer Controller.

## Procedure

- Configure the **Ingress** resource to use a specified subnet:

### Example Ingress resource configuration

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <application_name>
  annotations:
    alb.ingress.kubernetes.io/subnets: <subnet_id> ①
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <application_name>
                port:
                  number: 80
```

- ① Specifies the subnet to use.

- To use the Application Load Balancer in an Outpost, specify the Outpost subnet ID.
- To use the Application Load Balancer in the cloud, you must specify at least two subnets in different availability zones.

### 4.2.3. Preparing an AWS STS cluster for the AWS Load Balancer Operator

You can install the Amazon Web Services (AWS) Load Balancer Operator on a cluster that uses the Security Token Service (STS). Follow these steps to prepare your cluster before installing the Operator.

The AWS Load Balancer Operator relies on the **CredentialsRequest** object to bootstrap the Operator and the AWS Load Balancer Controller. The AWS Load Balancer Operator waits until the required secrets are created and available.

#### 4.2.3.1. Prerequisites

- You installed the OpenShift CLI (**oc**).
- You know the infrastructure ID of your cluster. To show this ID, run the following command in your CLI:

```
$ oc get infrastructure cluster -o=jsonpath="{.status.infrastructureName}"
```

- You know the OpenID Connect (OIDC) DNS information for your cluster. To show this information, enter the following command in your CLI:

```
$ oc get authentication.config cluster -o=jsonpath="{.spec.serviceAccountIssuer}"
```

- 1** An OIDC DNS example is <https://rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f>.

- You logged into the AWS Web Console, navigated to **IAM** → **Access management** → **Identity providers**, and located the OIDC Amazon Resource Name (ARN) information. An OIDC ARN example is `arn:aws:iam::777777777777:oidc-provider/<oidc_dns_url>`.

#### 4.2.3.2. Creating an IAM role for the AWS Load Balancer Operator

An additional Amazon Web Services (AWS) Identity and Access Management (IAM) role is required to successfully install the AWS Load Balancer Operator on a cluster that uses STS. The IAM role is required to interact with subnets and Virtual Private Clouds (VPCs). The AWS Load Balancer Operator generates the **CredentialsRequest** object with the IAM role to bootstrap itself.

You can create the IAM role by using the following options:

- Using the [Cloud Credential Operator utility \(ccocctl\)](#) and a predefined **CredentialsRequest** object.
- Using the AWS CLI and predefined AWS manifests.

Use the AWS CLI if your environment does not support the **ccocctl** command.

##### 4.2.3.2.1. Creating an AWS IAM role by using the Cloud Credential Operator utility

You can use the Cloud Credential Operator utility (**ccocctl**) to create an AWS IAM role for the AWS Load Balancer Operator. An AWS IAM role interacts with subnets and Virtual Private Clouds (VPCs).

#### Prerequisites

- You must extract and prepare the **ccocctl** binary.

#### Procedure

1. Download the **CredentialsRequest** custom resource (CR) and store it in a directory by running the following command:

```
$ curl --create-dirs -o <credentials_requests_dir>/operator.yaml
https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/main/hack/operator-credentials-request.yaml
```

2. Use the **ccocctl** utility to create an AWS IAM role by running the following command:

```
$ ccocctl aws create-iam-roles \
--name <name> \
--region=<aws_region> \
--credentials-requests-dir=<credentials_requests_dir> \
--identity-provider-arn <oidc_arn>
```

## Example output

```
2023/09/12 11:38:57 Role arn:aws:iam::777777777777:role/<name>-aws-load-balancer-operator-aws-load-balancer-operator created ①
2023/09/12 11:38:57 Saved credentials configuration to:
/home/user/<credentials_requests_dir>/manifests/aws-load-balancer-operator-aws-load-balancer-operator-credentials.yaml
2023/09/12 11:38:58 Updated Role policy for Role <name>-aws-load-balancer-operator-aws-load-balancer-operator created
```

- ① Note the Amazon Resource Name (ARN) of an AWS IAM role that was created for the AWS Load Balancer Operator, such as **arn:aws:iam::777777777777:role/<name>-aws-load-balancer-operator-aws-load-balancer-operator**.



### NOTE

The length of an AWS IAM role name must be less than or equal to 12 characters.

#### 4.2.3.2.2. Creating an AWS IAM role by using the AWS CLI

You can use the AWS Command Line Interface to create an IAM role for the AWS Load Balancer Operator. The IAM role is used to interact with subnets and Virtual Private Clouds (VPCs).

#### Prerequisites

- You must have access to the AWS Command Line Interface (**aws**).

#### Procedure

1. Generate a trust policy file by using your identity provider by running the following command:

```
$ cat <<EOF > albo-operator-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "<oidc_arn>" ①
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "<cluster_oidc_endpoint>:sub": "system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-operator-controller-manager" ②
                }
            }
        ]
    }
}
EOF
```

- 1 Specifies the Amazon Resource Name (ARN) of the OIDC identity provider, such as **arn:aws:iam::777777777777:oidc-provider/rh-oidc.s3.us-east-**
- 2 Specifies the service account for the AWS Load Balancer Controller. An example of **<cluster\_oidc\_endpoint>** is **rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f.**

2. Create the IAM role with the generated trust policy by running the following command:

```
$ aws iam create-role --role-name albo-operator --assume-role-policy-document file://albo-operator-trust-policy.json
```

### Example output

```
ROLE arn:aws:iam::<aws_account_number>:role/albo-operator 2023-08-02T12:13:22Z ①
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-controller-manager
PRINCIPAL arn:aws:iam:<aws_account_number>:oidc-provider/<cluster_oidc_endpoint>
```

- 1 Note the ARN of the created AWS IAM role that was created for the AWS Load Balancer Operator, such as **arn:aws:iam::777777777777:role/albo-operator**.

3. Download the permission policy for the AWS Load Balancer Operator by running the following command:

```
$ curl -o albo-operator-permission-policy.json
https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/main/hack/operator-permission-policy.json
```

4. Attach the permission policy for the AWS Load Balancer Controller to the IAM role by running the following command:

```
$ aws iam put-role-policy --role-name albo-operator --policy-name perms-policy-albo-operator --policy-document file://albo-operator-permission-policy.json
```

### 4.2.3.3. Configuring the ARN role for the AWS Load Balancer Operator

You can configure the Amazon Resource Name (ARN) role for the AWS Load Balancer Operator as an environment variable. You can configure the ARN role by using the CLI.

#### Prerequisites

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

#### Procedure

1. Create the **aws-load-balancer-operator** project by running the following command:

```
$ oc new-project aws-load-balancer-operator
```

2. Create the **OperatorGroup** object by running the following command:

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  targetNamespaces: []
EOF
```

3. Create the **Subscription** object by running the following command:

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
      - name: ROLEARN
        value: "<albo_role_arn>" ①
EOF
```

- ① Specifies the ARN role to be used in the **CredentialsRequest** to provision the AWS credentials for the AWS Load Balancer Operator. An example for **<albo\_role\_arn>** is **arn:aws:iam::<aws\_account\_number>:role/albo-operator**.



#### NOTE

The AWS Load Balancer Operator waits until the secret is created before moving to the **Available** status.

#### 4.2.3.4. Creating an IAM role for the AWS Load Balancer Controller

The **CredentialsRequest** object for the AWS Load Balancer Controller must be set with a manually provisioned IAM role.

You can create the IAM role by using the following options:

- Using [the Cloud Credential Operator utility \(\*\*ccocctl\*\*\)](#) and a predefined **CredentialsRequest** object.
- Using the AWS CLI and predefined AWS manifests.

Use the AWS CLI if your environment does not support the **ccocctl** command.

#### 4.2.3.4.1. Creating an AWS IAM role for the controller by using the Cloud Credential Operator utility

You can use the Cloud Credential Operator utility (**ccocctl**) to create an AWS IAM role for the AWS Load Balancer Controller. An AWS IAM role is used to interact with subnets and Virtual Private Clouds (VPCs).

##### Prerequisites

- You must extract and prepare the **ccocctl** binary.

##### Procedure

1. Download the **CredentialsRequest** custom resource (CR) and store it in a directory by running the following command:

```
$ curl --create-dirs -o <credentials_requests_dir>/controller.yaml
https://raw.githubusercontent.com/openshift/aws-load-balancer-
operator/main/hack/controller/controller-credentials-request.yaml
```

2. Use the **ccocctl** utility to create an AWS IAM role by running the following command:

```
$ ccocctl aws create-iam-roles \
--name <name> \
--region=<aws_region> \
--credentials-requests-dir=<credentials_requests_dir> \
--identity-provider-arn <oidc_arn>
```

##### Example output

```
2023/09/12 11:38:57 Role arn:aws:iam::777777777777:role/<name>-aws-load-balancer-
operator-aws-load-balancer-controller created ①
2023/09/12 11:38:57 Saved credentials configuration to:
/home/user/<credentials_requests_dir>/manifests/aws-load-balancer-operator-aws-load-
balancer-controller-credentials.yaml
2023/09/12 11:38:58 Updated Role policy for Role <name>-aws-load-balancer-operator-aws-
load-balancer-controller created
```

1

Note the Amazon Resource Name (ARN) of an AWS IAM role that was created for the AWS Load Balancer Controller, such as **arn:aws:iam::777777777777:role/<name>-aws-
load-balancer-operator-aws-load-balancer-controller**.



##### NOTE

The length of an AWS IAM role name must be less than or equal to 12 characters.

#### 4.2.3.4.2. Creating an AWS IAM role for the controller by using the AWS CLI

You can use the AWS command-line interface to create an AWS IAM role for the AWS Load Balancer Controller. An AWS IAM role is used to interact with subnets and Virtual Private Clouds (VPCs).

##### Prerequisites

- You must have access to the AWS command-line interface (**aws**).

## Procedure

1. Generate a trust policy file using your identity provider by running the following command:

```
$ cat <<EOF > albo-controller-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "<oidc_arn>" 1
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "<cluster_oidc_endpoint>:sub": "system:serviceaccount:aws-load-balancer-
operator:aws-load-balancer-operator-controller-manager" 2
                }
            }
        }
    ]
}
EOF
```

- 1** Specifies the Amazon Resource Name (ARN) of the OIDC identity provider, such as **arn:aws:iam::777777777777:oidc-provider/rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f**.
- 2** Specifies the service account for the AWS Load Balancer Controller. An example of **<cluster\_oidc\_endpoint>** is **rh-oidc.s3.us-east-1.amazonaws.com/28292va7ad7mr9r4he1fb09b14t59t4f**.

2. Create an AWS IAM role with the generated trust policy by running the following command:

```
$ aws iam create-role --role-name albo-controller --assume-role-policy-document file://albo-
controller-trust-policy.json
```

### Example output

```
ROLE arn:aws:iam:<aws_account_number>:role/albo-controller 2023-08-02T12:13:22Z 1
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-
operator-controller-manager
PRINCIPAL arn:aws:iam:<aws_account_number>:oidc-provider/<cluster_oidc_endpoint>
```

- 1** Note the ARN of an AWS IAM role for the AWS Load Balancer Controller, such as **arn:aws:iam::777777777777:role/albo-controller**.
3. Download the permission policy for the AWS Load Balancer Controller by running the following command:

```
$ curl -o albo-controller-permission-policy.json
https://raw.githubusercontent.com/openshift/aws-load-balancer-operator/main/assets/iam-
policy.json
```

4. Attach the permission policy for the AWS Load Balancer Controller to an AWS IAM role by running the following command:

```
$ aws iam put-role-policy --role-name albo-controller --policy-name perms-policy-albo-
controller --policy-document file://albo-controller-permission-policy.json
```

5. Create a YAML file that defines the **AWSLoadBalancerController** object:

#### Example sample-aws-lb-manual-creds.yaml file

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController 1
metadata:
  name: cluster 2
spec:
  credentialsRequestConfig:
    stsIAMRoleARN: <albc_role_arn> 3
```

- 1** Defines the **AWSLoadBalancerController** object.
- 2** Defines the AWS Load Balancer Controller name. All related resources use this instance name as a suffix.
- 3** Specifies the ARN role for the AWS Load Balancer Controller. The **CredentialsRequest** object uses this ARN role to provision the AWS credentials. An example of **<albc\_role\_arn>** is **arn:aws:iam::777777777777:role/albo-controller**.

#### 4.2.3.5. Additional resources

- [Configuring the Cloud Credential Operator utility](#)

#### 4.2.4. Installing the AWS Load Balancer Operator

The AWS Load Balancer Operator deploys and manages the AWS Load Balancer Controller. You can install the AWS Load Balancer Operator from the OperatorHub by using OpenShift Container Platform web console or CLI.

##### 4.2.4.1. Installing the AWS Load Balancer Operator by using the web console

You can install the AWS Load Balancer Operator by using the web console.

#### Prerequisites

- You have logged in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- Your cluster is configured with AWS as the platform type and cloud provider.

- If you are using a security token service (STS) or user-provisioned infrastructure, follow the related preparation steps. For example, if you are using AWS Security Token Service, see "Preparing for the AWS Load Balancer Operator on a cluster using the AWS Security Token Service (STS)".

## Procedure

1. Navigate to **Operators → OperatorHub** in the OpenShift Container Platform web console.
2. Select the **AWS Load Balancer Operator**. You can use the **Filter by keyword** text box or use the filter list to search for the AWS Load Balancer Operator from the list of Operators.
3. Select the **aws-load-balancer-operator** namespace.
4. On the **Install Operator** page, select the following options:
  - a. **Update the channel** as **stable-v1**.
  - b. **Installation mode** as **All namespaces on the cluster (default)**
  - c. **Installed Namespace** as **aws-load-balancer-operator**. If the **aws-load-balancer-operator** namespace does not exist, it gets created during the Operator installation.
  - d. Select **Update approval** as **Automatic** or **Manual**. By default, the **Update approval** is set to **Automatic**. If you select automatic updates, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention. If you select manual updates, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to update the Operator updated to the new version.
5. Click **Install**.

## Verification

- Verify that the AWS Load Balancer Operator shows the **Status** as **Succeeded** on the Installed Operators dashboard.

### 4.2.4.2. Installing the AWS Load Balancer Operator by using the CLI

You can install the AWS Load Balancer Operator by using the CLI.

## Prerequisites

- You are logged in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- Your cluster is configured with AWS as the platform type and cloud provider.
- You are logged into the OpenShift CLI (**oc**).

## Procedure

1. Create a **Namespace** object:
  - a. Create a YAML file that defines the **Namespace** object:

### Example namespace.yaml file

```
apiVersion: v1
kind: Namespace
metadata:
  name: aws-load-balancer-operator
```

- b. Create the **Namespace** object by running the following command:

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

2. Create an **OperatorGroup** object:

- a. Create a YAML file that defines the **OperatorGroup** object:

### Example operatorgroup.yaml file

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: aws-lb-operatorgroup
  namespace: aws-load-balancer-operator
spec:
  upgradeStrategy: Default
```

- b. Create the **OperatorGroup** object by running the following command:

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

3. Create a **Subscription** object:

- a. Create a YAML file that defines the **Subscription** object:

### Example subscription.yaml file

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: aws-load-balancer-operator
  namespace: aws-load-balancer-operator
spec:
  channel: stable-v1
  installPlanApproval: Automatic
  name: aws-load-balancer-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

- b. Create the **Subscription** object by running the following command:

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

## Verification

- Get the name of the install plan from the subscription:

```
$ oc -n aws-load-balancer-operator \
get subscription aws-load-balancer-operator \
--template='{{.status.installplan.name}}{{"\n"}}'
```

- Check the status of the install plan:

```
$ oc -n aws-load-balancer-operator \
get ip <install_plan_name> \
--template='{{.status.phase}}{{"\n"}}'
```

The output must be **Complete**.

#### 4.2.4.3. Creating the AWS Load Balancer Controller

You can install only a single instance of the **AWSLoadBalancerController** object in a cluster. You can create the AWS Load Balancer Controller by using CLI. The AWS Load Balancer Operator reconciles only the **cluster** named resource.

##### Prerequisites

- You have created the **echoserver** namespace.
- You have access to the OpenShift CLI (**oc**).

##### Procedure

- Create a YAML file that defines the **AWSLoadBalancerController** object:

##### Example sample-aws-lb.yaml file

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController 1
metadata:
  name: cluster 2
spec:
  subnetTagging: Auto 3
  additionalResourceTags: 4
    - key: example.org/security-scope
      value: staging
  ingressClass: alb 5
  config:
    replicas: 2 6
    enabledAddons: 7
      - AWSWAFv2 8
```

- Defines the **AWSLoadBalancerController** object.
- Defines the AWS Load Balancer Controller name. This instance name gets added as a suffix to all related resources.
- Configures the subnet tagging method for the AWS Load Balancer Controller. The following values are valid:

following values are valid:

- **Auto:** The AWS Load Balancer Operator determines the subnets that belong to the cluster and tags them appropriately. The Operator cannot determine the role correctly if the internal subnet tags are not present on internal subnet.
- **Manual:** You manually tag the subnets that belong to the cluster with the appropriate role tags. Use this option if you installed your cluster on user-provided infrastructure.

- 4 Defines the tags used by the AWS Load Balancer Controller when it provisions AWS resources.
- 5 Defines the ingress class name. The default value is **alb**.
- 6 Specifies the number of replicas of the AWS Load Balancer Controller.
- 7 Specifies annotations as an add-on for the AWS Load Balancer Controller.
- 8 Enables the **alb.ingress.kubernetes.io/wafv2-acl-arn** annotation.

2. Create the **AWSLoadBalancerController** object by running the following command:

```
$ oc create -f sample-aws-lb.yaml
```

3. Create a YAML file that defines the **Deployment** resource:

#### Example **sample-aws-lb.yaml** file

```
apiVersion: apps/v1
kind: Deployment 1
metadata:
  name: <echoserver> 2
  namespace: echoserver
spec:
  selector:
    matchLabels:
      app: echoserver
  replicas: 3 3
  template:
    metadata:
      labels:
        app: echoserver
    spec:
      containers:
        - image: openshift/origin-node
          command:
            - "/bin/socat"
          args:
            - TCP4-LISTEN:8080,reuseaddr,fork
            - EXEC:'/bin/bash -c \'printf \\\"HTTP/1.0 200 OK\\r\\n\\r\\n\\\"; sed -e \\\"/^\\r/q\\\"\''
      imagePullPolicy: Always
      name: echoserver
      ports:
        - containerPort: 8080
```

- 1 Defines the deployment resource.
- 2 Specifies the deployment name.
- 3 Specifies the number of replicas of the deployment.

4. Create a YAML file that defines the **Service** resource:

#### Example service-albo.yaml file

```
apiVersion: v1
kind: Service 1
metadata:
  name: <echoserver> 2
  namespace: echoserver
spec:
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
    type: NodePort
  selector:
    app: echoserver
```

- 1 Defines the service resource.
- 2 Specifies the service name.

5. Create a YAML file that defines the **Ingress** resource:

#### Example ingress-albo.yaml file

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <name> 1
  namespace: echoserver
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Exact
            backend:
              service:
                name: <echoserver> 2
                port:
                  number: 80
```

- 1 Specify a name for the **Ingress** resource.
- 2 Specifies the service name.

## Verification

- Save the status of the **Ingress** resource in the **HOST** variable by running the following command:

```
$ HOST=$(oc get ingress -n echoserver echoserver --template='{{(index .status.loadBalancer.ingress 0).hostname}}')
```

- Verify the status of the **Ingress** resource by running the following command:

```
$ curl $HOST
```

## 4.2.5. Configuring the AWS Load Balancer Operator

### 4.2.5.1. Trusting the certificate authority of the cluster-wide proxy

You can configure the cluster-wide proxy in the AWS Load Balancer Operator. After configuring the cluster-wide proxy, Operator Lifecycle Manager (OLM) automatically updates all the deployments of the Operators with the environment variables such as **HTTP\_PROXY**, **HTTPS\_PROXY**, and **NO\_PROXY**. These variables are populated to the managed controller by the AWS Load Balancer Operator.

- 1 Create the config map to contain the certificate authority (CA) bundle in the **aws-load-balancer-operator** namespace by running the following command:

```
$ oc -n aws-load-balancer-operator create configmap trusted-ca
```

- 2 To inject the trusted CA bundle into the config map, add the **config.openshift.io/inject-trusted-cabundle=true** label to the config map by running the following command:

```
$ oc -n aws-load-balancer-operator label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true
```

- 3 Update the AWS Load Balancer Operator subscription to access the config map in the AWS Load Balancer Operator deployment by running the following command:

```
$ oc -n aws-load-balancer-operator patch subscription aws-load-balancer-operator --type='merge' -p '{"spec":{"config":{"env":[{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}],"volumes":[{"name":"trusted-ca","configMap":{"name":"trusted-ca"}],"volumeMounts":[{"name":"trusted-ca","mountPath":"/etc/pki/tls/certs/albo-tls-ca-bundle.crt","subPath":"ca-bundle.crt"}]}}}'
```

- 4 After the AWS Load Balancer Operator is deployed, verify that the CA bundle is added to the **aws-load-balancer-operator-controller-manager** deployment by running the following command:

```
$ oc -n aws-load-balancer-operator exec deploy/aws-load-balancer-operator-controller-manager -c manager -- bash -c "ls -l /etc/pki/tls/certs/albo-tls-ca-bundle.crt; printenv TRUSTED_CA_CONFIGMAP_NAME"
```

### Example output

```
-rw-r--r--. 1 root 1000690000 5875 Jan 11 12:25 /etc/pki/tls/certs/albo-tls-ca-bundle.crt  
trusted-ca
```

5. Optional: Restart deployment of the AWS Load Balancer Operator every time the config map changes by running the following command:

```
$ oc -n aws-load-balancer-operator rollout restart deployment/aws-load-balancer-operator-controller-manager
```

### Additional resources

- [Certificate injection using Operators](#)

#### 4.2.5.2. Adding TLS termination on the AWS Load Balancer

You can route the traffic for the domain to pods of a service and add TLS termination on the AWS Load Balancer.

### Prerequisites

- You have an access to the OpenShift CLI (**oc**).

### Procedure

1. Create a YAML file that defines the **AWSLoadBalancerController** resource:

#### Example add-tls-termination-albc.yaml file

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController
metadata:
  name: cluster
spec:
  subnetTagging: Auto
  ingressClass: tls-termination ①
```

- 1 Defines the ingress class name. If the ingress class is not present in your cluster the AWS Load Balancer Controller creates one. The AWS Load Balancer Controller reconciles the additional ingress class values if **spec.controller** is set to **ingress.k8s.aws/alb**.

2. Create a YAML file that defines the **Ingress** resource:

#### Example add-tls-termination-ingress.yaml file

```
apiVersion: networking.k8s.io/v1
```

```

kind: Ingress
metadata:
  name: <example> ①
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing ②
    alb.ingress.kubernetes.io/certificate-arn: arn:aws:acm:us-west-2:xxxxxx ③
spec:
  ingressClassName: tls-termination ④
  rules:
  - host: example.com ⑤
    http:
      paths:
      - path: /
        pathType: Exact
        backend:
          service:
            name: <example_service> ⑥
            port:
              number: 80

```

- ① Specifies the ingress name.
- ② The controller provisions the load balancer for ingress in a public subnet to access the load balancer over the internet.
- ③ The Amazon Resource Name (ARN) of the certificate that you attach to the load balancer.
- ④ Defines the ingress class name.
- ⑤ Defines the domain for traffic routing.
- ⑥ Defines the service for traffic routing.

#### 4.2.5.3. Creating multiple ingress resources through a single AWS Load Balancer

You can route the traffic to different services with multiple ingress resources that are part of a single domain through a single AWS Load Balancer. Each ingress resource provides different endpoints of the domain.

##### Prerequisites

- You have an access to the OpenShift CLI (**oc**).

##### Procedure

1. Create an **IngressClassParams** resource YAML file, for example, **sample-single-lb-params.yaml**, as follows:

```

apiVersion: elbv2.k8s.aws/v1beta1 ①
kind: IngressClassParams
metadata:
  name: single-lb-params ②

```

```
spec:  
group:  
  name: single-lb ③
```

- ① Defines the API group and version of the **IngressClassParams** resource.
- ② Specifies the **IngressClassParams** resource name.
- ③ Specifies the **IngressGroup** resource name. All of the **Ingress** resources of this class belong to this **IngressGroup**.

2. Create the **IngressClassParams** resource by running the following command:

```
$ oc create -f sample-single-lb-params.yaml
```

3. Create the **IngressClass** resource YAML file, for example, **sample-single-lb-class.yaml**, as follows:

```
apiVersion: networking.k8s.io/v1 ①  
kind: IngressClass  
metadata:  
  name: single-lb ②  
spec:  
  controller: ingress.k8s.aws/alb ③  
parameters:  
  apiGroup: elbv2.k8s.aws ④  
  kind: IngressClassParams ⑤  
  name: single-lb-params ⑥
```

- ① Defines the API group and version of the **IngressClass** resource.
- ② Specifies the ingress class name.
- ③ Defines the controller name. The **ingress.k8s.aws/alb** value denotes that all ingress resources of this class should be managed by the AWS Load Balancer Controller.
- ④ Defines the API group of the **IngressClassParams** resource.
- ⑤ Defines the resource type of the **IngressClassParams** resource.
- ⑥ Defines the **IngressClassParams** resource name.

4. Create the **IngressClass** resource by running the following command:

```
$ oc create -f sample-single-lb-class.yaml
```

5. Create the **AWSLoadBalancerController** resource YAML file, for example, **sample-single-lb.yaml**, as follows:

```
apiVersion: networking.olm.openshift.io/v1  
kind: AWSLoadBalancerController  
metadata:  
  name: cluster
```

```
spec:
  subnetTagging: Auto
  ingressClass: single-lb ①
```

- ① Defines the name of the **IngressClass** resource.

6. Create the **AWSLoadBalancerController** resource by running the following command:

```
$ oc create -f sample-single-lb.yaml
```

7. Create the **Ingress** resource YAML file, for example, **sample-multiple-ingress.yaml**, as follows:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-1 ①
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing ②
    alb.ingress.kubernetes.io/group.order: "1" ③
    alb.ingress.kubernetes.io/target-type: instance ④
spec:
  ingressClassName: single-lb ⑤
  rules:
  - host: example.com ⑥
    http:
      paths:
      - path: /blog ⑦
        pathType: Prefix
        backend:
          service:
            name: example-1 ⑧
            port:
              number: 80 ⑨
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-2
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "2"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
  - host: example.com
    http:
      paths:
      - path: /store
        pathType: Prefix
        backend:
          service:
            name: example-2
```

```

port:
  number: 80
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-3
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "3"
    alb.ingress.kubernetes.io/target-type: instance
spec:
  ingressClassName: single-lb
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: example-3
                port:
                  number: 80

```

- 1 Specifies the ingress name.
- 2 Indicates the load balancer to provision in the public subnet to access the internet.
- 3 Specifies the order in which the rules from the multiple ingress resources are matched when the request is received at the load balancer.
- 4 Indicates that the load balancer will target OpenShift Container Platform nodes to reach the service.
- 5 Specifies the ingress class that belongs to this ingress.
- 6 Defines a domain name used for request routing.
- 7 Defines the path that must route to the service.
- 8 Defines the service name that serves the endpoint configured in the **Ingress** resource.
- 9 Defines the port on the service that serves the endpoint.

8. Create the **Ingress** resource by running the following command:

```
$ oc create -f sample-multiple-ingress.yaml
```

#### 4.2.5.4. AWS Load Balancer Operator logs

You can view the AWS Load Balancer Operator logs by using the **oc logs** command.

##### Procedure

- View the logs of the AWS Load Balancer Operator by running the following command:

```
$ oc logs -n aws-load-balancer-operator deployment/aws-load-balancer-operator-controller-manager -c manager
```

## 4.3. EBPF MANAGER OPERATOR

### 4.3.1. About the eBPF Manager Operator



#### IMPORTANT

eBPF Manager Operator 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](#).

#### 4.3.1.1. About Extended Berkeley Packet Filter (eBPF)

eBPF extends the original Berkeley Packet Filter for advanced network traffic filtering. It acts as a virtual machine inside the Linux kernel, allowing you to run sandboxed programs in response to events such as network packets, system calls, or kernel functions.

#### 4.3.1.2. About the eBPF Manager Operator

eBPF Manager simplifies the management and deployment of eBPF programs within Kubernetes, as well as enhancing the security around using eBPF programs. It utilizes Kubernetes custom resource definitions (CRDs) to manage eBPF programs packaged as OCI container images. This approach helps to delineate deployment permissions and enhance security by restricting program types deployable by specific users.

eBPF Manager is a software stack designed to manage eBPF programs within Kubernetes. It facilitates the loading, unloading, modifying, and monitoring of eBPF programs in Kubernetes clusters. It includes a daemon, CRDs, an agent, and an operator:

##### **bpfman**

A system daemon that manages eBPF programs via a gRPC API.

##### **eBPF CRDs**

A set of CRDs like XdpProgram and TcProgram for loading eBPF programs, and a bpfman-generated CRD (BpfProgram) for representing the state of loaded programs.

##### **bpfman-agent**

Runs within a daemonset container, ensuring eBPF programs on each node are in the desired state.

##### **bpfman-operator**

Manages the lifecycle of the bpfman-agent and CRDs in the cluster using the Operator SDK.

The eBPF Manager Operator offers the following features:

- Enhances security by centralizing eBPF program loading through a controlled daemon. eBPF

Manager has the elevated privileges so the applications don't need to be. eBPF program control is regulated by standard Kubernetes Role-based access control (RBAC), which can allow or deny an application's access to the different eBPF Manager CRDs that manage eBPF program loading and unloading.

- Provides detailed visibility into active eBPF programs, improving your ability to debug issues across the system.
- Facilitates the coexistence of multiple eBPF programs from different sources using protocols like libxdp for XDP and TC programs, enhancing interoperability.
- Streamlines the deployment and lifecycle management of eBPF programs in Kubernetes. Developers can focus on program interaction rather than lifecycle management, with support for existing eBPF libraries like Cilium, libbpf, and Aya.

#### 4.3.1.3. Additional resources

- [eBPF Documentation](#)
- [bpfman](#)
- [eBPF Manager custom resource definition \(CRD\) API specification](#)

#### 4.3.1.4. Next steps

- [Installing the eBPF Manager Operator](#)

### 4.3.2. Installing the eBPF Manager Operator

As a cluster administrator, you can install the eBPF Manager Operator by using the OpenShift Container Platform CLI or the web console.



#### IMPORTANT

eBPF Manager Operator 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](#).

#### 4.3.2.1. Installing the eBPF Manager Operator using the CLI

As a cluster administrator, you can install the Operator using the CLI.

##### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

##### Procedure

1. To create the **bpfman** namespace, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: bpfman
EOF
```

2. To create an **OperatorGroup** CR, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: bpfman-operators
  namespace: bpfman
EOF
```

3. Subscribe to the eBPF Manager Operator.

- a. To create a **Subscription** CR for the eBPF Manager Operator, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: bpfman-operator
  namespace: bpfman
spec:
  name: bpfman-operator
  channel: alpha
  source: community-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. To verify that the Operator is installed, enter the following command:

```
$ oc get ip -n bpfman
```

#### Example output

NAME	CSV	APPROVAL	APPROVED
install-ppjxl	security-profiles-operator.v0.8.5	Automatic	true

5. To verify the version of the Operator, enter the following command:

```
$ oc get csv -n bpfman
```

## Example output

NAME	DISPLAY	VERSION	REPLACES
PHASE			
bpfman-operator.v0.5.0	eBPF Manager Operator	0.5.0	bpfman-
operator.v0.4.2	Succeeded		

### 4.3.2.2. Installing the eBPF Manager Operator using the web console

As a cluster administrator, you can install the eBPF Manager Operator using the web console.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

#### Procedure

1. Install the eBPF Manager Operator:
  - a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
  - b. Select **eBPF Manager Operator** from the list of available Operators, and if prompted to **Show community Operator**, click **Continue**.
  - c. Click **Install**.
  - d. On the **Install Operator** page, under **Installed Namespace**, select **Operator recommended Namespace**.
  - e. Click **Install**.
2. Verify that the eBPF Manager Operator is installed successfully:
  - a. Navigate to the **Operators → Installed Operators** page.
  - b. Ensure that **eBPF Manager Operator** is listed in the **openshift-ingress-node-firewall** project with a **Status** of **InstallSucceeded**.



#### NOTE

During installation an Operator might display a **Failed** status. If the installation later succeeds with an **InstallSucceeded** message, you can ignore the **Failed** message.

If the Operator does not have a **Status** of **InstallSucceeded**, troubleshoot using the following steps:

- Inspect the **Operator Subscriptions** and **Install Plans** tabs for any failures or errors under **Status**.
- Navigate to the **Workloads → Pods** page and check the logs for pods in the **bpfman** project.

### 4.3.2.3. Next steps

- [Deploying a containerized eBPF program](#)
- [Configuring Ingress Node Firewall Operator to use the eBPF Manager Operator](#)

### 4.3.3. Deploying an eBPF program

As a cluster administrator, you can deploy containerized eBPF applications with the eBPF Manager Operator.

For the example eBPF program deployed in this procedure, the sample manifest does the following:

First, it creates basic Kubernetes objects like **Namespace**, **ServiceAccount**, and **ClusterRoleBinding**. It also creates a **XdpProgram** object, which is a custom resource definition (CRD) that eBPF Manager provides, that loads the eBPF XDP program. Each program type has its own CRD, but they are similar in what they do. For more information, see [Loading eBPF Programs On Kubernetes](#).

Second, it creates a daemon set which runs a user space program that reads the eBPF maps that the eBPF program is populating. This eBPF map is volume mounted using a Container Storage Interface (CSI) driver. By volume mounting the eBPF map in the container in lieu of accessing it on the host, the application pod can access the eBPF maps without being privileged. For more information on how the CSI is configured, see See [Deploying an eBPF enabled application On Kubernetes](#).



#### IMPORTANT

eBPF Manager Operator 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](#).

#### 4.3.3.1. Deploying a containerized eBPF program

As a cluster administrator, you can deploy an eBPF program to nodes on your cluster. In this procedure, a sample containerized eBPF program is installed in the **go-xdp-counter** namespace.

##### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.
- You have installed the eBPF Manager Operator.

##### Procedure

1. To download the manifest, enter the following command:

```
$ curl -L https://github.com/bpfman/bpfman/releases/download/v0.5.1/go-xdp-counter-install-selinux.yaml -o go-xdp-counter-install-selinux.yaml
```

- To deploy the sample eBPF application, enter the following command:

```
$ oc create -f go-xdp-counter-install-selinux.yaml
```

#### Example output

```
namespace/go-xdp-counter created
serviceaccount/bpfman-app-go-xdp-counter created
clusterrolebinding.rbac.authorization.k8s.io/xdp-binding created
daemonset.apps/go-xdp-counter-ds created
xdpprogram.bpfman.io/go-xdp-counter-example created
selinuxprofile.security-profiles-operator.x-k8s.io/bpfman-secure created
```

- To confirm that the eBPF sample application deployed successfully, enter the following command:

```
$ oc get all -o wide -n go-xdp-counter
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE	READINESS	GATES				
pod/go-xdp-counter-ds-4m9cw	1/1	Running	0	44s	10.129.0.92	ci-1n-dcbq7d2-72292-ztrkp-master-1
	<none>	<none>				
pod/go-xdp-counter-ds-7hzww	1/1	Running	0	44s	10.130.0.86	ci-1n-dcbq7d2-72292-ztrkp-master-2
	<none>	<none>				
pod/go-xdp-counter-ds-qm9zx	1/1	Running	0	44s	10.128.0.101	ci-1n-dcbq7d2-72292-ztrkp-master-0
	<none>	<none>				

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE
NODE SELECTOR	AGE	CONTAINERS		IMAGES	
SELECTOR					
daemonset.apps/go-xdp-counter-ds	3	3	3	3	<none> 44s
go-xdp-counter	quay.io/bpfman-userspace/go-xdp-counter:v0.5.0				name=go-xdp-counter

- To confirm that the example XDP program is running, enter the following command:

```
$ oc get xdpprogram go-xdp-counter-example
```

#### Example output

NAME	BPFFUNCTIONNAME	NODESELECTOR	STATUS
go-xdp-counter-example	xdp_stats	{}	ReconcileSuccess

- To confirm that the XDP program is collecting data, enter the following command:

```
$ oc logs <pod_name> -n go-xdp-counter
```

Replace **<pod\_name>** with the name of an XDP program pod, such as **go-xdp-counter-ds-4m9cw**.

#### Example output

-

```
2024/08/13 15:20:06 15016 packets received  
2024/08/13 15:20:06 93581579 bytes received  
  
2024/08/13 15:20:09 19284 packets received  
2024/08/13 15:20:09 99638680 bytes received  
  
2024/08/13 15:20:12 23522 packets received  
2024/08/13 15:20:12 105666062 bytes received  
  
2024/08/13 15:20:15 27276 packets received  
2024/08/13 15:20:15 112028608 bytes received  
  
2024/08/13 15:20:18 29470 packets received  
2024/08/13 15:20:18 112732299 bytes received  
  
2024/08/13 15:20:21 32588 packets received  
2024/08/13 15:20:21 113813781 bytes received
```

## 4.4. EXTERNAL DNS OPERATOR

### 4.4.1. External DNS Operator release notes

The External DNS Operator deploys and manages **ExternalDNS** to provide name resolution for services and routes from the external DNS provider to OpenShift Container Platform.



#### IMPORTANT

The External DNS Operator is only supported on the **x86\_64** architecture.

These release notes track the development of the External DNS Operator in OpenShift Container Platform.

#### 4.4.1.1. External DNS Operator 1.3.0

The following advisory is available for the External DNS Operator version 1.3.0:

- [RHEA-2024:8550 Product Enhancement Advisory](#)

This update includes a rebase to the 0.14.2 version of the upstream project.

#### 4.4.1.1.1. Bug fixes

Previously, the ExternalDNS Operator could not deploy operands on HCP clusters. With this release, the Operator deploys operands in a running and ready state. ([OCPBUGS-37059](#))

Previously, the ExternalDNS Operator was not using RHEL 9 as its building or base images. With this release, RHEL9 is the base. ([OCPBUGS-41683](#))

Previously, the godoc had a broken link for Infoblox provider. With this release, the godoc is revised for accuracy. Some links are removed while some other are replaced with GitHub permalinks. ([OCPBUGS-36797](#))

#### 4.4.1.2. External DNS Operator 1.2.0

The following advisory is available for the External DNS Operator version 1.2.0:

- [RHEA-2022:5867 ExternalDNS Operator 1.2 operator/operand containers](#)

#### 4.4.1.2.1. New features

- The External DNS Operator now supports AWS shared VPC. For more information, see [Creating DNS records in a different AWS Account using a shared VPC](#) .

#### 4.4.1.2.2. Bug fixes

- The update strategy for the operand changed from **Rolling** to **Recreate**. ([OCPBUGS-3630](#))

#### 4.4.1.3. External DNS Operator 1.1.1

The following advisory is available for the External DNS Operator version 1.1.1:

- [RHEA-2024:0536 ExternalDNS Operator 1.1 operator/operand containers](#)

#### 4.4.1.4. External DNS Operator 1.1.0

This release included a rebase of the operand from the upstream project version 0.13.1. The following advisory is available for the External DNS Operator version 1.1.0:

- [RHEA-2022:9086-01 ExternalDNS Operator 1.1 operator/operand containers](#)

#### 4.4.1.4.1. Bug fixes

- Previously, the ExternalDNS Operator enforced an empty **defaultMode** value for volumes, which caused constant updates due to a conflict with the OpenShift API. Now, the **defaultMode** value is not enforced and operand deployment does not update constantly. ([OCPBUGS-2793](#))

#### 4.4.1.5. External DNS Operator 1.0.1

The following advisory is available for the External DNS Operator version 1.0.1:

- [RHEA-2024:0537 ExternalDNS Operator 1.0 operator/operand containers](#)

#### 4.4.1.6. External DNS Operator 1.0.0

The following advisory is available for the External DNS Operator version 1.0.0:

- [RHEA-2022:5867 ExternalDNS Operator 1.0 operator/operand containers](#)

#### 4.4.1.6.1. Bug fixes

- Previously, the External DNS Operator issued a warning about the violation of the restricted SCC policy during ExternalDNS operand pod deployments. This issue has been resolved. ([BZ#2086408](#))

### 4.4.2. Understanding the External DNS Operator

The External DNS Operator deploys and manages **ExternalDNS** to provide the name resolution for services and routes from the external DNS provider to OpenShift Container Platform.

#### 4.4.2.1. External DNS Operator

The External DNS Operator implements the External DNS API from the **olm.openshift.io** API group. The External DNS Operator updates services, routes, and external DNS providers.

##### Prerequisites

- You have installed the **yq** CLI tool.

##### Procedure

You can deploy the External DNS Operator on demand from the OperatorHub. Deploying the External DNS Operator creates a **Subscription** object.

1. Check the name of an install plan by running the following command:

```
$ oc -n external-dns-operator get sub external-dns-operator -o yaml | yq '.status.installplan.name'
```

##### Example output

```
install-zcvlr
```

2. Check if the status of an install plan is **Complete** by running the following command:

```
$ oc -n external-dns-operator get ip <install_plan_name> -o yaml | yq '.status.phase'
```

##### Example output

```
Complete
```

3. View the status of the **external-dns-operator** deployment by running the following command:

```
$ oc get -n external-dns-operator deployment/external-dns-operator
```

##### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
external-dns-operator	1/1	1	1	23h

#### 4.4.2.2. Viewing External DNS Operator logs

You can view External DNS Operator logs by using the **oc logs** command.

##### Procedure

1. View the logs of the External DNS Operator by running the following command:

```
$ oc logs -n external-dns-operator deployment/external-dns-operator -c external-dns-operator
```

#### 4.4.2.2.1. External DNS Operator domain name limitations

The External DNS Operator uses the TXT registry which adds the prefix for TXT records. This reduces the maximum length of the domain name for TXT records. A DNS record cannot be present without a corresponding TXT record, so the domain name of the DNS record must follow the same limit as the TXT records. For example, a DNS record of <domain\_name\_from\_source> results in a TXT record of **external-dns-<record\_type>-<domain\_name\_from\_source>**.

The domain name of the DNS records generated by the External DNS Operator has the following limitations:

Record type	Number of characters
CNAME	44
Wildcard CNAME records on AzureDNS	42
A	48
Wildcard A records on AzureDNS	46

The following error appears in the External DNS Operator logs if the generated domain name exceeds any of the domain name limitations:

```
time="2022-09-02T08:53:57Z" level=error msg="Failure in zone test.example.io. [Id: /hostedzone/Z06988883Q0H0RL6UMXXX]"
time="2022-09-02T08:53:57Z" level=error msg="InvalidChangeBatch: [FATAL problem: DomainLabelTooLong (Domain label is too long) encountered with 'external-dns-a-hello-openshiftaaaaaaaaaaa-bbbbbbbbbbb-ccccccc']\n\tstatus code: 400, request id: e54dfd5a-06c6-47b0-bcb9-a4f7c3a4e0c6"
```

#### 4.4.3. Installing the External DNS Operator

You can install the External DNS Operator on cloud providers such as AWS, Azure, and GCP.

##### 4.4.3.1. Installing the External DNS Operator with OperatorHub

You can install the External DNS Operator by using the OpenShift Container Platform OperatorHub.

##### Procedure

1. Click **Operators → OperatorHub** in the OpenShift Container Platform web console.
2. Click **External DNS Operator**. You can use the **Filter by keyword** text box or the filter list to search for External DNS Operator from the list of Operators.
3. Select the **external-dns-operator** namespace.
4. On the **External DNS Operator** page, click **Install**.
5. On the **Install Operator** page, ensure that you selected the following options:

- a. Update the channel as **stable-v1**.
- b. Installation mode as **A specific name on the cluster**
- c. Installed namespace as **external-dns-operator**. If namespace **external-dns-operator** does not exist, it gets created during the Operator installation.
- d. Select **Approval Strategy** as **Automatic** or **Manual**. Approval Strategy is set to **Automatic** by default.
- e. Click **Install**.

If you select **Automatic** updates, the Operator Lifecycle Manager (OLM) automatically upgrades the running instance of your Operator without any intervention.

If you select **Manual** updates, the OLM creates an update request. As a cluster administrator, you must then manually approve that update request to have the Operator updated to the new version.

## Verification

Verify that the External DNS Operator shows the **Status** as **Succeeded** on the **Installed Operators** dashboard.

### 4.4.3.2. Installing the External DNS Operator by using the CLI

You can install the External DNS Operator by using the CLI.

#### Prerequisites

- You are logged in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- You are logged into the OpenShift CLI (**oc**).

#### Procedure

1. Create a **Namespace** object:
  - a. Create a YAML file that defines the **Namespace** object:

#### Example `namespace.yaml` file

```
apiVersion: v1
kind: Namespace
metadata:
  name: external-dns-operator
```

- b. Create the **Namespace** object by running the following command:

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

2. Create an **OperatorGroup** object:
  - a. Create a YAML file that defines the **OperatorGroup** object:

#### Example `operatorgroup.yaml` file

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: external-dns-operator
  namespace: external-dns-operator
spec:
  upgradeStrategy: Default
  targetNamespaces:
    - external-dns-operator

```

- b. Create the **OperatorGroup** object by running the following command:

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

3. Create a **Subscription** object:

- a. Create a YAML file that defines the **Subscription** object:

#### Example subscription.yaml file

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: external-dns-operator
  namespace: external-dns-operator
spec:
  channel: stable-v1
  installPlanApproval: Automatic
  name: external-dns-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace

```

- b. Create the **Subscription** object by running the following command:

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

### Verification

1. Get the name of the install plan from the subscription by running the following command:

```
$ oc -n external-dns-operator \
get subscription external-dns-operator \
--template='{{.status.installplan.name}}{{"\n"}}'
```

2. Verify that the status of the install plan is **Complete** by running the following command:

```
$ oc -n external-dns-operator \
get ip <install_plan_name> \
--template='{{.status.phase}}{{"\n"}}'
```

3. Verify that the status of the **external-dns-operator** pod is **Running** by running the following command:

```
$ oc -n external-dns-operator get pod
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
external-dns-operator-5584585fd7-5lwqm	2/2	Running	0	11m

- Verify that the catalog source of the subscription is **redhat-operators** by running the following command:

```
$ oc -n external-dns-operator get subscription
```

#### Example output

NAME	PACKAGE	SOURCE	CHANNEL
external-dns-operator	external-dns-operator	redhat-operators	stable-v1

- Check the **external-dns-operator** version by running the following command:

```
$ oc -n external-dns-operator get csv
```

#### Example output

NAME	DISPLAY	VERSION	REPLACES	PHASE
external-dns-operator.v<1.y.z>	ExternalDNS Operator	<1.y.z>		Succeeded

### 4.4.4. External DNS Operator configuration parameters

The External DNS Operator includes the following configuration parameters.

#### 4.4.4.1. External DNS Operator configuration parameters

The External DNS Operator includes the following configuration parameters:

Parameter	Description
<b>spec</b>	<p>Enables the type of a cloud provider.</p> <pre>spec:   provider:     type: AWS <b>1</b>     aws:       credentials:         name: aws-access-key <b>2</b></pre> <p><b>1</b> Defines available options such as AWS, GCP, Azure, and Infoblox.</p> <p><b>2</b> Defines a secret name for your cloud provider.</p>

Parameter	Description
<b>zones</b>	<p>Enables you to specify DNS zones by their domains. If you do not specify zones, the <b>ExternalDNS</b> resource discovers all of the zones present in your cloud provider account.</p> <p>zones:</p> <ul style="list-style-type: none"> <li>- "myzoneid" ①</li> </ul> <p>① Specifies the name of DNS zones.</p>
<b>domains</b>	<p>Enables you to specify AWS zones by their domains. If you do not specify domains, the <b>ExternalDNS</b> resource discovers all of the zones present in your cloud provider account.</p> <p>domains:</p> <ul style="list-style-type: none"> <li>- filterType: Include ①</li> <li>matchType: Exact ②</li> <li>name: "myzonedomain1.com" ③</li> <li>- filterType: Include</li> <li>matchType: Pattern ④</li> <li>pattern: ".*\.\.otherzonedomain\\\.com" ⑤</li> </ul> <p>① Ensures that the <b>ExternalDNS</b> resource includes the domain name.</p> <p>② Instructs <b>ExternalDNS</b> that the domain matching has to be exact as opposed to regular expression match.</p> <p>③ Defines the name of the domain.</p> <p>④ Sets the <b>regex-domain-filter</b> flag in the <b>ExternalDNS</b> resource. You can limit possible domains by using a Regex filter.</p> <p>⑤ Defines the regex pattern to be used by the <b>ExternalDNS</b> resource to filter the domains of the target zones.</p>
<b>source</b>	<p>Enables you to specify the source for the DNS records, <b>Service</b> or <b>Route</b>.</p> <p>source: ①</p> <p>type: Service ②</p> <p>service:</p> <p>serviceType: ③</p> <ul style="list-style-type: none"> <li>- LoadBalancer</li> <li>- ClusterIP</li> </ul> <p>labelFilter: ④</p> <p>matchLabels:</p> <p>external-dns.mydomain.org/publish: "yes"</p>

Parameter	Description
	<p>hostnameAnnotation: "Allow" <b>5</b>  fqdnTemplate:  - "{{.Name}}.myzonedomain.com" <b>6</b></p> <p><b>1</b> Defines the settings for the source of DNS records.</p> <p><b>2</b> The <b>ExternalDNS</b> resource uses the <b>Service</b> type as the source for creating DNS records.</p> <p><b>3</b> Sets the <b>service-type-filter</b> flag in the <b>ExternalDNS</b> resource. The <b>serviceType</b> contains the following fields:</p> <ul style="list-style-type: none"> <li>• <b>default: LoadBalancer</b></li> <li>• <b>expected: ClusterIP</b></li> <li>• <b>NodePort</b></li> <li>• <b>LoadBalancer</b></li> <li>• <b>ExternalName</b></li> </ul> <p><b>4</b> Ensures that the controller considers only those resources which matches with label filter.</p> <p><b>5</b> The default value for <b>hostnameAnnotation</b> is <b>Ignore</b> which instructs <b>ExternalDNS</b> to generate DNS records using the templates specified in the field <b>fqdnTemplates</b>. When the value is <b>Allow</b> the DNS records get generated based on the value specified in the <b>external-dns.kubernetes.io/hostname</b> annotation.</p> <p><b>6</b> The External DNS Operator uses a string to generate DNS names from sources that do not define a hostname, or to add a hostname suffix when paired with the fake source.</p>
	<pre>source: type: OpenShiftRoute <b>1</b> openshiftRouteOptions:   routerName: default <b>2</b> labelFilter:   matchLabels:     external-dns.mydomain.org/publish: "yes"</pre> <p><b>1</b> Creates DNS records.</p> <p><b>2</b> If the source type is <b>OpenShiftRoute</b>, then you can pass the Ingress Controller name. The <b>ExternalDNS</b> resource uses the canonical name of the Ingress Controller as the target for CNAME records.</p>

#### 4.4.5. Creating DNS records on AWS

You can create DNS records on AWS and AWS GovCloud by using the External DNS Operator.

##### 4.4.5.1. Creating DNS records on an public hosted zone for AWS by using Red Hat External DNS Operator

You can create DNS records on a public hosted zone for AWS by using the Red Hat External DNS Operator. You can use the same instructions to create DNS records on a hosted zone for AWS GovCloud.

## Procedure

- Check the user. The user must have access to the **kube-system** namespace. If you don't have the credentials, as you can fetch the credentials from the **kube-system** namespace to use the cloud provider client:

```
$ oc whoami
```

### Example output

```
system:admin
```

- Fetch the values from aws-creds secret present in **kube-system** namespace.

```
$ export AWS_ACCESS_KEY_ID=$(oc get secrets aws-creds -n kube-system --template={{.data.aws_access_key_id}} | base64 -d)
$ export AWS_SECRET_ACCESS_KEY=$(oc get secrets aws-creds -n kube-system --template={{.data.aws_secret_access_key}} | base64 -d)
```

- Get the routes to check the domain:

```
$ oc get routes --all-namespaces | grep console
```

### Example output

```
openshift-console      console      console-openshift-
console.apps.testextdnsoperator.apacshift.support      console      https
reencrypt/Redirect    None
openshift-console      downloads     downloads-openshift-
console.apps.testextdnsoperator.apacshift.support      downloads     http
edge/Redirect         None
```

- Get the list of dns zones to find the one which corresponds to the previously found route's domain:

```
$ aws route53 list-hosted-zones | grep testextdnsoperator.apacshift.support
```

### Example output

```
HOSTEDZONES terraform /hostedzone/Z02355203TNN1XXXX1J6O
testextdnsoperator.apacshift.support. 5
```

- Create **ExternalDNS** resource for **route** source:

```
$ cat <<EOF | oc create -f -
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-aws 1
spec:
  domains:
    - filterType: Include 2
```

```

matchType: Exact ③
name: testextdnsoperator.apacshift.support ④
provider:
  type: AWS ⑤
source: ⑥
  type: OpenShiftRoute ⑦
  openshiftRouteOptions:
    routerName: default ⑧
EOF

```

- ① Defines the name of external DNS resource.
- ② By default all hosted zones are selected as potential targets. You can include a hosted zone that you need.
- ③ The matching of the target zone's domain has to be exact (as opposed to regular expression match).
- ④ Specify the exact domain of the zone you want to update. The hostname of the routes must be subdomains of the specified domain.
- ⑤ Defines the **AWS Route53** DNS provider.
- ⑥ Defines options for the source of DNS records.
- ⑦ Defines OpenShift **route** resource as the source for the DNS records which gets created in the previously specified DNS provider.
- ⑧ If the source is **OpenShiftRoute**, then you can pass the OpenShift Ingress Controller name. External DNS Operator selects the canonical hostname of that router as the target while creating CNAME record.

6. Check the records created for OCP routes using the following command:

```
$ aws route53 list-resource-record-sets --hosted-zone-id Z02355203TNN1XXXX1J6O --
query "ResourceRecordSets[?Type == 'CNAME']" | grep console
```

#### 4.4.5.2. Creating DNS records in a different AWS Account using a shared VPC

You can use the ExternalDNS Operator to create DNS records in a different AWS account using a shared Virtual Private Cloud (VPC). By using a shared VPC, an organization can connect resources from multiple projects to a common VPC network. Organizations can then use VPC sharing to use a single Route 53 instance across multiple AWS accounts.

##### Prerequisites

- You have created two Amazon AWS accounts: one with a VPC and a Route 53 private hosted zone configured (Account A), and another for installing a cluster (Account B).
- You have created an IAM Policy and IAM Role with the appropriate permissions in Account A for Account B to create DNS records in the Route 53 hosted zone of Account A.
- You have installed a cluster in Account B into the existing VPC for Account A.

- You have installed the ExternalDNS Operator in the cluster in Account B.

## Procedure

1. Get the Role ARN of the IAM Role that you created to allow Account B to access Account A's Route 53 hosted zone by running the following command:

```
$ aws --profile account-a iam get-role --role-name user-rol1 | head -1
```

### Example output

```
ROLE arn:aws:iam::1234567890123:role/user-rol1 2023-09-14T17:21:54+00:00 3600 /  
AROA3SGB2ZRKRT5NISN1J6O user-rol1
```

2. Locate the private hosted zone to use with Account A's credentials by running the following command:

```
$ aws --profile account-a route53 list-hosted-zones | grep  
testextdnsoperator.apacshift.support
```

### Example output

```
HOSTEDZONES terraform /hostedzone/Z02355203TNN1XXXX1J6O  
testextdnsoperator.apacshift.support. 5
```

3. Create the **ExternalDNS** object by running the following command:

```
$ cat <<EOF | oc create -f -  
apiVersion: externaldns.olm.openshift.io/v1beta1  
kind: ExternalDNS  
metadata:  
  name: sample-aws  
spec:  
  domains:  
    - filterType: Include  
      matchType: Exact  
      name: testextdnsoperator.apacshift.support  
  provider:  
    type: AWS  
    aws:  
      assumeRole:  
        arn: arn:aws:iam::12345678901234:role/user-rol1 ①  
  source:  
    type: OpenShiftRoute  
    openshiftRouteOptions:  
      routerName: default  
EOF
```

① Specify the Role ARN to have DNS records created in Account A.

4. Check the records created for OpenShift Container Platform (OCP) routes by using the following command:

```
$ aws --profile account-a route53 list-resource-record-sets --hosted-zone-id Z02355203TNN1XXXX1J6O --query "ResourceRecordSets[?Type == 'CNAME']" | grep console-openshift-console
```

#### 4.4.6. Creating DNS records on Azure

You can create DNS records on Azure by using the External DNS Operator.



#### IMPORTANT

Using the External DNS Operator on a Microsoft Entra Workload ID-enabled cluster or a cluster that runs in Microsoft Azure Government (MAG) regions is not supported.

##### 4.4.6.1. Creating DNS records on an Azure public DNS zone

You can create DNS records on a public DNS zone for Azure by using the External DNS Operator.

#### Prerequisites

- You must have administrator privileges.
- The **admin** user must have access to the **kube-system** namespace.

#### Procedure

1. Fetch the credentials from the **kube-system** namespace to use the cloud provider client by running the following command:

```
$ CLIENT_ID=$(oc get secrets azure-credentials -n kube-system --template={{.data.azure_client_id}} | base64 -d)
$ CLIENT_SECRET=$(oc get secrets azure-credentials -n kube-system --template={{.data.azure_client_secret}} | base64 -d)
$ RESOURCE_GROUP=$(oc get secrets azure-credentials -n kube-system --template={{.data.azure_resourcegroup}} | base64 -d)
$ SUBSCRIPTION_ID=$(oc get secrets azure-credentials -n kube-system --template={{.data.azure_subscription_id}} | base64 -d)
$ TENANT_ID=$(oc get secrets azure-credentials -n kube-system --template={{.data.azure_tenant_id}} | base64 -d)
```

2. Log in to Azure by running the following command:

```
$ az login --service-principal -u "${CLIENT_ID}" -p "${CLIENT_SECRET}" --tenant "${TENANT_ID}"
```

3. Get a list of routes by running the following command:

```
$ oc get routes --all-namespaces | grep console
```

#### Example output

openshift-console	console	console-openshift-		
console.apps.test.azure.example.com		console		https reencrypt/Redirect
None				

```
openshift-console      downloads      downloads-openshift-
console.apps.test.azure.example.com      downloads      http   edge/Redirect
None
```

4. Get a list of DNS zones by running the following command:

```
$ az network dns zone list --resource-group "${RESOURCE_GROUP}"
```

5. Create a YAML file, for example, **external-dns-sample-azure.yaml**, that defines the **ExternalIDNS** object:

#### Example **external-dns-sample-azure.yaml** file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalIDNS
metadata:
  name: sample-azure 1
spec:
  zones:
    - "/subscriptions/1234567890/resourceGroups/test-azure-xxxxx-
rg/providers/Microsoft.Network/dnszones/test.azure.example.com" 2
  provider:
    type: Azure 3
  source:
    openshiftRouteOptions: 4
      routerName: default 5
    type: OpenShiftRoute 6
```

- 1** Specifies the External DNS name.
- 2** Defines the zone ID.
- 3** Defines the provider type.
- 4** You can define options for the source of DNS records.
- 5** If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. External DNS selects the canonical hostname of that router as the target while creating CNAME record.
- 6** Defines the **route** resource as the source for the Azure DNS records.

6. Check the DNS records created for OpenShift Container Platform routes by running the following command:

```
$ az network dns record-set list -g "${RESOURCE_GROUP}" -z test.azure.example.com |
grep console
```



#### NOTE

To create records on private hosted zones on private Azure DNS, you need to specify the private zone under the **zones** field which populates the provider type to **azure-private-dns** in the **ExternalIDNS** container arguments.

#### 4.4.7. Creating DNS records on GCP

You can create DNS records on Google Cloud Platform (GCP) by using the External DNS Operator.



##### IMPORTANT

Using the External DNS Operator on a cluster with GCP Workload Identity enabled is not supported. For more information about the GCP Workload Identity, see [GCP Workload Identity](#).

##### 4.4.7.1. Creating DNS records on a public managed zone for GCP

You can create DNS records on a public managed zone for GCP by using the External DNS Operator.

##### Prerequisites

- You must have administrator privileges.

##### Procedure

1. Copy the **gcp-credentials** secret in the **encoded-gcloud.json** file by running the following command:

```
$ oc get secret gcp-credentials -n kube-system --template='{{$.v := index .data "service_account.json"}}{{$.v}}' | base64 -d - > decoded-gcloud.json
```

2. Export your Google credentials by running the following command:

```
$ export GOOGLE_CREDENTIALS=decoded-gcloud.json
```

3. Activate your account by using the following command:

```
$ gcloud auth activate-service-account <client_email as per decoded-gcloud.json> --key-file=decoded-gcloud.json
```

4. Set your project by running the following command:

```
$ gcloud config set project <project_id as per decoded-gcloud.json>
```

5. Get a list of routes by running the following command:

```
$ oc get routes --all-namespaces | grep console
```

##### Example output

openshift-console	console	console-openshift-	
console.apps.test.gcp.example.com		console	https reencrypt/Redirect
None			
openshift-console	downloads	downloads-openshift-	
console.apps.test.gcp.example.com		downloads	http edge/Redirect
None			

- Get a list of managed zones by running the following command:

```
$ gcloud dns managed-zones list | grep test.gcp.example.com
```

### Example output

```
qe-cvs4g-private-zone test.gcp.example.com
```

- Create a YAML file, for example, **external-dns-sample-gcp.yaml**, that defines the **ExternalIDNS** object:

### Example **external-dns-sample-gcp.yaml** file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalIDNS
metadata:
  name: sample-gcp 1
spec:
  domains:
    - filterType: Include 2
      matchType: Exact 3
      name: test.gcp.example.com 4
  provider:
    type: GCP 5
  source:
    openshiftRouteOptions: 6
    routerName: default 7
    type: OpenShiftRoute 8
```

- 1** Specifies the External DNS name.
- 2** By default, all hosted zones are selected as potential targets. You can include your hosted zone.
- 3** The domain of the target must match the string defined by the **name** key.
- 4** Specify the exact domain of the zone you want to update. The hostname of the routes must be subdomains of the specified domain.
- 5** Defines the provider type.
- 6** You can define options for the source of DNS records.
- 7** If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. External DNS selects the canonical hostname of that router as the target while creating CNAME record.
- 8** Defines the **route** resource as the source for GCP DNS records.

- Check the DNS records created for OpenShift Container Platform routes by running the following command:

```
$ gcloud dns record-sets list --zone=qe-cvs4g-private-zone | grep console
```

## 4.4.8. Creating DNS records on Infoblox

You can create DNS records on Infoblox by using the External DNS Operator.

### 4.4.8.1. Creating DNS records on a public DNS zone on Infoblox

You can create DNS records on a public DNS zone on Infoblox by using the External DNS Operator.

#### Prerequisites

- You have access to the OpenShift CLI (**oc**).
- You have access to the Infoblox UI.

#### Procedure

1. Create a **secret** object with Infoblox credentials by running the following command:

```
$ oc -n external-dns-operator create secret generic infoblox-credentials --from-literal=EXTERNAL_DNS_INFOBLOX_WAPI_USERNAME=<infoblox_username> --from-literal=EXTERNAL_DNS_INFOBLOX_WAPI_PASSWORD=<infoblox_password>
```

2. Get a list of routes by running the following command:

```
$ oc get routes --all-namespaces | grep console
```

#### Example Output

```
openshift-console      console      console-openshift-console.apps.test.example.com
console      https  reencrypt/Redirect  None
openshift-console      downloads    downloads-openshift-
console.apps.test.example.com          downloads    http  edge/Redirect
None
```

3. Create a YAML file, for example, **external-dns-sample-infoblox.yaml**, that defines the **ExternalDNS** object:

#### Example **external-dns-sample-infoblox.yaml** file

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-infoblox 1
spec:
  provider:
    type: Infoblox 2
    infoblox:
      credentials:
        name: infoblox-credentials
        gridHost: ${INFOBLOX_GRID_PUBLIC_IP}
        wapiPort: 443
        wapiVersion: "2.3.1"
```

```

domains:
- filterType: Include
  matchType: Exact
  name: test.example.com
source:
  type: OpenShiftRoute 3
  openshiftRouteOptions:
    routerName: default 4

```

- 1** Specifies the External DNS name.
- 2** Defines the provider type.
- 3** You can define options for the source of DNS records.
- 4** If the source type is **OpenShiftRoute**, you can pass the OpenShift Ingress Controller name. External DNS selects the canonical hostname of that router as the target while creating CNAME record.

4. Create the **ExternalDNS** resource on Infoblox by running the following command:

```
$ oc create -f external-dns-sample-infoblox.yaml
```

5. From the Infoblox UI, check the DNS records created for **console** routes:

- a. Click **Data Management** → **DNS** → **Zones**.
- b. Select the zone name.

#### 4.4.9. Configuring the cluster-wide proxy on the External DNS Operator

After configuring the cluster-wide proxy, the Operator Lifecycle Manager (OLM) triggers automatic updates to all of the deployed Operators with the new contents of the **HTTP\_PROXY**, **HTTPS\_PROXY**, and **NO\_PROXY** environment variables.

##### 4.4.9.1. Trusting the certificate authority of the cluster-wide proxy

You can configure the External DNS Operator to trust the certificate authority of the cluster-wide proxy.

##### Procedure

1. Create the config map to contain the CA bundle in the **external-dns-operator** namespace by running the following command:

```
$ oc -n external-dns-operator create configmap trusted-ca
```

2. To inject the trusted CA bundle into the config map, add the **config.openshift.io/inject-trusted-cabundle=true** label to the config map by running the following command:

```
$ oc -n external-dns-operator label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true
```

3. Update the subscription of the External DNS Operator by running the following command:

```
$ oc -n external-dns-operator patch subscription external-dns-operator --type='json' -p='[{"op": "add", "path": "/spec/config", "value":{"env":[{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}]}]'
```

## Verification

- After the deployment of the External DNS Operator is completed, verify that the trusted CA environment variable is added to the **external-dns-operator** deployment by running the following command:

```
$ oc -n external-dns-operator exec deploy/external-dns-operator -c external-dns-operator --printenv TRUSTED_CA_CONFIGMAP_NAME
```

## Example output

```
trusted-ca
```

## 4.5. METALLB OPERATOR

### 4.5.1. About MetalLB and the MetalLB Operator

As a cluster administrator, you can add the MetalLB Operator to your cluster so that when a service of type **LoadBalancer** is added to the cluster, MetalLB can add an external IP address for the service. The external IP address is added to the host network for your cluster.

#### 4.5.1.1. When to use MetalLB

Using MetalLB is valuable when you have a bare-metal cluster, or an infrastructure that is like bare metal, and you want fault-tolerant access to an application through an external IP address.

You must configure your networking infrastructure to ensure that network traffic for the external IP address is routed from clients to the host network for the cluster.

After deploying MetalLB with the MetalLB Operator, when you add a service of type **LoadBalancer**, MetalLB provides a platform-native load balancer.

When external traffic enters your OpenShift Container Platform cluster through a MetalLB **LoadBalancer** service, the return traffic to the client has the external IP address of the load balancer as the source IP.

MetalLB operating in layer2 mode provides support for failover by utilizing a mechanism similar to IP failover. However, instead of relying on the virtual router redundancy protocol (VRRP) and keepalived, MetalLB leverages a gossip-based protocol to identify instances of node failure. When a failover is detected, another node assumes the role of the leader node, and a gratuitous ARP message is dispatched to broadcast this change.

MetalLB operating in layer3 or border gateway protocol (BGP) mode delegates failure detection to the network. The BGP router or routers that the OpenShift Container Platform nodes have established a connection with will identify any node failure and terminate the routes to that node.

Using MetalLB instead of IP failover is preferable for ensuring high availability of pods and services.

### 4.5.1.2. MetalLB Operator custom resources

The MetalLB Operator monitors its own namespace for the following custom resources:

#### MetalLB

When you add a **MetalLB** custom resource to the cluster, the MetalLB Operator deploys MetalLB on the cluster. The Operator only supports a single instance of the custom resource. If the instance is deleted, the Operator removes MetalLB from the cluster.

#### IPAddressPool

MetalLB requires one or more pools of IP addresses that it can assign to a service when you add a service of type **LoadBalancer**. An **IPAddressPool** includes a list of IP addresses. The list can be a single IP address that is set using a range, such as 1.1.1.1-1.1.1.1, a range specified in CIDR notation, a range specified as a starting and ending address separated by a hyphen, or a combination of the three. An **IPAddressPool** requires a name. The documentation uses names like **doc-example**, **doc-example-reserved**, and **doc-example-ipv6**. The MetalLB **controller** assigns IP addresses from a pool of addresses in an **IPAddressPool**. **L2Advertisement** and **BGPAdvertisement** custom resources enable the advertisement of a given IP from a given pool. You can assign IP addresses from an **IPAddressPool** to services and namespaces by using the **spec.serviceAllocation** specification in the **IPAddressPool** custom resource.



#### NOTE

A single **IPAddressPool** can be referenced by a L2 advertisement and a BGP advertisement.

#### BGPPeer

The BGP peer custom resource identifies the BGP router for MetalLB to communicate with, the AS number of the router, the AS number for MetalLB, and customizations for route advertisement. MetalLB advertises the routes for service load-balancer IP addresses to one or more BGP peers.

#### BFDProfile

The BFD profile custom resource configures Bidirectional Forwarding Detection (BFD) for a BGP peer. BFD provides faster path failure detection than BGP alone provides.

#### L2Advertisement

The L2Advertisement custom resource advertises an IP coming from an **IPAddressPool** using the L2 protocol.

#### BGPAdvertisement

The BGPAdvertisement custom resource advertises an IP coming from an **IPAddressPool** using the BGP protocol.

After you add the **MetalLB** custom resource to the cluster and the Operator deploys MetalLB, the **controller** and **speaker** MetalLB software components begin running.

MetalLB validates all relevant custom resources.

### 4.5.1.3. MetalLB software components

When you install the MetalLB Operator, the **metallb-operator-controller-manager** deployment starts a pod. The pod is the implementation of the Operator. The pod monitors for changes to all the relevant resources.

When the Operator starts an instance of MetalLB, it starts a **controller** deployment and a **speaker** daemon set.



## NOTE

You can configure deployment specifications in the MetalLB custom resource to manage how **controller** and **speaker** pods deploy and run in your cluster. For more information about these deployment specifications, see the *Additional resources* section.

### controller

The Operator starts the deployment and a single pod. When you add a service of type **LoadBalancer**, Kubernetes uses the **controller** to allocate an IP address from an address pool. In case of a service failure, verify you have the following entry in your **controller** pod logs:

#### Example output

```
"event": "ipAllocated", "ip": "172.22.0.201", "msg": "IP address assigned by controller"
```

### speaker

The Operator starts a daemon set for **speaker** pods. By default, a pod is started on each node in your cluster. You can limit the pods to specific nodes by specifying a node selector in the **MetalLB** custom resource when you start MetalLB. If the **controller** allocated the IP address to the service and service is still unavailable, read the **speaker** pod logs. If the **speaker** pod is unavailable, run the **oc describe pod -n** command.

For layer 2 mode, after the **controller** allocates an IP address for the service, the **speaker** pods use an algorithm to determine which **speaker** pod on which node will announce the load balancer IP address. The algorithm involves hashing the node name and the load balancer IP address. For more information, see "MetalLB and external traffic policy". The **speaker** uses Address Resolution Protocol (ARP) to announce IPv4 addresses and Neighbor Discovery Protocol (NDP) to announce IPv6 addresses.

For Border Gateway Protocol (BGP) mode, after the **controller** allocates an IP address for the service, each **speaker** pod advertises the load balancer IP address with its BGP peers. You can configure which nodes start BGP sessions with BGP peers.

Requests for the load balancer IP address are routed to the node with the **speaker** that announces the IP address. After the node receives the packets, the service proxy routes the packets to an endpoint for the service. The endpoint can be on the same node in the optimal case, or it can be on another node. The service proxy chooses an endpoint each time a connection is established.

#### 4.5.1.4. MetalLB and external traffic policy

With layer 2 mode, one node in your cluster receives all the traffic for the service IP address. With BGP mode, a router on the host network opens a connection to one of the nodes in the cluster for a new client connection. How your cluster handles the traffic after it enters the node is affected by the external traffic policy.

### cluster

This is the default value for **spec.externalTrafficPolicy**.

With the **cluster** traffic policy, after the node receives the traffic, the service proxy distributes the traffic to all the pods in your service. This policy provides uniform traffic distribution across the pods, but it obscures the client IP address and it can appear to the application in your pods that the traffic originates from the node rather than the client.

### local

With the **local** traffic policy, after the node receives the traffic, the service proxy only sends traffic to the pods on the same node. For example, if the **speaker** pod on node A announces the external service IP, then all traffic is sent to node A. After the traffic enters node A, the service proxy only sends traffic to pods for the service that are also on node A. Pods for the service that are on additional nodes do not receive any traffic from node A. Pods for the service on additional nodes act as replicas in case failover is needed.

This policy does not affect the client IP address. Application pods can determine the client IP address from the incoming connections.



### NOTE

The following information is important when configuring the external traffic policy in BGP mode.

Although MetalLB advertises the load balancer IP address from all the eligible nodes, the number of nodes loadbalancing the service can be limited by the capacity of the router to establish equal-cost multipath (ECMP) routes. If the number of nodes advertising the IP is greater than the ECMP group limit of the router, the router will use less nodes than the ones advertising the IP.

For example, if the external traffic policy is set to **local** and the router has an ECMP group limit set to 16 and the pods implementing a LoadBalancer service are deployed on 30 nodes, this would result in pods deployed on 14 nodes not receiving any traffic. In this situation, it would be preferable to set the external traffic policy for the service to **cluster**.

#### 4.5.1.5. MetalLB concepts for layer 2 mode

In layer 2 mode, the **speaker** pod on one node announces the external IP address for a service to the host network. From a network perspective, the node appears to have multiple IP addresses assigned to a network interface.



### NOTE

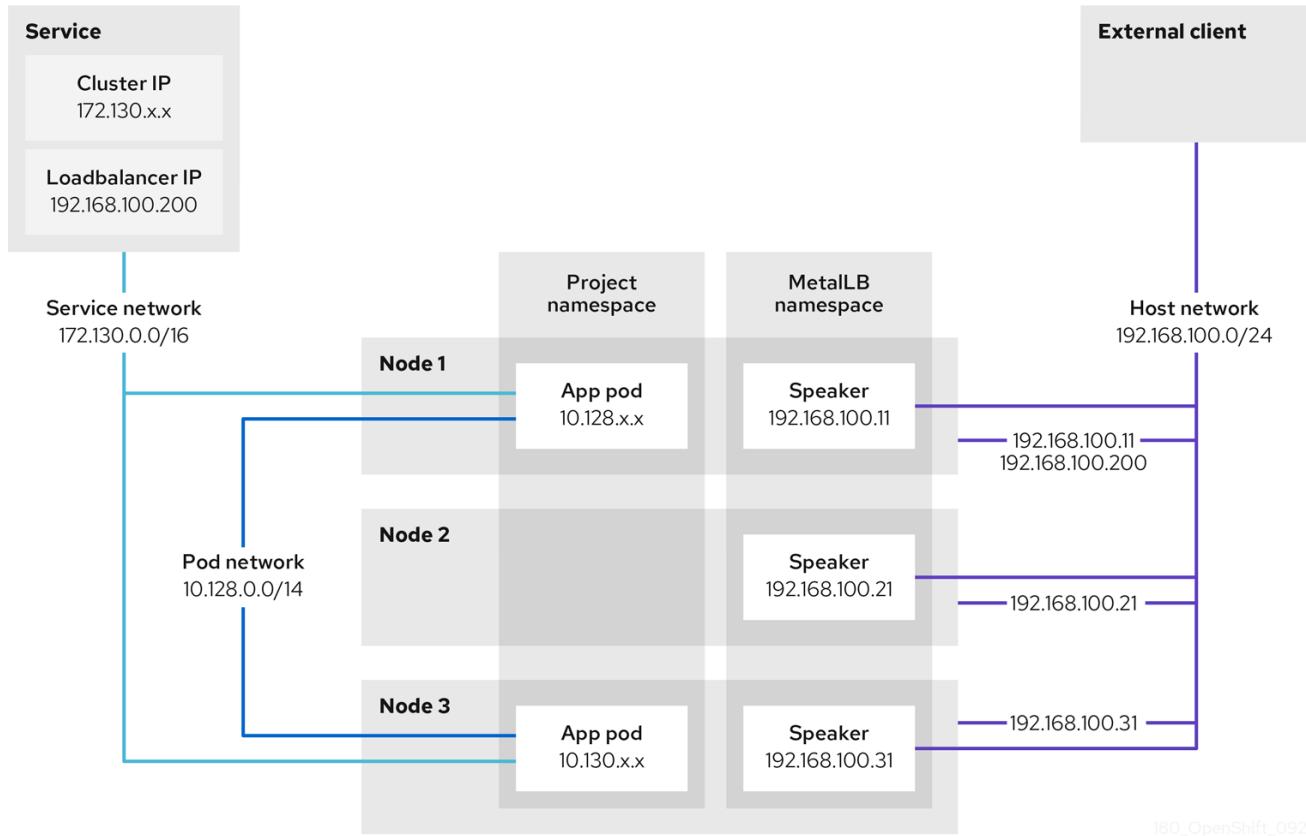
In layer 2 mode, MetalLB relies on ARP and NDP. These protocols implement local address resolution within a specific subnet. In this context, the client must be able to reach the VIP assigned by MetalLB that exists on the same subnet as the nodes announcing the service in order for MetalLB to work.

The **speaker** pod responds to ARP requests for IPv4 services and NDP requests for IPv6.

In layer 2 mode, all traffic for a service IP address is routed through one node. After traffic enters the node, the service proxy for the CNI network provider distributes the traffic to all the pods for the service.

Because all traffic for a service enters through a single node in layer 2 mode, in a strict sense, MetalLB does not implement a load balancer for layer 2. Rather, MetalLB implements a failover mechanism for layer 2 so that when a **speaker** pod becomes unavailable, a **speaker** pod on a different node can announce the service IP address.

When a node becomes unavailable, failover is automatic. The **speaker** pods on the other nodes detect that a node is unavailable and a new **speaker** pod and node take ownership of the service IP address from the failed node.



180\_OpenShift\_0921

The preceding graphic shows the following concepts related to MetalLB:

- An application is available through a service that has a cluster IP on the **172.130.0.0/16** subnet. That IP address is accessible from inside the cluster. The service also has an external IP address that MetalLB assigned to the service, **192.168.100.200**.
- Nodes 1 and 3 have a pod for the application.
- The **speaker** daemon set runs a pod on each node. The MetalLB Operator starts these pods.
- Each **speaker** pod is a host-networked pod. The IP address for the pod is identical to the IP address for the node on the host network.
- The **speaker** pod on node 1 uses ARP to announce the external IP address for the service, **192.168.100.200**. The **speaker** pod that announces the external IP address must be on the same node as an endpoint for the service and the endpoint must be in the **Ready** condition.
- Client traffic is routed to the host network and connects to the **192.168.100.200** IP address. After traffic enters the node, the service proxy sends the traffic to the application pod on the same node or another node according to the external traffic policy that you set for the service.
  - If the external traffic policy for the service is set to **cluster**, the node that advertises the **192.168.100.200** load balancer IP address is selected from the nodes where a **speaker** pod is running. Only that node can receive traffic for the service.
  - If the external traffic policy for the service is set to **local**, the node that advertises the **192.168.100.200** load balancer IP address is selected from the nodes where a **speaker** pod is running and at least an endpoint of the service. Only that node can receive traffic for the service. In the preceding graphic, either node 1 or 3 would advertise **192.168.100.200**.
- If node 1 becomes unavailable, the external IP address fails over to another node. On another

node that has an instance of the application pod and service endpoint, the **speaker** pod begins to announce the external IP address, **192.168.100.200** and the new node receives the client traffic. In the diagram, the only candidate is node 3.

#### 4.5.1.6. MetallB concepts for BGP mode

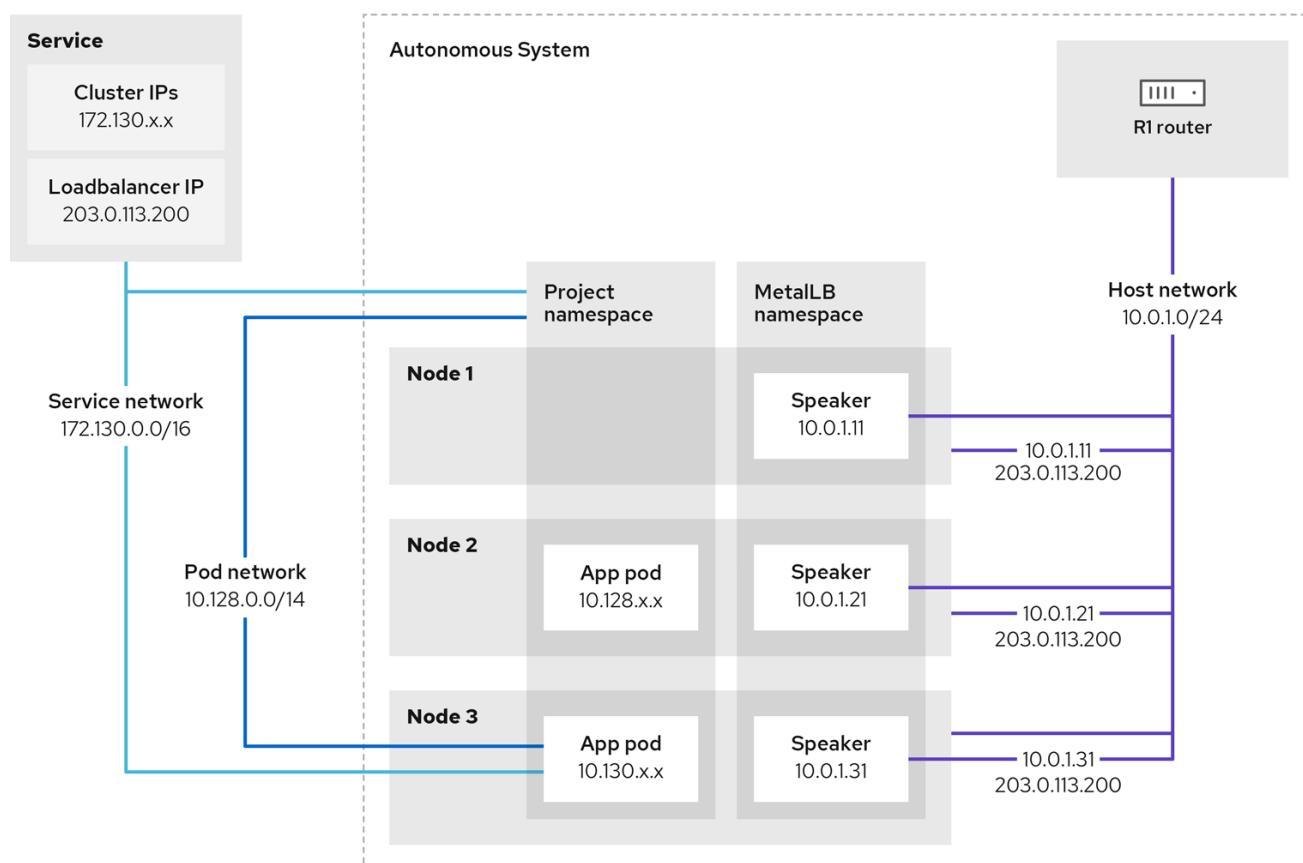
In BGP mode, by default each **speaker** pod advertises the load balancer IP address for a service to each BGP peer. It is also possible to advertise the IPs coming from a given pool to a specific set of peers by adding an optional list of BGP peers. BGP peers are commonly network routers that are configured to use the BGP protocol. When a router receives traffic for the load balancer IP address, the router picks one of the nodes with a **speaker** pod that advertised the IP address. The router sends the traffic to that node. After traffic enters the node, the service proxy for the CNI network plugin distributes the traffic to all the pods for the service.

The directly-connected router on the same layer 2 network segment as the cluster nodes can be configured as a BGP peer. If the directly-connected router is not configured as a BGP peer, you need to configure your network so that packets for load balancer IP addresses are routed between the BGP peers and the cluster nodes that run the **speaker** pods.

Each time a router receives new traffic for the load balancer IP address, it creates a new connection to a node. Each router manufacturer has an implementation-specific algorithm for choosing which node to initiate the connection with. However, the algorithms commonly are designed to distribute traffic across the available nodes for the purpose of balancing the network load.

If a node becomes unavailable, the router initiates a new connection with another node that has a **speaker** pod that advertises the load balancer IP address.

**Figure 4.1. MetallB topology diagram for BGP mode**



The preceding graphic shows the following concepts related to MetalLB:

- An application is available through a service that has an IPv4 cluster IP on the **172.130.0.0/16** subnet. That IP address is accessible from inside the cluster. The service also has an external IP address that MetalLB assigned to the service, **203.0.113.200**.
- Nodes 2 and 3 have a pod for the application.
- The **speaker** daemon set runs a pod on each node. The MetalLB Operator starts these pods. You can configure MetalLB to specify which nodes run the **speaker** pods.
- Each **speaker** pod is a host-networked pod. The IP address for the pod is identical to the IP address for the node on the host network.
- Each **speaker** pod starts a BGP session with all BGP peers and advertises the load balancer IP addresses or aggregated routes to the BGP peers. The **speaker** pods advertise that they are part of Autonomous System 65010. The diagram shows a router, R1, as a BGP peer within the same Autonomous System. However, you can configure MetalLB to start BGP sessions with peers that belong to other Autonomous Systems.
- All the nodes with a **speaker** pod that advertises the load balancer IP address can receive traffic for the service.
  - If the external traffic policy for the service is set to **cluster**, all the nodes where a speaker pod is running advertise the **203.0.113.200** load balancer IP address and all the nodes with a **speaker** pod can receive traffic for the service. The host prefix is advertised to the router peer only if the external traffic policy is set to cluster.
  - If the external traffic policy for the service is set to **local**, then all the nodes where a **speaker** pod is running and at least an endpoint of the service is running can advertise the **203.0.113.200** load balancer IP address. Only those nodes can receive traffic for the service. In the preceding graphic, nodes 2 and 3 would advertise **203.0.113.200**.
- You can configure MetalLB to control which **speaker** pods start BGP sessions with specific BGP peers by specifying a node selector when you add a BGP peer custom resource.
- Any routers, such as R1, that are configured to use BGP can be set as BGP peers.
- Client traffic is routed to one of the nodes on the host network. After traffic enters the node, the service proxy sends the traffic to the application pod on the same node or another node according to the external traffic policy that you set for the service.
- If a node becomes unavailable, the router detects the failure and initiates a new connection with another node. You can configure MetalLB to use a Bidirectional Forwarding Detection (BFD) profile for BGP peers. BFD provides faster link failure detection so that routers can initiate new connections earlier than without BFD.

#### 4.5.1.7. Limitations and restrictions

##### 4.5.1.7.1. Infrastructure considerations for MetalLB

MetalLB is primarily useful for on-premise, bare metal installations because these installations do not include a native load-balancer capability. In addition to bare metal installations, installations of OpenShift Container Platform on some infrastructures might not include a native load-balancer capability. For example, the following infrastructures can benefit from adding the MetalLB Operator:

- Bare metal
- VMware vSphere
- IBM Z® and IBM® LinuxONE
- IBM Z® and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM
- IBM Power®

#### 4.5.1.7.2. Limitations for layer 2 mode

##### 4.5.1.7.2.1. Single-node bottleneck

MetalLB routes all traffic for a service through a single node, the node can become a bottleneck and limit performance.

Layer 2 mode limits the ingress bandwidth for your service to the bandwidth of a single node. This is a fundamental limitation of using ARP and NDP to direct traffic.

##### 4.5.1.7.2.2. Slow failover performance

Failover between nodes depends on cooperation from the clients. When a failover occurs, MetalLB sends gratuitous ARP packets to notify clients that the MAC address associated with the service IP has changed.

Most client operating systems handle gratuitous ARP packets correctly and update their neighbor caches promptly. When clients update their caches quickly, failover completes within a few seconds. Clients typically fail over to a new node within 10 seconds. However, some client operating systems either do not handle gratuitous ARP packets at all or have outdated implementations that delay the cache update.

Recent versions of common operating systems such as Windows, macOS, and Linux implement layer 2 failover correctly. Issues with slow failover are not expected except for older and less common client operating systems.

To minimize the impact from a planned failover on outdated clients, keep the old node running for a few minutes after flipping leadership. The old node can continue to forward traffic for outdated clients until their caches refresh.

During an unplanned failover, the service IPs are unreachable until the outdated clients refresh their cache entries.

##### 4.5.1.7.2.3. Additional Network and MetalLB cannot use same network

Using the same VLAN for both MetalLB and an additional network interface set up on a source pod might result in a connection failure. This occurs when both the MetalLB IP and the source pod reside on the same node.

To avoid connection failures, place the MetalLB IP in a different subnet from the one where the source pod resides. This configuration ensures that traffic from the source pod will take the default gateway. Consequently, the traffic can effectively reach its destination by using the OVN overlay network, ensuring that the connection functions as intended.

#### 4.5.1.7.3. Limitations for BGP mode

#### 4.5.1.7.3.1. Node failure can break all active connections

MetalLB shares a limitation that is common to BGP-based load balancing. When a BGP session terminates, such as when a node fails or when a **speaker** pod restarts, the session termination might result in resetting all active connections. End users can experience a **Connection reset by peer** message.

The consequence of a terminated BGP session is implementation-specific for each router manufacturer. However, you can anticipate that a change in the number of **speaker** pods affects the number of BGP sessions and that active connections with BGP peers will break.

To avoid or reduce the likelihood of a service interruption, you can specify a node selector when you add a BGP peer. By limiting the number of nodes that start BGP sessions, a fault on a node that does not have a BGP session has no affect on connections to the service.

#### 4.5.1.7.3.2. Support for a single ASN and a single router ID only

When you add a BGP peer custom resource, you specify the **spec.myASN** field to identify the Autonomous System Number (ASN) that MetalLB belongs to. OpenShift Container Platform uses an implementation of BGP with MetalLB that requires MetalLB to belong to a single ASN. If you attempt to add a BGP peer and specify a different value for **spec.myASN** than an existing BGP peer custom resource, you receive an error.

Similarly, when you add a BGP peer custom resource, the **spec.routerID** field is optional. If you specify a value for this field, you must specify the same value for all other BGP peer custom resources that you add.

The limitation to support a single ASN and single router ID is a difference with the community-supported implementation of MetalLB.

#### 4.5.1.8. Additional resources

- Comparison: [Fault tolerant access to external IP addresses](#)
- [Removing IP failover](#)
- [Deployment specifications for MetalLB](#)

### 4.5.2. Installing the MetalLB Operator

As a cluster administrator, you can add the MetalLB Operator so that the Operator can manage the lifecycle for an instance of MetalLB on your cluster.

MetalLB and IP failover are incompatible. If you configured IP failover for your cluster, perform the steps to [remove IP failover](#) before you install the Operator.

#### 4.5.2.1. Installing the MetalLB Operator from the OperatorHub using the web console

As a cluster administrator, you can install the MetalLB Operator by using the OpenShift Container Platform web console.

##### Prerequisites

- Log in as a user with **cluster-admin** privileges.

## Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **OperatorHub**.
2. Type a keyword into the **Filter by keyword** box or scroll to find the Operator you want. For example, type **metallb** to find the MetalLB Operator. You can also filter options by **Infrastructure Features**. For example, select **Disconnected** if you want to see Operators that work in disconnected environments, also known as restricted network environments.
3. On the **Install Operator** page, accept the defaults and click **Install**.

## Verification

1. To confirm that the installation is successful:
  - a. Navigate to the **Operators** → **Installed Operators** page.
  - b. Check that the Operator is installed in the **openshift-operators** namespace and that its status is **Succeeded**.
2. If the Operator is not installed successfully, check the status of the Operator and review the logs:
  - a. Navigate to the **Operators** → **Installed Operators** page and inspect the **Status** column for any errors or failures.
  - b. Navigate to the **Workloads** → **Pods** page and check the logs in any pods in the **openshift-operators** project that are reporting issues.

### 4.5.2.2. Installing from OperatorHub using the CLI

Instead of using the OpenShift Container Platform web console, you can install an Operator from OperatorHub using the CLI. You can use the OpenShift CLI (**oc**) to install the MetalLB Operator.

It is recommended that when using the CLI you install the Operator in the **metallb-system** namespace.

## Prerequisites

- A cluster installed on bare-metal hardware.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

## Procedure

1. Create a namespace for the MetalLB Operator by entering the following command:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: metallb-system
EOF
```

2. Create an Operator group custom resource (CR) in the namespace:

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: metallb-operator
  namespace: metallb-system
EOF
```

3. Confirm the Operator group is installed in the namespace:

```
$ oc get operatorgroup -n metallb-system
```

### Example output

NAME	AGE
metallb-operator	14m

4. Create a **Subscription** CR:

- a. Define the **Subscription** CR and save the YAML file, for example, **metallb-sub.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metallb-operator-sub
  namespace: metallb-system
spec:
  channel: stable
  name: metallb-operator
  source: redhat-operators ①
  sourceNamespace: openshift-marketplace
```

**①** You must specify the **redhat-operators** value.

- b. To create the **Subscription** CR, run the following command:

```
$ oc create -f metallb-sub.yaml
```

5. Optional: To ensure BGP and BFD metrics appear in Prometheus, you can label the namespace as in the following command:

```
$ oc label ns metallb-system "openshift.io/cluster-monitoring=true"
```

## Verification

The verification steps assume the MetalLB Operator is installed in the **metallb-system** namespace.

1. Confirm the install plan is in the namespace:

```
$ oc get installplan -n metallb-system
```

## Example output

```
NAME      CSV          APPROVAL APPROVED
install-wzg94 metallb-operator.4.18.0-nnnnnnnnnnnn Automatic true
```



### NOTE

Installation of the Operator might take a few seconds.

2. To verify that the Operator is installed, enter the following command:

```
$ oc get clusterserviceversion -n metallb-system \
-o custom-columns=Name:.metadata.name,Phase:.status.phase
```

## Example output

```
Name          Phase
metallb-operator.4.18.0-nnnnnnnnnnnn Succeeded
```

### 4.5.2.3. Starting MetalLB on your cluster

After you install the Operator, you need to configure a single instance of a MetalLB custom resource. After you configure the custom resource, the Operator starts MetalLB on your cluster.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the MetalLB Operator.

#### Procedure

This procedure assumes the MetalLB Operator is installed in the **metallb-system** namespace. If you installed using the web console substitute **openshift-operators** for the namespace.

1. Create a single instance of a MetalLB custom resource:

```
$ cat << EOF | oc apply -f -
apiVersion: metallb.io/v1beta1
kind: MetallB
metadata:
  name: metallb
  namespace: metallb-system
EOF
```

#### Verification

Confirm that the deployment for the MetalLB controller and the daemon set for the MetalLB speaker are running.

1. Verify that the deployment for the controller is running:



```
$ oc get deployment -n metallb-system controller
```

### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
controller	1/1	1	1	11m

- Verify that the daemon set for the speaker is running:

```
$ oc get daemonset -n metallb-system speaker
```

### Example output

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE
SELECTOR						AGE
speaker	6	6	6	6	6	kubernetes.io/os=linux 18m

The example output indicates 6 speaker pods. The number of speaker pods in your cluster might differ from the example output. Make sure the output indicates one pod for each node in your cluster.

#### 4.5.2.4. Deployment specifications for MetalLB

When you start an instance of MetalLB using the **MetalLB** custom resource, you can configure deployment specifications in the **MetalLB** custom resource to manage how the **controller** or **speaker** pods deploy and run in your cluster. Use these deployment specifications to manage the following tasks:

- Select nodes for MetalLB pod deployment.
- Manage scheduling by using pod priority and pod affinity.
- Assign CPU limits for MetalLB pods.
- Assign a container RuntimeClass for MetalLB pods.
- Assign metadata for MetalLB pods.

##### 4.5.2.4.1. Limit speaker pods to specific nodes

By default, when you start MetalLB with the MetalLB Operator, the Operator starts an instance of a **speaker** pod on each node in the cluster. Only the nodes with a **speaker** pod can advertise a load balancer IP address. You can configure the **MetalLB** custom resource with a node selector to specify which nodes run the **speaker** pods.

The most common reason to limit the **speaker** pods to specific nodes is to ensure that only nodes with network interfaces on specific networks advertise load balancer IP addresses. Only the nodes with a running **speaker** pod are advertised as destinations of the load balancer IP address.

If you limit the **speaker** pods to specific nodes and specify **local** for the external traffic policy of a service, then you must ensure that the application pods for the service are deployed to the same nodes.

#### Example configuration to limit speaker pods to worker nodes

```
apiVersion: metallb.io/v1beta1
```

```

kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  nodeSelector: ①
    node-role.kubernetes.io/worker: ""
  speakerTolerations: ②
    - key: "Example"
      operator: "Exists"
      effect: "NoExecute"

```

- ① The example configuration specifies to assign the speaker pods to worker nodes, but you can specify labels that you assigned to nodes or any valid node selector.
- ② In this example configuration, the pod that this toleration is attached to tolerates any taint that matches the **key** value and **effect** value using the **operator**.

After you apply a manifest with the **spec.nodeSelector** field, you can check the number of pods that the Operator deployed with the **oc get daemonset -n metallb-system speaker** command. Similarly, you can display the nodes that match your labels with a command like **oc get nodes -l node-role.kubernetes.io/worker=**.

You can optionally allow the node to control which speaker pods should, or should not, be scheduled on them by using affinity rules. You can also limit these pods by applying a list of tolerations. For more information about affinity rules, taints, and tolerations, see the additional resources.

#### 4.5.2.4.2. Configuring pod priority and pod affinity in a MetalLB deployment

You can optionally assign pod priority and pod affinity rules to **controller** and **speaker** pods by configuring the **MetalLB** custom resource. The pod priority indicates the relative importance of a pod on a node and schedules the pod based on this priority. Set a high priority on your **controller** or **speaker** pod to ensure scheduling priority over other pods on the node.

Pod affinity manages relationships among pods. Assign pod affinity to the **controller** or **speaker** pods to control on what node the scheduler places the pod in the context of pod relationships. For example, you can use pod affinity rules to ensure that certain pods are located on the same node or nodes, which can help improve network communication and reduce latency between those components.

#### Prerequisites

- You are logged in as a user with **cluster-admin** privileges.
- You have installed the MetalLB Operator.
- You have started the MetalLB Operator on your cluster.

#### Procedure

1. Create a **PriorityClass** custom resource, such as **myPriorityClass.yaml**, to configure the priority level. This example defines a **PriorityClass** named **high-priority** with a value of **1000000**. Pods that are assigned this priority class are considered higher priority during scheduling compared to pods with lower priority classes:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
  value: 1000000
```

2. Apply the **PriorityClass** custom resource configuration:

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

3. Create a **MetalLB** custom resource, such as **MetalLBPodConfig.yaml**, to specify the **priorityClassName** and **podAffinity** values:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    priorityClassName: high-priority ①
    affinity:
      podAffinity: ②
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: metallb
            topologyKey: kubernetes.io/hostname
        speakerConfig:
          priorityClassName: high-priority
          affinity:
            podAffinity:
              requiredDuringSchedulingIgnoredDuringExecution:
              - labelSelector:
                  matchLabels:
                    app: metallb
                  topologyKey: kubernetes.io/hostname
```

- ① Specifies the priority class for the MetalLB controller pods. In this case, it is set to **high-priority**.
- ② Specifies that you are configuring pod affinity rules. These rules dictate how pods are scheduled in relation to other pods or nodes. This configuration instructs the scheduler to schedule pods that have the label **app: metallb** onto nodes that share the same hostname. This helps to co-locate MetalLB-related pods on the same nodes, potentially optimizing network communication, latency, and resource usage between these pods.

4. Apply the **MetalLB** custom resource configuration:

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

## Verification

- To view the priority class that you assigned to pods in the **metallb-system** namespace, run the following command:

```
$ oc get pods -n metallb-system -o custom-columns=NAME:.metadata.name,PRIORITY:.spec.priorityClassName
```

### Example output

NAME	PRIORITY
controller-584f5c8cd8-5zbvg	high-priority
metallb-operator-controller-manager-9c8d9985-szkqg	<none>
metallb-operator-webhook-server-c895594d4-shjgx	<none>
speaker-dddf7	high-priority

- To verify that the scheduler placed pods according to pod affinity rules, view the metadata for the pod's node or nodes by running the following command:

```
$ oc get pod -o=custom-columns=NODE:.spec.nodeName,NAME:.metadata.name -n metallb-system
```

#### 4.5.2.4.3. Configuring pod CPU limits in a MetalLB deployment

You can optionally assign pod CPU limits to **controller** and **speaker** pods by configuring the **MetalLB** custom resource. Defining CPU limits for the **controller** or **speaker** pods helps you to manage compute resources on the node. This ensures all pods on the node have the necessary compute resources to manage workloads and cluster housekeeping.

#### Prerequisites

- You are logged in as a user with **cluster-admin** privileges.
- You have installed the MetalLB Operator.

#### Procedure

- Create a **MetalLB** custom resource file, such as **CPULimits.yaml**, to specify the **cpu** value for the **controller** and **speaker** pods:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    resources:
      limits:
        cpu: "200m"
  speakerConfig:
    resources:
      limits:
        cpu: "300m"
```

2. Apply the **MetalLB** custom resource configuration:

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

## Verification

- To view compute resources for a pod, run the following command, replacing `<pod_name>` with your target pod:

```
$ oc describe pod <pod_name>
```

### 4.5.2.5. Additional resources

- Placing pods on specific nodes using node selectors
- Controlling pod placement using node taints
- Understanding pod priority
- Understanding pod affinity

### 4.5.2.6. Next steps

- Configuring MetalLB address pools

## 4.5.3. Upgrading the MetalLB Operator

A **Subscription** custom resource (CR) that subscribes the namespace to **metallb-system** by default, automatically sets the **installPlanApproval** parameter to **Automatic**. This means that when Red Hat-provided Operator catalogs include a newer version of the MetalLB Operator, the MetalLB Operator is automatically upgraded.

If you need to manually control upgrading the MetalLB Operator, set the **installPlanApproval** parameter to **Manual**.

### 4.5.3.1. Manually upgrading the MetalLB Operator

To manually control upgrading the MetalLB Operator, you must edit the **Subscription** custom resource (CR) that subscribes the namespace to **metallb-system**. A **Subscription** CR is created as part of the Operator installation and the CR has the **installPlanApproval** parameter set to **Automatic** by default.

#### Prerequisites

- You updated your cluster to the latest z-stream release.
- You used OperatorHub to install the MetalLB Operator.
- Access the cluster as a user with the **cluster-admin** role.

#### Procedure

- Get the YAML definition of the **metallb-operator** subscription in the **metallb-system** namespace by entering the following command:

```
$ oc -n metallb-system get subscription metallb-operator -o yaml
```

2. Edit the **Subscription** CR by setting the **installPlanApproval** parameter to **Manual**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metallb-operator
  namespace: metallb-system
# ...
spec:
  channel: stable
  installPlanApproval: Manual
  name: metallb-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
# ...
```

3. Find the latest OpenShift Container Platform 4.18 version of the MetalLB Operator by entering the following command:

```
$ oc -n metallb-system get csv
```

#### Example output

NAME	DISPLAY	VERSION	REPLACES	PHASE
metallb-operator.v4.18.0	MetalLB Operator	4.18.0		Succeeded

4. Check the install plan that exists in the namespace by entering the following command.

```
$ oc -n metallb-system get installplan
```

#### Example output that shows install-tsz2g as a manual install plan

NAME	CSV	APPROVAL	APPROVED
install-shpmd	metallb-operator.v4.18.0-202502261233	Automatic	true
install-tsz2g	metallb-operator.v4.18.0-202503102139	Manual	false

5. Edit the install plan that exists in the namespace by entering the following command. Ensure that you replace **<name\_of\_installplan>** with the name of the install plan, such as **install-tsz2g**.

```
$ oc edit installplan <name_of_installplan> -n metallb-system
```

- a. With the install plan open in your editor, set the **spec.approval** parameter to **Manual** and set the **spec.approved** parameter to **true**.



#### NOTE

After you edit the install plan, the upgrade operation starts. If you enter the **oc -n metallb-system get csv** command during the upgrade operation, the output might show the **Replacing** or the **Pending** status.

## Verification

- Verify the upgrade was successful by entering the following command:

```
$ oc -n metallb-system get csv
```

### Example output

NAME	DISPLAY	VERSION	REPLACE
PHASE			
metallb-operator.v<latest>.0-202503102139	MetalLB Operator	4.18.0-202503102139	
metallb-operator.v4.18.0-202502261233	Succeeded		

### 4.5.3.2. Additional resources

- [Introduction to OpenShift updates](#)
- [Installing the MetalLB Operator](#)

## 4.6. CLUSTER NETWORK OPERATOR IN OPENSHIFT CONTAINER PLATFORM

You can use the Cluster Network Operator (CNO) to deploy and manage cluster network components on an OpenShift Container Platform cluster, including the Container Network Interface (CNI) network plugin selected for the cluster during installation.

### 4.6.1. Cluster Network Operator

The Cluster Network Operator implements the **network** API from the **operator.openshift.io** API group. The Operator deploys the OVN-Kubernetes network plugin, or the network provider plugin that you selected during cluster installation, by using a daemon set.

#### Procedure

The Cluster Network Operator is deployed during installation as a Kubernetes **Deployment**.

- Run the following command to view the Deployment status:

```
$ oc get -n openshift-network-operator deployment/network-operator
```

### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
network-operator	1/1	1	1	56m

- Run the following command to view the state of the Cluster Network Operator:

```
$ oc get clusteroperator/network
```

### Example output

```
NAME      VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
network  4.16.1   True       False       False      50m
```

The following fields provide information about the status of the operator: **AVAILABLE**, **PROGRESSING**, and **DEGRADED**. The **AVAILABLE** field is **True** when the Cluster Network Operator reports an available status condition.

#### 4.6.2. Viewing the cluster network configuration

Every new OpenShift Container Platform installation has a **network.config** object named **cluster**.

##### Procedure

- Use the **oc describe** command to view the cluster network configuration:

```
$ oc describe network.config/cluster
```

##### Example output

```
Name:      cluster
Namespace:
Labels:    <none>
Annotations: <none>
API Version: config.openshift.io/v1
Kind:      Network
Metadata:
  Creation Timestamp: 2024-08-08T11:25:56Z
  Generation: 3
  Resource Version: 29821
  UID:        808dd2be-5077-4ff7-b6bb-21b7110126c7
Spec: ①
  Cluster Network:
    Cidr:      10.128.0.0/14
    Host Prefix: 23
  External IP:
    Policy:
  Network Diagnostics:
    Mode:
    Source Placement:
    Target Placement:
  Network Type: OVNKubernetes
  Service Network:
    172.30.0.0/16
Status: ②
  Cluster Network:
    Cidr:      10.128.0.0/14
    Host Prefix: 23
  Cluster Network MTU: 1360
  Conditions:
    Last Transition Time: 2024-08-08T11:51:50Z
    Message:
    Observed Generation: 0
    Reason:      AsExpected
    Status:      True
```

```
Type: NetworkDiagnosticsAvailable
Network Type: OVNKubernetes
Service Network:
  172.30.0.0/16
Events: <none>
```

- 1 The **Spec** field displays the configured state of the cluster network.
- 2 The **Status** field displays the current state of the cluster network configuration.

### 4.6.3. Viewing Cluster Network Operator status

You can inspect the status and view the details of the Cluster Network Operator using the **oc describe** command.

#### Procedure

- Run the following command to view the status of the Cluster Network Operator:

```
$ oc describe clusteroperators/network
```

### 4.6.4. Enabling IP forwarding globally

From OpenShift Container Platform 4.14 onward, global IP address forwarding is disabled on OVN-Kubernetes based cluster deployments to prevent undesirable effects for cluster administrators with nodes acting as routers. However, in some cases where an administrator expects traffic to be forwarded a new configuration parameter **ipForwarding** is available to allow forwarding of all IP traffic.

To re-enable IP forwarding for all traffic on OVN-Kubernetes managed interfaces set the **gatewayConfig.ipForwarding** specification in the Cluster Network Operator to **Global** following this procedure:

#### Procedure

- 1 Backup the existing network configuration by running the following command:

```
$ oc get network.operator cluster -o yaml > network-config-backup.yaml
```

- 2 Run the following command to modify the existing network configuration:

```
$ oc edit network.operator cluster
```

- a. Add or update the following block under **spec** as illustrated in the following example:

```
spec:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  serviceNetwork:
    - 172.30.0.0/16
  networkType: OVNKubernetes
  clusterNetworkMTU: 8900
  defaultNetwork:
```

```
ovnKubernetesConfig:
  gatewayConfig:
    ipForwarding: Global
```

- b. Save and close the file.
3. After applying the changes, the OpenShift Cluster Network Operator (CNO) applies the update across the cluster. You can monitor the progress by using the following command:

```
$ oc get clusteroperators network
```

The status should eventually report as **Available**, **Progressing=False**, and **Degraded=False**.

4. Alternatively, you can enable IP forwarding globally by running the following command:

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"gatewayConfig":{"ipForwarding": "Global"}}}}}' --type=merge
```



#### NOTE

The other valid option for this parameter is **Restricted** in case you want to revert this change. **Restricted** is the default and with that setting global IP address forwarding is disabled.

### 4.6.5. Viewing Cluster Network Operator logs

You can view Cluster Network Operator logs by using the **oc logs** command.

#### Procedure

- Run the following command to view the logs of the Cluster Network Operator:

```
$ oc logs --namespace=openshift-network-operator deployment/network-operator
```

### 4.6.6. Cluster Network Operator configuration

The configuration for the cluster network is specified as part of the Cluster Network Operator (CNO) configuration and stored in a custom resource (CR) object that is named **cluster**. The CR specifies the fields for the **Network** API in the **operator.openshift.io** API group.

The CNO configuration inherits the following fields during cluster installation from the **Network** API in the **Network.config.openshift.io** API group:

#### **clusterNetwork**

IP address pools from which pod IP addresses are allocated.

#### **serviceNetwork**

IP address pool for services.

#### **defaultNetwork.type**

Cluster network plugin. **OVNKubernetes** is the only supported plugin during installation.

**NOTE**

After cluster installation, you can only modify the **clusterNetwork** IP address range.

You can specify the cluster network plugin configuration for your cluster by setting the fields for the **defaultNetwork** object in the CNO object named **cluster**.

#### 4.6.6.1. Cluster Network Operator configuration object

The fields for the Cluster Network Operator (CNO) are described in the following table:

**Table 4.1. Cluster Network Operator configuration object**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	The name of the CNO object. This name is always <b>cluster</b> .
<b>spec.clusterNetwork</b>	<b>array</b>	A list specifying the blocks of IP addresses from which pod IP addresses are allocated and the subnet prefix length assigned to each individual node in the cluster. For example:  <pre>spec:   clusterNetwork:     - cidr: 10.128.0.0/19       hostPrefix: 23     - cidr: 10.128.32.0/19       hostPrefix: 23</pre>
<b>spec.serviceNetwork</b>	<b>array</b>	A block of IP addresses for services. The OVN-Kubernetes network plugin supports only a single IP address block for the service network. For example:  <pre>spec:   serviceNetwork:     - 172.30.0.0/14</pre> This value is ready-only and inherited from the <b>Network.config.openshift.io</b> object named <b>cluster</b> during cluster installation.
<b>spec.defaultNetwork</b>	<b>object</b>	Configures the network plugin for the cluster network.
<b>spec.kubeProxyConfig</b>	<b>object</b>	The fields for this object specify the kube-proxy configuration. If you are using the OVN-Kubernetes cluster network plugin, the kube-proxy configuration has no effect.



## IMPORTANT

For a cluster that needs to deploy objects across multiple networks, ensure that you specify the same value for the **clusterNetwork.hostPrefix** parameter for each network type that is defined in the **install-config.yaml** file. Setting a different value for each **clusterNetwork.hostPrefix** parameter can impact the OVN-Kubernetes network plugin, where the plugin cannot effectively route object traffic among different nodes.

### defaultNetwork object configuration

The values for the **defaultNetwork** object are defined in the following table:

**Table 4.2. defaultNetwork object**

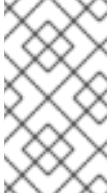
Field	Type	Description
<b>type</b>	<b>string</b>	<p><b>OVNKubernetes.</b> The Red Hat OpenShift Networking network plugin is selected during installation. This value cannot be changed after cluster installation.</p> <p> <b>NOTE</b></p> <p>OpenShift Container Platform uses the OVN-Kubernetes network plugin by default.</p>
<b>ovnKubernetesConfig</b>	<b>object</b>	This object is only valid for the OVN-Kubernetes network plugin.

### Configuration for the OVN-Kubernetes network plugin

The following table describes the configuration fields for the OVN-Kubernetes network plugin:

**Table 4.3. ovnKubernetesConfig object**

Field	Type	Description
<b>mtu</b>	<b>integer</b>	The maximum transmission unit (MTU) for the Geneve (Generic Network Virtualization Encapsulation) overlay network. This value is normally configured automatically.
<b>genevePort</b>	<b>integer</b>	The UDP port for the Geneve overlay network.
<b>ipsecConfig</b>	<b>object</b>	An object describing the IPsec mode for the cluster.
<b>ipv4</b>	<b>object</b>	Specifies a configuration object for IPv4 settings.
<b>ipv6</b>	<b>object</b>	Specifies a configuration object for IPv6 settings.
<b>policyAuditConfig</b>	<b>object</b>	Specify a configuration object for customizing network policy audit logging. If unset, the defaults audit log settings are used.

Field	Type	Description
<b>gatewayConfig</b>	<b>object</b>	<p>Optional: Specify a configuration object for customizing how egress traffic is sent to the node gateway. Valid values are <b>Shared</b> and <b>Local</b>. The default value is <b>Shared</b>. In the default setting, the Open vSwitch (OVS) outputs traffic directly to the node IP interface. In the <b>Local</b> setting, it traverses the host network; consequently, it gets applied to the routing table of the host.</p>  <p><b>NOTE</b></p> <p>While migrating egress traffic, you can expect some disruption to workloads and service traffic until the Cluster Network Operator (CNO) successfully rolls out the changes.</p>

**Table 4.4. ovnKubernetesConfig.ipv4 object**

Field	Type	Description
<b>internalTransitSwitchSubnet</b>	string	<p>If your existing network infrastructure overlaps with the <b>100.88.0.0/16</b> IPv4 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. The subnet for the distributed transit switch that enables east-west traffic. This subnet cannot overlap with any other subnets used by OVN-Kubernetes or on the host itself. It must be large enough to accommodate one IP address per node in your cluster.</p> <p>The default value is <b>100.88.0.0/16</b>.</p>
<b>internalJoinSubnet</b>	string	<p>If your existing network infrastructure overlaps with the <b>100.64.0.0/16</b> IPv4 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster. For example, if the <b>clusterNetwork.cidr</b> value is <b>10.128.0.0/14</b> and the <b>clusterNetwork.hostPrefix</b> value is <b>/23</b>, then the maximum number of nodes is <b>2^(23-14)=512</b>.</p> <p>The default value is <b>100.64.0.0/16</b>.</p>

**Table 4.5. ovnKubernetesConfig.ipv6 object**

Field	Type	Description
-------	------	-------------

Field	Type	Description
<b>internalTransitSwitchSubnet</b>	string	<p>If your existing network infrastructure overlaps with the <b>fd97::/64</b> IPv6 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. The subnet for the distributed transit switch that enables east-west traffic. This subnet cannot overlap with any other subnets used by OVN-Kubernetes or on the host itself. It must be large enough to accommodate one IP address per node in your cluster.</p> <p>The default value is <b>fd97::/64</b>.</p>
<b>internalJoinSubnet</b>	string	<p>If your existing network infrastructure overlaps with the <b>fd98::/64</b> IPv6 subnet, you can specify a different IP address range for internal use by OVN-Kubernetes. You must ensure that the IP address range does not overlap with any other subnet used by your OpenShift Container Platform installation. The IP address range must be larger than the maximum number of nodes that can be added to the cluster.</p> <p>The default value is <b>fd98::/64</b>.</p>

Table 4.6. `policyAuditConfig` object

Field	Type	Description
<b>rateLimit</b>	integer	The maximum number of messages to generate every second per node. The default value is <b>20</b> messages per second.
<b>maxFileSize</b>	integer	The maximum size for the audit log in bytes. The default value is <b>50000000</b> or 50 MB.
<b>maxLogFiles</b>	integer	The maximum number of log files that are retained.
<b>destination</b>	string	<p>One of the following additional audit log targets:</p> <p><b>libc</b> The libc <b>syslog()</b> function of the journald process on the host.</p> <p><b>udp:&lt;host&gt;:&lt;port&gt;</b> A syslog server. Replace <b>&lt;host&gt;:&lt;port&gt;</b> with the host and port of the syslog server.</p> <p><b>unix:&lt;file&gt;</b> A Unix Domain Socket file specified by <b>&lt;file&gt;</b>.</p> <p><b>null</b> Do not send the audit logs to any additional target.</p>
<b>syslogFacility</b>	string	The syslog facility, such as <b>kern</b> , as defined by RFC5424. The default value is <b>local0</b> .

**Table 4.7. gatewayConfig object**

Field	Type	Description
<b>routingViaHost</b>	<b>boolean</b>	<p>Set this field to <b>true</b> to send egress traffic from pods to the host networking stack. For highly-specialized installations and applications that rely on manually configured routes in the kernel routing table, you might want to route egress traffic to the host networking stack. By default, egress traffic is processed in OVN to exit the cluster and is not affected by specialized routes in the kernel routing table. The default value is <b>false</b>.</p> <p>This field has an interaction with the Open vSwitch hardware offloading feature. If you set this field to <b>true</b>, you do not receive the performance benefits of the offloading because egress traffic is processed by the host networking stack.</p>
<b>ipForwarding</b>	<b>object</b>	<p>You can control IP forwarding for all traffic on OVN-Kubernetes managed interfaces by using the <b>ipForwarding</b> specification in the <b>Network</b> resource. Specify <b>Restricted</b> to only allow IP forwarding for Kubernetes related traffic. Specify <b>Global</b> to allow forwarding of all IP traffic. For new installations, the default is <b>Restricted</b>. For updates to OpenShift Container Platform 4.14 or later, the default is <b>Global</b>.</p> <div style="display: flex; align-items: center;">  <span style="margin-left: 10px;"><b>NOTE</b></span> </div> <p>The default value of <b>Restricted</b> sets the IP forwarding to drop.</p>
<b>ipv4</b>	<b>object</b>	Optional: Specify an object to configure the internal OVN-Kubernetes masquerade address for host to service traffic for IPv4 addresses.
<b>ipv6</b>	<b>object</b>	Optional: Specify an object to configure the internal OVN-Kubernetes masquerade address for host to service traffic for IPv6 addresses.

**Table 4.8. gatewayConfig.ipv4 object**

Field	Type	Description
-------	------	-------------

Field	Type	Description
<b>internalMasqueradeSubnet</b>	<b>string</b>	<p>The masquerade IPv4 addresses that are used internally to enable host to service traffic. The host is configured with these IP addresses as well as the shared gateway bridge interface. The default value is <b>169.254.169.0/29</b>.</p>  <p><b>IMPORTANT</b></p> <p>For OpenShift Container Platform 4.17 and later versions, clusters use <b>169.254.0.0/17</b> as the default masquerade subnet. For upgraded clusters, there is no change to the default masquerade subnet.</p>

Table 4.9. `gatewayConfig.ipv6` object

Field	Type	Description
<b>internalMasqueradeSubnet</b>	<b>string</b>	<p>The masquerade IPv6 addresses that are used internally to enable host to service traffic. The host is configured with these IP addresses as well as the shared gateway bridge interface. The default value is <b>fd69::/125</b>.</p>  <p><b>IMPORTANT</b></p> <p>For OpenShift Container Platform 4.17 and later versions, clusters use <b>fd69::/112</b> as the default masquerade subnet. For upgraded clusters, there is no change to the default masquerade subnet.</p>

Table 4.10. `ipsecConfig` object

Field	Type	Description
<b>mode</b>	<b>string</b>	<p>Specifies the behavior of the IPsec implementation. Must be one of the following values:</p> <ul style="list-style-type: none"> <li>● <b>Disabled</b>: IPsec is not enabled on cluster nodes.</li> <li>● <b>External</b>: IPsec is enabled for network traffic with external hosts.</li> <li>● <b>Full</b>: IPsec is enabled for pod traffic and network traffic with external hosts.</li> </ul>

Field	Type	Description

**NOTE**

You can only change the configuration for your cluster network plugin during cluster installation, except for the **gatewayConfig** field that can be changed at runtime as a postinstallation activity.

### Example OVN-Kubernetes configuration with IPSec enabled

```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
    mtu: 1400
    genevePort: 6081
    ipsecConfig:
      mode: Full
```

#### 4.6.6.2. Cluster Network Operator example configuration

A complete CNO configuration is specified in the following example:

### Example Cluster Network Operator object

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  networkType: OVNKubernetes
  clusterNetworkMTU: 8900
```

#### 4.6.7. Additional resources

- [Network API in the `operator.openshift.io` API group](#)

- Expanding the cluster network IP address range
- How to configure OVN to use kernel routing table

## 4.7. DNS OPERATOR IN OPENSHIFT CONTAINER PLATFORM

In OpenShift Container Platform, the DNS Operator deploys and manages a CoreDNS instance to provide a name resolution service to pods inside the cluster, enables DNS-based Kubernetes Service discovery, and resolves internal **cluster.local** names.

### 4.7.1. Checking the status of the DNS Operator

The DNS Operator implements the **dns** API from the **operator.openshift.io** API group. The Operator deploys CoreDNS using a daemon set, creates a service for the daemon set, and configures the kubelet to instruct pods to use the CoreDNS service IP address for name resolution.

#### Procedure

The DNS Operator is deployed during installation with a **Deployment** object.

1. Use the **oc get** command to view the deployment status:

```
$ oc get -n openshift-dns-operator deployment/dns-operator
```

#### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
dns-operator	1/1	1	1	23h

2. Use the **oc get** command to view the state of the DNS Operator:

```
$ oc get clusteroperator/dns
```

#### Example output

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE	MESSAGE
dns	4.1.15-0.11	True	False	False	92m	

**AVAILABLE**, **PROGRESSING**, and **DEGRADED** provide information about the status of the Operator. **AVAILABLE** is **True** when at least 1 pod from the CoreDNS daemon set reports an **Available** status condition, and the DNS service has a cluster IP address.

### 4.7.2. View the default DNS

Every new OpenShift Container Platform installation has a **dns.operator** named **default**.

#### Procedure

1. Use the **oc describe** command to view the default **dns**:

```
$ oc describe dns.operator/default
```

#### Example output

```
Name:      default
Namespace:
Labels:    <none>
Annotations: <none>
API Version: operator.openshift.io/v1
Kind:      DNS
...
Status:
  Cluster Domain: cluster.local 1
  Cluster IP:    172.30.0.10 2
...
```

- 1** The Cluster Domain field is the base DNS domain used to construct fully qualified pod and service domain names.
- 2** The Cluster IP is the address pods query for name resolution. The IP is defined as the 10th address in the service CIDR range.

2. To find the service CIDR range of your cluster, use the **oc get** command:

```
$ oc get networks.config/cluster -o jsonpath='{$.status.serviceNetwork}'
```

#### Example output

```
[172.30.0.0/16]
```

### 4.7.3. Using DNS forwarding

You can use DNS forwarding to override the default forwarding configuration in the **/etc/resolv.conf** file in the following ways:

- Specify name servers (**spec.servers**) for every zone. If the forwarded zone is the ingress domain managed by OpenShift Container Platform, then the upstream name server must be authorized for the domain.
- Provide a list of upstream DNS servers (**spec.upstreamResolvers**).
- Change the default forwarding policy.



#### NOTE

A DNS forwarding configuration for the default domain can have both the default servers specified in the **/etc/resolv.conf** file and the upstream DNS servers.

#### Procedure

1. Modify the DNS Operator object named **default**:

```
$ oc edit dns.operator/default
```

After you issue the previous command, the Operator creates and updates the config map named **dns-default** with additional server configuration blocks based on **spec.servers**. If none

of the servers have a zone that matches the query, then name resolution falls back to the upstream DNS servers.

## Configuring DNS forwarding

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    negativeTTL: 0s
    positiveTTL: 0s
  logLevel: Normal
  nodePlacement: {}
  operatorLogLevel: Normal
  servers:
    - name: example-server 1
      zones:
        - example.com 2
      forwardPlugin:
        policy: Random 3
        upstreams: 4
        - 1.1.1.1
        - 2.2.2.2:5353
    upstreamResolvers: 5
    policy: Random 6
    protocolStrategy: "" 7
    transportConfig: {} 8
    upstreams:
      - type: SystemResolvConf 9
      - type: Network
        address: 1.2.3.4 10
        port: 53 11
    status:
      clusterDomain: cluster.local
      clusterIP: x.y.z.10
    conditions:
    ...
  
```

- 1 Must comply with the **rfc6335** service name syntax.
- 2 Must conform to the definition of a subdomain in the **rfc1123** service name syntax. The cluster domain, **cluster.local**, is an invalid subdomain for the **zones** field.
- 3 Defines the policy to select upstream resolvers listed in the **forwardPlugin**. Default value is **Random**. You can also use the values **RoundRobin**, and **Sequential**.
- 4 A maximum of 15 **upstreams** is allowed per **forwardPlugin**.
- 5 You can use **upstreamResolvers** to override the default forwarding policy and forward DNS resolution to the specified DNS resolvers (upstream resolvers) for the default domain. If you do not provide any upstream resolvers, the DNS name queries go to the servers declared in **/etc/resolv.conf**.

- 6 Determines the order in which upstream servers listed in **upstreams** are selected for querying. You can specify one of these values: **Random**, **RoundRobin**, or **Sequential**. The default is **RoundRobin**.
- 7 When omitted, the platform chooses a default, normally the protocol of the original client request. Set to **TCP** to specify that the platform should use TCP for all upstream DNS requests, even if the client request uses UDP.
- 8 Used to configure the transport type, server name, and optional custom CA or CA bundle to use when forwarding DNS requests to an upstream resolver.
- 9 You can specify two types of **upstreams**: **SystemResolvConf** or **Network**. **SystemResolvConf** configures the upstream to use `/etc/resolv.conf` and **Network** defines a **Networkresolver**. You can specify one or both.
- 10 If the specified type is **Network**, you must provide an IP address. The **address** field must be a valid IPv4 or IPv6 address.
- 11 If the specified type is **Network**, you can optionally provide a port. The **port** field must have a value between **1** and **65535**. If you do not specify a port for the upstream, the default port is **853**.

## Additional resources

- For more information on DNS forwarding, see the [CoreDNS forward documentation](#).

### 4.7.4. Checking DNS Operator status

You can inspect the status and view the details of the DNS Operator using the **oc describe** command.

#### Procedure

- View the status of the DNS Operator:

```
$ oc describe clusteroperators/dns
```

Though the messages and spelling might vary in a specific release, the expected status output looks like:

```
Status:
Conditions:
Last Transition Time: <date>
Message:      DNS "default" is available.
Reason:       AsExpected
Status:       True
Type:        Available
Last Transition Time: <date>
Message:      Desired and current number of DNSes are equal
Reason:       AsExpected
Status:       False
Type:        Progressing
Last Transition Time: <date>
Reason:       DNSNotDegraded
Status:       False
Type:        Degraded
```

```
Last Transition Time: <date>
Message:      DNS default is upgradeable: DNS Operator can be upgraded
Reason:       DNSUpgradeable
Status:        True
Type:         Upgradeable
```

#### 4.7.5. Viewing DNS Operator logs

You can view DNS Operator logs by using the **oc logs** command.

##### Procedure

- View the logs of the DNS Operator:

```
$ oc logs -n openshift-dns-operator deployment/dns-operator -c dns-operator
```

#### 4.7.6. Setting the CoreDNS log level

Log levels for CoreDNS and the CoreDNS Operator are set by using different methods. You can configure the CoreDNS log level to determine the amount of detail in logged error messages. The valid values for CoreDNS log level are **Normal**, **Debug**, and **Trace**. The default **logLevel** is **Normal**.



##### NOTE

The CoreDNS error log level is always enabled. The following log level settings report different error responses:

- logLevel: Normal** enables the "errors" class: **log . { class error }**.
- logLevel: Debug** enables the "denial" class: **log . { class denial error }**.
- logLevel: Trace** enables the "all" class: **log . { class all }**.

##### Procedure

- To set **logLevel** to **Debug**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Debug"}}' --type=merge
```

- To set **logLevel** to **Trace**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Trace"}}' --type=merge
```

##### Verification

- To ensure the desired log level was set, check the config map:

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

For example, after setting the **logLevel** to **Trace**, you should see this stanza in each server block:

```

errors
log . {
    class all
}

```

#### 4.7.7. Viewing the CoreDNS logs

You can view CoreDNS logs by using the **oc logs** command.

##### Procedure

- View the logs of a specific CoreDNS pod by entering the following command:

```
$ oc -n openshift-dns logs -c dns <core_dns_pod_name>
```

- Follow the logs of all CoreDNS pods by entering the following command:

```
$ oc -n openshift-dns logs -c dns -l dns.operator.openshift.io/daemonset-dns=default -f --max-log-requests=<number> ①
```

① Specifies the number of DNS pods to stream logs from. The maximum is 6.

#### 4.7.8. Setting the CoreDNS Operator log level

Log levels for CoreDNS and CoreDNS Operator are set by using different methods. Cluster administrators can configure the Operator log level to more quickly track down OpenShift DNS issues. The valid values for **operatorLogLevel** are **Normal**, **Debug**, and **Trace**. **Trace** has the most detailed information. The default **operatorLogLevel** is **Normal**. There are seven logging levels for Operator issues: Trace, Debug, Info, Warning, Error, Fatal, and Panic. After the logging level is set, log entries with that severity or anything above it will be logged.

- operatorLogLevel: "Normal"** sets **logrus.SetLogLevel("Info")**.
- operatorLogLevel: "Debug"** sets **logrus.SetLogLevel("Debug")**.
- operatorLogLevel: "Trace"** sets **logrus.SetLogLevel("Trace")**.

##### Procedure

- To set **operatorLogLevel** to **Debug**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Debug"}}' --type=merge
```

- To set **operatorLogLevel** to **Trace**, enter the following command:

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Trace"}}' --type=merge
```

##### Verification

- To review the resulting change, enter the following command:

```
$ oc get dnses.operator -A -oyaml
```

You should see two log level entries. The **operatorLogLevel** applies to OpenShift DNS Operator issues, and the **logLevel** applies to the daemonset of CoreDNS pods:

```
logLevel: Trace
operatorLogLevel: Debug
```

2. To review the logs for the daemonset, enter the following command:

```
$ oc logs -n openshift-dns ds/dns-default
```

#### 4.7.9. Tuning the CoreDNS cache

For CoreDNS, you can configure the maximum duration of both successful or unsuccessful caching, also known respectively as positive or negative caching. Tuning the cache duration of DNS query responses can reduce the load for any upstream DNS resolvers.



#### WARNING

Setting TTL fields to low values could lead to an increased load on the cluster, any upstream resolvers, or both.

#### Procedure

1. Edit the DNS Operator object named **default** by running the following command:

```
$ oc edit dns.operator.openshift.io/default
```

2. Modify the time-to-live (TTL) caching values:

#### Configuring DNS caching

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    positiveTTL: 1h ①
    negativeTTL: 0.5h10m ②
```

- ①** The string value **1h** is converted to its respective number of seconds by CoreDNS. If this field is omitted, the value is assumed to be **0s** and the cluster uses the internal default value of **900s** as a fallback.
- ②** The string value can be a combination of units such as **0.5h10m** and is converted to its respective number of seconds by CoreDNS. If this field is omitted, the value is assumed to be **0s** and the cluster uses the internal default value of **900s** as a fallback.

be **10s** and the cluster uses the internal default value of **30s** as a fallback.

## Verification

- To review the change, look at the config map again by running the following command:

```
oc get configmap/dns-default -n openshift-dns -o yaml
```

- Verify that you see entries that look like the following example:

```
cache 3600 {
    denial 9984 2400
}
```

## Additional resources

For more information on caching, see [CoreDNS cache](#).

### 4.7.10. Advanced tasks

#### 4.7.10.1. Changing the DNS Operator managementState

The DNS Operator manages the CoreDNS component to provide a name resolution service for pods and services in the cluster. The **managementState** of the DNS Operator is set to **Managed** by default, which means that the DNS Operator is actively managing its resources. You can change it to **Unmanaged**, which means the DNS Operator is not managing its resources.

The following are use cases for changing the DNS Operator **managementState**:

- You are a developer and want to test a configuration change to see if it fixes an issue in CoreDNS. You can stop the DNS Operator from overwriting the configuration change by setting the **managementState** to **Unmanaged**.
- You are a cluster administrator and have reported an issue with CoreDNS, but need to apply a workaround until the issue is fixed. You can set the **managementState** field of the DNS Operator to **Unmanaged** to apply the workaround.

## Procedure

- Change **managementState** to **Unmanaged** in the DNS Operator:

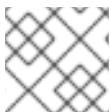
```
oc patch dns.operator.openshift.io default --type merge --patch '{"spec": {"managementState": "Unmanaged"}}'
```

- Review **managementState** of the DNS Operator using the **jsonpath** command-line JSON parser:

```
$ oc get dns.operator.openshift.io default -o jsonpath='{.spec.managementState}'
```

## Example output

```
"Unmanaged"
```

**NOTE**

You cannot upgrade while the **managementState** is set to **Unmanaged**.

#### 4.7.10.2. Controlling DNS pod placement

The DNS Operator has two daemon sets: one for CoreDNS called **dns-default** and one for managing the **/etc/hosts** file called **node-resolver**.

You can assign and run CoreDNS pods on specified nodes. For example, if the cluster administrator has configured security policies that prohibit communication between pairs of nodes, you can configure CoreDNS pods to run on a restricted set of nodes.

DNS service is available to all pods if the following circumstances are true:

- DNS pods are running on some nodes in the cluster.
- The nodes on which DNS pods are not running have network connectivity to nodes on which DNS pods are running,

The **node-resolver** daemon set must run on every node host because it adds an entry for the cluster image registry to support pulling images. The **node-resolver** pods have only one job: to look up the **image-registry.openshift-image-registry.svc** service's cluster IP address and add it to **/etc/hosts** on the node host so that the container runtime can resolve the service name.

As a cluster administrator, you can use a custom node selector to configure the daemon set for CoreDNS to run or not run on certain nodes.

#### Prerequisites

- You installed the **oc** CLI.
- You are logged in to the cluster as a user with **cluster-admin** privileges.
- Your DNS Operator **managementState** is set to **Managed**.

#### Procedure

- To allow the daemon set for CoreDNS to run on certain nodes, configure a taint and toleration:

1. Set a taint on the nodes that you want to control DNS pod placement by entering the following command:

```
$ oc adm taint nodes <node_name> dns-only=abc:NoExecute ①
```

- 1 Replace **<node\_name>** with the actual name of the node.

2. Modify the DNS Operator object named **default** to include the corresponding toleration by entering the following command:

```
$ oc edit dns.operator/default
```

3. Specify a taint key and a toleration for the taint. The following toleration matches the taint set on the nodes.

```

spec:
  nodePlacement:
    tolerations:
      - effect: NoExecute
        key: "dns-only" 1
        operator: Equal
        value: abc
      tolerationSeconds: 3600 2

```

- 1** If the **key** field is set to **dns-only**, it can be tolerated indefinitely.
- 2** The **tolerationSeconds** field is optional.

4. Optional: To specify node placement using a node selector, modify the default DNS Operator:

- a. Edit the DNS Operator object named **default** to include a node selector:

```

spec:
  nodePlacement:
    nodeSelector: 1
      node-role.kubernetes.io/control-plane: ""

```

- 1** This node selector ensures that the CoreDNS pods run only on control plane nodes.

#### 4.7.10.3. Configuring DNS forwarding with TLS

When working in a highly regulated environment, you might need the ability to secure DNS traffic when forwarding requests to upstream resolvers so that you can ensure additional DNS traffic and data privacy.

Be aware that CoreDNS caches forwarded connections for 10 seconds. CoreDNS will hold a TCP connection open for those 10 seconds if no request is issued. With large clusters, ensure that your DNS server is aware that it might get many new connections to hold open because you can initiate a connection per node. Set up your DNS hierarchy accordingly to avoid performance issues.

#### Procedure

1. Modify the DNS Operator object named **default**:

```
$ oc edit dns.operator/default
```

Cluster administrators can configure transport layer security (TLS) for forwarded DNS queries.

#### Configuring DNS forwarding with TLS

```

apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:

```

```

- name: example-server ①
  zones:
    - example.com ②
  forwardPlugin:
    transportConfig:
      transport: TLS ③
      tls:
        caBundle:
          name: mycacert
        serverName: dnstls.example.com ④
  policy: Random ⑤
  upstreams: ⑥
    - 1.1.1.1
    - 2.2.2.2:5353
  upstreamResolvers: ⑦
    transportConfig:
      transport: TLS
      tls:
        caBundle:
          name: mycacert
        serverName: dnstls.example.com
  upstreams:
    - type: Network ⑧
      address: 1.2.3.4 ⑨
      port: 53 ⑩

```

- ① Must comply with the **rfc6335** service name syntax.
- ② Must conform to the definition of a subdomain in the **rfc1123** service name syntax. The cluster domain, **cluster.local**, is an invalid subdomain for the **zones** field. The cluster domain, **cluster.local**, is an invalid **subdomain** for **zones**.
- ③ When configuring TLS for forwarded DNS queries, set the **transport** field to have the value **TLS**.
- ④ When configuring TLS for forwarded DNS queries, this is a mandatory server name used as part of the server name indication (SNI) to validate the upstream TLS server certificate.
- ⑤ Defines the policy to select upstream resolvers. Default value is **Random**. You can also use the values **RoundRobin**, and **Sequential**.
- ⑥ Required. Use it to provide upstream resolvers. A maximum of 15 **upstreams** entries are allowed per **forwardPlugin** entry.
- ⑦ Optional. You can use it to override the default policy and forward DNS resolution to the specified DNS resolvers (upstream resolvers) for the default domain. If you do not provide any upstream resolvers, the DNS name queries go to the servers in **/etc/resolv.conf**.
- ⑧ Only the **Network** type is allowed when using TLS and you must provide an IP address. **Network** type indicates that this upstream resolver should handle forwarded requests separately from the upstream resolvers listed in **/etc/resolv.conf**.
- ⑨ The **address** field must be a valid IPv4 or IPv6 address.
- ⑩

You can optionally provide a port. The **port** must have a value between **1** and **65535**. If you do not specify a port for the upstream, the default port is 853.



#### NOTE

If **servers** is undefined or invalid, the config map only contains the default server.

### Verification

1. View the config map:

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

#### Sample DNS ConfigMap based on TLS forwarding example

```
apiVersion: v1
data:
  Corefile: |
    example.com:5353 {
      forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
      forward . 3.3.3.3 4.4.4.4:5454 ①
    }
    .:5353 {
      errors
      health
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        upstream
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      forward . /etc/resolv.conf 1.2.3.4:53 {
        policy Random
      }
      cache 30
      reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

- ① Changes to the **forwardPlugin** triggers a rolling update of the CoreDNS daemon set.

### Additional resources

- For more information on DNS forwarding, see the [CoreDNS forward documentation](#).

## 4.8. INGRESS OPERATOR IN OPENSIFT CONTAINER PLATFORM

The Ingress Operator implements the **IngressController** API and is the component responsible for enabling external access to OpenShift Container Platform cluster services.

### 4.8.1. OpenShift Container Platform Ingress Operator

When you create your OpenShift Container Platform cluster, pods and services running on the cluster are each allocated their own IP addresses. The IP addresses are accessible to other pods and services running nearby but are not accessible to outside clients.

The Ingress Operator makes it possible for external clients to access your service by deploying and managing one or more HAProxy-based [Ingress Controllers](#) to handle routing. You can use the Ingress Operator to route traffic by specifying OpenShift Container Platform **Route** and Kubernetes **Ingress** resources. Configurations within the Ingress Controller, such as the ability to define **endpointPublishingStrategy** type and internal load balancing, provide ways to publish Ingress Controller endpoints.

### 4.8.2. The Ingress configuration asset

The installation program generates an asset with an **Ingress** resource in the **config.openshift.io** API group, **cluster-ingress-02-config.yml**.

#### YAML Definition of the Ingress resource

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.openshiftdemos.com
```

The installation program stores this asset in the **cluster-ingress-02-config.yml** file in the **manifests/** directory. This **Ingress** resource defines the cluster-wide configuration for Ingress. This Ingress configuration is used as follows:

- The Ingress Operator uses the domain from the cluster Ingress configuration as the domain for the default Ingress Controller.
- The OpenShift API Server Operator uses the domain from the cluster Ingress configuration. This domain is also used when generating a default host for a **Route** resource that does not specify an explicit host.

### 4.8.3. Ingress Controller configuration parameters

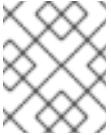
The **IngressController** custom resource (CR) includes optional configuration parameters that you can configure to meet specific needs for your organization.

Parameter	Description
-----------	-------------

Parameter	Description
<b>domain</b>	<p><b>domain</b> is a DNS name serviced by the Ingress Controller and is used to configure multiple features:</p> <ul style="list-style-type: none"> <li>For the <b>LoadBalancerService</b> endpoint publishing strategy, <b>domain</b> is used to configure DNS records. See <a href="#">endpointPublishingStrategy</a>.</li> <li>When using a generated default certificate, the certificate is valid for <b>domain</b> and its <b>subdomains</b>. See <a href="#">defaultCertificate</a>.</li> <li>The value is published to individual Route statuses so that users know where to target external DNS records.</li> </ul> <p>The <b>domain</b> value must be unique among all Ingress Controllers and cannot be updated.</p> <p>If empty, the default value is <code>ingress.config.openshift.io/cluster.spec.domain</code>.</p>
<b>replicas</b>	<p><b>replicas</b> is the number of Ingress Controller replicas. If not set, the default value is <b>2</b>.</p>

Parameter	Description
<b>endpointPublishingStrategy</b>	<p><b>endpointPublishingStrategy</b> is used to publish the Ingress Controller endpoints to other networks, enable load balancer integrations, and provide access to other systems.</p> <p>For cloud environments, use the <b>loadBalancer</b> field to configure the endpoint publishing strategy for your Ingress Controller.</p> <p>On GCP, AWS, and Azure you can configure the following <b>endpointPublishingStrategy</b> fields:</p> <ul style="list-style-type: none"> <li>• <b>loadBalancer.scope</b></li> <li>• <b>loadBalancer.allowedSourceRanges</b></li> </ul> <p>If not set, the default value is based on <b>infrastructure.config.openshift.io/cluster .status.platform</b>:</p> <ul style="list-style-type: none"> <li>• Azure: <b>LoadBalancerService</b> (with External scope)</li> <li>• Google Cloud Platform (GCP): <b>LoadBalancerService</b> (with External scope)</li> </ul> <p>For most platforms, the <b>endpointPublishingStrategy</b> value can be updated. On GCP, you can configure the following <b>endpointPublishingStrategy</b> fields:</p> <ul style="list-style-type: none"> <li>• <b>loadBalancer.scope</b></li> <li>• <b>loadbalancer.providerParameters.gcp.clientAccess</b></li> </ul> <p>For non-cloud environments, such as a bare-metal platform, use the <b>NodePortService</b>, <b>HostNetwork</b>, or <b>Private</b> fields to configure the endpoint publishing strategy for your Ingress Controller.</p> <p>If you do not set a value in one of these fields, the default value is based on binding ports specified in the <b>.status.platform</b> value in the <b>IngressController</b> CR.</p> <p>If you need to update the <b>endpointPublishingStrategy</b> value after your cluster is deployed, you can configure the following <b>endpointPublishingStrategy</b> fields:</p> <ul style="list-style-type: none"> <li>• <b>hostNetwork.protocol</b></li> <li>• <b>nodePort.protocol</b></li> <li>• <b>private.protocol</b></li> </ul>

Parameter	Description
<b>defaultCertificate</b>	<p>The <b>defaultCertificate</b> value is a reference to a secret that contains the default certificate that is served by the Ingress Controller. When Routes do not specify their own certificate, <b>defaultCertificate</b> is used.</p> <p>The secret must contain the following keys and data: * <b>tls.crt</b>: certificate file contents * <b>tls.key</b>: key file contents</p> <p>If not set, a wildcard certificate is automatically generated and used. The certificate is valid for the Ingress Controller <b>domain</b> and <b>subdomains</b>, and the generated certificate's CA is automatically integrated with the cluster's trust store.</p> <p>The in-use certificate, whether generated or user-specified, is automatically integrated with OpenShift Container Platform built-in OAuth server.</p>
<b>namespaceSelector</b>	<p><b>namespaceSelector</b> is used to filter the set of namespaces serviced by the Ingress Controller. This is useful for implementing shards.</p>
<b>routeSelector</b>	<p><b>routeSelector</b> is used to filter the set of Routes serviced by the Ingress Controller. This is useful for implementing shards.</p>
<b>nodePlacement</b>	<p><b>nodePlacement</b> enables explicit control over the scheduling of the Ingress Controller.</p> <p>If not set, the defaults values are used.</p> <div data-bbox="514 1176 620 1619" style="background-color: #f0f0f0; width: 67px; height: 197px; margin-right: 10px;"></div> <p><b>NOTE</b></p> <p>The <b>nodePlacement</b> parameter includes two parts, <b>nodeSelector</b> and <b>tolerations</b>. For example:</p> <pre data-bbox="742 1356 1064 1603"> nodePlacement:   nodeSelector:     matchLabels:       kubernetes.io/os: linux   tolerations:     - effect: NoSchedule       operator: Exists </pre>

Parameter	Description
<b>tlsSecurityProfile</b>	<p><b>tlsSecurityProfile</b> specifies settings for TLS connections for Ingress Controllers.</p> <p>If not set, the default value is based on the <b>apiservers.config.openshift.io/cluster</b> resource.</p> <p>When using the <b>Old</b>, <b>Intermediate</b>, and <b>Modern</b> profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the <b>Intermediate</b> profile deployed on release <b>X.Y.Z</b>, an upgrade to release <b>X.Y.Z+1</b> may cause a new profile configuration to be applied to the Ingress Controller, resulting in a rollout.</p> <p>The minimum TLS version for Ingress Controllers is <b>1.1</b>, and the maximum TLS version is <b>1.3</b>.</p>  <p><b>NOTE</b></p> <p>Ciphers and the minimum TLS version of the configured security profile are reflected in the <b>TLSProfile</b> status.</p>  <p><b>IMPORTANT</b></p> <p>The Ingress Operator converts the TLS <b>1.0</b> of an <b>Old</b> or <b>Custom</b> profile to <b>1.1</b>.</p>
<b>clientTLS</b>	<p><b>clientTLS</b> authenticates client access to the cluster and services; as a result, mutual TLS authentication is enabled. If not set, then client TLS is not enabled.</p> <p><b>clientTLS</b> has the required subfields, <b>spec.clientTLS.clientCertificatePolicy</b> and <b>spec.clientTLS.ClientCA</b>.</p> <p>The <b>ClientCertificatePolicy</b> subfield accepts one of the two values: <b>Required</b> or <b>Optional</b>. The <b>ClientCA</b> subfield specifies a config map that is in the openshift-config namespace. The config map should contain a CA certificate bundle.</p> <p>The <b>AllowedSubjectPatterns</b> is an optional value that specifies a list of regular expressions, which are matched against the distinguished name on a valid client certificate to filter requests. The regular expressions must use PCRE syntax. At least one pattern must match a client certificate's distinguished name; otherwise, the Ingress Controller rejects the certificate and denies the connection. If not specified, the Ingress Controller does not reject certificates based on the distinguished name.</p>

Parameter	Description
<b>routeAdmission</b>	<p><b>routeAdmission</b> defines a policy for handling new route claims, such as allowing or denying claims across namespaces.</p> <p><b>namespaceOwnership</b> describes how hostname claims across namespaces should be handled. The default is <b>Strict</b>.</p> <ul style="list-style-type: none"> <li>• <b>Strict</b>: does not allow routes to claim the same hostname across namespaces.</li> <li>• <b>InterNamespaceAllowed</b>: allows routes to claim different paths of the same hostname across namespaces.</li> </ul> <p><b>wildcardPolicy</b> describes how routes with wildcard policies are handled by the Ingress Controller.</p> <ul style="list-style-type: none"> <li>• <b>WildcardsAllowed</b>: Indicates routes with any wildcard policy are admitted by the Ingress Controller.</li> <li>• <b>WildcardsDisallowed</b>: Indicates only routes with a wildcard policy of <b>None</b> are admitted by the Ingress Controller. Updating <b>wildcardPolicy</b> from <b>WildcardsAllowed</b> to <b>WildcardsDisallowed</b> causes admitted routes with a wildcard policy of <b>Subdomain</b> to stop working. These routes must be recreated to a wildcard policy of <b>None</b> to be readmitted by the Ingress Controller. <b>WildcardsDisallowed</b> is the default setting.</li> </ul>

Parameter	Description
<b>IngressControllerLogging</b>	<p><b>logging</b> defines parameters for what is logged where. If this field is empty, operational logs are enabled but access logs are disabled.</p> <ul style="list-style-type: none"> <li>● <b>access</b> describes how client requests are logged. If this field is empty, access logging is disabled.</li> <li>○ <b>destination</b> describes a destination for log messages.</li> <li>■ <b>type</b> is the type of destination for logs:           <ul style="list-style-type: none"> <li>● <b>Container</b> specifies that logs should go to a sidecar container. The Ingress Operator configures the container, named <b>logs</b>, on the Ingress Controller pod and configures the Ingress Controller to write logs to the container. The expectation is that the administrator configures a custom logging solution that reads logs from this container. Using container logs means that logs may be dropped if the rate of logs exceeds the container runtime capacity or the custom logging solution capacity.</li> <li>● <b>Syslog</b> specifies that logs are sent to a Syslog endpoint. The administrator must specify an endpoint that can receive Syslog messages. The expectation is that the administrator has configured a custom Syslog instance.</li> <li>■ <b>container</b> describes parameters for the <b>Container</b> logging destination type. Currently there are no parameters for container logging, so this field must be empty.</li> <li>■ <b>syslog</b> describes parameters for the <b>Syslog</b> logging destination type:           <ul style="list-style-type: none"> <li>● <b>address</b> is the IP address of the syslog endpoint that receives log messages.</li> <li>● <b>port</b> is the UDP port number of the syslog endpoint that receives log messages.</li> <li>● <b>maxLength</b> is the maximum length of the syslog message. It must be between <b>480</b> and <b>4096</b> bytes. If this field is empty, the maximum length is set to the default value of <b>1024</b> bytes.</li> <li>● <b>facility</b> specifies the syslog facility of log messages. If this field is empty, the facility is <b>local1</b>. Otherwise, it must specify a valid syslog facility: <b>kern</b>, <b>user</b>, <b>mail</b>, <b>daemon</b>, <b>auth</b>, <b>syslog</b>, <b>lpr</b>, <b>news</b>, <b>uucp</b>, <b>cron</b>, <b>auth2</b>, <b>ftp</b>, <b>ntp</b>, <b>audit</b>, <b>alert</b>, <b>cron2</b>, <b>local0</b>, <b>local1</b>, <b>local2</b>, <b>local3</b>, <b>local4</b>, <b>local5</b>, <b>local6</b>, or <b>local7</b>.</li> <li>○ <b>httpLogFormat</b> specifies the format of the log message for an HTTP request. If this field is empty, log messages use the implementation's default HTTP log format. For HAProxy's default HTTP log format, see <a href="#">the HAProxy documentation</a>.</li> </ul> </li> </ul> </li> </ul>

Parameter	Description
<b>httpHeaders</b>	<p><b>httpHeaders</b> defines the policy for HTTP headers.</p> <p>By setting the <b>forwardedHeaderPolicy</b> for the <b>IngressControllerHTTPHeaders</b>, you specify when and how the Ingress Controller sets the <b>Forwarded</b>, <b>X-Forwarded-For</b>, <b>X-Forwarded-Host</b>, <b>X-Forwarded-Port</b>, <b>X-Forwarded-Proto</b>, and <b>X-Forwarded-Proto-Version</b> HTTP headers.</p> <p>By default, the policy is set to <b>Append</b>.</p> <ul style="list-style-type: none"> <li>• <b>Append</b> specifies that the Ingress Controller appends the headers, preserving any existing headers.</li> <li>• <b>Replace</b> specifies that the Ingress Controller sets the headers, removing any existing headers.</li> <li>• <b>IfNone</b> specifies that the Ingress Controller sets the headers if they are not already set.</li> <li>• <b>Never</b> specifies that the Ingress Controller never sets the headers, preserving any existing headers.</li> </ul> <p>By setting <b>headerNameCaseAdjustments</b>, you can specify case adjustments that can be applied to HTTP header names. Each adjustment is specified as an HTTP header name with the desired capitalization. For example, specifying <b>X-Forwarded-For</b> indicates that the <b>x-forwarded-for</b> HTTP header should be adjusted to have the specified capitalization.</p> <p>These adjustments are only applied to cleartext, edge-terminated, and re-encrypt routes, and only when using HTTP/1.</p> <p>For request headers, these adjustments are applied only for routes that have the <b>haproxy.router.openshift.io/h1-adjust-case=true</b> annotation. For response headers, these adjustments are applied to all HTTP responses. If this field is empty, no request headers are adjusted.</p> <p><b>actions</b> specifies options for performing certain actions on headers. Headers cannot be set or deleted for TLS passthrough connections. The <b>actions</b> field has additional subfields <b>spec.httpHeader.actions.response</b> and <b>spec.httpHeader.actions.request</b>:</p> <ul style="list-style-type: none"> <li>• The <b>response</b> subfield specifies a list of HTTP response headers to set or delete.</li> <li>• The <b>request</b> subfield specifies a list of HTTP request headers to set or delete.</li> </ul>

Parameter	Description
<b>httpCompression</b>	<p><b>httpCompression</b> defines the policy for HTTP traffic compression.</p> <ul style="list-style-type: none"> <li>• <b>mimeTypes</b> defines a list of MIME types to which compression should be applied. For example, <b>text/css; charset=utf-8</b>, <b>text/html</b>, <b>text/*</b>, <b>image/svg+xml</b>, <b>application/octet-stream</b>, <b>X-custom/customsub</b>, using the format pattern, <b>type/subtype</b>; <b>[;attribute=value]</b>. The <b>types</b> are: application, image, message, multipart, text, video, or a custom type prefaced by <b>X-</b>; e.g. To see the full notation for MIME types and subtypes, see <a href="#">RFC1341</a></li> </ul>
<b>httpErrorCodePages</b>	<p><b>httpErrorCodePages</b> specifies custom HTTP error code response pages. By default, an IngressController uses error pages built into the IngressController image.</p>
<b>httpCaptureCookies</b>	<p><b>httpCaptureCookies</b> specifies HTTP cookies that you want to capture in access logs. If the <b>httpCaptureCookies</b> field is empty, the access logs do not capture the cookies.</p> <p>For any cookie that you want to capture, the following parameters must be in your <b>IngressController</b> configuration:</p> <ul style="list-style-type: none"> <li>• <b>name</b> specifies the name of the cookie.</li> <li>• <b>maxLength</b> specifies the maximum length of the cookie.</li> <li>• <b>matchType</b> specifies if the field <b>name</b> of the cookie exactly matches the capture cookie setting or is a prefix of the capture cookie setting. The <b>matchType</b> field uses the <b>Exact</b> and <b>Prefix</b> parameters.</li> </ul> <p>For example:</p> <pre>httpCaptureCookies: - matchType: Exact   maxLength: 128   name: MYCOOKIE</pre>

Parameter	Description
<b>httpCaptureHeaders</b>	<p><b>httpCaptureHeaders</b> specifies the HTTP headers that you want to capture in the access logs. If the <b>httpCaptureHeaders</b> field is empty, the access logs do not capture the headers.</p> <p><b>httpCaptureHeaders</b> contains two lists of headers to capture in the access logs. The two lists of header fields are <b>request</b> and <b>response</b>. In both lists, the <b>name</b> field must specify the header name and the <b>maxLength</b> field must specify the maximum length of the header. For example:</p> <pre>httpCaptureHeaders:   request:     - maxLength: 256       name: Connection     - maxLength: 128       name: User-Agent   response:     - maxLength: 256       name: Content-Type     - maxLength: 256       name: Content-Length</pre>
<b>tuningOptions</b>	<p><b>tuningOptions</b> specifies options for tuning the performance of Ingress Controller pods.</p> <ul style="list-style-type: none"> <li>● <b>clientFinTimeout</b> specifies how long a connection is held open while waiting for the client response to the server closing the connection. The default timeout is <b>1s</b>.</li> <li>● <b>clientTimeout</b> specifies how long a connection is held open while waiting for a client response. The default timeout is <b>30s</b>.</li> <li>● <b>headerBufferBytes</b> specifies how much memory is reserved, in bytes, for Ingress Controller connection sessions. This value must be at least <b>16384</b> if HTTP/2 is enabled for the Ingress Controller. If not set, the default value is <b>32768</b> bytes. Setting this field not recommended because <b>headerBufferBytes</b> values that are too small can break the Ingress Controller, and <b>headerBufferBytes</b> values that are too large could cause the Ingress Controller to use significantly more memory than necessary.</li> <li>● <b>headerBufferMaxRewriteBytes</b> specifies how much memory should be reserved, in bytes, from <b>headerBufferBytes</b> for HTTP header rewriting and appending for Ingress Controller connection sessions. The minimum value for <b>headerBufferMaxRewriteBytes</b> is <b>4096</b>. <b>headerBufferBytes</b> must be greater than <b>headerBufferMaxRewriteBytes</b> for incoming HTTP requests. If not set, the default value is <b>8192</b> bytes. Setting this field not recommended because <b>headerBufferMaxRewriteBytes</b> values that are too small can break the Ingress Controller and <b>headerBufferMaxRewriteBytes</b> values that are too large could cause the Ingress Controller to use significantly more memory than necessary.</li> <li>● <b>healthCheckInterval</b> specifies how long the router waits between health checks. The default is <b>5s</b>.</li> </ul>

Parameter	Description
	<ul style="list-style-type: none"> <li>● <b>serverFinTimeout</b> specifies how long a connection is held open while waiting for the server response to the client that is closing the connection. The default timeout is <b>1s</b>.</li> <li>● <b>serverTimeout</b> specifies how long a connection is held open while waiting for a server response. The default timeout is <b>30s</b>.</li> <li>● <b>threadCount</b> specifies the number of threads to create per HAProxy process. Creating more threads allows each Ingress Controller pod to handle more connections, at the cost of more system resources being used. HAProxy supports up to <b>64</b> threads. If this field is empty, the Ingress Controller uses the default value of <b>4</b> threads. The default value can change in future releases. Setting this field is not recommended because increasing the number of HAProxy threads allows Ingress Controller pods to use more CPU time under load, and prevent other pods from receiving the CPU resources they need to perform. Reducing the number of threads can cause the Ingress Controller to perform poorly.</li> <li>● <b>tlsInspectDelay</b> specifies how long the router can hold data to find a matching route. Setting this value too short can cause the router to fall back to the default certificate for edge-terminated, reencrypted, or passthrough routes, even when using a better matched certificate. The default inspect delay is <b>5s</b>.</li> <li>● <b>tunnelTimeout</b> specifies how long a tunnel connection, including websockets, remains open while the tunnel is idle. The default timeout is <b>1h</b>.</li> <li>● <b>maxConnections</b> specifies the maximum number of simultaneous connections that can be established per HAProxy process. Increasing this value allows each ingress controller pod to handle more connections at the cost of additional system resources. Permitted values are <b>0</b>, <b>-1</b>, any value within the range <b>2000</b> and <b>2000000</b>, or the field can be left empty. <ul style="list-style-type: none"> <li>○ If this field is left empty or has the value <b>0</b>, the Ingress Controller will use the default value of <b>50000</b>. This value is subject to change in future releases.</li> <li>○ If the field has the value of <b>-1</b>, then HAProxy will dynamically compute a maximum value based on the available <b>ulimits</b> in the running container. This process results in a large computed value that will incur significant memory usage compared to the current default value of <b>50000</b>.</li> <li>○ If the field has a value that is greater than the current operating system limit, the HAProxy process will not start.</li> <li>○ If you choose a discrete value and the router pod is migrated to a new node, it is possible the new node does not have an identical <b>ulimit</b> configured. In such cases, the pod fails to start.</li> <li>○ If you have nodes with different <b>ulimits</b> configured, and you choose a discrete value, it is recommended to use the value of <b>-1</b> for this field so that the maximum number of connections is calculated at runtime.</li> </ul> </li> </ul>

Parameter	Description
<b>logEmptyRequests</b>	<p><b>logEmptyRequests</b> specifies connections for which no request is received and logged. These empty requests come from load balancer health probes or web browser speculative connections (preconnect) and logging these requests can be undesirable. However, these requests can be caused by network errors, in which case logging empty requests can be useful for diagnosing the errors. These requests can be caused by port scans, and logging empty requests can aid in detecting intrusion attempts. Allowed values for this field are <b>Log</b> and <b>Ignore</b>. The default value is <b>Log</b>.</p> <p>The <b>LoggingPolicy</b> type accepts either one of two values:</p> <ul style="list-style-type: none"> <li>• <b>Log</b>: Setting this value to <b>Log</b> indicates that an event should be logged.</li> <li>• <b>Ignore</b>: Setting this value to <b>Ignore</b> sets the <b>dontlognull</b> option in the HAProxy configuration.</li> </ul>
<b>HTTPEmptyRequestsPolicy</b>	<p><b>HTTPEmptyRequestsPolicy</b> describes how HTTP connections are handled if the connection times out before a request is received. Allowed values for this field are <b>Respond</b> and <b>Ignore</b>. The default value is <b>Respond</b>.</p> <p>The <b>HTTPEmptyRequestsPolicy</b> type accepts either one of two values:</p> <ul style="list-style-type: none"> <li>• <b>Respond</b>: If the field is set to <b>Respond</b>, the Ingress Controller sends an HTTP <b>400</b> or <b>408</b> response, logs the connection if access logging is enabled, and counts the connection in the appropriate metrics.</li> <li>• <b>Ignore</b>: Setting this option to <b>Ignore</b> adds the <b>http-ignore-probes</b> parameter in the HAProxy configuration. If the field is set to <b>Ignore</b>, the Ingress Controller closes the connection without sending a response, then logs the connection, or incrementing metrics.</li> </ul> <p>These connections come from load balancer health probes or web browser speculative connections (preconnect) and can be safely ignored. However, these requests can be caused by network errors, so setting this field to <b>Ignore</b> can impede detection and diagnosis of problems. These requests can be caused by port scans, in which case logging empty requests can aid in detecting intrusion attempts.</p>

#### 4.8.3.1. Ingress Controller TLS security profiles

TLS security profiles provide a way for servers to regulate which ciphers a connecting client can use when connecting to the server.

##### 4.8.3.1.1. Understanding TLS security profiles

You can use a TLS (Transport Layer Security) security profile to define which TLS ciphers are required by various OpenShift Container Platform components. The OpenShift Container Platform TLS security profiles are based on [Mozilla recommended configurations](#).

You can specify one of the following TLS security profiles for each component:

**Table 4.11. TLS security profiles**

Profile	Description
<b>Old</b>	<p>This profile is intended for use with legacy clients or libraries. The profile is based on the <a href="#">Old backward compatibility</a> recommended configuration.</p> <p>The <b>Old</b> profile requires a minimum TLS version of 1.0.</p>  <p><b>NOTE</b></p> <p>For the Ingress Controller, the minimum TLS version is converted from 1.0 to 1.1.</p>
<b>Intermediate</b>	<p>This profile is the default TLS security profile for the Ingress Controller, kubelet, and control plane. The profile is based on the <a href="#">Intermediate compatibility</a> recommended configuration.</p> <p>The <b>Intermediate</b> profile requires a minimum TLS version of 1.2.</p>  <p><b>NOTE</b></p> <p>This profile is the recommended configuration for the majority of clients.</p>
<b>Modern</b>	<p>This profile is intended for use with modern clients that have no need for backwards compatibility. This profile is based on the <a href="#">Modern compatibility</a> recommended configuration.</p> <p>The <b>Modern</b> profile requires a minimum TLS version of 1.3.</p>
<b>Custom</b>	<p>This profile allows you to define the TLS version and ciphers to use.</p> <div style="background-color: #ffffcc; padding: 10px;">  <p><b>WARNING</b></p> <p>Use caution when using a <b>Custom</b> profile, because invalid configurations can cause problems.</p> </div>



### NOTE

When using one of the predefined profile types, the effective profile configuration is subject to change between releases. For example, given a specification to use the Intermediate profile deployed on release X.Y.Z, an upgrade to release X.Y.Z+1 might cause a new profile configuration to be applied, resulting in a rollout.

#### 4.8.3.1.2. Configuring the TLS security profile for the Ingress Controller

To configure a TLS security profile for an Ingress Controller, edit the **IngressController** custom resource (CR) to specify a predefined or custom TLS security profile. If a TLS security profile is not configured, the default value is based on the TLS security profile set for the API server.

### Sample IngressController CR that configures the **Old** TLS security profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
...
```

The TLS security profile defines the minimum TLS version and the TLS ciphers for TLS connections for Ingress Controllers.

You can see the ciphers and the minimum TLS version of the configured TLS security profile in the **IngressController** custom resource (CR) under **Status.Tls Profile** and the configured TLS security profile under **Spec.Tls Security Profile**. For the **Custom** TLS security profile, the specific ciphers and minimum TLS version are listed under both parameters.



#### NOTE

The HAProxy Ingress Controller image supports TLS **1.3** and the **Modern** profile.

The Ingress Operator also converts the TLS **1.0** of an **Old** or **Custom** profile to **1.1**.

### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

### Procedure

1. Edit the **IngressController** CR in the **openshift-ingress-operator** project to configure the TLS security profile:

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. Add the **spec.tlsSecurityProfile** field:

### Sample IngressController CR for a **Custom** profile

```
apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    type: Custom 1
    custom: 2
    ciphers: 3
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
```

```

- ECDHE-RSA-AES128-GCM-SHA256
- ECDHE-ECDSA-AES128-GCM-SHA256
minTLSVersion: VersionTLS11
...

```

- 1 Specify the TLS security profile type (**Old**, **Intermediate**, or **Custom**). The default is **Intermediate**.
- 2 Specify the appropriate field for the selected type:
  - **old: {}**
  - **intermediate: {}**
  - **custom:**
- 3 For the **custom** type, specify a list of TLS ciphers and minimum accepted TLS version.

3. Save the file to apply the changes.

## Verification

- Verify that the profile is set in the **IngressController** CR:

```
$ oc describe IngressController default -n openshift-ingress-operator
```

## Example output

```

Name:      default
Namespace: openshift-ingress-operator
Labels:    <none>
Annotations: <none>
API Version: operator.openshift.io/v1
Kind:      IngressController
...
Spec:
...
Tls Security Profile:
Custom:
Ciphers:
ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-RSA-CHACHA20-POLY1305
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
Min TLS Version: VersionTLS11
Type:      Custom
...

```

### 4.8.3.1.3. Configuring mutual TLS authentication

You can configure the Ingress Controller to enable mutual TLS (mTLS) authentication by setting a **spec.clientTLS** value. The **clientTLS** value configures the Ingress Controller to verify client certificates. This configuration includes setting a **clientCA** value, which is a reference to a config map. The config

map contains the PEM-encoded CA certificate bundle that is used to verify a client's certificate. Optionally, you can also configure a list of certificate subject filters.

If the **clientCA** value specifies an X509v3 certificate revocation list (CRL) distribution point, the Ingress Operator downloads and manages a CRL config map based on the HTTP URI X509v3 **CRL Distribution Point** specified in each provided certificate. The Ingress Controller uses this config map during mTLS/TLS negotiation. Requests that do not provide valid certificates are rejected.

## Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a PEM-encoded CA certificate bundle.
- If your CA bundle references a CRL distribution point, you must have also included the end-entity or leaf certificate to the client CA bundle. This certificate must have included an HTTP URI under **CRL Distribution Points**, as described in RFC 5280. For example:

```
Issuer: C=US, O=Example Inc, CN=Example Global G2 TLS RSA SHA256 2020 CA1
Subject: SOME SIGNED CERT      X509v3 CRL Distribution Points:
Full Name:
URI:http://crl.example.com/example.crl
```

## Procedure

1. In the **openshift-config** namespace, create a config map from your CA bundle:

```
$ oc create configmap \
router-ca-certs-default \
--from-file=ca-bundle.pem=client-ca.crt \①
-n openshift-config
```

- 1 The config map data key must be **ca-bundle.pem**, and the data value must be a CA certificate in PEM format.

2. Edit the **IngressController** resource in the **openshift-ingress-operator** project:

```
$ oc edit IngressController default -n openshift-ingress-operator
```

3. Add the **spec.clientTLS** field and subfields to configure mutual TLS:

### Sample IngressController CR for a **clientTLS** profile that specifies filtering patterns

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:
```

```

name: router-ca-certs-default
allowedSubjectPatterns:
- "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"

```

4. Optional, get the Distinguished Name (DN) for **allowedSubjectPatterns** by entering the following command.

```
$ openssl x509 -in custom-cert.pem -noout -subject
subject= /CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift
```

#### 4.8.4. View the default Ingress Controller

The Ingress Operator is a core feature of OpenShift Container Platform and is enabled out of the box.

Every new OpenShift Container Platform installation has an **ingresscontroller** named default. It can be supplemented with additional Ingress Controllers. If the default **ingresscontroller** is deleted, the Ingress Operator will automatically recreate it within a minute.

##### Procedure

- View the default Ingress Controller:

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/default
```

#### 4.8.5. View Ingress Operator status

You can view and inspect the status of your Ingress Operator.

##### Procedure

- View your Ingress Operator status:

```
$ oc describe clusteroperators/ingress
```

#### 4.8.6. View Ingress Controller logs

You can view your Ingress Controller logs.

##### Procedure

- View your Ingress Controller logs:

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator -c
<container_name>
```

#### 4.8.7. View Ingress Controller status

You can view the status of a particular Ingress Controller.

##### Procedure

- View the status of an Ingress Controller:

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/<name>
```

#### 4.8.8. Creating a custom Ingress Controller

As a cluster administrator, you can create a new custom Ingress Controller. Because the default Ingress Controller might change during OpenShift Container Platform updates, creating a custom Ingress Controller can be helpful when maintaining a configuration manually that persists across cluster updates.

This example provides a minimal spec for a custom Ingress Controller. To further customize your custom Ingress Controller, see "Configuring the Ingress Controller".

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

##### Procedure

1. Create a YAML file that defines the custom **IngressController** object:

###### Example **custom-ingress-controller.yaml** file

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: <custom_name> ①
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: <custom-ingress-custom-certs> ②
  replicas: 1 ③
  domain: <custom_domain> ④
```

- 1 Specify the a custom **name** for the **IngressController** object.
- 2 Specify the name of the secret with the custom wildcard certificate.
- 3 Minimum replica needs to be ONE
- 4 Specify the domain to your domain name. The domain specified on the IngressController object and the domain used for the certificate must match. For example, if the domain value is "custom\_domain.mycompany.com", then the certificate must have SAN \*.custom\_domain.mycompany.com (with the \*. added to the domain).

2. Create the object by running the following command:

```
$ oc create -f custom-ingress-controller.yaml
```

#### 4.8.9. Configuring the Ingress Controller

### 4.8.9.1. Setting a custom default certificate

As an administrator, you can configure an Ingress Controller to use a custom certificate by creating a Secret resource and editing the **IngressController** custom resource (CR).

#### Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is signed by a trusted certificate authority or by a private trusted certificate authority that you configured in a custom PKI.
- Your certificate meets the following requirements:
  - The certificate is valid for the ingress domain.
  - The certificate uses the **subjectAltName** extension to specify a wildcard domain, such as `*.apps.ocp4.example.com`.
- You must have an **IngressController** CR. You may use the default one:

```
$ oc --namespace openshift-ingress-operator get ingresscontrollers
```

#### Example output

NAME	AGE
default	10m



#### NOTE

If you have intermediate certificates, they must be included in the **tls.crt** file of the secret containing a custom default certificate. Order matters when specifying a certificate; list your intermediate certificate(s) after any server certificate(s).

#### Procedure

The following assumes that the custom certificate and key pair are in the **tls.crt** and **tls.key** files in the current working directory. Substitute the actual path names for **tls.crt** and **tls.key**. You also may substitute another name for **custom-certs-default** when creating the Secret resource and referencing it in the IngressController CR.



#### NOTE

This action will cause the Ingress Controller to be redeployed, using a rolling deployment strategy.

1. Create a Secret resource containing the custom certificate in the **openshift-ingress** namespace using the **tls.crt** and **tls.key** files.

```
$ oc --namespace openshift-ingress create secret tls custom-certs-default --cert=tls.crt --key=tls.key
```

2. Update the IngressController CR to reference the new certificate secret:

```
$ oc patch --type=merge --namespace openshift-ingress-operator ingresscontrollers/default \
--patch '{"spec":{"defaultCertificate":{"name":"custom-certs-default"}}}'
```

3. Verify the update was effective:

```
$ echo Q |\
openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null |
openssl x509 -noout -subject -issuer -enddate
```

where:

#### **<domain>**

Specifies the base domain name for your cluster.

#### **Example output**

```
subject=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = *.apps.example.com
issuer=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = example.com
notAfter=May 10 08:32:45 2022 GM
```

#### **TIP**

You can alternatively apply the following YAML to set a custom default certificate:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: custom-certs-default
```

The certificate secret name should match the value used to update the CR.

Once the IngressController CR has been modified, the Ingress Operator updates the Ingress Controller's deployment to use the custom certificate.

#### **4.8.9.2. Removing a custom default certificate**

As an administrator, you can remove a custom certificate that you configured an Ingress Controller to use.

#### **Prerequisites**

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).
- You previously configured a custom default certificate for the Ingress Controller.

## Procedure

- To remove the custom certificate and restore the certificate that ships with OpenShift Container Platform, enter the following command:

```
$ oc patch -n openshift-ingress-operator ingresscontrollers/default \
--type json -p $'{"op": "remove", "path": "/spec/defaultCertificate"}'
```

There can be a delay while the cluster reconciles the new certificate configuration.

## Verification

- To confirm that the original cluster certificate is restored, enter the following command:

```
$ echo Q | \
openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null | \
openssl x509 -noout -subject -issuer -enddate
```

where:

### <domain>

Specifies the base domain name for your cluster.

## Example output

```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

### 4.8.9.3. Autoscaling an Ingress Controller

You can automatically scale an Ingress Controller to dynamically meet routing performance or availability requirements, such as the requirement to increase throughput.

The following procedure provides an example for scaling up the default Ingress Controller.

## Prerequisites

- You have the OpenShift CLI (**oc**) installed.
- You have access to an OpenShift Container Platform cluster as a user with the **cluster-admin** role.
- You installed the Custom Metrics Autoscaler Operator and an associated KEDA Controller.
  - You can install the Operator by using OperatorHub on the web console. After you install the Operator, you can create an instance of **KedaController**.

## Procedure

- Create a service account to authenticate with Thanos by running the following command:

```
$ oc create -n openshift-ingress-operator serviceaccount thanos && oc describe -n openshift-ingress-operator serviceaccount thanos
```

### Example output

```
Name:      thanos
Namespace:  openshift-ingress-operator
Labels:    <none>
Annotations: <none>
Image pull secrets: thanos-dockercfg-kfvf2
Mountable secrets:  thanos-dockercfg-kfvf2
Tokens:    <none>
Events:    <none>
```

2. Manually create the service account secret token with the following command:

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: thanos-token
  namespace: openshift-ingress-operator
  annotations:
    kubernetes.io/service-account.name: thanos
type: kubernetes.io/service-account-token
EOF
```

3. Define a **TriggerAuthentication** object within the **openshift-ingress-operator** namespace by using the service account's token.

- a. Create the **TriggerAuthentication** object and pass the value of the **secret** variable to the **TOKEN** parameter:

```
$ oc apply -f - <<EOF
apiVersion: keda.sh/v1alpha1
kind: TriggerAuthentication
metadata:
  name: keda-trigger-auth-prometheus
  namespace: openshift-ingress-operator
spec:
  secretTargetRef:
    - parameter: bearerToken
      name: thanos-token
      key: token
    - parameter: ca
      name: thanos-token
      key: ca.crt
EOF
```

4. Create and apply a role for reading metrics from Thanos:

- a. Create a new role, **thanos-metrics-reader.yaml**, that reads metrics from pods and nodes:

**thanos-metrics-reader.yaml**

-

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: thanos-metrics-reader
  namespace: openshift-ingress-operator
rules:
- apiGroups:
  - ""
    resources:
    - pods
    - nodes
    verbs:
    - get
- apiGroups:
  - metrics.k8s.io
    resources:
    - pods
    - nodes
    verbs:
    - get
    - list
    - watch
- apiGroups:
  - ""
    resources:
    - namespaces
    verbs:
    - get

```

- b. Apply the new role by running the following command:

```
$ oc apply -f thanos-metrics-reader.yaml
```

5. Add the new role to the service account by entering the following commands:

```
$ oc adm policy -n openshift-ingress-operator add-role-to-user thanos-metrics-reader -z
thanos --role=namespace=openshift-ingress-operator
```

```
$ oc adm policy -n openshift-ingress-operator add-cluster-role-to-user cluster-monitoring-view
-z thanos
```



#### NOTE

The argument **add-cluster-role-to-user** is only required if you use cross-namespace queries. The following step uses a query from the **kube-metrics** namespace which requires this argument.

6. Create a new **ScaledObject** YAML file, **ingress-autoscaler.yaml**, that targets the default Ingress Controller deployment:

#### Example ScaledObject definition

```
apiVersion: keda.sh/v1alpha1
```

```

kind: ScaledObject
metadata:
  name: ingress-scaler
  namespace: openshift-ingress-operator
spec:
  scaleTargetRef: ①
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 ②
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
    - type: prometheus
      metricType: AverageValue
      metadata:
        serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 ③
        namespace: openshift-ingress-operator ④
        metricName: 'kube-node-role'
        threshold: '1'
        query: 'sum(kube_node_role{role="worker",service="kube-state-metrics"})' ⑤
        authModes: "bearer"
      authenticationRef:
        name: keda-trigger-auth-prometheus

```

- ① The custom resource that you are targeting. In this case, the Ingress Controller.
- ② Optional: The maximum number of replicas. If you omit this field, the default maximum is set to 100 replicas.
- ③ The Thanos service endpoint in the **openshift-monitoring** namespace.
- ④ The Ingress Operator namespace.
- ⑤ This expression evaluates to however many worker nodes are present in the deployed cluster.



### IMPORTANT

If you are using cross-namespace queries, you must target port 9091 and not port 9092 in the **serverAddress** field. You also must have elevated privileges to read metrics from this port.

7. Apply the custom resource definition by running the following command:

```
$ oc apply -f ingress-autoscaler.yaml
```

### Verification

- Verify that the default Ingress Controller is scaled out to match the value returned by the **kube-state-metrics** query by running the following commands:

- Use the **grep** command to search the Ingress Controller YAML file for replicas:

```
$ oc get -n openshift-ingress-operator ingresscontroller/default -o yaml | grep replicas:
```

#### Example output

```
replicas: 3
```

- Get the pods in the **openshift-ingress** project:

```
$ oc get pods -n openshift-ingress
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
router-default-7b5df44ff-l9pmm	2/2	Running	0	17h
router-default-7b5df44ff-s5sl5	2/2	Running	0	3d22h
router-default-7b5df44ff-wwsth	2/2	Running	0	66s

#### Additional resources

- [Installing the custom metrics autoscaler](#)
- [Enabling monitoring for user-defined projects](#)
- [Understanding custom metrics autoscaler trigger authentications](#)
- [Understanding custom metrics autoscaler triggers](#)
- [Understanding how to add custom metrics autoscalers](#)

#### 4.8.9.4. Scaling an Ingress Controller

Manually scale an Ingress Controller to meeting routing performance or availability requirements such as the requirement to increase throughput. **oc** commands are used to scale the **IngressController** resource. The following procedure provides an example for scaling up the default **IngressController**.



#### NOTE

Scaling is not an immediate action, as it takes time to create the desired number of replicas.

#### Procedure

1. View the current number of available replicas for the default **IngressController**:

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o jsonpath='{$.status.availableReplicas}'
```

#### Example output

```
2
```

2. Scale the default **IngressController** to the desired number of replicas using the **oc patch** command. The following example scales the default **IngressController** to 3 replicas:

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":{"replicas":3}}' --type=merge
```

#### Example output

```
ingresscontroller.operator.openshift.io/default patched
```

3. Verify that the default **IngressController** scaled to the number of replicas that you specified:

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o jsonpath='{$.status.availableReplicas}'
```

#### Example output

```
3
```

#### TIP

You can alternatively apply the following YAML to scale an Ingress Controller to three replicas:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3
```

**1**

- 1** If you need a different amount of replicas, change the **replicas** value.

#### 4.8.9.5. Configuring Ingress access logging

You can configure the Ingress Controller to enable access logs. If you have clusters that do not receive much traffic, then you can log to a sidecar. If you have high traffic clusters, to avoid exceeding the capacity of the logging stack or to integrate with a logging infrastructure outside of OpenShift Container Platform, you can forward logs to a custom syslog endpoint. You can also specify the format for access logs.

Container logging is useful to enable access logs on low-traffic clusters when there is no existing Syslog logging infrastructure, or for short-term use while diagnosing problems with the Ingress Controller.

Syslog is needed for high-traffic clusters where access logs could exceed the OpenShift Logging stack's capacity, or for environments where any logging solution needs to integrate with an existing Syslog logging infrastructure. The Syslog use-cases can overlap.

#### Prerequisites

- Log in as a user with **cluster-admin** privileges.

## Procedure

Configure Ingress access logging to a sidecar.

- To configure Ingress access logging, you must specify a destination using **spec.logging.access.destination**. To specify logging to a sidecar container, you must specify **Container spec.logging.access.destination.type**. The following example is an Ingress Controller definition that logs to a **Container** destination:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
```

- When you configure the Ingress Controller to log to a sidecar, the operator creates a container named **logs** inside the Ingress Controller Pod:

```
$ oc -n openshift-ingress logs deployment.apps/router-default -c logs
```

### Example output

```
2020-05-11T19:11:50.135710+00:00 router-default-57dfc6cd95-bpmk6 router-default-
57dfc6cd95-bpmk6 haproxy[108]: 174.19.21.82:39654 [11/May/2020:19:11:50.133] public
be_http:hello-openshift:hello-openshift/pod:hello-openshift:hello-openshift:10.128.2.12:8080
0/0/1/0/1 200 142 - - -NI 1/1/0/0/0 0/0 "GET / HTTP/1.1"
```

Configure Ingress access logging to a Syslog endpoint.

- To configure Ingress access logging, you must specify a destination using **spec.logging.access.destination**. To specify logging to a Syslog endpoint destination, you must specify **Syslog** for **spec.logging.access.destination.type**. If the destination type is **Syslog**, you must also specify a destination endpoint using **spec.logging.access.destination.syslog.address** and you can specify a facility using **spec.logging.access.destination.syslog.facility**. The following example is an Ingress Controller definition that logs to a **Syslog** destination:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
```

```
syslog:
  address: 1.2.3.4
  port: 10514
```

**NOTE**

The **syslog** destination port must be UDP.

The **syslog** destination address must be an IP address. It does not support DNS hostname.

Configure Ingress access logging with a specific log format.

- You can specify **spec.logging.access.httpLogFormat** to customize the log format. The following example is an Ingress Controller definition that logs to a **syslog** endpoint with IP address 1.2.3.4 and port 10514:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
    httpLogFormat: '%ci:%cp [%t] %ft %b/%s %B %bq %HM %HU %HV'
```

Disable Ingress access logging.

- To disable Ingress access logging, leave **spec.logging** or **spec.logging.access** empty:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access: null
```

Allow the Ingress Controller to modify the HAProxy log length when using a sidecar.

- Use **spec.logging.access.destination.syslog.maxLength** if you are using **spec.logging.access.destination.type: Syslog**.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
```

```

metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          maxLength: 4096
          port: 10514

```

- Use **spec.logging.access.destination.container.maxLength** if you are using **spec.logging.access.destination.type: Container**.

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
        container:
          maxLength: 8192

```

#### 4.8.9.6. Setting Ingress Controller thread count

A cluster administrator can set the thread count to increase the amount of incoming connections a cluster can handle. You can patch an existing Ingress Controller to increase the amount of threads.

##### Prerequisites

- The following assumes that you already created an Ingress Controller.

##### Procedure

- Update the Ingress Controller to increase the number of threads:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec": {"tuningOptions": {"threadCount": 8}}}'
```



##### NOTE

If you have a node that is capable of running large amounts of resources, you can configure **spec.nodePlacement.nodeSelector** with labels that match the capacity of the intended node, and configure **spec.tuningOptions.threadCount** to an appropriately high value.

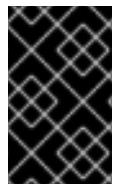
#### 4.8.9.7. Configuring an Ingress Controller to use an internal load balancer

When creating an Ingress Controller on cloud platforms, the Ingress Controller is published by a public cloud load balancer by default. As an administrator, you can create an Ingress Controller that uses an internal cloud load balancer.



#### WARNING

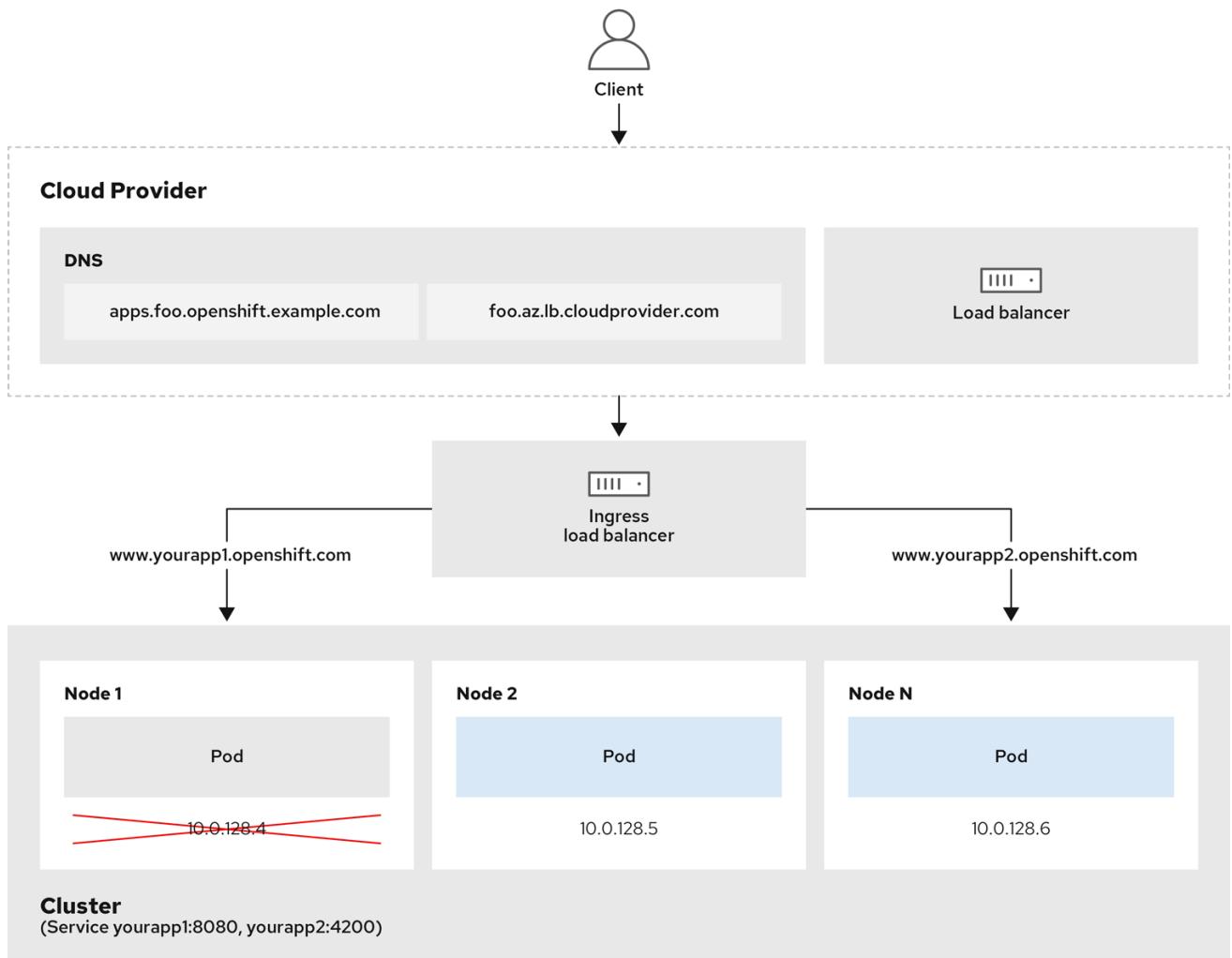
If your cloud provider is Microsoft Azure, you must have at least one public load balancer that points to your nodes. If you do not, all of your nodes will lose egress connectivity to the internet.



#### IMPORTANT

If you want to change the **scope** for an **IngressController**, you can change the **.spec.endpointPublishingStrategy.loadBalancer.scope** parameter after the custom resource (CR) is created.

Figure 4.2. Diagram of LoadBalancer



The preceding graphic shows the following concepts pertaining to OpenShift Container Platform Ingress LoadBalancerService endpoint publishing strategy:

- You can load balance externally, using the cloud provider load balancer, or internally, using the OpenShift Ingress Controller Load Balancer.
- You can use the single IP address of the load balancer and more familiar ports, such as 8080 and 4200 as shown on the cluster depicted in the graphic.
- Traffic from the external load balancer is directed at the pods, and managed by the load balancer, as depicted in the instance of a down node. See the [Kubernetes Services documentation](#) for implementation details.

## Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

## Procedure

1. Create an **IngressController** custom resource (CR) in a file named **<name>-ingress-controller.yaml**, such as in the following example:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: Internal 3
```

- 1 Replace **<name>** with a name for the **IngressController** object.
- 2 Specify the **domain** for the application published by the controller.
- 3 Specify a value of **Internal** to use an internal load balancer.

2. Create the Ingress Controller defined in the previous step by running the following command:

```
$ oc create -f <name>-ingress-controller.yaml 1
```

- 1 Replace **<name>** with the name of the **IngressController** object.

3. Optional: Confirm that the Ingress Controller was created by running the following command:

```
$ oc --all-namespaces=true get ingresscontrollers
```

#### 4.8.9.8. Configuring global access for an Ingress Controller on GCP

An Ingress Controller created on GCP with an internal load balancer generates an internal IP address for the service. A cluster administrator can specify the global access option, which enables clients in any region within the same VPC network and compute region as the load balancer, to reach the workloads running on your cluster.

For more information, see the GCP documentation for [global access](#).

##### Prerequisites

- You deployed an OpenShift Container Platform cluster on GCP infrastructure.
- You configured an Ingress Controller to use an internal load balancer.
- You installed the OpenShift CLI (**oc**).

##### Procedure

1. Configure the Ingress Controller resource to allow global access.



##### NOTE

You can also create an Ingress Controller and specify the global access option.

- a. Configure the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

- b. Edit the YAML file:

##### Sample clientAccess configuration to Global

```
spec:  
  endpointPublishingStrategy:  
  loadBalancer:  
    providerParameters:  
      gcp:  
        clientAccess: Global 1  
        type: GCP  
      scope: Internal  
      type: LoadBalancerService
```

**1**

Set **gcp.clientAccess** to **Global**.

- c. Save the file to apply the changes.

2. Run the following command to verify that the service allows global access:

```
$ oc -n openshift-ingress edit svc/router-default -o yaml
```

The output shows that global access is enabled for GCP with the annotation, **networking.gke.io/internal-load-balancer-allow-global-access**.

#### 4.8.9.9. Setting the Ingress Controller health check interval

A cluster administrator can set the health check interval to define how long the router waits between two consecutive health checks. This value is applied globally as a default for all routes. The default value is 5 seconds.

##### Prerequisites

- The following assumes that you already created an Ingress Controller.

##### Procedure

- Update the Ingress Controller to change the interval between back end health checks:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec": {"tuningOptions": {"healthCheckInterval": "8s"}}}'
```



##### NOTE

To override the **healthCheckInterval** for a single route, use the route annotation **router.openshift.io/haproxy.health.check.interval**

#### 4.8.9.10. Configuring the default Ingress Controller for your cluster to be internal

You can configure the **default** Ingress Controller for your cluster to be internal by deleting and recreating it.



##### WARNING

If your cloud provider is Microsoft Azure, you must have at least one public load balancer that points to your nodes. If you do not, all of your nodes will lose egress connectivity to the internet.



##### IMPORTANT

If you want to change the **scope** for an **IngressController**, you can change the **.spec.endpointPublishingStrategy.loadBalancer.scope** parameter after the custom resource (CR) is created.

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

##### Procedure

- Configure the **default** Ingress Controller for your cluster to be internal by deleting and recreating it.

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: Internal
EOF
```

#### 4.8.9.11. Configuring the route admission policy

Administrators and application developers can run applications in multiple namespaces with the same domain name. This is for organizations where multiple teams develop microservices that are exposed on the same hostname.



#### WARNING

Allowing claims across namespaces should only be enabled for clusters with trust between namespaces, otherwise a malicious user could take over a hostname. For this reason, the default admission policy disallows hostname claims across namespaces.

#### Prerequisites

- Cluster administrator privileges.

#### Procedure

- Edit the **.spec.routeAdmission** field of the **ingresscontroller** resource variable using the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec": {"routeAdmission": {"namespaceOwnership": "InterNamespaceAllowed"}}}' --type=merge
```

#### Sample Ingress Controller configuration

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
...
```

**TIP**

You can alternatively apply the following YAML to configure the route admission policy:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

#### 4.8.9.12. Using wildcard routes

The HAProxy Ingress Controller has support for wildcard routes. The Ingress Operator uses **wildcardPolicy** to configure the **ROUTER\_ALLOW\_WILDCARD\_ROUTES** environment variable of the Ingress Controller.

The default behavior of the Ingress Controller is to admit routes with a wildcard policy of **None**, which is backwards compatible with existing **IngressController** resources.

#### Procedure

- Configure the wildcard policy.
  - Use the following command to edit the **IngressController** resource:

```
$ oc edit IngressController
```

- Under **spec**, set the **wildcardPolicy** field to **WildcardsDisallowed** or **WildcardsAllowed**:
- ```
spec:
  routeAdmission:
    wildcardPolicy: WildcardsDisallowed # or WildcardsAllowed
```

#### 4.8.9.13. HTTP header configuration

OpenShift Container Platform provides different methods for working with HTTP headers. When setting or deleting headers, you can use specific fields in the Ingress Controller or an individual route to modify request and response headers. You can also set certain headers by using route annotations. The various ways of configuring headers can present challenges when working together.



#### NOTE

You can only set or delete headers within an **IngressController** or **Route** CR, you cannot append them. If an HTTP header is set with a value, that value must be complete and not require appending in the future. In situations where it makes sense to append a header, such as the X-Forwarded-For header, use the **spec.httpHeaders.forwardedHeaderPolicy** field, instead of **spec.httpHeaders.actions**.

##### 4.8.9.13.1. Order of precedence

When the same HTTP header is modified both in the Ingress Controller and in a route, HAProxy prioritizes the actions in certain ways depending on whether it is a request or response header.

- For HTTP response headers, actions specified in the Ingress Controller are executed after the actions specified in a route. This means that the actions specified in the Ingress Controller take precedence.
- For HTTP request headers, actions specified in a route are executed after the actions specified in the Ingress Controller. This means that the actions specified in the route take precedence.

For example, a cluster administrator sets the X-Frame-Options response header with the value **DENY** in the Ingress Controller using the following configuration:

### Example IngressController spec

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
        - name: X-Frame-Options
          action:
            type: Set
            set:
              value: DENY
```

A route owner sets the same response header that the cluster administrator set in the Ingress Controller, but with the value **SAMEORIGIN** using the following configuration:

### Example Route spec

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
        - name: X-Frame-Options
          action:
            type: Set
            set:
              value: SAMEORIGIN
```

When both the **IngressController** spec and **Route** spec are configuring the X-Frame-Options response header, then the value set for this header at the global level in the Ingress Controller takes precedence, even if a specific route allows frames. For a request header, the **Route** spec value overrides the **IngressController** spec value.

This prioritization occurs because the **haproxy.config** file uses the following logic, where the Ingress Controller is considered the front end and individual routes are considered the back end. The header value **DENY** applied to the front end configurations overrides the same header with the value **SAMEORIGIN** that is set in the back end:

```

frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'

```

Additionally, any actions defined in either the Ingress Controller or a route override values set using route annotations.

#### 4.8.9.13.2. Special case headers

The following headers are either prevented entirely from being set or deleted, or allowed under specific circumstances:

**Table 4.12. Special case header configuration options**

| Header name  | Configurable using <b>IngressController</b> spec | Configurable using <b>Route</b> spec | Reason for disallowment                                                                                                                                                                                                                                                   | Configurable using another method |
|--------------|--------------------------------------------------|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------|
| <b>proxy</b> | No                                               | No                                   | The <b>proxy</b> HTTP request header can be used to exploit vulnerable CGI applications by injecting the header value into the <b>HTTP_PROXY</b> environment variable. The <b>proxy</b> HTTP request header is also non-standard and prone to error during configuration. | No                                |
| <b>host</b>  | No                                               | Yes                                  | When the <b>host</b> HTTP request header is set using the <b>IngressController</b> CR, HAProxy can fail when looking up the correct route.                                                                                                                                | No                                |

| Header name                         | Configurable using <b>IngressController spec</b> | Configurable using <b>Route spec</b> | Reason for disallowment                                                                                                                                                                                                                             | Configurable using another method                                                                                                                                                                                 |
|-------------------------------------|--------------------------------------------------|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>strict-transport-security</b>    | No                                               | No                                   | The <b>strict-transport-security</b> HTTP response header is already handled using route annotations and does not need a separate implementation.                                                                                                   | Yes: the <b>haproxy.router.openshift.io/hsts_header</b> route annotation                                                                                                                                          |
| <b>cookie</b> and <b>set-cookie</b> | No                                               | No                                   | The cookies that HAProxy sets are used for session tracking to map client connections to particular backend servers. Allowing these headers to be set could interfere with HAProxy's session affinity and restrict HAProxy's ownership of a cookie. | <p>Yes:</p> <ul style="list-style-type: none"> <li>the <b>haproxy.router.openshift.io/disable_cookie</b> route annotation</li> <li>the <b>haproxy.router.openshift.io/cookie_name</b> route annotation</li> </ul> |

#### 4.8.9.14. Setting or deleting HTTP request and response headers in an Ingress Controller

You can set or delete certain HTTP request and response headers for compliance purposes or other reasons. You can set or delete these headers either for all routes served by an Ingress Controller or for specific routes.

For example, you might want to migrate an application running on your cluster to use mutual TLS, which requires that your application checks for an X-Forwarded-Client-Cert request header, but the OpenShift Container Platform default Ingress Controller provides an X-SSL-Client-Der request header.

The following procedure modifies the Ingress Controller to set the X-Forwarded-Client-Cert request header, and delete the X-SSL-Client-Der request header.

##### Prerequisites

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

- You have access to an OpenShift Container Platform cluster as a user with the **cluster-admin** role.

## Procedure

1. Edit the Ingress Controller resource:

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. Replace the X-SSL-Client-Der HTTP request header with the X-Forwarded-Client-Cert HTTP request header:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    actions: ①
      request: ②
        - name: X-Forwarded-Client-Cert ③
          action:
            type: Set ④
            set:
              value: "%{+Q}[ssl_c_der,base64]" ⑤
        - name: X-SSL-Client-Der
          action:
            type: Delete
```

- 1 The list of actions you want to perform on the HTTP headers.
- 2 The type of header you want to change. In this case, a request header.
- 3 The name of the header you want to change. For a list of available headers you can set or delete, see *HTTP header configuration*.
- 4 The type of action being taken on the header. This field can have the value **Set** or **Delete**.
- 5 When setting HTTP headers, you must provide a **value**. The value can be a string from a list of available directives for that header, for example **DENY**, or it can be a dynamic value that will be interpreted using HAProxy's dynamic value syntax. In this case, a dynamic value is added.



### NOTE

For setting dynamic header values for HTTP responses, allowed sample fetchers are **res.hdr** and **ssl\_c\_der**. For setting dynamic header values for HTTP requests, allowed sample fetchers are **req.hdr** and **ssl\_c\_der**. Both request and response dynamic values can use the **lower** and **base64** converters.

3. Save the file to apply the changes.

#### 4.8.9.15. Using X-Forwarded headers

You configure the HAProxy Ingress Controller to specify a policy for how to handle HTTP headers including **Forwarded** and **X-Forwarded-For**. The Ingress Operator uses the **HTTPHeaders** field to configure the **ROUTER\_SET\_FORWARDED\_HEADERS** environment variable of the Ingress Controller.

##### Procedure

1. Configure the **HTTPHeaders** field for the Ingress Controller.
  - a. Use the following command to edit the **IngressController** resource:
 

```
$ oc edit IngressController
```
  - b. Under **spec**, set the **HTTPHeaders** policy field to **Append**, **Replace**, **IfNone**, or **Never**:
 

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    forwardedHeaderPolicy: Append
```

##### Example use cases

As a cluster administrator, you can:

- Configure an external proxy that injects the **X-Forwarded-For** header into each request before forwarding it to an Ingress Controller.  
To configure the Ingress Controller to pass the header through unmodified, you specify the **never** policy. The Ingress Controller then never sets the headers, and applications receive only the headers that the external proxy provides.
- Configure the Ingress Controller to pass the **X-Forwarded-For** header that your external proxy sets on external cluster requests through unmodified.  
To configure the Ingress Controller to set the **X-Forwarded-For** header on internal cluster requests, which do not go through the external proxy, specify the **if-none** policy. If an HTTP request already has the header set through the external proxy, then the Ingress Controller preserves it. If the header is absent because the request did not come through the proxy, then the Ingress Controller adds the header.

As an application developer, you can:

- Configure an application-specific external proxy that injects the **X-Forwarded-For** header.  
To configure an Ingress Controller to pass the header through unmodified for an application's Route, without affecting the policy for other Routes, add an annotation **haproxy.router.openshift.io/set-forwarded-headers: if-none** or **haproxy.router.openshift.io/set-forwarded-headers: never** on the Route for the application.



#### NOTE

You can set the **haproxy.router.openshift.io/set-forwarded-headers** annotation on a per route basis, independent from the globally set value for the Ingress Controller.

#### 4.8.9.16. Enable or disable HTTP/2 on Ingress Controllers

You can enable or disable transparent end-to-end HTTP/2 connectivity in HAProxy. Application owners can use HTTP/2 protocol capabilities, including single connection, header compression, binary streams, and more.

You can enable or disable HTTP/2 connectivity for an individual Ingress Controller or for the entire cluster.



#### NOTE

If you enable or disable HTTP/2 connectivity for an individual Ingress Controller and for the entire cluster, the HTTP/2 configuration for the Ingress Controller takes precedence over the HTTP/2 configuration for the cluster.

To enable the use of HTTP/2 for a connection from the client to an HAProxy instance, a route must specify a custom certificate. A route that uses the default certificate cannot use HTTP/2. This restriction is necessary to avoid problems from connection coalescing, where the client re-uses a connection for different routes that use the same certificate.

Consider the following use cases for an HTTP/2 connection for each route type:

- For a re-encrypt route, the connection from HAProxy to the application pod can use HTTP/2 if the application supports using Application-Level Protocol Negotiation (ALPN) to negotiate HTTP/2 with HAProxy. You cannot use HTTP/2 with a re-encrypt route unless the Ingress Controller has HTTP/2 enabled.
- For a passthrough route, the connection can use HTTP/2 if the application supports using ALPN to negotiate HTTP/2 with the client. You can use HTTP/2 with a passthrough route if the Ingress Controller has HTTP/2 enabled or disabled.
- For an edge-terminated secure route, the connection uses HTTP/2 if the service specifies only **appProtocol: kubernetes.io/h2c**. You can use HTTP/2 with an edge-terminated secure route if the Ingress Controller has HTTP/2 enabled or disabled.
- For an insecure route, the connection uses HTTP/2 if the service specifies only **appProtocol: kubernetes.io/h2c**. You can use HTTP/2 with an insecure route if the Ingress Controller has HTTP/2 enabled or disabled.



## IMPORTANT

For non-passthrough routes, the Ingress Controller negotiates its connection to the application independently of the connection from the client. This means a client might connect to the Ingress Controller and negotiate HTTP/1.1. The Ingress Controller might then connect to the application, negotiate HTTP/2, and forward the request from the client HTTP/1.1 connection by using the HTTP/2 connection to the application.

This sequence of events causes an issue if the client subsequently tries to upgrade its connection from HTTP/1.1 to the WebSocket protocol. Consider that if you have an application that is intending to accept WebSocket connections, and the application attempts to allow for HTTP/2 protocol negotiation, the client fails any attempt to upgrade to the WebSocket protocol.

### 4.8.9.16.1. Enabling HTTP/2

You can enable HTTP/2 on a specific Ingress Controller, or you can enable HTTP/2 for the entire cluster.

#### Procedure

- To enable HTTP/2 on a specific Ingress Controller, enter the **oc annotate** command:

```
$ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name> ingress.operator.openshift.io/default-enable-http2=true 1
```

- Replace **<ingresscontroller\_name>** with the name of an Ingress Controller to enable HTTP/2.

- To enable HTTP/2 for the entire cluster, enter the **oc annotate** command:

```
$ oc annotate ingresses.config/cluster ingress.operator.openshift.io/default-enable-http2=true
```

#### TIP

Alternatively, you can apply the following YAML code to enable HTTP/2:

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
  annotations:
    ingress.operator.openshift.io/default-enable-http2: "true"
```

### 4.8.9.16.2. Disabling HTTP/2

You can disable HTTP/2 on a specific Ingress Controller, or you can disable HTTP/2 for the entire cluster.

#### Procedure

- To disable HTTP/2 on a specific Ingress Controller, enter the **oc annotate** command:

```
$ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name>
ingress.operator.openshift.io/default-enable-http2=false ①
```

- ① Replace <ingresscontroller\_name> with the name of an Ingress Controller to disable HTTP/2.

- To disable HTTP/2 for the entire cluster, enter the **oc annotate** command:

```
$ oc annotate ingresses.config/cluster ingress.operator.openshift.io/default-enable-
http2=false
```

## TIP

Alternatively, you can apply the following YAML code to disable HTTP/2:

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
  annotations:
    ingress.operator.openshift.io/default-enable-http2: "false"
```

### 4.8.9.17. Configuring the PROXY protocol for an Ingress Controller

A cluster administrator can configure [the PROXY protocol](#) when an Ingress Controller uses either the **HostNetwork**, **NodePortService**, or **Private** endpoint publishing strategy types. The PROXY protocol enables the load balancer to preserve the original client addresses for connections that the Ingress Controller receives. The original client addresses are useful for logging, filtering, and injecting HTTP headers. In the default configuration, the connections that the Ingress Controller receives only contain the source address that is associated with the load balancer.



#### WARNING

The default Ingress Controller with installer-provisioned clusters on non-cloud platforms that use a Keepalived Ingress Virtual IP (VIP) do not support the PROXY protocol.

The PROXY protocol enables the load balancer to preserve the original client addresses for connections that the Ingress Controller receives. The original client addresses are useful for logging, filtering, and injecting HTTP headers. In the default configuration, the connections that the Ingress Controller receives contain only the source IP address that is associated with the load balancer.



## IMPORTANT

For a passthrough route configuration, servers in OpenShift Container Platform clusters cannot observe the original client source IP address. If you need to know the original client source IP address, configure Ingress access logging for your Ingress Controller so that you can view the client source IP addresses.

For re-encrypt and edge routes, the OpenShift Container Platform router sets the **Forwarded** and **X-Forwarded-For** headers so that application workloads check the client source IP address.

For more information about Ingress access logging, see "Configuring Ingress access logging".

Configuring the PROXY protocol for an Ingress Controller is not supported when using the **LoadBalancerService** endpoint publishing strategy type. This restriction is because when OpenShift Container Platform runs in a cloud platform, and an Ingress Controller specifies that a service load balancer should be used, the Ingress Operator configures the load balancer service and enables the PROXY protocol based on the platform requirement for preserving source addresses.



## IMPORTANT

You must configure both OpenShift Container Platform and the external load balancer to use either the PROXY protocol or TCP.

This feature is not supported in cloud deployments. This restriction is because when OpenShift Container Platform runs in a cloud platform, and an Ingress Controller specifies that a service load balancer should be used, the Ingress Operator configures the load balancer service and enables the PROXY protocol based on the platform requirement for preserving source addresses.



## IMPORTANT

You must configure both OpenShift Container Platform and the external load balancer to either use the PROXY protocol or to use Transmission Control Protocol (TCP).

## Prerequisites

- You created an Ingress Controller.

## Procedure

1. Edit the Ingress Controller resource by entering the following command in your CLI:

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. Set the PROXY configuration:

- If your Ingress Controller uses the **HostNetwork** endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.hostNetwork.protocol** subfield to **PROXY**:

### Sample hostNetwork configuration to PROXY

```
# ...
spec:
```

```

endpointPublishingStrategy:
  hostNetwork:
    protocol: PROXY
    type: HostNetwork
  # ...

```

- If your Ingress Controller uses the **NodePortService** endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.nodePort.protocol** subfield to **PROXY**:

### Sample nodePort configuration to PROXY

```

# ...
spec:
  endpointPublishingStrategy:
    nodePort:
      protocol: PROXY
      type: NodePortService
  # ...

```

- If your Ingress Controller uses the **Private** endpoint publishing strategy type, set the **spec.endpointPublishingStrategy.private.protocol** subfield to **PROXY**:

### Sample private configuration to PROXY

```

# ...
spec:
  endpointPublishingStrategy:
    private:
      protocol: PROXY
      type: Private
  # ...

```

## Additional resources

- [Configuring Ingress access logging](#)

### 4.8.9.18. Specifying an alternative cluster domain using the `appsDomain` option

As a cluster administrator, you can specify an alternative to the default cluster domain for user-created routes by configuring the **appsDomain** field. The **appsDomain** field is an optional domain for OpenShift Container Platform to use instead of the default, which is specified in the **domain** field. If you specify an alternative domain, it overrides the default cluster domain for the purpose of determining the default host for a new route.

For example, you can use the DNS domain for your company as the default domain for routes and ingresses for applications running on your cluster.

## Prerequisites

- You deployed an OpenShift Container Platform cluster.
- You installed the **oc** command-line interface.

## Procedure

- Configure the **appsDomain** field by specifying an alternative default domain for user-created routes.
  - Edit the ingress **cluster** resource:

```
$ oc edit ingresses.config/cluster -o yaml
```

- Edit the YAML file:

#### Sample appsDomain configuration to test.example.com

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.example.com    1
  appsDomain: <test.example.com> 2
```

- 1 Specifies the default domain. You cannot modify the default domain after installation.
- 2 Optional: Domain for OpenShift Container Platform infrastructure to use for application routes. Instead of the default prefix, **apps**, you can use an alternative prefix like **test**.

- Verify that an existing route contains the domain name specified in the **appsDomain** field by exposing the route and verifying the route domain change:



#### NOTE

Wait for the **openshift-apiserver** finish rolling updates before exposing the route.

- Expose the route:

```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

#### Example output

| NAME            | HOST/PORT                                     | PATH | SERVICES | PORT |
|-----------------|-----------------------------------------------|------|----------|------|
| TERMINATION     | WILDCARD                                      |      |          |      |
| hello-openshift | hello_openshift-<my_project>.test.example.com |      |          |      |
|                 | 8080-tcp                                      | None |          |      |

#### 4.8.9.19. Converting HTTP header case

HAProxy lowercases HTTP header names by default; for example, changing **Host: xyz.com** to **host: xyz.com**. If legacy applications are sensitive to the capitalization of HTTP header names, use the Ingress Controller **spec.httpHeaders.headerNameCaseAdjustments** API field for a solution to accommodate legacy applications until they can be fixed.



## IMPORTANT

OpenShift Container Platform includes HAProxy 2.8. If you want to update to this version of the web-based load balancer, ensure that you add the **spec.httpHeaders.headerNameCaseAdjustments** section to your cluster's configuration file.

As a cluster administrator, you can convert the HTTP header case by entering the **oc patch** command or by setting the **HeaderNameCaseAdjustments** field in the Ingress Controller YAML file.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

### Procedure

- Capitalize an HTTP header by using the **oc patch** command.
  - Change the HTTP header from **host** to **Host** by running the following command:
 

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --patch='{"spec":{"httpHeaders":{"headerNameCaseAdjustments":["Host"]}}}'
```
  - Create a **Route** resource YAML file so that the annotation can be applied to the application.

#### Example of a route named my-application

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true ①
  name: <application_name>
  namespace: <application_name>
# ...
```

- ① Set **haproxy.router.openshift.io/h1-adjust-case** so that the Ingress Controller can adjust the **host** request header as specified.
- Specify adjustments by configuring the **HeaderNameCaseAdjustments** field in the Ingress Controller YAML configuration file.
    - The following example Ingress Controller YAML file adjusts the **host** header to **Host** for HTTP/1 requests to appropriately annotated routes:

#### Example Ingress Controller YAML

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
```

```

name: default
namespace: openshift-ingress-operator
spec:
  httpHeaders:
    headerNameCaseAdjustments:
      - Host

```

- b. The following example route enables HTTP response header name case adjustments by using the **haproxy.router.openshift.io/h1-adjust-case** annotation:

#### Example route YAML

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true ①
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application

```

- ① Set **haproxy.router.openshift.io/h1-adjust-case** to true.

#### 4.8.9.20. Using router compression

You configure the HAProxy Ingress Controller to specify router compression globally for specific MIME types. You can use the **mimeTypes** variable to define the formats of MIME types to which compression is applied. The types are: application, image, message, multipart, text, video, or a custom type prefaced by "X-". To see the full notation for MIME types and subtypes, see [RFC1341](#).



#### NOTE

Memory allocated for compression can affect the max connections. Additionally, compression of large buffers can cause latency, like heavy regex or long lists of regex.

Not all MIME types benefit from compression, but HAProxy still uses resources to try to compress if instructed to. Generally, text formats, such as html, css, and js, formats benefit from compression, but formats that are already compressed, such as image, audio, and video, benefit little in exchange for the time and resources spent on compression.

#### Procedure

- Configure the **httpCompression** field for the Ingress Controller.
  - Use the following command to edit the **IngressController** resource:
 

```
$ oc edit -n openshift-ingress-operator ingresscontrollers/default
```
  - Under **spec**, set the **httpCompression** policy field to **mimeTypes** and specify a list of MIME types that should have compression applied:

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpCompression:
    mimeTypes:
      - "text/html"
      - "text/css; charset=utf-8"
      - "application/json"
...

```

#### 4.8.9.21. Exposing router metrics

You can expose the HAProxy router metrics by default in Prometheus format on the default stats port, 1936. The external metrics collection and aggregation systems such as Prometheus can access the HAProxy router metrics. You can view the HAProxy router metrics in a browser in the HTML and comma separated values (CSV) format.

#### Prerequisites

- You configured your firewall to access the default stats port, 1936.

#### Procedure

1. Get the router pod name by running the following command:

```
$ oc get pods -n openshift-ingress
```

#### Example output

| NAME                           | READY | STATUS  | RESTARTS | AGE |
|--------------------------------|-------|---------|----------|-----|
| router-default-76bffb66c-46qwp | 1/1   | Running | 0        | 11h |

2. Get the router's username and password, which the router pod stores in the **/var/lib/haproxy/conf/metrics-auth/statsUsername** and **/var/lib/haproxy/conf/metrics-auth/statsPassword** files:

- a. Get the username by running the following command:

```
$ oc rsh <router_pod_name> cat metrics-auth/statsUsername
```

- b. Get the password by running the following command:

```
$ oc rsh <router_pod_name> cat metrics-auth/statsPassword
```

3. Get the router IP and metrics certificates by running the following command:

```
$ oc describe pod <router_pod>
```

4. Get the raw statistics in Prometheus format by running the following command:

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

5. Access the metrics securely by running the following command:

```
$ curl -u user:password https://<router_IP>:<stats_port>/metrics -k
```

6. Access the default stats port, 1936, by running the following command:

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

#### Example 4.1. Example output

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. Launch the stats window by entering the following URL in a browser:

```
http://<user>:<password>@<router_IP>:<stats_port>
```

8. Optional: Get the stats in CSV format by entering the following URL in a browser:

```
http://<user>:<password>@<router_ip>:1936/metrics;csv
```

#### 4.8.9.22. Customizing HAProxy error code response pages

As a cluster administrator, you can specify a custom error code response page for either 503, 404, or both error pages. The HAProxy router serves a 503 error page when the application pod is not running or a 404 error page when the requested URL does not exist. For example, if you customize the 503 error code response page, then the page is served when the application pod is not running, and the default 404 error code HTTP response page is served by the HAProxy router for an incorrect route or a non-existing route.

Custom error code response pages are specified in a config map then patched to the Ingress Controller. The config map keys have two available file names as follows: **error-page-503.http** and **error-page-404.http**.

Custom HTTP error code response pages must follow the [HAProxy HTTP error page configuration guidelines](#). Here is an example of the default OpenShift Container Platform HAProxy router [http 503 error code response page](#). You can use the default content as a template for creating your own custom page.

By default, the HAProxy router serves only a 503 error page when the application is not running or when the route is incorrect or non-existent. This default behavior is the same as the behavior on OpenShift Container Platform 4.8 and earlier. If a config map for the customization of an HTTP error code response is not provided, and you are using a custom HTTP error code response page, the router serves a default 404 or 503 error code response page.



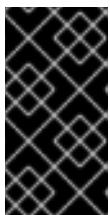
#### NOTE

If you use the OpenShift Container Platform default 503 error code page as a template for your customizations, the headers in the file require an editor that can use CRLF line endings.

#### Procedure

1. Create a config map named **my-custom-error-code-pages** in the **openshift-config** namespace:

```
$ oc -n openshift-config create configmap my-custom-error-code-pages \
--from-file=error-page-503.http \
--from-file=error-page-404.http
```



#### IMPORTANT

If you do not specify the correct format for the custom error code response page, a router pod outage occurs. To resolve this outage, you must delete or correct the config map and delete the affected router pods so they can be recreated with the correct information.

2. Patch the Ingress Controller to reference the **my-custom-error-code-pages** config map by name:

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec": {"httpErrorCodePages": {"name": "my-custom-error-code-pages"} }}' --type=merge
```

The Ingress Operator copies the **my-custom-error-code-pages** config map from the

**openshift-config** namespace to the **openshift-ingress** namespace. The Operator names the config map according to the pattern, **<your\_ingresscontroller\_name>-errorpages**, in the **openshift-ingress** namespace.

3. Display the copy:

```
$ oc get cm default-errorpages -n openshift-ingress
```

### Example output

| NAME               | DATA | AGE   |
|--------------------|------|-------|
| default-errorpages | 2    | 25s ① |

- ① The example config map name is **default-errorpages** because the **default** Ingress Controller custom resource (CR) was patched.

4. Confirm that the config map containing the custom error response page mounts on the router volume where the config map key is the filename that has the custom HTTP error code response:

- For 503 custom HTTP custom error code response:

```
$ oc -n openshift-ingress rsh <router_pod> cat  
/var/lib/haproxy/conf/error_code_pages/error-page-503.http
```

- For 404 custom HTTP custom error code response:

```
$ oc -n openshift-ingress rsh <router_pod> cat  
/var/lib/haproxy/conf/error_code_pages/error-page-404.http
```

## Verification

Verify your custom error code HTTP response:

1. Create a test project and application:

```
$ oc new-project test-ingress
```

```
$ oc new-app django-psql-example
```

2. For 503 custom http error code response:

- a. Stop all the pods for the application.

- b. Run the following curl command or visit the route hostname in the browser:

```
$ curl -vk <route_hostname>
```

3. For 404 custom http error code response:

- a. Visit a non-existent route or an incorrect route.

- b. Run the following curl command or visit the route hostname in the browser:

```
$ curl -vk <route_hostname>
```

- Check if the **errorfile** attribute is properly in the **haproxy.config** file:

```
$ oc -n openshift-ingress rsh <router> cat /var/lib/haproxy/conf/haproxy.config | grep errorfile
```

#### 4.8.9.23. Setting the Ingress Controller maximum connections

A cluster administrator can set the maximum number of simultaneous connections for OpenShift router deployments. You can patch an existing Ingress Controller to increase the maximum number of connections.

##### Prerequisites

- The following assumes that you already created an Ingress Controller

##### Procedure

- Update the Ingress Controller to change the maximum number of connections for HAProxy:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec": {"tuningOptions": {"maxConnections": 7500}}}'
```



##### WARNING

If you set the **spec.tuningOptions.maxConnections** value greater than the current operating system limit, the HAProxy process will not start. See the table in the "Ingress Controller configuration parameters" section for more information about this parameter.

#### 4.8.10. Additional resources

- [Configuring a custom PKI](#)

## 4.9. INGRESS NODE FIREWALL OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The Ingress Node Firewall Operator provides a stateless, eBPF-based firewall for managing node-level ingress traffic in OpenShift Container Platform.

### 4.9.1. Ingress Node Firewall Operator

The Ingress Node Firewall Operator provides ingress firewall rules at a node level by deploying the daemon set to nodes you specify and manage in the firewall configurations. To deploy the daemon set, you create an **IngressNodeFirewallConfig** custom resource (CR). The Operator applies the **IngressNodeFirewallConfig** CR to create ingress node firewall daemon set **daemon**, which run on all nodes that match the **nodeSelector**.

You configure **rules** of the **IngressNodeFirewall** CR and apply them to clusters using the **nodeSelector** and setting values to "true".



## IMPORTANT

The Ingress Node Firewall Operator supports only stateless firewall rules.

Network interface controllers (NICs) that do not support native XDP drivers will run at a lower performance.

For OpenShift Container Platform 4.14 or later, you must run Ingress Node Firewall Operator on RHEL 9.0 or later.

### 4.9.2. Installing the Ingress Node Firewall Operator

As a cluster administrator, you can install the Ingress Node Firewall Operator by using the OpenShift Container Platform CLI or the web console.

#### 4.9.2.1. Installing the Ingress Node Firewall Operator using the CLI

As a cluster administrator, you can install the Operator using the CLI.

##### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

##### Procedure

1. To create the **openshift-ingress-node-firewall** namespace, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: openshift-ingress-node-firewall
EOF
```

2. To create an **OperatorGroup** CR, enter the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ingress-node-firewall-operators
  namespace: openshift-ingress-node-firewall
EOF
```

3. Subscribe to the Ingress Node Firewall Operator.

- a. To create a **Subscription** CR for the Ingress Node Firewall Operator, enter the following

command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ingress-node-firewall-sub
  namespace: openshift-ingress-node-firewall
spec:
  name: ingress-node-firewall
  channel: stable
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

- To verify that the Operator is installed, enter the following command:

```
$ oc get ip -n openshift-ingress-node-firewall
```

#### Example output

| NAME          | CSV                                       | APPROVAL  | APPROVED |
|---------------|-------------------------------------------|-----------|----------|
| install-5cvnz | ingress-node-firewall.4.18.0-202211122336 | Automatic | true     |

- To verify the version of the Operator, enter the following command:

```
$ oc get csv -n openshift-ingress-node-firewall
```

#### Example output

| NAME                                      | DISPLAY                        | VERSION             | REPLACES                                            |
|-------------------------------------------|--------------------------------|---------------------|-----------------------------------------------------|
| PHASE                                     |                                |                     |                                                     |
| ingress-node-firewall.4.18.0-202211122336 | Ingress Node Firewall Operator | 4.18.0-202211122336 | ingress-node-firewall.4.18.0-202211102047 Succeeded |

### 4.9.2.2. Installing the Ingress Node Firewall Operator using the web console

As a cluster administrator, you can install the Operator using the web console.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.

#### Procedure

- Install the Ingress Node Firewall Operator:
  - In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
  - Select **Ingress Node Firewall Operator** from the list of available Operators, and then click **Install**.

- c. On the **Install Operator** page, under **Installed Namespace**, select **Operator recommended Namespace**.
  - d. Click **Install**.
2. Verify that the Ingress Node Firewall Operator is installed successfully:
    - a. Navigate to the **Operators → Installed Operators** page.
    - b. Ensure that **Ingress Node Firewall Operator** is listed in the **openshift-ingress-node-firewall** project with a **Status** of **InstallSucceeded**.



#### NOTE

During installation an Operator might display a **Failed** status. If the installation later succeeds with an **InstallSucceeded** message, you can ignore the **Failed** message.

If the Operator does not have a **Status** of **InstallSucceeded**, troubleshoot using the following steps:

- Inspect the **Operator Subscriptions** and **Install Plans** tabs for any failures or errors under **Status**.
- Navigate to the **Workloads → Pods** page and check the logs for pods in the **openshift-ingress-node-firewall** project.
- Check the namespace of the YAML file. If the annotation is missing, you can add the annotation **workload.openshift.io/allowed=management** to the Operator namespace with the following command:

```
$ oc annotate ns/openshift-ingress-node-firewall
workload.openshift.io/allowed=management
```



#### NOTE

For single-node OpenShift clusters, the **openshift-ingress-node-firewall** namespace requires the **workload.openshift.io/allowed=management** annotation.

### 4.9.3. Deploying Ingress Node Firewall Operator

#### Prerequisite

- The Ingress Node Firewall Operator is installed.

#### Procedure

To deploy the Ingress Node Firewall Operator, create a **IngressNodeFirewallConfig** custom resource that will deploy the Operator's daemon set. You can deploy one or multiple **IngressNodeFirewall** CRDs to nodes by applying firewall rules.

1. Create the **IngressNodeFirewallConfig** inside the **openshift-ingress-node-firewall** namespace named **ingressnodefirewallconfig**.

- Run the following command to deploy Ingress Node Firewall Operator rules:

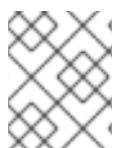
```
$ oc apply -f rule.yaml
```

#### 4.9.3.1. Ingress Node Firewall configuration object

The fields for the Ingress Node Firewall configuration object are described in the following table:

**Table 4.13. Ingress Node Firewall Configuration object**

| Field                              | Type           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>metadata.name</b>               | <b>string</b>  | The name of the CR object. The name of the firewall rules object must be <b>ingressnodefirewallconfig</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>metadata.namespace</b>          | <b>string</b>  | Namespace for the Ingress Firewall Operator CR object. The <b>IngressNodeFirewallConfig</b> CR must be created inside the <b>openshift-ingress-node-firewall</b> namespace.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>spec.nodeSelector</b>           | <b>string</b>  | A node selection constraint used to target nodes through specified node labels. For example:<br><br> <pre>spec:   nodeSelector:     node-role.kubernetes.io/worker: ""</pre><br> <b>NOTE</b><br>One label used in <b>nodeSelector</b> must match a label on the nodes in order for the daemon set to start. For example, if the node labels <b>node-role.kubernetes.io/worker</b> and <b>node-type.kubernetes.io/vm</b> are applied to a node, then at least one label must be set using <b>nodeSelector</b> for the daemon set to start. |
| <b>spec.ebpfProgramManagerMode</b> | <b>boolean</b> | Specifies if the Node Ingress Firewall Operator uses the eBPF Manager Operator or not to manage eBPF programs. This capability is a Technology Preview feature.<br><br>For more information about the support scope of Red Hat Technology Preview features, see <a href="#">Technology Preview Features Support Scope</a> .                                                                                                                                                                                                                                                                                                                                                                                    |



#### NOTE

The Operator consumes the CR and creates an ingress node firewall daemon set on all the nodes that match the **nodeSelector**.

#### Ingress Node Firewall Operator example configuration

A complete Ingress Node Firewall Configuration is specified in the following example:

## Example Ingress Node Firewall Configuration object

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```



### NOTE

The Operator consumes the CR and creates an ingress node firewall daemon set on all the nodes that match the **nodeSelector**.

### 4.9.3.2. Ingress Node Firewall rules object

The fields for the Ingress Node Firewall rules object are described in the following table:

**Table 4.14. Ingress Node Firewall rules object**

| Field                | Type          | Description                                                                                                                                                                  |
|----------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>metadata.name</b> | <b>string</b> | The name of the CR object.                                                                                                                                                   |
| <b>interfaces</b>    | <b>array</b>  | The fields for this object specify the interfaces to apply the firewall rules to. For example, <b>- en0</b> and <b>- en1</b> .                                               |
| <b>nodeSelector</b>  | <b>array</b>  | You can use <b>nodeSelector</b> to select the nodes to apply the firewall rules to. Set the value of your named <b>nodeSelector</b> labels to <b>true</b> to apply the rule. |
| <b>ingress</b>       | <b>object</b> | <b>ingress</b> allows you to configure the rules that allow outside access to the services on your cluster.                                                                  |

### Ingress object configuration

The values for the **ingress** object are defined in the following table:

**Table 4.15. ingress object**

| Field | Type | Description |
|-------|------|-------------|
|       |      |             |

| Field              | Type         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>sourceCIDRs</b> | <b>array</b> | <p>Allows you to set the CIDR block. You can configure multiple CIDRs from different address families.</p>  <p><b>NOTE</b></p> <p>Different CIDRs allow you to use the same order rule. In the case that there are multiple <b>IngressNodeFirewall</b> objects for the same nodes and interfaces with overlapping CIDRs, the <b>order</b> field will specify which rule is applied first. Rules are applied in ascending order.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>rules</b>       | <b>array</b> | <p>Ingress firewall <b>rules.order</b> objects are ordered starting at <b>1</b> for each <b>source.CIDR</b> with up to 100 rules per CIDR. Lower order rules are executed first.</p> <p><b>rules.protocolConfig.protocol</b> supports the following protocols: TCP, UDP, SCTP, ICMP and ICMPv6. ICMP and ICMPv6 rules can match against ICMP and ICMPv6 types or codes. TCP, UDP, and SCTP rules can match against a single destination port or a range of ports using <b>&lt;start : end-1&gt;</b> format.</p> <p>Set <b>rules.action</b> to <b>allow</b> to apply the rule or <b>deny</b> to disallow the rule.</p>  <p><b>NOTE</b></p> <p>Ingress firewall rules are verified using a verification webhook that blocks any invalid configuration. The verification webhook prevents you from blocking any critical cluster services such as the API server.</p> |

### Ingress Node Firewall rules object example

A complete Ingress Node Firewall configuration is specified in the following example:

#### Example Ingress Node Firewall configuration

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
```

```

name: ingressnodefirewall
spec:
  interfaces:
    - eth0
  nodeSelector:
    matchLabels:
      <ingress_firewall_label_name>: <label_value> ①
  ingress:
    - sourceCIDRs:
        - 172.16.0.0/12
      rules:
        - order: 10
          protocolConfig:
            protocol: ICMP
            icmp:
              icmpType: 8 #ICMP Echo request
            action: Deny
        - order: 20
          protocolConfig:
            protocol: TCP
            tcp:
              ports: "8000-9000"
            action: Deny
    - sourceCIDRs:
        - fc00:f853:ccd:e793::0/64
      rules:
        - order: 10
          protocolConfig:
            protocol: ICMPv6
            icmpv6:
              icmpType: 128 #ICMPV6 Echo request
            action: Deny

```

- ① A <label\_name> and a <label\_value> must exist on the node and must match the **nodeSelector** label and value applied to the nodes you want the **ingressfirewallconfig** CR to run on. The <label\_value> can be **true** or **false**. By using **nodeSelector** labels, you can target separate groups of nodes to apply different rules to using the **ingressfirewallconfig** CR.

### Zero trust Ingress Node Firewall rules object example

Zero trust Ingress Node Firewall rules can provide additional security to multi-interface clusters. For example, you can use zero trust Ingress Node Firewall rules to drop all traffic on a specific interface except for SSH.

A complete configuration of a zero trust Ingress Node Firewall rule set is specified in the following example:



#### IMPORTANT

Users need to add all ports their application will use to their allowlist in the following case to ensure proper functionality.

### Example zero trust Ingress Node Firewall rules

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
```

```

kind: IngressNodeFirewall
metadata:
  name: ingressnodefirewall-zero-trust
spec:
  interfaces:
    - eth1 1
  nodeSelector:
    matchLabels:
      <ingress_firewall_label_name>: <label_value> 2
  ingress:
    - sourceCIDRs:
        - 0.0.0.0/0 3
    rules:
      - order: 10
        protocolConfig:
          protocol: TCP
          tcp:
            ports: 22
            action: Allow
      - order: 20
        action: Deny 4

```

**1** Network-interface cluster

**2** The <label\_name> and <label\_value> needs to match the **nodeSelector** label and value applied to the specific nodes with which you wish to apply the **ingressfirewallconfig** CR.

**3** **0.0.0.0/0** set to match any CIDR

**4** **action** set to **Deny**



## IMPORTANT

eBPF Manager Operator integration 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](#).

### 4.9.4. Ingress Node Firewall Operator integration

The Ingress Node Firewall uses [eBPF](#) programs to implement some of its key firewall functionality. By default these eBPF programs are loaded into the kernel using a mechanism specific to the Ingress Node Firewall. You can configure the Ingress Node Firewall Operator to use the eBPF Manager Operator for loading and managing these programs instead.

When this integration is enabled, the following limitations apply:

- The Ingress Node Firewall Operator uses TCX if XDP is not available and TCX is incompatible with bpfman.

- The Ingress Node Firewall Operator daemon set pods remain in the **ContainerCreating** state until the firewall rules are applied.
- The Ingress Node Firewall Operator daemon set pods run as privileged.

#### 4.9.5. Configuring Ingress Node Firewall Operator to use the eBPF Manager Operator

The Ingress Node Firewall uses [eBPF](#) programs to implement some of its key firewall functionality. By default these eBPF programs are loaded into the kernel using a mechanism specific to the Ingress Node Firewall.

As a cluster administrator, you can configure the Ingress Node Firewall Operator to use the eBPF Manager Operator for loading and managing these programs instead, adding additional security and observability functionality.

##### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have an account with administrator privileges.
- You installed the Ingress Node Firewall Operator.
- You have installed the eBPF Manager Operator.

##### Procedure

1. Apply the following labels to the **ingress-node-firewall-system** namespace:

```
$ oc label namespace openshift-ingress-node-firewall \
  pod-security.kubernetes.io/enforce=privileged \
  pod-security.kubernetes.io/warn=privileged --overwrite
```

2. Edit the **IngressNodeFirewallConfig** object named **ingressnodefirewallconfig** and set the **ebpfProgramManagerMode** field:

##### Ingress Node Firewall Operator configuration object

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  ebpfProgramManagerMode: <ebpf_mode>
```

where:

**<ebpf\_mode>**: Specifies whether or not the Ingress Node Firewall Operator uses the eBPF Manager Operator to manage eBPF programs. Must be either **true** or **false**. If unset, eBPF Manager is not used.

#### 4.9.6. Viewing Ingress Node Firewall Operator rules

##### Procedure

- Run the following command to view all current rules :

```
$ oc get ingressnodefirewall
```

- Choose one of the returned <resource> names and run the following command to view the rules or configs:

```
$ oc get <resource> <name> -o yaml
```

#### 4.9.7. Troubleshooting the Ingress Node Firewall Operator

- Run the following command to list installed Ingress Node Firewall custom resource definitions (CRD):

```
$ oc get crds | grep ingressnodefirewall
```

##### Example output

| NAME                                                             | READY                | UP-TO-DATE | AVAILABLE | AGE |
|------------------------------------------------------------------|----------------------|------------|-----------|-----|
| ingressnodefirewallconfigs.ingressnodefirewall.openshift.io      | 2022-08-25T10:03:01Z |            |           |     |
| ingressnodefirewallnodedestates.ingressnodefirewall.openshift.io | 2022-08-25T10:03:00Z |            |           |     |
| ingressnodefirewalls.ingressnodefirewall.openshift.io            | 2022-08-25T10:03:00Z |            |           |     |

- Run the following command to view the state of the Ingress Node Firewall Operator:

```
$ oc get pods -n openshift-ingress-node-firewall
```

##### Example output

| NAME                                     | READY | STATUS  | RESTARTS | AGE   |
|------------------------------------------|-------|---------|----------|-------|
| ingress-node-firewall-controller-manager | 2/2   | Running | 0        | 5d21h |
| ingress-node-firewall-daemon-pqx56       | 3/3   | Running | 0        | 5d21h |

The following fields provide information about the status of the Operator: **READY**, **STATUS**, **AGE**, and **RESTARTS**. The **STATUS** field is **Running** when the Ingress Node Firewall Operator is deploying a daemon set to the assigned nodes.

- Run the following command to collect all ingress firewall node pods' logs:

```
$ oc adm must-gather – gather_ingress_node_firewall
```

The logs are available in the sos node's report containing eBPF **bptool** outputs at **/sos\_commands/ebpf**. These reports include lookup tables used or updated as the ingress firewall XDP handles packet processing, updates statistics, and emits events.

#### 4.9.8. Additional resources

- [About the eBPF Manager Operator](#)

## 4.10. SR-IOV OPERATOR

### 4.10.1. Installing the SR-IOV Network Operator

You can install the Single Root I/O Virtualization (SR-IOV) Network Operator on your cluster to manage SR-IOV network devices and network attachments.

#### 4.10.1.1. Installing the SR-IOV Network Operator

As a cluster administrator, you can install the Single Root I/O Virtualization (SR-IOV) Network Operator by using the OpenShift Container Platform CLI or the web console.

##### 4.10.1.1.1. CLI: Installing the SR-IOV Network Operator

As a cluster administrator, you can install the Operator using the CLI.

#### Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports SR-IOV.
- Install the OpenShift CLI (**oc**).
- An account with **cluster-admin** privileges.

#### Procedure

1. Create the **openshift-sriov-network-operator** namespace by entering the following command:

```
$ cat << EOF| oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
EOF
```

2. Create an **OperatorGroup** custom resource (CR) by entering the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
    - openshift-sriov-network-operator
EOF
```

3. Create a **Subscription** CR for the SR-IOV Network Operator by entering the following command:

```
$ cat << EOF| oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subscription
  namespace: openshift-sriov-network-operator
spec:
  channel: stable
  name: sriov-network-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

4. Create an **SriovoperatorConfig** resource by entering the following command:

```
$ cat <<EOF | oc create -f -
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: true
  enableOperatorWebhook: true
  logLevel: 2
  disableDrain: false
EOF
```

## Verification

- Check that the Operator is installed by entering the following command:

```
$ oc get csv -n openshift-sriov-network-operator \
-o custom-columns=Name:.metadata.name,Phase:.status.phase
```

### Example output

| Name                                       | Phase     |
|--------------------------------------------|-----------|
| sriov-network-operator.4.18.0-202406131906 | Succeeded |

#### 4.10.1.1.2. Web console: Installing the SR-IOV Network Operator

As a cluster administrator, you can install the Operator using the web console.

### Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports SR-IOV.
- Install the OpenShift CLI (**oc**).
- An account with **cluster-admin** privileges.

## Procedure

1. Install the SR-IOV Network Operator:
  - a. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
  - b. Select **SR-IOV Network Operator** from the list of available Operators, and then click **Install**.
  - c. On the **Install Operator** page, under **Installed Namespace**, select **Operator recommended Namespace**.
  - d. Click **Install**.
2. Verify that the SR-IOV Network Operator is installed successfully:
  - a. Navigate to the **Operators** → **Installed Operators** page.
  - b. Ensure that **SR-IOV Network Operator** is listed in the **openshift-sriov-network-operator** project with a **Status** of **InstallSucceeded**.



### NOTE

During installation an Operator might display a **Failed** status. If the installation later succeeds with an **InstallSucceeded** message, you can ignore the **Failed** message.

If the Operator does not appear as installed, to troubleshoot further:

- Inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.
- Navigate to the **Workloads** → **Pods** page and check the logs for pods in the **openshift-sriov-network-operator** project.
- Check the namespace of the YAML file. If the annotation is missing, you can add the annotation **workload.openshift.io/allowed=management** to the Operator namespace with the following command:

```
$ oc annotate ns/openshift-sriov-network-operator  
workload.openshift.io/allowed=management
```



### NOTE

For single-node OpenShift clusters, the annotation **workload.openshift.io/allowed=management** is required for the namespace.

#### 4.10.1.2. Next steps

- [Configuring the SR-IOV Network Operator](#)

#### 4.10.2. Configuring the SR-IOV Network Operator

The Single Root I/O Virtualization (SR-IOV) Network Operator manages the SR-IOV network devices and network attachments in your cluster.

#### 4.10.2.1. Configuring the SR-IOV Network Operator

- Create a **SriovOperatorConfig** custom resource (CR) to deploy all the SR-IOV Operator components:

- a. Create a file named **sriovOperatorConfig.yaml** using the following YAML:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default 1
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: false
  enableInjector: true 2
  enableOperatorWebhook: true 3
  logLevel: 2
  featureGates:
    metricsExporter: false
```

- 1** The only valid name for the **SriovOperatorConfig** resource is **default** and it must be in the namespace where the Operator is deployed.
- 2** The **enableInjector** field, if not specified in the CR or explicitly set to **true**, defaults to **false** or **<none>**, preventing any **network-resources-injector** pod from running in the namespace. The recommended setting is **true**.
- 3** The **enableOperatorWebhook** field, if not specified in the CR or explicitly set to true, defaults to **false** or **<none>**, preventing any **operator-webhook** pod from running in the namespace. The recommended setting is **true**.

- b. Create the resource by running the following command:

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

##### 4.10.2.1.1. SR-IOV Network Operator config custom resource

The fields for the **sriovoperatorconfig** custom resource are described in the following table:

Table 4.16. SR-IOV Network Operator config custom resource

| Field                      | Type          | Description                                                                                                                                                   |
|----------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>metadata.name</b>       | <b>string</b> | Specifies the name of the SR-IOV Network Operator instance. The default value is <b>default</b> . Do not set a different value.                               |
| <b>metadata.name space</b> | <b>string</b> | Specifies the namespace of the SR-IOV Network Operator instance. The default value is <b>openshift-sriov-network-operator</b> . Do not set a different value. |

| Field                                    | Type                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------------|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>spec.configDaemonNodeSelector</b>     | <b>string</b>          | Specifies the node selection to control scheduling the SR-IOV Network Config Daemon on selected nodes. By default, this field is not set and the Operator deploys the SR-IOV Network Config daemon set on worker nodes.                                                                                                                                                                                                                                                    |
| <b>spec.disableDrain</b>                 | <b>boolean</b>         | <p>Specifies whether to disable the node draining process or enable the node draining process when you apply a new policy to configure the NIC on a node. Setting this field to <b>true</b> facilitates software development and installing OpenShift Container Platform on a single node. By default, this field is not set.</p> <p>For single-node clusters, set this field to <b>true</b> after installing the Operator. This field must remain set to <b>true</b>.</p> |
| <b>spec.enableInjector</b>               | <b>boolean</b>         | Specifies whether to enable or disable the Network Resources Injector daemon set.                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>spec.enableOperatorWebhook</b>        | <b>boolean</b>         | Specifies whether to enable or disable the Operator Admission Controller webhook daemon set.                                                                                                                                                                                                                                                                                                                                                                               |
| <b>spec.logLevel</b>                     | <b>integer</b>         | Specifies the log verbosity level of the Operator. By default, this field is set to <b>0</b> , which shows only basic logs. Set to <b>2</b> to show all the available logs.                                                                                                                                                                                                                                                                                                |
| <b>spec.featureGates</b>                 | <b>map[string]bool</b> | Specifies whether to enable or disable the optional features. For example, <b>metricsExporter</b> .                                                                                                                                                                                                                                                                                                                                                                        |
| <b>spec.featureGates.metricsExporter</b> | <b>boolean</b>         | Specifies whether to enable or disable the SR-IOV Network Operator metrics. By default, this field is set to <b>false</b> .                                                                                                                                                                                                                                                                                                                                                |

| Field                                                | Type                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------------------------------------------|----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>spec.featureGates.mellanoxFirmwareReset</code> | <code>boolean</code> | <p>Specifies whether to reset the firmware on virtual function (VF) changes in the SR-IOV Network Operator. Some chipsets, such as the Intel C740 Series, do not completely power off the PCI-E devices, which is required to configure VFs on NVIDIA/Mellanox NICs. By default, this field is set to <b>false</b>.</p> <p><b>IMPORTANT</b></p> <p>The <code>spec.featureGates.mellanoxFirmwareReset</code> parameter 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.</p> <p>For more information about the support scope of Red Hat Technology Preview features, see <a href="#">Technology Preview Features Support Scope</a></p> |

#### 4.10.2.1.2. About the Network Resources Injector

The Network Resources Injector is a Kubernetes Dynamic Admission Controller application. It provides the following capabilities:

- Mutation of resource requests and limits in a pod specification to add an SR-IOV resource name according to an SR-IOV network attachment definition annotation.
- Mutation of a pod specification with a Downward API volume to expose pod annotations, labels, and huge pages requests and limits. Containers that run in the pod can access the exposed information as files under the `/etc/podnetinfo` path.

The Network Resources Injector is enabled by the SR-IOV Network Operator when the `enableInjector` is set to `true` in the `SriovOperatorConfig` CR. The `network-resources-injector` pod runs as a daemon set on all control plane nodes. The following is an example of Network Resources Injector pods running in a cluster with three control plane nodes:

```
$ oc get pods -n openshift-sriov-network-operator
```

#### Example output

| NAME                             | READY | STATUS  | RESTARTS | AGE |
|----------------------------------|-------|---------|----------|-----|
| network-resources-injector-5cz5p | 1/1   | Running | 0        | 10m |
| network-resources-injector-dwqpx | 1/1   | Running | 0        | 10m |
| network-resources-injector-lktz5 | 1/1   | Running | 0        | 10m |

#### 4.10.2.1.3. Disabling or enabling the Network Resources Injector

To disable or enable the Network Resources Injector, complete the following procedure.

## Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- You must have installed the SR-IOV Network Operator.

## Procedure

- Set the **enableInjector** field. Replace **<value>** with **false** to disable the feature or **true** to enable the feature.

```
$ oc patch sriovoperatorconfig default \
--type=merge -n openshift-sriov-network-operator \
--patch '{ "spec": { "enableInjector": <value> } }'
```

### TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioVOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: <value>
```

### 4.10.2.1.4. About the SR-IOV Network Operator admission controller webhook

The SR-IOV Network Operator Admission Controller webhook is a Kubernetes Dynamic Admission Controller application. It provides the following capabilities:

- Validation of the **SriovNetworkNodePolicy** CR when it is created or updated.
- Mutation of the **SriovNetworkNodePolicy** CR by setting the default value for the **priority** and **deviceType** fields when the CR is created or updated.

The SR-IOV Network Operator Admission Controller webhook is enabled by the Operator when the **enableOperatorWebhook** is set to **true** in the **SriovOperatorConfig** CR. The **operator-webhook** pod runs as a daemon set on all control plane nodes.



### NOTE

Use caution when disabling the SR-IOV Network Operator Admission Controller webhook. You can disable the webhook under specific circumstances, such as troubleshooting, or if you want to use unsupported devices. For information about configuring unsupported devices, see [Configuring the SR-IOV Network Operator to use an unsupported NIC](#).

The following is an example of the Operator Admission Controller webhook pods running in a cluster with three control plane nodes:

```
$ oc get pods -n openshift-sriov-network-operator
```

### Example output

| NAME                   | READY | STATUS  | RESTARTS | AGE |
|------------------------|-------|---------|----------|-----|
| operator-webhook-9jkw6 | 1/1   | Running | 0        | 16m |
| operator-webhook-kbr5p | 1/1   | Running | 0        | 16m |
| operator-webhook-rpfrl | 1/1   | Running | 0        | 16m |

#### 4.10.2.1.5. Disabling or enabling the SR-IOV Network Operator admission controller webhook

To disable or enable the admission controller webhook, complete the following procedure.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- You must have installed the SR-IOV Network Operator.

### Procedure

- Set the **enableOperatorWebhook** field. Replace **<value>** with **false** to disable the feature or **true** to enable it:

```
$ oc patch sriovoperatorconfig default --type=merge \
-n openshift-sriov-network-operator \
--patch '{ "spec": { "enableOperatorWebhook": <value> } }'
```

### TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableOperatorWebhook: <value>
```

#### 4.10.2.1.6. About custom node selectors

The SR-IOV Network Config daemon discovers and configures the SR-IOV network devices on cluster nodes. By default, it is deployed to all the **worker** nodes in the cluster. You can use node labels to specify on which nodes the SR-IOV Network Config daemon runs.

#### 4.10.2.1.7. Configuring a custom NodeSelector for the SR-IOV Network Config daemon

The SR-IOV Network Config daemon discovers and configures the SR-IOV network devices on cluster nodes. By default, it is deployed to all the **worker** nodes in the cluster. You can use node labels to specify on which nodes the SR-IOV Network Config daemon runs.

To specify the nodes where the SR-IOV Network Config daemon is deployed, complete the following procedure.



## IMPORTANT

When you update the **configDaemonNodeSelector** field, the SR-IOV Network Config daemon is recreated on each selected node. While the daemon is recreated, cluster users are unable to apply any new SR-IOV Network node policy or create new SR-IOV pods.

### Procedure

- To update the node selector for the operator, enter the following command:

```
$ oc patch sriovoperatorconfig default --type=json \
-n openshift-sriov-network-operator \
--patch '[{
  "op": "replace",
  "path": "/spec/configDaemonNodeSelector",
  "value": {"<node_label>}}
]'
```

Replace **<node\_label>** with a label to apply as in the following example: **"node-role.kubernetes.io/worker": ""**.

## TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    <node_label>
```

### 4.10.2.1.8. Configuring the SR-IOV Network Operator for single node installations

By default, the SR-IOV Network Operator drains workloads from a node before every policy change. The Operator performs this action to ensure that there no workloads using the virtual functions before the reconfiguration.

For installations on a single node, there are no other nodes to receive the workloads. As a result, the Operator must be configured not to drain the workloads from the single node.



## IMPORTANT

After performing the following procedure to disable draining workloads, you must remove any workload that uses an SR-IOV network interface before you change any SR-IOV network node policy.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- You must have installed the SR-IOV Network Operator.

### Procedure

- To set the **disableDrain** field to **true** and the **configDaemonNodeSelector** field to **node-role.kubernetes.io/master: ""**, enter the following command:

```
$ oc patch sriovoperatorconfig default --type=merge -n openshift-sriov-network-operator --patch '{ "spec": { "disableDrain": true, "configDaemonNodeSelector": { "node-role.kubernetes.io/master": "" } } }'
```

### TIP

You can alternatively apply the following YAML to update the Operator:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrvioOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: true
  configDaemonNodeSelector:
    node-role.kubernetes.io/master: ""
```

### 4.10.2.1.9. Deploying the SR-IOV Operator for hosted control planes

After you configure and deploy your hosting service cluster, you can create a subscription to the SR-IOV Operator on a hosted cluster. The SR-IOV pod runs on worker machines rather than the control plane.

### Prerequisites

You must configure and deploy the hosted cluster on AWS.

### Procedure

1. Create a namespace and an Operator group:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
```

```

---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
    - openshift-sriov-network-operator

```

2. Create a subscription to the SR-IOV Operator:

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subscription
  namespace: openshift-sriov-network-operator
spec:
  channel: stable
  name: sriov-network-operator
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
  source: redhat-operators
  sourceNamespace: openshift-marketplace

```

## Verification

1. To verify that the SR-IOV Operator is ready, run the following command and view the resulting output:

```
$ oc get csv -n openshift-sriov-network-operator
```

### Example output

| NAME                                       | DISPLAY                 | VERSION             | REPLACES                                   |
|--------------------------------------------|-------------------------|---------------------|--------------------------------------------|
| PHASE                                      |                         |                     |                                            |
| sriov-network-operator.4.18.0-202211021237 | SR-IOV Network Operator | 4.18.0-202211021237 | sriov-network-operator.4.18.0-202210290517 |
|                                            |                         |                     | Succeeded                                  |

2. To verify that the SR-IOV pods are deployed, run the following command:

```
$ oc get pods -n openshift-sriov-network-operator
```

### 4.10.2.2. About the SR-IOV network metrics exporter

The Single Root I/O Virtualization (SR-IOV) network metrics exporter reads the metrics for SR-IOV virtual functions (VFs) and exposes these VF metrics in Prometheus format. When the SR-IOV network metrics exporter is enabled, you can query the SR-IOV VF metrics by using the OpenShift Container Platform web console to monitor the networking activity of the SR-IOV pods.

When you query the SR-IOV VF metrics by using the web console, the SR-IOV network metrics exporter fetches and returns the VF network statistics along with the name and namespace of the pod that the VF is attached to.

The SR-IOV VF metrics that the metrics exporter reads and exposes in Prometheus format are described in the following table:

Table 4.17. SR-IOV VF metrics

| Metric                       | Description                                               | Example PromQL query to examine the VF metric                                                            |
|------------------------------|-----------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <b>sriov_vf_rx_bytes</b>     | Received bytes per virtual function.                      | <b>sriov_vf_rx_bytes * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b>     |
| <b>sriov_vf_tx_bytes</b>     | Transmitted bytes per virtual function.                   | <b>sriov_vf_tx_bytes * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b>     |
| <b>sriov_vf_rx_packets</b>   | Received packets per virtual function.                    | <b>sriov_vf_rx_packets * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b>   |
| <b>sriov_vf_tx_packets</b>   | Transmitted packets per virtual function.                 | <b>sriov_vf_tx_packets * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b>   |
| <b>sriov_vf_rx_dropped</b>   | Dropped packets upon receipt per virtual function.        | <b>sriov_vf_rx_dropped * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b>   |
| <b>sriov_vf_tx_dropped</b>   | Dropped packets during transmission per virtual function. | <b>sriov_vf_tx_dropped * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b>   |
| <b>sriov_vf_rx_multicast</b> | Received multicast packets per virtual function.          | <b>sriov_vf_rx_multicast * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b> |

| Metric                       | Description                                      | Example PromQL query to examine the VF metric                                                            |
|------------------------------|--------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <b>sriov_vf_rx_broadcast</b> | Received broadcast packets per virtual function. | <b>sriov_vf_rx_broadcast * on (pciAddr,node) group_left(pod,namespace,d ev_type) sriov_kubepoddevice</b> |
| <b>sriov_kubepoddevice</b>   | Virtual functions linked to active pods.         | -                                                                                                        |

You can also combine these queries with the kube-state-metrics to get more information about the SR-IOV pods. For example, you can use the following query to get the VF network statistics along with the application name from the standard Kubernetes pod label:

```
(sriov_vf_tx_packets * on (pciAddr,node) group_left(pod,namespace) sriov_kubepoddevice) * on (pod,namespace) group_left (label_app_kubernetes_io_name) kube_pod_labels
```

#### 4.10.2.2.1. Enabling the SR-IOV network metrics exporter

The Single Root I/O Virtualization (SR-IOV) network metrics exporter is disabled by default. To enable the metrics exporter, you must set the **spec.featureGates.metricsExporter** field to **true**.



#### IMPORTANT

When the metrics exporter is enabled, the SR-IOV Network Operator deploys the metrics exporter only on nodes with SR-IOV capabilities.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the SR-IOV Network Operator.

#### Procedure

1. Enable cluster monitoring by running the following command:

```
$ oc label ns/openshift-sriov-network-operator openshift.io/cluster-monitoring=true
```

To enable cluster monitoring, you must add the **openshift.io/cluster-monitoring=true** label in the namespace where you have installed the SR-IOV Network Operator.

2. Set the **spec.featureGates.metricsExporter** field to **true** by running the following command:

```
$ oc patch -n openshift-sriov-network-operator sriovoperatorconfig/default \
--type='merge' -p='{"spec": {"featureGates": {"metricsExporter": true}}}'
```

## Verification

- Check that the SR-IOV network metrics exporter is enabled by running the following command:

```
$ oc get pods -n openshift-sriov-network-operator
```

### Example output

| NAME                                   | READY | STATUS  | RESTARTS | AGE   |
|----------------------------------------|-------|---------|----------|-------|
| operator-webhook-hzfg4                 | 1/1   | Running | 0        | 5d22h |
| sriov-network-config-daemon-tr54m      | 1/1   | Running | 0        | 5d22h |
| sriov-network-metrics-exporter-z5d7t   | 1/1   | Running | 0        | 10s   |
| sriov-network-operator-cc6fd88bc-9bsmt | 1/1   | Running | 0        | 5d22h |

The **sriov-network-metrics-exporter** pod must be in the **READY** state.

- Optional: Examine the SR-IOV virtual function (VF) metrics by using the OpenShift Container Platform web console. For more information, see "Querying metrics".

## Additional resources

- [Querying metrics for all projects with the monitoring dashboard](#)
- [Querying metrics for user-defined projects as a developer](#)

### 4.10.2.3. Next steps

- [Configuring an SR-IOV network device](#)
- Optional: [Uninstalling the SR-IOV Network Operator](#)

### 4.10.3. Uninstalling the SR-IOV Network Operator

To uninstall the SR-IOV Network Operator, you must delete any running SR-IOV workloads, uninstall the Operator, and delete the webhooks that the Operator used.

#### 4.10.3.1. Uninstalling the SR-IOV Network Operator

As a cluster administrator, you can uninstall the SR-IOV Network Operator.

#### Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have the SR-IOV Network Operator installed.

#### Procedure

- Delete all SR-IOV custom resources (CRs):

```
$ oc delete sriovnetwork -n openshift-sriov-network-operator --all
```

```
| $ oc delete sriovnetworknodepolicy -n openshift-sriov-network-operator --all
```

```
| $ oc delete sriovibnetwork -n openshift-sriov-network-operator --all
```

```
| $ oc delete sriovoperatorconfigs -n openshift-sriov-network-operator --all
```

2. Follow the instructions in the "Deleting Operators from a cluster" section to remove the SR-IOV Network Operator from your cluster.
3. Delete the SR-IOV custom resource definitions that remain in the cluster after the SR-IOV Network Operator is uninstalled:

```
| $ oc delete crd sriovibnetworks.sriovnetwork.openshift.io
```

```
| $ oc delete crd sriovnetworknodepolicies.sriovnetwork.openshift.io
```

```
| $ oc delete crd sriovnetworknodestates.sriovnetwork.openshift.io
```

```
| $ oc delete crd sriovnetworkpoolconfigs.sriovnetwork.openshift.io
```

```
| $ oc delete crd sriovnetworks.sriovnetwork.openshift.io
```

```
| $ oc delete crd sriovoperatorconfigs.sriovnetwork.openshift.io
```

4. Delete the SR-IOV Network Operator namespace:

```
| $ oc delete namespace openshift-sriov-network-operator
```

## Additional resources

- [Deleting Operators from a cluster](#)

# CHAPTER 5. NETWORK SECURITY

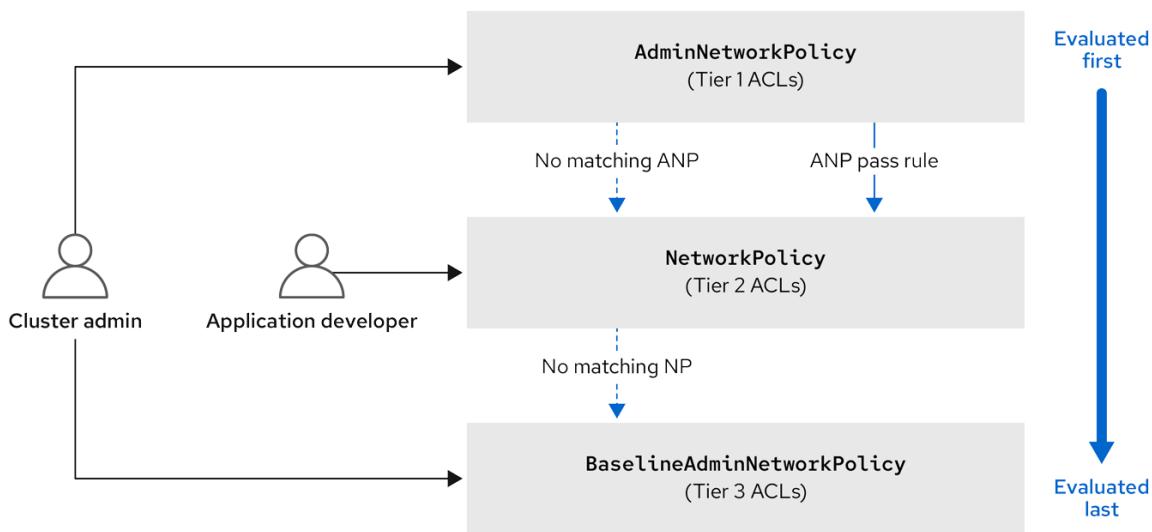
## 5.1. UNDERSTANDING NETWORK POLICY APIs

Kubernetes offers two features that users can use to enforce network security. One feature that allows users to enforce network policy is the **NetworkPolicy** API that is designed mainly for application developers and namespace tenants to protect their namespaces by creating namespace-scoped policies.

The second feature is **AdminNetworkPolicy** which consists of two APIs: the **AdminNetworkPolicy** (ANP) API and the **BaselineAdminNetworkPolicy** (BANP) API. ANP and BANP are designed for cluster and network administrators to protect their entire cluster by creating cluster-scoped policies. Cluster administrators can use ANPs to enforce non-overridable policies that take precedence over **NetworkPolicy** objects. Administrators can use BANP to set up and enforce optional cluster-scoped network policy rules that are overridable by users using **NetworkPolicy** objects when necessary. When used together, ANP, BANP, and network policy can achieve full multi-tenant isolation that administrators can use to secure their cluster.

OVN-Kubernetes CNI in OpenShift Container Platform implements these network policies using Access Control List (ACL) Tiers to evaluate and apply them. ACLs are evaluated in descending order from Tier 1 to Tier 3.

Tier 1 evaluates **AdminNetworkPolicy** (ANP) objects. Tier 2 evaluates **NetworkPolicy** objects. Tier 3 evaluates **BaselineAdminNetworkPolicy** (BANP) objects.



615\_OpenShift\_0324

ANPs are evaluated first. When the match is an ANP **allow** or **deny** rule, any existing **NetworkPolicy** and **BaselineAdminNetworkPolicy** (BANP) objects in the cluster are skipped from evaluation. When the match is an ANP **pass** rule, then evaluation moves from tier 1 of the ACL to tier 2 where the **NetworkPolicy** policy is evaluated. If no **NetworkPolicy** matches the traffic then evaluation moves from tier 2 ACLs to tier 3 ACLs where BANP is evaluated.

### 5.1.1. Key differences between AdminNetworkPolicy and NetworkPolicy custom resources

The following table explains key differences between the cluster scoped **AdminNetworkPolicy** API and the namespace scoped **NetworkPolicy** API.

| Policy elements                                  | AdminNetworkPolicy                                                                                                              | NetworkPolicy                                                                           |
|--------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| Applicable user                                  | Cluster administrator or equivalent                                                                                             | Namespace owners                                                                        |
| Scope                                            | Cluster                                                                                                                         | Namespaced                                                                              |
| Drop traffic                                     | Supported with an explicit <b>Deny</b> action set as a rule.                                                                    | Supported via implicit <b>Deny</b> isolation at policy creation time.                   |
| Delegate traffic                                 | Supported with an <b>Pass</b> action set as a rule.                                                                             | Not applicable                                                                          |
| Allow traffic                                    | Supported with an explicit <b>Allow</b> action set as a rule.                                                                   | The default action for all rules is to allow.                                           |
| Rule precedence within the policy                | Depends on the order in which they appear within an ANP. The higher the rule's position the higher the precedence.              | Rules are additive                                                                      |
| Policy precedence                                | Among ANPs the <b>priority</b> field sets the order for evaluation. The lower the priority number higher the policy precedence. | There is no policy ordering between policies.                                           |
| Feature precedence                               | Evaluated first via tier 1 ACL and BANP is evaluated last via tier 3 ACL.                                                       | Enforced after ANP and before BANP, they are evaluated in tier 2 of the ACL.            |
| Matching pod selection                           | Can apply different rules across namespaces.                                                                                    | Can apply different rules across pods in single namespace.                              |
| Cluster egress traffic                           | Supported via <b>nodes</b> and <b>networks</b> peers                                                                            | Supported through <b>ipBlock</b> field along with accepted CIDR syntax.                 |
| Cluster ingress traffic                          | Not supported                                                                                                                   | Not supported                                                                           |
| Fully qualified domain names (FQDN) peer support | Not supported                                                                                                                   | Not supported                                                                           |
| Namespace selectors                              | Supports advanced selection of Namespaces with the use of <b>namespaces.matchLabels</b> field                                   | Supports label based namespace selection with the use of <b>namespaceSelector</b> field |

## 5.2. ADMIN NETWORK POLICY

### 5.2.1. OVN-Kubernetes AdminNetworkPolicy

### 5.2.1.1. AdminNetworkPolicy

An **AdminNetworkPolicy** (ANP) is a cluster-scoped custom resource definition (CRD). As a OpenShift Container Platform administrator, you can use ANP to secure your network by creating network policies before creating namespaces. Additionally, you can create network policies on a cluster-scoped level that is non-overridable by **NetworkPolicy** objects.

The key difference between **AdminNetworkPolicy** and **NetworkPolicy** objects are that the former is for administrators and is cluster scoped while the latter is for tenant owners and is namespace scoped.

An ANP allows administrators to specify the following:

- A **priority** value that determines the order of its evaluation. The lower the value the higher the precedence.
- A set of pods that consists of a set of namespaces or namespace on which the policy is applied.
- A list of ingress rules to be applied for all ingress traffic towards the **subject**.
- A list of egress rules to be applied for all egress traffic from the **subject**.

#### AdminNetworkPolicy example

##### Example 5.1. Example YAML file for an ANP

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: sample-anp-deny-pass-rules ①
spec:
  priority: 50 ②
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: example.name ③
  ingress: ④
    - name: "deny-all-ingress-tenant-1" ⑤
      action: "Deny"
      from:
        - pods:
            namespaceSelector:
              matchLabels:
                custom-anp: tenant-1
            podSelector:
              matchLabels:
                custom-anp: tenant-1 ⑥
  egress: ⑦
    - name: "pass-all-egress-to-tenant-1"
      action: "Pass"
      to:
        - pods:
            namespaceSelector:
              matchLabels:
                custom-anp: tenant-1
```

```
podSelector:
  matchLabels:
    custom-anp: tenant-1
```

- 1 Specify a name for your ANP.
- 2 The **spec.priority** field supports a maximum of 100 ANPs in the range of values **0-99** in a cluster. The lower the value, the higher the precedence because the range is read in order from the lowest to highest value. Because there is no guarantee which policy takes precedence when ANPs are created at the same priority, set ANPs at different priorities so that precedence is deliberate.
- 3 Specify the namespace to apply the ANP resource.
- 4 ANP have both ingress and egress rules. ANP rules for **spec.ingress** field accepts values of **Pass**, **Deny**, and **Allow** for the **action** field.
- 5 Specify a name for the **ingress.name**.
- 6 Specify **podSelector.matchLabels** to select pods within the namespaces selected by **namespaceSelector.matchLabels** as ingress peers.
- 7 ANPs have both ingress and egress rules. ANP rules for **spec.egress** field accepts values of **Pass**, **Deny**, and **Allow** for the **action** field.

## Additional resources

- [Network Policy API Working Group](#)

### 5.2.1.1. AdminNetworkPolicy actions for rules

As an administrator, you can set **Allow**, **Deny**, or **Pass** as the **action** field for your **AdminNetworkPolicy** rules. Because OVN-Kubernetes uses a tiered ACLs to evaluate network traffic rules, ANP allow you to set very strong policy rules that can only be changed by an administrator modifying them, deleting the rule, or overriding them by setting a higher priority rule.

#### AdminNetworkPolicy Allow example

The following ANP that is defined at priority 9 ensures all ingress traffic is allowed from the **monitoring** namespace towards any tenant (all other namespaces) in the cluster.

#### Example 5.2. Example YAML file for a strongAllow ANP

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: allow-monitoring
spec:
  priority: 9
  subject:
    namespaces: {} # Use the empty selector with caution because it also selects OpenShift namespaces as well.
  ingress:
    - name: "allow-ingress-from-monitoring"
```

```

action: "Allow"
from:
- namespaces:
  matchLabels:
    kubernetes.io/metadata.name: monitoring
# ...

```

This is an example of a strong **Allow** ANP because it is non-overridable by all the parties involved. No tenants can block themselves from being monitored using **NetworkPolicy** objects and the monitoring tenant also has no say in what it can or cannot monitor.

#### AdminNetworkPolicy Deny example

The following ANP that is defined at priority 5 ensures all ingress traffic from the **monitoring** namespace is blocked towards restricted tenants (namespaces that have labels **security: restricted**).

#### Example 5.3. Example YAML file for a strongDeny ANP

```

apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: block-monitoring
spec:
  priority: 5
  subject:
    namespaces:
      matchLabels:
        security: restricted
  ingress:
    - name: "deny-ingress-from-monitoring"
      action: "Deny"
      from:
        - namespaces:
          matchLabels:
            kubernetes.io/metadata.name: monitoring
# ...

```

This is a strong **Deny** ANP that is non-overridable by all the parties involved. The restricted tenant owners cannot authorize themselves to allow monitoring traffic, and the infrastructure's monitoring service cannot scrape anything from these sensitive namespaces.

When combined with the strong **Allow** example, the **block-monitoring** ANP has a lower priority value giving it higher precedence, which ensures restricted tenants are never monitored.

#### AdminNetworkPolicy Pass example

The following ANP that is defined at priority 7 ensures all ingress traffic from the **monitoring** namespace towards internal infrastructure tenants (namespaces that have labels **security: internal**) are passed on to tier 2 of the ACLs and evaluated by the namespaces' **NetworkPolicy** objects.

#### Example 5.4. Example YAML file for a strongPass ANP

```

apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy

```

```

metadata:
  name: pass-monitoring
spec:
  priority: 7
  subject:
    namespaces:
      matchLabels:
        security: internal
  ingress:
    - name: "pass-ingress-from-monitoring"
      action: "Pass"
      from:
        - namespaces:
            matchLabels:
              kubernetes.io/metadata.name: monitoring
# ...

```

This example is a strong **Pass** action ANP because it delegates the decision to **NetworkPolicy** objects defined by tenant owners. This **pass-monitoring** ANP allows all tenant owners grouped at security level **internal** to choose if their metrics should be scraped by the infrastructures' monitoring service using namespace scoped **NetworkPolicy** objects.

## 5.2.2. OVN-Kubernetes BaselineAdminNetworkPolicy

### 5.2.2.1. BaselineAdminNetworkPolicy

**BaselineAdminNetworkPolicy** (BANP) is a cluster-scoped custom resource definition (CRD). As a OpenShift Container Platform administrator, you can use BANP to setup and enforce optional baseline network policy rules that are overridable by users using **NetworkPolicy** objects if need be. Rule actions for BANP are **allow** or **deny**.

The **BaselineAdminNetworkPolicy** resource is a cluster singleton object that can be used as a guardrail policy incase a passed traffic policy does not match any **NetworkPolicy** objects in the cluster. A BANP can also be used as a default security model that provides guardrails that intra-cluster traffic is blocked by default and a user will need to use **NetworkPolicy** objects to allow known traffic. You must use **default** as the name when creating a BANP resource.

A BANP allows administrators to specify:

- A **subject** that consists of a set of namespaces or namespace.
- A list of ingress rules to be applied for all ingress traffic towards the **subject**.
- A list of egress rules to be applied for all egress traffic from the **subject**.

#### BaselineAdminNetworkPolicy example

##### Example 5.5. Example YAML file for BANP

```

apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default ①
spec:
  subject:

```

```

namespaces:
  matchLabels:
    kubernetes.io/metadata.name: example.name 2
ingress: 3
  - name: "deny-all-ingress-from-tenant-1" 4
    action: "Deny"
    from:
      - pods:
          namespaceSelector:
            matchLabels:
              custom-banp: tenant-1 5
        podSelector:
          matchLabels:
            custom-banp: tenant-1 6
egress:
  - name: "allow-all-egress-to-tenant-1"
    action: "Allow"
    to:
      - pods:
          namespaceSelector:
            matchLabels:
              custom-banp: tenant-1
        podSelector:
          matchLabels:
            custom-banp: tenant-1

```

- 1** The policy name must be **default** because BANP is a singleton object.
- 2** Specify the namespace to apply the ANP to.
- 3** BANP have both ingress and egress rules. BANP rules for **spec.ingress** and **spec.egress** fields accepts values of **Deny** and **Allow** for the **action** field.
- 4** Specify a name for the **ingress.name**
- 5** Specify the namespaces to select the pods from to apply the BANP resource.
- 6** Specify **podSelector.matchLabels** name of the pods to apply the BANP resource.

### BaselineAdminNetworkPolicy Deny example

The following BANP singleton ensures that the administrator has set up a default deny policy for all ingress monitoring traffic coming into the tenants at **internal** security level. When combined with the "AdminNetworkPolicy Pass example", this deny policy acts as a guardrail policy for all ingress traffic that is passed by the ANP **pass-monitoring** policy.

#### Example 5.6. Example YAML file for a guardrailDeny rule

```

apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default
spec:
  subject:

```

```

namespaces:
  matchLabels:
    security: internal
ingress:
- name: "deny-ingress-from-monitoring"
  action: "Deny"
  from:
  - namespaces:
    matchLabels:
      kubernetes.io/metadata.name: monitoring
# ...

```

You can use an **AdminNetworkPolicy** resource with a **Pass** value for the **action** field in conjunction with the **BaselineAdminNetworkPolicy** resource to create a multi-tenant policy. This multi-tenant policy allows one tenant to collect monitoring data on their application while simultaneously not collecting data from a second tenant.

As an administrator, if you apply both the "AdminNetworkPolicy **Pass** action example" and the "BaselineAdminNetwork Policy **Deny** example", tenants are then left with the ability to choose to create a **NetworkPolicy** resource that will be evaluated before the BANP.

For example, Tenant 1 can set up the following **NetworkPolicy** resource to monitor ingress traffic:

#### Example 5.7. Example NetworkPolicy

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-monitoring
  namespace: tenant 1
spec:
  podSelector:
  policyTypes:
    - Ingress
  ingress:
    - from:
      - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...

```

In this scenario, Tenant 1's policy would be evaluated after the "AdminNetworkPolicy **Pass** action example" and before the "BaselineAdminNetwork Policy **Deny** example", which denies all ingress monitoring traffic coming into tenants with **security** level **internal**. With Tenant 1's **NetworkPolicy** object in place, they will be able to collect data on their application. Tenant 2, however, who does not have any **NetworkPolicy** objects in place, will not be able to collect data. As an administrator, you have not by default monitored internal tenants, but instead, you created a BANP that allows tenants to use **NetworkPolicy** objects to override the default behavior of your BANP.

### 5.2.3. Monitoring ANP and BANP

**AdminNetworkPolicy** and **BaselineAdminNetworkPolicy** resources have metrics that can be used for monitoring and managing your policies. See the following table for more details on the metrics.

### 5.2.3.1. Metrics for AdminNetworkPolicy

| Name                                                                       | Description                                                                                                                                                                                                   | Explanation                                                                                                                                   |
|----------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ovnkube_controller_admin_network_policies</code>                     | Not applicable                                                                                                                                                                                                | The total number of <b>AdminNetworkPolicy</b> resources in the cluster.                                                                       |
| <code>ovnkube_controller_baseline_admin_network_policies</code>            | Not applicable                                                                                                                                                                                                | The total number of <b>BaselineAdminNetworkPolicy</b> resources in the cluster. The value should be 0 or 1.                                   |
| <code>ovnkube_controller_admin_network_policies_rules</code>               | <ul style="list-style-type: none"> <li>● <b>direction</b>: specifies either <b>Ingress</b> or <b>Egress</b>.</li> <li>● <b>action</b>: specifies either <b>Pass</b>, <b>Allow</b>, or <b>Deny</b>.</li> </ul> | The total number of rules across all ANP policies in the cluster grouped by <b>direction</b> and <b>action</b> .                              |
| <code>ovnkube_controller_baseline_admin_network_policies_rules</code>      | <ul style="list-style-type: none"> <li>● <b>direction</b>: specifies either <b>Ingress</b> or <b>Egress</b>.</li> <li>● <b>action</b>: specifies either <b>Allow</b> or <b>Deny</b>.</li> </ul>               | The total number of rules across all BANP policies in the cluster grouped by <b>direction</b> and <b>action</b> .                             |
| <code>ovnkube_controller_admin_network_policies_db_objects</code>          | <b>table_name</b> : specifies either <b>ACL</b> or <b>Address_Set</b>                                                                                                                                         | The total number of OVN Northbound database (nbdb) objects that are created by all the ANP in the cluster grouped by the <b>table_name</b> .  |
| <code>ovnkube_controller_baseline_admin_network_policies_db_objects</code> | <b>table_name</b> : specifies either <b>ACL</b> or <b>Address_Set</b>                                                                                                                                         | The total number of OVN Northbound database (nbdb) objects that are created by all the BANP in the cluster grouped by the <b>table_name</b> . |

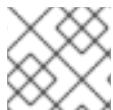
### 5.2.4. Egress nodes and networks peer for AdminNetworkPolicy

This section explains **nodes** and **networks** peers. Administrators can use the examples in this section to design **AdminNetworkPolicy** and **BaselineAdminNetworkPolicy** to control northbound traffic in their cluster.

#### 5.2.4.1. Northbound traffic controls for AdminNetworkPolicy and BaselineAdminNetworkPolicy

In addition to supporting east-west traffic controls, ANP and BANP also allow administrators to control their northbound traffic leaving the cluster or traffic leaving the node to other nodes in the cluster. End-users can do the following:

- Implement egress traffic control towards cluster nodes using **nodes** egress peer
- Implement egress traffic control towards Kubernetes API servers using **nodes** or **networks** egress peers
- Implement egress traffic control towards external destinations outside the cluster using **networks** peer



### NOTE

For ANP and BANP, **nodes** and **networks** peers can be specified for egress rules only.

#### 5.2.4.1.1. Using **nodes** peer to control egress traffic to cluster nodes

Using the **nodes** peer administrators can control egress traffic from pods to nodes in the cluster. A benefit of this is that you do not have to change the policy when nodes are added to or deleted from the cluster.

The following example allows egress traffic to the Kubernetes API server on port **6443** by any of the namespaces with a **restricted**, **confidential**, or **internal** level of security using the node selector peer. It also denies traffic to all worker nodes in your cluster from any of the namespaces with a **restricted**, **confidential**, or **internal** level of security.

#### Example 5.8. Example of ANPAllow egress using **nodes** peer

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: egress-security-allow
spec:
  egress:
    - action: Deny
      to:
        - nodes:
            matchExpressions:
              - key: node-role.kubernetes.io/worker
                operator: Exists
    - action: Allow
      name: allow-to-kubernetes-api-server-and-engr-dept-pods
      ports:
        - portNumber:
            port: 6443
            protocol: TCP
      to:
        - nodes: ①
          matchExpressions:
            - key: node-role.kubernetes.io/control-plane
              operator: Exists
        - pods: ②
          namespaceSelector:
          matchLabels:
```

```

    dept: engr
    podSelector: {}
priority: 55
subject: ③
namespaces:
  matchExpressions:
    - key: security ④
      operator: In
      values:
        - restricted
        - confidential
        - internal

```

- ① Specifies a node or set of nodes in the cluster using the **matchExpressions** field.
- ② Specifies all the pods labeled with **dept: engr**.
- ③ Specifies the subject of the ANP which includes any namespaces that match the labels used by the network policy. The example matches any of the namespaces with **restricted**, **confidential**, or **internal** level of **security**.
- ④ Specifies key/value pairs for **matchExpressions** field.

#### 5.2.4.1.2. Using networks peer to control egress traffic towards external destinations

Cluster administrators can use CIDR ranges in **networks** peer and apply a policy to control egress traffic leaving from pods and going to a destination configured at the IP address that is within the CIDR range specified with **networks** field.

The following example uses **networks** peer and combines ANP and BANP policies to restrict egress traffic.



#### IMPORTANT

Use the empty selector ({} ) in the **namespace** field for ANP and BANP with caution. When using an empty selector, it also selects OpenShift namespaces.

If you use values of **0.0.0.0/0** in a ANP or BANP **Deny** rule, you must set a higher priority ANP **Allow** rule to necessary destinations before setting the **Deny** to **0.0.0.0/0**.

#### Example 5.9. Example of ANP and BANP using**networks** peers

```

apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: network-as-egress-peer
spec:
  priority: 70
  subject:
    namespaces: {} # Use the empty selector with caution because it also selects OpenShift
    namespaces as well.
  egress:

```

```

- name: "deny-egress-to-external-dns-servers"
  action: "Deny"
  to:
    - networks: ①
      - 8.8.8.0/32
      - 8.8.4.0/32
      - 208.67.222.222/32
  ports:
    - portNumber:
        protocol: UDP
        port: 53
- name: "allow-all-egress-to-intranet"
  action: "Allow"
  to:
    - networks: ②
      - 89.246.180.0/22
      - 60.45.72.0/22
- name: "allow-all-intra-cluster-traffic"
  action: "Allow"
  to:
    - namespaces: {} # Use the empty selector with caution because it also selects OpenShift
                      namespaces as well.
- name: "pass-all-egress-to-internet"
  action: "Pass"
  to:
    - networks:
      - 0.0.0.0/0 ③
---
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default
spec:
  subject:
    namespaces: {} # Use the empty selector with caution because it also selects OpenShift
                    namespaces as well.
  egress:
    - name: "deny-all-egress-to-internet"
      action: "Deny"
      to:
        - networks:
          - 0.0.0.0/0 ④
---

```

① Use **networks** to specify a range of CIDR networks outside of the cluster.

② Specifies the CIDR ranges for the intra-cluster traffic from your resources.

③ ④ Specifies a **Deny** egress to everything by setting **networks** values to **0.0.0.0/0**. Make sure you have a higher priority **Allow** rule to necessary destinations before setting a **Deny** to **0.0.0.0/0** because this will deny all traffic including to Kubernetes API and DNS servers.

Collectively the **network-as-egress-peer** ANP and **default** BANP using **networks** peers enforces the following egress policy:

- All pods cannot talk to external DNS servers at the listed IP addresses.
- All pods can talk to rest of the company's intranet.
- All pods can talk to other pods, nodes, and services.
- All pods cannot talk to the internet. Combining the last ANP **Pass** rule and the strong BANP **Deny** rule a guardrail policy is created that secures traffic in the cluster.

#### 5.2.4.1.3. Using nodes peer and networks peer together

Cluster administrators can combine **nodes** and **networks** peer in your ANP and BANP policies.

##### Example 5.10. Example of nodes and networks peer

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: egress-peer-1 ①
spec:
  egress: ②
    - action: "Allow"
      name: "allow-egress"
      to:
        - nodes:
            matchExpressions:
              - key: worker-group
                operator: In
                values:
                  - workloads # Egress traffic from nodes with label worker-group: workloads is allowed.
        - networks:
            - 104.154.164.170/32
        - pods:
            namespaceSelector:
              matchLabels:
                apps: external-apps
            podSelector:
              matchLabels:
                app: web # This rule in the policy allows the traffic directed to pods labeled apps: web in
                projects with apps: external-apps to leave the cluster.
            - action: "Deny"
              name: "deny-egress"
              to:
                - nodes:
                    matchExpressions:
                      - key: worker-group
                        operator: In
                        values:
                          - infra # Egress traffic from nodes with label worker-group: infra is denied.
                - networks:
                    - 104.154.164.160/32 # Egress traffic to this IP address from cluster is denied.
                - pods:
                    namespaceSelector:
```

```

matchLabels:
  apps: internal-apps
  podSelector: {}
- action: "Pass"
  name: "pass-egress"
  to:
  - nodes:
    matchExpressions:
      - key: node-role.kubernetes.io/worker
        operator: Exists # All other egress traffic is passed to NetworkPolicy or BANP for evaluation.
priority: 30 3
subject: 4
namespaces:
  matchLabels:
    apps: all-apps

```

- 1** Specifies the name of the policy.
- 2** For **nodes** and **networks** peers, you can only use northbound traffic controls in ANP as **egress**.
- 3** Specifies the priority of the ANP, determining the order in which they should be evaluated. Lower priority rules have higher precedence. ANP accepts values of 0-99 with 0 being the highest priority and 99 being the lowest.
- 4** Specifies the set of pods in the cluster on which the rules of the policy are to be applied. In the example, any pods with the **apps: all-apps** label across all namespaces are the **subject** of the policy.

## 5.2.5. Troubleshooting AdminNetworkPolicy

### 5.2.5.1. Checking creation of ANP

To check that your **AdminNetworkPolicy** (ANP) and **BaselineAdminNetworkPolicy** (BANP) are created correctly, check the status outputs of the following commands: **oc describe anp** or **oc describe banp**.

A good status indicates **OVN DB plumbing was successful** and the **SetupSucceeded**.

#### Example 5.11. Example ANP with a good status

```

...
Conditions:
Last Transition Time: 2024-06-08T20:29:00Z
Message:      Setting up OVN DB plumbing was successful
Reason:       SetupSucceeded
Status:       True
Type:        Ready-In-Zone-ovn-control-plane Last Transition Time: 2024-06-08T20:29:00Z
Message:      Setting up OVN DB plumbing was successful
Reason:       SetupSucceeded
Status:       True
Type:        Ready-In-Zone-ovn-worker
Last Transition Time: 2024-06-08T20:29:00Z
Message:      Setting up OVN DB plumbing was successful

```

```

| Reason:      SetupSucceeded
| Status:      True
| Type:        Ready-In-Zone-ovn-worker2
| ...

```

If plumbing is unsuccessful, an error is reported from the respective zone controller.

#### Example 5.12. Example of an ANP with a bad status and error message

```

...
Status:
Conditions:
Last Transition Time: 2024-06-25T12:47:44Z
Message:      error attempting to add ANP cluster-control with priority 600 because, OVNK
only supports priority ranges 0-99
Reason:      SetupFailed
Status:      False
Type:        Ready-In-Zone-example-worker-1.example.example-org.net
Last Transition Time: 2024-06-25T12:47:45Z
Message:      error attempting to add ANP cluster-control with priority 600 because, OVNK
only supports priority ranges 0-99
Reason:      SetupFailed
Status:      False
Type:        Ready-In-Zone-example-worker-0.example.example-org.net
Last Transition Time: 2024-06-25T12:47:44Z
Message:      error attempting to add ANP cluster-control with priority 600 because, OVNK
only supports priority ranges 0-99
Reason:      SetupFailed
Status:      False
Type:        Ready-In-Zone-example-ctlplane-1.example.example-org.net
Last Transition Time: 2024-06-25T12:47:44Z
Message:      error attempting to add ANP cluster-control with priority 600 because, OVNK
only supports priority ranges 0-99
Reason:      SetupFailed
Status:      False
Type:        Ready-In-Zone-example-ctlplane-2.example.example-org.net
Last Transition Time: 2024-06-25T12:47:44Z
Message:      error attempting to add ANP cluster-control with priority 600 because, OVNK
only supports priority ranges 0-99
Reason:      SetupFailed
Status:      False
Type:        Ready-In-Zone-example-ctlplane-0.example.example-org.net
...

```

See the following section for **nbctl** commands to help troubleshoot unsuccessful policies.

##### 5.2.5.1.1. Using nbctl commands for ANP and BANP

To troubleshoot an unsuccessful setup, start by looking at OVN Northbound database (nbdb) objects including **ACL**, **AddressSet**, and **Port\_Group**. To view the nbdb, you need to be inside the pod on that node to view the objects in that node's database.

## Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- The OpenShift CLI (**oc**) installed.



### NOTE

To run ovn **nbctl** commands in a cluster, you must open a remote shell into the `nbdb` on the relevant node.

The following policy was used to generate outputs.

#### Example 5.13. AdminNetworkPolicy used to generate outputs

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: cluster-control
spec:
  priority: 34
  subject:
    namespaces:
      matchLabels:
        anp: cluster-control-anp # Only namespaces with this label have this ANP
  ingress:
    - name: "allow-from-ingress-router" # rule0
      action: "Allow"
      from:
        - namespaces:
            matchLabels:
              policy-group.network.openshift.io/ingress: ""
    - name: "allow-from-monitoring" # rule1
      action: "Allow"
      from:
        - namespaces:
            matchLabels:
              kubernetes.io/metadata.name: openshift-monitoring
  ports:
    - portNumber:
        protocol: TCP
        port: 7564
    - namedPort: "scrape"
  - name: "allow-from-open-tenants" # rule2
    action: "Allow"
    from:
      - namespaces: # open tenants
        matchLabels:
          tenant: open
  - name: "pass-from-restricted-tenants" # rule3
    action: "Pass"
    from:
      - namespaces: # restricted tenants
        matchLabels:
          tenant: restricted
```

```

- name: "default-deny" # rule4
  action: "Deny"
  from:
    - namespaces: {} # Use the empty selector with caution because it also selects OpenShift
      namespaces as well.
  egress:
    - name: "allow-to-dns" # rule0
      action: "Allow"
      to:
        - pods:
            namespaceSelector:
              matchLabels:
                kubernetes.io/metadata.name: openshift-dns
            podSelector:
              matchLabels:
                app: dns
  ports:
    - portNumber:
        protocol: UDP
        port: 5353
- name: "allow-to-kapi-server" # rule1
  action: "Allow"
  to:
    - nodes:
        matchExpressions:
          - key: node-role.kubernetes.io/control-plane
            operator: Exists
  ports:
    - portNumber:
        protocol: TCP
        port: 6443
- name: "allow-to-splunk" # rule2
  action: "Allow"
  to:
    - namespaces:
        matchLabels:
          tenant: splunk
  ports:
    - portNumber:
        protocol: TCP
        port: 8991
    - portNumber:
        protocol: TCP
        port: 8992
- name: "allow-to-open-tenants-and-intranet-and-worker-nodes" # rule3
  action: "Allow"
  to:
    - nodes: # worker-nodes
        matchExpressions:
          - key: node-role.kubernetes.io/worker
            operator: Exists
    - networks: # intranet
        - 172.29.0.0/30
        - 10.0.54.0/19
        - 10.0.56.38/32
        - 10.0.69.0/24

```

```

- namespaces: # open tenants
  matchLabels:
    tenant: open
- name: "pass-to-restricted-tenants" # rule4
  action: "Pass"
  to:
    - namespaces: # restricted tenants
      matchLabels:
        tenant: restricted
- name: "default-deny"
  action: "Deny"
  to:
    - networks:
      - 0.0.0.0/0

```

## Procedure

1. List pods with node information by running the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

### Example output

| NAME                                   | READY  | STATUS  | RESTARTS | AGE | IP         | NODE                                     |
|----------------------------------------|--------|---------|----------|-----|------------|------------------------------------------|
| NOMINATED NODE READINESS GATES         |        |         |          |     |            |                                          |
| ovnkube-control-plane-5c95487779-8k9fd | 2/2    | Running | 0        | 34m | 10.0.0.5   | ci-In-0tv5gg2-72292-6sjw5-master-0       |
|                                        | <none> | <none>  |          |     |            |                                          |
| ovnkube-control-plane-5c95487779-v2xn8 | 2/2    | Running | 0        | 34m | 10.0.0.3   | ci-In-0tv5gg2-72292-6sjw5-master-1       |
|                                        | <none> | <none>  |          |     |            |                                          |
| ovnkube-node-524dt                     | 8/8    | Running | 0        | 33m | 10.0.0.4   | ci-In-0tv5gg2-72292-6sjw5-master-2       |
|                                        | <none> | <none>  |          |     |            |                                          |
| ovnkube-node-gbwr9                     | 8/8    | Running | 0        | 24m | 10.0.128.4 | ci-In-0tv5gg2-72292-6sjw5-worker-c-s9gqt |
|                                        | <none> | <none>  |          |     |            |                                          |
| ovnkube-node-h4fpv                     | 8/8    | Running | 0        | 33m | 10.0.0.5   | ci-In-0tv5gg2-72292-6sjw5-master-0       |
|                                        | <none> | <none>  |          |     |            |                                          |
| ovnkube-node-j4hzw                     | 8/8    | Running | 0        | 24m | 10.0.128.2 | ci-In-0tv5gg2-72292-6sjw5-worker-a-hzbh5 |
|                                        | <none> | <none>  |          |     |            |                                          |
| ovnkube-node-wdhgv                     | 8/8    | Running | 0        | 33m | 10.0.0.3   | ci-In-0tv5gg2-72292-6sjw5-master-1       |
|                                        | <none> | <none>  |          |     |            |                                          |
| ovnkube-node-wfncn                     | 8/8    | Running | 0        | 24m | 10.0.128.3 | ci-In-0tv5gg2-72292-6sjw5-worker-b-5bb7f |
|                                        | <none> | <none>  |          |     |            |                                          |

2. Navigate into a pod to look at the northbound database by running the following command:

```
$ oc rsh -c nbdb -n openshift-ovn-kubernetes ovnkube-node-524dt
```

3. Run the following command to look at the ACLs nbdb:

```
$ ovn-nbctl find ACL 'external_ids{>=}{"k8s.ovn.org/owner-type"=AdminNetworkPolicy,"k8s.ovn.org/name"=cluster-control}'
```

Where, cluster-control

Specifies the name of the **AdminNetworkPolicy** you are troubleshooting.

### AdminNetworkPolicy

Specifies the type: **AdminNetworkPolicy** or **BaselineAdminNetworkPolicy**.

#### Example 5.14. Example output for ACLs

```

_uuid      : 0d5e4722-b608-4bb1-b625-23c323cc9926
action     : allow-related
direction   : to-lport
external_ids: {"direction=Ingress, gress-index=2", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Ingress:2:None", "k8s.ovn.org/name=cluster-control", "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=None}
label      : 0
log        : false
match      : "outport == @a14645450421485494999 && ((ip4.src == $a13730899355151937870))"
meter      : acl-logging
name       : "ANP:cluster-control:Ingress:2"
options    : {}
priority   : 26598
severity   : []
tier       : 1

_uuid      : b7be6472-df67-439c-8c9c-f55929f0a6e0
action     : drop
direction   : from-lport
external_ids: {"direction=Egress, gress-index=5", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Egress:5:None", "k8s.ovn.org/name=cluster-control", "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=None}
label      : 0
log        : false
match      : "inport == @a14645450421485494999 && ((ip4.dst == $a11452480169090787059))"
meter      : acl-logging
name       : "ANP:cluster-control:Egress:5"
options    : {"apply-after-lb=true"}
priority   : 26595
severity   : []
tier       : 1

_uuid      : 5a6e5bb4-36eb-4209-b8bc-c611983d4624
action     : pass
direction   : to-lport
external_ids: {"direction=Ingress, gress-index=3", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Ingress:3:None", "k8s.ovn.org/name=cluster-control", "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=None}
label      : 0
log        : false
match      : "outport == @a14645450421485494999 && ((ip4.src == $a764182844364804195))"
meter      : acl-logging
name       : "ANP:cluster-control:Ingress:3"
```

```

options      : {}
priority     : 26597
severity     : []
tier         : 1

_uuid       : 04f20275-c410-405c-a923-0e677f767889
action       : pass
direction    : from-lport
external_ids : {"direction=Egress, gress-index=4", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Egress:4:None", "k8s.ovn.org/name=cluster-control, "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=None}
label        : 0
log          : false
match        : "inport == @a14645450421485494999 && ((ip4.dst == $a5972452606168369118))"
meter        : acl-logging
name         : "ANP:cluster-control:Egress:4"
options       : {apply-after-lb="true"}
priority     : 26596
severity     : []
tier         : 1

_uuid       : 4b5d836a-e0a3-4088-825e-f9f0ca58e538
action       : drop
direction    : to-lport
external_ids : {"direction=Ingress, gress-index=4", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Ingress:4:None", "k8s.ovn.org/name=cluster-control, "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=None}
label        : 0
log          : false
match        : "outport == @a14645450421485494999 && ((ip4.src == $a13814616246365836720))"
meter        : acl-logging
name         : "ANP:cluster-control:Ingress:4"
options       : {}
priority     : 26596
severity     : []
tier         : 1

_uuid       : 5d09957d-d2cc-4f5a-9ddd-b97d9d772023
action       : allow-related
direction    : from-lport
external_ids : {"direction=Egress, gress-index=2", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Egress:2:tcp", "k8s.ovn.org/name=cluster-control, "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=tcp}
label        : 0
log          : false
match        : "inport == @a14645450421485494999 && ((ip4.dst == $a18396736153283155648)) && tcp && tcp.dst=={8991,8992}"
meter        : acl-logging
name         : "ANP:cluster-control:Egress:2"
options       : {apply-after-lb="true"}
priority     : 26598

```

```

severity      : []
tier         : 1

_uuid        : 1a68a5ed-e7f9-47d0-b55c-89184d97e81a
action       : allow-related
direction    : from-lport
external_ids : {direction=Egress, gress-index="1", "k8s.ovn.org/id"="default-network-
controller:AdminNetworkPolicy:cluster-control:Egress:1:tcp", "k8s.ovn.org/name"=cluster-
control, "k8s.ovn.org/owner-controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy, port-policy-protocol=tcp}
label        : 0
log          : false
match        : "inport == @a14645450421485494999 && ((ip4.dst ==
$a10706246167277696183)) && tcp && tcp.dst==6443"
meter        : acl-logging
name         : "ANP:cluster-control:Egress:1"
options      : {apply-after-lb="true"}
priority     : 26599
severity    : []
tier         : 1

_uuid        : aa1a224d-7960-4952-bdfb-35246bafbac8
action       : allow-related
direction    : to-lport
external_ids : {direction=Ingress, gress-index="1", "k8s.ovn.org/id"="default-network-
controller:AdminNetworkPolicy:cluster-control:Ingress:1:tcp", "k8s.ovn.org/name"=cluster-
control, "k8s.ovn.org/owner-controller"=default-network-controller, "k8s.ovn.org/owner-
type"=AdminNetworkPolicy, port-policy-protocol=tcp}
label        : 0
log          : false
match        : "outport == @a14645450421485494999 && ((ip4.src ==
$a6786643370959569281)) && tcp && tcp.dst==7564"
meter        : acl-logging
name         : "ANP:cluster-control:Ingress:1"
options      : {}
priority     : 26599
severity    : []
tier         : 1

_uuid        : 1a27d30e-3f96-4915-8ddd-ade7f22c117b
action       : allow-related
direction    : from-lport
external_ids : {direction=Egress, gress-index="3", "k8s.ovn.org/id"="default-network-
controller:AdminNetworkPolicy:cluster-control:Egress:3:None",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-network-
controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-protocol=None}
label        : 0
log          : false
match        : "inport == @a14645450421485494999 && ((ip4.dst ==
$a10622494091691694581))"
meter        : acl-logging
name         : "ANP:cluster-control:Egress:3"
options      : {apply-after-lb="true"}
priority     : 26597
severity    : []
tier         : 1

```

```

_uuid      : b23a087f-08f8-4225-8c27-4a9a9ee0c407
action     : allow-related
direction   : from-lport
external_ids: {"direction=Egress, gress-index=0", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Egress:0:udp", "k8s.ovn.org/name=cluster-control, "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=udp}
label      : 0
log        : false
match      : "inport == @a14645450421485494999 && ((ip4.dst ==
$a13517855690389298082)) && udp && udp.dst==5353"
meter      : acl-logging
name       : "ANP:cluster-control:Egress:0"
options    : {apply-after-lb="true"}
priority   : 26600
severity   : []
tier       : 1

_uuid      : d14ed5cf-2e06-496e-8cae-6b76d5dd5ccd
action     : allow-related
direction   : to-lport
external_ids: {"direction=Ingress, gress-index=0", "k8s.ovn.org/id=default-network-controller:AdminNetworkPolicy:cluster-control:Ingress:0:None", "k8s.ovn.org/name=cluster-control, "k8s.ovn.org/owner-controller=default-network-controller, "k8s.ovn.org/owner-type=AdminNetworkPolicy, port-policy-protocol=None}
label      : 0
log        : false
match      : "outport == @a14645450421485494999 && ((ip4.src ==
$a14545668191619617708))"
meter      : acl-logging
name       : "ANP:cluster-control:Ingress:0"
options    : {}
priority   : 26600
severity   : []
tier       : 1

```



### NOTE

The outputs for ingress and egress show you the logic of the policy in the ACL. For example, every time a packet matches the provided **match** the **action** is taken.

- Examine the specific ACL for the rule by running the following command:

```
$ ovn-nbctl find ACL 'external_ids{>=}{"k8s.ovn.org/owner-type=AdminNetworkPolicy,direction=Ingress,"k8s.ovn.org/name=cluster-control,gress-index=1"}'
```

### Where, **cluster-control**

Specifies the **name** of your ANP.

### Ingress

Specifies the **direction** of traffic either of type **Ingress** or **Egress**.

**1**

Specifies the rule you want to look at.

For the example ANP named **cluster-control** at **priority 34**, the following is an example output for **Ingress rule 1**:

#### Example 5.15. Example output

```
_uuid      : aa1a224d-7960-4952-bdfb-35246bafbac8
action     : allow-related
direction   : to-lport
external_ids: {direction=Ingress, gress-index="1", "k8s.ovn.org/id"="default-
network-controller:AdminNetworkPolicy:cluster-control:Ingress:1:tcp",
"k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-network-
controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy, port-policy-protocol=tcp}
label      : 0
log        : false
match      : "outport == @a14645450421485494999 && ((ip4.src ==
$a6786643370959569281)) && tcp && tcp.dst==7564"
meter      : acl-logging
name       : "ANP:cluster-control:Ingress:1"
options    : {}
priority   : 26599
severity   : []
tier       : 1
```

4. Run the following command to look at address sets in the nbdb:

```
$ ovn-nbctl find Address_Set 'external_ids{>=}{ "k8s.ovn.org/owner-
type"=AdminNetworkPolicy,"k8s.ovn.org/name"=cluster-control}'
```

#### Example 5.16. Example outputs forAddress\_Set

```
_uuid      : 56e89601-5552-4238-9fc3-8833f5494869
addresses  : ["192.168.194.135", "192.168.194.152", "192.168.194.193",
"192.168.194.254"]
external_ids: {direction=Egress, gress-index="1", ip-family=v4,
"k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
control:Egress:1:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}
name       : a10706246167277696183

_uuid      : 7df9330d-380b-4bdb-8acd-4eddeda2419c
addresses  : ["10.132.0.10", "10.132.0.11", "10.132.0.12", "10.132.0.13",
"10.132.0.14", "10.132.0.15", "10.132.0.16", "10.132.0.17", "10.132.0.5", "10.132.0.7",
"10.132.0.71", "10.132.0.75", "10.132.0.8", "10.132.0.81", "10.132.0.9", "10.132.2.10",
"10.132.2.11", "10.132.2.12", "10.132.2.14", "10.132.2.15", "10.132.2.3", "10.132.2.4",
"10.132.2.5", "10.132.2.6", "10.132.2.7", "10.132.2.8", "10.132.2.9", "10.132.3.64",
"10.132.3.65", "10.132.3.72", "10.132.3.73", "10.132.3.76", "10.133.0.10", "10.133.0.11",
"10.133.0.12", "10.133.0.13", "10.133.0.14", "10.133.0.15", "10.133.0.16", "10.133.0.17",
"10.133.0.18", "10.133.0.19", "10.133.0.20", "10.133.0.21", "10.133.0.22", "10.133.0.23",
"10.133.0.24", "10.133.0.25", "10.133.0.26", "10.133.0.27", "10.133.0.28", "10.133.0.29",
```

```

    "10.133.0.30", "10.133.0.31", "10.133.0.32", "10.133.0.33", "10.133.0.34", "10.133.0.35",
    "10.133.0.36", "10.133.0.37", "10.133.0.38", "10.133.0.39", "10.133.0.40", "10.133.0.41",
    "10.133.0.42", "10.133.0.44", "10.133.0.45", "10.133.0.46", "10.133.0.47", "10.133.0.48",
    "10.133.0.5", "10.133.0.6", "10.133.0.7", "10.133.0.8", "10.133.0.9", "10.134.0.10",
    "10.134.0.11", "10.134.0.12", "10.134.0.13", "10.134.0.14", "10.134.0.15", "10.134.0.16",
    "10.134.0.17", "10.134.0.18", "10.134.0.19", "10.134.0.20", "10.134.0.21", "10.134.0.22",
    "10.134.0.23", "10.134.0.24", "10.134.0.25", "10.134.0.26", "10.134.0.27", "10.134.0.28",
    "10.134.0.30", "10.134.0.31", "10.134.0.32", "10.134.0.33", "10.134.0.34", "10.134.0.35",
    "10.134.0.36", "10.134.0.37", "10.134.0.38", "10.134.0.4", "10.134.0.42", "10.134.0.9",
    "10.135.0.10", "10.135.0.11", "10.135.0.12", "10.135.0.13", "10.135.0.14", "10.135.0.15",
    "10.135.0.16", "10.135.0.17", "10.135.0.18", "10.135.0.19", "10.135.0.23", "10.135.0.24",
    "10.135.0.26", "10.135.0.27", "10.135.0.29", "10.135.0.3", "10.135.0.4", "10.135.0.40",
    "10.135.0.41", "10.135.0.42", "10.135.0.43", "10.135.0.44", "10.135.0.5", "10.135.0.6",
    "10.135.0.7", "10.135.0.8", "10.135.0.9"]

  external_ids : {direction=Ingress, gress-index="4", ip-family=v4,
  "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
  control:Ingress:4:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
  controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}
  name : a13814616246365836720

  _uuid : 84d76f13-ad95-4c00-8329-a0b1d023c289
  addresses : ["10.132.3.76", "10.135.0.44"]
  external_ids : {direction=Egress, gress-index="4", ip-family=v4,
  "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
  control:Egress:4:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
  controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}
  name : a5972452606168369118

  _uuid : 0c53e917-f7ee-4256-8f3a-9522c0481e52
  addresses : ["10.132.0.10", "10.132.0.11", "10.132.0.12", "10.132.0.13",
  "10.132.0.14", "10.132.0.15", "10.132.0.16", "10.132.0.17", "10.132.0.5", "10.132.0.7",
  "10.132.0.71", "10.132.0.75", "10.132.0.8", "10.132.0.81", "10.132.0.9", "10.132.2.10",
  "10.132.2.11", "10.132.2.12", "10.132.2.14", "10.132.2.15", "10.132.2.3", "10.132.2.4",
  "10.132.2.5", "10.132.2.6", "10.132.2.7", "10.132.2.8", "10.132.2.9", "10.132.3.64",
  "10.132.3.65", "10.132.3.72", "10.132.3.73", "10.132.3.76", "10.133.0.10", "10.133.0.11",
  "10.133.0.12", "10.133.0.13", "10.133.0.14", "10.133.0.15", "10.133.0.16", "10.133.0.17",
  "10.133.0.18", "10.133.0.19", "10.133.0.20", "10.133.0.21", "10.133.0.22", "10.133.0.23",
  "10.133.0.24", "10.133.0.25", "10.133.0.26", "10.133.0.27", "10.133.0.28", "10.133.0.29",
  "10.133.0.30", "10.133.0.31", "10.133.0.32", "10.133.0.33", "10.133.0.34", "10.133.0.35",
  "10.133.0.36", "10.133.0.37", "10.133.0.38", "10.133.0.39", "10.133.0.40", "10.133.0.41",
  "10.133.0.42", "10.133.0.44", "10.133.0.45", "10.133.0.46", "10.133.0.47", "10.133.0.48",
  "10.133.0.5", "10.133.0.6", "10.133.0.7", "10.133.0.8", "10.133.0.9", "10.134.0.10",
  "10.134.0.11", "10.134.0.12", "10.134.0.13", "10.134.0.14", "10.134.0.15", "10.134.0.16",
  "10.134.0.17", "10.134.0.18", "10.134.0.19", "10.134.0.20", "10.134.0.21", "10.134.0.22",
  "10.134.0.23", "10.134.0.24", "10.134.0.25", "10.134.0.26", "10.134.0.27", "10.134.0.28",
  "10.134.0.30", "10.134.0.31", "10.134.0.32", "10.134.0.33", "10.134.0.34", "10.134.0.35",
  "10.134.0.36", "10.134.0.37", "10.134.0.38", "10.134.0.4", "10.134.0.42", "10.134.0.9",
  "10.135.0.10", "10.135.0.11", "10.135.0.12", "10.135.0.13", "10.135.0.14", "10.135.0.15",
  "10.135.0.16", "10.135.0.17", "10.135.0.18", "10.135.0.19", "10.135.0.23", "10.135.0.24",
  "10.135.0.26", "10.135.0.27", "10.135.0.29", "10.135.0.3", "10.135.0.4", "10.135.0.40",
  "10.135.0.41", "10.135.0.42", "10.135.0.43", "10.135.0.44", "10.135.0.5", "10.135.0.6",
  "10.135.0.7", "10.135.0.8", "10.135.0.9"]

  external_ids : {direction=Egress, gress-index="2", ip-family=v4,
  "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
  control:Egress:2:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
  controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}

```

name : a18396736153283155648

\_uuid : 5228bf1b-dfd8-40ec-bfa8-95c5bf9aded9

addresses : []

external\_ids : {direction=Ingress, gress-index="0", ip-family=v4, "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-control:Ingress:0:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}

name : a14545668191619617708

\_uuid : 46530d69-70da-4558-8c63-884ec9dc4f25

addresses : ["10.132.2.10", "10.132.2.5", "10.132.2.6", "10.132.2.7", "10.132.2.8", "10.132.2.9", "10.133.0.47", "10.134.0.33", "10.135.0.10", "10.135.0.11", "10.135.0.12", "10.135.0.19", "10.135.0.24", "10.135.0.7", "10.135.0.8", "10.135.0.9"]

external\_ids : {direction=Ingress, gress-index="1", ip-family=v4, "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-control:Ingress:1:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}

name : a6786643370959569281

\_uuid : 65fdcdea-0b9f-4318-9884-1b51d231ad1d

addresses : ["10.132.3.72", "10.135.0.42"]

external\_ids : {direction=Ingress, gress-index="2", ip-family=v4, "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-control:Ingress:2:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}

name : a13730899355151937870

\_uuid : 73eabdb0-36bf-4ca3-b66d-156ac710df4c

addresses : ["10.0.32.0/19", "10.0.56.38/32", "10.0.69.0/24", "10.132.3.72", "10.135.0.42", "172.29.0.0/30", "192.168.194.103", "192.168.194.2"]

external\_ids : {direction=Egress, gress-index="3", ip-family=v4, "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-control:Egress:3:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}

name : a10622494091691694581

\_uuid : 50cdbef2-71b5-474b-914c-6fcfd1d7712d3

addresses : ["10.132.0.10", "10.132.0.11", "10.132.0.12", "10.132.0.13", "10.132.0.14", "10.132.0.15", "10.132.0.16", "10.132.0.17", "10.132.0.5", "10.132.0.7", "10.132.0.71", "10.132.0.75", "10.132.0.8", "10.132.0.81", "10.132.0.9", "10.132.2.10", "10.132.2.11", "10.132.2.12", "10.132.2.14", "10.132.2.15", "10.132.2.3", "10.132.2.4", "10.132.2.5", "10.132.2.6", "10.132.2.7", "10.132.2.8", "10.132.2.9", "10.132.3.64", "10.132.3.65", "10.132.3.72", "10.132.3.73", "10.132.3.76", "10.133.0.10", "10.133.0.11", "10.133.0.12", "10.133.0.13", "10.133.0.14", "10.133.0.15", "10.133.0.16", "10.133.0.17", "10.133.0.18", "10.133.0.19", "10.133.0.20", "10.133.0.21", "10.133.0.22", "10.133.0.23", "10.133.0.24", "10.133.0.25", "10.133.0.26", "10.133.0.27", "10.133.0.28", "10.133.0.29", "10.133.0.30", "10.133.0.31", "10.133.0.32", "10.133.0.33", "10.133.0.34", "10.133.0.35", "10.133.0.36", "10.133.0.37", "10.133.0.38", "10.133.0.39", "10.133.0.40", "10.133.0.41", "10.133.0.42", "10.133.0.44", "10.133.0.45", "10.133.0.46", "10.133.0.47", "10.133.0.48", "10.133.0.5", "10.133.0.6", "10.133.0.7", "10.133.0.8", "10.133.0.9", "10.134.0.10", "10.134.0.11", "10.134.0.12", "10.134.0.13", "10.134.0.14", "10.134.0.15", "10.134.0.16", "10.134.0.17", "10.134.0.18", "10.134.0.19", "10.134.0.20", "10.134.0.21", "10.134.0.22", "10.134.0.23", "10.134.0.24", "10.134.0.25", "10.134.0.26", "10.134.0.27", "10.134.0.28", "10.134.0.30", "10.134.0.31", "10.134.0.32", "10.134.0.33", "10.134.0.34", "10.134.0.35", "10.134.0.36", "10.134.0.37", "10.134.0.38", "10.134.0.4", "10.134.0.42", "10.134.0.9"]

```

    "10.135.0.10", "10.135.0.11", "10.135.0.12", "10.135.0.13", "10.135.0.14", "10.135.0.15",
    "10.135.0.16", "10.135.0.17", "10.135.0.18", "10.135.0.19", "10.135.0.23", "10.135.0.24",
    "10.135.0.26", "10.135.0.27", "10.135.0.29", "10.135.0.3", "10.135.0.4", "10.135.0.40",
    "10.135.0.41", "10.135.0.42", "10.135.0.43", "10.135.0.44", "10.135.0.5", "10.135.0.6",
    "10.135.0.7", "10.135.0.8", "10.135.0.9"]
  external_ids : {direction=Egress, gress-index="0", ip-family=v4,
  "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
  control:Egress:0:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
  controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}
  name : a13517855690389298082

  _uuid : 32a42f32-2d11-43dd-979d-a56d7ee6aa57
  addresses : ["10.132.3.76", "10.135.0.44"]
  external_ids : {direction=Ingress, gress-index="3", ip-family=v4,
  "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
  control:Ingress:3:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
  controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}
  name : a764182844364804195

  _uuid : 8fd3b977-6e1c-47aa-82b7-e3e3136c4a72
  addresses : ["0.0.0.0/0"]
  external_ids : {direction=Egress, gress-index="5", ip-family=v4,
  "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
  control:Egress:5:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
  controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}
  name : a11452480169090787059

```

- a. Examine the specific address set of the rule by running the following command:

```
$ ovn-nbctl find Address_Set 'external_ids{>=}{"k8s.ovn.org/owner-
type"=AdminNetworkPolicy,direction=Egress,"k8s.ovn.org/name"=cluster-control,gress-
index="5"}'
```

#### Example 5.17. Example outputs forAddress\_Set

```

  _uuid : 8fd3b977-6e1c-47aa-82b7-e3e3136c4a72
  addresses : ["0.0.0.0/0"]
  external_ids : {direction=Egress, gress-index="5", ip-family=v4,
  "k8s.ovn.org/id"="default-network-controller:AdminNetworkPolicy:cluster-
  control:Egress:5:v4", "k8s.ovn.org/name"=cluster-control, "k8s.ovn.org/owner-
  controller"=default-network-controller, "k8s.ovn.org/owner-type"=AdminNetworkPolicy}
  name : a11452480169090787059

```

5. Run the following command to look at the port groups in the nbdb:

```
$ ovn-nbctl find Port_Group 'external_ids{>=}{"k8s.ovn.org/owner-
type"=AdminNetworkPolicy,"k8s.ovn.org/name"=cluster-control}'
```

#### Example 5.18. Example outputs forPort\_Group

```
_uuid : f50acf71-7488-4b9a-b7b8-c8a024e99d21
```

```
acls      : [04f20275-c410-405c-a923-0e677f767889, 0d5e4722-b608-4bb1-b625-23c323cc9926, 1a27d30e-3f96-4915-8ddd-ade7f22c117b, 1a68a5ed-e7f9-47d0-b55c-89184d97e81a, 4b5d836a-e0a3-4088-825e-f9f0ca58e538, 5a6e5bb4-36eb-4209-b8bc-c611983d4624, 5d09957d-d2cc-4f5a-9ddd-b97d9d772023, aa1a224d-7960-4952-bdfb-35246bafbac8, b23a087f-08f8-4225-8c27-4a9a9ee0c407, b7be6472-df67-439c-8c9c-f55929f0a6e0, d14ed5cf-2e06-496e-8cae-6b76d5dd5cccd]
external_ids : {"k8s.ovn.org/id": "default-network-controller:AdminNetworkPolicy:cluster-control", "k8s.ovn.org/name": "cluster-control", "k8s.ovn.org/owner-controller": "default-network-controller", "k8s.ovn.org/owner-type": "AdminNetworkPolicy"}
name       : a14645450421485494999
ports      : [5e75f289-8273-4f8a-8798-8c10f7318833, de7e1b71-6184-445d-93e7-b20acadf41ea]
```

### 5.2.5.2. Additional resources

- [Tracing Openflow with ovnkube-trace](#)
- [Troubleshooting OVN-Kubernetes](#)

### 5.2.6. Best practices for AdminNetworkPolicy

This section provides best practices for the **AdminNetworkPolicy** and **BaselineAdminNetworkPolicy** resources.

#### 5.2.6.1. Designing AdminNetworkPolicy

When building **AdminNetworkPolicy** (ANP) resources, you might consider the following when creating your policies:

- You can create ANPs that have the same priority. If you do create two ANPs at the same priority, ensure that they do not apply overlapping rules to the same traffic. Only one rule per value is applied and there is no guarantee which rule is applied when there is more than one at the same priority value. Because there is no guarantee which policy takes precedence when overlapping ANPs are created, set ANPs at different priorities so that precedence is well defined.
- Administrators must create ANP that apply to user namespaces not system namespaces.



#### IMPORTANT

Applying ANP and **BaselineAdminNetworkPolicy** (BANP) to system namespaces (**default**, **kube-system**, any namespace whose name starts with **openshift**, etc) is not supported, and this can leave your cluster unresponsive and in a non-functional state.

- Because **0-100** is the supported priority range, you might design your ANP to use a middle range like **30-70**. This leaves some placeholder for priorities before and after. Even in the middle range, you might want to leave gaps so that as your infrastructure requirements evolve over time, you are able to insert new ANPs when needed at the right priority level. If you pack your ANPs, then you might need to recreate all of them to accommodate any changes in the future.
- When using **0.0.0.0/0** or **::/0** to create a strong **Deny** policy, ensure that you have higher priority **Allow** or **Pass** rules for essential traffic.

- Use **Allow** as your **action** field when you want to ensure that a connection is allowed no matter what. An **Allow** rule in an ANP means that the connection will always be allowed, and **NetworkPolicy** will be ignored.
- Use **Pass** as your **action** field to delegate the policy decision of allowing or denying the connection to the **NetworkPolicy** layer.
- Ensure that the selectors across multiple rules do not overlap so that the same IPs do not appear in multiple policies, which can cause performance and scale limitations.
- Avoid using **namedPorts** in conjunction with **PortNumber** and **PortRange** because this creates 6 ACLs and cause inefficiencies in your cluster.

#### 5.2.6.1.1. Considerations for using BaselineAdminNetworkPolicy

- You can define only a single **BaselineAdminNetworkPolicy** (BANP) resource within a cluster. The following are supported uses for BANP that administrators might consider in designing their BANP:
  - You can set a default deny policy for cluster-local ingress in user namespaces. This BANP will force developers to have to add **NetworkPolicy** objects to allow the ingress traffic that they want to allow, and if they do not add network policies for ingress it will be denied.
  - You can set a default deny policy for cluster-local egress in user namespaces. This BANP will force developers to have to add **NetworkPolicy** objects to allow the egress traffic that they want to allow, and if they do not add network policies it will be denied.
  - You can set a default allow policy for egress to the in-cluster DNS service. Such a BANP ensures that the namespaced users do not have to set an allow egress **NetworkPolicy** to the in-cluster DNS service.
  - You can set an egress policy that allows internal egress traffic to all pods but denies access to all external endpoints (i.e **0.0.0.0/0** and **::/0**). This BANP allows user workloads to send traffic to other in-cluster endpoints, but not to external endpoints by default. **NetworkPolicy** can then be used by developers in order to allow their applications to send traffic to an explicit set of external services.
- Ensure you scope your BANP so that it only denies traffic to user namespaces and not to system namespaces. This is because the system namespaces do not have **NetworkPolicy** objects to override your BANP.

#### 5.2.6.1.2. Differences to consider between AdminNetworkPolicy and NetworkPolicy

- Unlike **NetworkPolicy** objects, you must use explicit labels to reference your workloads within ANP and BANP rather than using the empty `({})` catch all selector to avoid accidental traffic selection.



#### IMPORTANT

An empty namespace selector applied to a infrastructure namespace can make your cluster unresponsive and in a non-functional state.

- In API semantics for ANP, you have to explicitly define allow or deny rules when you create the policy, unlike **NetworkPolicy** objects which have an implicit deny.

- Unlike **NetworkPolicy** objects, **AdminNetworkPolicy** objects ingress rules are limited to in-cluster pods and namespaces so you cannot, and do not need to, set rules for ingress from the host network.

## 5.3. NETWORK POLICY

### 5.3.1. About network policy

As a developer, you can define network policies that restrict traffic to pods in your cluster.

#### 5.3.1.1. About network policy

By default, all pods in a project are accessible from other pods and network endpoints. To isolate one or more pods in a project, you can create **NetworkPolicy** objects in that project to indicate the allowed incoming connections. Project administrators can create and delete **NetworkPolicy** objects within their own project.

If a pod is matched by selectors in one or more **NetworkPolicy** objects, then the pod will accept only connections that are allowed by at least one of those **NetworkPolicy** objects. A pod that is not selected by any **NetworkPolicy** objects is fully accessible.

A network policy applies to only the Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Internet Control Message Protocol (ICMP), and Stream Control Transmission Protocol (SCTP) protocols. Other protocols are not affected.



#### WARNING

- A network policy does not apply to the host network namespace. Pods with host networking enabled are unaffected by network policy rules. However, pods connecting to the host-networked pods might be affected by the network policy rules.
- Using the **namespaceSelector** field without the **podSelector** field set to `{}` will not include **hostNetwork** pods. You must use the **podSelector** set to `{}` with the **namespaceSelector** field in order to target **hostNetwork** pods when creating network policies.
- Network policies cannot block traffic from localhost or from their resident nodes.

The following example **NetworkPolicy** objects demonstrate supporting different scenarios:

- Deny all traffic:  
To make a project deny by default, add a **NetworkPolicy** object that matches all pods but accepts no traffic:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
```

```
spec:
podSelector: {}
ingress: []
```

- Only allow connections from the OpenShift Container Platform Ingress Controller: To make a project allow only connections from the OpenShift Container Platform Ingress Controller, add the following **NetworkPolicy** object.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            policy-group.network.openshift.io/ingress: ""
    podSelector: {}
    policyTypes:
      - Ingress
```

- Only accept connections from pods within a project:



### IMPORTANT

To allow ingress connections from **hostNetwork** pods in the same namespace, you need to apply the **allow-from-hostnetwork** policy together with the **allow-same-namespace** policy.

To make pods accept connections from other pods in the same project, but reject all other connections from pods in other projects, add the following **NetworkPolicy** object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector: {}
  ingress:
    - from:
      - podSelector: {}
```

- Only allow HTTP and HTTPS traffic based on pod labels: To enable only HTTP and HTTPS access to the pods with a specific label (**role=frontend** in following example), add a **NetworkPolicy** object similar to the following:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-http-and-https
spec:
  podSelector:
    matchLabels:
```

```

role: frontend
ingress:
- ports:
  - protocol: TCP
    port: 80
  - protocol: TCP
    port: 443

```

- Accept connections by using both namespace and pod selectors:

To match network traffic by combining namespace and pod selectors, you can use a **NetworkPolicy** object similar to the following:

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-pod-and-namespace-both
spec:
  podSelector:
    matchLabels:
      name: test-pods
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            project: project_name
      podSelector:
        matchLabels:
          name: test-pods

```

**NetworkPolicy** objects are additive, which means you can combine multiple **NetworkPolicy** objects together to satisfy complex network requirements.

For example, for the **NetworkPolicy** objects defined in previous samples, you can define both **allow-same-namespace** and **allow-http-and-https** policies within the same project. Thus allowing the pods with the label **role=frontend**, to accept any connection allowed by each policy. That is, connections on any port from pods in the same namespace, and connections on ports **80** and **443** from pods in any namespace.

#### 5.3.1.1. Using the allow-from-router network policy

Use the following **NetworkPolicy** to allow external traffic regardless of the router configuration:

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            policy-group.network.openshift.io/ingress: ""1
  podSelector: {}
  policyTypes:
    - Ingress

```

- 1 policy-group.network.openshift.io/ingress:''' label supports OVN-Kubernetes.

### 5.3.1.1.2. Using the allow-from-hostnetwork network policy

Add the following **allow-from-hostnetwork NetworkPolicy** object to direct traffic from the host network pods.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-hostnetwork
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/host-network: ""
  podSelector: {}
  policyTypes:
  - Ingress
```

### 5.3.1.2. Optimizations for network policy with OVN-Kubernetes network plugin

When designing your network policy, refer to the following guidelines:

- For network policies with the same **spec.podSelector** spec, it is more efficient to use one network policy with multiple **ingress** or **egress** rules, than multiple network policies with subsets of **ingress** or **egress** rules.
- Every **ingress** or **egress** rule based on the **podSelector** or **namespaceSelector** spec generates the number of OVS flows proportional to **number of pods selected by network policy + number of pods selected by ingress or egress rule**. Therefore, it is preferable to use the **podSelector** or **namespaceSelector** spec that can select as many pods as you need in one rule, instead of creating individual rules for every pod.

For example, the following policy contains two rules:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
  - from:
    - podSelector:
        matchLabels:
          role: backend
```

The following policy expresses those same two rules as one:

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchExpressions:
        - {key: role, operator: In, values: [frontend, backend]}

```

The same guideline applies to the **spec.podSelector** spec. If you have the same **ingress** or **egress** rules for different network policies, it might be more efficient to create one network policy with a common **spec.podSelector** spec. For example, the following two policies have different rules:

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy1
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy2
spec:
  podSelector:
    matchLabels:
      role: client
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend

```

The following network policy expresses those same two rules as one:

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy3
spec:
  podSelector:
    matchExpressions:
    - {key: role, operator: In, values: [db, client]}

```

```

  ingress:
    - from:
      - podSelector:
          matchLabels:
            role: frontend

```

You can apply this optimization when only multiple selectors are expressed as one. In cases where selectors are based on different labels, it may not be possible to apply this optimization. In those cases, consider applying some new labels for network policy optimization specifically.

#### 5.3.1.2.1. NetworkPolicy CR and external IPs in OVN-Kubernetes

In OVN-Kubernetes, the **NetworkPolicy** custom resource (CR) enforces strict isolation rules. If a service is exposed using an external IP, a network policy can block access from other namespaces unless explicitly configured to allow traffic.

To allow access to external IPs across namespaces, create a **NetworkPolicy** CR that explicitly permits ingress from the required namespaces and ensures traffic is allowed to the designated service ports. Without allowing traffic to the required ports, access might still be restricted.

#### Example output

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  annotations:
    name: <policy_name>
  namespace: openshift-ingress
spec:
  ingress:
    - ports:
        - port: 80
          protocol: TCP
    - ports:
        - port: 443
          protocol: TCP
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: <my_namespace>
  podSelector: {}
  policyTypes:
    - Ingress

```

where:

**<policy\_name>**

Specifies your name for the policy.

**<my\_namespace>**

Specifies the name of the namespace where the policy is deployed.

For more details, see "About network policy".

#### 5.3.1.3. Next steps

- [Creating a network policy](#)
- Optional: [Defining a default network policy for projects](#)

#### 5.3.1.4. Additional resources

- [Projects and namespaces](#)
- [Configuring multitenant isolation with network policy](#)
- [NetworkPolicy API](#)

### 5.3.2. Creating a network policy

As a user with the **admin** role, you can create a network policy for a namespace.

#### 5.3.2.1. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ①
spec:
  podSelector: ②
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: ③
      matchLabels:
        app: app
    ports: ④
    - protocol: TCP
      port: 27017
```

- ① The name of the NetworkPolicy object.
- ② A selector that describes the pods to which the policy applies. The policy object can only select pods in the project that defines the NetworkPolicy object.
- ③ A selector that matches the pods from which the policy object allows ingress traffic. The selector matches pods in the same namespace as the NetworkPolicy.
- ④ A list of one or more destination ports on which to accept traffic.

#### 5.3.2.2. Creating a network policy using the CLI

To define granular rules describing ingress or egress network traffic allowed for namespaces in your cluster, you can create a network policy.



## NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

## Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace that the network policy applies to.

## Procedure

1. Create a policy rule:
  - a. Create a **<policy\_name>.yaml** file:

```
$ touch <policy_name>.yaml
```

where:

**<policy\_name>**

Specifies the network policy file name.

- b. Define a network policy in the file that you just created, such as in the following examples:

### Deny ingress from all pods in all namespaces

This is a fundamental policy, blocking all cross-pod networking other than cross-pod traffic allowed by the configuration of other Network Policies.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress: []
```

### Allow ingress from all pods in the same namespace

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
```

```
    ingress:
      - from:
        - podSelector: {}
```

#### Allow ingress traffic to one pod from a particular namespace

This policy allows traffic to pods labelled **pod-a** from pods running in **namespace-y**.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-traffic-pod
spec:
  podSelector:
    matchLabels:
      pod: pod-a
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: namespace-y
```

- To create the network policy object, enter the following command:

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

where:

**<policy\_name>**

Specifies the network policy file name.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

#### Example output

```
networkpolicy.networking.k8s.io/deny-by-default created
```



#### NOTE

If you log in to the web console with **cluster-admin** privileges, you have a choice of creating a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

#### 5.3.2.3. Creating a default deny all network policy

This policy blocks all cross-pod networking other than network traffic allowed by the configuration of other deployed network policies and traffic between host-networked pods. This procedure enforces a strong deny policy by applying a **deny-by-default** policy in the **my-project** namespace.

**WARNING**

Without configuring a **NetworkPolicy** custom resource (CR) that allows traffic communication, the following policy might cause communication problems across your cluster.

**Prerequisites**

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace that the network policy applies to.

**Procedure**

1. Create the following YAML that defines a **deny-by-default** policy to deny ingress from all pods in all namespaces. Save the YAML in the **deny-by-default.yaml** file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: my-project 1
spec:
  podSelector: {} 2
  ingress: [] 3
```

- 1** **Specifies the namespace in which to deploy the policy. For example, the `my-project` namespace.**
- 2** If this field is empty, the configuration matches all the pods. Therefore, the policy applies to all pods in the **my-project** namespace.
- 3** There are no **ingress** rules specified. This causes incoming traffic to be dropped to all pods.

2. Apply the policy by entering the following command:

```
$ oc apply -f deny-by-default.yaml
```

**Example output**

```
networkpolicy.networking.k8s.io/deny-by-default created
```

### 5.3.2.4. Creating a network policy to allow traffic from external clients

With the **deny-by-default** policy in place you can proceed to configure a policy that allows traffic from external clients to a pod with the label **app=web**.



#### NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Follow this procedure to configure a policy that allows external service from the public Internet directly or by using a Load Balancer to access the pod. Traffic is only allowed to a pod with the label **app=web**.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace that the network policy applies to.

#### Procedure

1. Create a policy that allows traffic from the public Internet directly or by using a load balancer to access the pod. Save the YAML in the **web-allow-external.yaml** file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-external
  namespace: default
spec:
  policyTypes:
    - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

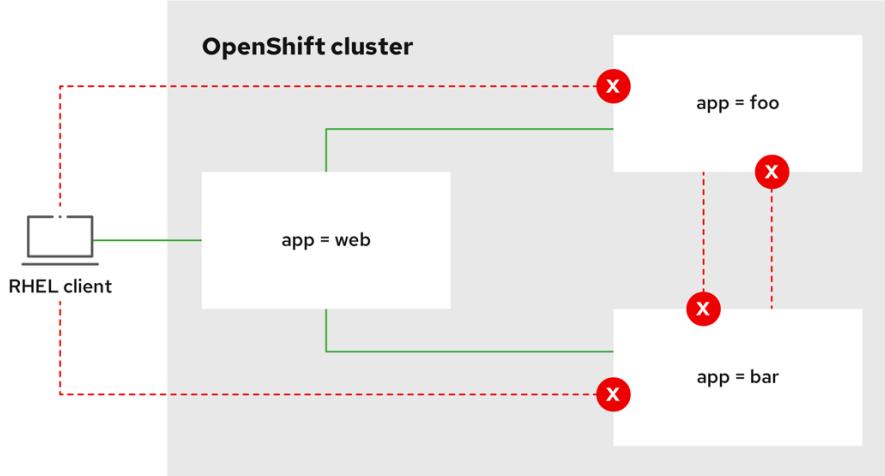
2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-external.yaml
```

#### Example output

```
networkpolicy.networking.k8s.io/web-allow-external created
```

This policy allows traffic from all resources, including external traffic as illustrated in the following diagram:



292\_OpenShift\_1122

### 5.3.2.5. Creating a network policy allowing traffic to an application from all namespaces



#### NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Follow this procedure to configure a policy that allows traffic from all pods in all namespaces to a particular application.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace that the network policy applies to.

#### Procedure

1. Create a policy that allows traffic from all pods in all namespaces to a particular application. Save the YAML in the **web-allow-all-namespaces.yaml** file:

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all-namespaces
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1
  policyTypes:
    - Ingress

```

```
ingress:
- from:
  - namespaceSelector: {} 2
```

- 1** Applies the policy only to **app:web** pods in default namespace.
- 2** Selects all pods in all namespaces.



### NOTE

By default, if you omit specifying a **namespaceSelector** it does not select any namespaces, which means the policy allows traffic only from the namespace the network policy is deployed to.

2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-all-namespaces.yaml
```

### Example output

```
networkpolicy.networking.k8s.io/web-allow-all-namespaces created
```

## Verification

1. Start a web service in the **default** namespace by entering the following command:

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```

2. Run the following command to deploy an **alpine** image in the **secondary** namespace and to start a shell:

```
$ oc run test-$RANDOM --namespace=secondary --rm -i -t --image=alpine -- sh
```

3. Run the following command in the shell and observe that the request is allowed:

```
# wget -qO- --timeout=2 http://web.default
```

### Expected output

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
```

```

<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

```

### 5.3.2.6. Creating a network policy allowing traffic to an application from a namespace



#### NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Follow this procedure to configure a policy that allows traffic to a pod with the label **app=web** from a particular namespace. You might want to do this to:

- Restrict traffic to a production database only to namespaces where production workloads are deployed.
- Enable monitoring tools deployed to a particular namespace to scrape metrics from the current namespace.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace that the network policy applies to.

#### Procedure

1. Create a policy that allows traffic from all pods in a particular namespaces with a label **purpose=production**. Save the YAML in the **web-allow-prod.yaml** file:

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1

```

```

policyTypes:
- Ingress
ingress:
- from:
  - namespaceSelector:
    matchLabels:
      purpose: production 2

```

- 1** Applies the policy only to **app:web** pods in the default namespace.
- 2** Restricts traffic to only pods in namespaces that have the label **purpose=production**.

2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-prod.yaml
```

#### Example output

```
networkpolicy.networking.k8s.io/web-allow-prod created
```

### Verification

1. Start a web service in the **default** namespace by entering the following command:
 

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```
2. Run the following command to create the **prod** namespace:
 

```
$ oc create namespace prod
```
3. Run the following command to label the **prod** namespace:
 

```
$ oc label namespace/prod purpose=production
```
4. Run the following command to create the **dev** namespace:
 

```
$ oc create namespace dev
```
5. Run the following command to label the **dev** namespace:
 

```
$ oc label namespace/dev purpose=testing
```
6. Run the following command to deploy an **alpine** image in the **dev** namespace and to start a shell:
 

```
$ oc run test-$RANDOM --namespace=dev --rm -i -t --image=alpine -- sh
```
7. Run the following command in the shell and observe that the request is blocked:
 

```
# wget -qO- --timeout=2 http://web.default
```

### Expected output

```
wget: download timed out
```

8. Run the following command to deploy an **alpine** image in the **prod** namespace and start a shell:

```
$ oc run test-$RANDOM --namespace=prod --rm -i -t --image=alpine -- sh
```

9. Run the following command in the shell and observe that the request is allowed:

```
# wget -qO- --timeout=2 http://web.default
```

### Expected output

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

#### 5.3.2.7. Additional resources

- [Accessing the web console](#)
- [Logging for egress firewall and network policy rules](#)

#### 5.3.3. Viewing a network policy

As a user with the **admin** role, you can view a network policy for a namespace.

##### 5.3.3.1. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ①
spec:
  podSelector: ②
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: ③
      matchLabels:
        app: app
    ports: ④
    - protocol: TCP
      port: 27017

```

- ① The name of the NetworkPolicy object.
- ② A selector that describes the pods to which the policy applies. The policy object can only select pods in the project that defines the NetworkPolicy object.
- ③ A selector that matches the pods from which the policy object allows ingress traffic. The selector matches pods in the same namespace as the NetworkPolicy.
- ④ A list of one or more destination ports on which to accept traffic.

### 5.3.3.2. Viewing network policies using the CLI

You can examine the network policies in a namespace.



#### NOTE

If you log in with a user with the **cluster-admin** role, then you can view any network policy in the cluster.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace where the network policy exists.

#### Procedure

- List network policies in a namespace:
  - To view network policy objects defined in a namespace, enter the following command:
 

```
$ oc get networkpolicy
```
  - Optional: To examine a specific network policy, enter the following command:

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

where:

**<policy\_name>**

Specifies the name of the network policy to inspect.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

For example:

```
$ oc describe networkpolicy allow-same-namespace
```

### Output for **oc describe** command

```
Name:      allow-same-namespace
Namespace: ns1
Created on: 2021-05-24 22:28:56 -0400 EDT
Labels:    <none>
Annotations: <none>
Spec:
PodSelector: <none> (Allowing the specific traffic to all pods in this namespace)
Allowing ingress traffic:
To Port: <any> (traffic allowed to all ports)
From:
PodSelector: <none>
Not affecting egress traffic
Policy Types: Ingress
```



#### NOTE

If you log in to the web console with **cluster-admin** privileges, you have a choice of viewing a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

### 5.3.4. Editing a network policy

As a user with the **admin** role, you can edit an existing network policy for a namespace.

#### 5.3.4.1. Editing a network policy

You can edit a network policy in a namespace.



#### NOTE

If you log in with a user with the **cluster-admin** role, then you can edit a network policy in any namespace in the cluster.

## Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace where the network policy exists.

## Procedure

1. Optional: To list the network policy objects in a namespace, enter the following command:

```
$ oc get networkpolicy
```

where:

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

2. Edit the network policy object.

- If you saved the network policy definition in a file, edit the file and make any necessary changes, and then enter the following command:

```
$ oc apply -n <namespace> -f <policy_file>.yaml
```

where:

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

**<policy\_file>**

Specifies the name of the file containing the network policy.

- If you need to update the network policy object directly, enter the following command:

```
$ oc edit networkpolicy <policy_name> -n <namespace>
```

where:

**<policy\_name>**

Specifies the name of the network policy.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

3. Confirm that the network policy object is updated.

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

where:

**<policy\_name>**

Specifies the name of the network policy.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

**NOTE**

If you log in to the web console with **cluster-admin** privileges, you have a choice of editing a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

### 5.3.4.2. Example NetworkPolicy object

The following annotates an example NetworkPolicy object:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ①
spec:
  podSelector: ②
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: ③
      matchLabels:
        app: app
    ports: ④
    - protocol: TCP
      port: 27017
```

① The name of the NetworkPolicy object.

② A selector that describes the pods to which the policy applies. The policy object can only select pods in the project that defines the NetworkPolicy object.

③ A selector that matches the pods from which the policy object allows ingress traffic. The selector matches pods in the same namespace as the NetworkPolicy.

④ A list of one or more destination ports on which to accept traffic.

### 5.3.4.3. Additional resources

- [Creating a network policy](#)

### 5.3.5. Deleting a network policy

As a user with the **admin** role, you can delete a network policy from a namespace.

#### 5.3.5.1. Deleting a network policy using the CLI

You can delete a network policy in a namespace.



### NOTE

If you log in with a user with the **cluster-admin** role, then you can delete any network policy in the cluster.

### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **admin** privileges.
- You are working in the namespace where the network policy exists.

### Procedure

- To delete a network policy object, enter the following command:

```
$ oc delete networkpolicy <policy_name> -n <namespace>
```

where:

**<policy\_name>**

Specifies the name of the network policy.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

### Example output

```
networkpolicy.networking.k8s.io/default-deny deleted
```



### NOTE

If you log in to the web console with **cluster-admin** privileges, you have a choice of deleting a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

## 5.3.6. Defining a default network policy for projects

As a cluster administrator, you can modify the new project template to automatically include network policies when you create a new project. If you do not yet have a customized template for new projects, you must first create one.

### 5.3.6.1. Modifying the template for new projects

As a cluster administrator, you can modify the default project template so that new projects are created using your custom requirements.

To create your own custom project template:

## Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

## Procedure

1. Log in as a user with **cluster-admin** privileges.

2. Generate the default project template:

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. Use a text editor to modify the generated **template.yaml** file by adding objects or modifying existing objects.

4. The project template must be created in the **openshift-config** namespace. Load your modified template:

```
$ oc create -f template.yaml -n openshift-config
```

5. Edit the project configuration resource using the web console or CLI.

- Using the web console:

- i. Navigate to the **Administration** → **Cluster Settings** page.
- ii. Click **Configuration** to view all configuration resources.
- iii. Find the entry for **Project** and click **Edit YAML**.

- Using the CLI:

- i. Edit the **project.config.openshift.io/cluster** resource:

```
$ oc edit project.config.openshift.io/cluster
```

6. Update the **spec** section to include the **projectRequestTemplate** and **name** parameters, and set the name of your uploaded project template. The default name is **project-request**.

## Project configuration resource with custom project template

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
# ...
spec:
  projectRequestTemplate:
    name: <template_name>
# ...
```

7. After you save your changes, create a new project to verify that your changes were successfully applied.

### 5.3.6.2. Adding network policies to the new project template

As a cluster administrator, you can add network policies to the default template for new projects. OpenShift Container Platform will automatically create all the **NetworkPolicy** objects specified in the template in the project.

#### Prerequisites

- Your cluster uses a default CNI network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes.
- You installed the OpenShift CLI (**oc**).
- You must log in to the cluster with a user with **cluster-admin** privileges.
- You must have created a custom default project template for new projects.

#### Procedure

1. Edit the default template for a new project by running the following command:

```
$ oc edit template <project_template> -n openshift-config
```

Replace **<project\_template>** with the name of the default template that you configured for your cluster. The default template name is **project-request**.

2. In the template, add each **NetworkPolicy** object as an element to the **objects** parameter. The **objects** parameter accepts a collection of one or more objects.  
In the following example, the **objects** parameter collection includes several **NetworkPolicy** objects.

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector: {}
  ingress:
    - from:
        - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
      - from:
          - namespaceSelector:
              matchLabels:
                policy-group.network.openshift.io/ingress:
      podSelector: {}
    policyTypes:
      - Ingress
- apiVersion: networking.k8s.io/v1
```

```

kind: NetworkPolicy
metadata:
  name: allow-from-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-kube-apiserver-operator
    podSelector:
      matchLabels:
        app: kube-apiserver-operator
  policyTypes:
  - Ingress
...

```

3. Optional: Create a new project to confirm that your network policy objects are created successfully by running the following commands:

- a. Create a new project:

```
$ oc new-project <project> ①
```

① Replace **<project>** with the name for the project you are creating.

- b. Confirm that the network policy objects in the new project template exist in the new project:

```
$ oc get networkpolicy
NAME                  POD-SELECTOR  AGE
allow-from-openshift-ingress  <none>     7s
allow-from-same-namespace   <none>     7s
```

### 5.3.7. Configuring multitenant isolation with network policy

As a cluster administrator, you can configure your network policies to provide multitenant network isolation.



#### NOTE

Configuring network policies as described in this section provides network isolation similar to the multitenant mode of OpenShift SDN in previous versions of OpenShift Container Platform.

#### 5.3.7.1. Configuring multitenant isolation by using network policy

You can configure your project to isolate it from pods and services in other project namespaces.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).

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

## Procedure

1. Create the following **NetworkPolicy** objects:
  - a. A policy named **allow-from-openshift-ingress**.

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: ""
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```



### NOTE

**policy-group.network.openshift.io/ingress: ""** is the preferred namespace selector label for OVN-Kubernetes.

- b. A policy named **allow-from-openshift-monitoring**:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-monitoring
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          network.openshift.io/policy-group: monitoring
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

- c. A policy named **allow-same-namespace**:

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
```

```

spec:
podSelector:
ingress:
- from:
  - podSelector: {}
EOF

```

- d. A policy named **allow-from-kube-apiserver-operator**:

```

$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-kube-apiserver-operator
    podSelector:
      matchLabels:
        app: kube-apiserver-operator
  policyTypes:
  - Ingress
EOF

```

For more details, see [New \*\*kube-apiserver-operator\*\* webhook controller validating health of webhook](#).

2. Optional: To confirm that the network policies exist in your current project, enter the following command:

```
$ oc describe networkpolicy
```

### Example output

```

Name:      allow-from-openshift-ingress
Namespace: example1
Created on: 2020-06-09 00:28:17 -0400 EDT
Labels:    <none>
Annotations: <none>
Spec:
PodSelector: <none> (Allowing the specific traffic to all pods in this namespace)
Allowing ingress traffic:
  To Port: <any> (traffic allowed to all ports)
  From:
    NamespaceSelector: policy-group.network.openshift.io/ingress:
Not affecting egress traffic
Policy Types: Ingress

```

```

Name:      allow-from-openshift-monitoring
Namespace: example1
Created on: 2020-06-09 00:29:57 -0400 EDT

```

```

Labels:    <none>
Annotations: <none>
Spec:
  PodSelector:  <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: monitoring
    Not affecting egress traffic
  Policy Types: Ingress

```

### 5.3.7.2. Next steps

- Defining a default network policy for a project

## 5.4. AUDIT LOGGING FOR NETWORK SECURITY

The OVN-Kubernetes network plugin uses Open Virtual Network (OVN) access control lists (ACLs) to manage **AdminNetworkPolicy**, **BaselineAdminNetworkPolicy**, **NetworkPolicy**, and **EgressFirewall** objects. Audit logging exposes **allow** and **deny** ACL events for **NetworkPolicy**, **EgressFirewall** and **BaselineAdminNetworkPolicy** custom resources (CR). Logging also exposes **allow**, **deny**, and **pass** ACL events for **AdminNetworkPolicy** (ANP) CR.



### NOTE

Audit logging is available for only the [OVN-Kubernetes network plugin](#).

### 5.4.1. Audit configuration

The configuration for audit logging is specified as part of the OVN-Kubernetes cluster network provider configuration. The following YAML illustrates the default values for the audit logging:

#### Audit logging configuration

```

apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
        syslogFacility: local0

```

The following table describes the configuration fields for audit logging.

**Table 5.1. policyAuditConfig object**

Field	Type	Description
<b>rateLimit</b>	integer	The maximum number of messages to generate every second per node. The default value is <b>20</b> messages per second.
<b>maxFileSize</b>	integer	The maximum size for the audit log in bytes. The default value is <b>50000000</b> or 50 MB.
<b>maxLogFiles</b>	integer	The maximum number of log files that are retained.
<b>destination</b>	string	<p>One of the following additional audit log targets:</p> <p><b>libc</b> The libc <b>syslog()</b> function of the journald process on the host.</p> <p><b>udp:&lt;host&gt;:&lt;port&gt;</b> A syslog server. Replace <b>&lt;host&gt;:&lt;port&gt;</b> with the host and port of the syslog server.</p> <p><b>unix:&lt;file&gt;</b> A Unix Domain Socket file specified by <b>&lt;file&gt;</b>.</p> <p><b>null</b> Do not send the audit logs to any additional target.</p>
<b>syslogFacility</b>	string	The syslog facility, such as <b>kern</b> , as defined by RFC5424. The default value is <b>local0</b> .

## 5.4.2. Audit logging

You can configure the destination for audit logs, such as a syslog server or a UNIX domain socket. Regardless of any additional configuration, an audit log is always saved to **/var/log/ovn/acl-audit-log.log** on each OVN-Kubernetes pod in the cluster.

You can enable audit logging for each namespace by annotating each namespace configuration with a **k8s.ovn.org/acl-logging** section. In the **k8s.ovn.org/acl-logging** section, you must specify **allow**, **deny**, or both values to enable audit logging for a namespace.



### NOTE

A network policy does not support setting the **Pass** action set as a rule.

The ACL-logging implementation logs access control list (ACL) events for a network. You can view these logs to analyze any potential security issues.

### Example namespace annotation

```
kind: Namespace
apiVersion: v1
metadata:
  name: example1
```

annotations:

```
k8s.ovn.org/acl-logging: |-  
 {  
   "deny": "info",  
   "allow": "info"  
 }
```

To view the default ACL logging configuration values, see the **policyAuditConfig** object in the **cluster-network-03-config.yml** file. If required, you can change the ACL logging configuration values for log file parameters in this file.

The logging message format is compatible with syslog as defined by RFC5424. The syslog facility is configurable and defaults to **local0**. The following example shows key parameters and their values outputted in a log message:

### Example logging message that outputs parameters and their values

```
<timestamp>|<message_serial>|acl_log(ovn_pinctrl0)|<severity>|name=<acl_name>, verdict=<verdict>, severity=<severity>, direction=<direction>: <flow>
```

Where:

- **<timestamp>** states the time and date for the creation of a log message.
- **<message\_serial>** lists the serial number for a log message.
- **acl\_log(ovn\_pinctrl0)** is a literal string that prints the location of the log message in the OVN-Kubernetes plugin.
- **<severity>** sets the severity level for a log message. If you enable audit logging that supports **allow** and **deny** tasks then two severity levels show in the log message output.
- **<name>** states the name of the ACL-logging implementation in the OVN Network Bridging Database (**nbdb**) that was created by the network policy.
- **<verdict>** can be either **allow** or **drop**.
- **<direction>** can be either **to-*I*port** or **from-*I*port** to indicate that the policy was applied to traffic going to or away from a pod.
- **<flow>** shows packet information in a format equivalent to the **OpenFlow** protocol. This parameter comprises Open vSwitch (OVS) fields.

The following example shows OVS fields that the **flow** parameter uses to extract packet information from system memory:

### Example of OVS fields used by the **flow** parameter to extract packet information

```
<proto>,vlan_tci=0x0000,dl_src=<src_mac>,dl_dst=<source_mac>,nw_src=<source_ip>,nw_dst=<target_ip>,nw_tos=<tos_dscp>,nw_ecn=<tos_ecn>,nw_ttl=<ip_ttl>,nw_frag=<fragment>,tp_src=<tcp_src_port>,tp_dst=<tcp_dst_port>,tcp_flags=<tcp_flags>
```

Where:

- **<proto>** states the protocol. Valid values are **tcp** and **udp**.

- **vlan\_tci=0x0000** states the VLAN header as **0** because a VLAN ID is not set for internal pod network traffic.
- **<src\_mac>** specifies the source for the Media Access Control (MAC) address.
- **<source\_mac>** specifies the destination for the MAC address.
- **<source\_ip>** lists the source IP address
- **<target\_ip>** lists the target IP address.
- **<tos\_dscp>** states Differentiated Services Code Point (DSCP) values to classify and prioritize certain network traffic over other traffic.
- **<tos\_ecn>** states Explicit Congestion Notification (ECN) values that indicate any congested traffic in your network.
- **<ip\_ttl>** states the Time To Live (TTP) information for an packet.
- **<fragment>** specifies what type of IP fragments or IP non-fragments to match.
- **<tcp\_src\_port>** shows the source for the port for TCP and UDP protocols.
- **<tcp\_dst\_port>** lists the destination port for TCP and UDP protocols.
- **<tcp\_flags>** supports numerous flags such as **SYN, ACK, PSH** and so on. If you need to set multiple values then each value is separated by a vertical bar (|). The UDP protocol does not support this parameter.



### NOTE

For more information about the previous field descriptions, go to the OVS manual page for **ovs-fields**.

### Example ACL deny log entry for a network policy

```
2023-11-02T16:28:54.139Z|00004|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:Ingress", verdict=drop, severity=alert, direction=to-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=1  
0.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn  
2023-11-02T16:28:55.187Z|00005|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:Ingress", verdict=drop, severity=alert, direction=to-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=1  
0.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn  
2023-11-02T16:28:57.235Z|00006|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:Ingress", verdict=drop, severity=alert, direction=to-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=1  
0.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn
```

The following table describes namespace annotation values:

**Table 5.2. Audit logging namespace annotation for [k8s.ovn.org/acl-logging](https://k8s.ovn.org/acl-logging)**

Field	Description
<b>deny</b>	Blocks namespace access to any traffic that matches an ACL rule with the <b>deny</b> action. The field supports <b>alert</b> , <b>warning</b> , <b>notice</b> , <b>info</b> , or <b>debug</b> values.
<b>allow</b>	Permits namespace access to any traffic that matches an ACL rule with the <b>allow</b> action. The field supports <b>alert</b> , <b>warning</b> , <b>notice</b> , <b>info</b> , or <b>debug</b> values.
<b>pass</b>	A <b>pass</b> action applies to an admin network policy's ACL rule. A <b>pass</b> action allows either the network policy in the namespace or the baseline admin network policy rule to evaluate all incoming and outgoing traffic. A network policy does not support a <b>pass</b> action.

## Additional resources

- Understanding network policy APIs

### 5.4.3. AdminNetworkPolicy audit logging

Audit logging is enabled per **AdminNetworkPolicy** CR by annotating an ANP policy with the **k8s.ovn.org/acl-logging** key such as in the following example:

#### Example 5.19. Example of annotation for AdminNetworkPolicy CR

```

apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  annotations:
    k8s.ovn.org/acl-logging: '{ "deny": "alert", "allow": "alert", "pass" : "warning" }'
  name: anp-tenant-log
spec:
  priority: 5
  subject:
    namespaces:
      matchLabels:
        tenant: backend-storage # Selects all pods owned by storage tenant.
  ingress:
    - name: "allow-all-ingress-product-development-and-customer" # Product development and
      customer tenant ingress to backend storage.
      action: "Allow"
      from:
        - pods:
            namespaceSelector:
              matchExpressions:
                - key: tenant
                  operator: In
                  values:
                    - product-development
                    - customer

```

```

    podSelector: {}
- name: "pass-all-ingress-product-security"
  action: "Pass"
  from:
    - namespaces:
      matchLabels:
        tenant: product-security
- name: "deny-all-ingress" # Ingress to backend from all other pods in the cluster.
  action: "Deny"
  from:
    - namespaces: {}
egress:
- name: "allow-all-egress-product-development"
  action: "Allow"
  to:
    - pods:
      namespaceSelector:
        matchLabels:
          tenant: product-development
      podSelector: {}
- name: "pass-egress-product-security"
  action: "Pass"
  to:
    - namespaces:
      matchLabels:
        tenant: product-security
- name: "deny-all-egress" # Egress from backend denied to all other pods.
  action: "Deny"
  to:
    - namespaces: {}

```

Logs are generated whenever a specific OVN ACL is hit and meets the action criteria set in your logging annotation. For example, an event in which any of the namespaces with the label **tenant: product-development** accesses the namespaces with the label **tenant: backend-storage**, a log is generated.



### NOTE

ACL logging is limited to 60 characters. If your ANP **name** field is long, the rest of the log will be truncated.

The following is a direction index for the examples log entries that follow:

Direction	Rule
-----------	------

Direction	Rule
Ingress	<p><b>Rule0</b> Allow from tenant <b>product-development</b> and <b>customer</b> to tenant <b>backend-storage</b>; Ingress0: <b>Allow</b></p> <p><b>Rule1</b> Pass from <b>product-security`to tenant `backend-storage</b>; Ingress1: <b>Pass</b></p> <p><b>Rule2</b> Deny ingress from all pods; Ingress2: <b>Deny</b></p>
Egress	<p><b>Rule0</b> Allow to <b>product-development</b>; Egress0: <b>Allow</b></p> <p><b>Rule1</b> Pass to <b>product-security</b>; Egress1: <b>Pass</b></p> <p><b>Rule2</b> Deny egress to all other pods; Egress2: <b>Deny</b></p>

#### Example 5.20. Example ACL log entry forAllow action of theAdminNetworkPolicy namedanp-tenant-log with Ingress:0 and Egress:0

```
2024-06-10T16:27:45.194Z|00052|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Ingress:0", verdict=allow, severity=alert, direction=to-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:1a,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.26,nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=57814,tp_dst=8080,tcp_flags=syn
```

```
2024-06-10T16:28:23.130Z|00059|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Ingress:0", verdict=allow, severity=alert, direction=to-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:18,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.24,nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=38620,tp_dst=8080,tcp_flags=ack
```

```
2024-06-10T16:28:38.293Z|00069|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Egress:0", verdict=allow, severity=alert, direction=from-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:19,dl_dst=0a:58:0a:80:02:1a,nw_src=10.128.2.25,nw_dst=10.128.2.26,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=47566,tp_dst=8080,tcp_flags=fin|ack=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=55704,tp_dst=8080,tcp_flags=ack
```

#### Example 5.21. Example ACL log entry forPass action of theAdminNetworkPolicy namedanp-tenant-log with Ingress:1 and Egress:1

```
2024-06-10T16:33:12.019Z|00075|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-log:Ingress:1", verdict=pass, severity=warning, direction=to-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:1b,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.27,nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=37394,tp_dst=8080,tcp_flags=ack
```

```
2024-06-10T16:35:04.209Z|00081|acl_log(ovn_pinctrl0)|INFO|name="ANP:anp-tenant-
```

```
log:Egress:1", verdict=pass, severity=warning, direction=from-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:19,dl_dst=0a:58:0a:80:02:1b,nw_src=10.128.2.25,nw_dst=10.128.2.27,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=34018,tp_dst=8080,tcp_flags=ack
```

**Example 5.22. Example ACL log entry forDeny action of theAdminNetworkPolicy namedanp-tenant-log with Egress:2 and Ingress2**

```
2024-06-10T16:43:05.287Z|00087|acl_log(ovn_pinctrl0)||INFO|name="ANP:anp-tenant-log:Egress:2", verdict=drop, severity=alert, direction=from-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:19,dl_dst=0a:58:0a:80:02:18,nw_src=10.128.2.25,nw_dst=10.128.2.24,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=51598,tp_dst=8080,tcp_flags=syn
```

```
2024-06-10T16:44:43.591Z|00090|acl_log(ovn_pinctrl0)||INFO|name="ANP:anp-tenant-log:Ingress:2", verdict=drop, severity=alert, direction=to-lport:  
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:1c,dl_dst=0a:58:0a:80:02:19,nw_src=10.128.2.28,nw_dst=10.128.2.25,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=33774,tp_dst=8080,tcp_flags=syn
```

The following table describes ANP annotation:

**Table 5.3. Audit logging AdminNetworkPolicy annotation**

Annotation	Value
<b>k8s.ovn.org/acl-logging</b>	You must specify at least one of <b>Allow</b> , <b>Deny</b> , or <b>Pass</b> to enable audit logging for a namespace.  <b>Deny</b> Optional: Specify <b>alert</b> , <b>warning</b> , <b>notice</b> , <b>info</b> , or <b>debug</b> . <b>Allow</b> Optional: Specify <b>alert</b> , <b>warning</b> , <b>notice</b> , <b>info</b> , or <b>debug</b> . <b>Pass</b> Optional: Specify <b>alert</b> , <b>warning</b> , <b>notice</b> , <b>info</b> , or <b>debug</b> .

#### 5.4.4. BaselineAdminNetworkPolicy audit logging

Audit logging is enabled in the **BaselineAdminNetworkPolicy** CR by annotating an BANP policy with the **k8s.ovn.org/acl-logging** key such as in the following example:

**Example 5.23. Example of annotation forBaselineAdminNetworkPolicy CR**

```
apiVersion: policy.networking.k8s.io/v1alpha1  
kind: BaselineAdminNetworkPolicy  
metadata:  
  annotations:  
    k8s.ovn.org/acl-logging: '{ "deny": "alert", "allow": "alert"}'  
  name: default  
spec:  
  subject:  
  namespaces:
```

```

matchLabels:
  tenant: workloads # Selects all workload pods in the cluster.

ingress:
- name: "default-allow-dns" # This rule allows ingress from dns tenant to all workloads.
  action: "Allow"
  from:
    - namespaces:
        matchLabels:
          tenant: dns

- name: "default-deny-dns" # This rule denies all ingress from all pods to workloads.
  action: "Deny"
  from:
    - namespaces: {} # Use the empty selector with caution because it also selects OpenShift
                      namespaces as well.

egress:
- name: "default-deny-dns" # This rule denies all egress from workloads. It will be applied when
no ANP or network policy matches.
  action: "Deny"
  to:
    - namespaces: {} # Use the empty selector with caution because it also selects OpenShift
                      namespaces as well.

```

In the example, an event in which any of the namespaces with the label **tenant: dns** accesses the namespaces with the label **tenant: workloads**, a log is generated.

The following is a direction index for the examples log entries that follow:

Direction	Rule
Ingress	<b>Rule0</b> Allow from tenant <b>dns</b> to tenant <b>workloads</b> ; Ingress0: <b>Allow</b> <b>Rule1</b> Deny to tenant <b>workloads</b> from all pods; Ingress1: <b>Deny</b>
Egress	<b>Rule0</b> Deny to all pods; Egress0: <b>Deny</b>

#### Example 5.24. Example ACL allow log entry forAllow action of default BANP with Ingress:0

```

2024-06-10T18:11:58.263Z|00022|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,nw_ds
t=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,tcp_flags=syn

2024-06-10T18:11:58.264Z|00023|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,nw_ds
t=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,tcp_flags=psh|

```

```

ack
2024-06-10T18:11:58.264Z|00024|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,tcp_flags=ack

2024-06-10T18:11:58.264Z|00025|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,tcp_flags=ack

2024-06-10T18:11:58.264Z|00026|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,tcp_flags=fin|ack

2024-06-10T18:11:58.264Z|00027|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:0",
verdict=allow, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:57,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.87,nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=60510,tp_dst=8080,tcp_flags=ack

```

#### **Example 5.25. Example ACL allow log entry for Allow action of default BANP with Egress:0 and Ingress:1**

```

2024-06-10T18:09:57.774Z|00016|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Egress:0",
verdict=drop, severity=alert, direction=from-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:56,dl_dst=0a:58:0a:82:02:57,nw_src=10.130.2.86,nw_dst=10.130.2.87,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=45614,tp_dst=8080,tcp_flags=syn

2024-06-10T18:09:58.809Z|00017|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Egress:0",
verdict=drop, severity=alert, direction=from-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:56,dl_dst=0a:58:0a:82:02:57,nw_src=10.130.2.86,nw_dst=10.130.2.87,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=45614,tp_dst=8080,tcp_flags=syn

2024-06-10T18:10:00.857Z|00018|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Egress:0",
verdict=drop, severity=alert, direction=from-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:56,dl_dst=0a:58:0a:82:02:57,nw_src=10.130.2.86,nw_dst=10.130.2.87,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=45614,tp_dst=8080,tcp_flags=syn

2024-06-10T18:10:25.414Z|00019|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:1",
verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:58,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.88,nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=40630,tp_dst=8080,tcp_flags=syn

2024-06-10T18:10:26.457Z|00020|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:1",
verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:58,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.88,nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=40630,tp_dst=8080,tcp_flags=syn

2024-06-10T18:10:28.505Z|00021|acl_log(ovn_pinctrl0)|INFO|name="BANP:default:Ingress:1",
verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:82:02:58,dl_dst=0a:58:0a:82:02:56,nw_src=10.130.2.88,nw_dst=10.130.2.86,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,tp_src=40630,tp_dst=8080,tcp_flags=syn

```

The following table describes BANP annotation:

**Table 5.4. Audit logging BaselineAdminNetworkPolicy annotation**

Annotation	Value
<b>k8s.ovn.org/acl-logging</b>	<p>You must specify at least one of <b>Allow</b> or <b>Deny</b> to enable audit logging for a namespace.</p> <p><b>Deny</b> Optional: Specify <b>alert</b>, <b>warning</b>, <b>notice</b>, <b>info</b>, or <b>debug</b>.</p> <p><b>Allow</b> Optional: Specify <b>alert</b>, <b>warning</b>, <b>notice</b>, <b>info</b>, or <b>debug</b>.</p>

### 5.4.5. Configuring egress firewall and network policy auditing for a cluster

As a cluster administrator, you can customize audit logging for your cluster.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster with a user with **cluster-admin** privileges.

#### Procedure

- To customize the audit logging configuration, enter the following command:

```
$ oc edit network.operator.openshift.io/cluster
```

#### TIP

You can alternatively customize and apply the following YAML to configure audit logging:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
      syslogFacility: local0
```

#### Verification

1. To create a namespace with network policies complete the following steps:
  - a. Create a namespace for verification:

```
$ cat <<EOF| oc create -f -
kind: Namespace
apiVersion: v1
metadata:
  name: verify-audit-logging
  annotations:
    k8s.ovn.org/acl-logging: '{ \"deny\": \"alert\", \"allow\": \"alert\" }'
EOF
```

### Example output

```
namespace/verify-audit-logging created
```

- Create network policies for the namespace:

```
$ cat <<EOF| oc create -n verify-audit-logging -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
spec:
  podSelector:
    matchLabels:
  policyTypes:
    - Ingress
    - Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-same-namespace
  namespace: verify-audit-logging
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector: {}
  egress:
    - to:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: verify-audit-logging
EOF
```

### Example output

```
networkpolicy.networking.k8s.io/deny-all created
networkpolicy.networking.k8s.io/allow-from-same-namespace created
```

- Create a pod for source traffic in the **default** namespace:

```
$ cat <<EOF| oc create -n default -f -
apiVersion: v1
kind: Pod
metadata:
  name: client
spec:
  containers:
    - name: client
      image: registry.access.redhat.com/rhel7/rhel-tools
      command: ["/bin/sh", "-c"]
      args:
        ["sleep inf"]
EOF
```

3. Create two pods in the **verify-audit-logging** namespace:

```
$ for name in client server; do
cat <<EOF| oc create -n verify-audit-logging -f -
apiVersion: v1
kind: Pod
metadata:
  name: ${name}
spec:
  containers:
    - name: ${name}
      image: registry.access.redhat.com/rhel7/rhel-tools
      command: ["/bin/sh", "-c"]
      args:
        ["sleep inf"]
EOF
done
```

#### Example output

```
pod/client created
pod/server created
```

4. To generate traffic and produce network policy audit log entries, complete the following steps:

- a. Obtain the IP address for pod named **server** in the **verify-audit-logging** namespace:

```
$ POD_IP=$(oc get pods server -n verify-audit-logging -o jsonpath='{.status.podIP}')
```

- b. Ping the IP address from the previous command from the pod named **client** in the **default** namespace and confirm that all packets are dropped:

```
$ oc exec -it client -n default -- /bin/ping -c 2 $POD_IP
```

#### Example output

```
PING 10.128.2.55 (10.128.2.55) 56(84) bytes of data.
--- 10.128.2.55 ping statistics ---
```

2 packets transmitted, 0 received, 100% packet loss, time 2041ms

- c. Ping the IP address saved in the **POD\_IP** shell environment variable from the pod named **client** in the **verify-audit-logging** namespace and confirm that all packets are allowed:

```
$ oc exec -it client -n verify-audit-logging -- /bin/ping -c 2 $POD_IP
```

#### Example output

```
PING 10.128.0.86 (10.128.0.86) 56(84) bytes of data.
64 bytes from 10.128.0.86: icmp_seq=1 ttl=64 time=2.21 ms
64 bytes from 10.128.0.86: icmp_seq=2 ttl=64 time=0.440 ms

--- 10.128.0.86 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.440/1.329/2.219/0.890 ms
```

5. Display the latest entries in the network policy audit log:

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-
headers=true | awk '{ print $1 }') ; do
    oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
done
```

#### Example output

```
2023-11-02T16:28:54.139Z|00004|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-!port:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,n
w_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp
_flags=syn
2023-11-02T16:28:55.187Z|00005|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-!port:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,n
w_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp
_flags=syn
2023-11-02T16:28:57.235Z|00006|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-!port:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,n
w_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp
_flags=syn
2023-11-02T16:49:57.909Z|00028|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-
!port:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:57.909Z|00029|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-
!port:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:58.932Z|00030|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-
!port:
```

```
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:58.932Z|00031|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
```

#### 5.4.6. Enabling egress firewall and network policy audit logging for a namespace

As a cluster administrator, you can enable audit logging for a namespace.

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster with a user with **cluster-admin** privileges.

##### Procedure

- To enable audit logging for a namespace, enter the following command:

```
$ oc annotate namespace <namespace> \
  k8s.ovn.org/acl-logging='{ "deny": "alert", "allow": "notice" }'
```

where:

##### <namespace>

Specifies the name of the namespace.

##### TIP

You can alternatively apply the following YAML to enable audit logging:

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "alert",
        "allow": "notice"
      }
```

##### Example output

```
namespaces/verify-audit-logging annotated
```

##### Verification

- Display the latest entries in the audit log:

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-headers=true | awk '{ print $1 }') ; do
    oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
done
```

### Example output

```
2023-11-02T16:49:57.909Z|00028|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-  
lport:  
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,  
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0  
2023-11-02T16:49:57.909Z|00029|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-  
lport:  
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,  
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0  
2023-11-02T16:49:58.932Z|00030|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-  
lport:  
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,  
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0  
2023-11-02T16:49:58.932Z|00031|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-  
lport:  
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,  
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
```

### 5.4.7. Disabling egress firewall and network policy audit logging for a namespace

As a cluster administrator, you can disable audit logging for a namespace.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster with a user with **cluster-admin** privileges.

#### Procedure

- To disable audit logging for a namespace, enter the following command:

```
$ oc annotate --overwrite namespace <namespace> k8s.ovn.org/acl-logging-
```

where:

**<namespace>**

Specifies the name of the namespace.

**TIP**

You can alternatively apply the following YAML to disable audit logging:

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: null
```

**Example output**

```
namespace/verify-audit-logging annotated
```

## 5.4.8. Additional resources

- [About network policy](#)
- [Configuring an egress firewall for a project](#)

# 5.5. EGRESS FIREWALL

## 5.5.1. Viewing an egress firewall for a project

As a cluster administrator, you can list the names of any existing egress firewalls and view the traffic rules for a specific egress firewall.

### 5.5.1.1. Viewing an EgressFirewall object

You can view an EgressFirewall object in your cluster.

#### Prerequisites

- A cluster using the OVN-Kubernetes network plugin.
- Install the OpenShift Command-line Interface (CLI), commonly known as **oc**.
- You must log in to the cluster.

#### Procedure

1. Optional: To view the names of the EgressFirewall objects defined in your cluster, enter the following command:

```
$ oc get egressfirewall --all-namespaces
```

2. To inspect a policy, enter the following command. Replace **<policy\_name>** with the name of the policy to inspect.

```
$ oc describe egressfirewall <policy_name>
```

## Example output

```
Name: default
Namespace: project1
Created: 20 minutes ago
Labels: <none>
Annotations: <none>
Rule: Allow to 1.2.3.0/24
Rule: Allow to www.example.com
Rule: Deny to 0.0.0.0/0
```

### 5.5.2. Editing an egress firewall for a project

As a cluster administrator, you can modify network traffic rules for an existing egress firewall.

#### 5.5.2.1. Editing an EgressFirewall object

As a cluster administrator, you can update the egress firewall for a project.

##### Prerequisites

- A cluster using the OVN-Kubernetes network plugin.
- Install the OpenShift CLI (**oc**).
- You must log in to the cluster as a cluster administrator.

##### Procedure

1. Find the name of the EgressFirewall object for the project. Replace **<project>** with the name of the project.

```
$ oc get -n <project> egressfirewall
```

2. Optional: If you did not save a copy of the EgressFirewall object when you created the egress network firewall, enter the following command to create a copy.

```
$ oc get -n <project> egressfirewall <name> -o yaml > <filename>.yaml
```

Replace **<project>** with the name of the project. Replace **<name>** with the name of the object. Replace **<filename>** with the name of the file to save the YAML to.

3. After making changes to the policy rules, enter the following command to replace the EgressFirewall object. Replace **<filename>** with the name of the file containing the updated EgressFirewall object.

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

### 5.5.3. Removing an egress firewall from a project

As a cluster administrator, you can remove an egress firewall from a project to remove all restrictions on network traffic from the project that leaves the OpenShift Container Platform cluster.

### 5.5.3.1. Removing an EgressFirewall object

As a cluster administrator, you can remove an egress firewall from a project.

#### Prerequisites

- A cluster using the OVN-Kubernetes network plugin.
- Install the OpenShift CLI (**oc**).
- You must log in to the cluster as a cluster administrator.

#### Procedure

1. Find the name of the EgressFirewall object for the project. Replace **<project>** with the name of the project.

```
$ oc get -n <project> egressfirewall
```

2. Enter the following command to delete the EgressFirewall object. Replace **<project>** with the name of the project and **<name>** with the name of the object.

```
$ oc delete -n <project> egressfirewall <name>
```

### 5.5.4. Configuring an egress firewall for a project

As a cluster administrator, you can create an egress firewall for a project that restricts egress traffic leaving your OpenShift Container Platform cluster.

#### 5.5.4.1. How an egress firewall works in a project

As a cluster administrator, you can use an *egress firewall* to limit the external hosts that some or all pods can access from within the cluster. An egress firewall supports the following scenarios:

- A pod can only connect to internal hosts and cannot initiate connections to the public internet.
- A pod can only connect to the public internet and cannot initiate connections to internal hosts that are outside the OpenShift Container Platform cluster.
- A pod cannot reach specified internal subnets or hosts outside the OpenShift Container Platform cluster.
- A pod can connect to only specific external hosts.

For example, you can allow one project access to a specified IP range but deny the same access to a different project. Or you can restrict application developers from updating from Python pip mirrors, and force updates to come only from approved sources.



#### NOTE

Egress firewall does not apply to the host network namespace. Pods with host networking enabled are unaffected by egress firewall rules.

You configure an egress firewall policy by creating an EgressFirewall custom resource (CR) object. The egress firewall matches network traffic that meets any of the following criteria:

- An IP address range in CIDR format
- A DNS name that resolves to an IP address
- A port number
- A protocol that is one of the following protocols: TCP, UDP, and SCTP

### IMPORTANT

If your egress firewall includes a deny rule for **0.0.0.0/0**, access to your OpenShift Container Platform API servers is blocked. You must either add allow rules for each IP address or use the **nodeSelector** type allow rule in your egress policy rules to connect to API servers.

The following example illustrates the order of the egress firewall rules necessary to ensure API server access:

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
  namespace: <namespace> ①
spec:
  egress:
    - to:
        cidrSelector: <api_server_address_range> ②
        type: Allow
      # ...
    - to:
        cidrSelector: 0.0.0.0/0 ③
        type: Deny
```

- ① The namespace for the egress firewall.
- ② The IP address range that includes your OpenShift Container Platform API servers.
- ③ A global deny rule prevents access to the OpenShift Container Platform API servers.

To find the IP address for your API servers, run **oc get ep kubernetes -n default**.

For more information, see [BZ#1988324](#).



### WARNING

Egress firewall rules do not apply to traffic that goes through routers. Any user with permission to create a Route CR object can bypass egress firewall policy rules by creating a route that points to a forbidden destination.

#### 5.5.4.1.1. Limitations of an egress firewall

An egress firewall has the following limitations:

- No project can have more than one EgressFirewall object.
- A maximum of one EgressFirewall object with a maximum of 8,000 rules can be defined per project.
- If you are using the OVN-Kubernetes network plugin with shared gateway mode in Red Hat OpenShift Networking, return ingress replies are affected by egress firewall rules. If the egress firewall rules drop the ingress reply destination IP, the traffic is dropped.

Violating any of these restrictions results in a broken egress firewall for the project. Consequently, all external network traffic is dropped, which can cause security risks for your organization.

An Egress Firewall resource can be created in the **kube-node-lease**, **kube-public**, **kube-system**, **openshift** and **openshift-** projects.

#### 5.5.4.1.2. Matching order for egress firewall policy rules

The egress firewall policy rules are evaluated in the order that they are defined, from first to last. The first rule that matches an egress connection from a pod applies. Any subsequent rules are ignored for that connection.

#### 5.5.4.1.3. How Domain Name Server (DNS) resolution works

If you use DNS names in any of your egress firewall policy rules, proper resolution of the domain names is subject to the following restrictions:

- Domain name updates are polled based on a time-to-live (TTL) duration. By default, the duration is 30 minutes. When the egress firewall controller queries the local name servers for a domain name, if the response includes a TTL and the TTL is less than 30 minutes, the controller sets the duration for that DNS name to the returned value. Each DNS name is queried after the TTL for the DNS record expires.
- The pod must resolve the domain from the same local name servers when necessary. Otherwise the IP addresses for the domain known by the egress firewall controller and the pod can be different. If the IP addresses for a hostname differ, the egress firewall might not be enforced consistently.
- Because the egress firewall controller and pods asynchronously poll the same local name server, the pod might obtain the updated IP address before the egress controller does, which causes a race condition. Due to this current limitation, domain name usage in EgressFirewall objects is only recommended for domains with infrequent IP address changes.



#### NOTE

Using DNS names in your egress firewall policy does not affect local DNS resolution through CoreDNS.

However, if your egress firewall policy uses domain names, and an external DNS server handles DNS resolution for an affected pod, you must include egress firewall rules that permit access to the IP addresses of your DNS server.

#### 5.5.4.1.3.1. Improved DNS resolution and resolving wildcard domain names

There might be situations where the IP addresses associated with a DNS record change frequently, or you might want to specify wildcard domain names in your egress firewall policy rules.

In this situation, the OVN-Kubernetes cluster manager creates a **DNSNameResolver** custom resource object for each unique DNS name used in your egress firewall policy rules. This custom resource stores the following information:



## IMPORTANT

Improved DNS resolution for egress firewall rules 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](#).

## Example DNSNameResolver CR definition

```
apiVersion: networking.openshift.io/v1alpha1
kind: DNSNameResolver
spec:
  name: www.example.com. 1
status:
  resolvedNames:
    - dnsName: www.example.com. 2
      resolvedAddress:
        - ip: "1.2.3.4" 3
      ttlSeconds: 60 4
      lastLookupTime: "2023-08-08T15:07:04Z" 5
```

- 1** The DNS name. This can be either a standard DNS name or a wildcard DNS name. For a wildcard DNS name, the DNS name resolution information contains all of the DNS names that match the wildcard DNS name.
- 2** The resolved DNS name matching the **spec.name** field. If the **spec.name** field contains a wildcard DNS name, then multiple **dnsName** entries are created that contain the standard DNS names that match the wildcard DNS name when resolved. If the wildcard DNS name can also be successfully resolved, then this field also stores the wildcard DNS name.
- 3** The current IP addresses associated with the DNS name.
- 4** The last time-to-live (TTL) duration.
- 5** The last lookup time.

If during DNS resolution the DNS name in the query matches any name defined in a **DNSNameResolver** CR, then the previous information is updated accordingly in the CR **status** field. For unsuccessful DNS wildcard name lookups, the request is retried after a default TTL of 30 minutes.

The OVN-Kubernetes cluster manager watches for updates to an **EgressFirewall** custom resource object, and creates, modifies, or deletes **DNSNameResolver** CRs associated with those egress firewall policies when that update occurs.



### WARNING

Do not modify **DNSNameResolver** custom resources directly. This can lead to unwanted behavior of your egress firewall.

#### 5.5.4.2. EgressFirewall custom resource (CR) object

You can define one or more rules for an egress firewall. A rule is either an **Allow** rule or a **Deny** rule, with a specification for the traffic that the rule applies to.

The following YAML describes an EgressFirewall CR object:

#### EgressFirewall object

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: <name> ①
spec:
  egress: ②
  ...
  
```

- ① The name for the object must be **default**.
- ② A collection of one or more egress network policy rules as described in the following section.

#### 5.5.4.2.1. EgressFirewall rules

The following YAML describes an egress firewall rule object. The user can select either an IP address range in CIDR format, a domain name, or use the **nodeSelector** to allow or deny egress traffic. The **egress** stanza expects an array of one or more objects.

#### Egress policy rule stanza

```
egress:
  - type: <type> ①
    to: ②
      cidrSelector: <cidr> ③
      dnsName: <dns_name> ④
      nodeSelector: <label_name>: <label_value> ⑤
    ports: ⑥
    ...
  
```

- ① The type of rule. The value must be either **Allow** or **Deny**.

- 2 A stanza describing an egress traffic match rule that specifies the **cidrSelector** field or the **dnsName** field. You cannot use both fields in the same rule.
- 3 An IP address range in CIDR format.
- 4 A DNS domain name.
- 5 Labels are key/value pairs that the user defines. Labels are attached to objects, such as pods. The **nodeSelector** allows for one or more node labels to be selected and attached to pods.
- 6 Optional: A stanza describing a collection of network ports and protocols for the rule.

## Ports stanza

```
ports:
- port: <port> ①
  protocol: <protocol> ②
```

- 1 A network port, such as **80** or **443**. If you specify a value for this field, you must also specify a value for **protocol**.
- 2 A network protocol. The value must be either **TCP**, **UDP**, or **SCTP**.

### 5.5.4.2.2. Example EgressFirewall CR objects

The following example defines several egress firewall policy rules:

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress: ①
    - type: Allow
      to:
        cidrSelector: 1.2.3.0/24
    - type: Deny
      to:
        cidrSelector: 0.0.0.0/0
```

- 1 A collection of egress firewall policy rule objects.

The following example defines a policy rule that denies traffic to the host at the **172.16.1.1/32** IP address, if the traffic is using either the TCP protocol and destination port **80** or any protocol and destination port **443**.

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress:
```

```

- type: Deny
  to:
    cidrSelector: 172.16.1.1/32
  ports:
    - port: 80
      protocol: TCP
    - port: 443
  
```

#### 5.5.4.2.3. Example nodeSelector for EgressFirewall

As a cluster administrator, you can allow or deny egress traffic to nodes in your cluster by specifying a label using **nodeSelector**. Labels can be applied to one or more nodes. The following is an example with the **region=east** label:

```

apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress:
    - to:
        nodeSelector:
          matchLabels:
            region: east
      type: Allow
  
```

#### TIP

Instead of adding manual rules per node IP address, use node selectors to create a label that allows pods behind an egress firewall to access host network pods.

#### 5.5.4.3. Creating an egress firewall policy object

As a cluster administrator, you can create an egress firewall policy object for a project.



#### IMPORTANT

If the project already has an EgressFirewall object defined, you must edit the existing policy to make changes to the egress firewall rules.

#### Prerequisites

- A cluster that uses the OVN-Kubernetes network plugin.
- Install the OpenShift CLI (**oc**).
- You must log in to the cluster as a cluster administrator.

#### Procedure

1. Create a policy rule:
  - a. Create a **<policy\_name>.yaml** file where **<policy\_name>** describes the egress policy rules.

- b. In the file you created, define an egress policy object.
2. Enter the following command to create the policy object. Replace **<policy\_name>** with the name of the policy and **<project>** with the project that the rule applies to.

```
$ oc create -f <policy_name>.yaml -n <project>
```

In the following example, a new EgressFirewall object is created in a project named **project1**:

```
$ oc create -f default.yaml -n project1
```

#### Example output

```
egressfirewall.k8s.ovn.org/v1 created
```

3. Optional: Save the **<policy\_name>.yaml** file so that you can make changes later.

## 5.6. CONFIGURING IPSEC ENCRYPTION

By enabling IPsec, you can encrypt both internal pod-to-pod cluster traffic between nodes and external traffic between pods and IPsec endpoints external to your cluster. All pod-to-pod network traffic between nodes on the OVN-Kubernetes cluster network is encrypted with IPsec in *Transport mode*.

IPsec is disabled by default. You can enable IPsec either during or after installing the cluster. For information about cluster installation, see [OpenShift Container Platform installation overview](#).

The following support limitations exist for IPsec on a OpenShift Container Platform cluster:

- On IBM Cloud®, IPsec supports only NAT-T. Encapsulating Security Payload (ESP) is not supported on this platform.
- If your cluster uses [hosted control planes](#) for Red Hat OpenShift Container Platform, IPsec is not supported for IPsec encryption of either pod-to-pod or traffic to external hosts.
- Using ESP hardware offloading on any network interface is not supported if one or more of those interfaces is attached to Open vSwitch (OVS). Enabling IPsec for your cluster triggers the use of IPsec with interfaces attached to OVS. By default, OpenShift Container Platform disables ESP hardware offloading on any interfaces attached to OVS.
- If you enabled IPsec for network interfaces that are not attached to OVS, a cluster administrator must manually disable ESP hardware offloading on each interface that is not attached to OVS.
- IPsec is not supported on Red Hat Enterprise Linux (RHEL) compute nodes because of a **libreswan** incompatibility issue between a host and an **ovn-ipsec** container that exist in each compute node. See ([OCPBUGS-53316](#)).

The following list outlines key tasks in the IPsec documentation:

- Enable and disable IPsec after cluster installation.
- Configure IPsec encryption for traffic between the cluster and external hosts.
- Verify that IPsec encrypts traffic between pods on different nodes.

### 5.6.1. Modes of operation

When using IPsec on your OpenShift Container Platform cluster, you can choose from the following operating modes:

**Table 5.5. IPsec modes of operation**

Mode	Description	Default
<b>Disabled</b>	No traffic is encrypted. This is the cluster default.	Yes
<b>Full</b>	Pod-to-pod traffic is encrypted as described in "Types of network traffic flows encrypted by pod-to-pod IPsec". Traffic to external nodes may be encrypted after you complete the required configuration steps for IPsec.	No
<b>External</b>	Traffic to external nodes may be encrypted after you complete the required configuration steps for IPsec.	No

### 5.6.2. Prerequisites

For IPsec support for encrypting traffic to external hosts, ensure that the following prerequisites are met:

- The OVN-Kubernetes network plugin must be configured in local gateway mode, where **ovnKubernetesConfig.gatewayConfig.routingViaHost=true**.
- The NMState Operator is installed. This Operator is required for specifying the IPsec configuration. For more information, see [Kubernetes NMState Operator](#).



#### NOTE

The NMState Operator is supported on Google Cloud Platform (GCP) only for configuring IPsec.

- The Butane tool (**butane**) is installed. To install Butane, see [Installing Butane](#).

These prerequisites are required to add certificates into the host NSS database and to configure IPsec to communicate with external hosts.

### 5.6.3. Network connectivity requirements when IPsec is enabled

You must configure the network connectivity between machines to allow OpenShift Container Platform cluster components to communicate. Each machine must be able to resolve the hostnames of all other machines in the cluster.

**Table 5.6. Ports used for all-machine to all-machine communications**

Protocol	Port	Description
UDP	500	IPsec IKE packets

Protocol	Port	Description
	<b>4500</b>	IPsec NAT-T packets
ESP	N/A	IPsec Encapsulating Security Payload (ESP)

## 5.6.4. IPsec encryption for pod-to-pod traffic

For IPsec encryption of pod-to-pod traffic, the following sections describe which specific pod-to-pod traffic is encrypted, what kind of encryption protocol is used, and how X.509 certificates are handled. These sections do not apply to IPsec encryption between the cluster and external hosts, which you must configure manually for your specific external network infrastructure.

### 5.6.4.1. Types of network traffic flows encrypted by pod-to-pod IPsec

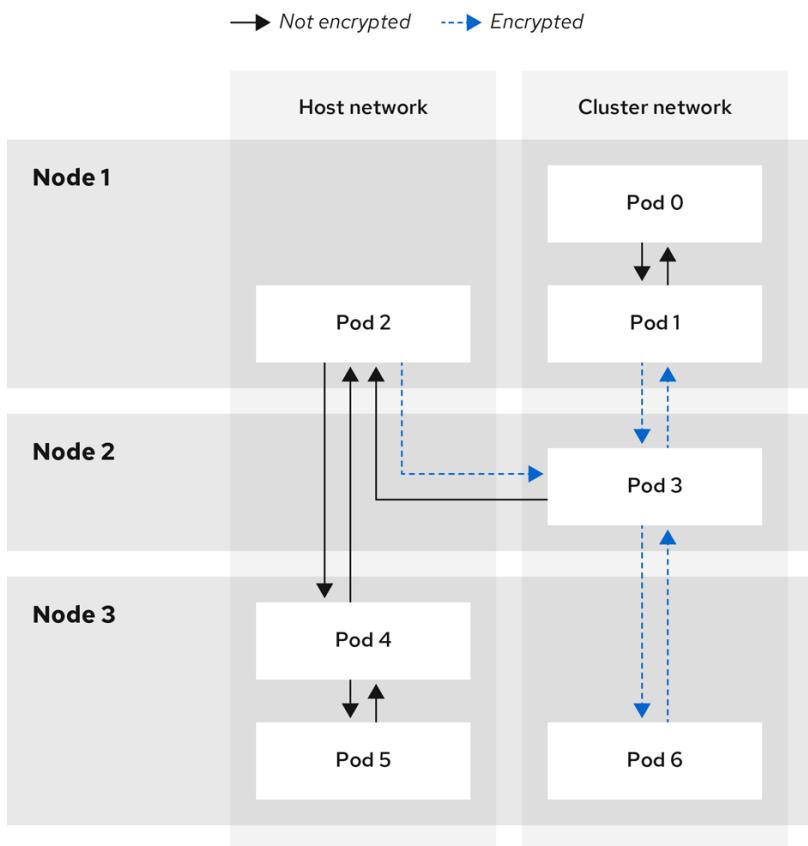
With IPsec enabled, only the following network traffic flows between pods are encrypted:

- Traffic between pods on different nodes on the cluster network
- Traffic from a pod on the host network to a pod on the cluster network

The following traffic flows are not encrypted:

- Traffic between pods on the same node on the cluster network
- Traffic between pods on the host network
- Traffic from a pod on the cluster network to a pod on the host network

The encrypted and unencrypted flows are illustrated in the following diagram:



138\_OpenShift\_0421

#### 5.6.4.2. Encryption protocol and IPsec mode

The encrypt cipher used is **AES-GCM-16-256**. The integrity check value (ICV) is **16** bytes. The key length is **256** bits.

The IPsec mode used is *Transport mode*, a mode that encrypts end-to-end communication by adding an Encapsulated Security Payload (ESP) header to the IP header of the original packet and encrypts the packet data. OpenShift Container Platform does not currently use or support IPsec *Tunnel mode* for pod-to-pod communication.

#### 5.6.4.3. Security certificate generation and rotation

The Cluster Network Operator (CNO) generates a self-signed X.509 certificate authority (CA) that is used by IPsec for encryption. Certificate signing requests (CSRs) from each node are automatically fulfilled by the CNO.

The CA is valid for 10 years. The individual node certificates are valid for 5 years and are automatically rotated after 4 1/2 years elapse.

#### 5.6.5. IPsec encryption for external traffic

OpenShift Container Platform supports IPsec encryption for traffic to external hosts with TLS certificates that you must supply.

##### 5.6.5.1. Supported platforms

This feature is supported on the following platforms:

- Bare metal

- Google Cloud Platform (GCP)
- Red Hat OpenStack Platform (RHOSP)
- VMware vSphere



### IMPORTANT

If you have Red Hat Enterprise Linux (RHEL) worker nodes, these do not support IPsec encryption for external traffic.

If your cluster uses hosted control planes for Red Hat OpenShift Container Platform, configuring IPsec for encrypting traffic to external hosts is not supported.

#### 5.6.5.2. Limitations

Ensure that the following prohibitions are observed:

- IPv6 configuration is not currently supported by the NMState Operator when configuring IPsec for external traffic.
- Certificate common names (CN) in the provided certificate bundle must not begin with the **ovs\_** prefix, because this naming can conflict with pod-to-pod IPsec CN names in the Network Security Services (NSS) database of each node.

#### 5.6.6. Enabling IPsec encryption

As a cluster administrator, you can enable pod-to-pod IPsec encryption, IPsec encryption between the cluster, and external IPsec endpoints.

You can configure IPsec in either of the following modes:

- **Full:** Encryption for pod-to-pod and external traffic
- **External:** Encryption for external traffic



### NOTE

If you configure IPsec in **Full** mode, you must also complete the "Configuring IPsec encryption for external traffic" procedure.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- You are logged in to the cluster as a user with **cluster-admin** privileges.
- You have reduced the size of your cluster MTU by **46** bytes to allow for the overhead of the IPsec ESP header.

#### Procedure

1. To enable IPsec encryption, enter the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge -p \
'{
  "spec": {
    "defaultNetwork": {
      "ovnKubernetesConfig": {
        "ipsecConfig": {
          "mode": "<mode>" ①
        }
      }
    }
}'
```

- ① Specify **External** to encrypt traffic to external hosts or specify **Full** to encrypt pod-to-pod traffic and, optionally, traffic to external hosts. By default, IPsec is disabled.
2. Encrypt external traffic with IPsec by completing the "Configuring IPsec encryption for external traffic" procedure.

## Verification

1. To find the names of the OVN-Kubernetes data plane pods, enter the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -l=app=ovnkube-node
```

### Example output

ovnkube-node-5xqbf	8/8	Running	0	28m
ovnkube-node-6mwcx	8/8	Running	0	29m
ovnkube-node-ck5fr	8/8	Running	0	31m
ovnkube-node-fr4ld	8/8	Running	0	26m
ovnkube-node-wgs4l	8/8	Running	0	33m
ovnkube-node-zfvcl	8/8	Running	0	34m
...				

2. Verify that IPsec is enabled on your cluster by running the following command:



### NOTE

As a cluster administrator, you can verify that IPsec is enabled between pods on your cluster when IPsec is configured in **Full** mode. This step does not verify whether IPsec is working between your cluster and external hosts.

```
$ oc -n openshift-ovn-kubernetes rsh ovnkube-node-<XXXXXX> ovn-nbctl --no-leader-only get nb_global . ipsec ①
```

- ① Where **<XXXXXX>** specifies the random sequence of letters for a pod from the previous step.

### Example output

```
true
```

## 5.6.7. Configuring IPsec encryption for external traffic

As a cluster administrator, to encrypt external traffic with IPsec you must configure IPsec for your network infrastructure, including providing PKCS#12 certificates. Because this procedure uses Butane to create machine configs, you must have the **butane** command installed.



## NOTE

After you apply the machine config, the Machine Config Operator reboots affected nodes in your cluster to rollout the new machine config.

## Prerequisites

- Install the OpenShift CLI (**oc**).
- You have installed the **butane** utility on your local computer.
- You have installed the NMState Operator on the cluster.
- You are logged in to the cluster as a user with **cluster-admin** privileges.
- You have an existing PKCS#12 certificate for the IPsec endpoint and a CA cert in PEM format.
- You enabled IPsec in either **Full** or **External** mode on your cluster.
- The OVN-Kubernetes network plugin must be configured in local gateway mode, where **ovnKubernetesConfig.gatewayConfig.routingViaHost=true**.

## Procedure

1. Create an IPsec configuration with an NMState Operator node network configuration policy. For more information, see [Libreswan as an IPsec VPN implementation](#) .
  - a. To identify the IP address of the cluster node that is the IPsec endpoint, enter the following command:
 

```
$ oc get nodes
```
  - b. Create a file named **ipsec-config.yaml** that contains a node network configuration policy for the NMState Operator, such as in the following examples. For an overview about **NodeNetworkConfigurationPolicy** objects, see [The Kubernetes NMState project](#) .

### Example NMState IPsec transport configuration

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ipsec-config
spec:
  nodeSelector:
    kubernetes.io/hostname: "<hostname>" ①
  desiredState:
    interfaces:
      - name: <interface_name> ②
        type: ipsec
        libreswan:
          left: <cluster_node> ③
```

```

leftid: '%fromcert'
leftrsasigkey: '%cert'
leftcert: left_server
leftmodecfgclient: false
right: <external_host> ④
rightid: '%fromcert'
rightrsasigkey: '%cert'
rightsubnet: <external_address>/32 ⑤
ikev2: insist
type: transport

```

- ① Specifies the host name to apply the policy to. This host serves as the left side host in the IPsec configuration.
- ② Specifies the name of the interface to create on the host.
- ③ Specifies the host name of the cluster node that terminates the IPsec tunnel on the cluster side. The name should match SAN **[Subject Alternate Name]** from your supplied PKCS#12 certificates.
- ④ Specifies the external host name, such as **host.example.com**. The name should match the SAN **[Subject Alternate Name]** from your supplied PKCS#12 certificates.
- ⑤ Specifies the IP address of the external host, such as **10.1.2.3/32**.

### Example NMState IPsec tunnel configuration

```

apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ipsec-config
spec:
  nodeSelector:
    kubernetes.io/hostname: "<hostname>" ①
  desiredState:
    interfaces:
      - name: <interface_name> ②
        type: ipsec
        libreswan:
          left: <cluster_node> ③
          leftid: '%fromcert'
          leftmodecfgclient: false
          lefrsasigkey: '%cert'
          leftcert: left_server
          right: <external_host> ④
          rightid: '%fromcert'
          rightrsasigkey: '%cert'
          rightsubnet: <external_address>/32 ⑤
          ikev2: insist
          type: tunnel

```

- ① Specifies the host name to apply the policy to. This host serves as the left side host in the IPsec configuration.

- 2 Specifies the name of the interface to create on the host.
- 3 Specifies the host name of the cluster node that terminates the IPsec tunnel on the cluster side. The name should match SAN **[Subject Alternate Name]** from your supplied PKCS#12 certificates.
- 4 Specifies the external host name, such as **host.example.com**. The name should match the SAN **[Subject Alternate Name]** from your supplied PKCS#12 certificates.
- 5 Specifies the IP address of the external host, such as **10.1.2.3/32**.

c. To configure the IPsec interface, enter the following command:

```
$ oc create -f ipsec-config.yaml
```

2. Provide the following certificate files to add to the Network Security Services (NSS) database on each host. These files are imported as part of the Butane configuration in subsequent steps.

- **left\_server.p12**: The certificate bundle for the IPsec endpoints
- **ca.pem**: The certificate authority that you signed your certificates with

3. Create a machine config to add your certificates to the cluster:

a. To create Butane config files for the control plane and worker nodes, enter the following command:



#### NOTE

The [Butane version](#) you specify in the config file should match the OpenShift Container Platform version and always ends in **0**. For example, **4.18.0**. See "Creating machine configs with Butane" for information about Butane.

```
$ for role in master worker; do
  cat >> "99-ipsec-${role}-endpoint-config.bu" <<-EOF
  variant: openshift
  version: 4.18.0
  metadata:
    name: 99-${role}-import-certs
    labels:
      machineconfiguration.openshift.io/role: $role
  systemd:
    units:
      - name: ipsec-import.service
        enabled: true
        contents: |
          [Unit]
          Description=Import external certs into ipsec NSS
          Before=ipsec.service

          [Service]
          Type=oneshot
          ExecStart=/usr/local/bin/ipsec-addcert.sh
          RemainAfterExit=false
EOF
done
```

```

StandardOutput=journal

[Install]
WantedBy=multi-user.target
storage:
files:
- path: /etc/pki/certs/ca.pem
  mode: 0400
  overwrite: true
  contents:
    local: ca.pem
- path: /etc/pki/certs/left_server.p12
  mode: 0400
  overwrite: true
  contents:
    local: left_server.p12
- path: /usr/local/bin/ipsec-addcert.sh
  mode: 0740
  overwrite: true
  contents:
    inline: |
      #!/bin/bash -e
      echo "importing cert to NSS"
      certutil -A -n "CA" -t "CT,C,C" -d /var/lib/ipsec/nss/ -i /etc/pki/certs/ca.pem
      pk12util -W "" -i /etc/pki/certs/left_server.p12 -d /var/lib/ipsec/nss/
      certutil -M -n "left_server" -t "u,u,u" -d /var/lib/ipsec/nss/
EOF
done

```

- b. To transform the Butane files that you created in the previous step into machine configs, enter the following command:

```

$ for role in master worker; do
  butane -d . 99-ipsec-${role}-endpoint-config.bu -o ./99-ipsec-$role-endpoint-config.yaml
done

```

4. To apply the machine configs to your cluster, enter the following command:

```

$ for role in master worker; do
  oc apply -f 99-ipsec-${role}-endpoint-config.yaml
done

```



### IMPORTANT

As the Machine Config Operator (MCO) updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated before external IPsec connectivity is available.

5. Check the machine config pool status by entering the following command:

```
$ oc get mcp
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

**NOTE**

By default, the MCO updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

6. To confirm that IPsec machine configs rolled out successfully, enter the following commands:

- a. Confirm that the IPsec machine configs were created:

```
$ oc get mc | grep ipsec
```

**Example output**

80-ipsec-master-extensions	3.2.0	6d15h
80-ipsec-worker-extensions	3.2.0	6d15h

- b. Confirm that the IPsec extension are applied to control plane nodes:

```
$ oc get mcp master -o yaml | grep 80-ipsec-master-extensions -c
```

**Expected output**

```
2
```

- c. Confirm that the IPsec extension are applied to worker nodes:

```
$ oc get mcp worker -o yaml | grep 80-ipsec-worker-extensions -c
```

**Expected output**

```
2
```

**Additional resources**

- For more information about the nmstate IPsec API, see [IPsec Encryption](#)

### 5.6.8. Disabling IPsec encryption for an external IPsec endpoint

As a cluster administrator, you can remove an existing IPsec tunnel to an external host.

**Prerequisites**

- Install the OpenShift CLI (**oc**).
- You are logged in to the cluster as a user with **cluster-admin** privileges.
- You enabled IPsec in either **Full** or **External** mode on your cluster.

**Procedure**

1. Create a file named **remove-ipsec-tunnel.yaml** with the following YAML:

```

kind: NodeNetworkConfigurationPolicy
apiVersion: nmstate.io/v1
metadata:
  name: <name>
spec:
  nodeSelector:
    kubernetes.io/hostname: <node_name>
  desiredState:
    interfaces:
      - name: <tunnel_name>
        type: ipsec
        state: absent

```

where:

#### **name**

Specifies a name for the node network configuration policy.

#### **node\_name**

Specifies the name of the node where the IPsec tunnel that you want to remove exists.

#### **tunnel\_name**

Specifies the interface name for the existing IPsec tunnel.

2. To remove the IPsec tunnel, enter the following command:

```
$ oc apply -f remove-ipsec-tunnel.yaml
```

### 5.6.9. Disabling IPsec encryption

As a cluster administrator, you can disable IPsec encryption.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster with a user with **cluster-admin** privileges.

#### Procedure

1. To disable IPsec encryption, enter the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{
  "spec": {
    "defaultNetwork": {
      "ovnKubernetesConfig": {
        "ipsecConfig": {
          "mode": "Disabled"
        }
      }
    }
  }
}'
```

2. Optional: You can increase the size of your cluster MTU by **46** bytes because there is no longer any overhead from the IPsec ESP header in IP packets.

## 5.6.10. Additional resources

- [Configuring a VPN with IPsec](#) in Red Hat Enterprise Linux (RHEL) 9
- [Installing Butane](#)
- [About the OVN-Kubernetes Container Network Interface \(CNI\) network plugin](#)
- [Changing the MTU for the cluster network](#)
- [Network \[operator.openshift.io/v1\] API](#)

## 5.7. ZERO TRUST NETWORKING

Zero trust is an approach to designing security architectures based on the premise that every interaction begins in an untrusted state. This contrasts with traditional architectures, which might determine trustworthiness based on whether communication starts inside a firewall. More specifically, zero trust attempts to close gaps in security architectures that rely on implicit trust models and one-time authentication.

OpenShift Container Platform can add some zero trust networking capabilities to containers running on the platform without requiring changes to the containers or the software running in them. There are also several products that Red Hat offers that can further augment the zero trust networking capabilities of containers. If you have the ability to change the software running in the containers, then there are other projects that Red Hat supports that can add further capabilities.

Explore the following targeted capabilities of zero trust networking.

### 5.7.1. Root of trust

Public certificates and private keys are critical to zero trust networking. These are used to identify components to one another, authenticate, and to secure traffic. The certificates are signed by other certificates, and there is a chain of trust to a root certificate authority (CA). Everything participating in the network needs to ultimately have the public key for a root CA so that it can validate the chain of trust. For public-facing things, these are usually the set of root CAs that are globally known, and whose keys are distributed with operating systems, web browsers, and so on. However, it is possible to run a private CA for a cluster or a corporation if the certificate of the private CA is distributed to all parties.

Leverage:

- OpenShift Container Platform: OpenShift creates a [cluster CA at installation](#) that is used to secure the cluster resources. However, OpenShift Container Platform can also create and sign [certificates for services](#) in the cluster, and can inject the cluster CA bundle into a pod if requested. [Service certificates](#) created and signed by OpenShift Container Platform have a 26-month time to live (TTL) and are rotated automatically at 13 months. They can also be rotated manually if necessary.
- [OpenShift cert-manager Operator](#): cert-manager allows you to request keys that are signed by an external root of trust. There are many configurable issuers to integrate with external issuers, along with ways to run with a delegated signing certificate. The cert-manager API can be used by other software in zero trust networking to request the necessary certificates (for example, Red Hat OpenShift Service Mesh), or can be used directly by customer software.

### 5.7.2. Traffic authentication and encryption

Ensure that all traffic on the wire is encrypted and the endpoints are identifiable. An example of this is Mutual TLS, or mTLS, which is a method for mutual authentication.

Leverage:

- OpenShift Container Platform: With transparent [pod-to-pod IPsec](#), the source and destination of the traffic can be identified by the IP address. There is the capability for egress traffic to be [encrypted using IPsec](#). By using the [egress IP](#) feature, the source IP address of the traffic can be used to identify the source of the traffic inside the cluster.
- [Red Hat OpenShift Service Mesh](#): Provides powerful [mTLS capabilities](#) that can transparently augment traffic leaving a pod to provide authentication and encryption.
- [OpenShift cert-manager Operator](#): Use custom resource definitions (CRDs) to request certificates that can be mounted for your programs to use for SSL/TLS protocols.

### 5.7.3. Identification and authentication

After you have the ability to mint certificates using a CA, you can use it to establish trust relationships by verification of the identity of the other end of a connection – either a user or a client machine. This also requires management of certificate lifecycles to limit use if compromised.

Leverage:

- OpenShift Container Platform: Cluster-signed [service certificates](#) to ensure that a client is talking to a trusted endpoint. This requires that the service uses SSL/TLS and that the client uses the [cluster CA](#). The client identity must be provided using some other means.
- [Red Hat Single Sign-On](#): Provides request authentication integration with enterprise user directories or third-party identity providers.
- [Red Hat OpenShift Service Mesh](#): [Transparent upgrade](#) of connections to mTLS, auto-rotation, custom certificate expiration, and request authentication with JSON web token (JWT).
- [OpenShift cert-manager Operator](#): Creation and management of certificates for use by your application. Certificates can be controlled by CRDs and mounted as secrets, or your application can be changed to interact directly with the cert-manager API.

### 5.7.4. Inter-service authorization

It is critical to be able to control access to services based on the identity of the requester. This is done by the platform and does not require each application to implement it. That allows better auditing and inspection of the policies.

Leverage:

- OpenShift Container Platform: Can enforce isolation in the networking layer of the platform using the Kubernetes [NetworkPolicy](#) and [AdminNetworkPolicy](#) objects.
- [Red Hat OpenShift Service Mesh](#): Sophisticated L4 and L7 [control of traffic](#) using standard Istio objects and using mTLS to identify the source and destination of traffic and then apply policies based on that information.

### 5.7.5. Transaction-level verification

In addition to the ability to identify and authenticate connections, it is also useful to control access to individual transactions. This can include rate-limiting by source, observability, and semantic validation that a transaction is well formed.

Leverage:

- [Red Hat OpenShift Service Mesh](#) : Perform L7 inspection of requests, rejecting malformed HTTP requests, transaction-level [observability and reporting](#). Service Mesh can also provide [request-based authentication](#) using JWT.

### 5.7.6. Risk assessment

As the number of security policies in a cluster increase, visualization of what the policies allow and deny becomes increasingly important. These tools make it easier to create, visualize, and manage cluster security policies.

Leverage:

- [Red Hat OpenShift Service Mesh](#) : Create and visualize Kubernetes **NetworkPolicy** and **AdminNetworkPolicy**, and OpenShift Networking **EgressFirewall** objects using the [OpenShift web console](#).
- [Red Hat Advanced Cluster Security for Kubernetes](#) : Advanced [visualization of objects](#).

### 5.7.7. Site-wide policy enforcement and distribution

After deploying applications on a cluster, it becomes challenging to manage all of the objects that make up the security rules. It becomes critical to be able to apply site-wide policies and audit the deployed objects for compliance with the policies. This should allow for delegation of some permissions to users and cluster administrators within defined bounds, and should allow for exceptions to the policies if necessary.

Leverage:

- [Red Hat OpenShift Service Mesh](#) : RBAC to [control policy objects](#) and delegate control.
- [Red Hat Advanced Cluster Security for Kubernetes](#) : [Policy enforcement](#) engine.
- [Red Hat Advanced Cluster Management \(RHACM\) for Kubernetes](#) : Centralized policy control.

### 5.7.8. Observability for constant, and retrospective, evaluation

After you have a running cluster, you want to be able to observe the traffic and verify that the traffic comports with the defined rules. This is important for intrusion detection, forensics, and is helpful for operational load management.

Leverage:

- [Network Observability Operator](#): Allows for inspection, monitoring, and alerting on network connections to pods and nodes in the cluster.
- [Red Hat Advanced Cluster Management \(RHACM\) for Kubernetes](#) : Monitors, collects, and evaluates system-level events such as process execution, network connections and flows, and privilege escalation. It can determine a baseline for a cluster, and then detect anomalous activity and alert you about it.

- [Red Hat OpenShift Service Mesh](#): Can [monitor traffic](#) entering and leaving a pod.
- [Red Hat OpenShift Distributed Tracing Platform](#): For suitably instrumented applications, you can see all traffic associated with a particular action as it splits into sub-requests to microservices. This allows you to identify bottlenecks within a distributed application.

### 5.7.9. Endpoint security

It is important to be able to trust that the software running the services in your cluster has not been compromised. For example, you might need to ensure that certified images are run on trusted hardware, and have policies to only allow connections to or from an endpoint based on endpoint characteristics.

Leverage:

- OpenShift Container Platform: Secureboot can ensure that the nodes in the cluster are running trusted software, so the platform itself (including the container runtime) have not been tampered with. You can configure OpenShift Container Platform to only run images that have been [signed by certain signatures](#).
- [Red Hat Trusted Artifact Signer](#): This can be used in a trusted build chain and produce signed container images.

### 5.7.10. Extending trust outside of the cluster

You might want to extend trust outside of the cluster by allowing a cluster to mint CAs for a subdomain. Alternatively, you might want to attest to workload identity in the cluster to a remote endpoint.

Leverage:

- [OpenShift cert-manager Operator](#): You can use cert-manager to manage delegated CAs so that you can distribute trust across different clusters, or through your organization.
- [Red Hat OpenShift Service Mesh](#): Can use SPIFFE to provide remote attestation of workloads to endpoints running in remote or local clusters.

# CHAPTER 6. CONFIGURING AN INGRESS CONTROLLER FOR MANUAL DNS MANAGEMENT

As a cluster administrator, when you create an Ingress Controller, the Operator manages the DNS records automatically. This has some limitations when the required DNS zone is different from the cluster DNS zone or when the DNS zone is hosted outside the cloud provider.

As a cluster administrator, you can configure an Ingress Controller to stop automatic DNS management and start manual DNS management. Set **dnsManagementPolicy** to specify when it should be automatically or manually managed.

When you change an Ingress Controller from **Managed** to **Unmanaged** DNS management policy, the Operator does not clean up the previous wildcard DNS record provisioned on the cloud. When you change an Ingress Controller from **Unmanaged** to **Managed** DNS management policy, the Operator attempts to create the DNS record on the cloud provider if it does not exist or updates the DNS record if it already exists.



## IMPORTANT

When you set **dnsManagementPolicy** to **unmanaged**, you have to manually manage the lifecycle of the wildcard DNS record on the cloud provider.

## 6.1. MANAGED DNS MANAGEMENT POLICY

The **Managed** DNS management policy for Ingress Controllers ensures that the lifecycle of the wildcard DNS record on the cloud provider is automatically managed by the Operator.

## 6.2. UNMANAGED DNS MANAGEMENT POLICY

The **Unmanaged** DNS management policy for Ingress Controllers ensures that the lifecycle of the wildcard DNS record on the cloud provider is not automatically managed, instead it becomes the responsibility of the cluster administrator.



## NOTE

On the AWS cloud platform, if the domain on the Ingress Controller does not match with **dnsConfig.Spec.BaseDomain** then the DNS management policy is automatically set to **Unmanaged**.

## 6.3. CREATING A CUSTOM INGRESS CONTROLLER WITH THE UNMANAGED DNS MANAGEMENT POLICY

As a cluster administrator, you can create a new custom Ingress Controller with the **Unmanaged** DNS management policy.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Create a custom resource (CR) file named **sample-ingress.yaml** containing the following:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> ①
spec:
  domain: <domain> ②
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External ③
  dnsManagementPolicy: Unmanaged ④
```

- 1 Specify the **<name>** with a name for the **IngressController** object.
- 2 Specify the **domain** based on the DNS record that was created as a prerequisite.
- 3 Specify the **scope** as **External** to expose the load balancer externally.
- 4 **dnsManagementPolicy** indicates if the Ingress Controller is managing the lifecycle of the wildcard DNS record associated with the load balancer. The valid values are **Managed** and **Unmanaged**. The default value is **Managed**.

2. Save the file to apply the changes.

```
oc apply -f <name>.yaml ①
```

## 6.4. MODIFYING AN EXISTING INGRESS CONTROLLER

As a cluster administrator, you can modify an existing Ingress Controller to manually manage the DNS record lifecycle.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Modify the chosen **IngressController** to set **dnsManagementPolicy**:

```
SCOPE=$(oc -n openshift-ingress-operator get ingresscontroller <name> -o=jsonpath=".status.endpointPublishingStrategy.loadBalancer.scope")  
  
oc -n openshift-ingress-operator patch ingresscontrollers/<name> --type=merge --patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":{"dnsManagementPolicy":"Unmanaged","scope":"$SCOPE"}}}}'
```

2. Optional: You can delete the associated DNS record in the cloud provider.

## 6.5. ADDITIONAL RESOURCES

- [Ingress Controller configuration parameters](#)

# CHAPTER 7. VERIFYING CONNECTIVITY TO AN ENDPOINT

The Cluster Network Operator (CNO) runs a controller, the connectivity check controller, that performs a connection health check between resources within your cluster. By reviewing the results of the health checks, you can diagnose connection problems or eliminate network connectivity as the cause of an issue that you are investigating.

## 7.1. CONNECTION HEALTH CHECKS PERFORMED

To verify that cluster resources are reachable, a TCP connection is made to each of the following cluster API services:

- Kubernetes API server service
- Kubernetes API server endpoints
- OpenShift API server service
- OpenShift API server endpoints
- Load balancers

To verify that services and service endpoints are reachable on every node in the cluster, a TCP connection is made to each of the following targets:

- Health check target service
- Health check target endpoints

## 7.2. IMPLEMENTATION OF CONNECTION HEALTH CHECKS

The connectivity check controller orchestrates connection verification checks in your cluster. The results for the connection tests are stored in **PodNetworkConnectivity** objects in the **openshift-network-diagnostics** namespace. Connection tests are performed every minute in parallel.

The Cluster Network Operator (CNO) deploys several resources to the cluster to send and receive connectivity health checks:

### Health check source

This program deploys in a single pod replica set managed by a **Deployment** object. The program consumes **PodNetworkConnectivity** objects and connects to the **spec.targetEndpoint** specified in each object.

### Health check target

A pod deployed as part of a daemon set on every node in the cluster. The pod listens for inbound health checks. The presence of this pod on every node allows for the testing of connectivity to each node.

You can configure the nodes which network connectivity sources and targets run on with a node selector. Additionally, you can specify permissible *tolerations* for source and target pods. The configuration is defined in the singleton **cluster** custom resource of the **Network** API in the **config.openshift.io/v1** API group.

Pod scheduling occurs after you have updated the configuration. Therefore, you must apply node labels that you intend to use in your selectors before updating the configuration. Labels applied after updating your network connectivity check pod placement are ignored.

Refer to the default configuration in the following YAML:

### Default configuration for connectivity source and target pods

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
  networkDiagnostics: ①
    mode: "All" ②
    sourcePlacement: ③
      nodeSelector:
        checkNodes: groupA
      tolerations:
        - key: myTaint
          effect: NoSchedule
          operator: Exists
    targetPlacement: ④
      nodeSelector:
        checkNodes: groupB
      tolerations:
        - key: myOtherTaint
          effect: NoExecute
          operator: Exists
```

- ① ① Specifies the network diagnostics configuration. If a value is not specified or an empty object is specified, and **spec.disableNetworkDiagnostics=true** is set in the **network.operator.openshift.io** custom resource named **cluster**, network diagnostics are disabled. If set, this value overrides **spec.disableNetworkDiagnostics=true**.
- ② Specifies the diagnostics mode. The value can be the empty string, **All**, or **Disabled**. The empty string is equivalent to specifying **All**.
- ③ Optional: Specifies a selector for connectivity check source pods. You can use the **nodeSelector** and **tolerations** fields to further specify the **sourceNode** pods. You do not have to use both **nodeSelector** and **tolerations**, however, for both the source and target pods. These are optional fields that can be omitted.
- ④ Optional: Specifies a selector for connectivity check target pods. You can use the **nodeSelector** and **tolerations** fields to further specify the **targetNode** pods. You do not have to use both **nodeSelector** and **tolerations**, however, for both the source and target pods. These are optional fields that can be omitted.

## 7.3. CONFIGURING POD CONNECTIVITY CHECK PLACEMENT

As a cluster administrator, you can configure which nodes the connectivity check pods run by modifying the **network.config.openshift.io** object named **cluster**.

## Prerequisites

- Install the OpenShift CLI (**oc**).

## Procedure

1. To edit the connectivity check configuration, enter the following command:

```
$ oc edit network.config.openshift.io cluster
```

2. In the text editor, update the **networkDiagnostics** stanza to specify the node selectors that you want for the source and target pods.

3. To commit your changes, save your changes and exit the text editor.

## Verification

To verify that the source and target pods are running on the intended nodes, enter the following command:

```
$ oc get pods -n openshift-network-diagnostics -o wide
```

## Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
<b>NOMINATED NODE READINESS GATES</b>						
network-check-source-84c69dbd6b-p8f7n	1/1	Running	0	9h	10.131.0.8	ip-10-0-40-197.us-east-2.compute.internal
network-check-target-46pct	1/1	Running	0	9h	10.131.0.6	ip-10-0-40-197.us-east-2.compute.internal
network-check-target-8kwgf	1/1	Running	0	9h	10.128.2.4	ip-10-0-95-74.us-east-2.compute.internal
network-check-target-jc6n7	1/1	Running	0	9h	10.129.2.4	ip-10-0-21-151.us-east-2.compute.internal
network-check-target-lvwnn	1/1	Running	0	9h	10.128.0.7	ip-10-0-17-129.us-east-2.compute.internal
network-check-target-nslvj	1/1	Running	0	9h	10.130.0.7	ip-10-0-89-148.us-east-2.compute.internal
network-check-target-z2sfx	1/1	Running	0	9h	10.129.0.4	ip-10-0-60-253.us-east-2.compute.internal

## 7.4. PODNETWORKCONNECTIVITYCHECK OBJECT FIELDS

The **PodNetworkConnectivityCheck** object fields are described in the following tables.

Table 7.1. PodNetworkConnectivityCheck object fields

Field	Type	Description
-------	------	-------------

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	The name of the object in the following format: <b>&lt;source&gt;-to-&lt;target&gt;</b> . The destination described by <b>&lt;target&gt;</b> includes one of following strings: <ul style="list-style-type: none"> <li>• <b>load-balancer-api-external</b></li> <li>• <b>load-balancer-api-internal</b></li> <li>• <b>kubernetes-apiserver-endpoint</b></li> <li>• <b>kubernetes-apiserver-service-cluster</b></li> <li>• <b>network-check-target</b></li> <li>• <b>openshift-apiserver-endpoint</b></li> <li>• <b>openshift-apiserver-service-cluster</b></li> </ul>
<b>metadata.namespace</b>	<b>string</b>	The namespace that the object is associated with. This value is always <b>openshift-network-diagnostics</b> .
<b>spec.sourcePod</b>	<b>string</b>	The name of the pod where the connection check originates, such as <b>network-check-source-596b4c6566-rgh92</b> .
<b>spec.targetEndpoint</b>	<b>string</b>	The target of the connection check, such as <b>api.devcluster.example.com:6443</b> .
<b>spec.tlsClientCert</b>	<b>object</b>	Configuration for the TLS certificate to use.
<b>spec.tlsClientCert.name</b>	<b>string</b>	The name of the TLS certificate used, if any. The default value is an empty string.
<b>status</b>	<b>object</b>	An object representing the condition of the connection test and logs of recent connection successes and failures.
<b>status.conditions</b>	<b>array</b>	The latest status of the connection check and any previous statuses.
<b>status.failures</b>	<b>array</b>	Connection test logs from unsuccessful attempts.
<b>status.outages</b>	<b>array</b>	Connection test logs covering the time periods of any outages.
<b>status.successes</b>	<b>array</b>	Connection test logs from successful attempts.

The following table describes the fields for objects in the **status.conditions** array:

**Table 7.2. status.conditions**

Field	Type	Description
<b>lastTransitionTime</b>	<b>string</b>	The time that the condition of the connection transitioned from one status to another.
<b>message</b>	<b>string</b>	The details about last transition in a human readable format.
<b>reason</b>	<b>string</b>	The last status of the transition in a machine readable format.
<b>status</b>	<b>string</b>	The status of the condition.
<b>type</b>	<b>string</b>	The type of the condition.

The following table describes the fields for objects in the **status.conditions** array:

**Table 7.3. status.outages**

Field	Type	Description
<b>end</b>	<b>string</b>	The timestamp from when the connection failure is resolved.
<b>endLogs</b>	<b>array</b>	Connection log entries, including the log entry related to the successful end of the outage.
<b>message</b>	<b>string</b>	A summary of outage details in a human readable format.
<b>start</b>	<b>string</b>	The timestamp from when the connection failure is first detected.
<b>startLogs</b>	<b>array</b>	Connection log entries, including the original failure.

### Connection log fields

The fields for a connection log entry are described in the following table. The object is used in the following fields:

- **status.failures[]**
- **status.successes[]**
- **status.outages[].startLogs[]**
- **status.outages[].endLogs[]**

**Table 7.4. Connection log object**

Field	Type	Description
<b>latency</b>	<b>string</b>	Records the duration of the action.
<b>message</b>	<b>string</b>	Provides the status in a human readable format.
<b>reason</b>	<b>string</b>	Provides the reason for status in a machine readable format. The value is one of <b>TCPConnect</b> , <b>TCPConnectError</b> , <b>DNSResolve</b> , <b>DNSError</b> .
<b>success</b>	<b>boolean</b>	Indicates if the log entry is a success or failure.
<b>time</b>	<b>string</b>	The start time of connection check.

## 7.5. VERIFYING NETWORK CONNECTIVITY FOR AN ENDPOINT

As a cluster administrator, you can verify the connectivity of an endpoint, such as an API server, load balancer, service, or pod, and verify that network diagnostics is enabled.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

### Procedure

1. To confirm that network diagnostics are enabled, enter the following command:

```
$ oc get network.config.openshift.io cluster -o yaml
```

### Example output

```
# ...
status:
# ...
conditions:
- lastTransitionTime: "2024-05-27T08:28:39Z"
  message: ""
  reason: AsExpected
  status: "True"
  type: NetworkDiagnosticsAvailable
```

2. To list the current **PodNetworkConnectivityCheck** objects, enter the following command:

```
$ oc get podnetworkconnectivitycheck -n openshift-network-diagnostics
```

### Example output

NAME	AGE
------	-----

```

network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
endpoint-ci-In-x5sv9rb-f76d1-4rzrp-master-0 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
endpoint-ci-In-x5sv9rb-f76d1-4rzrp-master-1 73m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
endpoint-ci-In-x5sv9rb-f76d1-4rzrp-master-2 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-
service-cluster 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-default-
service-cluster 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-
external 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-
internal 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
In-x5sv9rb-f76d1-4rzrp-master-0 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
In-x5sv9rb-f76d1-4rzrp-master-1 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
In-x5sv9rb-f76d1-4rzrp-master-2 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh 74m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
In-x5sv9rb-f76d1-4rzrp-worker-c-n8mbf 74m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
In-x5sv9rb-f76d1-4rzrp-worker-d-4hnrz 74m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-
service-cluster 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-In-x5sv9rb-f76d1-4rzrp-master-0 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-In-x5sv9rb-f76d1-4rzrp-master-1 75m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-In-x5sv9rb-f76d1-4rzrp-master-2 74m
network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
service-cluster 75m

```

3. View the connection test logs:

- From the output of the previous command, identify the endpoint that you want to review the connectivity logs for.
- To view the object, enter the following command:

```
$ oc get podnetworkconnectivitycheck <name> \
-n openshift-network-diagnostics -o yaml
```

where **<name>** specifies the name of the **PodNetworkConnectivityCheck** object.

#### Example output

```

apiVersion: controlplane.operator.openshift.io/v1alpha1
kind: PodNetworkConnectivityCheck
metadata:
  name: network-check-source-ci-In-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-
apiserver-endpoint-ci-In-x5sv9rb-f76d1-4rzrp-master-0

```

```
namespace: openshift-network-diagnostics
...
spec:
sourcePod: network-check-source-7c88f6d9f-hmg2f
targetEndpoint: 10.0.0.4:6443
tlsClientCert:
  name: ""
status:
conditions:
- lastTransitionTime: "2021-01-13T20:11:34Z"
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnectSuccess
  status: "True"
  type: Reachable
failures:
- latency: 2.241775ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
    to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
    connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:10:34Z"
- latency: 2.582129ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
    to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
    connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:09:34Z"
- latency: 3.483578ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
    to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
    connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:08:34Z"
outages:
- end: "2021-01-13T20:11:34Z"
  endLogs:
  - latency: 2.032018ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      tcp connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T20:11:34Z"
  - latency: 2.241775ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:10:34Z"
  - latency: 2.582129ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
```

```
connect: connection refused'
reason: TCPConnectError
success: false
time: "2021-01-13T20:09:34Z"
- latency: 3.483578ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
    failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:08:34Z"
message: Connectivity restored after 2m59.999789186s
start: "2021-01-13T20:08:34Z"
startLogs:
- latency: 3.483578ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
    failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:08:34Z"
successes:
- latency: 2.845865ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:14:34Z"
- latency: 2.926345ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:13:34Z"
- latency: 2.895796ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:12:34Z"
- latency: 2.696844ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:11:34Z"
- latency: 1.502064ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:10:34Z"
- latency: 1.388857ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
```

```
success: true
time: "2021-01-13T21:09:34Z"
- latency: 1.906383ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
success: true
time: "2021-01-13T21:08:34Z"
- latency: 2.089073ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
success: true
time: "2021-01-13T21:07:34Z"
- latency: 2.156994ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
success: true
time: "2021-01-13T21:06:34Z"
- latency: 1.777043ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
success: true
time: "2021-01-13T21:05:34Z"
```

# CHAPTER 8. CHANGING THE MTU FOR THE CLUSTER NETWORK

As a cluster administrator, you can change the MTU for the cluster network after cluster installation. This change is disruptive as cluster nodes must be rebooted to finalize the MTU change.

## 8.1. ABOUT THE CLUSTER MTU

During installation the maximum transmission unit (MTU) for the cluster network is detected automatically based on the MTU of the primary network interface of nodes in the cluster. You do not usually need to override the detected MTU.

You might want to change the MTU of the cluster network for several reasons:

- The MTU detected during cluster installation is not correct for your infrastructure.
- Your cluster infrastructure now requires a different MTU, such as from the addition of nodes that need a different MTU for optimal performance.

Only the OVN-Kubernetes cluster network plugin supports changing the MTU value.

### 8.1.1. Service interruption considerations

When you initiate an MTU change on your cluster the following effects might impact service availability:

- At least two rolling reboots are required to complete the migration to a new MTU. During this time, some nodes are not available as they restart.
- Specific applications deployed to the cluster with shorter timeout intervals than the absolute TCP timeout interval might experience disruption during the MTU change.

### 8.1.2. MTU value selection

When planning your MTU migration there are two related but distinct MTU values to consider.

- **Hardware MTU:** This MTU value is set based on the specifics of your network infrastructure.
- **Cluster network MTU:** This MTU value is always less than your hardware MTU to account for the cluster network overlay overhead. The specific overhead is determined by your network plugin. For OVN-Kubernetes, the overhead is **100** bytes.

If your cluster requires different MTU values for different nodes, you must subtract the overhead value for your network plugin from the lowest MTU value that is used by any node in your cluster. For example, if some nodes in your cluster have an MTU of **9001**, and some have an MTU of **1500**, you must set this value to **1400**.



#### IMPORTANT

To avoid selecting an MTU value that is not acceptable by a node, verify the maximum MTU value (**maxmtu**) that is accepted by the network interface by using the **ip -d link** command.

### 8.1.3. How the migration process works

The following table summarizes the migration process by segmenting between the user-initiated steps in the process and the actions that the migration performs in response.

**Table 8.1. Live migration of the cluster MTU**

User-initiated steps	OpenShift Container Platform activity
<p>Set the following values in the Cluster Network Operator configuration:</p> <ul style="list-style-type: none"> <li>• <b>spec.migration.mtu.machine.to</b></li> <li>• <b>spec.migration.mtu.network.from</b></li> <li>• <b>spec.migration.mtu.network.to</b></li> </ul>	<p><b>Cluster Network Operator (CNO)</b> Confirms that each field is set to a valid value.</p> <ul style="list-style-type: none"> <li>• The <b>mtu.machine.to</b> must be set to either the new hardware MTU or to the current hardware MTU if the MTU for the hardware is not changing. This value is transient and is used as part of the migration process. Separately, if you specify a hardware MTU that is different from your existing hardware MTU value, you must manually configure the MTU to persist by other means, such as with a machine config, DHCP setting, or a Linux kernel command line.</li> <li>• The <b>mtu.network.from</b> field must equal the <b>network.status.clusterNetworkMTU</b> field, which is the current MTU of the cluster network.</li> <li>• The <b>mtu.network.to</b> field must be set to the target cluster network MTU and must be lower than the hardware MTU to allow for the overlay overhead of the network plugin. For OVN-Kubernetes, the overhead is <b>100</b> bytes.</li> </ul> <p>If the values provided are valid, the CNO writes out a new temporary configuration with the MTU for the cluster network set to the value of the <b>mtu.network.to</b> field.</p> <p><b>Machine Config Operator (MCO)</b> Performs a rolling reboot of each node in the cluster.</p>
<p>Reconfigure the MTU of the primary network interface for the nodes on the cluster. You can use a variety of methods to accomplish this, including:</p> <ul style="list-style-type: none"> <li>• Deploying a new NetworkManager connection profile with the MTU change</li> <li>• Changing the MTU through a DHCP server setting</li> <li>• Changing the MTU through boot parameters</li> </ul>	N/A
<p>Set the <b>mtu</b> value in the CNO configuration for the network plugin and set <b>spec.migration</b> to <b>null</b>.</p>	<p><b>Machine Config Operator (MCO)</b> Performs a rolling reboot of each node in the cluster with the new MTU configuration.</p>

## 8.2. CHANGING THE CLUSTER NETWORK MTU

As a cluster administrator, you can increase or decrease the maximum transmission unit (MTU) for your cluster.



### IMPORTANT

You cannot roll back an MTU value for nodes during the MTU migration process, but you can roll back the value after the MTU migration process completes.

The migration is disruptive and nodes in your cluster might be temporarily unavailable as the MTU update takes effect.

The following procedure describes how to change the cluster network MTU by using either machine configs, Dynamic Host Configuration Protocol (DHCP), or an ISO image. If you use either the DHCP or ISO approaches, you must refer to configuration artifacts that you kept after installing your cluster to complete the procedure.

### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster using an account with **cluster-admin** permissions.
- You have identified the target MTU for your cluster. The MTU for the OVN-Kubernetes network plugin must be set to **100** less than the lowest hardware MTU value in your cluster.
- If your nodes are physical machines, ensure that the cluster network and the connected network switches support jumbo frames.
- If your nodes are virtual machines (VMs), ensure that the hypervisor and the connected network switches support jumbo frames.

### Procedure

1. To obtain the current MTU for the cluster network, enter the following command:

```
$ oc describe network.config cluster
```

### Example output

```
...
Status:
Cluster Network:
  Cidr:      10.217.0.0/22
  Host Prefix: 23
  Cluster Network MTU: 1400
  Network Type: OVNKubernetes
  Service Network:
    10.217.4.0/23
...
...
```

2. Prepare your configuration for the hardware MTU:

- If your hardware MTU is specified with DHCP, update your DHCP configuration such as with the following dnsmasq configuration:

```
    dhcp-option-force=26,<mtu>
```

where:

**<mtu>**

Specifies the hardware MTU for the DHCP server to advertise.

- If your hardware MTU is specified with a kernel command line with PXE, update that configuration accordingly.
- If your hardware MTU is specified in a NetworkManager connection configuration, complete the following steps. This approach is the default for OpenShift Container Platform if you do not explicitly specify your network configuration with DHCP, a kernel command line, or some other method. Your cluster nodes must all use the same underlying network configuration for the following procedure to work unmodified.
  - i. Find the primary network interface by entering the following command:
 

```
$ oc debug node/<node_name> -- chroot /host nmcli -g connection.interface-name c show ovs-if-phys0
```

where:

**<node\_name>**

Specifies the name of a node in your cluster.

- ii. Create the following NetworkManager configuration in the **<interface>-mtu.conf** file:

### Example NetworkManager connection configuration

```
[connection-<interface>-mtu]
match-device=interface-name:<interface>
ethernet.mtu=<mtu>
```

where:

**<mtu>**

Specifies the new hardware MTU value.

**<interface>**

Specifies the primary network interface name.

- iii. Create two **MachineConfig** objects, one for the control plane nodes and another for the worker nodes in your cluster:

- A. Create the following Butane config in the **control-plane-interface.bu** file:

**NOTE**

The [Butane version](#) you specify in the config file should match the OpenShift Container Platform version and always ends in **0**. For example, **4.18.0**. See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.18.0
metadata:
  name: 01-control-plane-interface
  labels:
    machineconfiguration.openshift.io/role: master
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf 1
      contents:
        local: <interface>-mtu.conf 2
      mode: 0600
```

- 1** Specify the NetworkManager connection name for the primary network interface.
- 2** Specify the local filename for the updated NetworkManager configuration file from the previous step.

B. Create the following Butane config in the **worker-interface.bu** file:

**NOTE**

The [Butane version](#) you specify in the config file should match the OpenShift Container Platform version and always ends in **0**. For example, **4.18.0**. See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.18.0
metadata:
  name: 01-worker-interface
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf 1
      contents:
        local: <interface>-mtu.conf 2
      mode: 0600
```

- 1** Specify the NetworkManager connection name for the primary network interface.
- 2** Specify the local filename for the updated NetworkManager configuration file from the previous step.

- C. Create **MachineConfig** objects from the Butane configs by running the following command:

```
$ for manifest in control-plane-interface worker-interface; do
    butane --files-dir . $manifest.bu > $manifest.yaml
done
```



### WARNING

Do not apply these machine configs until explicitly instructed later in this procedure. Applying these machine configs now causes a loss of stability for the cluster.

3. To begin the MTU migration, specify the migration configuration by entering the following command. The Machine Config Operator performs a rolling reboot of the nodes in the cluster in preparation for the MTU change.

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
'{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to": <overlay_to> } , "machine": { "to" : <machine_to> } } } }'
```

where:

#### **<overlay\_from>**

Specifies the current cluster network MTU value.

#### **<overlay\_to>**

Specifies the target MTU for the cluster network. This value is set relative to the value of **<machine\_to>**. For OVN-Kubernetes, this value must be **100** less than the value of **<machine\_to>**.

#### **<machine\_to>**

Specifies the MTU for the primary network interface on the underlying host network.

### Example that increases the cluster MTU

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
'{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 9000 } , "machine": { "to" : 9100} } } }'
```

4. As the Machine Config Operator updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get machineconfigpools
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.



## NOTE

By default, the Machine Config Operator updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

5. Confirm the status of the new machine configuration on the hosts:

- a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

```
$ oc describe node | egrep "hostname|machineconfig"
```

### Example output

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

- b. Verify that the following statements are true:

- The value of **machineconfiguration.openshift.io/state** field is **Done**.
- The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

- c. To confirm that the machine config is correct, enter the following command:

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

where **<config\_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

The machine config must include the following update to the systemd configuration:

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

6. Update the underlying network interface MTU value:

- If you are specifying the new MTU with a NetworkManager connection configuration, enter the following command. The MachineConfig Operator automatically performs a rolling reboot of the nodes in your cluster.

```
$ for manifest in control-plane-interface worker-interface; do
  oc create -f $manifest.yaml
done
```

- If you are specifying the new MTU with a DHCP server option or a kernel command line and PXE, make the necessary changes for your infrastructure.

- As the Machine Config Operator updates machines in each machine config pool, it reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get machineconfigpools
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.



#### NOTE

By default, the Machine Config Operator updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

- Confirm the status of the new machine configuration on the hosts:

- To list the machine configuration state and the name of the applied machine configuration, enter the following command:

```
$ oc describe node | egrep "hostname|machineconfig"
```

#### Example output

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

Verify that the following statements are true:

- The value of **machineconfiguration.openshift.io/state** field is **Done**.
- The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

- To confirm that the machine config is correct, enter the following command:

```
$ oc get machineconfig <config_name> -o yaml | grep path:
```

where **<config\_name>** is the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

If the machine config is successfully deployed, the previous output contains the **/etc/NetworkManager/conf.d/99-<interface>-mtu.conf** file path and the **ExecStart=/usr/local/bin/mtu-migration.sh** line.

- To finalize the MTU migration, enter the following command for the OVN-Kubernetes network plugin:

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
'{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu": <mtu> }}}}'
```

where:

**<mtu>**

Specifies the new cluster network MTU that you specified with **<overlay\_to>**.

- After finalizing the MTU migration, each machine config pool node is rebooted one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get machineconfigpools
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

## Verification

- To get the current MTU for the cluster network, enter the following command:

```
$ oc describe network.config cluster
```

- Get the current MTU for the primary network interface of a node:

- To list the nodes in your cluster, enter the following command:

```
$ oc get nodes
```

- To obtain the current MTU setting for the primary network interface on a node, enter the following command:

```
$ oc debug node/<node> -- chroot /host ip address show <interface>
```

where:

**<node>**

Specifies a node from the output from the previous step.

**<interface>**

Specifies the primary network interface name for the node.

## Example output

```
ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8051
```

## 8.3. ADDITIONAL RESOURCES

- Using advanced networking options for PXE and ISO installations
- Manually creating NetworkManager profiles in key file format
- Configuring a dynamic Ethernet connection using nmcli

# CHAPTER 9. CONFIGURING THE NODE PORT SERVICE RANGE

As a cluster administrator, you can expand the available node port range. If your cluster uses of a large number of node ports, you might need to increase the number of available ports.

The default port range is **30000-32767**. You can never reduce the port range, even if you first expand it beyond the default range.

## 9.1. PREREQUISITES

- Your cluster infrastructure must allow access to the ports that you specify within the expanded range. For example, if you expand the node port range to **30000-32900**, the inclusive port range of **32768-32900** must be allowed by your firewall or packet filtering configuration.

## 9.2. EXPANDING THE NODE PORT RANGE

You can expand the node port range for the cluster.



### IMPORTANT

You can expand the node port range into the protected port range, which is between 0 and 32767. However, after expansion, you cannot change the range. Attempting to change the range returns the following error: **The Network "cluster" is invalid: spec.serviceNodePortRange: Invalid value: "30000-32767": new service node port range 30000-32767 does not completely cover the previous range 0-32767.**

Before making changes, ensure that the new range you set is appropriate for your cluster.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster with a user with **cluster-admin** privileges.

### Procedure

1. To expand the node port range, enter the following command. Replace **<port>** with the largest port number in the new range.

```
$ oc patch network.config.openshift.io cluster --type=merge -p \  
{  
  "spec":  
    { "serviceNodePortRange": "30000-<port>" }  
}
```

**TIP**

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

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-<port>"
```

**Example output**

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

2. To confirm that the configuration is active, enter the following command. It can take several minutes for the update 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":["30000-33000"]
```

### 9.3. ADDITIONAL RESOURCES

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

# CHAPTER 10. 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

## 10.1. 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
```

- 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}]
```

## 10.2. ADDITIONAL RESOURCES

- [Red Hat OpenShift Network Calculator](#)
- [About the OVN-Kubernetes network plugin](#)

# CHAPTER 11. CONFIGURING IP FAILOVER

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

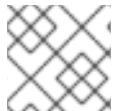
IP failover uses [Keepalived](#) to host a set of externally accessible Virtual IP (VIP) addresses on a set of hosts. Each VIP address is only serviced by a single host at a time. Keepalived uses the Virtual Router Redundancy Protocol (VRRP) to determine which host, from the set of hosts, services which 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 **OPENSOURCE\_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.

## 11.1. IP FAILOVER ENVIRONMENT VARIABLES

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

**Table 11.1. IP failover environment variables**

Variable Name	Default	Description
<b>OPENSHIFT_HA_MONITOR_PORT</b>	<b>80</b>	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 <b>0</b> , the test always passes.
<b>OPENSHIFT_HA_NETWORK_INTERFACE</b>		The interface name that IP failover uses to send Virtual Router Redundancy Protocol (VRRP) traffic. The default value is <b>eth0</b> .
<b>OPENSHIFT_HA_REPLICA_COUNT</b>	<b>2</b>	If your cluster uses the OVN-Kubernetes network plugin, set this value to <b>br-ex</b> 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 <b>br-ex</b> bridge.

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

## 11.2. 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:v4.18
          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: "0"
- 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 ⑯

```

- 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 **OPENSPLIT\_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 **0**, 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.

## 11.3. 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**.

## 11.4. 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 1
        value: preempt_delay 300
...

```

- 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.

## 11.5. 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.

## 11.6. 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 **OPENSIFT\_HA\_VIP\_GROUPS** variable, for example:

#### Example Deployment YAML for IP failover configuration

```
...  
spec:  
  env:  
    - name: OPENSIFT_HA_VIP_GROUPS 1  
      value: "3"  
...
```

- 1 If **OPENSIFT\_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 **OPENSIFT\_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.

## 11.7. 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](#)

## 11.8. 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:v4.18
          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

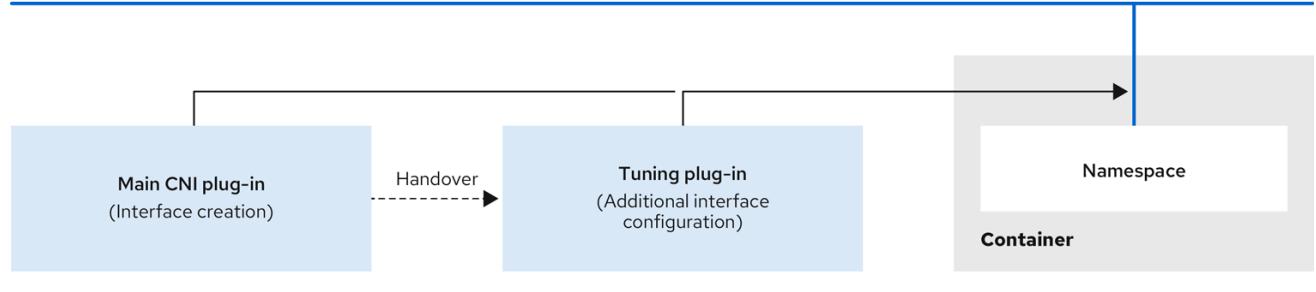
```
remove-failover.sh: OpenShift IP Failover service terminating.
```

- Removing ip\_vs module ...
- Cleaning up ...
- Releasing VIPs (interface eth0) ...

# CHAPTER 12. CONFIGURING SYSTEM CONTROLS AND INTERFACE ATTRIBUTES USING THE TUNING PLUGIN

In Linux, sysctl allows an administrator to modify kernel parameters at runtime. You can modify interface-level network sysctls using the tuning Container Network Interface (CNI) meta plugin. The tuning CNI meta plugin operates in a chain with a main CNI plugin as illustrated.

## Network



264\_OpenShift\_0722

The main CNI plugin assigns the interface and passes this interface to the tuning CNI meta plugin at runtime. You can change some sysctls and several interface attributes such as promiscuous mode, all-multicast mode, MTU, and MAC address in the network namespace by using the tuning CNI meta plugin.

## 12.1. CONFIGURING SYSTEM CONTROLS BY USING THE TUNING CNI

The following procedure configures the tuning CNI to change the interface-level network `net.ipv4.conf.IFNAME.accept_redirects` sysctl. This example enables accepting and sending ICMP-redirected packets. In the tuning CNI meta plugin configuration, the interface name is represented by the **IFNAME** token and is replaced with the actual name of the interface at runtime.

### 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> ①
  namespace: default ②
spec:
  config: '{
    "cniVersion": "0.4.0", ③
    "name": "<name>", ④
    "plugins": [
      {
        "type": "<main_CNI_plugin>" ⑤
      },
      {
        "type": "tuning", ⑥
        "sysctl": {
          "net.ipv4.conf.IFNAME.accept_redirects": "1" ⑦
        }
      }
    ]
  }'
  
```

```

    }
]
}
```

- 1 Specifies the name for the additional network attachment to create. The name must be unique within the specified namespace.
- 2 Specifies the namespace that the object is associated with.
- 3 Specifies the CNI specification version.
- 4 Specifies the name for the configuration. It is recommended to match the configuration name to the name value of the network attachment definition.
- 5 Specifies the name of the main CNI plugin to configure.
- 6 Specifies the name of the CNI meta plugin.
- 7 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.

An example YAML file is shown here:

```

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"}}, {"type": "tun"}]
```

2. Apply the YAML by running the following command:

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

### Example output

```
networkattachmentdefinition.k8s.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 1
spec:
  containers:
    - name: podexample
      image: centos
      command: ["/bin/bash", "-c", "sleep INF"]
      securityContext:
        runAsUser: 2000 2
        runAsGroup: 3000 3
        allowPrivilegeEscalation: false 4
        capabilities: 5
          drop: ["ALL"]
      securityContext:
        runAsNonRoot: true 6
        seccompProfile: 7
        type: RuntimeDefault

```

- 1** Specify the name of the configured **NetworkAttachmentDefinition**.
- 2** **runAsUser** controls which user ID the container is run with.
- 3** **runAsGroup** controls which primary group ID the containers is run with.
- 4** **allowPrivilegeEscalation** determines 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.
- 5** **capabilities** permit privileged actions without giving full root access. This policy ensures all capabilities are dropped from the pod.
- 6** **runAsNonRoot: true** requires that the container will run with a user with any UID other than 0.
- 7** **RuntimeDefault** enables 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
```

## 12.2. ENABLING ALL-MULTICAST MODE BY USING THE TUNING CNI

You can enable all-multicast mode by using the tuning Container Network Interface (CNI) meta plugin.

The following procedure describes how to configure the tuning CNI to enable the all-multicast mode.

### 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> ①
  namespace: default ②
spec:
  config: '{
    "cniVersion": "0.4.0", ③
    "name": "<name>", ④
    "plugins": [
      {
        "type": "<main_CNI_plugin>" ⑤
      },
      {
        "type": "tuning", ⑥
        "allmulti": true ⑦
      }
    ]
  }'
```

- ① Specifies the name for the additional network attachment to create. The name must be unique within the specified namespace.
- ② Specifies the namespace that the object is associated with.
- ③ Specifies the CNI specification version.
- ④ Specifies the name for the configuration. Match the configuration name to the name value of the network attachment definition.

- 5 Specifies the name of the main CNI plugin to configure.
- 6 Specifies the name of the CNI meta plugin.
- 7 Changes the all-multicast mode of interface. If enabled, all multicast packets on the network will be received by the interface.

An example YAML file is shown here:

```
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 ①
spec:
  containers:
    - name: podexample
      image: centos
      command: ["/bin/bash", "-c", "sleep INF"]
      securityContext:
        runAsUser: 2000 ②
```

```

runAsGroup: 3000 3
allowPrivilegeEscalation: false 4
capabilities:
  drop: ["ALL"]
securityContext:
  runAsNonRoot: true 6
  seccompProfile: 7
  type: RuntimeDefault

```

- 1** Specifies the name of the configured **NetworkAttachmentDefinition**.
- 2** Specifies the user ID the container is run with.
- 3** Specifies which primary group ID the containers is run with.
- 4** Specifies if a pod can request privilege escalation. If unspecified, it defaults to **true**. This boolean directly controls whether the **no\_new\_privs** flag gets set on the container process.
- 5** Specifies the container capabilities. The **drop: ["ALL"]** statement indicates that all Linux capabilities are dropped from the pod, providing a more restrictive security profile.
- 6** Specifies that the container will run with a user with any UID other than 0.
- 7** Specifies the container's seccomp profile. In this case, the type is set to **RuntimeDefault**. Seccomp is a Linux kernel feature that restricts the system calls available to a process, enhancing security by minimizing the attack surface.

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 ①
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 ②
```

- ① **eth0@if22** is the primary interface
- ② **net1@if24** is the secondary interface configured with the network-attachment-definition that supports the all-multicast mode (ALLMULTI flag)

## 12.3. 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 13. USING THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

As a cluster administrator, you can use the Stream Control Transmission Protocol (SCTP) on a bare-metal cluster.

## 13.1. SUPPORT FOR SCTP ON OPENSHIFT CONTAINER PLATFORM

As a cluster administrator, you can enable SCTP on the hosts in the cluster. On Red Hat Enterprise Linux CoreOS (RHCOS), the SCTP module is disabled by default.

SCTP is a reliable message based protocol that runs on top of an IP network.

When enabled, you can use SCTP as a protocol with pods, services, and network policy. A **Service** object must be defined with the **type** parameter set to either the **ClusterIP** or **NodePort** value.

### 13.1.1. Example configurations using SCTP protocol

You can configure a pod or service to use SCTP by setting the **protocol** parameter to the **SCTP** value in the pod or service object.

In the following example, a pod is configured to use SCTP:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: project1
  name: example-pod
spec:
  containers:
    - name: example-pod
...
  ports:
    - containerPort: 30100
      name: sctpserver
      protocol: SCTP
```

In the following example, a service is configured to use SCTP:

```
apiVersion: v1
kind: Service
metadata:
  namespace: project1
  name: sctpserver
spec:
...
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30100
      targetPort: 30100
      type: ClusterIP
```

In the following example, a **NetworkPolicy** object is configured to apply to SCTP network traffic on port **80** from any pods with a specific label:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-sctp-on-http
spec:
  podSelector:
    matchLabels:
      role: web
  ingress:
  - ports:
    - protocol: SCTP
      port: 80
```

## 13.2. ENABLING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

As a cluster administrator, you can load and enable the blacklisted SCTP kernel module on worker nodes in your cluster.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

### Procedure

1. Create a file named **load-sctp-module.yaml** that contains the following YAML definition:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: load-sctp-module
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: /etc/modprobe.d/sctp-blacklist.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,
        - path: /etc/modules-load.d/sctp-load.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,sctp
```

- To create the **MachineConfig** object, enter the following command:

```
$ oc create -f load-sctp-module.yaml
```

- Optional: To watch the status of the nodes while the MachineConfig Operator applies the configuration change, enter the following command. When the status of a node transitions to **Ready**, the configuration update is applied.

```
$ oc get nodes
```

### 13.3. VERIFYING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) IS ENABLED

You can verify that SCTP is working on a cluster by creating a pod with an application that listens for SCTP traffic, associating it with a service, and then connecting to the exposed service.

#### Prerequisites

- Access to the internet from the cluster to install the **nc** package.
- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

#### Procedure

- Create a pod starts an SCTP listener:

- Create a file named **sctp-server.yaml** that defines a pod with the following YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpserver
  labels:
    app: sctpserver
spec:
  containers:
    - name: sctpserver
      image: registry.access.redhat.com/ubi9/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
      ports:
        - containerPort: 30102
          name: sctpserver
          protocol: SCTP
```

- Create the pod by entering the following command:

```
$ oc create -f sctp-server.yaml
```

- Create a service for the SCTP listener pod.

- a. Create a file named **sctp-service.yaml** that defines a service with the following YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: sctpservice
  labels:
    app: sctpserver
spec:
  type: NodePort
  selector:
    app: sctpserver
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30102
      targetPort: 30102
```

- b. To create the service, enter the following command:

```
$ oc create -f sctp-service.yaml
```

3. Create a pod for the SCTP client.

- a. Create a file named **sctp-client.yaml** with the following YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpclient
  labels:
    app: sctpclient
spec:
  containers:
    - name: sctpclient
      image: registry.access.redhat.com/ubi9/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
```

- b. To create the **Pod** object, enter the following command:

```
$ oc apply -f sctp-client.yaml
```

4. Run an SCTP listener on the server.

- a. To connect to the server pod, enter the following command:

```
$ oc rsh sctpserver
```

- b. To start the SCTP listener, enter the following command:

```
$ nc -l 30102 --sctp
```

5. Connect to the SCTP listener on the server.
  - a. Open a new terminal window or tab in your terminal program.
  - b. Obtain the IP address of the **sctpservice** service. Enter the following command:

```
$ oc get services sctpservice -o go-template='{{.spec.clusterIP}}{{"\n"}}'
```

- c. To connect to the client pod, enter the following command:

```
$ oc rsh sctpclient
```

- d. To start the SCTP client, enter the following command. Replace <cluster\_IP> with the cluster IP address of the **sctpservice** service.

```
# nc <cluster_IP> 30102 --sctp
```

# CHAPTER 14. USING PTP HARDWARE

## 14.1. ABOUT PRECISION TIME PROTOCOL IN OPENSHIFT CLUSTER NODES

Precision Time Protocol (PTP) is used to synchronize clocks in a network. When used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, and is more accurate than Network Time Protocol (NTP).



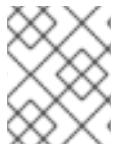
### IMPORTANT

If your **openshift-sdn** cluster with PTP uses the User Datagram Protocol (UDP) for hardware time stamping and you migrate to the OVN-Kubernetes plugin, the hardware time stamping cannot be applied to primary interface devices, such as an Open vSwitch (OVS) bridge. As a result, UDP version 4 configurations cannot work with a **br-ex** interface.

You can configure **linuxptp** services and use PTP-capable hardware in OpenShift Container Platform cluster nodes.

Use the OpenShift Container Platform web console or OpenShift CLI (**oc**) to install PTP by deploying the PTP Operator. The PTP Operator creates and manages the **linuxptp** services and provides the following features:

- Discovery of the PTP-capable devices in the cluster.
- Management of the configuration of **linuxptp** services.
- Notification of PTP clock events that negatively affect the performance and reliability of your application with the PTP Operator **cloud-event-proxy** sidecar.



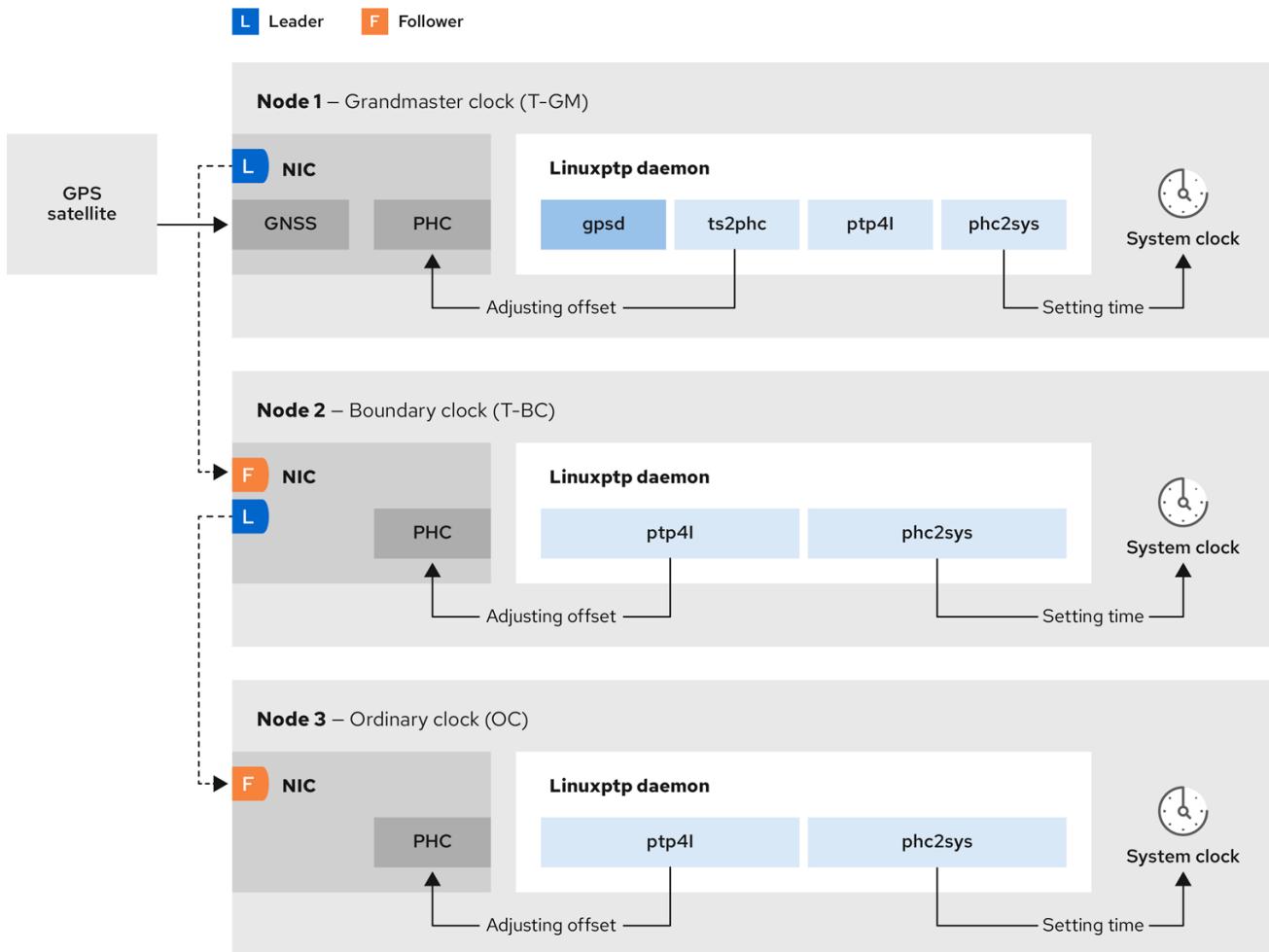
### NOTE

The PTP Operator works with PTP-capable devices on clusters provisioned only on bare-metal infrastructure.

### 14.1.1. Elements of a PTP domain

PTP is used to synchronize multiple nodes connected in a network, with clocks for each node. The clocks synchronized by PTP are organized in a leader-follower hierarchy. The hierarchy is created and updated automatically by the best master clock (BMC) algorithm, which runs on every clock. Follower clocks are synchronized to leader clocks, and follower clocks can themselves be the source for other downstream clocks.

Figure 14.1. PTP nodes in the network



319\_OpenShift\_1123

The three primary types of PTP clocks are described below.

### Grandmaster clock

The grandmaster clock provides standard time information to other clocks across the network and ensures accurate and stable synchronisation. It writes time stamps and responds to time requests from other clocks. Grandmaster clocks synchronize to a Global Navigation Satellite System (GNSS) time source. The Grandmaster clock is the authoritative source of time in the network and is responsible for providing time synchronization to all other devices.

### Boundary clock

The boundary clock has ports in two or more communication paths and can be a source and a destination to other destination clocks at the same time. The boundary clock works as a destination clock upstream. The destination clock receives the timing message, adjusts for delay, and then creates a new source time signal to pass down the network. The boundary clock produces a new timing packet that is still correctly synced with the source clock and can reduce the number of connected devices reporting directly to the source clock.

### Ordinary clock

The ordinary clock has a single port connection that can play the role of source or destination clock, depending on its position in the network. The ordinary clock can read and write timestamps.

### Advantages of PTP over NTP

One of the main advantages that PTP has over NTP is the hardware support present in various network interface controllers (NIC) and network switches. The specialized hardware allows PTP to account for

delays in message transfer and improves the accuracy of time synchronization. To achieve the best possible accuracy, it is recommended that all networking components between PTP clocks are PTP hardware enabled.

Hardware-based PTP provides optimal accuracy, since the NIC can timestamp the PTP packets at the exact moment they are sent and received. Compare this to software-based PTP, which requires additional processing of the PTP packets by the operating system.



## IMPORTANT

Before enabling PTP, ensure that NTP is disabled for the required nodes. You can disable the chrony time service (**chronyd**) using a **MachineConfig** custom resource. For more information, see [Disabling chrony time service](#).

### 14.1.2. Overview of **linuxptp** and **gpsd** in OpenShift Container Platform nodes

OpenShift Container Platform uses the PTP Operator with **linuxptp** and **gpsd** packages for high precision network synchronization. The **linuxptp** package provides tools and daemons for PTP timing in networks. Cluster hosts with Global Navigation Satellite System (GNSS) capable NICs use **gpsd** to interface with GNSS clock sources.

The **linuxptp** package includes the **ts2phc**, **pmc**, **ptp4l**, and **phc2sys** programs for system clock synchronization.

#### **ts2phc**

**ts2phc** synchronizes the PTP hardware clock (PHC) across PTP devices with a high degree of precision. **ts2phc** is used in grandmaster clock configurations. It receives the precision timing signal a high precision clock source such as Global Navigation Satellite System (GNSS). GNSS provides an accurate and reliable source of synchronized time for use in large distributed networks. GNSS clocks typically provide time information with a precision of a few nanoseconds.

The **ts2phc** system daemon sends timing information from the grandmaster clock to other PTP devices in the network by reading time information from the grandmaster clock and converting it to PHC format. PHC time is used by other devices in the network to synchronize their clocks with the grandmaster clock.

#### **pmc**

**pmc** implements a PTP management client (**pmc**) according to IEEE standard 1588.1588. **pmc** provides basic management access for the **ptp4l** system daemon. **pmc** reads from standard input and sends the output over the selected transport, printing any replies it receives.

#### **ptp4l**

**ptp4l** implements the PTP boundary clock and ordinary clock and runs as a system daemon. **ptp4l** does the following:

- Synchronizes the PHC to the source clock with hardware time stamping
- Synchronizes the system clock to the source clock with software time stamping

#### **phc2sys**

**phc2sys** synchronizes the system clock to the PHC on the network interface controller (NIC). The **phc2sys** system daemon continuously monitors the PHC for timing information. When it detects a timing error, the PHC corrects the system clock.

The **gpsd** package includes the **ubxtool**, **gspipe**, **gpsd**, programs for GNSS clock synchronization with the host clock.

## ubxtool

**ubxtool** CLI allows you to communicate with a u-blox GPS system. The **ubxtool** CLI uses the u-blox binary protocol to communicate with the GPS.

## gpsspipe

**gpsspipe** connects to **gpsd** output and pipes it to **stdout**.

## gpsd

**gpsd** is a service daemon that monitors one or more GPS or AIS receivers connected to the host.

### 14.1.3. Overview of GNSS timing for PTP grandmaster clocks

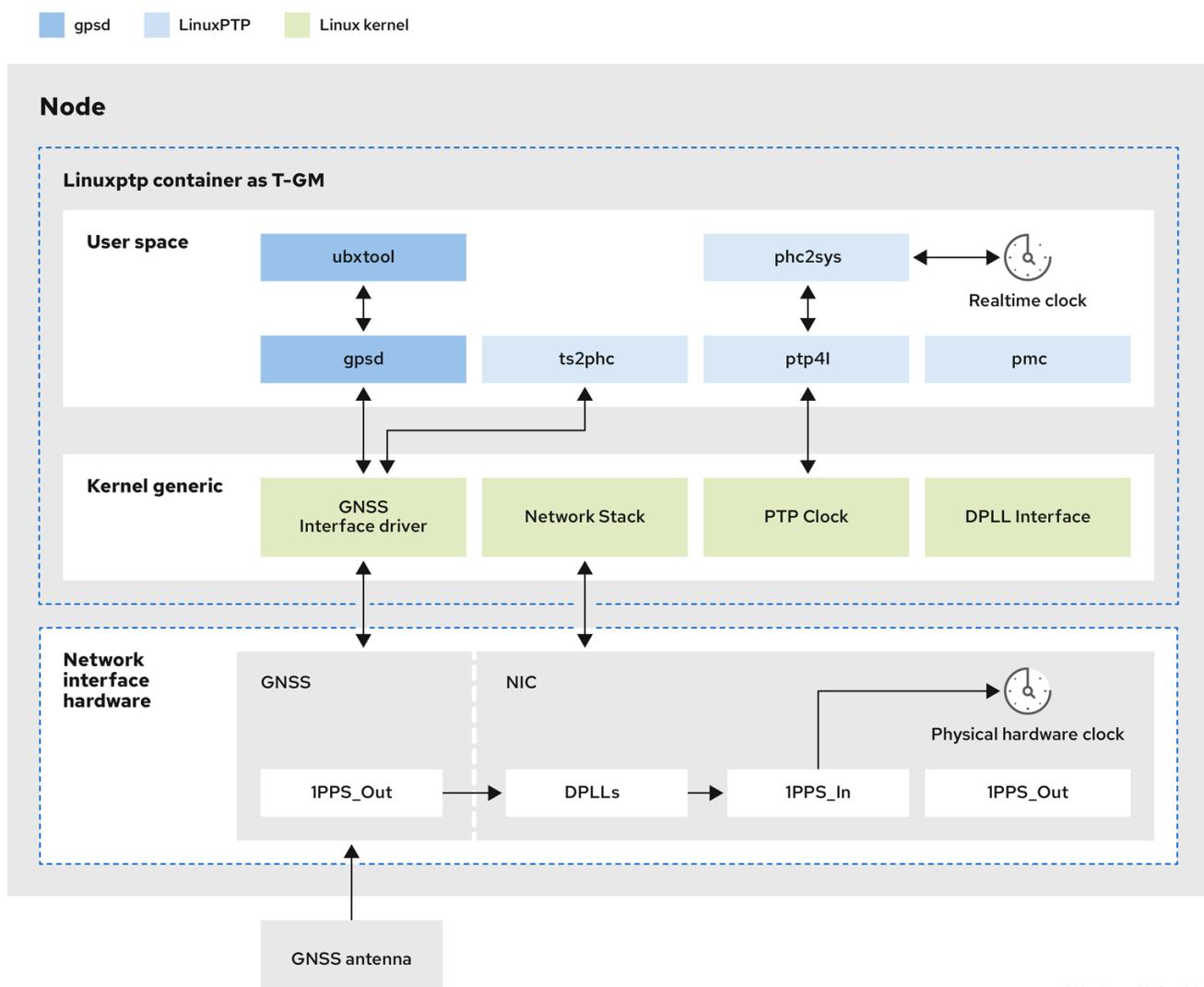
OpenShift Container Platform supports receiving precision PTP timing from Global Navigation Satellite System (GNSS) sources and grandmaster clocks (T-GM) in the cluster.



#### IMPORTANT

OpenShift Container Platform supports PTP timing from GNSS sources with Intel E810 Westport Channel NICs only.

Figure 14.2. Overview of Synchronization with GNSS and T-GM



319\_OpenShift\_1023

## Global Navigation Satellite System (GNSS)

GNSS is a satellite-based system used to provide positioning, navigation, and timing information to receivers around the globe. In PTP, GNSS receivers are often used as a highly accurate and stable reference clock source. These receivers receive signals from multiple GNSS satellites, allowing them to calculate precise time information. The timing information obtained from GNSS is used as a reference by the PTP grandmaster clock.

By using GNSS as a reference, the grandmaster clock in the PTP network can provide highly accurate timestamps to other devices, enabling precise synchronization across the entire network.

### Digital Phase-Locked Loop (DPLL)

DPLL provides clock synchronization between different PTP nodes in the network. DPLL compares the phase of the local system clock signal with the phase of the incoming synchronization signal, for example, PTP messages from the PTP grandmaster clock. The DPLL continuously adjusts the local clock frequency and phase to minimize the phase difference between the local clock and the reference clock.

### Handling leap second events in GNSS-synced PTP grandmaster clocks

A leap second is a one-second adjustment that is occasionally applied to Coordinated Universal Time (UTC) to keep it synchronized with International Atomic Time (TAI). UTC leap seconds are unpredictable. Internationally agreed leap seconds are listed in [leap-seconds.list](#). This file is regularly updated by the International Earth Rotation and Reference Systems Service (IERS). An unhandled leap second can have a significant impact on far edge RAN networks. It can cause the far edge RAN application to immediately disconnect voice calls and data sessions.

#### 14.1.4. About PTP and clock synchronization error events

Cloud native applications such as virtual RAN (vRAN) require access to notifications about hardware timing events that are critical to the functioning of the overall network. PTP clock synchronization errors can negatively affect the performance and reliability of your low-latency application, for example, a vRAN application running in a distributed unit (DU).

Loss of PTP synchronization is a critical error for a RAN network. If synchronization is lost on a node, the radio might be shut down and the network Over the Air (OTA) traffic might be shifted to another node in the wireless network. Fast event notifications mitigate against workload errors by allowing cluster nodes to communicate PTP clock sync status to the vRAN application running in the DU.

Event notifications are available to vRAN applications running on the same DU node. A publish/subscribe REST API passes events notifications to the messaging bus. Publish/subscribe messaging, or pub-sub messaging, is an asynchronous service-to-service communication architecture where any message published to a topic is immediately received by all of the subscribers to the topic.

The PTP Operator generates fast event notifications for every PTP-capable network interface. You can access the events by using a **cloud-event-proxy** sidecar container over an HTTP message bus.



#### NOTE

PTP fast event notifications are available for network interfaces configured to use PTP ordinary clocks, PTP grandmaster clocks, or PTP boundary clocks.

#### 14.1.5. 2-card E810 NIC configuration reference

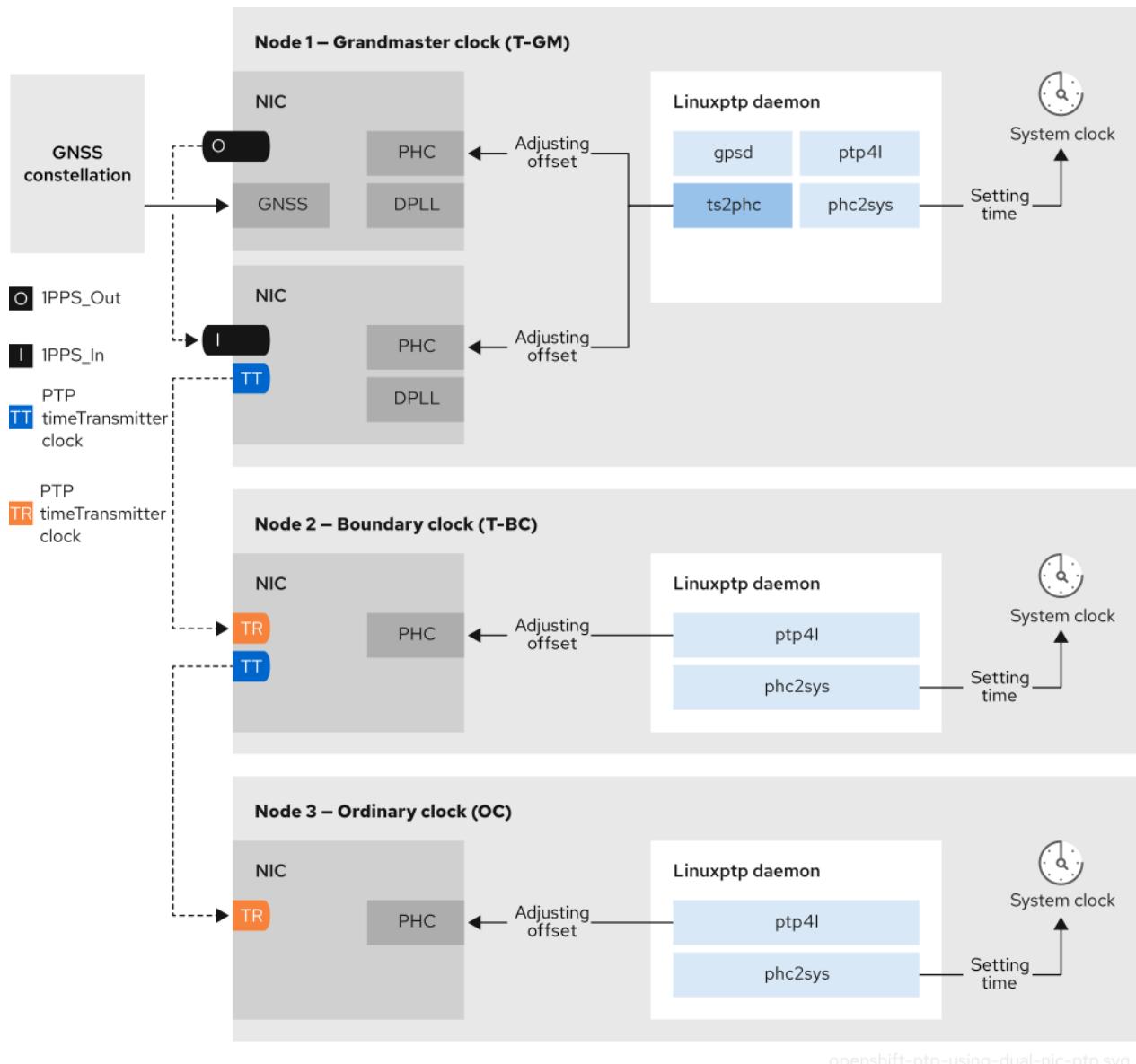
OpenShift Container Platform supports single and dual-NIC Intel E810 hardware for PTP timing in grandmaster clocks (T-GM) and boundary clocks (T-BC).

##### Dual NIC grandmaster clock

You can use a cluster host that has dual-NIC hardware as PTP grandmaster clock. One NIC receives timing information from the global navigation satellite system (GNSS). The second NIC receives the timing information from the first using the SMA1 Tx/Rx connections on the E810 NIC faceplate. The system clock on the cluster host is synchronized from the NIC that is connected to the GNSS satellite.

Dual NIC grandmaster clocks are a feature of distributed RAN (D-RAN) configurations where the Remote Radio Unit (RRU) and Baseband Unit (BBU) are located at the same radio cell site. D-RAN distributes radio functions across multiple sites, with backhaul connections linking them to the core network.

**Figure 14.3. Dual NIC grandmaster clock**



### NOTE

In a dual-NIC T-GM configuration, a single **ts2phc** program operate on two PTP hardware clocks (PHCs), one for each NIC.

### Dual NIC boundary clock

For 5G telco networks that deliver mid-band spectrum coverage, each virtual distributed unit (vDU) requires connections to 6 radio units (RUs). To make these connections, each vDU host requires 2 NICs configured as boundary clocks.

Dual NIC hardware allows you to connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

### Highly available system clock with dual-NIC boundary clocks

You can configure Intel E810-XXVDA4 Salem channel dual-NIC hardware as dual PTP boundary clocks that provide timing for a highly available system clock. This configuration is useful when you have multiple time sources on different NICs. High availability ensures that the node does not lose timing synchronization if one of the two timing sources is lost or disconnected.

Each NIC is connected to the same upstream leader clock. Highly available boundary clocks use multiple PTP domains to synchronize with the target system clock. When a T-BC is highly available, the host system clock can maintain the correct offset even if one or more **ptp4l** instances syncing the NIC PHC clock fails. If any single SFP port or cable failure occurs, the boundary clock stays in sync with the leader clock.

Boundary clock leader source selection is done using the A-BMCA algorithm. For more information, see [ITU-T recommendation G.8275.1](#).

### 14.1.6. 3-card Intel E810 PTP grandmaster clock

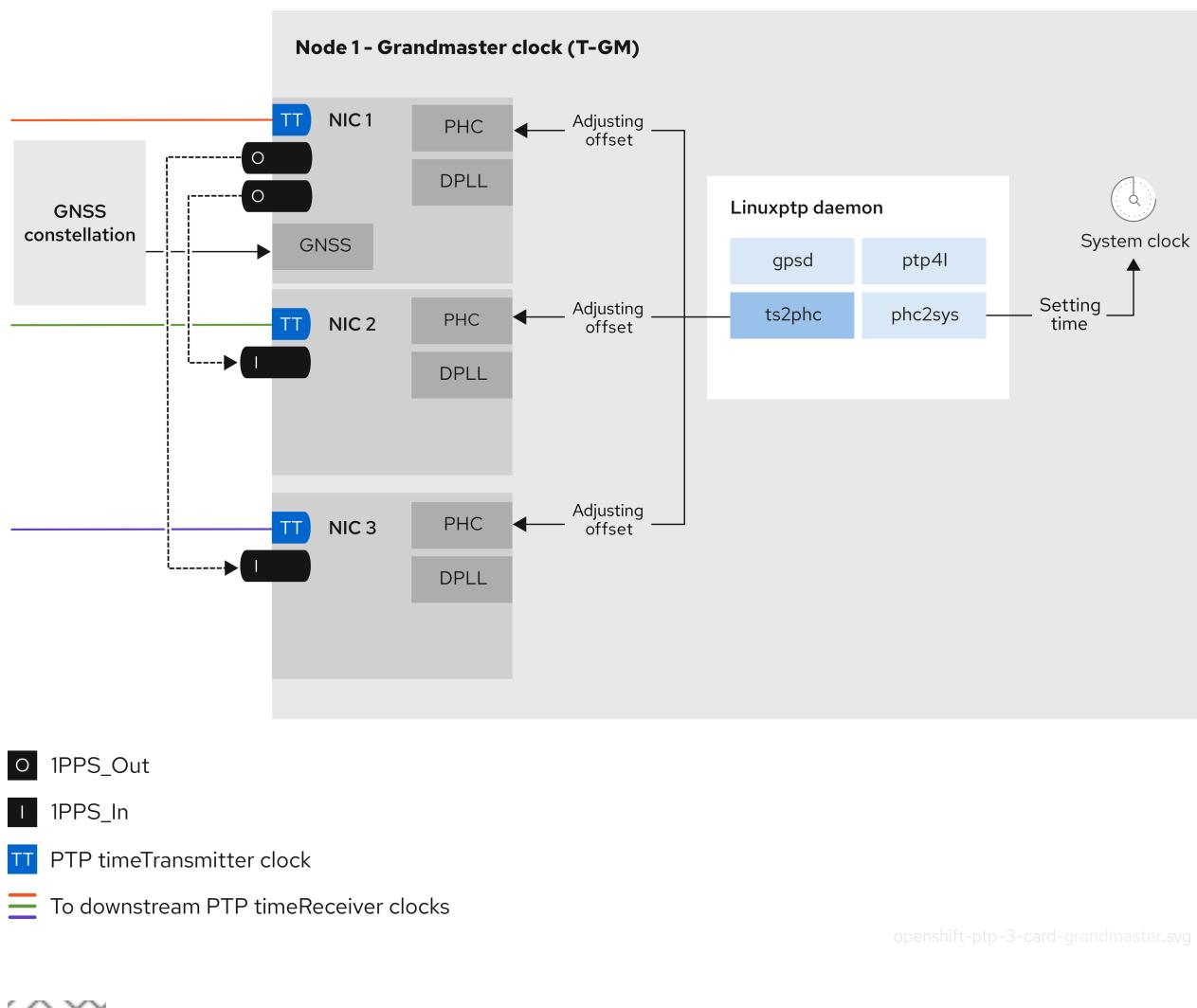
OpenShift Container Platform supports cluster hosts with 3 Intel E810 NICs as PTP grandmaster clocks (T-GM).

#### 3-card grandmaster clock

You can use a cluster host that has 3 NICs as PTP grandmaster clock. One NIC receives timing information from the global navigation satellite system (GNSS). The second and third NICs receive the timing information from the first by using the SMA1 Tx/Rx connections on the E810 NIC faceplate. The system clock on the cluster host is synchronized from the NIC that is connected to the GNSS satellite.

3-card NIC grandmaster clocks can be used for distributed RAN (D-RAN) configurations where the Radio Unit (RU) is connected directly to the distributed unit (DU) without a front haul switch, for example, if the RU and DU are located in the same radio cell site. D-RAN distributes radio functions across multiple sites, with backhaul connections linking them to the core network.

Figure 14.4. 3-card Intel E810 PTP grandmaster clock



### NOTE

In a 3-card T-GM configuration, a single **ts2phc** process reports as 3 **ts2phc** instances in the system.

## 14.2. CONFIGURING PTP DEVICES

The PTP Operator adds the **NodePtpDevice.ptp.openshift.io** custom resource definition (CRD) to OpenShift Container Platform.

When installed, the PTP Operator searches your cluster for Precision Time Protocol (PTP) capable network devices on each node. The Operator creates and updates a **NodePtpDevice** custom resource (CR) object for each node that provides a compatible PTP-capable network device.

Network interface controller (NIC) hardware with built-in PTP capabilities sometimes require a device-specific configuration. You can use hardware-specific NIC features for supported hardware with the PTP Operator by configuring a plugin in the **PtpConfig** custom resource (CR). The **linuxptp-daemon** service uses the named parameters in the **plugin** stanza to start **linuxptp** processes, **ptp4l** and **phc2sys**, based on the specific hardware configuration.



## IMPORTANT

In OpenShift Container Platform 4.18, the Intel E810 NIC is supported with a **PtpConfig** plugin.

### 14.2.1. Installing the PTP Operator using the CLI

As a cluster administrator, you can install the Operator by using the CLI.

#### Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports PTP.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create a namespace for the PTP Operator.
  - a. Save the following YAML in the **ptp-namespace.yaml** file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    name: openshift-ptp
    openshift.io/cluster-monitoring: "true"
```

- b. Create the **Namespace** CR:

```
$ oc create -f ptp-namespace.yaml
```

2. Create an Operator group for the PTP Operator.

- a. Save the following YAML in the **ptp-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
    - openshift-ptp
```

- b. Create the **OperatorGroup** CR:

```
$ oc create -f ptp-operatorgroup.yaml
```

3. Subscribe to the PTP Operator.
  - a. Save the following YAML in the **ptp-sub.yaml** file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

- b. Create the **Subscription** CR:

```
$ oc create -f ptp-sub.yaml
```

4. To verify that the Operator is installed, enter the following command:

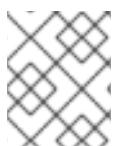
```
$ oc get csv -n openshift-ptp -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

#### Example output

Name	Phase
4.18.0-202301261535	Succeeded

### 14.2.2. Installing the PTP Operator by using the web console

As a cluster administrator, you can install the PTP Operator by using the web console.



#### NOTE

You have to create the namespace and Operator group as mentioned in the previous section.

#### Procedure

1. Install the PTP Operator using the OpenShift Container Platform web console:
  - a. In the OpenShift Container Platform web console, click **Operators** → **OperatorHub**.
  - b. Choose **PTP Operator** from the list of available Operators, and then click **Install**.
  - c. On the **Install Operator** page, under **A specific namespace on the cluster** select **openshift-ptp**. Then, click **Install**.
2. Optional: Verify that the PTP Operator installed successfully:
  - a. Switch to the **Operators** → **Installed Operators** page.

- b. Ensure that **PTP Operator** is listed in the `openshift-ptp` project with a **Status** of `InstallSucceeded`.



#### NOTE

During installation an Operator might display a **Failed** status. If the installation later succeeds with an `InstallSucceeded` message, you can ignore the **Failed** message.

If the Operator does not appear as installed, to troubleshoot further:

- Go to the **Operators → Installed Operators** page and inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.
- Go to the **Workloads → Pods** page and check the logs for pods in the `openshift-ptp` project.

### 14.2.3. Discovering PTP-capable network devices in your cluster

Identify PTP-capable network devices that exist in your cluster so that you can configure them

#### Prerequisites

- You installed the PTP Operator.

#### Procedure

- To return a complete list of PTP capable network devices in your cluster, run the following command:

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

#### Example output

```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    creationTimestamp: "2022-01-27T15:16:28Z"
    generation: 1
    name: dev-worker-0 1
    namespace: openshift-ptp
    resourceVersion: "6538103"
    uid: d42fc9ad-bcbf-4590-b6d8-b676c642781a
  spec: {}
  status:
    devices: 2
    - name: eno1
    - name: eno2
    - name: eno3
    - name: eno4
```

```

- name: enp5s0f0
- name: enp5s0f1
...

```

- 1 The value for the **name** parameter is the same as the name of the parent node.
- 2 The **devices** collection includes a list of the PTP capable devices that the PTP Operator discovers for the node.

#### 14.2.4. Configuring `linuxptp` services as a grandmaster clock

You can configure the `linuxptp` services (`ptp4l`, `phc2sys`, `ts2phc`) as grandmaster clock (T-GM) by creating a **PtpConfig** custom resource (CR) that configures the host NIC.

The **ts2phc** utility allows you to synchronize the system clock with the PTP grandmaster clock so that the node can stream precision clock signal to downstream PTP ordinary clocks and boundary clocks.



#### NOTE

Use the following example **PtpConfig** CR as the basis to configure `linuxptp` services as T-GM for an Intel Westport Channel E810-XXVDA4T network interface.

To configure PTP fast events, set appropriate values for `ptp4IOpts`, `ptp4IConf`, and `ptpClockThreshold`. `ptpClockThreshold` is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

#### Prerequisites

- For T-GM clocks in production environments, install an Intel E810 Westport Channel NIC in the bare-metal cluster host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

#### Procedure

1. Create the **PtpConfig** CR. For example:
  - a. Depending on your requirements, use one of the following T-GM configurations for your deployment. Save the YAML in the `grandmaster-clock-ptp-config.yaml` file:

##### Example 14.1. PTP grandmaster clock configuration for E810 NIC

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"

```

```
ptp4lOpts: "-2 --summary_interval -4"
phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s $iface_master -n 24
ptpSchedulingPolicy: SCHED_FIFO
ptpSchedulingPriority: 10
ptpSettings:
  logReduce: "true"
plugins:
  e810:
    enableDefaultConfig: false
    settings:
      LocalMaxHoldoverOffSet: 1500
      LocalHoldoverTimeout: 14400
      MaxInSpecOffset: 1500
    pins: $e810_pins
    # "$iface_master":
    #  "U.FL2": "0 2"
    #  "U.FL1": "0 1"
    #  "SMA2": "0 2"
    #  "SMA1": "0 1"
    ublxCmds:
      - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
        - "-P"
        - "29.20"
        - "-z"
        - "CFG-HW-ANT_CFG_VOLTCTRL,1"
      reportOutput: false
      - args: #ubxtool -P 29.20 -e GPS
        - "-P"
        - "29.20"
        - "-e"
        - "GPS"
      reportOutput: false
      - args: #ubxtool -P 29.20 -d Galileo
        - "-P"
        - "29.20"
        - "-d"
        - "Galileo"
      reportOutput: false
      - args: #ubxtool -P 29.20 -d GLONASS
        - "-P"
        - "29.20"
        - "-d"
        - "GLONASS"
      reportOutput: false
      - args: #ubxtool -P 29.20 -d BeiDou
        - "-P"
        - "29.20"
        - "-d"
        - "BeiDou"
      reportOutput: false
      - args: #ubxtool -P 29.20 -d SBAS
        - "-P"
        - "29.20"
        - "-d"
        - "SBAS"
      reportOutput: false
```

```
- args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
  - "-P"
  - "29.20"
  - "-t"
  - "-w"
  - "5"
  - "-v"
  - "1"
  - "-e"
  - "SURVEYIN,600,50000"
reportOutput: true
- args: #ubxtool -P 29.20 -p MON-HW
  - "-P"
  - "29.20"
  - "-p"
  - "MON-HW"
reportOutput: true
- args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
  - "-P"
  - "29.20"
  - "-p"
  - "CFG-MSG,1,38,248"
  reportOutput: true
ts2phcOpts: " "
ts2phcConf: |
[nmea]
ts2phc.master 1
[global]
use_syslog 0
verbose 1
logging_level 7
ts2phc.pulsewidth 100000000
#cat /dev/GNSS to find available serial port
#example value of gnss_serialport is /dev/ttyGNSS_1700_0
ts2phc.nmea_serialport $gnss_serialport
[$iface_master]
ts2phc.extts_polarity rising
ts2phc.extts_correction 0
ptp4lConf: |
[$iface_master]
masterOnly 1
[$iface_master_1]
masterOnly 1
[$iface_master_2]
masterOnly 1
[$iface_master_3]
masterOnly 1
[global]
#
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
```

```
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
```

```

first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0x20
recommend:
- profile: "grandmaster"
  priority: 4
  match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"

```



### NOTE

For E810 Westport Channel NICs, set the value for **ts2phc.nmea\_serialport** to **/dev/gnss0**.

- Create the CR by running the following command:

```
$ oc create -f grandmaster-clock-ptp-config.yaml
```

### Verification

- Check that the **PtpConfig** profile is applied to the node.

- a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-74m2g	3/3	Running	3	4d15h	10.16.230.7	compute-1.example.com
ptp-operator-5f4f48d7c-x7zkf	1/1	Running	1	4d15h	10.128.1.145	compute-1.example.com

- b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

```
$ oc logs linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
```

#### Example output

```
ts2phc[94980.334]: [ts2phc.0.config] nmea delay: 98690975 ns
ts2phc[94980.334]: [ts2phc.0.config] ens3f0 extts index 0 at 1676577329.999999999 corr
0 src 1676577330.901342528 diff -1
ts2phc[94980.334]: [ts2phc.0.config] ens3f0 master offset      -1 s2 freq     -1
ts2phc[94980.441]: [ts2phc.0.config] nmea sentence:
GNRMC,195453.00,A,4233.24427,N,07126.64420,W,0.008,,160223,,,A,V
phc2sys[94980.450]: [ptp4l.0.config] CLOCK_REALTIME phc offset    943 s2 freq -
89604 delay   504
phc2sys[94980.512]: [ptp4l.0.config] CLOCK_REALTIME phc offset   1000 s2 freq -
89264 delay   474
```

#### 14.2.4.1. Configuring linuxptp services as a grandmaster clock for dual E810 NICs

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock (T-GM) for 2 E810 NICs by creating a **PtpConfig** custom resource (CR) that configures the NICs.

You can configure the **linuxptp** services as a T-GM for the following E810 NICs:

- Intel E810-XXVDA4T Westport Channel NIC
- Intel E810-CQDA2T Logan Beach NIC

For distributed RAN (D-RAN) use cases, you can configure PTP for 2 NICs as follows:

- NIC 1 is synced to the global navigation satellite system (GNSS) time source.
- NIC 2 is synced to the 1PPS timing output provided by NIC one. This configuration is provided by the PTP hardware plugin in the **PtpConfig** CR.

The 2-card PTP T-GM configuration uses one instance of **ptp4l** and one instance of **ts2phc**. The **ptp4l** and **ts2phc** programs are each configured to operate on two PTP hardware clocks (PHCs), one for each NIC. The host system clock is synchronized from the NIC that is connected to the GNSS time source.



## NOTE

Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as T-GM for dual Intel E810 network interfaces.

To configure PTP fast events, set appropriate values for **ptp4IOpts**, **ptp4IConf**, and **ptpClockThreshold**. **ptpClockThreshold** is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

## Prerequisites

- For T-GM clocks in production environments, install two Intel E810 NICs in the bare-metal cluster host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

## Procedure

1. Create the **PtpConfig** CR. For example:

- a. Save the following YAML in the **grandmaster-clock-ptp-config-dual-nics.yaml** file:

### Example 14.2. PTP grandmaster clock configuration for dual E810 NICs

```
# In this example two cards $iface_nic1 and $iface_nic2 are connected via
# SMA1 ports by a cable and $iface_nic2 receives 1PPS signals from $iface_nic1
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"
      ptp4IOpts: "-2 --summary_interval -4"
      phc2sysOpts: -r -u 0 -m N 8 -R 16 -s $iface_nic1 -n 24
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      plugins:
        e810:
          enableDefaultConfig: false
          settings:
            LocalMaxHoldoverOffset: 1500
            LocalHoldoverTimeout: 14400
            MaxInSpecOffset: 1500
            pins: $e810_pins
            # "$iface_nic1":
            #   "U.FL2": "0 2"
            #   "U.FL1": "0 1"
            #   "SMA2": "0 2"
```

```
# "SMA1": "2 1"
# "$iface_nic2":
# "U.FL2": "0 2"
# "U.FL1": "0 1"
# "SMA2": "0 2"
# "SMA1": "1 1"
ublxCmds:
- args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
  - "-P"
  - "29.20"
  - "-z"
  - "CFG-HW-ANT_CFG_VOLTCTRL,1"
  reportOutput: false
- args: #ubxtool -P 29.20 -e GPS
  - "-P"
  - "29.20"
  - "-e"
  - "GPS"
  reportOutput: false
- args: #ubxtool -P 29.20 -d Galileo
  - "-P"
  - "29.20"
  - "-d"
  - "Galileo"
  reportOutput: false
- args: #ubxtool -P 29.20 -d GLONASS
  - "-P"
  - "29.20"
  - "-d"
  - "GLONASS"
  reportOutput: false
- args: #ubxtool -P 29.20 -d BeiDou
  - "-P"
  - "29.20"
  - "-d"
  - "BeiDou"
  reportOutput: false
- args: #ubxtool -P 29.20 -d SBAS
  - "-P"
  - "29.20"
  - "-d"
  - "SBAS"
  reportOutput: false
- args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
  - "-P"
  - "29.20"
  - "-t"
  - "-w"
  - "5"
  - "-v"
  - "1"
  - "-e"
  - "SURVEYIN,600,50000"
  reportOutput: true
- args: #ubxtool -P 29.20 -p MON-HW
  - "-P"
```

```
- "29.20"
- "-p"
- "MON-HW"
reportOutput: true
- args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
- "-P"
- "29.20"
- "-p"
- "CFG-MSG,1,38,248"
reportOutput: true
ts2phcOpts: " "
ts2phcConf: |
[nmea]
ts2phc.master 1
[global]
use_syslog 0
verbose 1
logging_level 7
ts2phc.pulsewidth 100000000
#cat /dev/GNSS to find available serial port
#example value of gnss_serialport is /dev/ttyGNSS_1700_0
ts2phc.nmea_serialport $gnss_serialport
[$iface_nic1]
ts2phc.extts_polarity rising
ts2phc.extts_correction 0
[$iface_nic2]
ts2phc.master 0
ts2phc.extts_polarity rising
#this is a measured value in nanoseconds to compensate for SMA cable delay
ts2phc.extts_correction -10
ptp4lConf: |
[$iface_nic1]
masterOnly 1
[$iface_nic1_1]
masterOnly 1
[$iface_nic1_2]
masterOnly 1
[$iface_nic1_3]
masterOnly 1
[$iface_nic2]
masterOnly 1
[$iface_nic2_1]
masterOnly 1
[$iface_nic2_2]
masterOnly 1
[$iface_nic2_3]
masterOnly 1
[global]
#
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
```

```
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
```

```

first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 1
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0x20
recommend:
- profile: "grandmaster"
  priority: 4
  match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"

```



#### NOTE

Set the value for **ts2phc.nmea\_serialport** to **/dev/gnss0**.

- Create the CR by running the following command:

```
$ oc create -f grandmaster-clock-ptp-config-dual-nics.yaml
```

#### Verification

- Check that the **PtpConfig** profile is applied to the node.
  - Get the list of pods in the **openshift-ptp** namespace by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-74m2g	3/3	Running	3	4d15h	10.16.230.7	compute-1.example.com
ptp-operator-5f4f48d7c-x7zkf	1/1	Running	1	4d15h	10.128.1.145	compute-1.example.com

- b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

```
$ oc logs linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
```

### Example output

```
ts2phc[509863.660]: [ts2phc.0.config] nmea delay: 347527248 ns
ts2phc[509863.660]: [ts2phc.0.config] ens2f0 extts index 0 at 1705516553.000000000
corr 0 src 1705516553.652499081 diff 0
ts2phc[509863.660]: [ts2phc.0.config] ens2f0 master offset      0 s2 freq   -0
I0117 18:35:16.000146 1633226 stats.go:57] state updated for ts2phc =s2
I0117 18:35:16.000163 1633226 event.go:417] dpll State s2, gnss State s2, tspc state
s2, gm state s2,
ts2phc[1705516516]:[ts2phc.0.config] ens2f0 nmea_status 1 offset 0 s2
GM[1705516516]:[ts2phc.0.config] ens2f0 T-GM-STATUS s2
ts2phc[509863.677]: [ts2phc.0.config] ens7f0 extts index 0 at 1705516553.000000010
corr -10 src 1705516553.652499081 diff 0
ts2phc[509863.677]: [ts2phc.0.config] ens7f0 master offset      0 s2 freq   -0
I0117 18:35:16.016597 1633226 stats.go:57] state updated for ts2phc =s2
phc2sys[509863.719]: [ptp4l.0.config] CLOCK_REALTIME phc offset     -6 s2 freq
+15441 delay  510
phc2sys[509863.782]: [ptp4l.0.config] CLOCK_REALTIME phc offset     -7 s2 freq
+15438 delay  502
```

#### 14.2.4.2. Configuring linuxptp services as a grandmaster clock for 3 E810 NICs

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock (T-GM) for 3 E810 NICs by creating a **PtpConfig** custom resource (CR) that configures the NICs.

You can configure the **linuxptp** services as a T-GM with 3 NICs for the following E810 NICs:

- Intel E810-XXVDA4T Westport Channel NIC
- Intel E810-CQDA2T Logan Beach NIC

For distributed RAN (D-RAN) use cases, you can configure PTP for 3 NICs as follows:

- NIC 1 is synced to the Global Navigation Satellite System (GNSS)
- NICs 2 and 3 are synced to NIC 1 with 1PPS faceplate connections

Use the following example **PtpConfig** CRs as the basis to configure `linuxptp` services as a 3-card Intel E810 T-GM.

## Prerequisites

- For T-GM clocks in production environments, install 3 Intel E810 NICs in the bare-metal cluster host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

## Procedure

1. Create the **PtpConfig** CR. For example:

- a. Save the following YAML in the `three-nic-grandmaster-clock-ptp-config.yaml` file:

### Example 14.3. PTP grandmaster clock configuration for 3 E810 NICs

```
# In this example, the three cards are connected via SMA cables:
# - $iface_timeTx1 has the GNSS signal input
# - $iface_timeTx2 SMA1 is connected to $iface_timeTx1 SMA1
# - $iface_timeTx3 SMA1 is connected to $iface_timeTx1 SMA2
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: gm-3card
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
    - name: grandmaster
      ptp4lOpts: -2 --summary_interval -4
      phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s $iface_timeTx1 -n 24
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      plugins:
        e810:
          enableDefaultConfig: false
          settings:
            LocalHoldoverTimeout: 14400
            LocalMaxHoldoverOffSet: 1500
            MaxInSpecOffset: 1500
          pins:
            # Syntax guide:
            # - The 1st number in each pair must be one of:
            #   0 - Disabled
            #   1 - RX
            #   2 - TX
            # - The 2nd number in each pair must match the channel number
            $iface_timeTx1:
```

```
SMA1: 2 1
SMA2: 2 2
U.FL1: 0 1
U.FL2: 0 2
$iface_timeTx2:
SMA1: 1 1
SMA2: 0 2
U.FL1: 0 1
U.FL2: 0 2
$iface_timeTx3:
SMA1: 1 1
SMA2: 0 2
U.FL1: 0 1
U.FL2: 0 2
ublxCmds:
- args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
  - "-P"
  - "29.20"
  - "-z"
  - "CFG-HW-ANT_CFG_VOLTCTRL,1"
  reportOutput: false
- args: #ubxtool -P 29.20 -e GPS
  - "-P"
  - "29.20"
  - "-e"
  - "GPS"
  reportOutput: false
- args: #ubxtool -P 29.20 -d Galileo
  - "-P"
  - "29.20"
  - "-d"
  - "Galileo"
  reportOutput: false
- args: #ubxtool -P 29.20 -d GLONASS
  - "-P"
  - "29.20"
  - "-d"
  - "GLONASS"
  reportOutput: false
- args: #ubxtool -P 29.20 -d BeiDou
  - "-P"
  - "29.20"
  - "-d"
  - "BeiDou"
  reportOutput: false
- args: #ubxtool -P 29.20 -d SBAS
  - "-P"
  - "29.20"
  - "-d"
  - "SBAS"
  reportOutput: false
- args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
  - "-P"
  - "29.20"
  - "-t"
  - "-w"
```

```

    - "5"
    - "-v"
    - "1"
    - "-e"
    - "SURVEYIN,600,50000"
reportOutput: true
- args: #ubxtool -P 29.20 -p MON-HW
    - "-P"
    - "29.20"
    - "-p"
    - "MON-HW"
reportOutput: true
- args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
    - "-P"
    - "29.20"
    - "-p"
    - "CFG-MSG,1,38,248"
reportOutput: true
ts2phcOpts: " "
ts2phcConf: |
[nmea]
ts2phc.master 1
[global]
use_syslog 0
verbose 1
logging_level 7
ts2phc.pulsewidth 100000000
#example value of nmea_serialport is /dev/gnss0
ts2phc.nmea_serialport (?<gnss_serialport>[/\w\s/]+)
leapfile /usr/share/zoneinfo/leap-seconds.list
[$iface_timeTx1]
ts2phc.extts_polarity rising
ts2phc.extts_correction 0
[$iface_timeTx2]
ts2phc.master 0
ts2phc.extts_polarity rising
#this is a measured value in nanoseconds to compensate for SMA cable delay
ts2phc.extts_correction -10
[$iface_timeTx3]
ts2phc.master 0
ts2phc.extts_polarity rising
#this is a measured value in nanoseconds to compensate for SMA cable delay
ts2phc.extts_correction -10
ptp4lConf: |
[$iface_timeTx1]
masterOnly 1
[$iface_timeTx1_1]
masterOnly 1
[$iface_timeTx1_2]
masterOnly 1
[$iface_timeTx1_3]
masterOnly 1
[$iface_timeTx2]
masterOnly 1
[$iface_timeTx2_1]
masterOnly 1

```

```
[$iface_timeTx2_2]
masterOnly 1
[$iface_timeTx2_3]
masterOnly 1
[$iface_timeTx3]
masterOnly 1
[$iface_timeTx3_1]
masterOnly 1
[$iface_timeTx3_2]
masterOnly 1
[$iface_timeTx3_3]
masterOnly 1
[global]
#
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
```

```
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 1
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0x20
ptpClockThreshold:
```

```

holdOverTimeout: 5
maxOffsetThreshold: 1500
minOffsetThreshold: -1500
recommend:
- profile: grandmaster
  priority: 4
  match:
    - nodeLabel: node-role.kubernetes.io/$mcp

```

**NOTE**

Set the value for **ts2phc.nmea\_serialport** to **/dev/gnss0**.

- Create the CR by running the following command:

```
$ oc create -f three-nic-grandmaster-clock-ptp-config.yaml
```

**Verification**

- Check that the **PtpConfig** profile is applied to the node.

- Get the list of pods in the **openshift-ptp** namespace by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

**Example output**

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-74m3q	3/3	Running	3	4d15h	10.16.230.7	compute-1.example.com
ptp-operator-5f4f48d7c-x6zkn	1/1	Running	1	4d15h	10.128.1.145	compute-1.example.com

- Check that the profile is correct. Run the following command, and examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile:

```
$ oc logs linuxptp-daemon-74m3q -n openshift-ptp -c linuxptp-daemon-container
```

**Example output**

```

ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
/dev/ptp11 ①
ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
/dev/ptp7 ②
ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
/dev/ptp14 ③
ts2phc[2527.586]: [ts2phc.0.config:7] nmea delay: 56308811 ns
ts2phc[2527.586]: [ts2phc.0.config:6] /dev/ptp14 offset      0 s2 freq   +0 ④
ts2phc[2527.587]: [ts2phc.0.config:6] /dev/ptp7 offset      0 s2 freq   +0 ⑤
ts2phc[2527.587]: [ts2phc.0.config:6] /dev/ptp11 offset      0 s2 freq   -0 ⑥
I0324 14:25:05.000439 106907 stats.go:61] state updated for ts2phc =s2

```

```
I0324 14:25:05.000504 106907 event.go:419] dpll State s2, gnss State s2, tsphc state
s2, gm state s2,
I0324 14:25:05.000906 106907 stats.go:61] state updated for ts2phc =s2
I0324 14:25:05.001059 106907 stats.go:61] state updated for ts2phc =s2
ts2phc[1742826305]:[ts2phc.0.config] ens4f0 nmea_status 1 offset 0 s2
GM[1742826305]:[ts2phc.0.config] ens4f0 T-GM-STATUS s2 7
```

1 2 3 **ts2phc** is updating the PTP hardware clock.

4 5 6 Estimated PTP device offset between PTP device and the reference clock is 0 nanoseconds. The PTP device is in sync with the leader clock.

7 T-GM is in a locked state (s2).

## Additional resources

- Configuring the PTP fast event notifications publisher

### 14.2.5. Grandmaster clock PtpConfig configuration reference

The following reference information describes the configuration options for the **PtpConfig** custom resource (CR) that configures the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock.

**Table 14.1. PtpConfig configuration options for PTP Grandmaster clock**

PtpConfig CR field	Description
<b>plugins</b>	<p>Specify an array of <b>.exec.cmdline</b> options that configure the NIC for grandmaster clock operation. Grandmaster clock configuration requires certain PTP pins to be disabled.</p> <p>The plugin mechanism allows the PTP Operator to do automated hardware configuration. For the Intel Westport Channel NIC or the Intel Logan Beach NIC, when the <b>enableDefaultConfig</b> field is set to <b>true</b>, the PTP Operator runs a hard-coded script to do the required configuration for the NIC.</p>
<b>ptp4IOpts</b>	<p>Specify system configuration options for the <b>ptp4l</b> service. The options should not include the network interface name <b>-i &lt;interface&gt;</b> and service config file <b>-f /etc/ptp4l.conf</b> because the network interface name and the service config file are automatically appended.</p>
<b>ptp4lConf</b>	<p>Specify the required configuration to start <b>ptp4l</b> as a grandmaster clock. For example, the <b>ens2f1</b> interface synchronizes downstream connected devices. For grandmaster clocks, set <b>clockClass</b> to <b>6</b> and set <b>clockAccuracy</b> to <b>0x27</b>. Set <b>timeSource</b> to <b>0x20</b> for when receiving the timing signal from a Global navigation satellite system (GNSS).</p>
<b>tx_timestamp_timeout</b>	<p>Specify the maximum amount of time to wait for the transmit (TX) timestamp from the sender before discarding the data.</p>
<b>boundary_clock_jbod</b>	<p>Specify the JBOD boundary clock time delay value. This value is used to correct the time values that are passed between the network time devices.</p>

PtpConfig CR field	Description
<b>phc2sysOpts</b>	<p>Specify system config options for the <b>phc2sys</b> service. If this field is empty the PTP Operator does not start the <b>phc2sys</b> service.</p>  <p><b>NOTE</b></p> <p>Ensure that the network interface listed here is configured as grandmaster and is referenced as required in the <b>ts2phcConf</b> and <b>ptp4lConf</b> fields.</p>
<b>ptpSchedulingPolicy</b>	Configure the scheduling policy for <b>ptp4l</b> and <b>phc2sys</b> processes. Default value is <b>SCHED_OTHER</b> . Use <b>SCHED_FIFO</b> on systems that support FIFO scheduling.
<b>ptpSchedulingPriority</b>	Set an integer value from 1-65 to configure FIFO priority for <b>ptp4l</b> and <b>phc2sys</b> processes when <b>ptpSchedulingPolicy</b> is set to <b>SCHED_FIFO</b> . The <b>ptpSchedulingPriority</b> field is not used when <b>ptpSchedulingPolicy</b> is set to <b>SCHED_OTHER</b> .
<b>ptpClockThreshold</b>	Optional. If <b>ptpClockThreshold</b> stanza is not present, default values are used for <b>ptpClockThreshold</b> fields. Stanza shows default <b>ptpClockThreshold</b> values. <b>ptpClockThreshold</b> values configure how long after the PTP master clock is disconnected before PTP events are triggered. <b>holdOverTimeout</b> is the time value in seconds before the PTP clock event state changes to <b>FREERUN</b> when the PTP master clock is disconnected. The <b>maxOffsetThreshold</b> and <b>minOffsetThreshold</b> settings configure offset values in nanoseconds that compare against the values for <b>CLOCK_REALTIME (phc2sys)</b> or master offset ( <b>ptp4l</b> ). When the <b>ptp4l</b> or <b>phc2sys</b> offset value is outside this range, the PTP clock state is set to <b>FREERUN</b> . When the offset value is within this range, the PTP clock state is set to <b>LOCKED</b> .
<b>ts2phcConf</b>	<p>Sets the configuration for the <b>ts2phc</b> command.</p> <p><b>leapfile</b> is the default path to the current leap seconds definition file in the PTP Operator container image.</p> <p><b>ts2phc.nmea_serialport</b> is the serial port device that is connected to the NMEA GPS clock source. When configured, the GNSS receiver is accessible on <b>/dev/gnss&lt;id&gt;</b>. If the host has multiple GNSS receivers, you can find the correct device by enumerating either of the following devices:</p> <ul style="list-style-type: none"> <li>● <b>/sys/class/net/&lt;eth_port&gt;/device/gnss/</b></li> <li>● <b>/sys/class/gnss/gnss&lt;id&gt;/device/</b></li> </ul>
<b>ts2phcOpts</b>	Set options for the <b>ts2phc</b> command.
<b>recommend</b>	Specify an array of one or more <b>recommend</b> objects that define rules on how the <b>profile</b> should be applied to nodes.

PtpConfig CR field	Description
<b>.recommend.profile</b>	Specify the <b>.recommend.profile</b> object name that is defined in the <b>profile</b> section.
<b>.recommend.priority</b>	Specify the <b>priority</b> with an integer value between <b>0</b> and <b>99</b> . A larger number gets lower priority, so a priority of <b>99</b> is lower than a priority of <b>10</b> . If a node can be matched with multiple profiles according to rules defined in the <b>match</b> field, the profile with the higher priority is applied to that node.
<b>.recommend.match</b>	Specify <b>.recommend.match</b> rules with <b>nodeLabel</b> or <b>nodeName</b> values.
<b>.recommend.match.nodeLabel</b>	Set <b>nodeLabel</b> with the <b>key</b> of the <b>node.Labels</b> field from the node object by using the <b>oc get nodes --show-labels</b> command. For example, <b>node-role.kubernetes.io/worker</b> .
<b>.recommend.match.nodeName</b>	Set <b>nodeName</b> with the value of the <b>node.Name</b> field from the node object by using the <b>oc get nodes</b> command. For example, <b>compute-1.example.com</b> .

#### 14.2.5.1. Grandmaster clock class sync state reference

The following table describes the PTP grandmaster clock (T-GM) **gm.ClockClass** states. Clock class states categorize T-GM clocks based on their accuracy and stability with regard to the Primary Reference Time Clock (PRTC) or other timing source.

Holdover specification is the amount of time a PTP clock can maintain synchronization without receiving updates from the primary time source.

Table 14.2. T-GM clock class states

Clock class state	Description
<b>gm.ClockClass 6</b>	T-GM clock is connected to a PRTC in <b>LOCKED</b> mode. For example, the PRTC is traceable to a GNSS time source.
<b>gm.ClockClass 7</b>	T-GM clock is in <b>HOLDOVER</b> mode, and within holdover specification. The clock source might not be traceable to a category 1 frequency source.
<b>gm.ClockClass 248</b>	T-GM clock is in <b>FREERUN</b> mode.

For more information, see "[Phase/time traceability information](#)", ITU-T G.8275.1/Y.1369.1 [Recommendations](#).

#### 14.2.5.2. Intel E810 NIC hardware configuration reference

Use this information to understand how to use the [Intel E810 hardware plugin](#) to configure the E810 network interface as PTP grandmaster clock. Hardware pin configuration determines how the network

interface interacts with other components and devices in the system. The Intel E810 NIC has four connectors for external 1PPS signals: **SMA1**, **SMA2**, **U.FL1**, and **U.FL2**.

**Table 14.3. Intel E810 NIC hardware connectors configuration**

Hardware pin	Recommended setting	Description
<b>U.FL1</b>	<b>0 1</b>	Disables the <b>U.FL1</b> connector input. The <b>U.FL1</b> connector is output-only.
<b>U.FL2</b>	<b>0 2</b>	Disables the <b>U.FL2</b> connector output. The <b>U.FL2</b> connector is input-only.
<b>SMA1</b>	<b>0 1</b>	Disables the <b>SMA1</b> connector input. The <b>SMA1</b> connector is bidirectional.
<b>SMA2</b>	<b>0 2</b>	Disables the <b>SMA2</b> connector output. The <b>SMA2</b> connector is bidirectional.



#### NOTE

**SMA1** and **U.FL1** connectors share channel one. **SMA2** and **U.FL2** connectors share channel two.

Set **spec.profile.plugins.e810.ulbxCmds** parameters to configure the GNSS clock in the **PtpConfig** custom resource (CR).



#### IMPORTANT

You must configure an offset value to compensate for T-GM GPS antenna cable signal delay. To configure the optimal T-GM antenna offset value, make precise measurements of the GNSS antenna cable signal delay. Red Hat cannot assist in this measurement or provide any values for the required delay offsets.

Each of these **ulbxCmds** stanzas correspond to a configuration that is applied to the host NIC by using **ubxtool** commands. For example:

```
ulbxCmds:
  - args:
    - "-P"
    - "29.20"
    - "-z"
    - "CFG-HW-ANT_CFG_VOLTCTRL,1"
    - "-z"
    - "CFG-TP-ANT_CABLEDELAY,<antenna_delay_offset>"①
  reportOutput: false
```

- 1** Measured T-GM antenna delay offset in nanoseconds. To get the required delay offset value, you must measure the cable delay using external test equipment.

The following table describes the equivalent **ubxtool** commands:

**Table 14.4. Intel E810 ublxCmds configuration**

ubxtool command	Description
<b>ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1 -z CFG-TP-ANT_CABLEDELAY,&lt;antenna_delay_offset&gt;</b>	Enables antenna voltage control, allows antenna status to be reported in the <b>UBX-MON-RF</b> and <b>UBX-INF-NOTICE</b> log messages, and sets a <b>&lt;antenna_delay_offset&gt;</b> value in nanoseconds that offsets the GPS antenna cable signal delay.
<b>ubxtool -P 29.20 -e GPS</b>	Enables the antenna to receive GPS signals.
<b>ubxtool -P 29.20 -d Galileo</b>	Configures the antenna to receive signal from the Galileo GPS satellite.
<b>ubxtool -P 29.20 -d GLONASS</b>	Disables the antenna from receiving signal from the GLONASS GPS satellite.
<b>ubxtool -P 29.20 -d BeiDou</b>	Disables the antenna from receiving signal from the BeiDou GPS satellite.
<b>ubxtool -P 29.20 -d SBAS</b>	Disables the antenna from receiving signal from the SBAS GPS satellite.
<b>ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000</b>	Configures the GNSS receiver survey-in process to improve its initial position estimate. This can take up to 24 hours to achieve an optimal result.
<b>ubxtool -P 29.20 -p MON-HW</b>	Runs a single automated scan of the hardware and reports on the NIC state and configuration settings.

#### 14.2.5.3. Dual E810 NIC configuration reference

Use this information to understand how to use the [Intel E810 hardware plugin](#) to configure a pair of E810 network interfaces as PTP grandmaster clock (T-GM).

Before you configure the dual-NIC cluster host, you must connect the two NICs with an SMA1 cable using the 1PPS faceplate connections.

When you configure a dual-NIC T-GM, you need to compensate for the 1PPS signal delay that occurs when you connect the NICs using the SMA1 connection ports. Various factors such as cable length, ambient temperature, and component and manufacturing tolerances can affect the signal delay. To compensate for the delay, you must calculate the specific value that you use to offset the signal delay.

**Table 14.5. E810 dual-NIC T-GM PtpConfig CR reference**

PtpConfig field	Description
<b>spec.profile.plugins.e810.pins</b>	<p>Configure the E810 hardware pins using the PTP Operator E810 hardware plugin.</p> <ul style="list-style-type: none"> <li>• Pin <b>2 1</b> enables the <b>1PPS OUT</b> connection for <b>SMA1</b> on NIC one.</li> <li>• Pin <b>1 1</b> enables the <b>1PPS IN</b> connection for <b>SMA1</b> on NIC two.</li> </ul>
<b>spec.profile.ts2phcConf</b>	<p>Use the <b>ts2phcConf</b> field to configure parameters for NIC one and NIC two. Set <b>ts2phc.master 0</b> for NIC two. This configures the timing source for NIC two from the 1PPS input, not GNSS. Configure the <b>ts2phc.extts_correction</b> value for NIC two to compensate for the delay that is incurred for the specific SMA cable and cable length that you use. The value that you configure depends on your specific measurements and SMA1 cable length.</p>
<b>spec.profile.ptp4lConf</b>	<p>Set the value of <b>boundary_clock_jbod</b> to 1 to enable support for multiple NICs.</p>

#### 14.2.5.4. 3-card E810 NIC configuration reference

Use this information to understand how to configure 3 E810 NICs as PTP grandmaster clock (T-GM).

Before you configure the 3-card cluster host, you must connect the 3 NICs by using the 1PPS faceplate connections. The primary NIC **1PPS\_out** outputs feed the other 2 NICs.

When you configure a 3-card T-GM, you need to compensate for the 1PPS signal delay that occurs when you connect the NICs by using the SMA1 connection ports. Various factors such as cable length, ambient temperature, and component and manufacturing tolerances can affect the signal delay. To compensate for the delay, you must calculate the specific value that you use to offset the signal delay.

**Table 14.6. 3-card E810 T-GM PtpConfig CR reference**

PtpConfig field	Description
<b>spec.profile.plugins.e810.pins</b>	<p>Configure the E810 hardware pins with the PTP Operator E810 hardware plugin.</p> <ul style="list-style-type: none"> <li>• <b>\$iface_timeTx1.SMA1</b> enables the <b>1PPS OUT</b> connection for <b>SMA1</b> on NIC 1.</li> <li>• <b>\$iface_timeTx1.SMA2</b> enables the <b>1PPS OUT</b> connection for <b>SMA2</b> on NIC 1.</li> <li>• <b>\$iface_timeTx2.SMA1</b> and <b>\$iface_timeTx3.SMA1</b> enables the <b>1PPS IN</b> connection for <b>SMA1</b> on NIC 2 and NIC 3.</li> <li>• <b>\$iface_timeTx2.SMA2</b> and <b>\$iface_timeTx3.SMA2</b> disables the <b>SMA2</b> connection on NIC 2 and NIC 3.</li> </ul>

PtpConfig field	Description
<b>spec.profile.ts2phcConf</b>	Use the <b>ts2phcConf</b> field to configure parameters for the NICs. Set <b>ts2phc.master 0</b> for NIC 2 and NIC 3. This configures the timing source for NIC 2 and NIC 3 from the 1PPS input, not GNSS. Configure the <b>ts2phc.extts_correction</b> value for NIC 2 and NIC 3 to compensate for the delay that is incurred for the specific SMA cable and cable length that you use. The value that you configure depends on your specific measurements and SMA1 cable length.
<b>spec.profile.ptp4lConf</b>	Set the value of <b>boundary_clock_jbod</b> to 1 to enable support for multiple NICs.

#### 14.2.6. Holdover in a grandmaster clock with GNSS as the source

Holdover allows the grandmaster (T-GM) clock to maintain synchronization performance when the global navigation satellite system (GNSS) source is unavailable. During this period, the T-GM clock relies on its internal oscillator and holdover parameters to reduce timing disruptions.

You can define the holdover behavior by configuring the following holdover parameters in the **PTPConfig** custom resource (CR):

##### **MaxInSpecOffset**

Specifies the maximum allowed offset in nanoseconds. If the T-GM clock exceeds the **MaxInSpecOffset** value, it transitions to the **FREERUN** state (clock class state **gm.ClockClass 248**).

##### **LocalHoldoverTimeout**

Specifies the maximum duration, in seconds, for which the T-GM clock remains in the holdover state before transitioning to the **FREERUN** state.

##### **LocalMaxHoldoverOffSet**

Specifies the maximum offset that the T-GM clock can reach during the holdover state in nanoseconds.

If the **MaxInSpecOffset** value is less than the **LocalMaxHoldoverOffSet** value, and the T-GM clock exceeds the maximum offset value, the T-GM clock transitions from the holdover state to the **FREERUN** state.



##### IMPORTANT

If the **LocalMaxHoldoverOffSet** value is less than the **MaxInSpecOffset** value, the holdover timeout occurs before the clock reaches the maximum offset. To resolve this issue, set the **MaxInSpecOffset** field and the **LocalMaxHoldoverOffSet** field to the same value.

For information about clock class states, see "Grandmaster clock class sync state reference" document.

The T-GM clock uses the holdover parameters **LocalMaxHoldoverOffSet** and **LocalHoldoverTimeout** to calculate the slope. Slope is the rate at which the phase offset changes over time. It is measured in nanoseconds per second, where the set value indicates how much the offset increases over a given time period.

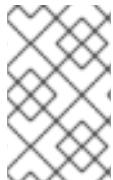
The T-GM clock uses the slope value to predict and compensate for time drift, so reducing timing disruptions during holdover. The T-GM clock uses the following formula to calculate the slope:

- Slope = **localMaxHoldoverOffset / localHoldoverTimeout**

For example, if the **LocalHoldOverTimeout** parameter is set to 60 seconds, and the **LocalMaxHoldoverOffset** parameter is set to 3000 nanoseconds, the slope is calculated as follows:

$$\text{Slope} = 3000 \text{ nanoseconds} / 60 \text{ seconds} = 50 \text{ nanoseconds per second}$$

The T-GM clock reaches the maximum offset in 60 seconds.



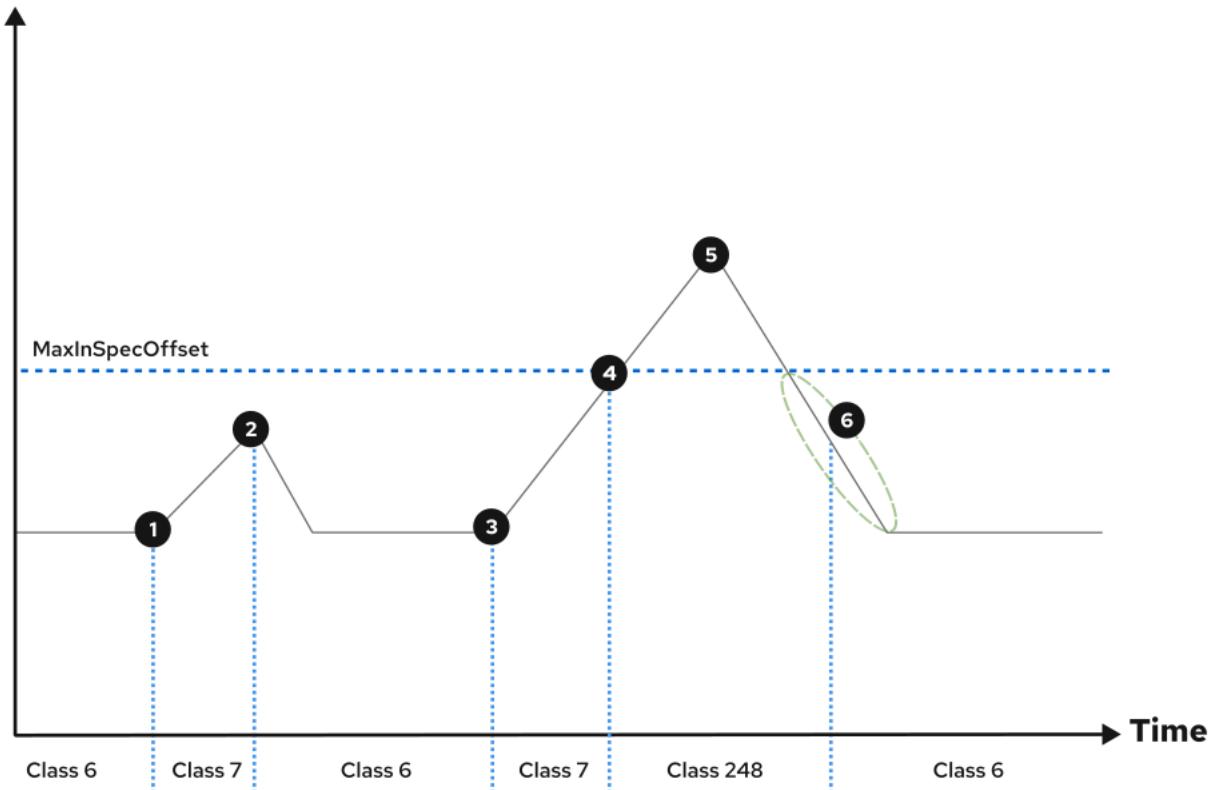
### NOTE

The phase offset is converted from picoseconds to nanoseconds. As a result, the calculated phase offset during holdover is expressed in nanoseconds, and the resulting slope is expressed in nanoseconds per second.

The following figure illustrates the holdover behavior in a T-GM clock with GNSS as the source:

**Figure 14.5. Holdover in a T-GM clock with GNSS as the source**

#### Time error



openshift-pg-holdover-tgm-clock-with-gnss.svg

- 1 The GNSS signal is lost, causing the T-GM clock to enter the **HOLDOVER** mode. The T-GM clock maintains time accuracy by using its internal clock.
- 2 The GNSS signal is restored and the T-GM clock re-enters the **LOCKED** mode. When the GNSS signal is restored, the T-GM clock re-enters the **LOCKED** mode only after all dependent components in the synchronization chain, such as **ts2phc** offset, digital phase-locked loop (DPLL) phase offset, and GNSS offset, reach a stable **LOCKED** mode.

- 3 The GNSS signal is lost again, and the T-GM clock re-enters the **HOLDOVER** mode. The time error begins to increase.
- 4 The time error exceeds the **MaxInSpecOffset** threshold due to prolonged loss of traceability.
- 5 The GNSS signal is restored, and the T-GM clock resumes synchronization. The time error starts to decrease.
- 6 The time error decreases and falls back within the **MaxInSpecOffset** threshold.

## Additional resources

- [Grandmaster clock class sync state reference](#)

### 14.2.7. Configuring dynamic leap seconds handling for PTP grandmaster clocks

The PTP Operator container image includes the latest **leap-seconds.list** file that is available at the time of release. You can configure the PTP Operator to automatically update the leap second file by using Global Positioning System (GPS) announcements.

Leap second information is stored in an automatically generated **ConfigMap** resource named **leap-configmap** in the **openshift-ptp** namespace. The PTP Operator mounts the **leap-configmap** resource as a volume in the **linuxptp-daemon** pod that is accessible by the **ts2phc** process.

If the GPS satellite broadcasts new leap second data, the PTP Operator updates the **leap-configmap** resource with the new data. The **ts2phc** process picks up the changes automatically.



#### NOTE

The following procedure is provided as reference. The 4.18 version of the PTP Operator enables automatic leap second management by default.

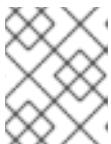
## Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the PTP Operator and configured a PTP grandmaster clock (T-GM) in the cluster.

## Procedure

1. Configure automatic leap second handling in the **phc2sysOpts** section of the **PtpConfig** CR. Set the following options:

```
phc2sysOpts: -r -u 0 -m -N 8 -R 16 -S 2 -s ens2f0 -n 24 ①
```



#### NOTE

Previously, the T-GM required an offset adjustment in the **phc2sys** configuration (**-O -37**) to account for historical leap seconds. This is no longer needed.

- Configure the Intel e810 NIC to enable periodical reporting of **NAV-TIMELS** messages by the GPS receiver in the **spec.profile.plugins.e810.ublxCmds** section of the **PtpConfig** CR. For example:

```
- args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
  - "-P"
  - "29.20"
  - "-p"
  - "CFG-MSG,1,38,248"
```

## Verification

- Validate that the configured T-GM is receiving **NAV-TIMELS** messages from the connected GPS. Run the following command:

```
$ oc -n openshift-ptp -c linuxptp-daemon-container exec -it $(oc -n openshift-ptp get pods -o name | grep daemon) -- ubxtool -t -p NAV-TIMELS -P 29.20
```

## Example output

```
1722509534.4417
UBX-NAV-STATUS:
iTOW 384752000 gpsFix 5 flags 0xdd fixStat 0x0 flags2 0x8
ttff 18261, msss 1367642864

1722509534.4419
UBX-NAV-TIMELS:
iTOW 384752000 version 0 reserved2 0 0 0 srcOfCurrLs 2
currLs 18 srcOfLsChange 2 lsChange 0 timeToLsEvent 70376866
dateOfLsGpsWn 2441 dateOfLsGpsDn 7 reserved2 0 0 0
valid x3

1722509534.4421
UBX-NAV-CLOCK:
iTOW 384752000 clkB 784281 clkD 435 tAcc 3 fAcc 215

1722509535.4477
UBX-NAV-STATUS:
iTOW 384753000 gpsFix 5 flags 0xdd fixStat 0x0 flags2 0x8
ttff 18261, msss 1367643864

1722509535.4479
UBX-NAV-CLOCK:
iTOW 384753000 clkB 784716 clkD 435 tAcc 3 fAcc 218
```

- Validate that the **leap-configmap** resource has been successfully generated by the PTP Operator and is up to date with the latest version of the [leap-seconds.list](#). Run the following command:

```
$ oc -n openshift-ptp get configmap leap-configmap -o jsonpath='{.data.<node_name>}'
```

**1** Replace **<node\_name>** with the node where you have installed and configured the PTP T-GM clock with automatic leap second management. Escape special characters in the node name. For example, **node-1\example\com**.

## Example output

```
# Do not edit
# This file is generated automatically by linuxptp-daemon
#$ 3913697179
#@ 4291747200
2272060800 10 # 1 Jan 1972
2287785600 11 # 1 Jul 1972
2303683200 12 # 1 Jan 1973
2335219200 13 # 1 Jan 1974
2366755200 14 # 1 Jan 1975
2398291200 15 # 1 Jan 1976
2429913600 16 # 1 Jan 1977
2461449600 17 # 1 Jan 1978
2492985600 18 # 1 Jan 1979
2524521600 19 # 1 Jan 1980
2571782400 20 # 1 Jul 1981
2603318400 21 # 1 Jul 1982
2634854400 22 # 1 Jul 1983
2698012800 23 # 1 Jul 1985
2776982400 24 # 1 Jan 1988
2840140800 25 # 1 Jan 1990
2871676800 26 # 1 Jan 1991
2918937600 27 # 1 Jul 1992
2950473600 28 # 1 Jul 1993
2982009600 29 # 1 Jul 1994
3029443200 30 # 1 Jan 1996
3076704000 31 # 1 Jul 1997
3124137600 32 # 1 Jan 1999
3345062400 33 # 1 Jan 2006
3439756800 34 # 1 Jan 2009
3550089600 35 # 1 Jul 2012
3644697600 36 # 1 Jul 2015
3692217600 37 # 1 Jan 2017

#h e65754d4 8f39962b aa854a61 661ef546 d2af0bfa
```

### 14.2.8. Configuring linuxptp services as a boundary clock

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**) as boundary clock by creating a **PtpConfig** custom resource (CR) object.



#### NOTE

Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as the boundary clock for your particular hardware and environment. This example CR does not configure PTP fast events. To configure PTP fast events, set appropriate values for **ptp4IOpts**, **ptp4IConf**, and **ptpClockThreshold**. **ptpClockThreshold** is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

#### Prerequisites

- Install the OpenShift CLI (**oc**).

- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

## Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **boundary-clock-ptp-config.yaml** file.

### Example PTP boundary clock configuration

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: boundary-clock
      ptpt4lOpts: "-2"
      phc2sysOpts: "-a -r -n 24"
      ptptSchedulingPolicy: SCHED_FIFO
      ptptSchedulingPriority: 10
      ptptSettings:
        logReduce: "true"
      ptpt4lConf: |
        # The interface name is hardware-specific
        [$iface_slave]
        masterOnly 0
        [$iface_master_1]
        masterOnly 1
        [$iface_master_2]
        masterOnly 1
        [$iface_master_3]
        masterOnly 1
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 248
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison G.8275.x
        G.8275.defaultDS.localPriority 128
        #
```

```
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
```

```

p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0xA0
recommend:
- profile: boundary-clock
  priority: 4
  match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"

```

Table 14.7. PTP boundary clock CR configuration options

CR field	Description
<b>name</b>	The name of the <b>PtpConfig</b> CR.
<b>profile</b>	Specify an array of one or more <b>profile</b> objects.
<b>name</b>	Specify the name of a profile object which uniquely identifies a profile object.
<b>ptp4lOpts</b>	Specify system config options for the <b>ptp4l</b> service. The options should not include the network interface name <b>-i &lt;interface&gt;</b> and service config file <b>-f /etc/ptp4l.conf</b> because the network interface name and the service config file are automatically appended.
<b>ptp4lConf</b>	Specify the required configuration to start <b>ptp4l</b> as boundary clock. For example, <b>ens1f0</b> synchronizes from a grandmaster clock and <b>ens1f3</b> synchronizes connected devices.
<b>&lt;interface_1&gt;</b>	The interface that receives the synchronization clock.

CR field	Description
<code>&lt;interface_2&gt;</code>	The interface that sends the synchronization clock.
<code>tx_timestamp_time_out</code>	For Intel Columbiaville 800 Series NICs, set <code>tx_timestamp_timeout</code> to <b>50</b> .
<code>boundary_clock_jbod</code>	For Intel Columbiaville 800 Series NICs, ensure <code>boundary_clock_jbod</code> is set to <b>0</b> . For Intel Fortville X710 Series NICs, ensure <code>boundary_clock_jbod</code> is set to <b>1</b> .
<code>phc2sysOpts</code>	Specify system config options for the <code>phc2sys</code> service. If this field is empty, the PTP Operator does not start the <code>phc2sys</code> service.
<code>ptpSchedulingPolicy</code>	Scheduling policy for ptp4l and phc2sys processes. Default value is <b>SCHED_OTHER</b> . Use <b>SCHED_FIFO</b> on systems that support FIFO scheduling.
<code>ptpSchedulingPriority</code>	Integer value from 1-65 used to set FIFO priority for <code>ptp4l</code> and <code>phc2sys</code> processes when <code>ptpSchedulingPolicy</code> is set to <b>SCHED_FIFO</b> . The <code>ptpSchedulingPriority</code> field is not used when <code>ptpSchedulingPolicy</code> is set to <b>SCHED_OTHER</b> .
<code>ptpClockThreshold</code>	Optional. If <code>ptpClockThreshold</code> is not present, default values are used for the <code>ptpClockThreshold</code> fields. <code>ptpClockThreshold</code> configures how long after the PTP master clock is disconnected before PTP events are triggered. <code>holdOverTimeout</code> is the time value in seconds before the PTP clock event state changes to <b>FREERUN</b> when the PTP master clock is disconnected. The <code>maxOffsetThreshold</code> and <code>minOffsetThreshold</code> settings configure offset values in nanoseconds that compare against the values for <b>CLOCK_REALTIME (phc2sys)</b> or master offset ( <code>ptp4l</code> ). When the <code>ptp4l</code> or <code>phc2sys</code> offset value is outside this range, the PTP clock state is set to <b>FREERUN</b> . When the offset value is within this range, the PTP clock state is set to <b>LOCKED</b> .
<code>recommend</code>	Specify an array of one or more <code>recommend</code> objects that define rules on how the <code>profile</code> should be applied to nodes.
<code>.recommend.profile</code>	Specify the <code>.recommend.profile</code> object name defined in the <code>profile</code> section.
<code>.recommend.priority</code>	Specify the <code>priority</code> with an integer value between <b>0</b> and <b>99</b> . A larger number gets lower priority, so a priority of <b>99</b> is lower than a priority of <b>10</b> . If a node can be matched with multiple profiles according to rules defined in the <code>match</code> field, the profile with the higher priority is applied to that node.
<code>.recommend.match</code>	Specify <code>.recommend.match</code> rules with <code>nodeLabel</code> or <code>nodeName</code> values.

CR field	Description
<code>.recommend.match.nodeLabel</code>	Set <b>nodeLabel</b> with the <b>key</b> of the <b>node.Labels</b> field from the node object by using the <b>oc get nodes --show-labels</b> command. For example, <b>node-role.kubernetes.io/worker</b> .
<code>.recommend.match.nodeName</code>	Set <b>nodeName</b> with the value of the <b>node.Name</b> field from the node object by using the <b>oc get nodes</b> command. For example, <b>compute-1.example.com</b> .

2. Create the CR by running the following command:

```
$ oc create -f boundary-clock-ptp-config.yaml
```

## Verification

- Check that the **PtpConfig** profile is applied to the node.
- Get the list of pods in the **openshift-ptp** namespace by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-4xkbb	1/1	Running	0	43m	10.1.196.24	compute-0.example.com
linuxptp-daemon-tdspf	1/1	Running	0	43m	10.1.196.25	compute-1.example.com
ptp-operator-657bbb64c8-2f8sj	1/1	Running	0	43m	10.129.0.61	control-plane-1.example.com

- Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
```

### Example output

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] -----
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4IOpts: -2
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] -----
```

## Additional resources

- Configuring FIFO priority scheduling for PTP hardware
- Configuring the PTP fast event notifications publisher

#### 14.2.8.1. Configuring linuxptp services as boundary clocks for dual-NIC hardware

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**) as boundary clocks for dual-NIC hardware by creating a **PtpConfig** custom resource (CR) object for each NIC.

Dual NIC hardware allows you to connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

#### Procedure

- 1 Create two separate **PtpConfig** CRs, one for each NIC, using the reference CR in "Configuring linuxptp services as a boundary clock" as the basis for each CR. For example:
  - a. Create **boundary-clock-ptp-config-nic1.yaml**, specifying values for **phc2sysOpts**:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic1
  namespace: openshift-ptp
spec:
  profile:
    - name: "profile1"
      ptp4lOpts: "-2 --summary_interval -4"
      ptp4lConf: | ①
        [ens5f1]
        masterOnly 1
        [ens5f0]
        masterOnly 0
      ...
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ②
```

- Specify the required interfaces to start **ptp4l** as a boundary clock. For example, **ens5f0** synchronizes from a grandmaster clock and **ens5f1** synchronizes connected devices.
- Required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

- b. Create **boundary-clock-ptp-config-nic2.yaml**, removing the **phc2sysOpts** field altogether to disable the **phc2sys** service for the second NIC:

```
apiVersion: ptp.openshift.io/v1
```

```

kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
    - name: "profile2"
      ptp4lOpts: "-2 --summary_interval -4"
      ptp4lConf: | ①
        [ens7f1]
        masterOnly 1
        [ens7f0]
        masterOnly 0
...
  
```

- ① Specify the required interfaces to start **ptp4l** as a boundary clock on the second NIC.



#### NOTE

You must completely remove the **phc2sysOpts** field from the second **PtpConfig** CR to disable the **phc2sys** service on the second NIC.

2. Create the dual-NIC **PtpConfig** CRs by running the following commands:

- a. Create the CR that configures PTP for the first NIC:

```
$ oc create -f boundary-clock-ptp-config-nic1.yaml
```

- b. Create the CR that configures PTP for the second NIC:

```
$ oc create -f boundary-clock-ptp-config-nic2.yaml
```

#### Verification

- Check that the PTP Operator has applied the **PtpConfig** CRs for both NICs. Examine the logs for the **linuxptp** daemon corresponding to the node that has the dual-NIC hardware installed. For example, run the following command:

```
$ oc logs linuxptp-daemon-cvgr6 -n openshift-ptp -c linuxptp-daemon-container
```

#### Example output

```

ptp4l[80828.335]: [ptp4l.1.config] master offset      5 s2 freq -5727 path delay   519
ptp4l[80828.343]: [ptp4l.0.config] master offset     -5 s2 freq -10607 path delay   533
phc2sys[80828.390]: [ptp4l.0.config] CLOCK_REALTIME phc offset      1 s2 freq -87239
delay   539
  
```

#### 14.2.8.2. Configuring **linuxptp** as a highly available system clock for dual-NIC Intel E810 PTP boundary clocks

You can configure the **linuxptp** services **ptp4l** and **phc2sys** as a highly available (HA) system clock for dual PTP boundary clocks (T-BC).

The highly available system clock uses multiple time sources from dual-NIC Intel E810 Salem channel hardware configured as two boundary clocks. Two boundary clocks instances participate in the HA setup, each with its own configuration profile. You connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

Create two **PtpConfig** custom resource (CR) objects that configure the NICs as T-BC and a third **PtpConfig** CR that configures high availability between the two NICs.



### IMPORTANT

You set **phc2SysOpts** options once in the **PtpConfig** CR that configures HA. Set the **phc2sysOpts** field to an empty string in the **PtpConfig** CRs that configure the two NICs. This prevents individual **phc2sys** processes from being set up for the two profiles.

The third **PtpConfig** CR configures a highly available system clock service. The CR sets the **ptp4IOpts** field to an empty string to prevent the **ptp4l** process from running. The CR adds profiles for the **ptp4l** configurations under the **spec.profile.ptpSettings.haProfiles** key and passes the kernel socket path of those profiles to the **phc2sys** service. When a **ptp4l** failure occurs, the **phc2sys** service switches to the backup **ptp4l** configuration. When the primary profile becomes active again, the **phc2sys** service reverts to the original state.



### IMPORTANT

Ensure that you set **spec.recommend.priority** to the same value for all three **PtpConfig** CRs that you use to configure HA.

## Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.
- Configure a cluster node with Intel E810 Salem channel dual-NIC.

## Procedure

1. Create two separate **PtpConfig** CRs, one for each NIC, using the CRs in "Configuring linuxptp services as boundary clocks for dual-NIC hardware" as a reference for each CR.
  - a. Create the **ha-ptp-config-nic1.yaml** file, specifying an empty string for the **phc2sysOpts** field. For example:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ha-ptp-config-nic1
  namespace: openshift-ptp
spec:
  profile:
    - name: "ha-ptp-config-profile1"
      ptp4IOpts: "-2 --summary_interval -4"
      ptp4IConf: | 1
        [ens5f1]
```

```

masterOnly 1
[ens5f0]
masterOnly 0
#...
phc2sysOpts: "" ②

```

- ① Specify the required interfaces to start **ptp4l** as a boundary clock. For example, **ens5f0** synchronizes from a grandmaster clock and **ens5f1** synchronizes connected devices.
- ② Set **phc2sysOpts** with an empty string. These values are populated from the **spec.profile.ptpSettings.haProfiles** field of the **PtpConfig** CR that configures high availability.

- b. Apply the **PtpConfig** CR for NIC 1 by running the following command:

```
$ oc create -f ha-ptp-config-nic1.yaml
```

- c. Create the **ha-ptp-config-nic2.yaml** file, specifying an empty string for the **phc2sysOpts** field. For example:

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ha-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
    - name: "ha-ptp-config-profile2"
      ptp4IOpts: "-2 --summary_interval -4"
      ptp4ICnf: |
        [ens7f1]
        masterOnly 1
        [ens7f0]
        masterOnly 0
      #...
      phc2sysOpts: ""

```

- d. Apply the **PtpConfig** CR for NIC 2 by running the following command:

```
$ oc create -f ha-ptp-config-nic2.yaml
```

2. Create the **PtpConfig** CR that configures the HA system clock. For example:

- a. Create the **ptp-config-for-ha.yaml** file. Set **haProfiles** to match the **metadata.name** fields that are set in the **PtpConfig** CRs that configure the two NICs. For example: **haProfiles: ha-ptp-config-nic1,ha-ptp-config-nic2**

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-ha
  namespace: openshift-ptp
  annotations: {}
spec:

```

```

profile:
  - name: "boundary-ha"
    ptp4lOpts: "" 1
    phc2sysOpts: "-a -r -n 24"
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
      haProfiles: "$profile1,$profile2"
recommend:
  - profile: "boundary-ha"
    priority: 4
    match:
      - nodeLabel: "node-role.kubernetes.io/$mcp"

```

- 1** Set the **ptp4lOpts** field to an empty string. If it is not empty, the **p4ptl** process starts with a critical error.



### IMPORTANT

Do not apply the high availability **PtpConfig** CR before the **PtpConfig** CRs that configure the individual NICs.

- Apply the HA **PtpConfig** CR by running the following command:

```
$ oc create -f ptpt-config-for-ha.yaml
```

### Verification

- Verify that the PTP Operator has applied the **PtpConfig** CRs correctly. Perform the following steps:
  - Get the list of pods in the **openshift-ptp** namespace by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-4xkrb	1/1	Running	0	43m	10.1.196.24	compute-0.example.com
ptp-operator-657bbq64c8-2f8sj	1/1	Running	0	43m	10.129.0.61	control-plane-1.example.com



### NOTE

There should be only one **linuxptp-daemon** pod.

- Check that the profile is correct by running the following command. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile.

```
$ oc logs linuxptp-daemon-4xkrb -n openshift-ptp -c linuxptp-daemon-container
```

## Example output

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] -----
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: ha-ptp-config-profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4IOpts: -2
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] -----
```

### 14.2.9. Configuring linuxptp services as an ordinary clock

You can configure **linuxptp** services (**ptp4l**, **phc2sys**) as ordinary clock by creating a **PtpConfig** custom resource (CR) object.



#### NOTE

Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as an ordinary clock for your particular hardware and environment. This example CR does not configure PTP fast events. To configure PTP fast events, set appropriate values for **ptp4IOpts**, **ptp4ICnf**, and **ptpClockThreshold**. **ptpClockThreshold** is required only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

#### Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **ordinary-clock-ptp-config.yaml** file.

#### Example PTP ordinary clock configuration

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ordinary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: ordinary-clock
      # The interface name is hardware-specific
      interface: $interface
      ptp4IOpts: "-2 -s"
      phc2sysOpts: "-a -r -n 24"
```

```
ptpSchedulingPolicy: SCHED_FIFO
ptpSchedulingPriority: 10
ptpSettings:
    logReduce: "true"
    ptp4lConf: |
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 1
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 255
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison G.8275.x
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
        logAnnounceInterval -3
        logSyncInterval -4
        logMinDelayReqInterval -4
        logMinPdelayReqInterval -4
        announceReceiptTimeout 3
        syncReceiptTimeout 0
        delayAsymmetry 0
        fault_reset_interval -4
        neighborPropDelayThresh 20000000
        masterOnly 0
        G.8275.portDS.localPriority 128
        #
        # Run time options
        #
        assume_two_step 0
        logging_level 6
        path_trace_enabled 0
        follow_up_info 0
        hybrid_e2e 0
        inhibit_multicast_service 0
        net_sync_monitor 0
        tc_spanning_tree 0
        tx_timestamp_timeout 50
        unicast_listen 0
        unicast_master_table 0
        unicast_req_duration 3600
        use_syslog 1
        verbose 0
        summary_interval 0
```

```
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type OC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 0
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0xA0
recommend:
- profile: ordinary-clock
priority: 4
match:
- nodeLabel: "node-role.kubernetes.io/$mcp"
```

Table 14.8. PTP ordinary clock CR configuration options

CR field	Description
<b>name</b>	The name of the <b>PtpConfig</b> CR.
<b>profile</b>	Specify an array of one or more <b>profile</b> objects. Each profile must be uniquely named.
<b>interface</b>	Specify the network interface to be used by the <b>ptp4l</b> service, for example <b>ens787f1</b> .
<b>ptp4lOpts</b>	Specify system config options for the <b>ptp4l</b> service, for example <b>-2</b> to select the IEEE 802.3 network transport. The options should not include the network interface name <b>-i &lt;interface&gt;</b> and service config file <b>-f /etc/ptp4l.conf</b> because the network interface name and the service config file are automatically appended. Append <b>--summary_interval -4</b> to use PTP fast events with this interface.
<b>phc2sysOpts</b>	Specify system config options for the <b>phc2sys</b> service. If this field is empty, the PTP Operator does not start the <b>phc2sys</b> service. For Intel Columbiaiville 800 Series NICs, set <b>phc2sysOpts</b> options to <b>-a -r -m -n 24 -N 8 -R 16. -m</b> prints messages to <b>stdout</b> . The <b>linuxptp-daemon DaemonSet</b> parses the logs and generates Prometheus metrics.
<b>ptp4lConf</b>	Specify a string that contains the configuration to replace the default <b>/etc/ptp4l.conf</b> file. To use the default configuration, leave the field empty.
<b>tx_timestamp_time_out</b>	For Intel Columbiaiville 800 Series NICs, set <b>tx_timestamp_timeout</b> to <b>50</b> .
<b>boundary_clock_jbod</b>	For Intel Columbiaiville 800 Series NICs, set <b>boundary_clock_jbod</b> to <b>0</b> .
<b>ptpSchedulingPolicy</b>	Scheduling policy for <b>ptp4l</b> and <b>phc2sys</b> processes. Default value is <b>SCHED_OTHER</b> . Use <b>SCHED_FIFO</b> on systems that support FIFO scheduling.
<b>ptpSchedulingPriority</b>	Integer value from 1-65 used to set FIFO priority for <b>ptp4l</b> and <b>phc2sys</b> processes when <b>ptpSchedulingPolicy</b> is set to <b>SCHED_FIFO</b> . The <b>ptpSchedulingPriority</b> field is not used when <b>ptpSchedulingPolicy</b> is set to <b>SCHED_OTHER</b> .

CR field	Description
<b>ptpClockThreshold</b>	Optional. If <b>ptpClockThreshold</b> is not present, default values are used for the <b>ptpClockThreshold</b> fields. <b>ptpClockThreshold</b> configures how long after the PTP master clock is disconnected before PTP events are triggered. <b>holdOverTimeout</b> is the time value in seconds before the PTP clock event state changes to <b>FREERUN</b> when the PTP master clock is disconnected. The <b>maxOffsetThreshold</b> and <b>minOffsetThreshold</b> settings configure offset values in nanoseconds that compare against the values for <b>CLOCK_REALTIME (phc2sys)</b> or master offset ( <b>ptp4l</b> ). When the <b>ptp4l</b> or <b>phc2sys</b> offset value is outside this range, the PTP clock state is set to <b>FREERUN</b> . When the offset value is within this range, the PTP clock state is set to <b>LOCKED</b> .
<b>recommend</b>	Specify an array of one or more <b>recommend</b> objects that define rules on how the <b>profile</b> should be applied to nodes.
<b>.recommend.profile</b>	Specify the <b>.recommend.profile</b> object name defined in the <b>profile</b> section.
<b>.recommend.priority</b>	Set <b>.recommend.priority</b> to <b>0</b> for ordinary clock.
<b>.recommend.match</b>	Specify <b>.recommend.match</b> rules with <b>nodeLabel</b> or <b>nodeName</b> values.
<b>.recommend.match.nodeLabel</b>	Set <b>nodeLabel</b> with the <b>key</b> of the <b>node.Labels</b> field from the node object by using the <b>oc get nodes --show-labels</b> command. For example, <b>node-role.kubernetes.io/worker</b> .
<b>.recommend.match.nodeName</b>	Set <b>nodeName</b> with the value of the <b>node.Name</b> field from the node object by using the <b>oc get nodes</b> command. For example, <b>compute-1.example.com</b> .

2. Create the **PtpConfig** CR by running the following command:

```
$ oc create -f ordinary-clock-ptp-config.yaml
```

## Verification

- Check that the **PtpConfig** profile is applied to the node.
  - Get the list of pods in the **openshift-ptp** namespace by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
------	-------	--------	----------	-----	----	------

```

linuxptp-daemon-4xkbb      1/1   Running  0      43m  10.1.196.24  compute-
0.example.com
linuxptp-daemon-tdspf      1/1   Running  0      43m  10.1.196.25  compute-
1.example.com
ptp-operator-657bbb64c8-2f8sj 1/1   Running  0      43m  10.129.0.61  control-
plane-1.example.com

```

- b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
```

### Example output

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] -----
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4IOpts: -2 -s
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] -----
```

### Additional resources

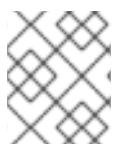
- [Configuring FIFO priority scheduling for PTP hardware](#)
- [Configuring the PTP fast event notifications publisher](#)

#### 14.2.9.1. Intel Columbiaville E800 series NIC as PTP ordinary clock reference

The following table describes the changes that you must make to the reference PTP configuration to use Intel Columbiaville E800 series NICs as ordinary clocks. Make the changes in a **PtpConfig** custom resource (CR) that you apply to the cluster.

**Table 14.9. Recommended PTP settings for Intel Columbiaville NIC**

PTP configuration	Recommended setting
<b>phc2sysOpts</b>	<b>-a -r -m -n 24 -N 8 -R 16</b>
<b>tx_timestamp_timeout</b>	<b>50</b>
<b>boundary_clock_jbod</b>	<b>0</b>



### NOTE

For **phc2sysOpts**, **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

### Additional resources

- For a complete example CR that configures **linuxptp** services as an ordinary clock with PTP fast events, see [Configuring linuxptp services as ordinary clock](#).

#### 14.2.10. Configuring FIFO priority scheduling for PTP hardware

In telco or other deployment types that require low latency performance, PTP daemon threads run in a constrained CPU footprint alongside the rest of the infrastructure components. By default, PTP threads run with the **SCHED\_OTHER** policy. Under high load, these threads might not get the scheduling latency they require for error-free operation.

To mitigate against potential scheduling latency errors, you can configure the PTP Operator **linuxptp** services to allow threads to run with a **SCHED\_FIFO** policy. If **SCHED\_FIFO** is set for a **PtpConfig** CR, then **ptp4l** and **phc2sys** will run in the parent container under **chrt** with a priority set by the **ptpSchedulingPriority** field of the **PtpConfig** CR.



#### NOTE

Setting **ptpSchedulingPolicy** is optional, and is only required if you are experiencing latency errors.

#### Procedure

1. Edit the **PtpConfig** CR profile:

```
$ oc edit PtpConfig -n openshift-ptp
```

2. Change the **ptpSchedulingPolicy** and **ptpSchedulingPriority** fields:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
    - name: "profile1"
...
  ptpSchedulingPolicy: SCHED_FIFO ①
  ptpSchedulingPriority: 10 ②
```

- 1 Scheduling policy for **ptp4l** and **phc2sys** processes. Use **SCHED\_FIFO** on systems that support FIFO scheduling.
- 2 Required. Sets the integer value 1-65 used to configure FIFO priority for **ptp4l** and **phc2sys** processes.

3. Save and exit to apply the changes to the **PtpConfig** CR.

#### Verification

1. Get the name of the **linuxptp-daemon** pod and corresponding node where the **PtpConfig** CR has been applied:

```
$ oc get pods -n openshift-ptp -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-gmv2n	3/3	Running	0	1d17h	10.1.196.24	compute-0.example.com
linuxptp-daemon-lgm55	3/3	Running	0	1d17h	10.1.196.25	compute-1.example.com
ptp-operator-3r4dcvf7f4-zndk7	1/1	Running	0	1d7h	10.129.0.61	control-plane-1.example.com

- Check that the **ptp4l** process is running with the updated **chrt** FIFO priority:

```
$ oc -n openshift-ptp logs linuxptp-daemon-lgm55 -c linuxptp-daemon-container|grep chrt
```

### Example output

```
I1216 19:24:57.091872 1600715 daemon.go:285] /bin/chrt -f 65 /usr/sbin/ptp4l -f /var/run/ptp4l.0.config -2 --summary_interval -4 -m
```

## 14.2.11. Configuring log filtering for linuxptp services

The **linuxptp** daemon generates logs that you can use for debugging purposes. In telco or other deployment types that feature a limited storage capacity, these logs can add to the storage demand.

To reduce the number log messages, you can configure the **PtpConfig** custom resource (CR) to exclude log messages that report the **master offset** value. The **master offset** log message reports the difference between the current node's clock and the master clock in nanoseconds.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

### Procedure

- Edit the **PtpConfig** CR:

```
$ oc edit PtpConfig -n openshift-ptp
```

- In **spec.profile**, add the **ptpSettings.logReduce** specification and set the value to **true**:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
```

```

profile:
- name: "profile1"
...
ptpSettings:
  logReduce: "true"

```

**NOTE**

For debugging purposes, you can revert this specification to **False** to include the master offset messages.

- Save and exit to apply the changes to the **PtpConfig** CR.

**Verification**

- Get the name of the **linuxptp-daemon** pod and corresponding node where the **PtpConfig** CR has been applied:

```
$ oc get pods -n openshift-ptp -o wide
```

**Example output**

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-gmv2n	3/3	Running	0	1d17h	10.1.196.24	compute-0.example.com
linuxptp-daemon-lgm55	3/3	Running	0	1d17h	10.1.196.25	compute-1.example.com
ptp-operator-3r4dcvf7f4-zndk7	1/1	Running	0	1d7h	10.129.0.61	control-plane-1.example.com

- Verify that master offset messages are excluded from the logs by running the following command:

```
$ oc -n openshift-ptp logs <linux_daemon_container> -c linuxptp-daemon-container | grep "master offset" ①
```

- ① <linux\_daemon\_container> is the name of the **linuxptp-daemon** pod, for example **linuxptp-daemon-gmv2n**.

When you configure the **logReduce** specification, this command does not report any instances of **master offset** in the logs of the **linuxptp** daemon.

**14.2.12. Troubleshooting common PTP Operator issues**

Troubleshoot common problems with the PTP Operator by performing the following steps.

**Prerequisites**

- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

- Install the PTP Operator on a bare-metal cluster with hosts that support PTP.

## Procedure

1. Check the Operator and operands are successfully deployed in the cluster for the configured nodes.

```
$ oc get pods -n openshift-ptp -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-lmvgn	3/3	Running	0	4d17h	10.1.196.24	compute-0.example.com
linuxptp-daemon-qhfg7	3/3	Running	0	4d17h	10.1.196.25	compute-1.example.com
ptp-operator-6b8dcf7f4-zndk7	1/1	Running	0	5d7h	10.129.0.61	control-plane-1.example.com



### NOTE

When the PTP fast event bus is enabled, the number of ready **linuxptp-daemon** pods is **3/3**. If the PTP fast event bus is not enabled, **2/2** is displayed.

2. Check that supported hardware is found in the cluster.

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io
```

### Example output

NAME	AGE
control-plane-0.example.com	10d
control-plane-1.example.com	10d
compute-0.example.com	10d
compute-1.example.com	10d
compute-2.example.com	10d

3. Check the available PTP network interfaces for a node:

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io <node_name> -o yaml
```

where:

**<node\_name>**

Specifies the node you want to query, for example, **compute-0.example.com**.

### Example output

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2021-09-14T16:52:33Z"
```

```

generation: 1
name: compute-0.example.com
namespace: openshift-ptp
resourceVersion: "177400"
uid: 30413db0-4d8d-46da-9bef-737bacd548fd
spec: {}
status:
devices:
- name: eno1
- name: eno2
- name: eno3
- name: eno4
- name: enp5s0f0
- name: enp5s0f1

```

4. Check that the PTP interface is successfully synchronized to the primary clock by accessing the **linuxptp-daemon** pod for the corresponding node.

- a. Get the name of the **linuxptp-daemon** pod and corresponding node you want to troubleshoot by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxptp-daemon-lmvgn	3/3	Running	0	4d17h	10.1.196.24	compute-0.example.com
linuxptp-daemon-qhfg7	3/3	Running	0	4d17h	10.1.196.25	compute-1.example.com
ptp-operator-6b8dcfb7f4-zndk7	1/1	Running	0	5d7h	10.129.0.61	control-plane-1.example.com

- b. Remote shell into the required **linuxptp-daemon** container:

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container <linux_daemon_container>
```

where:

<linux\_daemon\_container>

is the container you want to diagnose, for example **linuxptp-daemon-lmvgn**.

- c. In the remote shell connection to the **linuxptp-daemon** container, use the PTP Management Client (**pmc**) tool to diagnose the network interface. Run the following **pmc** command to check the sync status of the PTP device, for example **ptp4l**.

```
# pmc -u -f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

#### Example output when the node is successfully synced to the primary clock

```

sending: GET PORT_DATA_SET
40a6b7.ffe.166ef0-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
portIdentity      40a6b7.ffe.166ef0-1

```

```

portState          SLAVE
logMinDelayReqInterval -4
peerMeanPathDelay   0
logAnnounceInterval -3
announceReceiptTimeout 3
logSyncInterval     -4
delayMechanism      1
logMinPdelayReqInterval -4
versionNumber        2

```

- For GNSS-sourced grandmaster clocks, verify that the in-tree NIC ice driver is correct by running the following command, for example:

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-74m2g ethtool -i ens7f0
```

#### Example output

```

driver: ice
version: 5.14.0-356.bz2232515.el9.x86_64
firmware-version: 4.20 0x8001778b 1.3346.0

```

- For GNSS-sourced grandmaster clocks, verify that the **linuxptp-daemon** container is receiving signal from the GNSS antenna. If the container is not receiving the GNSS signal, the **/dev/gnss0** file is not populated. To verify, run the following command:

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-jnz6r cat /dev/gnss0
```

#### Example output

```

$GNRMC,125223.00,A,4233.24463,N,07126.64561,W,0.000,,300823,,,A,V*0A
$GNVTG,,T,,M,0.000,N,0.000,K,A*3D
$GNGGA,125223.00,4233.24463,N,07126.64561,W,1,12,99.99,98.6,M,-33.1,M,,*7E
$GNGSA,A,3,25,17,19,11,12,06,05,04,09,20,,,99.99,99.99,99.99,1*37
$GPGSV,3,1,10,04,12,039,41,05,31,222,46,06,50,064,48,09,28,064,42,1*62

```

### 14.2.13. Getting the DPLL firmware version for the CGU in an Intel 800 series NIC

You can get the digital phase-locked loop (DPLL) firmware version for the Clock Generation Unit (CGU) in an Intel 800 series NIC by opening a debug shell to the cluster node and querying the NIC hardware.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed an Intel 800 series NIC in the cluster host.
- You have installed the PTP Operator on a bare-metal cluster with hosts that support PTP.

#### Procedure

- Start a debug pod by running the following command:

```
$ oc debug node/<node_name>
```

where:

<node\_name>

Is the node where you have installed the Intel 800 series NIC.

- Check the CGU firmware version in the NIC by using the **devlink** tool and the bus and device name where the NIC is installed. For example, run the following command:

```
sh-4.4# devlink dev info <bus_name>/<device_name> | grep cgu
```

where:

<bus\_name>

Is the bus where the NIC is installed. For example, **pci**.

<device\_name>

Is the NIC device name. For example, **0000:51:00.0**.

### Example output

```
cgu.id 36 ①  
fw.cgu 8032.16973825.6021 ②
```

① CGU hardware revision number

② The DPLL firmware version running in the CGU, where the DPLL firmware version is **6201**, and the DPLL model is **8032**. The string **16973825** is a shorthand representation of the binary version of the DPLL firmware version (**1.3.0.1**).



### NOTE

The firmware version has a leading nibble and 3 octets for each part of the version number. The number **16973825** in binary is **0001 0000 0011 0000 0000 0001**. Use the binary value to decode the firmware version. For example:

Table 14.10. DPLL firmware version

Binary part	Decimal value
<b>0001</b>	1
<b>0000 0011</b>	3
<b>0000 0000</b>	0
<b>0000 0001</b>	1

### 14.2.14. Collecting PTP Operator data

You can use the **oc adm must-gather** command to collect information about your cluster, including features and objects associated with PTP Operator.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).
- You have installed the PTP Operator.

#### Procedure

- To collect PTP Operator data with **must-gather**, you must specify the PTP Operator **must-gather** image.

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/ptp-must-gather-rhel9:v4.18
```

## 14.3. DEVELOPING PTP EVENTS CONSUMER APPLICATIONS WITH THE REST API V2

When developing consumer applications that make use of Precision Time Protocol (PTP) events on a bare-metal cluster node, you deploy your consumer application in a separate application pod. The consumer application subscribes to PTP events by using the PTP events REST API v2.



#### NOTE

The following information provides general guidance for developing consumer applications that use PTP events. A complete events consumer application example is outside the scope of this information.

#### Additional resources

- [PTP events REST API v2 reference](#)

### 14.3.1. About the PTP fast event notifications framework

Use the Precision Time Protocol (PTP) fast event REST API v2 to subscribe cluster applications to PTP events that the bare-metal cluster node generates.



#### NOTE

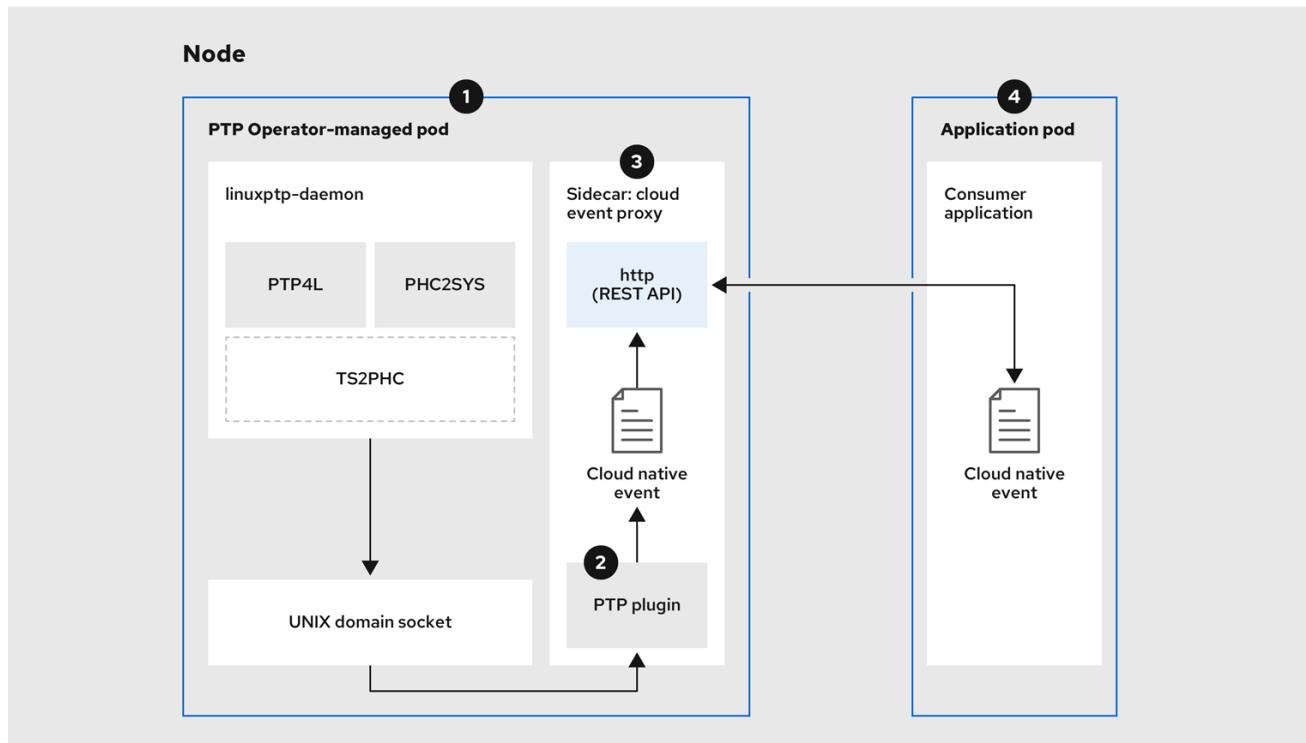
The fast events notifications framework uses a REST API for communication. The PTP events REST API v1 and v2 are based on the *O-RAN O-Cloud Notification API Specification for Event Consumers 4.0* that is available from [O-RAN ALLIANCE Specifications](#).

Only the PTP events REST API v2 is O-RAN v4 compliant.

### 14.3.2. Retrieving PTP events with the PTP events REST API v2

Applications subscribe to PTP events by using an O-RAN v4 compatible REST API in the producer-side cloud event proxy sidecar. The **cloud-event-proxy** sidecar container can access the same resources as the primary application container without using any of the resources of the primary application and with no significant latency.

**Figure 14.6. Overview of consuming PTP fast events from the PTP event producer REST API v2**



319\_OpenShift\_0323

### **1** Event is generated on the cluster host

The **linuxptp-daemon** process in the PTP Operator-managed pod runs as a Kubernetes **DaemonSet** and manages the various **linuxptp** processes (**ptp4l**, **phc2sys**, and optionally for grandmaster clocks, **ts2phc**). The **linuxptp-daemon** passes the event to the **UNIX domain socket**.

### **2** Event is passed to the cloud-event-proxy sidecar

The PTP plugin reads the event from the **UNIX domain socket** and passes it to the **cloud-event-proxy** sidecar in the PTP Operator-managed pod. **cloud-event-proxy** delivers the event from the Kubernetes infrastructure to Cloud-Native Network Functions (CNFs) with low latency.

### **3** Event is published

The **cloud-event-proxy** sidecar in the PTP Operator-managed pod processes the event and publishes the event by using the PTP events REST API v2.

### **4** Consumer application requests a subscription and receives the subscribed event

The consumer application sends an API request to the producer **cloud-event-proxy** sidecar to create a PTP events subscription. Once subscribed, the consumer application listens to the address specified in the resource qualifier and receives and processes the PTP events.

#### 14.3.3. Configuring the PTP fast event notifications publisher

To start using PTP fast event notifications for a network interface in your cluster, you must enable the fast event publisher in the PTP Operator **PtpOperatorConfig** custom resource (CR) and configure **ntpClockThreshold** values in a **PtpConfig** CR that you create.

## Prerequisites

- You have installed the OpenShift Container Platform CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the PTP Operator.

## Procedure

1. Modify the default PTP Operator config to enable PTP fast events.

- a. Save the following YAML in the **ntp-operatorconfig.yaml** file:

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    apiVersion: "2.0" 1
    enableEventPublisher: true 2
```

**1** Enable the PTP events REST API v2 for the PTP event producer by setting the **ntpEventConfig.apiVersion** to "2.0". The default value is "1.0".

**2** Enable PTP fast event notifications by setting **enableEventPublisher** to **true**.



### NOTE

In OpenShift Container Platform 4.13 or later, you do not need to set the **spec.ptpEventConfig.transportHost** field in the **PtpOperatorConfig** resource when you use HTTP transport for PTP events.

- b. Update the **PtpOperatorConfig** CR:

```
$ oc apply -f ntp-operatorconfig.yaml
```

2. Create a **PtpConfig** custom resource (CR) for the PTP enabled interface, and set the required values for **ntpClockThreshold** and **ntp4IOpts**. The following YAML illustrates the required values that you must set in the **PtpConfig** CR:

```
spec:
  profile:
    - name: "profile1"
      interface: "enp5s0f0"
```

```

ptp4lOpts: "-2 -s --summary_interval -4" 1
phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
ptp4lConf: "" 3
ptpClockThreshold: 4
  holdOverTimeout: 5
  maxOffsetThreshold: 100
  minOffsetThreshold: -100

```

- 1** Append **--summary\_interval -4** to use PTP fast events.
- 2** Required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.
- 3** Specify a string that contains the configuration to replace the default **/etc/ptp4l.conf** file. To use the default configuration, leave the field empty.
- 4** Optional. If the **ptpClockThreshold** stanza is not present, default values are used for the **ptpClockThreshold** fields. The stanza shows default **ptpClockThreshold** values. The **ptpClockThreshold** values configure how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK\_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.

## Additional resources

- For a complete example CR that configures **linuxptp** services as an ordinary clock with PTP fast events, see [Configuring linuxptp services as ordinary clock](#).

### 14.3.4. PTP events REST API v2 consumer application reference

PTP event consumer applications require the following features:

1. A web service running with a **POST** handler to receive the cloud native PTP events JSON payload
2. A **createSubscription** function to subscribe to the PTP events producer
3. A **getCurrentState** function to poll the current state of the PTP events producer

The following example Go snippets illustrate these requirements:

#### Example PTP events consumer server function in Go

```

func server() {
    http.HandleFunc("/event", getEvent)
    http.ListenAndServe(":9043", nil)
}

func getEvent(w http.ResponseWriter, req *http.Request) {
    defer req.Body.Close()
}

```

```

bodyBytes, err := io.ReadAll(req.Body)
if err != nil {
    log.Errorf("error reading event %v", err)
}
e := string(bodyBytes)
if e != "" {
    processEvent(bodyBytes)
    log.Infof("received event %s", string(bodyBytes))
}
w.WriteHeader(http.StatusNoContent)
}

```

### Example PTP events createSubscription function in Go

```

import (
"github.com/redhat-cne/sdk-go/pkg/pubsub"
"github.com/redhat-cne/sdk-go/pkg/types"
v1pubsub "github.com/redhat-cne/sdk-go/v1/pubsub"
)

// Subscribe to PTP events using v2 REST API
s1,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/sync-state")
s2,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/lock-state")
s3,_:=createsubscription("/cluster/node/<node_name>/sync/gnss-status/gnss-sync-status")
s4,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state")
s5,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/clock-class")

// Create PTP event subscriptions POST
func createSubscription(resourceAddress string) (sub pubsub.PubSub, err error) {
    var status int
    apiPath := "/api/ocloudNotifications/v2/"
    localAPIAddr := "consumer-events-subscription-service.cloud-events.svc.cluster.local:9043" // vDU
    service API address
    apiAddr := "ptp-event-publisher-service-<node_name>.openshift-ptp.svc.cluster.local:9043" ①
    apiVersion := "2.0"

    subURL := &types.URI{URL: url.URL{Scheme: "http",
        Host: apiAddr
        Path: fmt.Sprintf("%s%s", apiPath, "subscriptions")}}
    endpointURL := &types.URI{URL: url.URL{Scheme: "http",
        Host: localAPIAddr,
        Path: "event"}}

    sub = v1pubsub.NewPubSub(endpointURL, resourceAddress, apiVersion)
    var subB []byte

    if subB, err = json.Marshal(&sub); err == nil {
        rc := restclient.New()
        if status, subB = rc.PostWithReturn(subURL, subB); status != http.StatusCreated {
            err = fmt.Errorf("error in subscription creation api at %s, returned status %d", subURL, status)
        } else {
            err = json.Unmarshal(subB, &sub)
        }
    } else {
        err = fmt.Errorf("failed to marshal subscription for %s", resourceAddress)
    }
}

```

```

    }
    return
}

```

- 1 Replace <node\_name> with the FQDN of the node that is generating the PTP events. For example, **compute-1.example.com**.

### Example PTP events consumer getCurrentState function in Go

```

//Get PTP event state for the resource
func getCurrentState(resource string) {
    //Create publisher
    url := &types.URI{URL: url.URL{Scheme: "http",
        Host: "ptp-event-publisher-service-<node_name>.openshift-ptp.svc.cluster.local:9043", 1
        Path: fmt.Sprintf("/api/ocloudNotifications/v2/%s/CurrentState",resource)}
    rc := restclient.New()
    status, event := rc.Get(url)
    if status != http.StatusOK {
        log.Errorf("CurrentState:error %d from url %s, %s", status, url.String(), event)
    } else {
        log.Debugf("Got CurrentState: %s ", event)
    }
}

```

- 1 Replace <node\_name> with the FQDN of the node that is generating the PTP events. For example, **compute-1.example.com**.

### 14.3.5. Reference event consumer deployment and service CRs using PTP events REST API v2

Use the following example PTP event consumer custom resources (CRs) as a reference when deploying your PTP events consumer application for use with the PTP events REST API v2.

#### Reference cloud event consumer namespace

```

apiVersion: v1
kind: Namespace
metadata:
  name: cloud-events
  labels:
    security.openshift.io/scc.podSecurityLabelSync: "false"
    pod-security.kubernetes.io/audit: "privileged"
    pod-security.kubernetes.io/enforce: "privileged"
    pod-security.kubernetes.io/warn: "privileged"
  name: cloud-events
  openshift.io/cluster-monitoring: "true"
  annotations:
    workload.openshift.io/allowed: management

```

#### Reference cloud event consumer deployment

```

apiVersion: apps/v1

```

```

kind: Deployment
metadata:
  name: cloud-consumer-deployment
  namespace: cloud-events
  labels:
    app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      annotations:
        target.workload.openshift.io/management: '{"effect": "PreferredDuringScheduling"}'
      labels:
        app: consumer
    spec:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      serviceAccountName: consumer-sa
      containers:
        - name: cloud-event-consumer
          image: cloud-event-consumer
          imagePullPolicy: Always
          args:
            - "--local-api-addr=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
            - "--api-path=/api/ocloudNotifications/v2/"
            - "--api-addr=127.0.0.1:8089"
            - "--api-version=2.0"
            - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043"
      env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: CONSUMER_TYPE
          value: "PTP"
        - name: ENABLE_STATUS_CHECK
          value: "true"
      volumes:
        - name: pubsubstore
          emptyDir: {}

```

## Reference cloud event consumer service account

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: consumer-sa
  namespace: cloud-events

```

## Reference cloud event consumer service

```

apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  sessionAffinity: None
  type: ClusterIP

```

#### 14.3.6. Subscribing to PTP events with the REST API v2

Deploy your **cloud-event-consumer** application container and subscribe the **cloud-event-consumer** application to PTP events posted by the **cloud-event-proxy** container in the pod managed by the PTP Operator.

Subscribe consumer applications to PTP events by sending a **POST** request to **[http://ptp-event-publisher-service-NODE\\_NAME.openshift-  
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions](http://ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions)** passing the appropriate subscription request payload.



#### NOTE

**9043** is the default port for the **cloud-event-proxy** container deployed in the PTP event producer pod. You can configure a different port for your application as required.

#### Additional resources

- [api/ocloudNotifications/v2/subscriptions](#)

#### 14.3.7. Verifying that the PTP events REST API v2 consumer application is receiving events

Verify that the **cloud-event-consumer** container in the application pod is receiving Precision Time Protocol (PTP) events.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed and configured the PTP Operator.
- You have deployed a cloud events application pod and PTP events consumer application.

## Procedure

1. Check the logs for the deployed events consumer application. For example, run the following command:

```
$ oc -n cloud-events logs -f deployment/cloud-consumer-deployment
```

### Example output

```
time = "2024-09-02T13:49:01Z"
level = info msg = "transport host path is set to ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043"
time = "2024-09-02T13:49:01Z"
level = info msg = "apiVersion=2.0, updated apiAddr=ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043, apiPath=/api/ocloudNotifications/v2/"
time = "2024-09-02T13:49:01Z"
level = info msg = "Starting local API listening to :9043"
time = "2024-09-02T13:49:06Z"
level = info msg = "transport host path is set to ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043"
time = "2024-09-02T13:49:06Z"
level = info msg = "checking for rest service health"
time = "2024-09-02T13:49:06Z"
level = info msg = "health check http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/health"
time = "2024-09-02T13:49:07Z"
level = info msg = "rest service returned healthy status"
time = "2024-09-02T13:49:07Z"
level = info msg = "healthy publisher; subscribing to events"
time = "2024-09-02T13:49:07Z"
level = info msg = "received event {"specversion":"1.0","id":"ab423275-f65d-4760-97af-5b0b846605e4","source":"/sync/ptp-status/clock-class","type":"event.sync.ptp-status.ptp-clock-class-change","time":"2024-09-02T13:49:07.226494483Z","data":{"version":1.0,"values":[{"ResourceAddress":"/cluster/node/compute-1.example.com/ptp-not-set","data_type":"metric","value_type":"decimal64.3","value":0}]}"
```

2. Optional. Test the REST API by using **oc** and port-forwarding port **9043** from the **linuxptp-daemon** deployment. For example, run the following command:

```
$ oc port-forward -n openshift-ptp ds/linuxptp-daemon 9043:9043
```

### Example output

```
Forwarding from 127.0.0.1:9043 -> 9043
Forwarding from [::1]:9043 -> 9043
Handling connection for 9043
```

Open a new shell prompt and test the REST API v2 endpoints:

```
$ curl -X GET http://localhost:9043/api/ocloudNotifications/v2/health
```

### Example output

OK

### 14.3.8. Monitoring PTP fast event metrics

You can monitor PTP fast events metrics from cluster nodes where the **linuxptp-daemon** is running. You can also monitor PTP fast event metrics in the OpenShift Container Platform web console by using the preconfigured and self-updating Prometheus monitoring stack.

#### Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in as a user with **cluster-admin** privileges.
- Install and configure the PTP Operator on a node with PTP-capable hardware.

#### Procedure

1. Start a debug pod for the node by running the following command:

```
$ oc debug node/<node_name>
```

2. Check for PTP metrics exposed by the **linuxptp-daemon** container. For example, run the following command:

```
sh-4.4# curl http://localhost:9091/metrics
```

#### Example output

```
# HELP cne_api_events_published Metric to get number of events published by the rest api
# TYPE cne_api_events_published gauge
cne_api_events_published{address="/cluster/node/compute-1.example.com-sync/gnss-
status/gnss-sync-status",status="success"} 1
cne_api_events_published{address="/cluster/node/compute-1.example.com-sync/ptp-
status/lock-state",status="success"} 94
cne_api_events_published{address="/cluster/node/compute-1.example.com-sync/ptp-
status/class-change",status="success"} 18
cne_api_events_published{address="/cluster/node/compute-1.example.com-sync/sync-
status/os-clock-sync-state",status="success"} 27
```

3. Optional. You can also find PTP events in the logs for the **cloud-event-proxy** container. For example, run the following command:

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

4. To view the PTP event in the OpenShift Container Platform web console, copy the name of the PTP metric you want to query, for example, **openshift\_ptp\_offset\_ns**.
5. In the OpenShift Container Platform web console, click **Observe** → **Metrics**.
6. Paste the PTP metric name into the **Expression** field, and click **Run queries**.

#### Additional resources

- Accessing metrics as a developer

### 14.3.9. PTP fast event metrics reference

The following table describes the PTP fast events metrics that are available from cluster nodes where the **linuxptp-daemon** service is running.

Table 14.11. PTP fast event metrics

Metric	Description	Example
<b>openshift_ptp_clock_class</b>	Returns the PTP clock class for the interface. Possible values for PTP clock class are 6 ( <b>LOCKED</b> ), 7 ( <b>PRC UNLOCKED IN-SPEC</b> ), 52 ( <b>PRC UNLOCKED OUT-OF-SPEC</b> ), 187 ( <b>PRC UNLOCKED OUT-OF-SPEC</b> ), 135 ( <b>T-BC HOLDOVER IN-SPEC</b> ), 165 ( <b>T-BC HOLDOVER OUT-OF-SPEC</b> ), 248 ( <b>DEFAULT</b> ), or 255 ( <b>SLAVE ONLY CLOCK</b> ).	{node="compute-1.example.com",process="ptp4l"} 6
<b>openshift_ptp_clock_state</b>	Returns the current PTP clock state for the interface. Possible values for PTP clock state are <b>FREERUN</b> , <b>LOCKED</b> , or <b>HOLDOVER</b> .	{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1
<b>openshift_ptp_delay_ns</b>	Returns the delay in nanoseconds between the primary clock sending the timing packet and the secondary clock receiving the timing packet.	{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0
<b>openshift_ptp_ha_profile_status</b>	Returns the current status of the highly available system clock when there are multiple time sources on different NICs. Possible values are 0 ( <b>INACTIVE</b> ) and 1 ( <b>ACTIVE</b> ).	{node="node1",process="phc2sys",profile="profile1"} 1{node="node1",process="phc2sys",profile="profile2"} 0
<b>openshift_ptp_frequency_adjustment_ns</b>	Returns the frequency adjustment in nanoseconds between 2 PTP clocks. For example, between the upstream clock and the NIC, between the system clock and the NIC, or between the PTP hardware clock ( <b>phc</b> ) and the NIC.	{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -6768
<b>openshift_ptp_interface_role</b>	Returns the configured PTP clock role for the interface. Possible values are 0 ( <b>PASSIVE</b> ), 1 ( <b>SLAVE</b> ), 2 ( <b>MASTER</b> ), 3 ( <b>FAULTY</b> ), 4 ( <b>UNKNOWN</b> ), or 5 ( <b>LISTENING</b> ).	{iface="ens2f0", node="compute-1.example.com", process="ptp4l"} 2
<b>openshift_ptp_max_offset_ns</b>	Returns the maximum offset in nanoseconds between 2 clocks or interfaces. For example, between the upstream GNSS clock and the NIC ( <b>ts2phc</b> ), or between the PTP hardware clock ( <b>phc</b> ) and the system clock ( <b>phc2sys</b> ).	{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 1.038099569e+09

Metric	Description	Example
<b>openshift_ptp_offset_ns</b>	Returns the offset in nanoseconds between the DPLL clock or the GNSS clock source and the NIC hardware clock.	<code>{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -9</code>
<b>openshift_ptp_process_restart_count</b>	Returns a count of the number of times the <b>ptp4l</b> and <b>ts2phc</b> processes were restarted.	<code>{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1</code>
<b>openshift_ptp_process_status</b>	Returns a status code that shows whether the PTP processes are running or not.	<code>{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1</code>
<b>openshift_ptp_threshold</b>	Returns values for <b>HoldOverTimeout</b> , <b>MaxOffsetThreshold</b> , and <b>MinOffsetThreshold</b> . <ul style="list-style-type: none"> <li>• <b>holdOverTimeout</b> is the time value in seconds before the PTP clock event state changes to <b>FREERUN</b> when the PTP master clock is disconnected.</li> <li>• <b>maxOffsetThreshold</b> and <b>minOffsetThreshold</b> are offset values in nanoseconds that compare against the values for <b>CLOCK_REALTIME</b> (<b>phc2sys</b>) or master offset (<b>ptp4l</b>) values that you configure in the <b>PtpConfig</b> CR for the NIC.</li> </ul>	<code>{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5</code>

#### PTP fast event metrics only when T-GM is enabled

The following table describes the PTP fast event metrics that are available only when PTP grandmaster clock (T-GM) is enabled.

Table 14.12. PTP fast event metrics when T-GM is enabled

Metric	Description	Example
<b>openshift_ptp_frequency_status</b>	Returns the current status of the digital phase-locked loop (DPLL) frequency for the NIC. Possible values are -1 ( <b>UNKNOWN</b> ), 0 ( <b>INVALID</b> ), 1 ( <b>FREERUN</b> ), 2 ( <b>LOCKED</b> ), 3 ( <b>LOCKED_HO_ACQ</b> ), or 4 ( <b>HOLDOVER</b> ).	<code>{from="dpll",iface="ens2fx",node ="compute-1.example.com",process="dpll"} 3</code>

Metric	Description	Example
<b>openshift_ptp_nm ea_status</b>	Returns the current status of the NMEA connection. NMEA is the protocol that is used for 1PPS NIC connections. Possible values are 0 ( <b>UNAVAILABLE</b> ) and 1 ( <b>AVAILABLE</b> ).	{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1
<b>openshift_ptp_phase_status</b>	Returns the status of the DLL phase for the NIC. Possible values are -1 ( <b>UNKNOWN</b> ), 0 ( <b>INVALID</b> ), 1 ( <b>FREERUN</b> ), 2 ( <b>LOCKED</b> ), 3 ( <b>LOCKED_HO_ACQ</b> ), or 4 ( <b>HOLDOVER</b> ).	{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3
<b>openshift_ptp_pps_status</b>	Returns the current status of the NIC 1PPS connection. You use the 1PPS connection to synchronize timing between connected NICs. Possible values are 0 ( <b>UNAVAILABLE</b> ) and 1 ( <b>AVAILABLE</b> ).	{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1
<b>openshift_ptp_gnss_status</b>	Returns the current status of the global navigation satellite system (GNSS) connection. GNSS provides satellite-based positioning, navigation, and timing services globally. Possible values are 0 ( <b>NOFIX</b> ), 1 ( <b>DEAD RECKONING ONLY</b> ), 2 ( <b>2D-FIX</b> ), 3 ( <b>3D-FIX</b> ), 4 ( <b>GPS+DEAD RECKONING FIX</b> ), 5, ( <b>TIME ONLY FIX</b> ).	{from="gnss",iface="ens2fx",node="compute-1.example.com",process="gnss"} 3

## 14.4. PTP EVENTS REST API V2 REFERENCE

Use the following REST API v2 endpoints to subscribe the **cloud-event-consumer** application to Precision Time Protocol (PTP) events posted at <http://localhost:9043/api/ocloudNotifications/v2> in the PTP events producer pod.

- [api/ocloudNotifications/v2/subscriptions](#)
  - **POST**: Creates a new subscription
  - **GET**: Retrieves a list of subscriptions
  - **DELETE**: Deletes all subscriptions
- [api/ocloudNotifications/v2/subscriptions/{subscription\\_id}](#)
  - **GET**: Returns details for the specified subscription ID
  - **DELETE**: Deletes the subscription associated with the specified subscription ID
- [api/ocloudNotifications/v2/health](#)
  - **GET**: Returns the health status of **ocloudNotifications** API
- [api/ocloudNotifications/v2/publishers](#)
  - **GET**: Returns a list of PTP event publishers for the cluster node

- [api/ocloudnotifications/v2/{resource\\_address}/CurrentState](#)

- **GET:** Returns the current state of the event type specified by the `{resource_address}`.

## 14.4.1. PTP events REST API v2 endpoints

### 14.4.1.1. api/ocloudNotifications/v2/subscriptions

**HTTP method**

**GET** [api/ocloudNotifications/v2/subscriptions](#)

#### Description

Returns a list of subscriptions. If subscriptions exist, a **200 OK** status code is returned along with the list of subscriptions.

#### Example API response

```
[
{
  "ResourceAddress": "/cluster/node/compute-1.example.com-sync/status/os-clock-sync-state",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "SubscriptionId": "ccedbf08-3f96-4839-a0b6-2eb0401855ed",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/ccedbf08-3f96-4839-a0b6-
2eb0401855ed"
},
{
  "ResourceAddress": "/cluster/node/compute-1.example.com-sync/ptp-status/clock-class",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "SubscriptionId": "a939a656-1b7d-4071-8cf1-f99af6e931f2",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/a939a656-1b7d-4071-8cf1-
f99af6e931f2"
},
{
  "ResourceAddress": "/cluster/node/compute-1.example.com-sync/ptp-status/lock-state",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "SubscriptionId": "ba4564a3-4d9e-46c5-b118-591d3105473c",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/ba4564a3-4d9e-46c5-b118-
591d3105473c"
},
{
  "ResourceAddress": "/cluster/node/compute-1.example.com-sync/gnss-status/gnss-sync-status",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "SubscriptionId": "ea0d772e-f00a-4889-98be-51635559b4fb",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/ea0d772e-f00a-4889-98be-
51635559b4fb"
},
{
```

```

    "ResourceAddress": "/cluster/node/compute-1.example.com-sync/status/sync-state",
    "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
    "SubscriptionId": "762999bf-b4a0-4bad-abe8-66e646b65754",
    "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/762999bf-b4a0-4bad-abe8-
66e646b65754"
}
]

```

**HTTP method****POST api/ocloudNotifications/v2/subscriptions****Description**

Creates a new subscription for the required event by passing the appropriate payload.

You can subscribe to the following PTP events:

- **sync-state** events
- **lock-state** events
- **gnss-sync-status events** events
- **os-clock-sync-state** events
- **clock-class** events

**Table 14.13. Query parameters**

Parameter	Type
subscription	data

**Example sync-state subscription payload**

```
{
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "ResourceAddress": "/cluster/node/{node_name}/sync/status/sync-state"
}
```

**Example PTP lock-state events subscription payload**

```
{
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "ResourceAddress": "/cluster/node/{node_name}/sync/ptp-status/lock-state"
}
```

**Example PTP gnss-sync-status events subscription payload**

```
{
```

```

  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "ResourceAddress": "/cluster/node/{node_name}/sync/gnss-status/gnss-sync-status"
}

```

### Example PTP os-clock-sync-state events subscription payload

```

{
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "ResourceAddress": "/cluster/node/{node_name}/sync/sync-status/os-clock-sync-state"
}

```

### Example PTP clock-class events subscription payload

```

{
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "ResourceAddress": "/cluster/node/{node_name}/sync/ptp-status/clock-class"
}

```

### Example API response

```

{
  "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "SubscriptionId": "620283f3-26cd-4a6d-b80a-bdc4b614a96a",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/620283f3-26cd-4a6d-b80a-
bdc4b614a96a"
}

```

The following subscription status events are possible:

**Table 14.14. PTP events REST API v2 subscription status codes**

Status code	Description
<b>201 Created</b>	Indicates that the subscription is created
<b>400 Bad Request</b>	Indicates that the server could not process the request because it was malformed or invalid
<b>404 Not Found</b>	Indicates that the subscription resource is not available
<b>409 Conflict</b>	Indicates that the subscription already exists

#### HTTP method

**DELETE api/ocloudNotifications/v2/subscriptions**

#### Description

Deletes all subscriptions.

### Example API response

```
{
  "status": "deleted all subscriptions"
}
```

#### 14.4.1.2. `api/ocloudNotifications/v2/subscriptions/{subscription_id}`

HTTP method

**GET `api/ocloudNotifications/v2/subscriptions/{subscription_id}`**

#### Description

Returns details for the subscription with ID **subscription\_id**.

Table 14.15. Global path parameters

Parameter	Type
<b>subscription_id</b>	string

### Example API response

```
{
  "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
  "SubscriptionId": "620283f3-26cd-4a6d-b80a-bdc4b614a96a",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-
ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/620283f3-26cd-4a6d-b80a-
bdc4b614a96a"
}
```

HTTP method

**DELETE `api/ocloudNotifications/v2/subscriptions/{subscription_id}`**

#### Description

Deletes the subscription with ID **subscription\_id**.

Table 14.16. Global path parameters

Parameter	Type
<b>subscription_id</b>	string

Table 14.17. HTTP response codes

HTTP response	Description
204 No Content	Success

#### 14.4.1.3. api/ocloudNotifications/v2/health

**HTTP method**

**GET api/ocloudNotifications/v2/health/**

##### Description

Returns the health status for the **ocloudNotifications** REST API.

**Table 14.18. HTTP response codes**

HTTP response	Description
200 OK	Success

#### 14.4.1.4. api/ocloudNotifications/v2/publishers

**HTTP method**

**GET api/ocloudNotifications/v2/publishers**

##### Description

Returns a list of publisher details for the cluster node. The system generates notifications when the relevant equipment state changes.

You can use equipment synchronization status subscriptions together to deliver a detailed view of the overall synchronization health of the system.

##### Example API response

```
[
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com-sync-sync-status-sync-state",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "4ea72bfa-185c-4703-9694-cdd0434cd570",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/4ea72bfa-185c-4703-9694-cdd0434cd570"
  },
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com-sync-sync-status/os-clock-sync-state",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "71fbb38e-a65d-41fc-823b-d76407901087",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/71fbb38e-a65d-41fc-823b-d76407901087"
  },
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com-sync-ptp-status/clock-class",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "7bc27cad-03f4-44a9-8060-a029566e7926",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/7bc27cad-03f4-44a9-8060-a029566e7926"
  },
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com-sync-ptp-status/lock-state",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "6e7b6736-f359-46b9-991c-fbaed25eb554",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/6e7b6736-f359-46b9-991c-fbaed25eb554"
  }
]
```

```

991c-fbaed25eb554"
},
{
  "ResourceAddress": "/cluster/node/compute-1.example.com-sync/gnss-status/gnss-sync-status",
  "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
  "SubscriptionId": "31bb0a45-7892-45d4-91dd-13035b13ed18",
  "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/31bb0a45-7892-45d4-
91dd-13035b13ed18"
}
]

```

Table 14.19. HTTP response codes

HTTP response	Description
200 OK	Success

#### 14.4.1.5. api/ocloudNotifications/v2/{resource\_address}/CurrentState

##### HTTP method

**GET api/ocloudNotifications/v2/cluster/node/{node\_name}/sync/ptp-status/lock-state/CurrentState**

**GET api/ocloudNotifications/v2/cluster/node/{node\_name}/sync/sync-status/os-clock-sync-state/CurrentState**

**GET api/ocloudNotifications/v2/cluster/node/{node\_name}/sync/ptp-status/clock-class/CurrentState**

**GET api/ocloudNotifications/v2/cluster/node/{node\_name}/sync/sync-status/sync-state/CurrentState**

**GET api/ocloudNotifications/v2/cluster/node/{node\_name}/sync/gnss-status/gnss-sync-state/CurrentState**

##### Description

Returns the current state of the **os-clock-sync-state**, **clock-class**, **lock-state**, **gnss-sync-status**, or **sync-state** events for the cluster node.

- **os-clock-sync-state** notifications describe the host operating system clock synchronization state. Can be in **LOCKED** or **FREERUN** state.
- **clock-class** notifications describe the current state of the PTP clock class.
- **lock-state** notifications describe the current status of the PTP equipment lock state. Can be in **LOCKED**, **HOLDOVER** or **FREERUN** state.
- **sync-state** notifications describe the current status of the least synchronized of the PTP clock **lock-state** and **os-clock-sync-state** states.
- **gnss-sync-status** notifications describe the GNSS clock synchronization state.

Table 14.20. Global path parameters

Parameter	Type
<b>resource_address</b>	string

### Example lock-state API response

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com-sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

### Example os-clock-sync-state API response

```
{
  "specversion": "0.3",
  "id": "4f51fe99-faea-4e66-9112-66c5c9b9afcb",
  "source": "/cluster/node/compute-1.example.com-sync/sync-status/os-clock-sync-state",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "subject": "/cluster/node/compute-1.example.com-sync/sync-status/os-clock-sync-state",
  "datacontenttype": "application/json",
  "time": "2022-11-29T17:44:22.202Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

```

        "valueType": "decimal64.3",
        "value": "27"
    }
]
}
}
```

### Example clock-class API response

```
{
  "id": "064c9e67-5ad4-4afb-98ff-189c6aa9c205",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/node/compute-1.example.com-sync/ptp-status/clock-class",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
```

### Example sync-state API response

```
{
  "specversion": "0.3",
  "id": "8c9d6ecb-ae9f-4106-82c4-0a778a79838d",
  "source": "/sync/sync-status/sync-state",
  "type": "event.sync.sync-status.synchronization-state-change",
  "subject": "/cluster/node/compute-1.example.com-sync/sync-status/sync-state",
  "datacontenttype": "application/json",
  "time": "2024-08-28T14:50:57.327585316Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com-sync/sync-status/sync-state",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      }
    ]
  }
}
```

### Example gnss-sync-state API response

```
{
```

```

"id": "435e1f2a-6854-4555-8520-767325c087d7",
"type": "event.sync.gnss-status.gnss-state-change",
"source": "/cluster/node/compute-1.example.com-sync/gnss-status/gnss-sync-status",
"dataContentType": "application/json",
"time": "2023-09-27T19:35:33.42347206Z",
"data": {
  "version": "1.0",
  "values": [
    {
      "resource": "/cluster/node/compute-1.example.com/ens2fx/master",
      "dataType": "notification",
      "valueType": "enumeration",
      "value": "SYNCHRONIZED"
    },
    {
      "resource": "/cluster/node/compute-1.example.com/ens2fx/master",
      "dataType": "metric",
      "valueType": "decimal64.3",
      "value": "5"
    }
  ]
}

```

## 14.5. DEVELOPING PTP EVENTS CONSUMER APPLICATIONS WITH THE REST API V1

When developing consumer applications that make use of Precision Time Protocol (PTP) events on a bare-metal cluster node, you deploy your consumer application in a separate application pod. The consumer application subscribes to PTP events by using the PTP events REST API v1.



### NOTE

The following information provides general guidance for developing consumer applications that use PTP events. A complete events consumer application example is outside the scope of this information.



### IMPORTANT

PTP events REST API v1 and events consumer application sidecar is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

### Additional resources

- [PTP events REST API v1 reference](#)

#### 14.5.1. About the PTP fast event notifications framework

Use the Precision Time Protocol (PTP) fast event REST API v2 to subscribe cluster applications to PTP events that the bare-metal cluster node generates.



## NOTE

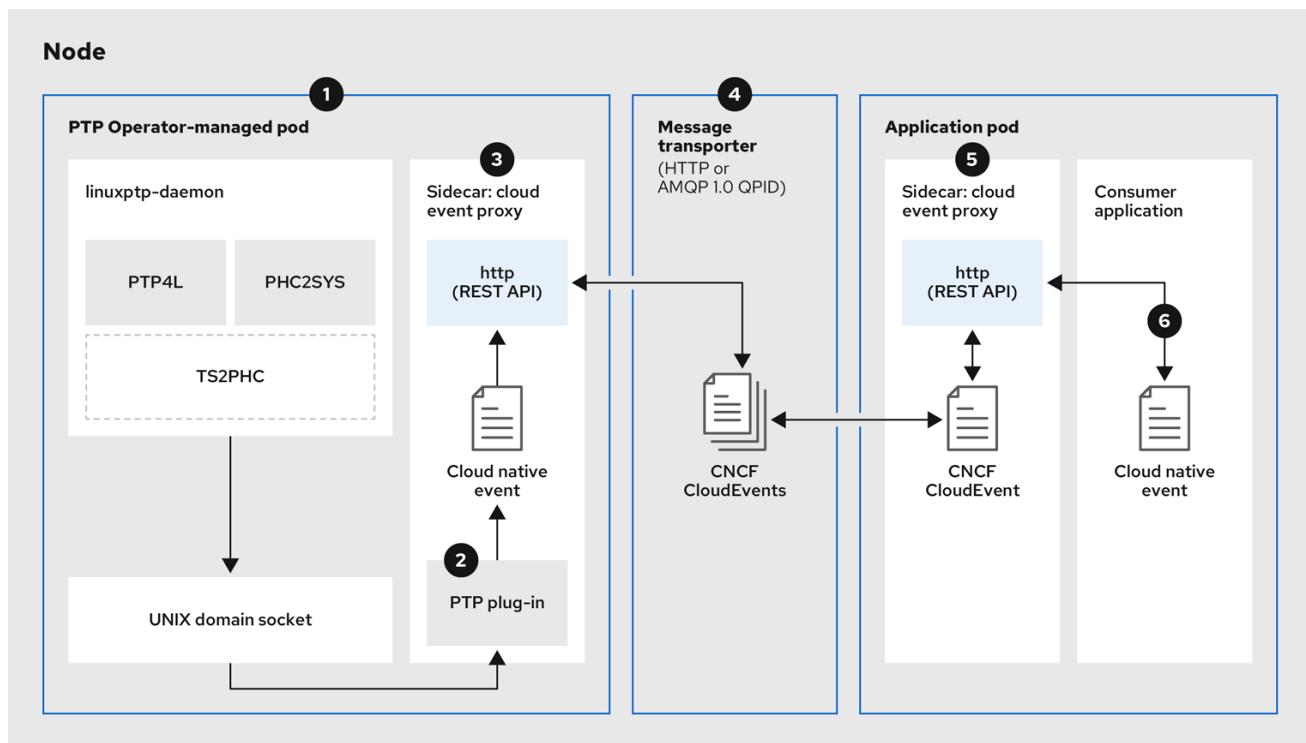
The fast events notifications framework uses a REST API for communication. The PTP events REST API v1 and v2 are based on the *O-RAN O-Cloud Notification API Specification for Event Consumers 4.0* that is available from [O-RAN ALLIANCE Specifications](#).

Only the PTP events REST API v2 is O-RAN v4 compliant.

### 14.5.2. Retrieving PTP events with the PTP events REST API v1

Applications run the **cloud-event-proxy** container in a sidecar pattern to subscribe to PTP events. The **cloud-event-proxy** sidecar container can access the same resources as the primary application container without using any of the resources of the primary application and with no significant latency.

Figure 14.7. Overview of PTP fast events with consumer sidecar and HTTP message transport



319\_OpenShift\_0323

#### 1 Event is generated on the cluster host

**linuxptp-daemon** in the PTP Operator-managed pod runs as a Kubernetes **DaemonSet** and manages the various **linuxptp** processes (**ptp4l**, **phc2sys**, and optionally for grandmaster clocks, **ts2phc**). The **linuxptp-daemon** passes the event to the UNIX domain socket.

#### 2 Event is passed to the cloud-event-proxy sidecar

The PTP plugin reads the event from the UNIX domain socket and passes it to the **cloud-event-proxy** sidecar in the PTP Operator-managed pod. **cloud-event-proxy** delivers the event from the Kubernetes infrastructure to Cloud-Native Network Functions (CNFs) with low latency.

### 3 Event is persisted

The **cloud-event-proxy** sidecar in the PTP Operator-managed pod processes the event and publishes the cloud-native event by using a REST API.

### 4 Message is transported

The message transporter transports the event to the **cloud-event-proxy** sidecar in the application pod over HTTP.

### 5 Event is available from the REST API

The **cloud-event-proxy** sidecar in the Application pod processes the event and makes it available by using the REST API.

### 6 Consumer application requests a subscription and receives the subscribed event

The consumer application sends an API request to the **cloud-event-proxy** sidecar in the application pod to create a PTP events subscription. The **cloud-event-proxy** sidecar creates an HTTP messaging listener protocol for the resource specified in the subscription.

The **cloud-event-proxy** sidecar in the application pod receives the event from the PTP Operator-managed pod, unwraps the cloud events object to retrieve the data, and posts the event to the consumer application. The consumer application listens to the address specified in the resource qualifier and receives and processes the PTP event.

#### 14.5.3. Configuring the PTP fast event notifications publisher

To start using PTP fast event notifications for a network interface in your cluster, you must enable the fast event publisher in the PTP Operator **PtpOperatorConfig** custom resource (CR) and configure **ptpClockThreshold** values in a **PtpConfig** CR that you create.

##### Prerequisites

- You have installed the OpenShift Container Platform CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the PTP Operator.

##### Procedure

1. Modify the default PTP Operator config to enable PTP fast events.
  - a. Save the following YAML in the **ptp-operatorconfig.yaml** file:

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    enableEventPublisher: true ①
```

- 1 Enable PTP fast event notifications by setting `enableEventPublisher` to `true`.



### NOTE

In OpenShift Container Platform 4.13 or later, you do not need to set the `spec.ptpEventConfig.transportHost` field in the `PtpOperatorConfig` resource when you use HTTP transport for PTP events.

- b. Update the `PtpOperatorConfig` CR:

```
$ oc apply -f ptp-operatorconfig.yaml
```

- 2 Create a `PtpConfig` custom resource (CR) for the PTP enabled interface, and set the required values for `ptpClockThreshold` and `ptp4IOpts`. The following YAML illustrates the required values that you must set in the `PtpConfig` CR:

```
spec:
profile:
- name: "profile1"
interface: "enp5s0f0"
ptp4IOpts: "-2 -s --summary_interval -4" ①
phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ②
ptp4ICnf: "" ③
ptpClockThreshold: ④
holdOverTimeout: 5
maxOffsetThreshold: 100
minOffsetThreshold: -100
```

- ① Append `--summary_interval -4` to use PTP fast events.
- ② Required `phc2sysOpts` values. `-m` prints messages to `stdout`. The `linuxptp-daemon DaemonSet` parses the logs and generates Prometheus metrics.
- ③ Specify a string that contains the configuration to replace the default `/etc/ptp4l.conf` file. To use the default configuration, leave the field empty.
- ④ Optional. If the `ptpClockThreshold` stanza is not present, default values are used for the `ptpClockThreshold` fields. The stanza shows default `ptpClockThreshold` values. The `ptpClockThreshold` values configure how long after the PTP master clock is disconnected before PTP events are triggered. `holdOverTimeout` is the time value in seconds before the PTP clock event state changes to `FREERUN` when the PTP master clock is disconnected. The `maxOffsetThreshold` and `minOffsetThreshold` settings configure offset values in nanoseconds that compare against the values for `CLOCK_REALTIME` (`phc2sys`) or master offset (`ptp4l`). When the `ptp4l` or `phc2sys` offset value is outside this range, the PTP clock state is set to `FREERUN`. When the offset value is within this range, the PTP clock state is set to `LOCKED`.

### Additional resources

- For a complete example CR that configures `linuxptp` services as an ordinary clock with PTP fast events, see [Configuring linuxptp services as ordinary clock](#).

#### 14.5.4. PTP events consumer application reference

PTP event consumer applications require the following features:

1. A web service running with a **POST** handler to receive the cloud native PTP events JSON payload
2. A **createSubscription** function to subscribe to the PTP events producer
3. A **getCurrentState** function to poll the current state of the PTP events producer

The following example Go snippets illustrate these requirements:

##### Example PTP events consumer server function in Go

```
func server() {
    http.HandleFunc("/event", getEvent)
    http.ListenAndServe("localhost:8989", nil)
}

func getEvent(w http.ResponseWriter, req *http.Request) {
    defer req.Body.Close()
    bodyBytes, err := io.ReadAll(req.Body)
    if err != nil {
        log.Errorf("error reading event %v", err)
    }
    e := string(bodyBytes)
    if e != "" {
        processEvent(bodyBytes)
        log.Infof("received event %s", string(bodyBytes))
    } else {
        w.WriteHeader(http.StatusNoContent)
    }
}
```

##### Example PTP events createSubscription function in Go

```
import (
    "github.com/redhat-cne/sdk-go/pkg/pubsub"
    "github.com/redhat-cne/sdk-go/pkg/types"
    v1pubsub "github.com/redhat-cne/sdk-go/v1/pubsub"
)

// Subscribe to PTP events using REST API
s1,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state") ①
s2,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/class-change")
s3,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/lock-state")

// Create PTP event subscriptions POST
func createSubscription(resourceAddress string) (sub pubsub.PubSub, err error) {
    var status int
    apiPath:= "/api/ocloudNotifications/v1/"
    localAPIAddr:=localhost:8989 // vDU service API address
    apiAddr:= "localhost:8089" // event framework API address
```

```

subURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: apiAddr
    Path: fmt.Sprintf("%s%s", apiPath, "subscriptions")}}
endpointURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: localAPIAddr,
    Path: "event"}}
sub = v1pubsub.NewPubSub(endpointURL, resourceAddress)
var subB []byte

if subB, err = json.Marshal(&sub); err == nil {
    rc := restclient.New()
    if status, subB = rc.PostWithReturn(subURL, subB); status != http.StatusCreated {
        err = fmt.Errorf("error in subscription creation api at %s, returned status %d", subURL, status)
    } else {
        err = json.Unmarshal(subB, &sub)
    }
} else {
    err = fmt.Errorf("failed to marshal subscription for %s", resourceAddress)
}
return
}

```

- 1 Replace <node\_name> with the FQDN of the node that is generating the PTP events. For example, **compute-1.example.com**.

### Example PTP events consumer getCurrentState function in Go

```

//Get PTP event state for the resource
func getCurrentState(resource string) {
    //Create publisher
    url := &types.URI{URL: url.URL{Scheme: "http",
        Host: localhost:8989,
        Path: fmt.Sprintf("/api/ocloudNotifications/v1/%s/CurrentState",resource})}
    rc := restclient.New()
    status, event := rc.Get(url)
    if status != http.StatusOK {
        log.Errorf("GetCurrentState:error %d from url %s, %s", status, url.String(), event)
    } else {
        log.Debugf("Got CurrentState: %s ", event)
    }
}

```

#### 14.5.5. Reference cloud-event-proxy deployment and service CRs

Use the following example **cloud-event-proxy** deployment and subscriber service CRs as a reference when deploying your PTP events consumer application.

##### Reference cloud-event-proxy deployment with HTTP transport

```

apiVersion: apps/v1
kind: Deployment
metadata:
    name: event-consumer-deployment

```

```

namespace: <namespace>
labels:
  app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
    spec:
      serviceAccountName: sidecar-consumer-sa
      containers:
        - name: event-subscriber
          image: event-subscriber-app
        - name: cloud-event-proxy-as-sidecar
          image: openshift4/ose-cloud-event-proxy
      args:
        - "--metrics-addr=127.0.0.1:9091"
        - "--store-path=/store"
        - "--transport-host=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
        - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043"
        - "--api-port=8089"
      env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: NODE_IP
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
      volumeMounts:
        - name: pubsubstore
          mountPath: /store
      ports:
        - name: metrics-port
          containerPort: 9091
        - name: sub-port
          containerPort: 9043
      volumes:
        - name: pubsubstore
          emptyDir: {}

```

## Reference cloud-event-proxy subscriber service

```

apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service

```

```

namespace: cloud-events
labels:
  app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP

```

#### 14.5.6. Subscribing to PTP events with the REST API v1

Deploy your **cloud-event-consumer** application container and **cloud-event-proxy** sidecar container in a separate application pod.

Subscribe the **cloud-event-consumer** application to PTP events posted by the **cloud-event-proxy** container at **<http://localhost:8089/api/ocloudNotifications/v1/>** in the application pod.



#### NOTE

**9089** is the default port for the **cloud-event-consumer** container deployed in the application pod. You can configure a different port for your application as required.

#### Additional resources

- [api/ocloudNotifications/v1/subscriptions](#)

#### 14.5.7. Verifying that the PTP events REST API v1 consumer application is receiving events

Verify that the **cloud-event-proxy** container in the application pod is receiving PTP events.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed and configured the PTP Operator.

#### Procedure

1. Get the list of active **linuxptp-daemon** pods. Run the following command:

```
$ oc get pods -n openshift-ptp
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
linuxptp-daemon-2t78p	3/3	Running	0	8h
linuxptp-daemon-k8n88	3/3	Running	0	8h

2. Access the metrics for the required consumer-side **cloud-event-proxy** container by running the following command:

```
$ oc exec -it <linuxptp-daemon> -n openshift-ptp -c cloud-event-proxy -- curl  
127.0.0.1:9091/metrics
```

where:

<linuxptp-daemon>

Specifies the pod you want to query, for example, **linuxptp-daemon-2t78p**.

### Example output

```
# HELP cne_transport_connections_resets Metric to get number of connection resets
# TYPE cne_transport_connections_resets gauge
cne_transport_connection_reset 1
# HELP cne_transport_receiver Metric to get number of receiver created
# TYPE cne_transport_receiver gauge
cne_transport_receiver{address="/cluster/node/compute-1.example.com/ptp",status="active"} 2
cne_transport_receiver{address="/cluster/node/compute-1.example.com/redfish/event",status="active"} 2
# HELP cne_transport_sender Metric to get number of sender created
# TYPE cne_transport_sender gauge
cne_transport_sender{address="/cluster/node/compute-1.example.com/ptp",status="active"} 1
cne_transport_sender{address="/cluster/node/compute-1.example.com/redfish/event",status="active"} 1
# HELP cne_events_ack Metric to get number of events produced
# TYPE cne_events_ack gauge
cne_events_ack{status="success",type="/cluster/node/compute-1.example.com/ptp"} 18
cne_events_ack{status="success",type="/cluster/node/compute-1.example.com/redfish/event"} 18
# HELP cne_events_transport_published Metric to get number of events published by the transport
# TYPE cne_events_transport_published gauge
cne_events_transport_published{address="/cluster/node/compute-1.example.com/ptp",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-1.example.com/ptp",status="success"} 18
cne_events_transport_published{address="/cluster/node/compute-1.example.com/redfish/event",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-1.example.com/redfish/event",status="success"} 18
# HELP cne_events_transport_received Metric to get number of events received by the transport
# TYPE cne_events_transport_received gauge
cne_events_transport_received{address="/cluster/node/compute-1.example.com/ptp",status="success"} 18
cne_events_transport_received{address="/cluster/node/compute-1.example.com/redfish/event",status="success"} 18
```

```

# HELP cne_events_api_published Metric to get number of events published by the rest
api
# TYPE cne_events_api_published gauge
cne_events_api_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 19
cne_events_api_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 19
# HELP cne_events_received Metric to get number of events received
# TYPE cne_events_received gauge
cne_events_received{status="success",type="/cluster/node/compute-1.example.com/ptp"} 18
cne_events_received{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being
served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status
code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 4
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0

```

#### 14.5.8. Monitoring PTP fast event metrics

You can monitor PTP fast events metrics from cluster nodes where the **linuxptp-daemon** is running. You can also monitor PTP fast event metrics in the OpenShift Container Platform web console by using the preconfigured and self-updating Prometheus monitoring stack.

##### Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in as a user with **cluster-admin** privileges.
- Install and configure the PTP Operator on a node with PTP-capable hardware.

##### Procedure

1. Start a debug pod for the node by running the following command:

```
$ oc debug node/<node_name>
```

2. Check for PTP metrics exposed by the **linuxptp-daemon** container. For example, run the following command:

```
sh-4.4# curl http://localhost:9091/metrics
```

##### Example output

```

# HELP cne_api_events_published Metric to get number of events published by the rest api
# TYPE cne_api_events_published gauge

```

```
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",status="success"} 1
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",status="success"} 94
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/class-change",status="success"} 18
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",status="success"} 27
```

3. Optional. You can also find PTP events in the logs for the **cloud-event-proxy** container. For example, run the following command:

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

4. To view the PTP event in the OpenShift Container Platform web console, copy the name of the PTP metric you want to query, for example, **openshift\_ptp\_offset\_ns**.
5. In the OpenShift Container Platform web console, click **Observe** → **Metrics**.
6. Paste the PTP metric name into the **Expression** field, and click **Run queries**.

#### Additional resources

- [Accessing metrics as a developer](#)

#### 14.5.9. PTP fast event metrics reference

The following table describes the PTP fast events metrics that are available from cluster nodes where the **linuxptp-daemon** service is running.

**Table 14.21. PTP fast event metrics**

Metric	Description	Example
<b>openshift_ptp_clock_class</b>	Returns the PTP clock class for the interface. Possible values for PTP clock class are 6 ( <b>LOCKED</b> ), 7 ( <b>PRC UNLOCKED IN-SPEC</b> ), 52 ( <b>PRC UNLOCKED OUT-OF-SPEC</b> ), 187 ( <b>PRC UNLOCKED OUT-OF-SPEC</b> ), 135 ( <b>T-BC HOLDOVER IN-SPEC</b> ), 165 ( <b>T-BC HOLDOVER OUT-OF-SPEC</b> ), 248 ( <b>DEFAULT</b> ), or 255 ( <b>SLAVE ONLY CLOCK</b> ).	{node="compute-1.example.com",process="ptp4l"} 6
<b>openshift_ptp_clock_state</b>	Returns the current PTP clock state for the interface. Possible values for PTP clock state are <b>FREERUN</b> , <b>LOCKED</b> , or <b>HOLDOVER</b> .	{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1
<b>openshift_ptp_delay_ns</b>	Returns the delay in nanoseconds between the primary clock sending the timing packet and the secondary clock receiving the timing packet.	{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0

Metric	Description	Example
<code>openshift_ptp_ha_profile_status</code>	Returns the current status of the highly available system clock when there are multiple time sources on different NICs. Possible values are 0 ( <b>INACTIVE</b> ) and 1 ( <b>ACTIVE</b> ).	<code>{node="node1",process="phc2sys",profile="profile1"} 1{node="node1",process="phc2sys",profile="profile2"} 0</code>
<code>openshift_ptp_frequency_adjustment_ns</code>	Returns the frequency adjustment in nanoseconds between 2 PTP clocks. For example, between the upstream clock and the NIC, between the system clock and the NIC, or between the PTP hardware clock ( <b>phc</b> ) and the NIC.	<code>{from="phc",iface="CLOCK_REALTIME",node="compute-1.example.com",process="phc2sys"} -6768</code>
<code>openshift_ptp_interface_role</code>	Returns the configured PTP clock role for the interface. Possible values are 0 ( <b>PASSIVE</b> ), 1 ( <b>SLAVE</b> ), 2 ( <b>MASTER</b> ), 3 ( <b>FAULTY</b> ), 4 ( <b>UNKNOWN</b> ), or 5 ( <b>LISTENING</b> ).	<code>{iface="ens2f0", node="compute-1.example.com",process="ptp4l"} 2</code>
<code>openshift_ptp_max_offset_ns</code>	Returns the maximum offset in nanoseconds between 2 clocks or interfaces. For example, between the upstream GNSS clock and the NIC ( <b>ts2phc</b> ), or between the PTP hardware clock ( <b>phc</b> ) and the system clock ( <b>phc2sys</b> ).	<code>{from="master", iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1.038099569e+09</code>
<code>openshift_ptp_offset_ns</code>	Returns the offset in nanoseconds between the DPLL clock or the GNSS clock source and the NIC hardware clock.	<code>{from="phc",iface="CLOCK_REALTIME",node="compute-1.example.com",process="phc2sys"} -9</code>
<code>openshift_ptp_process_restart_count</code>	Returns a count of the number of times the <b>ptp4l</b> and <b>ts2phc</b> processes were restarted.	<code>{config="ptp4l.0.config",node="compute-1.example.com",process="phc2sys"} 1</code>
<code>openshift_ptp_process_status</code>	Returns a status code that shows whether the PTP processes are running or not.	<code>{config="ptp4l.0.config",node="compute-1.example.com",process="phc2sys"} 1</code>

Metric	Description	Example
<code>openshift_ptp_threshold</code>	<p>Returns values for <b>HoldOverTimeout</b>, <b>MaxOffsetThreshold</b>, and <b>MinOffsetThreshold</b>.</p> <ul style="list-style-type: none"> <li>• <b>holdOverTimeout</b> is the time value in seconds before the PTP clock event state changes to <b>FREERUN</b> when the PTP master clock is disconnected.</li> <li>• <b>maxOffsetThreshold</b> and <b>minOffsetThreshold</b> are offset values in nanoseconds that compare against the values for <b>CLOCK_REALTIME</b> (<b>phc2sys</b>) or master offset (<b>ptp4l</b>) values that you configure in the <b>PtpConfig</b> CR for the NIC.</li> </ul>	<code>{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5</code>

#### PTP fast event metrics only when T-GM is enabled

The following table describes the PTP fast event metrics that are available only when PTP grandmaster clock (T-GM) is enabled.

Table 14.22. PTP fast event metrics when T-GM is enabled

Metric	Description	Example
<code>openshift_ptp_frequency_status</code>	Returns the current status of the digital phase-locked loop (DPLL) frequency for the NIC. Possible values are -1 ( <b>UNKNOWN</b> ), 0 ( <b>INVALID</b> ), 1 ( <b>FREERUN</b> ), 2 ( <b>LOCKED</b> ), 3 ( <b>LOCKED_HO_ACQ</b> ), or 4 ( <b>HOLDOVER</b> ).	<code>{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3</code>
<code>openshift_ptp_nmea_status</code>	Returns the current status of the NMEA connection. NMEA is the protocol that is used for 1PPS NIC connections. Possible values are 0 ( <b>UNAVAILABLE</b> ) and 1 ( <b>AVAILABLE</b> ).	<code>{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1</code>
<code>openshift_ptp_phase_status</code>	Returns the status of the DPLL phase for the NIC. Possible values are -1 ( <b>UNKNOWN</b> ), 0 ( <b>INVALID</b> ), 1 ( <b>FREERUN</b> ), 2 ( <b>LOCKED</b> ), 3 ( <b>LOCKED_HO_ACQ</b> ), or 4 ( <b>HOLDOVER</b> ).	<code>{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3</code>
<code>openshift_ptp_pps_status</code>	Returns the current status of the NIC 1PPS connection. You use the 1PPS connection to synchronize timing between connected NICs. Possible values are 0 ( <b>UNAVAILABLE</b> ) and 1 ( <b>AVAILABLE</b> ).	<code>{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1</code>

Metric	Description	Example
<b>openshift_ptp_gns_s_status</b>	Returns the current status of the global navigation satellite system (GNSS) connection. GNSS provides satellite-based positioning, navigation, and timing services globally. Possible values are 0 ( <b>NOFIX</b> ), 1 ( <b>DEAD RECKONING ONLY</b> ), 2 ( <b>2D-FIX</b> ), 3 ( <b>3D-FIX</b> ), 4 ( <b>GPS+DEAD RECKONING FIX</b> ), 5, ( <b>TIME ONLY FIX</b> ).	<pre>{from="gnss",iface="ens2fx",node="compute-1.example.com",process="gnss"} 3</pre>

## 14.6. PTP EVENTS REST API V1 REFERENCE

Use the following Precision Time Protocol (PTP) fast event REST API v1 endpoints to subscribe the **cloud-event-consumer** application to PTP events posted by the **cloud-event-proxy** container at <http://localhost:8089/api/ocloudNotifications/v1/> in the application pod.



### IMPORTANT

PTP events REST API v1 and events consumer application sidecar is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

The following API endpoints are available:

- [api/ocloudNotifications/v1/subscriptions](#)
  - **POST**: Creates a new subscription
  - **GET**: Retrieves a list of subscriptions
  - **DELETE**: Deletes all subscriptions
- [api/ocloudNotifications/v1/subscriptions/{subscription\\_id}](#)
  - **GET**: Returns details for the specified subscription ID
  - **DELETE**: Deletes the subscription associated with the specified subscription ID
- [api/ocloudNotifications/v1/health](#)
  - **GET**: Returns the health status of **ocloudNotifications** API
- [api/ocloudNotifications/v1/publishers](#)
  - **GET**: Returns a list of PTP event publishers for the cluster node
- [api/ocloudnotifications/v1/{resource\\_address}/CurrentState](#)
  - **GET**: Returns the current state of one the following event types: **sync-state**, **os-clock-state**, **clock-state**, **clock-offset**, **clock-offset-change**, **clock-offset-change-state**, **clock-offset-change-state-state**

**sync-state**, **clock-class**, **lock-state**, or **gnss-sync-status** events

## 14.6.1. PTP events REST API v1 endpoints

### 14.6.1.1. api/ocloudNotifications/v1/subscriptions

HTTP method

**GET api/ocloudNotifications/v1/subscriptions**

#### Description

Returns a list of subscriptions. If subscriptions exist, a **200 OK** status code is returned along with the list of subscriptions.

#### Example API response

```
[  
 {  
   "id": "75b1ad8f-c807-4c23-acf5-56f4b7ee3826",  
   "endpointUri": "http://localhost:9089/event",  
   "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/75b1ad8f-c807-4c23-  
acf5-56f4b7ee3826",  
   "resource": "/cluster/node/compute-1.example.com/ptp"  
 }  
]
```

HTTP method

**POST api/ocloudNotifications/v1/subscriptions**

#### Description

Creates a new subscription for the required event by passing the appropriate payload. If a subscription is successfully created, or if it already exists, a **201 Created** status code is returned. You can subscribe to the following PTP events:

- **lock-state** events
- **os-clock-sync-state** events
- **clock-class** events
- **gnss-sync-status** events
- **sync-state** events

Table 14.23. Query parameters

Parameter	Type
subscription	data

#### Example PTP events subscription payload

```
{  
   "endpointUri": "http://localhost:8989/event",  
   "resource": "/cluster/node/compute-1.example.com/ptp"
```

```
}
```

### Example PTP lock-state events subscription payload

```
{
  "endpointUri": "http://localhost:8989/event",
  "resource": "/cluster/node/{node_name}/sync/ptp-status/lock-state"
}
```

### Example PTP os-clock-sync-state events subscription payload

```
{
  "endpointUri": "http://localhost:8989/event",
  "resource": "/cluster/node/{node_name}/sync/sync-status/os-clock-sync-state"
}
```

### Example PTP clock-class events subscription payload

```
{
  "endpointUri": "http://localhost:8989/event",
  "resource": "/cluster/node/{node_name}/sync/ptp-status/clock-class"
}
```

### Example PTP gnss-sync-status events subscription payload

```
{
  "endpointUri": "http://localhost:8989/event",
  "resource": "/cluster/node/{node_name}/sync/gnss-status/gnss-sync-status"
}
```

### Example sync-state subscription payload

```
{
  "endpointUri": "http://localhost:8989/event",
  "resource": "/cluster/node/{node_name}/sync/sync-status/sync-state"
}
```

#### HTTP method

**DELETE api/ocloudNotifications/v1/subscriptions**

#### Description

Deletes all subscriptions.

### Example API response

```
{
  "status": "deleted all subscriptions"
}
```

### 14.6.1.2. api/ocloudNotifications/v1/subscriptions/{subscription\_id}

#### HTTP method

**GET api/ocloudNotifications/v1/subscriptions/{subscription\_id}****Description**

Returns details for the subscription with ID **subscription\_id**.

**Table 14.24. Global path parameters**

Parameter	Type
<b>subscription_id</b>	string

**Example API response**

```
{
  "id": "48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "endpointUri": "http://localhost:9089/event",
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

**HTTP method****DELETE api/ocloudNotifications/v1/subscriptions/{subscription\_id}****Description**

Deletes the subscription with ID **subscription\_id**.

**Table 14.25. Global path parameters**

Parameter	Type
<b>subscription_id</b>	string

**Example API response**

```
{
  "status": "OK"
}
```

**14.6.1.3. api/ocloudNotifications/v1/health****HTTP method****GET api/ocloudNotifications/v1/health/****Description**

Returns the health status for the **ocloudNotifications** REST API.

**Example API response**

```
OK
```

#### 14.6.1.4. api/ocloudNotifications/v1/publishers



##### IMPORTANT

The **api/ocloudNotifications/v1/publishers** endpoint is only available from the cloud-event-proxy container in the PTP Operator managed pod. It is not available for consumer applications in the application pod.

##### HTTP method

##### GET api/ocloudNotifications/v1/publishers

##### Description

Returns a list of publisher details for the cluster node. The system generates notifications when the relevant equipment state changes.

You can use equipment synchronization status subscriptions together to deliver a detailed view of the overall synchronization health of the system.

##### Example API response

```
[
  {
    "id": "0fa415ae-a3cf-4299-876a-589438bacf75",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/0fa415ae-a3cf-4299-876a-589438bacf75",
    "resource": "/cluster/node/compute-1.example.com-sync/sync-status/os-clock-sync-state"
  },
  {
    "id": "28cd82df-8436-4f50-bbd9-7a9742828a71",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/28cd82df-8436-4f50-bbd9-7a9742828a71",
    "resource": "/cluster/node/compute-1.example.com-sync/ptp-status/clock-class"
  },
  {
    "id": "44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "resource": "/cluster/node/compute-1.example.com-sync/ptp-status/lock-state"
  },
  {
    "id": "778da345d-4567-67b0-a43f0-rty885a456",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/778da345d-4567-67b0-a43f0-rty885a456",
    "resource": "/cluster/node/compute-1.example.com-sync/gnss-status/gnss-sync-status"
  }
]
```

#### 14.6.1.5. api/ocloudNotifications/v1/{resource\_address}/CurrentState

##### HTTP method

**GET api/ocloudNotifications/v1/cluster/node/{node\_name}/sync/ptp-status/lock-state/CurrentState**

**GET api/ocloudNotifications/v1/cluster/node/{node\_name}/sync/sync-status/os-clock-sync-state/CurrentState**

**GET api/ocloudNotifications/v1/cluster/node/{node\_name}/sync/ptp-status/clock-class/CurrentState**

**GET api/ocloudNotifications/v1/cluster/node/{node\_name}/sync/sync-status/sync-state/CurrentState**

**GET api/ocloudNotifications/v1/cluster/node/{node\_name}/sync/gnss-status/gnss-sync-state/CurrentState**

#### Description

Returns the current state of the **os-clock-sync-state**, **clock-class**, **lock-state**, **gnss-sync-status**, or **sync-state** events for the cluster node.

- **os-clock-sync-state** notifications describe the host operating system clock synchronization state. Can be in **LOCKED** or **FREERUN** state.
- **clock-class** notifications describe the current state of the PTP clock class.
- **lock-state** notifications describe the current status of the PTP equipment lock state. Can be in **LOCKED**, **HOLDOVER** or **FREERUN** state.
- **sync-state** notifications describe the current status of the least synchronized of the **ptp-status/lock-state** and **sync-status/os-clock-sync-state** endpoints.
- **gnss-sync-status** notifications describe the GNSS clock synchronization state.

Table 14.26. Global path parameters

Parameter	Type
<b>resource_address</b>	string

#### Example lock-state API response

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      }
    ]
  }
}
```

```
{
  "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
  "data_type": "metric",
  "value_type": "decimal64.3",
  "value": "29"
}
]
}
}
```

### Example os-clock-sync-state API response

```
{
  "specversion": "0.3",
  "id": "4f51fe99-feaa-4e66-9112-66c5c9b9afcb",
  "source": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "subject": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "datacontenttype": "application/json",
  "time": "2022-11-29T17:44:22.202Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "27"
      }
    ]
  }
}
```

### Example clock-class API response

```
{
  "id": "064c9e67-5ad4-4afb-98ff-189c6aa9c205",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/clock-class",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
```

```

        }
    ]
}
}
```

### Example sync-state API response

```
{
  "specversion": "0.3",
  "id": "8c9d6ecb-ae9f-4106-82c4-0a778a79838d",
  "source": "/sync/sync-status/sync-state",
  "type": "event.sync.sync-status.synchronization-state-change",
  "subject": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
  "datacontenttype": "application/json",
  "time": "2024-08-28T14:50:57.327585316Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      }
    ]
  }
}
```

### Example gnss-sync-status API response

```
{
  "id": "435e1f2a-6854-4555-8520-767325c087d7",
  "type": "event.sync.gnss-status.gnss-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",
  "dataContentType": "application/json",
  "time": "2023-09-27T19:35:33.42347206Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens2fx/master",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens2fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "5"
      }
    ]
  }
}
```

# CHAPTER 15. CIDR RANGE DEFINITIONS

If your cluster uses OVN-Kubernetes, you must specify non-overlapping ranges for Classless Inter-Domain Routing (CIDR) subnet ranges.



## IMPORTANT

For OpenShift Container Platform 4.17 and later versions, clusters use **169.254.0.0/17** for IPv4 and **fd69::/112** for IPv6 as the default masquerade subnet. These ranges should also be avoided by users. For upgraded clusters, there is no change to the default masquerade subnet.

The following subnet types and are mandatory for a cluster that uses OVN-Kubernetes:

- **Join:** Uses a join switch to connect gateway routers to distributed routers. A join switch reduces the number of IP addresses for a distributed router. For a cluster that uses the OVN-Kubernetes plugin, an IP address from a dedicated subnet is assigned to any logical port that attaches to the join switch.
- **Masquerade:** Prevents collisions for identical source and destination IP addresses that are sent from a node as hairpin traffic to the same node after a load balancer makes a routing decision.
- **Transit:** A transit switch is a type of distributed switch that spans across all nodes in the cluster. A transit switch routes traffic between different zones. For a cluster that uses the OVN-Kubernetes plugin, an IP address from a dedicated subnet is assigned to any logical port that attaches to the transit switch.



## NOTE

You can change the join, masquerade, and transit CIDR ranges for your cluster as a post-installation task.

OVN-Kubernetes, the default network provider in OpenShift Container Platform 4.14 and later versions, internally uses the following IP address subnet ranges:

- **V4JoinSubnet:** **100.64.0.0/16**
- **V6JoinSubnet:** **fd98::/64**
- **V4TransitSwitchSubnet:** **100.88.0.0/16**
- **V6TransitSwitchSubnet:** **fd97::/64**
- **defaultV4MasqueradeSubnet:** **169.254.0.0/17**
- **defaultV6MasqueradeSubnet:** **fd69::/112**



## IMPORTANT

The previous list includes join, transit, and masquerade IPv4 and IPv6 address subnets. If your cluster uses OVN-Kubernetes, do not include any of these IP address subnet ranges in any other CIDR definitions in your cluster or infrastructure.

## Additional resources

- For more information about configuring join subnets or transit subnets, see [Configuring OVN-Kubernetes internal IP address subnets](#).

## 15.1. MACHINE CIDR

In the Machine classless inter-domain routing (CIDR) field, you must specify the IP address range for machines or cluster nodes.



### NOTE

Machine CIDR ranges cannot be changed after creating your cluster.

The default is **10.0.0.0/16**. This range must not conflict with any connected networks.

#### Additional resources

- [Cluster Network Operator configuration](#)

## 15.2. SERVICE CIDR

In the Service CIDR field, you must specify the IP address range for services. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed from within the cluster. The default is **172.30.0.0/16**.

## 15.3. POD CIDR

In the pod CIDR field, you must specify the IP address range for pods.

The pod CIDR is the same as the **clusterNetwork** CIDR and the cluster CIDR. The range must be large enough to accommodate your workload. The address block must not overlap with any external service accessed from within the cluster. The default is **10.128.0.0/14**. You can expand the range after cluster installation.

#### Additional resources

- [Cluster Network Operator configuration](#)
- [Configuring the cluster network range](#)

## 15.4. HOST PREFIX

In the Host Prefix field, you must specify the subnet prefix length assigned to pods scheduled to individual machines. The host prefix determines the pod IP address pool for each machine.

For example, if the host prefix is set to **/23**, each machine is assigned a **/23** subnet from the pod CIDR address range. The default is **/23**, allowing 510 cluster nodes, and 510 pod IP addresses per node.

## 15.5. CIDR RANGES FOR HOSTED CONTROL PLANES

For deploying hosted control planes on OpenShift Container Platform, use the following required Classless Inter-Domain Routing (CIDR) subnet ranges:

- **v4InternalSubnet**: 100.65.0.0/16 (OVN-Kubernetes)
- **clusterNetwork**: 10.132.0.0/14 (pod network)
- **serviceNetwork**: 172.31.0.0/16

For more information about OpenShift Container Platform CIDR range definitions, see "CIDR range definitions".

# CHAPTER 16. MULTIPLE NETWORKS

## 16.1. UNDERSTANDING MULTIPLE NETWORKS

By default, OVN-Kubernetes serves as the Container Network Interface (CNI) of an OpenShift Container Platform cluster. With OVN-Kubernetes as the default CNI of a cluster, OpenShift Container Platform administrators or users can leverage [user-defined networks \(UDNs\)](#) or [NetworkAttachmentDefinition \(NADs\)](#) to create one, or multiple, default networks that handle all ordinary network traffic of the cluster. Both user-defined networks and Network Attachment Definitions can serve as the following network types:

- **Primary networks:** Act as the primary network for the pod. By default, all traffic passes through the primary network unless a pod route is configured to send traffic through other networks.
- **Secondary networks:** Act as secondary, non-default networks for a pod. Secondary networks provide separate interfaces dedicated to specific traffic types or purposes. Only pod traffic that is explicitly configured to use a secondary network is routed through its interface.

However, during cluster installation, OpenShift Container Platform administrators can configure alternative default secondary pod networks by leveraging the Multus CNI plugin. With Multus, multiple CNI plugins such as ipvlan, macvlan, or Network Attachment Definitions can be used together to serve as secondary networks for pods.



### NOTE

User-defined networks are only available when OVN-Kubernetes is used as the CNI. They are not supported for use with other CNIs.

You can define an secondary network based on the available CNI plugins and attach one or more of these networks to your pods. You can define more than one secondary network for your cluster depending on your needs. This gives you flexibility when you configure pods that deliver network functionality, such as switching or routing.

For a complete list of supported CNI plugins, see ["Secondary networks in OpenShift Container Platform"](#).

For information about user-defined networks, see [About user-defined networks \(UDNs\)](#).

For information about Network Attachment Definitions, see [Creating primary networks using a NetworkAttachmentDefinition](#).

### 16.1.1. Usage scenarios for a secondary network

You can use a secondary network, also known as a *secondary network*, in situations where network isolation is needed, including data plane and control plane separation. Isolating network traffic is useful for the following performance and security reasons:

1. Performance

**Traffic management:** You can send traffic on two different planes to manage how much traffic is along each plane.

2. Security

**Network isolation:** You can send sensitive traffic onto a network plane that is managed specifically for security considerations, and you can separate private data that must not be shared between tenants or customers.

All of the pods in the cluster still use the cluster-wide default network to maintain connectivity across the cluster. Every pod has an **eth0** interface that is attached to the cluster-wide pod network. You can view the interfaces for a pod by using the **oc exec -it <pod\_name> -- ip a** command. If you add secondary network interfaces that use Multus CNI, they are named **net1**, **net2**, ..., **netN**.

To attach secondary network interfaces to a pod, you must create configurations that define how the interfaces are attached. You specify each interface by using either a **UserDefinedNetwork** custom resource (CR) or a **NetworkAttachmentDefinition** CR. A CNI configuration inside each of these CRs defines how that interface is created.

For more information about creating a **UserDefinedNetwork** CR, see [About user-defined networks](#).

For more information about creating a NetworkAttachmentDefinition CR, see [Creating primary networks using a NetworkAttachmentDefinition](#).

### 16.1.2. Secondary networks in OpenShift Container Platform

OpenShift Container Platform provides the following CNI plugins for creating secondary networks in your cluster:

- **bridge:** [Configure a bridge-based secondary network](#) to allow pods on the same host to communicate with each other and the host.
- **host-device:** [Configure a host-device secondary network](#) to allow pods access to a physical Ethernet network device on the host system.
- **ipvlan:** [Configure an ipvlan-based secondary network](#) to allow pods on a host to communicate with other hosts and pods on those hosts, similar to a macvlan-based secondary network. Unlike a macvlan-based secondary network, each pod shares the same MAC address as the parent physical network interface.
- **vlan:** [Configure a VLAN-based secondary network](#) to allow VLAN-based network isolation and connectivity for pods.
- **macvlan:** [Configure a macvlan-based secondary network](#) to allow pods on a host to communicate with other hosts and pods on those hosts by using a physical network interface. Each pod that is attached to a macvlan-based secondary network is provided a unique MAC address.
- **TAP:** [Configure a TAP-based secondary network](#) to create a tap device inside the container namespace. A TAP device enables user space programs to send and receive network packets.
- **SR-IOV:** [Configure an SR-IOV based secondary network](#) to allow pods to attach to a virtual function (VF) interface on SR-IOV capable hardware on the host system.
- **route-override:** [Configure a route-override based secondary network](#) to allow pods to override and set routes.

### 16.1.3. UserDefinedNetwork and NetworkAttachmentDefinition support matrix

The **UserDefinedNetwork** and **NetworkAttachmentDefinition** custom resources (CRs) provide cluster administrators and users the ability to create customizable network configurations and define their own

network topologies, ensure network isolation, manage IP addressing for workloads, and configure advanced network features. A third CR, **ClusterUserDefinedNetwork**, is also available, which allows administrators the ability to create and define secondary networks spanning multiple namespaces at the cluster level.

User-defined networks and network attachment definitions can serve as both the primary and secondary network interface, and each support **layer2** and **layer3** topologies; a third network topology, Localnet, is also supported with network attachment definitions with secondary networks.



### NOTE

As of OpenShift Container Platform 4.18, the Localnet topology is unavailable for use with the **UserDefinedNetwork** and **ClusterUserDefinedNetwork** CRs. It is only available for **NetworkAttachmentDefinition** CRs that leverage secondary networks.

The following section highlights the supported features of the **UserDefinedNetwork** and **NetworkAttachmentDefinition** CRs when they are used as either the primary or secondary network. A separate table for the **ClusterUserDefinedNetwork** CR is also included.

**Table 16.1. Primary network support matrix forUserDefinedNetwork and NetworkAttachmentDefinition CRs**

Network feature	Layer2 topology	Layer3 topology
east-west traffic	✓	✓
north-south traffic	✓	✓
Persistent IPs	✓	X
Services	✓	✓
Routes	X	X
<b>EgressIP</b> resource	✓	✓
Multicast <sup>[1]</sup>	X	✓
<b>NetworkPolicy</b> resource <sup>[2]</sup>	✓	✓
<b>MultinetworkPolicy</b> resource	X	X

1. Multicast must be enabled in the namespace, and it is only available between OVN-Kubernetes network pods. For more information about multicast, see "Enabling multicast for a project".
2. When creating a **UserDefinedNetwork** CR with a primary network type, network policies must be created *after* the **UserDefinedNetwork** CR.

**Table 16.2. Secondary network support matrix forUserDefinedNetwork and NetworkAttachmentDefinition CRs**

Network feature	Layer2 topology	Layer3 topology	Localnet topology [1]
east-west traffic	✓	✓	✓ <b>(NetworkAttachmentDefinition CR only)</b>
north-south traffic	X	X	✓ <b>(NetworkAttachmentDefinition CR only)</b>
Persistent IPs	✓	X	✓ <b>(NetworkAttachmentDefinition CR only)</b>
Services	X	X	X
Routes	X	X	X
<b>EgressIP</b> resource	X	X	X
Multicast	X	X	X
<b>NetworkPolicy</b> resource	X	X	X
<b>MultinetworkPolicy</b> resource	✓	✓	✓ <b>(NetworkAttachmentDefinition CR only)</b>

1. The Localnet topology is unavailable for use with the **UserDefinedNetwork** CR. It is only supported on secondary networks for **NetworkAttachmentDefinition** CRs.

Table 16.3. Support matrix for **ClusterUserDefinedNetwork** CRs

Network feature	Layer2 topology	Layer3 topology
east-west traffic	✓	✓
north-south traffic	✓	✓
Persistent IPs	✓	X
Services	✓	✓
Routes	X	X

Network feature	Layer2 topology	Layer3 topology
<b>EgressIP</b> resource	✓	✓
Multicast <sup>[1]</sup>	X	✓
<b>MultinetworkPolicy</b> resource	X	X
<b>NetworkPolicy</b> resource <sup>[2]</sup>	✓	✓

1. Multicast must be enabled in the namespace, and it is only available between OVN-Kubernetes network pods. For more information, see "About multicast".
2. When creating a **ClusterUserDefinedNetwork** CR with a primary network type, network policies must be created *after* the **UserDefinedNetwork** CR.

#### Additional resources

- [Enabling multicast for a project](#)

## 16.2. PRIMARY NETWORKS

### 16.2.1. About user-defined networks

Before the implementation of user-defined networks (UDN), the OVN-Kubernetes CNI plugin for OpenShift Container Platform only supported a Layer 3 topology on the primary or *main* network. Due to Kubernetes design principles: all pods are attached to the main network, all pods communicate with each other by their IP addresses, and inter-pod traffic is restricted according to network policy.

While the Kubernetes design is useful for simple deployments, this Layer 3 topology restricts customization of primary network segment configurations, especially for modern multi-tenant deployments.

UDN improves the flexibility and segmentation capabilities of the default Layer 3 topology for a Kubernetes pod network by enabling custom Layer 2 and Layer 3 network segments, where all these segments are isolated by default. These segments act as either primary or secondary networks for container pods and virtual machines that use the default OVN-Kubernetes CNI plugin. UDNs enable a wide range of network architectures and topologies, enhancing network flexibility, security, and performance.



#### NOTE

Nodes that use **cgroupv1** Linux Control Groups (cgroup) must be reconfigured from **cgroupv1** to **cgroupv2** before creating a user-defined network. For more information, see [Configuring Linux cgroup](#).

A cluster administrator can use a UDN to create and define primary or secondary networks that span multiple namespaces at the cluster level by leveraging the **ClusterUserDefinedNetwork** custom resource (CR). Additionally, a cluster administrator or a cluster user can use a UDN to define secondary networks at the namespace level with the **UserDefinedNetwork** CR.

The following sections further emphasize the benefits and limitations of user-defined networks, the best practices when creating a **ClusterUserDefinedNetwork** or **UserDefinedNetwork** CR, how to create the CR, and additional configuration details that might be relevant to your deployment.

### 16.2.1.1. Benefits of a user-defined network

User-defined networks provide the following benefits:

1. Enhanced network isolation for security
  - **Tenant isolation:** Namespaces can have their own isolated primary network, similar to how tenants are isolated in Red Hat OpenStack Platform (RHOSP). This improves security by reducing the risk of cross-tenant traffic.
2. Network flexibility
  - **Layer 2 and layer 3 support** Cluster administrators can configure primary networks as layer 2 or layer 3 network types.
3. Simplified network management
  - **Reduced network configuration complexity:** With user-defined networks, the need for complex network policies are eliminated because isolation can be achieved by grouping workloads in different networks.
4. Advanced capabilities
  - **Consistent and selectable IP addressing** Users can specify and reuse IP subnets across different namespaces and clusters, providing a consistent networking environment.
  - **Support for multiple networks** The user-defined networking feature allows administrators to connect multiple namespaces to a single network, or to create distinct networks for different sets of namespaces.
5. Simplification of application migration from Red Hat OpenStack Platform (RHOSP)
  - **Network parity:** With user-defined networking, the migration of applications from OpenStack to OpenShift Container Platform is simplified by providing similar network isolation and configuration options.

Developers and administrators can create a user-defined network that is namespace scoped using the custom resource. An overview of the process is as follows:

1. An administrator creates a namespace for a user-defined network with the **k8s.ovn.org/primary-user-defined-network** label.
2. The **UserDefinedNetwork** CR is created by either the cluster administrator or the user.
3. The user creates pods in the namespace.

### 16.2.1.2. Limitations of a user-defined network

While user-defined networks (UDN) offer highly customizable network configuration options, there are limitations that cluster administrators and developers should be aware of when implementing and managing these networks. Consider the following limitations before implementing a UDN.

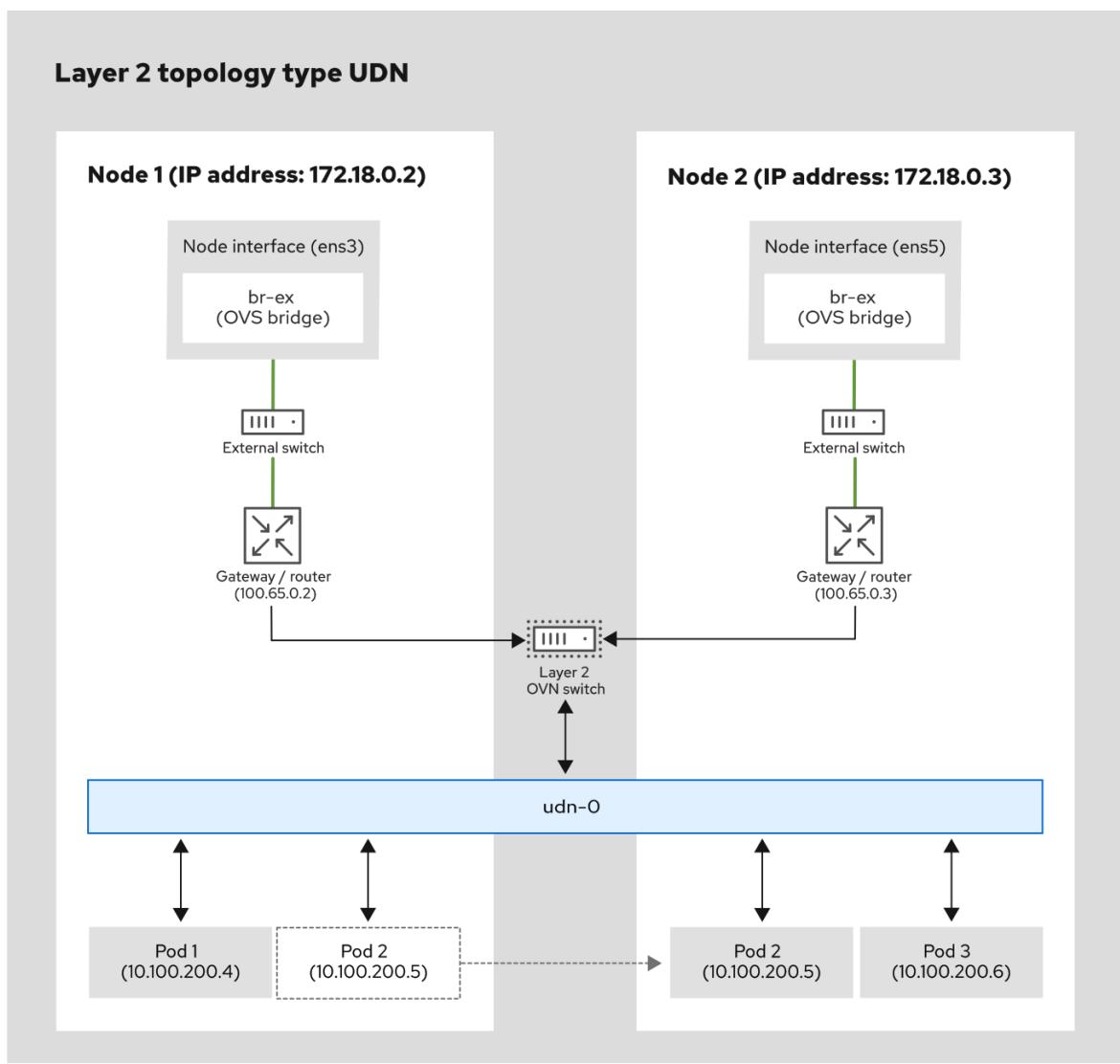
- **DNS limitations:**

- DNS lookups for pods resolve to the pod’s IP address on the cluster default network. Even if a pod is part of a user-defined network, DNS lookups will not resolve to the pod’s IP address on that user-defined network. However, DNS lookups for services and external entities will function as expected.
  - When a pod is assigned to a primary UDN, it can access the Kubernetes API (KAPI) and DNS services on the cluster’s default network.
- **Initial network assignment** You must create the namespace and network before creating pods. Assigning a namespace with pods to a new network or creating a UDN in an existing namespace will not be accepted by OVN-Kubernetes.
  - **Health check limitations** Kubelet health checks are performed by the cluster default network, which does not confirm the network connectivity of the primary interface on the pod. Consequently, scenarios where a pod appears healthy by the default network, but has broken connectivity on the primary interface, are possible with user-defined networks.
  - **Network policy limitations** Network policies that enable traffic between namespaces connected to different user-defined primary networks are not effective. These traffic policies do not take effect because there is no connectivity between these isolated networks.
  - **Creation and modification limitation:** The **ClusterUserDefinedNetwork** CR and the **UserDefinedNetwork** CR cannot be modified after being created.
  - **Default network service access:** A user-defined network pod is isolated from the default network, which means that most default network services are inaccessible. For example, a user-defined network pod cannot currently access the OpenShift Container Platform image registry. Because of this limitation, source-to-image builds do not work in a user-defined network namespace. Additionally, other functions do not work, including functions to create applications based on the source code in a Git repository, such as **oc new-app <command>**, and functions to create applications from an OpenShift Container Platform template that use source-to-image builds. This limitation might also affect other **openshift-\* .svc** services.
  - **Connectivity limitation:** NodePort services on user-defined networks are not guaranteed isolation. For example, NodePort traffic from a pod to a service on the same node is not accessible, whereas traffic from a pod on a different node succeeds.

#### 16.2.1.3. Layer 2 and layer 3 topologies

A flat layer 2 topology creates a virtual switch that is distributed across all nodes in a cluster. Virtual machines and pods connect to this virtual switch so that all these components can communicate with each other within the same subnet. A flat layer 2 topology is useful for live migration of virtual machines across nodes that exist in a cluster. The following diagram shows a flat layer 2 topology with two nodes that use the virtual switch for live migration purposes:

Figure 16.1. A flat layer 2 topology that uses a virtual switch for component communication



504\_OpenShift\_udn\_L2\_0325

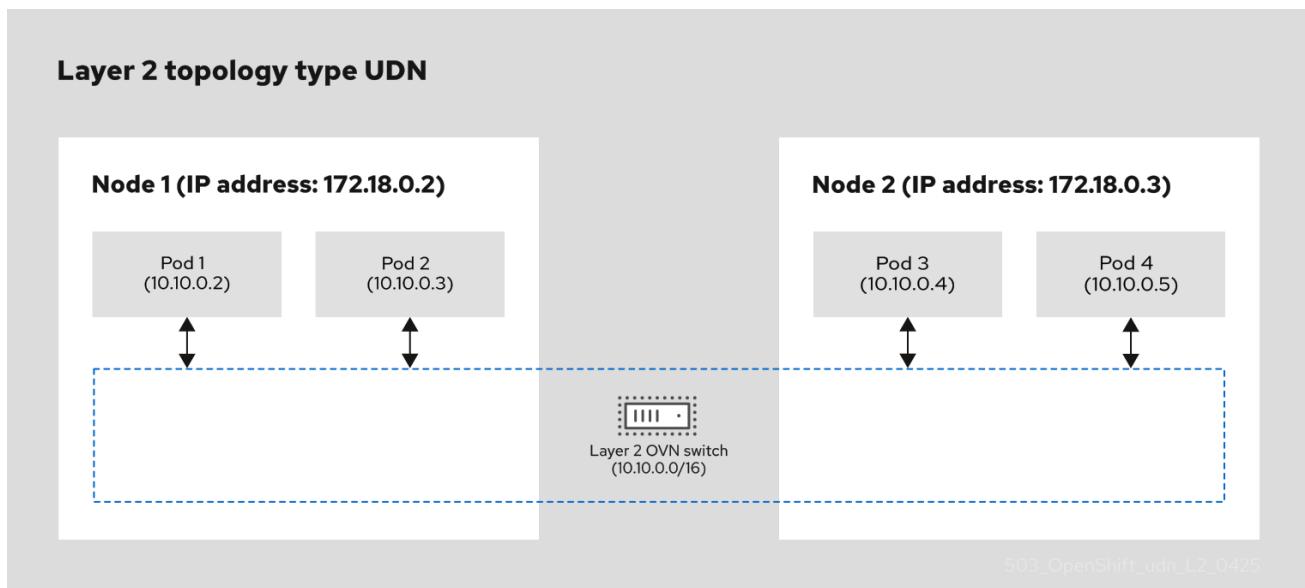
If you decide not to specify a layer 2 subnet, then you must manually configure IP addresses for each pod in your cluster. When you do not specify a layer 2 subnet, port security is limited to preventing Media Access Control (MAC) spoofing only, and does not include IP spoofing. A layer 2 topology creates a single broadcast domain that can be challenging in large network environments, where the topology might cause a broadcast storm that can degrade network performance.

To access more configurable options for your network, you can integrate a layer 2 topology with a user-defined network (UDN). The following diagram shows two nodes that use a UDN with a layer 2 topology that includes pods that exist on each node. Each node includes two interfaces:

- A node interface, which is a compute node that connects networking components to the node.
- An Open vSwitch (OVS) bridge such as **br-ex**, which creates an layer 2 OVN switch so that pods can communicate with each other and share resources.

An external switch connects these two interfaces, while the gateway or router handles routing traffic between the external switch and the layer 2 OVN switch. VMs and pods in a node can use the UDN to communicate with each other. The layer 2 OVN switch handles node traffic over a UDN so that live migrate of a VM from one node to another is possible.

Figure 16.2. A user-defined network (UDN) that uses a layer 2 topology



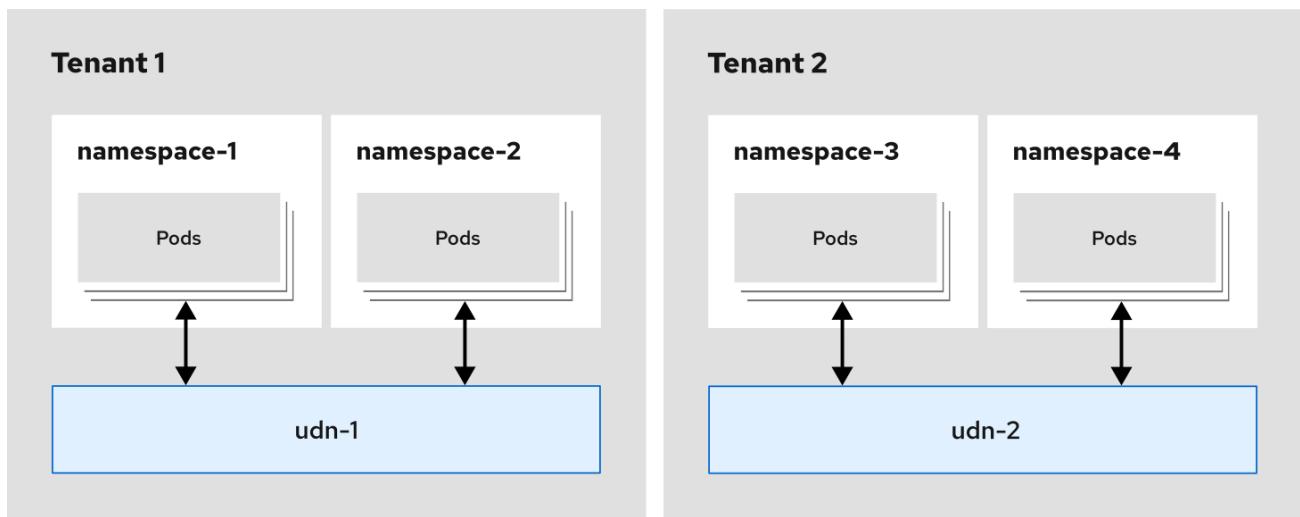
A layer 3 topology creates a unique layer 2 segment for each node in a cluster. The layer 3 routing mechanism interconnects these segments so that virtual machines and pods that are hosted on different nodes can communicate with each other. A layer 3 topology can effectively manage large broadcast domains by assigning each domain to a specific node, so that broadcast traffic has a reduced scope. To configure a layer 3 topology, you must configure **cidr** and **hostSubnet** parameters.

#### 16.2.1.4. About the ClusterUserDefinedNetwork CR

The **ClusterUserDefinedNetwork** (UDN) custom resource (CR) provides cluster-scoped network segmentation and isolation for administrators only.

The following diagram demonstrates how a cluster administrator can use the **ClusterUserDefinedNetwork** CR to create network isolation between tenants. This network configuration allows a network to span across many namespaces. In the diagram, network isolation is achieved through the creation of two user-defined networks, **udn-1** and **udn-2**. These networks are not connected and the **spec.namespaceSelector.matchLabels** field is used to select different namespaces. For example, **udn-1** configures and isolates communication for **namespace-1** and **namespace-2**, while **udn-2** configures and isolates communication for **namespace-3** and **namespace-4**. Isolated tenants (Tenants 1 and Tenants 2) are created by separating namespaces while also allowing pods in the same namespace to communicate.

Figure 16.3. Tenant isolation using a ClusterUserDefinedNetwork CR



528\_OpenShift\_0225

#### 16.2.1.4.1. Best practices for ClusterUserDefinedNetwork CRs

Before setting up a **ClusterUserDefinedNetwork** custom resource (CR), users should consider the following information:

- A **ClusterUserDefinedNetwork** CR is intended for use by cluster administrators and should not be used by non-administrators. If used incorrectly, it might result in security issues with your deployment, cause disruptions, or break the cluster network.
- **ClusterUserDefinedNetwork** CRs should not select the **default** namespace. This can result in no isolation and, as a result, could introduce security risks to the cluster.
- **ClusterUserDefinedNetwork** CRs should not select **openshift-\*** namespaces.
- OpenShift Container Platform administrators should be aware that all namespaces of a cluster are selected when one of the following conditions are met:
  - The **matchLabels** selector is left empty.
  - The **matchExpressions** selector is left empty.
  - The **namespaceSelector** is initialized, but does not specify **matchExpressions** or **matchLabel**. For example: **namespaceSelector: {}**.
- For primary networks, the namespace used for the **ClusterUserDefinedNetwork** CR must include the **k8s.ovn.org/primary-user-defined-network** label. This label cannot be updated, and can only be added when the namespace is created. The following conditions apply with the **k8s.ovn.org/primary-user-defined-network** namespace label:
  - If the namespace is missing the **k8s.ovn.org/primary-user-defined-network** label and a pod is created, the pod attaches itself to the default network.
  - If the namespace is missing the **k8s.ovn.org/primary-user-defined-network** label and a primary **ClusterUserDefinedNetwork** CR is created that matches the namespace, an error is reported and the network is not created.

- If the namespace is missing the **k8s.ovn.org/primary-user-defined-network** label and a primary **ClusterUserDefinedNetwork** CR already exists, a pod in the namespace is created and attached to the default network.
- If the namespace *has* the label, and a primary **ClusterUserDefinedNetwork** CR does not exist, a pod in the namespace is not created until the **ClusterUserDefinedNetwork** CR is created.

#### 16.2.1.4.2. Creating a ClusterUserDefinedNetwork CR by using the CLI

The following procedure creates a **ClusterUserDefinedNetwork** custom resource (CR) by using the CLI. Based upon your use case, create your request using either the **cluster-layer-two-udn.yaml** example for a **Layer2** topology type or the **cluster-layer-three-udn.yaml** example for a **Layer3** topology type.



#### IMPORTANT

- The **ClusterUserDefinedNetwork** CR is intended for use by cluster administrators and should not be used by non-administrators. If used incorrectly, it might result in security issues with your deployment, cause disruptions, or break the cluster network.
- OpenShift Virtualization only supports the **Layer2** topology.

#### Prerequisites

- You have logged in as a user with **cluster-admin** privileges.

#### Procedure

1. Optional: For a **ClusterUserDefinedNetwork** CR that uses a primary network, create a namespace with the **k8s.ovn.org/primary-user-defined-network** label by entering the following command:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: <cudn_namespace_name>
  labels:
    k8s.ovn.org/primary-user-defined-network: ""
EOF
```

2. Create a request for either a **Layer2** or **Layer3** topology type cluster-wide user-defined network:

- a. Create a YAML file, such as **cluster-layer-two-udn.yaml**, to define your request for a **Layer2** topology as in the following example:

```
apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  name: <cudn_name> ①
spec:
  namespaceSelector: ②
  matchLabels: ③
```

```

"<label_1_key>": "<label_1_value>" ④
"<label_2_key>": "<label_2_value>" ⑤
network: ⑥
topology: Layer2 ⑦
layer2: ⑧
role: Primary ⑨
subnets:
  - "2001:db8::/64"
  - "10.100.0.0/16" ⑩

```

- ① Name of your **ClusterUserDefinedNetwork** CR.
- ② A label query over the set of namespaces that the cluster UDN CR applies to. Uses the standard Kubernetes **MatchLabel** selector. Must not point to **default** or **openshift-\*** namespaces.
- ③ Uses the **matchLabels** selector type, where terms are evaluated with an **AND** relationship.
- ④ ⑤ Because the **matchLabels** selector type is used, provisions namespaces that contain both **<label\_1\_key>=<label\_1\_value>** and **<label\_2\_key>=<label\_2\_value>** labels.
- ⑥ Describes the network configuration.
- ⑦ The **topology** field describes the network configuration; accepted values are **Layer2** and **Layer3**. Specifying a **Layer2** topology type creates one logical switch that is shared by all nodes.
- ⑧ This field specifies the topology configuration. It can be **layer2** or **layer3**.
- ⑨ Specifies **Primary** or **Secondary**. **Primary** is the only **role** specification supported in 4.18.
- ⑩ For **Layer2** topology types the following specifies config details for the **subnet** field:
  - The subnets field is optional.
  - The subnets field is of type **string** and accepts standard CIDR formats for both IPv4 and IPv6.
  - The subnets field accepts one or two items. For two items, they must be of a different family. For example, subnets values of **10.100.0.0/16** and **2001:db8::/64**.
  - **Layer2** subnets can be omitted. If omitted, users must configure static IP addresses for the pods. As a consequence, port security only prevents MAC spoofing. For more information, see "Configuring pods with a static IP address".

- b. Create a YAML file, such as **cluster-layer-three-udn.yaml**, to define your request for a **Layer3** topology as in the following example:

```

apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  name: <cudn_name> ①
spec:

```

```

namespaceSelector: ②
  matchExpressions: ③
    - key: kubernetes.io/metadata.name ④
      operator: In ⑤
      values: ["<example_namespace_one>", "<example_namespace_two>"] ⑥
network: ⑦
  topology: Layer3 ⑧
  layer3: ⑨
    role: Primary ⑩
    subnets: ⑪
      - cidr: 10.100.0.0/16
        hostSubnet: 24

```

- ① Name of your **ClusterUserDefinedNetwork** CR.
- ② A label query over the set of namespaces that the cluster UDN applies to. Uses the standard Kubernetes **MatchLabel** selector. Must not point to **default** or **openshift-\*** namespaces.
- ③ Uses the **matchExpressions** selector type, where terms are evaluated with an **OR** relationship.
- ④ Specifies the label key to match.
- ⑤ Specifies the operator. Valid values include: **In**, **NotIn**, **Exists**, and **DoesNotExist**.
- ⑥ Because the **matchExpressions** type is used, provisions namespaces matching either **<example\_namespace\_one>** or **<example\_namespace\_two>**.
- ⑦ Describes the network configuration.
- ⑧ The **topology** field describes the network configuration; accepted values are **Layer2** and **Layer3**. Specifying a **Layer3** topology type creates a layer 2 segment per node, each with a different subnet. Layer 3 routing is used to interconnect node subnets.
- ⑨ This field specifies the topology configuration. Valid values are **layer2** or **layer3**.
- ⑩ Specifies a **Primary** or **Secondary** role. **Primary** is the only **role** specification supported in 4.18.
- ⑪ For **Layer3** topology types the following specifies config details for the **subnet** field:
  - The **subnets** field is mandatory.
  - The type for the **subnets** field is **cidr** and **hostSubnet**:
    - **cidr** is the cluster subnet and accepts a string value.
    - **hostSubnet** specifies the nodes subnet prefix that the cluster subnet is split to.
    - For IPv6, only a /64 length is supported for **hostSubnet**.

3. Apply your request by running the following command:

```
$ oc create --validate=true -f <example_cluster_udn>.yaml
```

Where **<example\_cluster\_udn>.yaml** is the name of your **Layer2** or **Layer3** configuration file.

- Verify that your request is successful by running the following command:

```
$ oc get clusteruserdefinednetwork <cudn_name> -o yaml
```

Where **<cudn\_name>** is the name you created of your cluster-wide user-defined network.

### Example output

```
apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  creationTimestamp: "2024-12-05T15:53:00Z"
  finalizers:
  - k8s.ovn.org/user-defined-network-protection
  generation: 1
  name: my-cudn
  resourceVersion: "47985"
  uid: 16ee0fcf-74d1-4826-a6b7-25c737c1a634
spec:
  namespaceSelector:
    matchExpressions:
    - key: custom.network.selector
      operator: In
      values:
      - example-namespace-1
      - example-namespace-2
      - example-namespace-3
  network:
    layer3:
      role: Primary
      subnets:
      - cidr: 10.100.0.0/16
    topology: Layer3
  status:
    conditions:
    - lastTransitionTime: "2024-11-19T16:46:34Z"
      message: 'NetworkAttachmentDefinition has been created in following namespaces:
      [example-namespace-1, example-namespace-2, example-namespace-3]'
      reason: NetworkAttachmentDefinitionReady
      status: "True"
    type: NetworkCreated
```

#### 16.2.1.4.3. Creating a ClusterUserDefinedNetwork CR by using the web console

You can create a **ClusterUserDefinedNetwork** custom resource (CR) with a **Layer2** topology in the OpenShift Container Platform web console.



## NOTE

Currently, creation of a **ClusterUserDefinedNetwork** CR with a **Layer3** topology is not supported when using the OpenShift Container Platform web console.

## Prerequisites

- You have access to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- You have created a namespace and applied the **k8s.ovn.org/primary-user-defined-network** label.

## Procedure

1. From the **Administrator** perspective, click **Networking** → **UserDefinedNetworks**.
2. Click **ClusterUserDefinedNetwork**.
3. In the **Name** field, specify a name for the cluster-scoped UDN.
4. Specify a value in the **Subnet** field.
5. In the **Project(s) Match Labels** field, add the appropriate labels to select namespaces that the cluster UDN applies to.
6. Click **Create**. The cluster-scoped UDN serves as the default primary network for pods located in namespaces that contain the labels that you specified in step 5.

## Additional resources

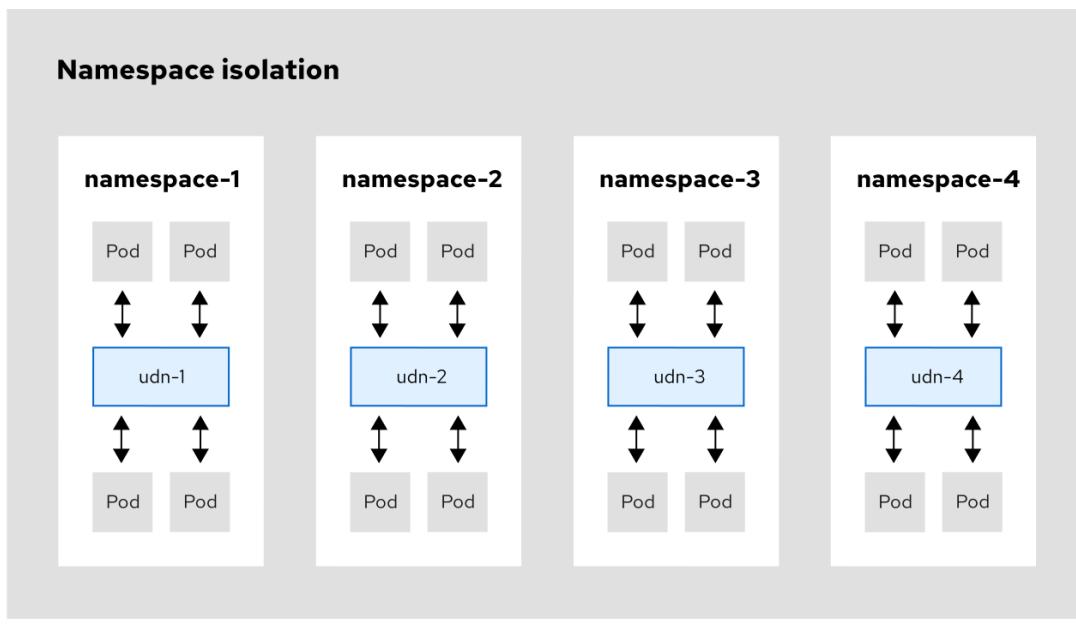
- [Configuring pods with a static IP address](#)

### 16.2.1.5. About the UserDefinedNetwork CR

The **UserDefinedNetwork** (UDN) custom resource (CR) provides advanced network segmentation and isolation for users and administrators.

The following diagram shows four cluster namespaces, where each namespace has a single assigned user-defined network (UDN), and each UDN has an assigned custom subnet for its pod IP allocations. The OVN-Kubernetes handles any overlapping UDN subnets. Without using the Kubernetes network policy, a pod attached to a UDN can communicate with other pods in that UDN. By default, these pods are isolated from communicating with pods that exist in other UDNs. For microsegmentation, you can apply network policy within a UDN. You can assign one or more UDNs to a namespace, with a limitation of only one primary UDN to a namespace, and one or more namespaces to a UDN.

Figure 16.4. Namespace isolation using a UserDefinedNetwork CR



527-OpenShift-UDN-isolation-012025

#### 16.2.1.5.1. Best practices for UserDefinedNetwork CRs

Before setting up a **UserDefinedNetwork** custom resource (CR), you should consider the following information:

- **openshift-\*** namespaces should not be used to set up a **UserDefinedNetwork** CR.
- **UserDefinedNetwork** CRs should not be created in the default namespace. This can result in no isolation and, as a result, could introduce security risks to the cluster.
- For primary networks, the namespace used for the **UserDefinedNetwork** CR must include the **k8s.ovn.org/primary-user-defined-network** label. This label cannot be updated, and can only be added when the namespace is created. The following conditions apply with the **k8s.ovn.org/primary-user-defined-network** namespace label:
  - If the namespace is missing the **k8s.ovn.org/primary-user-defined-network** label and a pod is created, the pod attaches itself to the default network.
  - If the namespace is missing the **k8s.ovn.org/primary-user-defined-network** label and a primary **UserDefinedNetwork** CR is created that matches the namespace, a status error is reported and the network is not created.
  - If the namespace is missing the **k8s.ovn.org/primary-user-defined-network** label and a primary **UserDefinedNetwork** CR already exists, a pod in the namespace is created and attached to the default network.
  - If the namespace *has* the label, and a primary **UserDefinedNetwork** CR does not exist, a pod in the namespace is not created until the **UserDefinedNetwork** CR is created.
- 2 masquerade IP addresses are required for user defined networks. You must reconfigure your masquerade subnet to be large enough to hold the required number of networks.



## IMPORTANT

- For OpenShift Container Platform 4.17 and later, clusters use **169.254.0.0/17** for IPv4 and **fd69::/12** for IPv6 as the default masquerade subnet. These ranges should be avoided by users. For updated clusters, there is no change to the default masquerade subnet.
- Changing the cluster's masquerade subnet is unsupported after a user-defined network has been configured for a project. Attempting to modify the masquerade subnet after a **UserDefinedNetwork** CR has been set up can disrupt the network connectivity and cause configuration issues.
- Ensure tenants are using the **UserDefinedNetwork** resource and not the **NetworkAttachmentDefinition** (NAD) CR. This can create security risks between tenants.
- When creating network segmentation, you should only use the **NetworkAttachmentDefinition** CR if user-defined network segmentation cannot be completed using the **UserDefinedNetwork** CR.
- The cluster subnet and services CIDR for a **UserDefinedNetwork** CR cannot overlap with the default cluster subnet CIDR. OVN-Kubernetes network plugin uses **100.64.0.0/16** as the default join subnet for the network. You must not use that value to configure a **UserDefinedNetwork** CR's **joinSubnets** field. If the default address values are used anywhere in the network for the cluster you must override the default values by setting the **joinSubnets** field. For more information, see "Additional configuration details for user-defined networks".

### 16.2.1.5.2. Creating a UserDefinedNetwork CR by using the CLI

The following procedure creates a **UserDefinedNetwork** CR that is namespace scoped. Based upon your use case, create your request by using either the **my-layer-two-udn.yaml** example for a **Layer2** topology type or the **my-layer-three-udn.yaml** example for a **Layer3** topology type.

#### Prerequisites

- You have logged in with **cluster-admin** privileges, or you have **view** and **edit** role-based access control (RBAC).

#### Procedure

1. Optional: For a **UserDefinedNetwork** CR that uses a primary network, create a namespace with the **k8s.ovn.org/primary-user-defined-network** label by entering the following command:

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: <udn_namespace_name>
  labels:
    k8s.ovn.org/primary-user-defined-network: ""
EOF
```

2. Create a request for either a **Layer2** or **Layer3** topology type user-defined network:

- a. Create a YAML file, such as **my-layer-two-udn.yaml**, to define your request for a **Layer2** topology as in the following example:

```

apiVersion: k8s.ovn.org/v1
kind: UserDefinedNetwork
metadata:
  name: udn-1 1
  namespace: <some_custom_namespace>
spec:
  topology: Layer2 2
  layer2: 3
    role: Primary 4
  subnets:
    - "10.0.0.0/24"
    - "2001:db8::/60" 5

```

- 1** Name of your **UserDefinedNetwork** resource. This should not be **default** or duplicate any global namespaces created by the Cluster Network Operator (CNO).
- 2** The **topology** field describes the network configuration; accepted values are **Layer2** and **Layer3**. Specifying a **Layer2** topology type creates one logical switch that is shared by all nodes.
- 3** This field specifies the topology configuration. It can be **layer2** or **layer3**.
- 4** Specifies a **Primary** or **Secondary** role.
- 5** For **Layer2** topology types the following specifies config details for the **subnet** field:
  - The subnets field is optional.
  - The subnets field is of type **string** and accepts standard CIDR formats for both IPv4 and IPv6.
  - The subnets field accepts one or two items. For two items, they must be of a different family. For example, subnets values of **10.100.0.0/16** and **2001:db8::/64**.
  - **Layer2** subnets can be omitted. If omitted, users must configure IP addresses for the pods. As a consequence, port security only prevents MAC spoofing.
  - The **Layer2 subnets** field is mandatory when the **ipamLifecycle** field is specified.

- b. Create a YAML file, such as **my-layer-three-udn.yaml**, to define your request for a **Layer3** topology as in the following example:

```

apiVersion: k8s.ovn.org/v1
kind: UserDefinedNetwork
metadata:
  name: udn-2-primary 1
  namespace: <some_custom_namespace>
spec:
  topology: Layer3 2
  layer3: 3
    role: Primary 4
  subnets: 5
    - cidr: 10.150.0.0/16
      hostSubnet: 24

```

```

    - cidr: 2001:db8::/60
      hostSubnet: 64
    # ...
  
```

- 1 Name of your **UserDefinedNetwork** resource. This should not be **default** or duplicate any global namespaces created by the Cluster Network Operator (CNO).
- 2 The **topology** field describes the network configuration; accepted values are **Layer2** and **Layer3**. Specifying a **Layer3** topology type creates a layer 2 segment per node, each with a different subnet. Layer 3 routing is used to interconnect node subnets.
- 3 This field specifies the topology configuration. Valid values are **layer2** or **layer3**.
- 4 Specifies a **Primary** or **Secondary** role.
- 5 For **Layer3** topology types the following specifies config details for the **subnet** field:
  - The **subnets** field is mandatory.
  - The type for the **subnets** field is **cidr** and **hostSubnet**:
    - **cidr** is equivalent to the **clusterNetwork** configuration settings of a cluster. The IP addresses in the CIDR are distributed to pods in the user defined network. This parameter accepts a string value.
    - **hostSubnet** defines the per-node subnet prefix.
    - For IPv6, only a /64 length is supported for **hostSubnet**.

- 3 Apply your request by running the following command:

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

Where **<my\_layer\_two\_udn>.yaml** is the name of your **Layer2** or **Layer3** configuration file.

- 4 Verify that your request is successful by running the following command:

```
$ oc get userdefinednetworks udn-1 -n <some_custom_namespace> -o yaml
```

Where **some\_custom\_namespace** is the namespace you created for your user-defined network.

### Example output

```

apiVersion: k8s.ovn.org/v1
kind: UserDefinedNetwork
metadata:
  creationTimestamp: "2024-08-28T17:18:47Z"
  finalizers:
  - k8s.ovn.org/user-defined-network-protection
  generation: 1
  name: udn-1
  namespace: some-custom-namespace
  resourceVersion: "53313"
  uid: f483626d-6846-48a1-b88e-6bbeb8bcde8c
  
```

```

spec:
layer2:
  role: Primary
  subnets:
    - 10.0.0.0/24
    - 2001:db8::/60
topology: Layer2
status:
  conditions:
    - lastTransitionTime: "2024-08-28T17:18:47Z"
      message: NetworkAttachmentDefinition has been created
      reason: NetworkAttachmentDefinitionReady
      status: "True"
      type: NetworkCreated

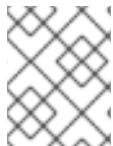
```

## Additional resources

- [Default cluster roles](#)

### 16.2.1.5.3. Creating a UserDefinedNetwork CR by using the web console

You can create a **UserDefinedNetwork** custom resource (CR) with a **Layer2** topology and **Primary** role by using the OpenShift Container Platform web console.



#### NOTE

Currently, creation of a **UserDefinedNetwork** CR with a **Layer3** topology or a **Secondary** role are not supported when using the OpenShift Container Platform web console.

## Prerequisites

- You have access to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- You have created a namespace and applied the **k8s.ovn.org/primary-user-defined-network** label.

## Procedure

1. From the **Administrator** perspective, click **Networking** → **UserDefinedNetworks**.
2. Click **Create UserDefinedNetwork**.
3. From the **Project name** list, select the namespace that you previously created.
4. Specify a value in the **Subnet** field.
5. Click **Create**. The user-defined network serves as the default primary network for pods that you create in this namespace.

### 16.2.1.6. Additional configuration details for user-defined networks

The following table explains additional configurations for **ClusterUserDefinedNetwork** and **UserDefinedNetwork** custom resources (CRs) that are optional. It is not recommended to set these fields without explicit need and understanding of OVN-Kubernetes network topology.

## 1. Optional configurations for user-defined networks

CUDN field	UDN field	Type	Description
<b>spec.network.&lt;topology&gt;.joinSubnets</b>	<b>spec.&lt;topology&gt;.joinSubnets</b>	object	<p>When omitted, the platform sets default values for the <b>joinSubnets</b> field of <b>100.65.0.0/16</b> for IPv4 and <b>fd99::/64</b> for IPv6. If the default address values are used anywhere in the cluster's network you must override it by setting the <b>joinSubnets</b> field. If you choose to set this field, ensure it does not conflict with other subnets in the cluster such as the cluster subnet, the <b>default</b> network cluster subnet, and the masquerade subnet.</p> <p>The <b>joinSubnets</b> field configures the routing between different segments within a user-defined network. Dual-stack clusters can set 2 subnets, one for each IP family; otherwise, only 1 subnet is allowed. This field is only allowed for the <b>Primary</b> network.</p>

<b>spec.network.&lt;topology&gt;.ipam.lifecycle</b>	<b>spec.&lt;topology&gt;.ipam.lifecycle</b>	object	<p>The <b>spec.ipam.lifecycle</b> field configures the IP address management system (IPAM). You might use this field for virtual workloads to ensure persistent IP addresses. The only allowed value is <b>Persistent</b>, which ensures that your virtual workloads have persistent IP addresses across reboots and migration. These are assigned by the container network interface (CNI) and used by OVN-Kubernetes to program pod IP addresses. You must not change this for pod annotations.</p> <p>Setting a value of Persistent is only supported when <b>ipam.mode</b> parameter is set to <b>Enabled</b>.</p>
-----------------------------------------------------	---------------------------------------------	--------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

<b>spec.network.&lt;topology&gt;.ipam.mode</b>	<b>spec.network.&lt;topology&gt;.ipam.mode</b>	object	<p>The <b>mode</b> parameter controls how much of the IP configuration is managed by OVN-Kubernetes. The following options are available:</p> <p><b>Enabled:</b> When enabled, OVN-Kubernetes applies the IP configuration to the SDN infrastructure and assigns IP addresses from the selected subnet to the individual pods. This is the default setting. When set to <b>Enabled</b>, the <b>subnets</b> field must be defined. <b>Enabled</b> is the default configuration.</p> <p><b>Disabled:</b> When disabled, OVN-Kubernetes only assigns MAC addresses and provides layer 2 communication, which allows users to configure IP addresses. <b>Disabled</b> is only available for layer 2 (secondary) networks. By disabling IPAM, features that rely on selecting pods by IP, for example, network policy, services, and so on, no longer function. Additionally, IP port security is also disabled for interfaces attached to this network. The <b>subnets</b> field must be empty when <b>spec.ipam.mode</b> is set to <b>Disabled</b>.</p>
<b>spec.network.&lt;topology&gt;.mtu</b>	<b>spec.&lt;topology&gt;.mtu</b>	integer	<p>The maximum transmission units (MTU). The default value is <b>1400</b>. The boundary for IPv4 is <b>576</b>, and for IPv6 it is <b>1280</b>.</p>

where:

### <topology>

Is one of **layer2** or **layer3**.

#### 16.2.1.7. User-defined network status condition types

The following tables explain the status condition types returned for **ClusterUserDefinedNetwork** and **UserDefinedNetwork** CRs when describing the resource. These conditions can be used to troubleshoot your deployment.

**Table 16.4. NetworkCreated condition types (ClusterDefinedNetwork and UserDefinedNetwork CRs)**

Condition type	Status	Reason and Message														
<b>NetworkCreated</b>	<b>True</b>	<p>When <b>True</b>, the following reason and message is returned:</p> <table border="1"> <thead> <tr> <th>Reason</th><th>Message</th></tr> </thead> <tbody> <tr> <td><b>NetworkAttachmentDefinitionCreated</b></td><td>'NetworkAttachmentDefinition has been created in following namespaces: [example-namespace-1, example-namespace-2, example-namespace-3]'</td></tr> </tbody> </table>	Reason	Message	<b>NetworkAttachmentDefinitionCreated</b>	'NetworkAttachmentDefinition has been created in following namespaces: [example-namespace-1, example-namespace-2, example-namespace-3]'										
Reason	Message															
<b>NetworkAttachmentDefinitionCreated</b>	'NetworkAttachmentDefinition has been created in following namespaces: [example-namespace-1, example-namespace-2, example-namespace-3]'															
<b>NetworkCreated</b>	<b>False</b>	<p>When <b>False</b>, one of the following messages is returned:</p> <table border="1"> <thead> <tr> <th>Reason</th><th>Message</th></tr> </thead> <tbody> <tr> <td><b>SyncError</b></td><td><b>failed to generate NetworkAttachmentDefinition</b></td></tr> <tr> <td><b>SyncError</b></td><td><b>failed to update NetworkAttachmentDefinition</b></td></tr> <tr> <td><b>SyncError</b></td><td><b>primary network already exist in namespace "&lt;namespace_name&gt;": "&lt;primary_network_name&gt;"</b></td></tr> <tr> <td><b>SyncError</b></td><td><b>failed to create NetworkAttachmentDefinition: create NAD error</b></td></tr> <tr> <td><b>SyncError</b></td><td><b>foreign NetworkAttachmentDefinition with the desired name already exist</b></td></tr> <tr> <td><b>SyncError</b></td><td><b>failed to add finalizer to UserDefinedNetwork</b></td></tr> </tbody> </table>	Reason	Message	<b>SyncError</b>	<b>failed to generate NetworkAttachmentDefinition</b>	<b>SyncError</b>	<b>failed to update NetworkAttachmentDefinition</b>	<b>SyncError</b>	<b>primary network already exist in namespace "&lt;namespace_name&gt;": "&lt;primary_network_name&gt;"</b>	<b>SyncError</b>	<b>failed to create NetworkAttachmentDefinition: create NAD error</b>	<b>SyncError</b>	<b>foreign NetworkAttachmentDefinition with the desired name already exist</b>	<b>SyncError</b>	<b>failed to add finalizer to UserDefinedNetwork</b>
Reason	Message															
<b>SyncError</b>	<b>failed to generate NetworkAttachmentDefinition</b>															
<b>SyncError</b>	<b>failed to update NetworkAttachmentDefinition</b>															
<b>SyncError</b>	<b>primary network already exist in namespace "&lt;namespace_name&gt;": "&lt;primary_network_name&gt;"</b>															
<b>SyncError</b>	<b>failed to create NetworkAttachmentDefinition: create NAD error</b>															
<b>SyncError</b>	<b>foreign NetworkAttachmentDefinition with the desired name already exist</b>															
<b>SyncError</b>	<b>failed to add finalizer to UserDefinedNetwork</b>															

Condition type	Status	Reason and Message	
		NetworkAttachmentDefinitionDeleted	NetworkAttachmentDefinition is being deleted: [<namespace>/<nad_name>]

Table 16.5. NetworkAllocationSucceeded condition types (UserDefinedNetwork CRs)

Condition type	Status	Reason and Message	
<b>NetworkAllocationSucceeded</b>	<b>True</b>	When <b>True</b> , the following reason and message is returned:	
		Reason	Message
		<b>NetworkAllocationSucceeded</b>	<b>Network allocation succeeded for all synced nodes.</b>
<b>NetworkAllocationSucceeded</b>	<b>False</b>	When <b>False</b> , the following message is returned:	
		Reason	Message
		<b>InternalError</b>	<b>Network allocation failed for at least one node: [&lt;node_name&gt;], check UDN events for more info.</b>

#### 16.2.1.8. Opening default network ports on user-defined network pods

By default, pods on a user-defined network (UDN) are isolated from the default network. This means that default network pods, such as those running monitoring services (Prometheus or Alertmanager) or the OpenShift Container Platform image registry, cannot initiate connections to UDN pods.

To allow default network pods to connect to a user-defined network pod, you can use the **k8s.ovn.org/open-default-ports** annotation. This annotation opens specific ports on the user-defined network pod for access from the default network.

The following pod specification allows incoming TCP connections on port **80** and UDP traffic on port **53** from the default network:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.ovn.org/open-default-ports: |
      - protocol: tcp
        port: 80
      - protocol: udp
        port: 53
    # ...
```

**NOTE**

Open ports are accessible on the pod’s default network IP, not its UDN network IP.

### 16.2.2. Creating primary networks using a NetworkAttachmentDefinition

The following sections explain how to create and manage primary networks using the **NetworkAttachmentDefinition** (NAD) resource.

#### 16.2.2.1. Approaches to managing a primary network

You can manage the life cycle of a primary network created by NAD with one of the following two approaches:

- By modifying the Cluster Network Operator (CNO) configuration. With this method, the CNO automatically creates and manages the **NetworkAttachmentDefinition** object. In addition to managing the object lifecycle, the CNO ensures that a DHCP is available for a primary network that uses a DHCP assigned IP address.
- By applying a YAML manifest. With this method, you can manage the primary network directly by creating an **NetworkAttachmentDefinition** object. This approach allows for the invocation of multiple CNI plugins in order to attach primary network interfaces in a pod.

Each approach is mutually exclusive and you can only use one approach for managing a primary network at a time. For either approach, the primary network is managed by a Container Network Interface (CNI) plugin that you configure.

**NOTE**

When deploying OpenShift Container Platform nodes with multiple network interfaces on Red Hat OpenStack Platform (RHOSP) with OVN SDN, DNS configuration of the secondary interface might take precedence over the DNS configuration of the primary interface. In this case, remove the DNS nameservers for the subnet ID that is attached to the secondary interface by running the following command:

```
$ openstack subnet set --dns-nameserver 0.0.0.0 <subnet_id>
```

#### 16.2.2.2. Creating a primary network attachment with the Cluster Network Operator

The Cluster Network Operator (CNO) manages additional network definitions. When you specify a primary network to create, the CNO creates the **NetworkAttachmentDefinition** CRD automatically.

**IMPORTANT**

Do not edit the **NetworkAttachmentDefinition** CRDs that the Cluster Network Operator manages. Doing so might disrupt network traffic on your primary network.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Optional: Create the namespace for the primary networks:

```
$ oc create namespace <namespace_name>
```

2. To edit the CNO configuration, enter the following command:

```
$ oc edit networks.operator.openshift.io cluster
```

3. Modify the CR that you are creating by adding the configuration for the primary network that you are creating, as in the following example CR.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
  additionalNetworks:
  - name: tertiary-net
    namespace: namespace2
    type: Raw
    rawCNIConfig: |-
      {
        "cniVersion": "0.3.1",
        "name": "tertiary-net",
        "type": "ipvlan",
        "master": "eth1",
        "mode": "I2",
        "ipam": {
          "type": "static",
          "addresses": [
            {
              "address": "192.168.1.23/24"
            }
          ]
        }
      }
```

4. Save your changes and quit the text editor to commit your changes.

## Verification

- Confirm that the CNO created the **NetworkAttachmentDefinition** CRD by running the following command. There might be a delay before the CNO creates the CRD.

```
$ oc get network-attachment-definitions -n <namespace>
```

where:

**<namespace>**

Specifies the namespace for the network attachment that you added to the CNO configuration.

## Example output

NAME	AGE
test-network-1	14m

### 16.2.2.2.1. Configuration for a primary network attachment

A primary network is configured by using the **NetworkAttachmentDefinition** API in the **k8s.cni.cncf.io** API group.

The configuration for the API is described in the following table:

**Table 16.6. NetworkAttachmentDefinition API fields**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	The name for the primary network.
<b>metadata.namespace</b>	<b>string</b>	The namespace that the object is associated with.
<b>spec.config</b>	<b>string</b>	The CNI plugin configuration in JSON format.

### 16.2.2.3. Creating a primary network attachment by applying a YAML manifest

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You are working in the namespace where the NAD is to be deployed.

#### Procedure

1. Create a YAML file with your primary network configuration, such as in the following example:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: next-net
spec:
  config: |-  

  {  

    "cniVersion": "0.3.1",  

    "name": "work-network",  

    "namespace": "namespace2", ①  

    "type": "host-device",  

    "device": "eth1",  

    "ipam": {  

      "type": "dhcp"  

    }  

  }
```

- 1** Optional: You can specify a namespace to which the NAD is applied. If you are working in the namespace where the NAD is to be deployed, this spec is not necessary.

2. To create the primary network, enter the following command:

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

where:

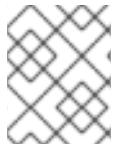
**<file>**

Specifies the name of the file contained the YAML manifest.

## 16.3. SECONDARY NETWORKS

### 16.3.1. Creating secondary networks on OVN-Kubernetes

As a cluster administrator, you can configure a secondary network for your cluster using the **NetworkAttachmentDefinition** (NAD) resource.



#### NOTE

Support for user-defined networks as a secondary network will be added in a future version of OpenShift Container Platform.

#### 16.3.1.1. Configuration for an OVN-Kubernetes secondary network

The Red Hat OpenShift Networking OVN-Kubernetes network plugin allows the configuration of secondary network interfaces for pods. To configure secondary network interfaces, you must define the configurations in the **NetworkAttachmentDefinition** custom resource definition (CRD).



#### NOTE

Pod and multi-network policy creation might remain in a pending state until the OVN-Kubernetes control plane agent in the nodes processes the associated **network-attachment-definition** CRD.

You can configure an OVN-Kubernetes secondary network in layer 2, layer 3, or localnet topologies. For more information about features supported on these topologies, see "UserDefinedNetwork and NetworkAttachmentDefinition support matrix".

The following sections provide example configurations for each of the topologies that OVN-Kubernetes currently allows for secondary networks.



#### NOTE

Networks names must be unique. For example, creating multiple **NetworkAttachmentDefinition** CRDs with different configurations that reference the same network is unsupported.

#### 16.3.1.1.1. Supported platforms for OVN-Kubernetes secondary network

You can use an OVN-Kubernetes secondary network with the following supported platforms:

- Bare metal
- IBM Power®
- IBM Z®
- IBM® LinuxONE
- VMware vSphere
- Red Hat OpenStack Platform (RHOSP)

#### 16.3.1.1.2. OVN-Kubernetes network plugin JSON configuration table

The following table describes the configuration parameters for the OVN-Kubernetes CNI network plugin:

**Table 16.7. OVN-Kubernetes network plugin JSON configuration table**

Field	Type	Description
<b>cniVersion</b>	<b>string</b>	The CNI specification version. The required value is <b>0.3.1</b> .
<b>name</b>	<b>string</b>	The name of the network. These networks are not namespaced. For example, a network named <b>I2-network</b> can be referenced by <b>NetworkAttachmentDefinition</b> custom resources (CRs) that exist in different namespaces. This configuration allows pods that use the <b>NetworkAttachmentDefinition</b> CR in different namespaces to communicate over the same secondary network. However, the <b>NetworkAttachmentDefinition</b> CRs must share the same network-specific parameters, such as <b>topology</b> , <b>subnets</b> , <b>mtu</b> , <b>excludeSubnets</b> , and <b>vlanID</b> . The <b>vlanID</b> parameter applies only when the <b>topology</b> field is set to <b>localnet</b> .
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure. This value must be set to <b>ovn-k8s-cni-overlay</b> .
<b>topology</b>	<b>string</b>	The topological configuration for the network. Must be one of <b>layer2</b> or <b>localnet</b> .
<b>subnets</b>	<b>string</b>	<p>The subnet to use for the network across the cluster.</p> <p>For "<b>topology": "layer2</b>" deployments, IPv6 (<b>2001:DBB::/64</b>) and dual-stack (<b>192.168.100.0/24,2001:DBB::/64</b>) subnets are supported.</p> <p>When omitted, the logical switch implementing the network only provides layer 2 communication, and users must configure IP addresses for the pods. Port security only prevents MAC spoofing.</p>
<b>mtu</b>	<b>string</b>	The maximum transmission unit (MTU). The default value, <b>1300</b> , is automatically set by the kernel.

Field	Type	Description
<b>netAttachDefName</b>	<b>string</b>	The metadata <b>namespace</b> and <b>name</b> of the network attachment definition CRD where this configuration is included. For example, if this configuration is defined in a <b>NetworkAttachmentDefinition</b> CRD in namespace <b>ns1</b> named <b>I2-network</b> , this should be set to <b>ns1/I2-network</b> .
<b>excludeSubnets</b>	<b>string</b>	A comma-separated list of CIDRs and IP addresses. IP addresses are removed from the assignable IP address pool and are never passed to the pods.
<b>vlanID</b>	<b>integer</b>	If topology is set to <b>localnet</b> , the specified VLAN tag is assigned to traffic from this secondary network. The default is to not assign a VLAN tag.

#### 16.3.1.1.3. Compatibility with multi-network policy

The multi-network policy API, which is provided by the **MultiNetworkPolicy** custom resource definition (CRD) in the **k8s.cni.cncf.io** API group, is compatible with an OVN-Kubernetes secondary network. When defining a network policy, the network policy rules that can be used depend on whether the OVN-Kubernetes secondary network defines the **subnets** field. Refer to the following table for details:

Table 16.8. Supported multi-network policy selectors based on **subnets** CNI configuration

<b>subnets</b> field specified	Allowed multi-network policy selectors
Yes	<ul style="list-style-type: none"> <li>• <b>podSelector</b> and <b>namespaceSelector</b></li> <li>• <b>ipBlock</b></li> </ul>
No	<ul style="list-style-type: none"> <li>• <b>ipBlock</b></li> </ul>

For example, the following multi-network policy is valid only if the **subnets** field is defined in the secondary network CNI configuration for the secondary network named **blue2**:

#### Example multi-network policy that uses a pod selector

```

apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: blue2
spec:
  podSelector:
  ingress:
    - from:
      - podSelector: {}

```

The following example uses the **ipBlock** network policy selector, which is always valid for an OVN-Kubernetes secondary network:

### Example multi-network policy that uses an IP block selector

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: ingress-ipblock
  annotations:
    k8s.v1.cni.cncf.io/policy-for: default/flatl2net
spec:
  podSelector:
    matchLabels:
      name: access-control
  policyTypes:
    - Ingress
  ingress:
    - from:
        - ipBlock:
            cidr: 10.200.0.0/30
```

#### 16.3.1.1.4. Configuration for a localnet switched topology

The switched **localnet** topology interconnects the workloads created as Network Attachment Definitions (NADs) through a cluster-wide logical switch to a physical network.

You must map a secondary network to the OVN bridge to use it as an OVN-Kubernetes secondary network. Bridge mappings allow network traffic to reach the physical network. A bridge mapping associates a physical network name, also known as an interface label, to a bridge created with Open vSwitch (OVS).

You can create an **NodeNetworkConfigurationPolicy** (NNCP) object, part of the **nmstate.io/v1** API group, to declaratively create the mapping. This API is provided by the NMState Operator. By using this API you can apply the bridge mapping to nodes that match your specified **nodeSelector** expression, such as **node-role.kubernetes.io/worker: "**. With this declarative approach, the NMState Operator applies secondary network configuration to all nodes specified by the node selector automatically and transparently.

When attaching a secondary network, you can either use the existing **br-ex** bridge or create a new bridge. Which approach to use depends on your specific network infrastructure. Consider the following approaches:

- If your nodes include only a single network interface, you must use the existing bridge. This network interface is owned and managed by OVN-Kubernetes and you must not remove it from the **br-ex** bridge or alter the interface configuration. If you remove or alter the network interface, your cluster network will stop working correctly.
- If your nodes include several network interfaces, you can attach a different network interface to a new bridge, and use that for your secondary network. This approach provides for traffic isolation from your primary cluster network.

The **localnet1** network is mapped to the **br-ex** bridge in the following example:

### Example mapping for sharing a bridge

```

apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: mapping ①
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: "②"
  desiredState:
    ovn:
      bridge-mappings:
        - localnet: localnet1 ③
          bridge: br-ex ④
          state: present ⑤

```

- ① The name for the configuration object.
- ② A node selector that specifies the nodes to apply the node network configuration policy to.
- ③ The name for the secondary network from which traffic is forwarded to the OVS bridge. This secondary network must match the name of the **spec.config.name** field of the **NetworkAttachmentDefinition** CRD that defines the OVN-Kubernetes secondary network.
- ④ The name of the OVS bridge on the node. This value is required only if you specify **state: present**.
- ⑤ The state for the mapping. Must be either **present** to add the bridge or **absent** to remove the bridge. The default value is **present**.

The following JSON example configures a localnet secondary network that is named **localnet1**:

```
{
  "cniVersion": "0.3.1",
  "name": "ns1-localnet-network",
  "type": "ovn-k8s-cni-overlay",
  "topology": "localnet",
  "physicalNetworkName": "localnet1",
  "subnets": "202.10.130.112/28",
  "vlanID": 33,
  "mtu": 1500,
  "netAttachDefName": "ns1/localnet-network",
  "excludeSubnets": "10.100.200.0/29"
}
```

In the following example, the **localnet2** network interface is attached to the **ovs-br1** bridge. Through this attachment, the network interface is available to the OVN-Kubernetes network plugin as a secondary network.

### Example mapping for nodes with multiple interfaces

```

apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ovs-br1-multiple-networks ①
spec:
  nodeSelector:

```

```

node-role.kubernetes.io/worker: "2"
desiredState:
  interfaces:
    - name: ovs-br1 3
      description: |-  

        A dedicated OVS bridge with eth1 as a port  

        allowing all VLANs and untagged traffic
      type: ovs-bridge
      state: up
      bridge:
        allow-extra-patch-ports: true
        options:
          stp: false
          mcast-snooping-enable: true 4
      port:
        - name: eth1 5
  ovn:
    bridge-mappings:
      - localnet: localnet2 6
        bridge: ovs-br1 7
        state: present 8

```

- 1 Specifies the name of the configuration object.
- 2 Specifies a node selector that identifies the nodes to which the node network configuration policy applies.
- 3 Specifies a new OVS bridge that operates separately from the default bridge used by OVN-Kubernetes for cluster traffic.
- 4 Specifies whether to enable multicast snooping. When enabled, multicast snooping prevents network devices from flooding multicast traffic to all network members. By default, an OVS bridge does not enable multicast snooping. The default value is **false**.
- 5 Specifies the network device on the host system to associate with the new OVS bridge.
- 6 Specifies the name of the secondary network that forwards traffic to the OVS bridge. This name must match the value of the **spec.config.name** field in the **NetworkAttachmentDefinition** CRD that defines the OVN-Kubernetes secondary network.
- 7 Specifies the name of the OVS bridge on the node. The value is required only when **state: present** is set.
- 8 Specifies the state of the mapping. Valid values are **present** to add the bridge or **absent** to remove the bridge. The default value is **present**.

The following JSON example configures a localnet secondary network that is named **localnet2**:

```
{
  "cniVersion": "0.3.1",
  "name": "ns1-localnet-network",
  "type": "ovn-k8s-cni-overlay",
  "topology": "localnet",
  "physicalNetworkName": "localnet2",
  "subnets": "202.10.130.112/28",
```

```

    "vlanID": 33,
    "mtu": 1500,
    "netAttachDefName": "ns1/localnet-network",
    "excludeSubnets": "10.100.200.0/29"
}

```

#### 16.3.1.4.1. Configuration for a layer 2 switched topology

The switched (layer 2) topology networks interconnect the workloads through a cluster-wide logical switch. This configuration can be used for IPv6 and dual-stack deployments.



#### NOTE

Layer 2 switched topology networks only allow for the transfer of data packets between pods within a cluster.

The following JSON example configures a switched secondary network:

```

{
  "cniVersion": "0.3.1",
  "name": "l2-network",
  "type": "ovn-k8s-cni-overlay",
  "topology": "layer2",
  "subnets": "10.100.200.0/24",
  "mtu": 1300,
  "netAttachDefName": "ns1/l2-network",
  "excludeSubnets": "10.100.200.0/29"
}

```

#### 16.3.1.5. Configuring pods for secondary networks

You must specify the secondary network attachments through the **k8s.v1.cni.cncf.io/networks** annotation.

The following example provisions a pod with two secondary attachments, one for each of the attachment configurations presented in this guide.

```

apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: l2-network
  name: tinypod
  namespace: ns1
spec:
  containers:
    - args:
      - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container

```

### 16.3.1.1.6. Configuring pods with a static IP address

The following example provisions a pod with a static IP address.



#### NOTE

- You can specify the IP address for the secondary network attachment of a pod only when the secondary network attachment, a namespaced-scoped object, uses a layer 2 or localnet topology.
- Specifying a static IP address for the pod is only possible when the attachment configuration does not feature subnets.

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: [
      {
        "name": "l2-network", 1
        "mac": "02:03:04:05:06:07", 2
        "interface": "myiface1", 3
        "ips": [
          "192.0.2.20/24"
        ] 4
      }
    ]
  name: tinypod
  namespace: ns1
spec:
  containers:
    - args:
        - pause
      image: k8s.gcr.io/e2e-test-images/agnhost:2.36
      imagePullPolicy: IfNotPresent
      name: agnhost-container
```

- 1** The name of the network. This value must be unique across all **NetworkAttachmentDefinition** CRDs.
- 2** The MAC address to be assigned for the interface.
- 3** The name of the network interface to be created for the pod.
- 4** The IP addresses to be assigned to the network interface.

### 16.3.2. Creating secondary networks with other CNI plugins

The specific configuration fields for secondary networks are described in the following sections.

#### 16.3.2.1. Configuration for a bridge secondary network

The following object describes the configuration parameters for the Bridge CNI plugin:

Table 16.9. Bridge CNI plugin JSON configuration object

Field	Type	Description
<b>cniVersion</b>	<b>string</b>	The CNI specification version. The <b>0.3.1</b> value is required.
<b>name</b>	<b>string</b>	The value for the <b>name</b> parameter you provided previously for the CNO configuration.
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure: <b>bridge</b> .
<b>ipam</b>	<b>object</b>	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition.
<b>bridge</b>	<b>string</b>	Optional: Specify the name of the virtual bridge to use. If the bridge interface does not exist on the host, it is created. The default value is <b>cnio</b> .
<b>ipMasq</b>	<b>boolean</b>	Optional: Set to <b>true</b> to enable IP masquerading for traffic that leaves the virtual network. The source IP address for all traffic is rewritten to the bridge's IP address. If the bridge does not have an IP address, this setting has no effect. The default value is <b>false</b> .
<b>isGateway</b>	<b>boolean</b>	Optional: Set to <b>true</b> to assign an IP address to the bridge. The default value is <b>false</b> .
<b>isDefaultGateway</b>	<b>boolean</b>	Optional: Set to <b>true</b> to configure the bridge as the default gateway for the virtual network. The default value is <b>false</b> . If <b>isDefaultGateway</b> is set to <b>true</b> , then <b>isGateway</b> is also set to <b>true</b> automatically.
<b>forceAddress</b>	<b>boolean</b>	Optional: Set to <b>true</b> to allow assignment of a previously assigned IP address to the virtual bridge. When set to <b>false</b> , if an IPv4 address or an IPv6 address from overlapping subsets is assigned to the virtual bridge, an error occurs. The default value is <b>false</b> .
<b>hairpinMode</b>	<b>boolean</b>	Optional: Set to <b>true</b> to allow the virtual bridge to send an Ethernet frame back through the virtual port it was received on. This mode is also known as <i>reflective relay</i> . The default value is <b>false</b> .
<b>promiscMode</b>	<b>boolean</b>	Optional: Set to <b>true</b> to enable promiscuous mode on the bridge. The default value is <b>false</b> .
<b>vlan</b>	<b>string</b>	Optional: Specify a virtual LAN (VLAN) tag as an integer value. By default, no VLAN tag is assigned.
<b>preserveDefaultVlan</b>	<b>string</b>	Optional: Indicates whether the default vlan must be preserved on the <b>veth</b> end connected to the bridge. Defaults to true.

Field	Type	Description
<b>vlanTrunk</b>	<b>list</b>	Optional: Assign a VLAN trunk tag. The default value is <b>none</b> .
<b>mtu</b>	<b>integer</b>	Optional: Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
<b>enabledad</b>	<b>boolean</b>	Optional: Enables duplicate address detection for the container side <b>veth</b> . The default value is <b>false</b> .
<b>macspoofchk</b>	<b>boolean</b>	Optional: Enables mac spoof check, limiting the traffic originating from the container to the mac address of the interface. The default value is <b>false</b> .



#### NOTE

The VLAN parameter configures the VLAN tag on the host end of the **veth** and also enables the **vlan\_filtering** feature on the bridge interface.



#### NOTE

To configure an uplink for an L2 network, you must allow the VLAN on the uplink interface by using the following command:

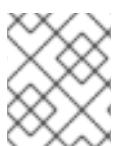
```
$ bridge vlan add vid VLAN_ID dev DEV
```

#### 16.3.2.1.1. Bridge CNI plugin configuration example

The following example configures a secondary network named **bridge-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "bridge-net",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
  }
}
```

#### 16.3.2.2. Configuration for a host device secondary network



#### NOTE

Specify your network device by setting only one of the following parameters: **device**, **hwaddr**, **kernelpath**, or **pciBusID**.

The following object describes the configuration parameters for the host-device CNI plugin:

**Table 16.10. Host device CNI plugin JSON configuration object**

Field	Type	Description
<b>cniVersion</b>	<b>string</b>	The CNI specification version. The <b>0.3.1</b> value is required.
<b>name</b>	<b>string</b>	The value for the <b>name</b> parameter you provided previously for the CNO configuration.
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure: <b>host-device</b> .
<b>device</b>	<b>string</b>	Optional: The name of the device, such as <b>eth0</b> .
<b>hwaddr</b>	<b>string</b>	Optional: The device hardware MAC address.
<b>kernelpath</b>	<b>string</b>	Optional: The Linux kernel device path, such as <b>/sys/devices/pci0000:00/0000:00:1f.6</b> .
<b>pciBusID</b>	<b>string</b>	Optional: The PCI address of the network device, such as <b>0000:00:1f.6</b> .

#### 16.3.2.2.1. host-device configuration example

The following example configures a secondary network named **hostdev-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "hostdev-net",
  "type": "host-device",
  "device": "eth1"
}
```

#### 16.3.2.3. Configuration for a VLAN secondary network

The following object describes the configuration parameters for the VLAN, **vlan**, CNI plugin:

**Table 16.11. VLAN CNI plugin JSON configuration object**

Field	Type	Description
<b>cniVersion</b>	<b>string</b>	The CNI specification version. The <b>0.3.1</b> value is required.
<b>name</b>	<b>string</b>	The value for the <b>name</b> parameter you provided previously for the CNO configuration.
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure: <b>vlan</b> .

Field	Type	Description
<b>master</b>	<b>string</b>	The Ethernet interface to associate with the network attachment. If a <b>master</b> is not specified, the interface for the default network route is used.
<b>vlanId</b>	<b>integer</b>	Set the ID of the <b>vlan</b> .
<b>ipam</b>	<b>object</b>	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition.
<b>mtu</b>	<b>integer</b>	Optional: Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
<b>dns</b>	<b>integer</b>	Optional: DNS information to return. For example, a priority-ordered list of DNS nameservers.
<b>linkInContainer</b>	<b>boolean</b>	Optional: Specifies whether the <b>master</b> interface is in the container network namespace or the main network namespace. Set the value to <b>true</b> to request the use of a container namespace <b>master</b> interface.



## IMPORTANT

A **NetworkAttachmentDefinition** custom resource definition (CRD) with a **vlan** configuration can be used only on a single pod in a node because the CNI plugin cannot create multiple **vlan** subinterfaces with the same **vlanId** on the same **master** interface.

### 16.3.2.3.1. VLAN configuration example

The following example demonstrates a **vlan** configuration with a secondary network that is named **vlan-net**:

```
{
  "name": "vlan-net",
  "cniVersion": "0.3.1",
  "type": "vlan",
  "master": "eth0",
  "mtu": 1500,
  "vlanId": 5,
  "linkInContainer": false,
  "ipam": {
    "type": "host-local",
    "subnet": "10.1.1.0/24"
  },
  "dns": {
    "nameservers": [ "10.1.1.1", "8.8.8.8" ]
  }
}
```

#### 16.3.2.4. Configuration for an IPVLAN secondary network

The following object describes the configuration parameters for the IPVLAN, **ipvlan**, CNI plugin:

**Table 16.12. IPVLAN CNI plugin JSON configuration object**

Field	Type	Description
<b>cniVersion</b>	<b>string</b>	The CNI specification version. The <b>0.3.1</b> value is required.
<b>name</b>	<b>string</b>	The value for the <b>name</b> parameter you provided previously for the CNO configuration.
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure: <b>ipvlan</b> .
<b>ipam</b>	<b>object</b>	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition. This is required unless the plugin is chained.
<b>mode</b>	<b>string</b>	Optional: The operating mode for the virtual network. The value must be <b>I2</b> , <b>I3</b> , or <b>I3s</b> . The default value is <b>I2</b> .
<b>master</b>	<b>string</b>	Optional: The Ethernet interface to associate with the network attachment. If a <b>master</b> is not specified, the interface for the default network route is used.
<b>mtu</b>	<b>integer</b>	Optional: Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
<b>linkInContainer</b>	<b>boolean</b>	Optional: Specifies whether the <b>master</b> interface is in the container network namespace or the main network namespace. Set the value to <b>true</b> to request the use of a container namespace <b>master</b> interface.



#### IMPORTANT

- The **ipvlan** object does not allow virtual interfaces to communicate with the **master** interface. Therefore the container is not able to reach the host by using the **ipvlan** interface. Be sure that the container joins a network that provides connectivity to the host, such as a network supporting the Precision Time Protocol (**PTP**).
- A single **master** interface cannot simultaneously be configured to use both **macvlan** and **ipvlan**.
- For IP allocation schemes that cannot be interface agnostic, the **ipvlan** plugin can be chained with an earlier plugin that handles this logic. If the **master** is omitted, then the previous result must contain a single interface name for the **ipvlan** plugin to enslave. If **ipam** is omitted, then the previous result is used to configure the **ipvlan** interface.

#### 16.3.2.4.1. IPVLAN CNI plugin configuration example

The following example configures a secondary network named **ipvlan-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "ipvlan-net",
  "type": "ipvlan",
  "master": "eth1",
  "linkInContainer": false,
  "mode": "l3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

#### 16.3.2.5. Configuration for a MACVLAN secondary network

The following object describes the configuration parameters for the MAC Virtual LAN (MACVLAN) Container Network Interface (CNI) plugin:

**Table 16.13. MACVLAN CNI plugin JSON configuration object**

Field	Type	Description
<b>cniVersion</b>	<b>string</b>	The CNI specification version. The <b>0.3.1</b> value is required.
<b>name</b>	<b>string</b>	The value for the <b>name</b> parameter you provided previously for the CNO configuration.
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure: <b>macvlan</b> .
<b>ipam</b>	<b>object</b>	The configuration object for the IPAM CNI plugin. The plugin manages IP address assignment for the attachment definition.
<b>mode</b>	<b>string</b>	Optional: Configures traffic visibility on the virtual network. Must be either <b>bridge</b> , <b>passthru</b> , <b>private</b> , or <b>vepa</b> . If a value is not provided, the default value is <b>bridge</b> .
<b>master</b>	<b>string</b>	Optional: The host network interface to associate with the newly created macvlan interface. If a value is not specified, then the default route interface is used.
<b>mtu</b>	<b>integer</b>	Optional: The maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.

Field	Type	Description
<b>linkInContainer</b>	<b>boolean</b>	Optional: Specifies whether the <b>master</b> interface is in the container network namespace or the main network namespace. Set the value to <b>true</b> to request the use of a container namespace <b>master</b> interface.

**NOTE**

If you specify the **master** key for the plugin configuration, use a different physical network interface than the one that is associated with your primary network plugin to avoid possible conflicts.

#### 16.3.2.5.1. MACVLAN CNI plugin configuration example

The following example configures a secondary network named **macvlan-net**:

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
  "linkInContainer": false,
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
  }
}
```

#### 16.3.2.6. Configuration for a TAP secondary network

The following object describes the configuration parameters for the TAP CNI plugin:

Table 16.14. TAP CNI plugin JSON configuration object

Field	Type	Description
<b>cniVersion</b>	<b>string</b>	The CNI specification version. The <b>0.3.1</b> value is required.
<b>name</b>	<b>string</b>	The value for the <b>name</b> parameter you provided previously for the CNO configuration.
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure: <b>tap</b> .
<b>mac</b>	<b>string</b>	Optional: Request the specified MAC address for the interface.

Field	Type	Description
<b>mtu</b>	<b>integer</b>	Optional: Set the maximum transmission unit (MTU) to the specified value. The default value is automatically set by the kernel.
<b>selinuxcontext</b>	<b>string</b>	Optional: The SELinux context to associate with the tap device.   <b>NOTE</b> The value <b>system_u:system_r:container_t:s0</b> is required for OpenShift Container Platform.
<b>multiQueue</b>	<b>boolean</b>	Optional: Set to <b>true</b> to enable multi-queue.
<b>owner</b>	<b>integer</b>	Optional: The user owning the tap device.
<b>group</b>	<b>integer</b>	Optional: The group owning the tap device.
<b>bridge</b>	<b>string</b>	Optional: Set the tap device as a port of an already existing bridge.

#### 16.3.2.6.1. Tap configuration example

The following example configures a secondary network named **mynet**:

```
{
  "name": "mynet",
  "cniVersion": "0.3.1",
  "type": "tap",
  "mac": "00:11:22:33:44:55",
  "mtu": 1500,
  "selinuxcontext": "system_u:system_r:container_t:s0",
  "multiQueue": true,
  "owner": 0,
  "group": 0
  "bridge": "br1"
}
```

#### 16.3.2.6.2. Setting SELinux boolean for the TAP CNI plugin

To create the tap device with the **container\_t** SELinux context, enable the **container\_use\_devices** boolean on the host by using the Machine Config Operator (MCO).

#### Prerequisites

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

#### Procedure

1. Create a new YAML file named, such as **setsebool-container-use-devices.yaml**, with the following details:

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-setsebool
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - enabled: true
          name: setsebool.service
          contents: |
            [Unit]
            Description=Set SELinux boolean for the TAP CNI plugin
            Before=kubelet.service

            [Service]
            Type=oneshot
            ExecStart=/usr/sbin/setsebool container_use_devices=on
            RemainAfterExit=true

            [Install]
            WantedBy=multi-user.target graphical.target

```

2. Create the new **MachineConfig** object by running the following command:

```
$ oc apply -f setsebool-container-use-devices.yaml
```



#### NOTE

Applying any changes to the **MachineConfig** object causes all affected nodes to gracefully reboot after the change is applied. This update can take some time to be applied.

3. Verify the change is applied by running the following command:

```
$ oc get machineconfigpools
```

#### Expected output

	NAME	CONFIG	UPDATED	UPDATING	DEGRADED
	MACHINECOUNT	READYMACHINECOUNT	UPDATEDMACHINECOUNT		
	DEGRADED	MACHINECOUNT	AGE		
master	rendered-master-e5e0c8e8be9194e7c5a882e047379dfa False	3	3	0	7d2h
worker	rendered-worker-d6c9ca107fba6cd76cdcbfcfedcafa0f2 3	3	3	0	7d

**NOTE**

All nodes should be in the updated and ready state.

### 16.3.2.7. Configuring routes using the route-override plugin on a secondary network

The following object describes the configuration parameters for the **route-override** CNI plugin:

**Table 16.15. Route override CNI plugin JSON configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The name of the CNI plugin to configure: <b>route-override</b> .
<b>flushroutes</b>	<b>boolean</b>	Optional: Set to <b>true</b> to flush any existing routes.
<b>flushgateway</b>	<b>boolean</b>	Optional: Set to <b>true</b> to flush the default route namely the gateway route.
<b>delroutes</b>	<b>object</b>	Optional: Specify the list of routes to delete from the container namespace.
<b>addroutes</b>	<b>object</b>	Optional: Specify the list of routes to add to the container namespace. Each route is a dictionary with <b>dst</b> and optional <b>gw</b> fields. If <b>gw</b> is omitted, the plugin uses the default gateway value.
<b>skipcheck</b>	<b>boolean</b>	Optional: Set this to <b>true</b> to skip the check command. By default, CNI plugins verify the network setup during the container lifecycle. When modifying routes dynamically with <b>route-override</b> , skipping this check ensures the final configuration reflects the updated routes.

#### 16.3.2.7.1. Route-override plugin configuration example

The **route-override** CNI is a type of CNI that it is designed to be used when chained with a parent CNI. It does not operate independently, but relies on the parent CNI to first create the network interface and assign IP addresses before it can modify the routing rules.

The following example configures a secondary network named **mymacvlan**. The parent CNI creates a network interface attached to **eth1** and assigns an IP address in the **192.168.1.0/24** range using **host-local** IPAM. The **route-override** CNI is then chained to the parent CNI and modifies the routing rules by flushing existing routes, deleting the route to **192.168.0.0/24**, and adding a new route for **192.168.0.0/24** with a custom gateway.

```
{
  "cniVersion": "0.3.0",
  "name": "mymacvlan",
  "plugins": [
    {
      "type": "macvlan",
      "master": "eth1",
      "linkLocal": true
    }
  ]
}
```

1

```

    "mode": "bridge",
    "ipam": {
        "type": "host-local",
        "subnet": "192.168.1.0/24"
    }
},
{
    "type": "route-override", ②
    "flushroutes": true,
    "delroutes": [
        {
            "dst": "192.168.0.0/24"
        }
    ],
    "addroutes": [
        {
            "dst": "192.168.0.0/24",
            "gw": "10.1.254.254"
        }
    ]
}
]
}

```

- ① The parent CNI creates a network interface attached to **eth1**.  
 ② The chained **route-override** CNI modifies the routing rules.

## Additional resources

- For more information about enabling an SELinux boolean on a node, see [Setting SELinux booleans](#).

### 16.3.3. Attaching a pod to a secondary network

As a cluster user you can attach a pod to a secondary network.

#### 16.3.3.1. Adding a pod to a secondary network

You can add a pod to a secondary network. The pod continues to send normal cluster-related network traffic over the default network.

When a pod is created, a secondary networks is attached to the pod. However, if a pod already exists, you cannot attach a secondary network to it.

The pod must be in the same namespace as the secondary network.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

#### Procedure

- Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

- a. To attach a secondary network without any customization, add an annotation with the following format. Replace <network> with the name of the secondary network to associate with the pod:

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

- 1** To specify more than one secondary network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same secondary network multiple times, that pod will have multiple network interfaces attached to that network.

- b. To attach a secondary network with customizations, add an annotation with the following format:

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-|
      [
        {
          "name": "<network>", 1
          "namespace": "<namespace>", 2
          "default-route": ["<default-route>"] 3
        }
      ]
```

- 1** Specify the name of the secondary network defined by a **NetworkAttachmentDefinition** object.
- 2** Specify the namespace where the **NetworkAttachmentDefinition** object is defined.
- 3** Optional: Specify an override for the default route, such as **192.168.17.1**.

- To create the pod, enter the following command. Replace <name> with the name of the pod.

```
$ oc create -f <name>.yaml
```

- Optional: To Confirm that the annotation exists in the **Pod** CR, enter the following command, replacing <name> with the name of the pod.

```
$ oc get pod <name> -o yaml
```

In the following example, the **example-pod** pod is attached to the **net1** secondary network:

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
```

```
k8s.v1.cni.cncf.io/network-status: |- ①
  [
    {
      "name": "ovn-kubernetes",
      "interface": "eth0",
      "ips": [
        "10.128.2.14"
      ],
      "default": true,
      "dns": {}
    },
    {
      "name": "macvlan-bridge",
      "interface": "net1",
      "ips": [
        "20.2.2.100"
      ],
      "mac": "22:2f:60:a5:f8:00",
      "dns": {}
    }
  ]
name: example-pod
namespace: default
spec:
...
status:
...
```

- ① The **k8s.v1.cni.cncf.io/network-status** parameter is a JSON array of objects. Each object describes the status of a secondary network attached to the pod. The annotation value is stored as a plain text value.

#### 16.3.3.1.1. Specifying pod-specific addressing and routing options

When attaching a pod to a secondary network, you may want to specify further properties about that network in a particular pod. This allows you to change some aspects of routing, as well as specify static IP addresses and MAC addresses. To accomplish this, you can use the JSON formatted annotations.

#### Prerequisites

- The pod must be in the same namespace as the secondary network.
- Install the OpenShift CLI (**oc**).
- You must log in to the cluster.

#### Procedure

To add a pod to a secondary network while specifying addressing and/or routing options, complete the following steps:

1. Edit the **Pod** resource definition. If you are editing an existing **Pod** resource, run the following command to edit its definition in the default editor. Replace **<name>** with the name of the **Pod** resource to edit.

```
$ oc edit pod <name>
```

2. In the **Pod** resource definition, add the **k8s.v1.cni.cncf.io/networks** parameter to the pod

**metadata** mapping. The **k8s.v1.cni.cncf.io/networks** accepts a JSON string of a list of objects that reference the name of **NetworkAttachmentDefinition** custom resource (CR) names in addition to specifying additional properties.

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[<network>[,<network>,...]]' 1
```

- 1** Replace **<network>** with a JSON object as shown in the following examples. The single quotes are required.

3. In the following example the annotation specifies which network attachment will have the default route, using the **default-route** parameter.

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
annotations:
  k8s.v1.cni.cncf.io/networks: '[
    {
      "name": "net1"
    },
    {
      "name": "net2", 1
      "default-route": ["192.0.2.1"] 2
    }
  ]'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

- 1** The **name** key is the name of the secondary network to associate with the pod.
- 2** The **default-route** key specifies a value of a gateway for traffic to be routed over if no other routing entry is present in the routing table. If more than one **default-route** key is specified, this will cause the pod to fail to become active.

The default route will cause any traffic that is not specified in other routes to be routed to the gateway.



## IMPORTANT

Setting the default route to an interface other than the default network interface for OpenShift Container Platform may cause traffic that is anticipated for pod-to-pod traffic to be routed over another interface.

To verify the routing properties of a pod, the **oc** command may be used to execute the **ip** command within a pod.

```
$ oc exec -it <pod_name> -- ip route
```



## NOTE

You may also reference the pod's **k8s.v1.cni.cncf.io/network-status** to see which secondary network has been assigned the default route, by the presence of the **default-route** key in the JSON-formatted list of objects.

To set a static IP address or MAC address for a pod you can use the JSON formatted annotations. This requires you create networks that specifically allow for this functionality. This can be specified in a rawCNIConfig for the CNO.

1. Edit the CNO CR by running the following command:

```
$ oc edit networks.operator.openshift.io cluster
```

The following YAML describes the configuration parameters for the CNO:

### Cluster Network Operator YAML configuration

```
name: <name> ①
namespace: <namespace> ②
rawCNIConfig: '{ ③
...
}'
type: Raw
```

- ① Specify a name for the secondary network attachment that you are creating. The name must be unique within the specified **namespace**.
- ② Specify the namespace to create the network attachment in. If you do not specify a value, then the **default** namespace is used.
- ③ Specify the CNI plugin configuration in JSON format, which is based on the following template.

The following object describes the configuration parameters for utilizing static MAC address and IP address using the macvlan CNI plugin:

### macvlan CNI plugin JSON configuration object using static IP and MAC address

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", ①
  "plugins": [{ ②
    "type": "macvlan",
    "capabilities": { "ips": true }, ③
    "master": "eth0", ④
    "mode": "bridge",
    "ipam": {
      "type": "static"
    }
  }, {
    "capabilities": { "mac": true }, ⑤
  }
```

```

        "type": "tuning"
    }]
}

```

- 1 Specifies the name for the secondary network attachment to create. The name must be unique within the specified **namespace**.
- 2 Specifies an array of CNI plugin configurations. The first object specifies a macvlan plugin configuration and the second object specifies a tuning plugin configuration.
- 3 Specifies that a request is made to enable the static IP address functionality of the CNI plugin runtime configuration capabilities.
- 4 Specifies the interface that the macvlan plugin uses.
- 5 Specifies that a request is made to enable the static MAC address functionality of a CNI plugin.

The above network attachment can be referenced in a JSON formatted annotation, along with keys to specify which static IP and MAC address will be assigned to a given pod.

Edit the pod with:

```
$ oc edit pod <name>
```

### macvlan CNI plugin JSON configuration object using static IP and MAC address

```

apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "<name>", ①
        "ips": [ "192.0.2.205/24" ], ②
        "mac": "CA:FE:C0:FF:EE:00" ③
      }
    ]'

```

- 1 Use the **<name>** as provided when creating the **rawCNIConfig** above.
- 2 Provide an IP address including the subnet mask.
- 3 Provide the MAC address.



#### NOTE

Static IP addresses and MAC addresses do not have to be used at the same time, you may use them individually, or together.

To verify the IP address and MAC properties of a pod with secondary networks, use the **oc** command to execute the **ip** command within a pod.

```
$ oc exec -it <pod_name> -- ip a
```

### 16.3.4. Configuring multi-network policy

Administrators can use the **MultiNetworkPolicy** API to create multiple network policies that manage traffic for pods attached to secondary networks. For example, you can create policies that allow or deny traffic based on specific ports, IPs/ranges, or labels.

Multi-network policies can be used to manage traffic on secondary networks in the cluster. These policies cannot manage the default cluster network or primary network of user-defined networks.

As a cluster administrator, you can configure a multi-network policy for any of the following network types:

- Single-Root I/O Virtualization (SR-IOV)
- MAC Virtual Local Area Network (MacVLAN)
- IP Virtual Local Area Network (IPVLAN)
- Bond Container Network Interface (CNI) over SR-IOV
- OVN-Kubernetes secondary networks



#### NOTE

Support for configuring multi-network policies for SR-IOV secondary networks is only supported with kernel network interface controllers (NICs). SR-IOV is not supported for Data Plane Development Kit (DPDK) applications.

#### 16.3.4.1. Differences between multi-network policy and network policy

Although the **MultiNetworkPolicy** API implements the **NetworkPolicy** API, there are several important differences:

- You must use the **MultiNetworkPolicy** API:

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
```

- You must use the **multi-networkpolicy** resource name when using the CLI to interact with multi-network policies. For example, you can view a multi-network policy object with the **oc get multi-networkpolicy <name>** command where **<name>** is the name of a multi-network policy.
- You must specify an annotation with the name of the network attachment definition that defines the secondary network:

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
```

where:

**<network\_name>**

Specifies the name of a network attachment definition.

#### 16.3.4.2. Enabling multi-network policy for the cluster

As a cluster administrator, you can enable multi-network policy support on your cluster.

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster with a user with **cluster-admin** privileges.

##### Procedure

1. Create the **multinetwork-enable-patch.yaml** file with the following YAML:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  useMultiNetworkPolicy: true
```

2. Configure the cluster to enable multi-network policy:

```
$ oc patch network.operator.openshift.io cluster --type=merge --patch-file=multinetwork-
enable-patch.yaml
```

##### Example output

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

#### 16.3.4.3. Supporting multi-network policies in IPv6 networks

The ICMPv6 Neighbor Discovery Protocol (NDP) is a set of messages and processes that enable devices to discover and maintain information about neighboring nodes. NDP plays a crucial role in IPv6 networks, facilitating the interaction between devices on the same link.

The Cluster Network Operator (CNO) deploys the iptables implementation of multi-network policy when the **useMultiNetworkPolicy** parameter is set to **true**.

To support multi-network policies in IPv6 networks the Cluster Network Operator deploys the following set of rules in every pod affected by a multi-network policy:

##### Multi-network policy custom rules

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: multi-networkpolicy-custom-rules
  namespace: openshift-multus
data:
```

```
custom-v6-rules.txt: |
# accept NDP
-p icmpv6 --icmpv6-type neighbor-solicitation -j ACCEPT ①
-p icmpv6 --icmpv6-type neighbor-advertisement -j ACCEPT ②
# accept RA/RS
-p icmpv6 --icmpv6-type router-solicitation -j ACCEPT ③
-p icmpv6 --icmpv6-type router-advertisement -j ACCEPT ④
```

- ① This rule allows incoming ICMPv6 neighbor solicitation messages, which are part of the neighbor discovery protocol (NDP). These messages help determine the link-layer addresses of neighboring nodes.
- ② This rule allows incoming ICMPv6 neighbor advertisement messages, which are part of NDP and provide information about the link-layer address of the sender.
- ③ This rule permits incoming ICMPv6 router solicitation messages. Hosts use these messages to request router configuration information.
- ④ This rule allows incoming ICMPv6 router advertisement messages, which give configuration information to hosts.



#### NOTE

You cannot edit these predefined rules.

These rules collectively enable essential ICMPv6 traffic for correct network functioning, including address resolution and router communication in an IPv6 environment. With these rules in place and a multi-network policy denying traffic, applications are not expected to experience connectivity issues.

#### 16.3.4.4. Working with multi-network policy

As a cluster administrator, you can create, edit, view, and delete multi-network policies.

##### 16.3.4.4.1. Prerequisites

- You have enabled multi-network policy support for your cluster.

##### 16.3.4.4.2. Creating a multi-network policy using the CLI

To define granular rules describing ingress or egress network traffic allowed for namespaces in your cluster, you can create a multi-network policy.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace that the multi-network policy applies to.

## Procedure

1. Create a policy rule:
  - a. Create a **<policy\_name>.yaml** file:

```
$ touch <policy_name>.yaml
```

where:

### **<policy\_name>**

Specifies the multi-network policy file name.

- b. Define a multi-network policy in the file that you just created, such as in the following examples:

### **Deny ingress from all pods in all namespaces**

This is a fundamental policy, blocking all cross-pod networking other than cross-pod traffic allowed by the configuration of other Network Policies.

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress: []
```

where:

### **<network\_name>**

Specifies the name of a network attachment definition.

### **Allow ingress from all pods in the same namespace**

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

where:

**<network\_name>**

Specifies the name of a network attachment definition.

**Allow ingress traffic to one pod from a particular namespace**

This policy allows traffic to pods labelled **pod-a** from pods running in **namespace-y**.

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-traffic-pod
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      pod: pod-a
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              kubernetes.io/metadata.name: namespace-y
```

where:

**<network\_name>**

Specifies the name of a network attachment definition.

**Restrict traffic to a service**

This policy when applied ensures every pod with both labels **app=bookstore** and **role=api** can only be accessed by pods with label **app=bookstore**. In this example the application could be a REST API server, marked with labels **app=bookstore** and **role=api**.

This example addresses the following use cases:

- Restricting the traffic to a service to only the other microservices that need to use it.
- Restricting the connections to a database to only permit the application using it.

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: api-allow
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
    - from:
```

```

    - podSelector:
      matchLabels:
        app: bookstore
  
```

where:

#### **<network\_name>**

Specifies the name of a network attachment definition.

2. To create the multi-network policy object, enter the following command:

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

where:

#### **<policy\_name>**

Specifies the multi-network policy file name.

#### **<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

#### Example output

```
multinetworkpolicy.k8s.cni.cncf.io/deny-by-default created
```



#### NOTE

If you log in to the web console with **cluster-admin** privileges, you have a choice of creating a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

#### 16.3.4.4.3. Editing a multi-network policy

You can edit a multi-network policy in a namespace.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace where the multi-network policy exists.

#### Procedure

1. Optional: To list the multi-network policy objects in a namespace, enter the following command:

```
$ oc get multinetworkpolicy
```

where:

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

2. Edit the multi-network policy object.

- If you saved the multi-network policy definition in a file, edit the file and make any necessary changes, and then enter the following command.

```
$ oc apply -n <namespace> -f <policy_file>.yaml
```

where:

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

**<policy\_file>**

Specifies the name of the file containing the network policy.

- If you need to update the multi-network policy object directly, enter the following command:

```
$ oc edit multi-networkpolicy <policy_name> -n <namespace>
```

where:

**<policy\_name>**

Specifies the name of the network policy.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

3. Confirm that the multi-network policy object is updated.

```
$ oc describe multi-networkpolicy <policy_name> -n <namespace>
```

where:

**<policy\_name>**

Specifies the name of the multi-network policy.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

**NOTE**



If you log in to the web console with **cluster-admin** privileges, you have a choice of editing a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

#### 16.3.4.4.4. Viewing multi-network policies using the CLI

You can examine the multi-network policies in a namespace.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace where the multi-network policy exists.

#### Procedure

- List multi-network policies in a namespace:
  - To view multi-network policy objects defined in a namespace, enter the following command:
 

```
$ oc get multi-networkpolicy
```
  - Optional: To examine a specific multi-network policy, enter the following command:
 

```
$ oc describe multi-networkpolicy <policy_name> -n <namespace>
```

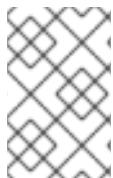
where:

**<policy\_name>**

Specifies the name of the multi-network policy to inspect.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.



#### NOTE

If you log in to the web console with **cluster-admin** privileges, you have a choice of viewing a network policy in any namespace in the cluster directly in YAML or from a form in the web console.

#### 16.3.4.4.5. Deleting a multi-network policy using the CLI

You can delete a multi-network policy in a namespace.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace where the multi-network policy exists.

#### Procedure

- To delete a multi-network policy object, enter the following command:

```
$ oc delete multi-networkpolicy <policy_name> -n <namespace>
```

where:

**<policy\_name>**

Specifies the name of the multi-network policy.

**<namespace>**

Optional: Specifies the namespace if the object is defined in a different namespace than the current namespace.

#### Example output

```
multinetworkpolicy.k8s.cni.cncf.io/default-deny deleted
```



#### NOTE

If you log in to the web console with **cluster-admin** privileges, you have a choice of deleting a network policy in any namespace in the cluster directly in YAML or from the policy in the web console through the **Actions** menu.

#### 16.3.4.4.6. Creating a default deny all multi-network policy

This policy blocks all cross-pod networking other than network traffic allowed by the configuration of other deployed network policies and traffic between host-networked pods. This procedure enforces a strong deny policy by applying a **deny-by-default** policy in the **my-project** namespace.



#### WARNING

Without configuring a **NetworkPolicy** custom resource (CR) that allows traffic communication, the following policy might cause communication problems across your cluster.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace that the multi-network policy applies to.

#### Procedure

- Create the following YAML that defines a **deny-by-default** policy to deny ingress from all pods in all namespaces. Save the YAML in the **deny-by-default.yaml** file:

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  namespace: my-project 1
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <namespace_name>/<network_name> 2
spec:
  podSelector: {} 3
  policyTypes: 4
    - Ingress 5
  ingress: [] 6
```

- 1** Specifies the namespace in which to deploy the policy. For example, the **my-project** namespace.
- 2** Specifies the name of a network attachment definition.
- 3** If this field is empty, the configuration matches all the pods. Therefore, the policy applies to all pods in the **my-project** namespace.
- 4** Specifies a list of rule types that the **NetworkPolicy** relates to.
- 5** Specifies **Ingress** only **policyTypes**.
- 6** Specifies **ingress** rules. If not specified, all incoming traffic is dropped to all pods.

- Apply the policy by entering the following command:

```
$ oc apply -f deny-by-default.yaml
```

#### Example output

```
multinetworkpolicy.k8s.cni.cncf.io/deny-by-default created
```

##### 16.3.4.4.7. Creating a multi-network policy to allow traffic from external clients

With the **deny-by-default** policy in place you can proceed to configure a policy that allows traffic from external clients to a pod with the label **app=web**.



#### NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Follow this procedure to configure a policy that allows external service from the public Internet directly or by using a Load Balancer to access the pod. Traffic is only allowed to a pod with the label **app=web**.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace that the multi-network policy applies to.

## Procedure

1. Create a policy that allows traffic from the public Internet directly or by using a load balancer to access the pod. Save the YAML in the **web-allow-external.yaml** file:

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-external
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  policyTypes:
    - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

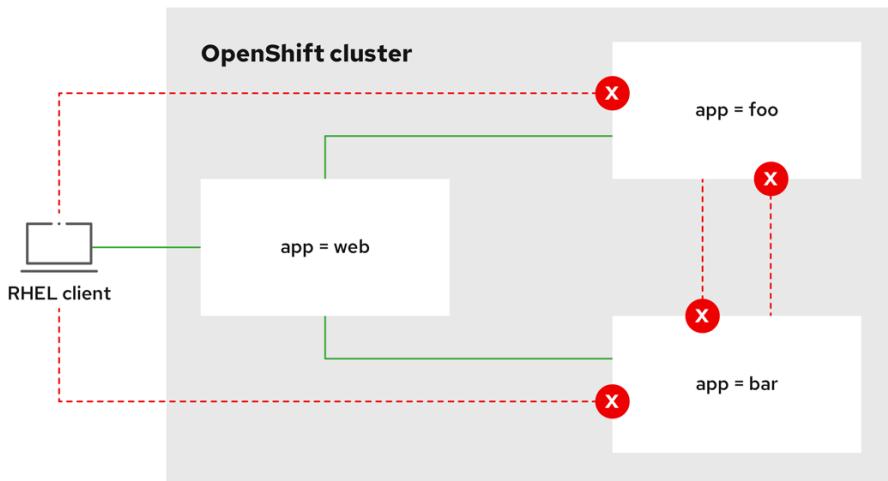
2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-external.yaml
```

### Example output

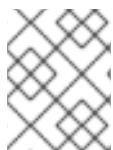
```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-external created
```

This policy allows traffic from all resources, including external traffic as illustrated in the following diagram:



292\_OpenShift\_1122

#### 16.3.4.4.8. Creating a multi-network policy allowing traffic to an application from all namespaces



#### NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Follow this procedure to configure a policy that allows traffic from all pods in all namespaces to a particular application.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace that the multi-network policy applies to.

#### Procedure

1. Create a policy that allows traffic from all pods in all namespaces to a particular application. Save the YAML in the **web-allow-all-namespaces.yaml** file:

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-all-namespaces
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: web ①
  policyTypes:
```

```

    - Ingress
ingress:
  - from:
    - namespaceSelector: {} 2

```

- 1** Applies the policy only to **app:web** pods in default namespace.
- 2** Selects all pods in all namespaces.



#### NOTE

By default, if you omit specifying a **namespaceSelector** it does not select any namespaces, which means the policy allows traffic only from the namespace the network policy is deployed to.

2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-all-namespaces.yaml
```

#### Example output

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-all-namespaces created
```

#### Verification

1. Start a web service in the **default** namespace by entering the following command:

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```

2. Run the following command to deploy an **alpine** image in the **secondary** namespace and to start a shell:

```
$ oc run test-$RANDOM --namespace=secondary --rm -i -t --image=alpine -- sh
```

3. Run the following command in the shell and observe that the request is allowed:

```
# wget -qO- --timeout=2 http://web.default
```

#### Expected output

```

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>

```

```

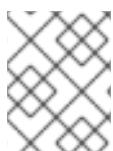
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

```

#### 16.3.4.4.9. Creating a multi-network policy allowing traffic to an application from a namespace



##### NOTE

If you log in with a user with the **cluster-admin** role, then you can create a network policy in any namespace in the cluster.

Follow this procedure to configure a policy that allows traffic to a pod with the label **app=web** from a particular namespace. You might want to do this to:

- Restrict traffic to a production database only to namespaces where production workloads are deployed.
- Enable monitoring tools deployed to a particular namespace to scrape metrics from the current namespace.

#### Prerequisites

- Your cluster uses a network plugin that supports **NetworkPolicy** objects, such as the OVN-Kubernetes network plugin, with **mode: NetworkPolicy** set.
- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You are working in the namespace that the multi-network policy applies to.

#### Procedure

1. Create a policy that allows traffic from all pods in a particular namespaces with a label **purpose=production**. Save the YAML in the **web-allow-prod.yaml** file:

```

apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-prod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:

```

```

podSelector:
  matchLabels:
    app: web 1
policyTypes:
- Ingress
ingress:
- from:
  - namespaceSelector:
    matchLabels:
      purpose: production 2

```

- 1** Applies the policy only to **app:web** pods in the default namespace.
- 2** Restricts traffic to only pods in namespaces that have the label **purpose=production**.

## 2. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-prod.yaml
```

### Example output

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-prod created
```

## Verification

1. Start a web service in the **default** namespace by entering the following command:
 

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```
2. Run the following command to create the **prod** namespace:
 

```
$ oc create namespace prod
```
3. Run the following command to label the **prod** namespace:
 

```
$ oc label namespace/prod purpose=production
```
4. Run the following command to create the **dev** namespace:
 

```
$ oc create namespace dev
```
5. Run the following command to label the **dev** namespace:
 

```
$ oc label namespace/dev purpose=testing
```
6. Run the following command to deploy an **alpine** image in the **dev** namespace and to start a shell:
 

```
$ oc run test-$RANDOM --namespace=dev --rm -i -t --image=alpine -- sh
```
7. Run the following command in the shell and observe that the request is blocked:

```
# wget -qO- --timeout=2 http://web.default
```

#### Expected output

```
wget: download timed out
```

8. Run the following command to deploy an **alpine** image in the **prod** namespace and start a shell:

```
$ oc run test-$RANDOM --namespace=prod --rm -i -t --image=alpine -- sh
```

9. Run the following command in the shell and observe that the request is allowed:

```
# wget -qO- --timeout=2 http://web.default
```

#### Expected output

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

#### 16.3.4.5. Additional resources

- [About network policy](#)
- [Understanding multiple networks](#)
- [Configuring a macvlan network](#)
- [Configuring an SR-IOV network device](#)

#### 16.3.5. Removing a pod from a secondary network

As a cluster user you can remove a pod from a secondary network.

### 16.3.5.1. Removing a pod from a secondary network

You can remove a pod from a secondary network only by deleting the pod.

#### Prerequisites

- A secondary network is attached to the pod.
- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

#### Procedure

- To delete the pod, enter the following command:

```
$ oc delete pod <name> -n <namespace>
```

- **<name>** is the name of the pod.
- **<namespace>** is the namespace that contains the pod.

### 16.3.6. Editing a secondary network

As a cluster administrator you can modify the configuration for an existing secondary network.

### 16.3.6.1. Modifying a secondary network attachment definition

As a cluster administrator, you can make changes to an existing secondary network. Any existing pods attached to the secondary network will not be updated.

#### Prerequisites

- You have configured a secondary network for your cluster.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

To edit a secondary network for your cluster, complete the following steps:

1. Run the following command to edit the Cluster Network Operator (CNO) CR in your default text editor:

```
$ oc edit networks.operator.openshift.io cluster
```

2. In the **additionalNetworks** collection, update the secondary network with your changes.
3. Save your changes and quit the text editor to commit your changes.
4. Optional: Confirm that the CNO updated the **NetworkAttachmentDefinition** object by running

the following command. Replace **<network-name>** with the name of the secondary network to display. There might be a delay before the CNO updates the **NetworkAttachmentDefinition** object to reflect your changes.

```
$ oc get network-attachment-definitions <network-name> -o yaml
```

For example, the following console output displays a **NetworkAttachmentDefinition** object that is named **net1**:

```
$ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\n" .spec.config}}'
{
  "cniVersion": "0.3.1",
  "type": "macvlan",
  "master": "ens5",
  "mode": "bridge",
  "ipam": {
    "type": "static",
    "routes": [
      {
        "dst": "0.0.0.0/0",
        "gw": "10.128.2.1"
      }
    ],
    "addresses": [
      {
        "address": "10.128.2.100/23",
        "gateway": "10.128.2.1"
      }
    ],
    "dns": {
      "nameservers": [
        "172.30.0.10"
      ],
      "domain": "us-west-2.compute.internal",
      "search": [
        "us-west-2.compute.internal"
      ]
    }
  }
}
```

### 16.3.7. Configuring IP address assignment on secondary networks

The following sections give instructions and information for how to configure IP address assignments for secondary networks.

#### 16.3.7.1. Configuration of IP address assignment for a network attachment

For secondary networks, IP addresses can be assigned using an IP Address Management (IPAM) CNI plugin, which supports various assignment methods, including Dynamic Host Configuration Protocol (DHCP) and static assignment.

The DHCP IPAM CNI plugin responsible for dynamic assignment of IP addresses operates with two distinct components:

- **CNI Plugin**: Responsible for integrating with the Kubernetes networking stack to request and release IP addresses.
- **DHCP IPAM CNI Daemon**: A listener for DHCP events that coordinates with existing DHCP servers in the environment to handle IP address assignment requests. This daemon is *not* a DHCP server itself.

For networks requiring **type: dhcp** in their IPAM configuration, ensure the following:

- A DHCP server is available and running in the environment. The DHCP server is external to the cluster and is expected to be part of the customer's existing network infrastructure.
- The DHCP server is appropriately configured to serve IP addresses to the nodes.

In cases where a DHCP server is unavailable in the environment, it is recommended to use the Whereabouts IPAM CNI plugin instead. The Whereabouts CNI provides similar IP address management capabilities without the need for an external DHCP server.



#### NOTE

Use the Whereabouts CNI plugin when there is no external DHCP server or where static IP address management is preferred. The Whereabouts plugin includes a reconciler daemon to manage stale IP address allocations.

A DHCP lease must be periodically renewed throughout the container's lifetime, so a separate daemon, the DHCP IPAM CNI Daemon, is required. To deploy the DHCP IPAM CNI daemon, modify the Cluster Network Operator (CNO) configuration to trigger the deployment of this daemon as part of the secondary network setup.

#### 16.3.7.1.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

**Table 16.16. ipam static configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>static</b> is required.
<b>addresses</b>	<b>array</b>	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
<b>routes</b>	<b>array</b>	An array of objects specifying routes to configure inside the pod.
<b>dns</b>	<b>array</b>	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

**Table 16.17. ipam.addresses[] array**

Field	Type	Description
<b>address</b>	<b>string</b>	An IP address and network prefix that you specify. For example, if you specify <b>10.10.21.10/24</b> , then the secondary network is assigned an IP address of <b>10.10.21.10</b> and the netmask is <b>255.255.255.0</b> .
<b>gateway</b>	<b>string</b>	The default gateway to route egress network traffic to.

**Table 16.18. ipam.routes[] array**

Field	Type	Description
<b>dst</b>	<b>string</b>	The IP address range in CIDR format, such as <b>192.168.17.0/24</b> or <b>0.0.0.0/0</b> for the default route.
<b>gw</b>	<b>string</b>	The gateway where network traffic is routed.

**Table 16.19. ipam.dns object**

Field	Type	Description
<b>nameservers</b>	<b>array</b>	An array of one or more IP addresses for to send DNS queries to.

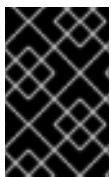
Field	Type	Description
<b>domain</b>	<b>array</b>	The default domain to append to a hostname. For example, if the domain is set to <b>example.com</b> , a DNS lookup query for <b>example-host</b> is rewritten as <b>example-host.example.com</b> .
<b>search</b>	<b>array</b>	An array of domain names to append to an unqualified hostname, such as <b>example-host</b> , during a DNS lookup query.

## Static IP address assignment configuration example

```
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}
```

### 16.3.7.1.2. Dynamic IP address (DHCP) assignment configuration

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.



#### IMPORTANT

For an Ethernet network attachment, the SR-IOV Network Operator does not create a DHCP server deployment; the Cluster Network Operator is responsible for creating the minimal DHCP server deployment.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

#### Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |->
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
```

```

    "ipam": {
      "type": "dhcp"
    }
  }
# ...

```

The following table describes the configuration parameters for dynamic IP address assignment with DHCP.

**Table 16.20. ipam DHCP configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>dhcp</b> is required.

The following JSON example describes the configuration p for dynamic IP address assignment with DHCP.

#### Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

##### 16.3.7.1.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to a secondary network without the use of a DHCP server.

The Whereabouts CNI plugin also supports overlapping IP address ranges and configuration of the same CIDR range multiple times within separate **NetworkAttachmentDefinition** CRDs. This provides greater flexibility and management capabilities in multi-tenant environments.

##### 16.3.7.1.3.1. Dynamic IP address configuration objects

The following table describes the configuration objects for dynamic IP address assignment with Whereabouts:

**Table 16.21. ipam whereabouts configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>whereabouts</b> is required.
<b>range</b>	<b>string</b>	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
<b>exclude</b>	<b>array</b>	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.

Field	Type	Description
<b>network_name</b>	<b>string</b>	Optional: Helps ensure that each group or domain of pods gets its own set of IP addresses, even if they share the same range of IP addresses. Setting this field is important for keeping networks separate and organized, notably in multi-tenant environments.

### 16.3.7.1.3.2. Dynamic IP address assignment configuration that uses Whereabouts

The following example shows a dynamic address assignment configuration that uses Whereabouts:

#### Whereabouts dynamic IP address assignment

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

### 16.3.7.1.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges

The following example shows a dynamic IP address assignment that uses overlapping IP address ranges for multi-tenant networks.

#### NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", ①
  }
}
```

- ① Optional. If set, must match the **network\_name** of NetworkAttachmentDefinition 2.

#### NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", ①
  }
}
```

- 1 Optional. If set, must match the **network\_name** of **NetworkAttachmentDefinition** 1.

#### 16.3.7.1.4. Creating a **whereabouts-reconciler** daemon set

The **Whereabouts reconciler** is responsible for managing dynamic IP address assignments for the pods within a cluster by using the **Whereabouts IP Address Management (IPAM)** solution. It ensures that each pod gets a unique IP address from the specified IP address range. It also handles IP address releases when pods are deleted or scaled down.



#### NOTE

You can also use a **NetworkAttachmentDefinition** custom resource definition (CRD) for dynamic IP address assignment.

The **whereabouts-reconciler** daemon set is automatically created when you configure a secondary network through the Cluster Network Operator. It is not automatically created when you configure a secondary network from a YAML manifest.

To trigger the deployment of the **whereabouts-reconciler** daemon set, you must manually create a **whereabouts-shim** network attachment by editing the Cluster Network Operator custom resource (CR) file.

Use the following procedure to deploy the **whereabouts-reconciler** daemon set.

#### Procedure

1. Edit the **Network.operator.openshift.io** custom resource (CR) by running the following command:

```
$ oc edit network.operator.openshift.io cluster
```

2. Include the **additionalNetworks** section shown in this example YAML extract within the **spec** definition of the custom resource (CR):

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
# ...
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    rawCNIConfig: |-
      {
        "name": "whereabouts-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "whereabouts"
        }
      }
    type: Raw
# ...
```

3. Save the file and exit the text editor.
4. Verify that the **whereabouts-reconciler** daemon set deployed successfully by running the following command:

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

### Example output

```
pod/whereabouts-reconciler-jnp6g 1/1 Running 0 6s
pod/whereabouts-reconciler-k76gg 1/1 Running 0 6s
pod/whereabouts-reconciler-k86t9 1/1 Running 0 6s
pod/whereabouts-reconciler-p4sxw 1/1 Running 0 6s
pod/whereabouts-reconciler-rvfdv 1/1 Running 0 6s
pod/whereabouts-reconciler-svzw9 1/1 Running 0 6s
daemonset.apps/whereabouts-reconciler 6 6 6 6 kubernetes.io/os=linux 6s
```

#### 16.3.7.1.5. Configuring the Whereabouts IP reconciler schedule

The Whereabouts IPAM CNI plugin runs the IP reconciler daily. This process cleans up any stranded IP allocations that might result in exhausting IPs and therefore prevent new pods from getting an IP allocated to them.

Use this procedure to change the frequency at which the IP reconciler runs.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have deployed the **whereabouts-reconciler** daemon set, and the **whereabouts-reconciler** pods are up and running.

#### Procedure

1. Run the following command to create a **ConfigMap** object named **whereabouts-config** in the **openshift-multus** namespace with a specific cron expression for the IP reconciler:

```
$ oc create configmap whereabouts-config -n openshift-multus --from-literal=reconciler_cron_expression="*/15 * * * *"
```

This cron expression indicates the IP reconciler runs every 15 minutes. Adjust the expression based on your specific requirements.



#### NOTE

The **whereabouts-reconciler** daemon set can only consume a cron expression pattern that includes five asterisks. The sixth, which is used to denote seconds, is currently not supported.

2. Retrieve information about resources related to the **whereabouts-reconciler** daemon set and pods within the **openshift-multus** namespace by running the following command:

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

#### Example output

pod/whereabouts-reconciler-2p7hw	1/1	Running	0	4m14s
pod/whereabouts-reconciler-76jk7	1/1	Running	0	4m14s
pod/whereabouts-reconciler-94zw6	1/1	Running	0	4m14s
pod/whereabouts-reconciler-mfh68	1/1	Running	0	4m14s
pod/whereabouts-reconciler-pgshz	1/1	Running	0	4m14s
pod/whereabouts-reconciler-xn5xz	1/1	Running	0	4m14s
daemonset.apps/whereabouts-reconciler	6	6	6	6
kubernetes.io/os=linux				4m16s

3. Run the following command to verify that the **whereabouts-reconciler** pod runs the IP reconciler with the configured interval:

```
$ oc -n openshift-multus logs whereabouts-reconciler-2p7hw
```

#### Example output

```
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..2024_02_16_33_54.1375928161": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..2024_02_16_33_54.1375928161": CHMOD
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..data_tmp": RENAME
2024-02-02T16:33:54Z [verbose] using expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] configuration updated to file "/cron-schedule/..data". New cron expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] successfully updated CRON configuration id "00c2d1c9-631d-403f-bb86-73ad104a6817" - new cron expression: */15 * * * *
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/config": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..2024_02_16_26_17.3874177937": REMOVE
2024-02-02T16:45:00Z [verbose] starting reconciler run
2024-02-02T16:45:00Z [debug] NewReconcileLooper - inferred connection data
2024-02-02T16:45:00Z [debug] listing IP pools
2024-02-02T16:45:00Z [debug] no IP addresses to cleanup
2024-02-02T16:45:00Z [verbose] reconciler success
```

#### 16.3.7.1.6. Creating a configuration for assignment of dual-stack IP addresses dynamically

Dual-stack IP address assignment can be configured with the **ipRanges** parameter for:

- IPv4 addresses
- IPv6 addresses
- multiple IP address assignment

#### Procedure

1. Set **type** to **whereabouts**.
2. Use **ipRanges** to allocate IP addresses as shown in the following example:

```

cniVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
    - name: whereabouts-shim
      namespace: default
      type: Raw
      rawCNIConfig: |->
        {
          "name": "whereabouts-dual-stack",
          "cniVersion": "0.3.1",
          "type": "bridge",
          "ipam": {
            "type": "whereabouts",
            "ipRanges": [
              {"range": "192.168.10.0/24"},
              {"range": "2001:db8::/64"}
            ]
          }
        }
    
```

3. Attach network to a pod. For more information, see "Adding a pod to a secondary network".
4. Verify that all IP addresses are assigned.
5. Run the following command to ensure the IP addresses are assigned as metadata.

```
$ oc exec -it mypod -- ip a
```

### 16.3.8. Configuring the master interface in the container network namespace

The following section provides instructions and information for how to create and manage a MAC-VLAN, IP-VLAN, and VLAN subinterface based on a master interface.

#### 16.3.8.1. About configuring the master interface in the container network namespace

You can create a MAC-VLAN, an IP-VLAN, or a VLAN subinterface that is based on a **master** interface that exists in a container namespace. You can also create a **master** interface as part of the pod network configuration in a separate network attachment definition CRD.

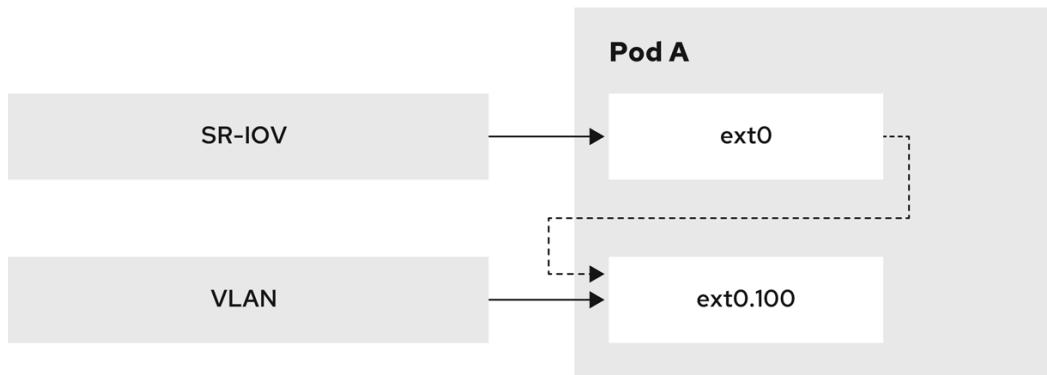
To use a container namespace **master** interface, you must specify **true** for the **linkInContainer** parameter that exists in the subinterface configuration of the **NetworkAttachmentDefinition** CRD.

##### 16.3.8.1.1. Creating multiple VLANs on SR-IOV VFs

An example use case for utilizing this feature is to create multiple VLANs based on SR-IOV VFs. To do so, begin by creating an SR-IOV network and then define the network attachments for the VLAN interfaces.

The following example shows how to configure the setup illustrated in this diagram.

**Figure 16.5. Creating VLANs**



345\_OpenShift\_0823

## Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.

## Procedure

1. Create a dedicated container namespace where you want to deploy your pod by using the following command:

```
$ oc new-project test-namespace
```

2. Create an SR-IOV node policy:

- a. Create an **SriovNetworkNodePolicy** object, and then save the YAML in the **sriov-node-network-policy.yaml** file:

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnic
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: false
  needVhostNet: true
  nicSelector:
    vendor: "15b3" ①
    devicID: "101b" ②
    rootDevices: ["00:05.0"]
    numVfs: 10
    priority: 99
  resourceName: sriovnic
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  
```

**NOTE**

The SR-IOV network node policy configuration example, with the setting **deviceType: netdevice**, is tailored specifically for Mellanox Network Interface Cards (NICs).

- 1 The vendor hexadecimal code of the SR-IOV network device. The value **15b3** is associated with a Mellanox NIC.
- 2 The device hexadecimal code of the SR-IOV network device.

- b. Apply the YAML by running the following command:

```
$ oc apply -f sriov-node-network-policy.yaml
```

**NOTE**

Applying this might take some time due to the node requiring a reboot.

3. Create an SR-IOV network:

- a. Create the **SriovNetwork** custom resource (CR) for the additional secondary SR-IOV network attachment as in the following example CR. Save the YAML as the file **sriov-network-attachment.yaml**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: test-namespace
  resourceName: sriovnic
  spoofChk: "off"
  trust: "on"
```

- b. Apply the YAML by running the following command:

```
$ oc apply -f sriov-network-attachment.yaml
```

4. Create the VLAN secondary network:

- a. Using the following YAML example, create a file named **vlan100-additional-network-configuration.yaml**:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: vlan-100
  namespace: test-namespace
spec:
  config: |
    {
```

```

    "cniVersion": "0.4.0",
    "name": "vlan-100",
    "plugins": [
      {
        "type": "vlan",
        "master": "ext0", 1
        "mtu": 1500,
        "vlanId": 100,
        "linkInContainer": true, 2
        "ipam": {"type": "whereabouts", "ipRanges": [{"range": "1.1.1.0/24"}]}
      }
    ]
  }
}

```

- 1** The VLAN configuration needs to specify the **master** name. This can be configured in the pod networks annotation.
- 2** The **linkInContainer** parameter must be specified.

- b. Apply the YAML file by running the following command:

```
$ oc apply -f vlan100-additional-network-configuration.yaml
```

5. Create a pod definition by using the earlier specified networks:

- a. Using the following YAML example, create a file named **pod-a.yaml** file:



### NOTE

The manifest below includes 2 resources:

- Namespace with security labels
- Pod definition with appropriate network annotation

```

apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "sriov-network",

```

```

    "namespace": "test-namespace",
    "interface": "ext0" ①
},
{
  "name": "vlan-100",
  "namespace": "test-namespace",
  "interface": "ext0.100"
}
]
spec:
  securityContext:
    runAsNonRoot: true
  containers:
    - name: nginx-container
      image: nginxinc/nginx-unprivileged:latest
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
      ports:
        - containerPort: 80
      seccompProfile:
        type: "RuntimeDefault"

```

① The name to be used as the **master** for the VLAN interface.

b. Apply the YAML file by running the following command:

```
$ oc apply -f pod-a.yaml
```

6. Get detailed information about the **nginx-pod** within the **test-namespace** by running the following command:

```
$ oc describe pods nginx-pod -n test-namespace
```

### Example output

```

Name:      nginx-pod
Namespace: test-namespace
Priority:  0
Node:      worker-1/10.46.186.105
Start Time: Mon, 14 Aug 2023 16:23:13 -0400
Labels:    <none>
Annotations: k8s.ovn.org/pod-networks:
            {"default":{"ip_addresses":
              ["10.131.0.26/23"],"mac_address":"0a:58:0a:83:00:1a","gateway_ips":["10.131.0.1"],"routes":
              [{"dest":"10.128.0.0..."}]}
             k8s.v1.cni.cncf.io/network-status:
             [
               {
                 "name": "ovn-kubernetes",
                 "interface": "eth0",
                 "ips": [
                   "10.131.0.26"
                 ],
               }
             ]

```

```

    "mac": "0a:58:0a:83:00:1a",
    "default": true,
    "dns": {}
},{
    "name": "test-namespace/sriov-network",
    "interface": "ext0",
    "mac": "6e:a7:5e:3f:49:1b",
    "dns": {},
    "device-info": {
        "type": "pci",
        "version": "1.0.0",
        "pci": {
            "pci-address": "0000:d8:00.2"
        }
    }
},{
    "name": "test-namespace/vlan-100",
    "interface": "ext0.100",
    "ips": [
        "1.1.1.1"
    ],
    "mac": "6e:a7:5e:3f:49:1b",
    "dns": {}
}]
k8s.v1.cni.cncf.io/networks:
[ { "name": "sriov-network", "namespace": "test-namespace", "interface": "ext0" }, {
    "name": "vlan-100", "namespace": "test-namespace", "i...
        openshift.io/scc: privileged
Status:     Running
IP:         10.131.0.26
IPs:
    IP: 10.131.0.26

```

#### 16.3.8.1.2. Creating a subinterface based on a bridge master interface in a container namespace

You can create a subinterface based on a bridge **master** interface that exists in a container namespace. Creating a subinterface can be applied to other types of interfaces.

#### Prerequisites

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

#### Procedure

1. Create a dedicated container namespace where you want to deploy your pod by entering the following command:
 

```
$ oc new-project test-namespace
```
2. Using the following YAML example, create a bridge **NetworkAttachmentDefinition** custom resource definition (CRD) file named **bridge-nad.yaml**:

```

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "bridge-network",
    "type": "bridge",
    "bridge": "br-001",
    "isGateway": true,
    "ipMasq": true,
    "hairpinMode": true,
    "ipam": {
      "type": "host-local",
      "subnet": "10.0.0.0/24",
      "routes": [{"dst": "0.0.0.0/0"}]
    }
  }'

```

- Run the following command to apply the **NetworkAttachmentDefinition** CRD to your OpenShift Container Platform cluster:

```
$ oc apply -f bridge-nad.yaml
```

- Verify that you successfully created a **NetworkAttachmentDefinition** CRD by entering the following command:

```
$ oc get network-attachment-definitions
```

### Example output

NAME	AGE
bridge-network	15s

- Using the following YAML example, create a file named **ipvlan-additional-network-configuration.yaml** for the IPVLAN secondary network configuration:

```

apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: ipvlan-net
  namespace: test-namespace
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "ipvlan-net",
    "type": "ipvlan",
    "master": "net1", 1
    "mode": "l3",
    "linkInContainer": true, 2
    "ipam": {"type": "whereabouts", "ipRanges": [{"range": "10.0.0.0/24"}]}
  }'

```

- 1 Specifies the ethernet interface to associate with the network attachment. This is subsequently configured in the pod networks annotation.
- 2 Specifies that the **master** interface is in the container network namespace.

6. Apply the YAML file by running the following command:

```
$ oc apply -f ipvlan-additional-network-configuration.yaml
```

7. Verify that the **NetworkAttachmentDefinition** CRD has been created successfully by running the following command:

```
$ oc get network-attachment-definitions
```

### Example output

NAME	AGE
bridge-network	87s
ipvlan-net	9s

8. Using the following YAML example, create a file named **pod-a.yaml** for the pod definition:

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-a
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '['
      {
        "name": "bridge-network",
        "interface": "net1" ①
      },
      {
        "name": "ipvlan-net",
        "interface": "net2"
      }
    ]'
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: test-pod
      image: quay.io/openshifttest/hello-
sdn@sha256:c89445416459e7adea9a5a416b3365ed3d74f2491beb904d61dc8d1eb89a72a4

  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

- 1 Specifies the name to be used as the **master** for the IPvLAN interface.

9. Apply the YAML file by running the following command:

```
$ oc apply -f pod-a.yaml
```

10. Verify that the pod is running by using the following command:

```
$ oc get pod -n test-namespace
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
pod-a	1/1	Running	0	2m36s

11. Show network interface information about the **pod-a** resource within the **test-namespace** by running the following command:

```
$ oc exec -n test-namespace pod-a -- ip a
```

#### Example output

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    qlen 1000
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
            inet6 ::1/128 scope host
                valid_lft forever preferred_lft forever
3: eth0@if105: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue
    state UP group default
        link/ether 0a:58:0a:d9:00:5d brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 10.217.0.93/23 brd 10.217.1.255 scope global eth0
            valid_lft forever preferred_lft forever
            inet6 fe80::488b:91ff:fe84:a94b/64 scope link
                valid_lft forever preferred_lft forever
4: net1@if107: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    state UP group default
        link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff link-netnsid 0
        inet 10.0.0.2/24 brd 10.0.0.255 scope global net1
            valid_lft forever preferred_lft forever
            inet6 fe80::bcda:bdff:fe7e:f437/64 scope link
                valid_lft forever preferred_lft forever
5: net2@net1: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue
    state UNKNOWN group default
        link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff
        inet 10.0.0.1/24 brd 10.0.0.255 scope global net2
            valid_lft forever preferred_lft forever
            inet6 fe80::beda:bd00:17e:f437/64 scope link
                valid_lft forever preferred_lft forever
```

This output shows that the network interface **net2** is associated with the physical interface **net1**.

## 16.3.9. Removing an additional network

As a cluster administrator you can remove an additional network attachment.

### 16.3.9.1. Removing a secondary network attachment definition

As a cluster administrator, you can remove a secondary network from your OpenShift Container Platform cluster. The secondary network is not removed from any pods it is attached to.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

To remove a secondary network from your cluster, complete the following steps:

1. Edit the Cluster Network Operator (CNO) in your default text editor by running the following command:

```
$ oc edit networks.operator.openshift.io cluster
```

2. Modify the CR by removing the configuration that the CNO created from the **additionalNetworks** collection for the secondary network that you want to remove.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: [] ①
```

- 1 If you are removing the configuration mapping for the only secondary network attachment definition in the **additionalNetworks** collection, you must specify an empty collection.
3. To remove a network attachment definition from the network of your cluster, enter the following command:

```
$ oc delete net-attach-def <name_of_NAD> ①
```

  - 1 Replace **<name\_of\_NAD>** with the name of your network attachment definition.
4. Save your changes and quit the text editor to commit your changes.
5. Optional: Confirm that the secondary network CR was deleted by running the following command:

```
$ oc get network-attachment-definition --all-namespaces
```

## 16.4. VIRTUAL ROUTING AND FORWARDING

## 16.4.1. About virtual routing and forwarding

Virtual routing and forwarding (VRF) devices combined with IP rules provide the ability to create virtual routing and forwarding domains. VRF reduces the number of permissions needed by CNF, and provides increased visibility of the network topology of secondary networks. VRF is used to provide multi-tenancy functionality, for example, where each tenant has its own unique routing tables and requires different default gateways.

Processes can bind a socket to the VRF device. Packets through the binded socket use the routing table associated with the VRF device. An important feature of VRF is that it impacts only OSI model layer 3 traffic and above so L2 tools, such as LLDP, are not affected. This allows higher priority IP rules such as policy based routing to take precedence over the VRF device rules directing specific traffic.

### 16.4.1.1. Benefits of secondary networks for pods for telecommunications operators

In telecommunications use cases, each CNF can potentially be connected to multiple different networks sharing the same address space. These secondary networks can potentially conflict with the cluster's main network CIDR. Using the CNI VRF plugin, network functions can be connected to different customers' infrastructure using the same IP address, keeping different customers isolated. IP addresses are overlapped with OpenShift Container Platform IP space. The CNI VRF plugin also reduces the number of permissions needed by CNF and increases the visibility of network topologies of secondary networks.

## 16.5. ASSIGNING A SECONDARY NETWORK TO A VRF

As a cluster administrator, you can configure a secondary network for a virtual routing and forwarding (VRF) domain by using the CNI VRF plugin. The virtual network that this plugin creates is associated with the physical interface that you specify.

Using a secondary network with a VRF instance has the following advantages:

### Workload isolation

Isolate workload traffic by configuring a VRF instance for the secondary network.

### Improved security

Enable improved security through isolated network paths in the VRF domain.

### Multi-tenancy support

Support multi-tenancy through network segmentation with a unique routing table in the VRF domain for each tenant.



### NOTE

Applications that use VRFs must bind to a specific device. The common usage is to use the **SO\_BINDTODEVICE** option for a socket. The **SO\_BINDTODEVICE** option binds the socket to the device that is specified in the passed interface name, for example, **eth1**. To use the **SO\_BINDTODEVICE** option, the application must have **CAP\_NET\_RAW** capabilities.

Using a VRF through the **ip vrf exec** command is not supported in OpenShift Container Platform pods. To use VRF, bind applications directly to the VRF interface.

### Additional resources

- [About virtual routing and forwarding](#)

## 16.5.1. Creating a secondary network attachment with the CNI VRF plugin

The Cluster Network Operator (CNO) manages secondary network definitions. When you specify a secondary network to create, the CNO creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.



### NOTE

Do not edit the **NetworkAttachmentDefinition** CRs that the Cluster Network Operator manages. Doing so might disrupt network traffic on your secondary network.

To create a secondary network attachment with the CNI VRF plugin, perform the following procedure.

#### Prerequisites

- Install the OpenShift Container Platform CLI (oc).
- Log in to the OpenShift cluster as a user with cluster-admin privileges.

#### Procedure

- 1 Create the **Network** custom resource (CR) for the additional network attachment and insert the **rawCNICConfig** configuration for the secondary network, as in the following example CR. Save the YAML as the file **additional-network-attachment.yaml**.

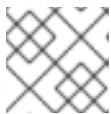
```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
    - name: test-network-1
      namespace: additional-network-1
      type: Raw
      rawCNICConfig: '{
        "cniVersion": "0.3.1",
        "name": "macvlan-vrf",
        "plugins": [ ①
          {
            "type": "macvlan",
            "master": "eth1",
            "ipam": {
              "type": "static",
              "addresses": [
                {
                  "address": "191.168.1.23/24"
                }
              ]
            }
          },
          {
            "type": "vrf", ②
            "vrfname": "vrf-1", ③
          }
        ]
      }'
```

```

    "table": 1001 ④
  }]
}'

```

- 1 **plugins** must be a list. The first item in the list must be the secondary network underpinning the VRF network. The second item in the list is the VRF plugin configuration.
- 2 **type** must be set to **vrf**.
- 3 **vrfname** is the name of the VRF that the interface is assigned to. If it does not exist in the pod, it is created.
- 4 Optional. **table** is the routing table ID. By default, the **tableid** parameter is used. If it is not specified, the CNI assigns a free routing table ID to the VRF.



#### NOTE

VRF functions correctly only when the resource is of type **netdevice**.

2. Create the **Network** resource:

```
$ oc create -f additional-network-attachment.yaml
```

3. Confirm that the CNO created the **NetworkAttachmentDefinition** CR by running the following command. Replace **<namespace>** with the namespace that you specified when configuring the network attachment, for example, **additional-network-1**.

```
$ oc get network-attachment-definitions -n <namespace>
```

#### Example output

NAME	AGE
additional-network-1	14m



#### NOTE

There might be a delay before the CNO creates the CR.

### Verification

1. Create a pod and assign it to the secondary network with the VRF instance:
  - a. Create a YAML file that defines the **Pod** resource:

#### Example pod-additional-net.yaml file

```

apiVersion: v1
kind: Pod
metadata:
  name: pod-additional-net
  annotations:
    k8s.v1.cni.cncf.io/networks: '['

```

```
{  
    "name": "test-network-1" ①  
}  
]  
spec:  
containers:  
- name: example-pod-1  
  command: ["/bin/bash", "-c", "sleep 9000000"]  
  image: centos:8
```

- ① Specify the name of the secondary network with the VRF instance.

- b. Create the **Pod** resource by running the following command:

```
$ oc create -f pod-additional-net.yaml
```

#### Example output

```
pod/test-pod created
```

2. Verify that the pod network attachment is connected to the VRF secondary network. Start a remote session with the pod and run the following command:

```
$ ip vrf show
```

#### Example output

Name	Table
vrf-1	1001

3. Confirm that the VRF interface is the controller for the secondary interface:

```
$ ip link
```

#### Example output

```
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red  
state UP mode
```

# CHAPTER 17. HARDWARE NETWORKS

## 17.1. ABOUT SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) HARDWARE NETWORKS

The Single Root I/O Virtualization (SR-IOV) specification is a standard for a type of PCI device assignment that can share a single device with multiple pods.

You can configure a Single Root I/O Virtualization (SR-IOV) device in your cluster by using the [SR-IOV Operator](#).

SR-IOV can segment a compliant network device, recognized on the host node as a physical function (PF), into multiple virtual functions (VFs). The VF is used like any other network device. The SR-IOV network device driver for the device determines how the VF is exposed in the container:

- **netdevice** driver: A regular kernel network device in the **netns** of the container
- **vfio-pci** driver: A character device mounted in the container

You can use SR-IOV network devices with additional networks on your OpenShift Container Platform cluster installed on bare metal or Red Hat OpenStack Platform (RHOSP) infrastructure for applications that require high bandwidth or low latency.

You can configure multi-network policies for SR-IOV networks. The support for this is technology preview and SR-IOV additional networks are only supported with kernel NICs. They are not supported for Data Plane Development Kit (DPDK) applications.



### NOTE

Creating multi-network policies on SR-IOV networks might not deliver the same performance to applications compared to SR-IOV networks without a multi-network policy configured.



### IMPORTANT

Multi-network policies for SR-IOV network 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](#).

You can enable SR-IOV on a node by using the following command:

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```

### Additional resources

- [Installing the SR-IOV Network Operator](#)

## 17.1.1. Components that manage SR-IOV network devices

The SR-IOV Network Operator creates and manages the components of the SR-IOV stack. The Operator performs the following functions:

- Orchestrates discovery and management of SR-IOV network devices
- Generates **NetworkAttachmentDefinition** custom resources for the SR-IOV Container Network Interface (CNI)
- Creates and updates the configuration of the SR-IOV network device plugin
- Creates node specific **SriovNetworkNodeState** custom resources
- Updates the **spec.interfaces** field in each **SriovNetworkNodeState** custom resource

The Operator provisions the following components:

### SR-IOV network configuration daemon

A daemon set that is deployed on worker nodes when the SR-IOV Network Operator starts. The daemon is responsible for discovering and initializing SR-IOV network devices in the cluster.

### SR-IOV Network Operator webhook

A dynamic admission controller webhook that validates the Operator custom resource and sets appropriate default values for unset fields.

### SR-IOV Network resources injector

A dynamic admission controller webhook that provides functionality for patching Kubernetes pod specifications with requests and limits for custom network resources such as SR-IOV VFs. The SR-IOV network resources injector adds the **resource** field to only the first container in a pod automatically.

### SR-IOV network device plugin

A device plugin that discovers, advertises, and allocates SR-IOV network virtual function (VF) resources. Device plugins are used in Kubernetes to enable the use of limited resources, typically in physical devices. Device plugins give the Kubernetes scheduler awareness of resource availability, so that the scheduler can schedule pods on nodes with sufficient resources.

### SR-IOV CNI plugin

A CNI plugin that attaches VF interfaces allocated from the SR-IOV network device plugin directly into a pod.

### SR-IOV InfiniBand CNI plugin

A CNI plugin that attaches InfiniBand (IB) VF interfaces allocated from the SR-IOV network device plugin directly into a pod.



### NOTE

The SR-IOV Network resources injector and SR-IOV Network Operator webhook are enabled by default and can be disabled by editing the **default SriovOperatorConfig** CR. Use caution when disabling the SR-IOV Network Operator Admission Controller webhook. You can disable the webhook under specific circumstances, such as troubleshooting, or if you want to use unsupported devices.

## 17.1.1.1. Supported platforms

The SR-IOV Network Operator is supported on the following platforms:

- Bare metal
- Red Hat OpenStack Platform (RHOSP)

### 17.1.1.2. Supported devices

OpenShift Container Platform supports the following network interface controllers:

**Table 17.1. Supported network interface controllers**

Manufacturer	Model	Vendor ID	Device ID
Broadcom	BCM57414	14e4	16d7
Broadcom	BCM57508	14e4	1750
Broadcom	BCM57504	14e4	1751
Intel	X710	8086	1572
Intel	X710 Backplane	8086	1581
Intel	X710 Base T	8086	15ff
Intel	XL710	8086	1583
Intel	XXV710	8086	158b
Intel	E810-CQDA2	8086	1592
Intel	E810-2CQDA2	8086	1592
Intel	E810-XXVDA2	8086	159b
Intel	E810-XXVDA4	8086	1593
Intel	E810-XXVDA4T	8086	1593
Intel	Ice E810-XXV Backplane	8086	1599
Intel	Ice E823L Backplane	8086	124c
Intel	Ice E823L SFP	8086	124d
Marvell	OCTEON Fusion CNF105XX	177d	ba00
Marvell	OCTEON10 CN10XXX	177d	b900

Manufacturer	Model	Vendor ID	Device ID
Mellanox	MT27700 Family [ConnectX-4]	15b3	1013
Mellanox	MT27710 Family [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 Family [ConnectX-6]	15b3	101b
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	Mellanox MT2910 Family [ConnectX-7]	15b3	1021
Mellanox	MT42822 BlueField-2 in ConnectX-6 NIC mode	15b3	a2d6
Pensando <sup>[1]</sup>	DSC-25 dual-port 25G distributed services card for ionic driver	0x1dd8	0x1002
Pensando <sup>[1]</sup>	DSC-100 dual-port 100G distributed services card for ionic driver	0x1dd8	0x1003
Silicom	STS Family	8086	1591

1. OpenShift SR-IOV is supported, but you must set a static, Virtual Function (VF) media access control (MAC) address using the SR-IOV CNI config file when using SR-IOV.



#### NOTE

For the most up-to-date list of supported cards and compatible OpenShift Container Platform versions available, see [Openshift Single Root I/O Virtualization \(SR-IOV\) and PTP hardware networks Support Matrix](#).

#### 17.1.2. Additional resources

- [Configuring multi-network policy](#)

#### 17.1.3. Next steps

- [Configuring the SR-IOV Network Operator](#)
- [Configuring an SR-IOV network device](#)
- If you use OpenShift Virtualization: [Connecting a virtual machine to an SR-IOV network](#)

- Configuring an SR-IOV network attachment
- Ethernet network attachement: Adding a pod to an SR-IOV additional network
- InfiniBand network attachement: Adding a pod to an SR-IOV additional network

## 17.2. CONFIGURING AN SR-IOV NETWORK DEVICE

You can configure a Single Root I/O Virtualization (SR-IOV) device in your cluster.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.2.1. SR-IOV network node configuration object

You specify the SR-IOV network device configuration for a node by creating an SR-IOV network node policy. The API object for the policy is part of the **sriovnetwork.openshift.io** API group.

The following YAML describes an SR-IOV network node policy:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> ①
  namespace: openshift-sriov-network-operator ②
spec:
  resourceName: <sriov_resource_name> ③
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" ④
  priority: <priority> ⑤
  mtu: <mtu> ⑥
  needVhostNet: false ⑦
  numVfs: <num> ⑧
  externallyManaged: false ⑨
  nicSelector: ⑩
    vendor: "<vendor_code>" ⑪
    deviceID: "<device_id>" ⑫
    pfNames: ["<pf_name>", ...] ⑬
    rootDevices: ["<pci_bus_id>", ...] ⑭
    netFilter: "<filter_string>" ⑮
  deviceType: <device_type> ⑯
  isRdma: false ⑰
  linkType: <link_type> ⑱
  eSwitchMode: "switchdev" ⑲
  excludeTopology: false ⑳
```

- ① The name for the custom resource object.
- ② The namespace where the SR-IOV Network Operator is installed.
- ③ The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.

When specifying a name, be sure to use the accepted syntax expression `^[a-zA-Z0-9_]+$` in the **resourceName**.

- 4 The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed on selected nodes only.



### IMPORTANT

The SR-IOV Network Operator applies node network configuration policies to nodes in sequence. Before applying node network configuration policies, the SR-IOV Network Operator checks if the machine config pool (MCP) for a node is in an unhealthy state such as **Degraded** or **Updating**. If a node is in an unhealthy MCP, the process of applying node network configuration policies to all targeted nodes in the cluster pauses until the MCP returns to a healthy state.

To avoid a node in an unhealthy MCP from blocking the application of node network configuration policies to other nodes, including nodes in other MCPs, you must create a separate node network configuration policy for each MCP.

- 5 Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.
- 6 Optional: The maximum transmission unit (MTU) of the physical function and all its virtual functions. The maximum MTU value can vary for different network interface controller (NIC) models.



### IMPORTANT

If you want to create virtual function on the default network interface, ensure that the MTU is set to a value that matches the cluster MTU.

If you want to modify the MTU of a single virtual function while the function is assigned to a pod, leave the MTU value blank in the SR-IOV network node policy. Otherwise, the SR-IOV Network Operator reverts the MTU of the virtual function to the MTU value defined in the SR-IOV network node policy, which might trigger a node drain.

- 7 Optional: Set **needVhostNet** to **true** to mount the **/dev/vhost-net** device in the pod. Use the mounted **/dev/vhost-net** device with Data Plane Development Kit (DPDK) to forward traffic to the kernel network stack.
- 8 The number of the virtual functions (VF) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.
- 9 The **externallyManaged** field indicates whether the SR-IOV Network Operator manages all, or only a subset of virtual functions (VFs). With the value set to **false** the SR-IOV Network Operator manages and configures all VFs on the PF.



## NOTE

When **externallyManaged** is set to **true**, you must manually create the Virtual Functions (VFs) on the physical function (PF) before applying the **SriovNetworkNodePolicy** resource. If the VFs are not pre-created, the SR-IOV Network Operator's webhook will block the policy request.

When **externallyManaged** is set to **false**, the SR-IOV Network Operator automatically creates and manages the VFs, including resetting them if necessary.

To use VFs on the host system, you must create them through NMState, and set **externallyManaged** to **true**. In this mode, the SR-IOV Network Operator does not modify the PF or the manually managed VFs, except for those explicitly defined in the **nicSelector** field of your policy. However, the SR-IOV Network Operator continues to manage VFs that are used as pod secondary interfaces.

- 10 The NIC selector identifies the device to which this resource applies. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally.

If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.

- 11 Optional: The vendor hexadecimal vendor identifier of the SR-IOV network device. The only allowed values are **8086** (Intel) and **15b3** (Mellanox).

- 12 Optional: The device hexadecimal device identifier of the SR-IOV network device. For example, **101b** is the device ID for a Mellanox ConnectX-6 device.

- 13 Optional: An array of one or more physical function (PF) names the resource must apply to.

- 14 Optional: An array of one or more PCI bus addresses the resource must apply to. For example **0000:02:00.1**.

- 15 Optional: The platform-specific network filter. The only supported platform is Red Hat OpenStack Platform (RHOSP). Acceptable values use the following format: **openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx**. Replace **xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** with the value from the **/var/config/openstack/latest/network\_data.json** metadata file. This filter ensures that VFs are associated with a specific OpenStack network. The operator uses this filter to map the VFs to the appropriate network based on metadata provided by the OpenStack platform.

- 16 Optional: The driver to configure for the VFs created from this resource. The only allowed values are **netdevice** and **vfio-pci**. The default value is **netdevice**.

For a Mellanox NIC to work in DPDK mode on bare metal nodes, use the **netdevice** driver type and set **isRdma** to **true**.

- 17 Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**.

If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode.

Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.

**NOTE**

You cannot set the **isRdma** parameter to **true** for intel NICs.

- 18** Optional: The link type for the VFs. The default value is **eth** for Ethernet. Change this value to 'ib' for InfiniBand.

When **linkType** is set to **ib**, **isRdma** is automatically set to **true** by the SR-IOV Network Operator webhook. When **linkType** is set to **ib**, **deviceType** should not be set to **vfio-pci**.

Do not set linkType to **eth** for SriovNetworkNodePolicy, because this can lead to an incorrect number of available devices reported by the device plugin.

- 19** Optional: To enable hardware offloading, you must set the **eSwitchMode** field to "**switchdev**". For more information about hardware offloading, see "Configuring hardware offloading".
- 20** Optional: To exclude advertising an SR-IOV network resource's NUMA node to the Topology Manager, set the value to **true**. The default value is **false**.

### 17.2.1.1. SR-IOV network node configuration examples

The following example describes the configuration for an InfiniBand device:

#### Example configuration for an InfiniBand device

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: <sriov_resource_name>
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: <num>
  nicSelector:
    vendor: "<vendor_code>"
    deviceID: "<device_id>"
    rootDevices:
      - "<pci_bus_id>"
  linkType: <link_type>
  isRdma: true
# ...
```

The following example describes the configuration for an SR-IOV network device in a RHOSP virtual machine:

#### Example configuration for an SR-IOV device in a virtual machine

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name>
  namespace: openshift-sriov-network-operator
```

```

spec:
  resourceName: <sriov_resource_name>
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 1 ①
  nicSelector:
    vendor: "<vendor_code>"
    deviceID: "<device_id>"
    netFilter: "openstack/NetworkID:ea24bd04-8674-4f69-b0ee-fa0b3bd20509" ②
# ...

```

- ① When configuring the node network policy for a virtual machine, the **numVfs** parameter is always set to **1**.
- ② When the virtual machine is deployed on RHOSP, the **netFilter** parameter must refer to a network ID. Valid values for **netFilter** are available from an **SriovNetworkNodeState** object.

### 17.2.1.2. Automated discovery of SR-IOV network devices

The SR-IOV Network Operator searches your cluster for SR-IOV capable network devices on worker nodes. The Operator creates and updates a **SriovNetworkNodeState** custom resource (CR) for each worker node that provides a compatible SR-IOV network device.

The CR is assigned the same name as the worker node. The **status.interfaces** list provides information about the network devices on a node.



#### IMPORTANT

Do not modify a **SriovNetworkNodeState** object. The Operator creates and manages these resources automatically.

#### 17.2.1.2.1. Example **SriovNetworkNodeState** object

The following YAML is an example of a **SriovNetworkNodeState** object created by the SR-IOV Network Operator:

#### An **SriovNetworkNodeState** object

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 ①
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: ②

```

```

- deviceID: "1017"
  driver: mlx5_core
  mtu: 1500
  name: ens785f0
  pciAddress: "0000:18:00.0"
  totalvfs: 8
  vendor: 15b3
- deviceID: "1017"
  driver: mlx5_core
  mtu: 1500
  name: ens785f1
  pciAddress: "0000:18:00.1"
  totalvfs: 8
  vendor: 15b3
- deviceID: 158b
  driver: i40e
  mtu: 1500
  name: ens817f0
  pciAddress: 0000:81:00.0
  totalvfs: 64
  vendor: "8086"
- deviceID: 158b
  driver: i40e
  mtu: 1500
  name: ens817f1
  pciAddress: 0000:81:00.1
  totalvfs: 64
  vendor: "8086"
- deviceID: 158b
  driver: i40e
  mtu: 1500
  name: ens803f0
  pciAddress: 0000:86:00.0
  totalvfs: 64
  vendor: "8086"
syncStatus: Succeeded

```

- 1 The value of the **name** field is the same as the name of the worker node.
- 2 The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

### 17.2.1.3. Configuring the SR-IOV Network Operator on Mellanox cards when Secure Boot is enabled

The SR-IOV Network Operator supports an option to skip the firmware configuration for Mellanox devices. This option allows you to create virtual functions by using the SR-IOV Network Operator when the system has secure boot enabled. You must manually configure and allocate the number of virtual functions in the firmware before switching the system to secure boot.



#### NOTE

The number of virtual functions in the firmware is the maximum number of virtual functions that you can request in the policy.

## Procedure

- Configure the virtual functions (VFs) by running the following command when the system is without a secure boot when using the sriov-config daemon:

```
$ mstconfig -d -0001:b1:00.1 set SRIOV_EN=1 NUM_OF_VFS=16 ① ②
```

- ① The **SRIOV\_EN** environment variable enables the SR-IOV Network Operator support on the Mellanox card.
- ② The **NUM\_OF\_VFS** environment variable specifies the number of virtual functions to enable in the firmware.

- Configure the SR-IOV Network Operator by disabling the Mellanox plugin. See the following **SriovOperatorConfig** example configuration:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector: {}
  configurationMode: daemon
  disableDrain: false
  disablePlugins:
    - mellanox
  enableInjector: true
  enableOperatorWebhook: true
  logLevel: 2
```

- Reboot the system to enable the virtual functions and the configuration settings.
- Check the virtual functions (VFs) after rebooting the system by running the following command:

```
$ oc -n openshift-sriov-network-operator get sriovnetworknodestate.sriovnetwork.openshift.io
worker-0 -oyaml
```

## Example output

```
- devicID: 101d
  driver: mlx5_core
  eSwitchMode: legacy
  linkSpeed: -1 Mb/s
  linkType: ETH
  mac: 08:c0:eb:96:31:25
  mtu: 1500
  name: ens3f1np1
  pciAddress: 0000:b1:00.1 ①
  totalvfs: 16
  vendor: 15b3
```

- 1** The **totalvfs** value is the same number used in the **mstconfig** command earlier in the procedure.

5. Enable secure boot to prevent unauthorized operating systems and malicious software from loading during the device's boot process.

- a. Enable secure boot using the BIOS (Basic Input/Output System).

Secure Boot: Enabled  
Secure Boot Policy: Standard  
Secure Boot Mode: Mode Deployed

- b. Reboot the system.

#### 17.2.1.4. Virtual function (VF) partitioning for SR-IOV devices

In some cases, you might want to split virtual functions (VFs) from the same physical function (PF) into multiple resource pools. For example, you might want some of the VFs to load with the default driver and the remaining VFs load with the **vfio-pci** driver. In such a deployment, the **pfNames** selector in your **SriovNetworkNodePolicy** custom resource (CR) can be used to specify a range of VFs for a pool using the following format: **<pfname>#<first\_vf>-<last\_vf>**.

For example, the following YAML shows the selector for an interface named **netpf0** with VF **2** through **7**:

pfNames: ["**netpf0#2-7**"]

- **netpf0** is the PF interface name.
- **2** is the first VF index (0-based) that is included in the range.
- **7** is the last VF index (0-based) that is included in the range.

You can select VFs from the same PF by using different policy CRs if the following requirements are met:

- The **numVfs** value must be identical for policies that select the same PF.
- The VF index must be in the range of **0** to **<numVfs>-1**. For example, if you have a policy with **numVfs** set to **8**, then the **<first\_vf>** value must not be smaller than **0**, and the **<last\_vf>** must not be larger than **7**.
- The VFs ranges in different policies must not overlap.
- The **<first\_vf>** must not be larger than the **<last\_vf>**.

The following example illustrates NIC partitioning for an SR-IOV device.

The policy **policy-net-1** defines a resource pool **net-1** that contains the VF **0** of PF **netpf0** with the default VF driver. The policy **policy-net-1-dpdk** defines a resource pool **net-1-dpdk** that contains the VF **8** to **15** of PF **netpf0** with the **vfio** VF driver.

Policy **policy-net-1**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
```

```

metadata:
  name: policy-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#0-0"]
  deviceType: netdevice

```

Policy **policy-net-1-dpdk**:

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1-dpdk
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#8-15"]
  deviceType: vfio-pci

```

### Verifying that the interface is successfully partitioned

Confirm that the interface partitioned to virtual functions (VFs) for the SR-IOV device by running the following command.

\$ ip link show <interface> ①

- 1 Replace **<interface>** with the interface that you specified when partitioning to VFs for the SR-IOV device, for example, **ens3f1**.

### Example output

```

5: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
link/ether 3c:fd:fe:d1:bc:01 brd ff:ff:ff:ff:ff:ff

vf 0  link/ether 5a:e7:88:25:ea:a0 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 1  link/ether 3e:1d:36:d7:3d:49 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 2  link/ether ce:09:56:97:df:f9 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 3  link/ether 5e:91:cf:88:d1:38 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 4  link/ether e6:06:a1:96:2f:de brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off

```

#### 17.2.1.5. A test pod template for clusters that use SR-IOV on OpenStack

The following **testpmd** pod demonstrates container creation with huge pages, reserved CPUs, and the SR-IOV port.

### An example testpmd pod

```

apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
  # ...
spec:
  containers:
    - name: testpmd
      command: ["sleep", "99999"]
      image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
      securityContext:
        capabilities:
          add: ["IPC_LOCK", "SYS_ADMIN"]
        privileged: true
        runAsUser: 0
      resources:
        requests:
          memory: 1000Mi
          hugepages-1Gi: 1Gi
          cpu: '2'
          openshift.io/sriov1: 1
        limits:
          hugepages-1Gi: 1Gi
          cpu: '2'
          memory: 1000Mi
          openshift.io/sriov1: 1
      volumeMounts:
        - mountPath: /dev/hugepages
          name: hugepage
          readOnly: False
  runtimeClassName: performance-cnf-performanceprofile ①
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

- ① This example assumes that the name of the performance profile is **cnf-performance profile**.

#### 17.2.1.6. A test pod template for clusters that use OVS hardware offloading on OpenStack

The following **testpmd** pod demonstrates Open vSwitch (OVS) hardware offloading on Red Hat OpenStack Platform (RHOSP).

### An example testpmd pod

```

apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    k8s.v1.cni.cncf.io/networks: hwoffload1
spec:
  runtimeClassName: performance-cnf-performanceprofile ①
  containers:
    - name: testpmd
      command: ["sleep", "99999"]
      image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
      securityContext:
        capabilities:
          add: ["IPC_LOCK", "SYS_ADMIN"]
        privileged: true
        runAsUser: 0
      resources:
        requests:
          memory: 1000Mi
          hugepages-1Gi: 1Gi
          cpu: '2'
        limits:
          hugepages-1Gi: 1Gi
          cpu: '2'
          memory: 1000Mi
      volumeMounts:
        - mountPath: /mnt/huge
          name: hugepage
          readOnly: False
      volumes:
        - name: hugepage
          emptyDir:
            medium: HugePages

```

- ① If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

#### 17.2.1.7. Huge pages resource injection for Downward API

When a pod specification includes a resource request or limit for huge pages, the Network Resources Injector automatically adds Downward API fields to the pod specification to provide the huge pages information to the container.

The Network Resources Injector adds a volume that is named **podnetinfo** and is mounted at **/etc/podnetinfo** for each container in the pod. The volume uses the Downward API and includes a file for huge pages requests and limits. The file naming convention is as follows:

- **/etc/podnetinfo/hugepages\_1G\_request\_<container-name>**
- **/etc/podnetinfo/hugepages\_1G\_limit\_<container-name>**
- **/etc/podnetinfo/hugepages\_2M\_request\_<container-name>**

- `/etc/podnetinfo/hugepages_2M_limit_<container-name>`

The paths specified in the previous list are compatible with the **app-netutil** library. By default, the library is configured to search for resource information in the `/etc/podnetinfo` directory. If you choose to specify the Downward API path items yourself manually, the **app-netutil** library searches for the following paths in addition to the paths in the previous list.

- `/etc/podnetinfo/hugepages_request`
- `/etc/podnetinfo/hugepages_limit`
- `/etc/podnetinfo/hugepages_1G_request`
- `/etc/podnetinfo/hugepages_1G_limit`
- `/etc/podnetinfo/hugepages_2M_request`
- `/etc/podnetinfo/hugepages_2M_limit`

As with the paths that the Network Resources Injector can create, the paths in the preceding list can optionally end with a `_<container-name>` suffix.

### 17.2.2. Configuring SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition to OpenShift Container Platform. You can configure an SR-IOV network device by creating a SriovNetworkNodePolicy custom resource (CR).



#### NOTE

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes. Reboot only happens in the following cases:

- With Mellanox NICs (**mlx5** driver) a node reboot happens every time the number of virtual functions (VFs) increase on a physical function (PF).
- With Intel NICs, a reboot only happens if the kernel parameters do not include **intel\_iommu=on** and **iommu=pt**.

It might take several minutes for a configuration change to apply.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.
- You have enough available nodes in your cluster to handle the evicted workload from drained nodes.
- You have not selected any control plane nodes for SR-IOV network device configuration.

#### Procedure

1. Create an **SriovNetworkNodePolicy** object, and then save the YAML in the `<name>-sriov-node-network.yaml` file. Replace `<name>` with the name for this configuration.
2. Optional: Label the SR-IOV capable cluster nodes with **SriovNetworkNodePolicy.Spec.NodeSelector** if they are not already labeled. For more information about labeling nodes, see "Understanding how to update labels on nodes".
3. Create the **SriovNetworkNodePolicy** object:

```
$ oc create -f <name>-sriov-node-network.yaml
```

where `<name>` specifies the name for this configuration.

After applying the configuration update, all the pods in **sriov-network-operator** namespace transition to the **Running** status.

4. To verify that the SR-IOV network device is configured, enter the following command. Replace `<node_name>` with the name of a node with the SR-IOV network device that you just configured.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

## Additional resources

- [Understanding how to update labels on nodes](#) .

### 17.2.3. Creating a non-uniform memory access (NUMA) aligned SR-IOV pod

You can create a NUMA aligned SR-IOV pod by restricting SR-IOV and the CPU resources allocated from the same NUMA node with **restricted** or **single-numa-node** Topology Manager policies.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have configured the CPU Manager policy to **static**. For more information on CPU Manager, see the "Additional resources" section.
- You have configured the Topology Manager policy to **single-numa-node**.



#### NOTE

When **single-numa-node** is unable to satisfy the request, you can configure the Topology Manager policy to **restricted**. For more flexible SR-IOV network resource scheduling, see *Excluding SR-IOV network topology during NUMA-aware scheduling* in the Additional resources section.

#### Procedure

1. Create the following SR-IOV pod spec, and then save the YAML in the `<name>-sriov-pod.yaml` file. Replace `<name>` with a name for this pod.

The following example shows an SR-IOV pod spec:

```

apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: <name> 1
spec:
  containers:
    - name: sample-container
      image: <image> 2
      command: ["sleep", "infinity"]
      resources:
        limits:
          memory: "1Gi" 3
          cpu: "2" 4
        requests:
          memory: "1Gi"
          cpu: "2"

```

- 1** Replace **<name>** with the name of the SR-IOV network attachment definition CR.
- 2** Replace **<image>** with the name of the **sample-pod** image.
- 3** To create the SR-IOV pod with guaranteed QoS, set **memory limits** equal to **memory requests**.
- 4** To create the SR-IOV pod with guaranteed QoS, set **cpu limits** equals to **cpu requests**.

2. Create the sample SR-IOV pod by running the following command:

```
$ oc create -f <filename> 1
```

- 1** Replace **<filename>** with the name of the file you created in the previous step.

3. Confirm that the **sample-pod** is configured with guaranteed QoS.

```
$ oc describe pod sample-pod
```

4. Confirm that the **sample-pod** is allocated with exclusive CPUs.

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. Confirm that the SR-IOV device and CPUs that are allocated for the **sample-pod** are on the same NUMA node.

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

#### 17.2.4. Exclude the SR-IOV network topology for NUMA-aware scheduling

You can exclude advertising the Non-Uniform Memory Access (NUMA) node for the SR-IOV network to the Topology Manager for more flexible SR-IOV network deployments during NUMA-aware pod scheduling.

In some scenarios, it is a priority to maximize CPU and memory resources for a pod on a single NUMA node. By not providing a hint to the Topology Manager about the NUMA node for the pod’s SR-IOV network resource, the Topology Manager can deploy the SR-IOV network resource and the pod CPU and memory resources to different NUMA nodes. This can add to network latency because of the data transfer between NUMA nodes. However, it is acceptable in scenarios when workloads require optimal CPU and memory performance.

For example, consider a compute node, **compute-1**, that features two NUMA nodes: **numa0** and **numa1**. The SR-IOV-enabled NIC is present on **numa0**. The CPUs available for pod scheduling are present on **numa1** only. By setting the **excludeTopology** specification to **true**, the Topology Manager can assign CPU and memory resources for the pod to **numa1** and can assign the SR-IOV network resource for the same pod to **numa0**. This is only possible when you set the **excludeTopology** specification to **true**. Otherwise, the Topology Manager attempts to place all resources on the same NUMA node.

### 17.2.5. Troubleshooting SR-IOV configuration

After following the procedure to configure an SR-IOV network device, the following sections address some error conditions.

To display the state of nodes, run the following command:

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name>
```

where: **<node\_name>** specifies the name of a node with an SR-IOV network device.

#### Error output: Cannot allocate memory

```
"lastSyncError": "write /sys/bus/pci/devices/0000:3b:00.1/sriov_numvfs: cannot allocate memory"
```

When a node indicates that it cannot allocate memory, check the following items:

- Confirm that global SR-IOV settings are enabled in the BIOS for the node.
- Confirm that VT-d is enabled in the BIOS for the node.

#### Additional resources

- [Using CPU Manager](#)

### 17.2.6. Next steps

- [Configuring an SR-IOV network attachment](#)

## 17.3. CONFIGURING AN SR-IOV ETHERNET NETWORK ATTACHMENT

You can configure an Ethernet network attachment for an Single Root I/O Virtualization (SR-IOV) device in the cluster.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.3.1. Ethernet device configuration object

You can configure an Ethernet network device by defining an **SriovNetwork** object.

The following YAML describes an **SriovNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> ①
  namespace: openshift-sriov-network-operator ②
spec:
  resourceName: <sriov_resource_name> ③
  networkNamespace: <target_namespace> ④
  vlan: <vlan> ⑤
  spoofChk: "<spoof_check>" ⑥
  ipam: |- ⑦
    {}
  linkState: <link_state> ⑧
  maxTxRate: <max_tx_rate> ⑨
  minTxRate: <min_tx_rate> ⑩
  vlanQoS: <vlan_qos> ⑪
  trust: "<trust_vf>" ⑫
  capabilities: <capabilities> ⑬
```

- ① A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.
- ② The namespace where the SR-IOV Network Operator is installed.
- ③ The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.
- ④ The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.
- ⑤ Optional: A Virtual LAN (VLAN) ID for the additional network. The integer value must be from **0** to **4095**. The default value is **0**.
- ⑥ Optional: The spoof check mode of the VF. The allowed values are the strings "**on**" and "**off**".

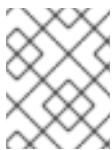


#### IMPORTANT

You must enclose the value you specify in quotes or the object is rejected by the SR-IOV Network Operator.

- ⑦ A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.
- ⑧ Optional: The link state of virtual function (VF). Allowed value are **enable**, **disable** and **auto**.
- ⑨ Optional: A maximum transmission rate, in Mbps, for the VF.
- ⑩ Optional: A minimum transmission rate, in Mbps, for the VF. This value must be less than or equal to

the maximum transmission rate.



### NOTE

Intel NICs do not support the **minTxRate** parameter. For more information, see [BZ#1772847](#).

- 11 Optional: An IEEE 802.1p priority level for the VF. The default value is **0**.
- 12 Optional: The trust mode of the VF. The allowed values are the strings "**on**" and "**off**".



### IMPORTANT

You must enclose the value that you specify in quotes, or the SR-IOV Network Operator rejects the object.

- 13 Optional: The capabilities to configure for this additional network. You can specify '{ "ips": true }' to enable IP address support or '{ "mac": true }' to enable MAC address support.

#### 17.3.1.1. Creating a configuration for assignment of dual-stack IP addresses dynamically

Dual-stack IP address assignment can be configured with the **ipRanges** parameter for:

- IPv4 addresses
- IPv6 addresses
- multiple IP address assignment

#### Procedure

1. Set **type** to **whereabouts**.
2. Use **ipRanges** to allocate IP addresses as shown in the following example:

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
    - name: whereabouts-shim
      namespace: default
      type: Raw
      rawCNIConfig: |- 
        {
          "name": "whereabouts-dual-stack",
          "cniVersion": "0.3.1",
          "type": "bridge",
          "ipam": {
            "type": "whereabouts",
            "ipRanges": [
              {"range": "192.168.10.0/24"},
              {"range": "2001:db8::/64"}
            ]
          }
        }
```

```
[  
}  
}
```

3. Attach network to a pod. For more information, see "Adding a pod to a secondary network".
4. Verify that all IP addresses are assigned.
5. Run the following command to ensure the IP addresses are assigned as metadata.

```
$ oc exec -it mypod -- ip a
```

### 17.3.1.2. Configuration of IP address assignment for a network attachment

For secondary networks, IP addresses can be assigned using an IP Address Management (IPAM) CNI plugin, which supports various assignment methods, including Dynamic Host Configuration Protocol (DHCP) and static assignment.

The DHCP IPAM CNI plugin responsible for dynamic assignment of IP addresses operates with two distinct components:

- **CNI Plugin**: Responsible for integrating with the Kubernetes networking stack to request and release IP addresses.
- **DHCP IPAM CNI Daemon**: A listener for DHCP events that coordinates with existing DHCP servers in the environment to handle IP address assignment requests. This daemon is *not* a DHCP server itself.

For networks requiring **type: dhcp** in their IPAM configuration, ensure the following:

- A DHCP server is available and running in the environment. The DHCP server is external to the cluster and is expected to be part of the customer's existing network infrastructure.
- The DHCP server is appropriately configured to serve IP addresses to the nodes.

In cases where a DHCP server is unavailable in the environment, it is recommended to use the Whereabouts IPAM CNI plugin instead. The Whereabouts CNI provides similar IP address management capabilities without the need for an external DHCP server.



#### NOTE

Use the Whereabouts CNI plugin when there is no external DHCP server or where static IP address management is preferred. The Whereabouts plugin includes a reconciler daemon to manage stale IP address allocations.

A DHCP lease must be periodically renewed throughout the container's lifetime, so a separate daemon, the DHCP IPAM CNI Daemon, is required. To deploy the DHCP IPAM CNI daemon, modify the Cluster Network Operator (CNO) configuration to trigger the deployment of this daemon as part of the secondary network setup.

#### 17.3.1.2.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

**Table 17.2. ipam static configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>static</b> is required.
<b>addresses</b>	<b>array</b>	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
<b>routes</b>	<b>array</b>	An array of objects specifying routes to configure inside the pod.
<b>dns</b>	<b>array</b>	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

Table 17.3. **ipam.addresses[]** array

Field	Type	Description
<b>address</b>	<b>string</b>	An IP address and network prefix that you specify. For example, if you specify <b>10.10.21.10/24</b> , then the secondary network is assigned an IP address of <b>10.10.21.10</b> and the netmask is <b>255.255.255.0</b> .
<b>gateway</b>	<b>string</b>	The default gateway to route egress network traffic to.

Table 17.4. **ipam.routes[]** array

Field	Type	Description
<b>dst</b>	<b>string</b>	The IP address range in CIDR format, such as <b>192.168.17.0/24</b> or <b>0.0.0.0/0</b> for the default route.
<b>gw</b>	<b>string</b>	The gateway where network traffic is routed.

Table 17.5. **ipam.dns** object

Field	Type	Description
<b>nameservers</b>	<b>array</b>	An array of one or more IP addresses for to send DNS queries to.
<b>domain</b>	<b>array</b>	The default domain to append to a hostname. For example, if the domain is set to <b>example.com</b> , a DNS lookup query for <b>example-host</b> is rewritten as <b>example-host.example.com</b> .
<b>search</b>	<b>array</b>	An array of domain names to append to an unqualified hostname, such as <b>example-host</b> , during a DNS lookup query.

## Static IP address assignment configuration example

```
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}
```

### 17.3.1.2.2. Dynamic IP address (DHCP) assignment configuration

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.



#### IMPORTANT

For an Ethernet network attachment, the SR-IOV Network Operator does not create a DHCP server deployment; the Cluster Network Operator is responsible for creating the minimal DHCP server deployment.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

#### Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
    #
  # ...
```

The following table describes the configuration parameters for dynamic IP address assignment with DHCP.

**Table 17.6. ipam DHCP configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>dhcp</b> is required.

The following JSON example describes the configuration p for dynamic IP address assignment with DHCP.

### Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

#### 17.3.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to a secondary network without the use of a DHCP server.

The Whereabouts CNI plugin also supports overlapping IP address ranges and configuration of the same CIDR range multiple times within separate **NetworkAttachmentDefinition** CRDs. This provides greater flexibility and management capabilities in multi-tenant environments.

##### 17.3.1.2.3.1. Dynamic IP address configuration objects

The following table describes the configuration objects for dynamic IP address assignment with Whereabouts:

Table 17.7. ipam whereabouts configuration object

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>whereabouts</b> is required.
<b>range</b>	<b>string</b>	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
<b>exclude</b>	<b>array</b>	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.
<b>network_name</b>	<b>string</b>	Optional: Helps ensure that each group or domain of pods gets its own set of IP addresses, even if they share the same range of IP addresses. Setting this field is important for keeping networks separate and organized, notably in multi-tenant environments.

#### 17.3.1.2.3.2. Dynamic IP address assignment configuration that uses Whereabouts

The following example shows a dynamic address assignment configuration that uses Whereabouts:

## Whereabouts dynamic IP address assignment

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

### 17.3.1.2.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges

The following example shows a dynamic IP address assignment that uses overlapping IP address ranges for multi-tenant networks.

#### NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", ①
  }
}
```

- ① Optional. If set, must match the **network\_name** of NetworkAttachmentDefinition 2.

#### NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", ①
  }
}
```

- ① Optional. If set, must match the **network\_name** of NetworkAttachmentDefinition 1.

## 17.3.2. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating an **SriovNetwork** object. When you create an **SriovNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.

**NOTE**

Do not modify or delete an **SriovNetwork** object if it is attached to any pods in a **running** state.

**Prerequisites**

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

**Procedure**

1. Create a **SriovNetwork** object, and then save the YAML in the **<name>.yaml** file, where **<name>** is a name for this additional network. The object specification might resemble the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |- 
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
```

2. To create the object, enter the following command:

```
$ oc create -f <name>.yaml
```

where **<name>** specifies the name of the additional network.

3. Optional: To confirm that the **NetworkAttachmentDefinition** object that is associated with the **SriovNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the networkNamespace you specified in the **SriovNetwork** object.

```
$ oc get net-attach-def -n <namespace>
```

**17.3.3. Assigning an SR-IOV network to a VRF**

As a cluster administrator, you can assign an SR-IOV network interface to your VRF domain by using the CNI VRF plugin.

To do this, add the VRF configuration to the optional **metaPlugins** parameter of the **SriovNetwork** resource.

**NOTE**

Applications that use VRFs need to bind to a specific device. The common usage is to use the **SO\_BINDTODEVICE** option for a socket. **SO\_BINDTODEVICE** binds the socket to a device that is specified in the passed interface name, for example, **eth1**. To use **SO\_BINDTODEVICE**, the application must have **CAP\_NET\_RAW** capabilities.

Using a VRF through the **ip vrf exec** command is not supported in OpenShift Container Platform pods. To use VRF, bind applications directly to the VRF interface.

### 17.3.3.1. Creating an additional SR-IOV network attachment with the CNI VRF plugin

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.

**NOTE**

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To create an additional SR-IOV network attachment with the CNI VRF plugin, perform the following procedure.

#### Prerequisites

- Install the OpenShift Container Platform CLI (oc).
- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

#### Procedure

1. Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR. Save the YAML as the file **sriov-network-attachment.yaml**.

#### Example SriovNetwork custom resource (CR) example

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: example-network
  namespace: additional-sriov-network-1
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {
          "dst": "0.0.0.0/0"
        }],
      "gateway": "10.56.217.1"
    }
```

```
vlan: 0
resourceName: intelnics
metaPlugins : |
{
  "type": "vrf", ①
  "vrfname": "example-vrf-name" ②
}
```

- ① **type** must be set to **vrf**.
- ② **vrfname** is the name of the VRF that the interface is assigned to. If it does not exist in the pod, it is created.

## 2. Create the **SriovNetwork** resource:

```
$ oc create -f sriov-network-attachment.yaml
```

### Verifying that the **NetworkAttachmentDefinition** CR is successfully created

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

```
$ oc get network-attachment-definitions -n <namespace> ①
```

- ① Replace **<namespace>** with the namespace that you specified when configuring the network attachment, for example, **additional-sriov-network-1**.

### Example output

NAME	AGE
additional-sriov-network-1	14m



### NOTE

There might be a delay before the SR-IOV Network Operator creates the CR.

### Verifying that the additional SR-IOV network attachment is successful

To verify that the VRF CNI is correctly configured and that the additional SR-IOV network attachment is attached, do the following:

1. Create an SR-IOV network that uses the VRF CNI.
2. Assign the network to a pod.
3. Verify that the pod network attachment is connected to the SR-IOV additional network. Remote shell into the pod and run the following command:

```
$ ip vrf show
```

### Example output

Name	Table
red	10

4. Confirm that the VRF interface is **master** of the secondary interface by running the following command:

```
$ ip link
```

#### Example output

```
...
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red
state UP mode
```

```
...
```

#### 17.3.4. Runtime configuration for an Ethernet-based SR-IOV attachment

When attaching a pod to an additional network, you can specify a runtime configuration to make specific customizations for the pod. For example, you can request a specific MAC hardware address.

You specify the runtime configuration by setting an annotation in the pod specification. The annotation key is **k8s.v1.cni.cncf.io/networks**, and it accepts a JSON object that describes the runtime configuration.

The following JSON describes the runtime configuration options for an Ethernet-based SR-IOV network attachment.

```
[  
  {  
    "name": "<name>", ①  
    "mac": "<mac_address>", ②  
    "ips": ["<cidr_range>"] ③  
  }  
,
```

- ① The name of the SR-IOV network attachment definition CR.
- ② Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify `{"mac": true}` in the **SriovNetwork** object.
- ③ Optional: IP addresses for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify `{"ips": true}` in the **SriovNetwork** object.

#### Example runtime configuration

```
apiVersion: v1  
kind: Pod  
metadata:  
  name: sample-pod
```

```

annotations:
k8s.v1.cni.cncf.io/networks: |-  

[  

 {  

 "name": "net1",  

 "mac": "20:04:0f:f1:88:01",  

 "ips": ["192.168.10.1/24", "2001::1/64"]  

 }  

]  

spec:  

containers:  

- name: sample-container  

image: <image>  

imagePullPolicy: IfNotPresent  

command: ["sleep", "infinity"]

```

### 17.3.5. Adding a pod to a secondary network

You can add a pod to a secondary network. The pod continues to send normal cluster-related network traffic over the default network.

When a pod is created, a secondary networks is attached to the pod. However, if a pod already exists, you cannot attach a secondary network to it.

The pod must be in the same namespace as the secondary network.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

#### Procedure

1. Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

- To attach a secondary network without any customization, add an annotation with the following format. Replace **<network>** with the name of the secondary network to associate with the pod:

```

metadata:  

annotations:  

k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] ①

```

- ①** To specify more than one secondary network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same secondary network multiple times, that pod will have multiple network interfaces attached to that network.

- To attach a secondary network with customizations, add an annotation with the following format:

```

metadata:  

annotations:

```

```
k8s.v1.cni.cncf.io/networks: |-  
[  
 {  
   "name": "<network>", ①  
   "namespace": "<namespace>", ②  
   "default-route": ["<default-route>"] ③  
 }  
]
```

- ① Specify the name of the secondary network defined by a **NetworkAttachmentDefinition** object.
- ② Specify the namespace where the **NetworkAttachmentDefinition** object is defined.
- ③ Optional: Specify an override for the default route, such as **192.168.17.1**.

2. To create the pod, enter the following command. Replace **<name>** with the name of the pod.

```
$ oc create -f <name>.yaml
```

3. Optional: To Confirm that the annotation exists in the **Pod** CR, enter the following command, replacing **<name>** with the name of the pod.

```
$ oc get pod <name> -o yaml
```

In the following example, the **example-pod** pod is attached to the **net1** secondary network:

```
$ oc get pod example-pod -o yaml  
apiVersion: v1  
kind: Pod  
metadata:  
  annotations:  
    k8s.v1.cni.cncf.io/networks: macvlan-bridge  
    k8s.v1.cni.cncf.io/network-status: |- ①  
      [  
        {"  
          "name": "ovn-kubernetes",  
          "interface": "eth0",  
          "ips": [  
            "10.128.2.14"  
          ],  
          "default": true,  
          "dns": {}  
        },  
        {"  
          "name": "macvlan-bridge",  
          "interface": "net1",  
          "ips": [  
            "20.2.2.100"  
          ],  
          "mac": "22:2f:60:a5:f8:00",  
          "dns": {}  
        }  
      ]  
    name: example-pod  
    namespace: default  
    spec:
```

...

status:

...

- 1 The **k8s.v1.cni.cncf.io/network-status** parameter is a JSON array of objects. Each object describes the status of a secondary network attached to the pod. The annotation value is stored as a plain text value.

### 17.3.5.1. Exposing MTU for vfio-pci SR-IOV devices to pod

After adding a pod to an additional network, you can check that the MTU is available for the SR-IOV network.

#### Procedure

- 1 Check that the pod annotation includes MTU by running the following command:

```
$ oc describe pod example-pod
```

The following example shows the sample output:

```
"mac": "20:04:0f:f1:88:01",
  "mtu": 1500,
  "dns": {},
  "device-info": {
    "type": "pci",
    "version": "1.1.0",
    "pci": {
      "pci-address": "0000:86:01.3"
    }
}
```

- 2 Verify that the MTU is available in **/etc/podnetinfo** inside the pod by running the following command:

```
$ oc exec example-pod -n sriov-tests -- cat /etc/podnetinfo/annotations | grep mtu
```

The following example shows the sample output:

```
k8s.v1.cni.cncf.io/network-status="[{
  "name": "ovn-kubernetes",
  "interface": "eth0",
  "ips": [
    "10.131.0.67"
  ],
  "mac": "0a:58:0a:83:00:43",
  "default": true,
  "dns": {}
}, {
  "name": "sriov-tests/sriov-nic-1",
  "interface": "net1",
  "ips": [
    "192.168.10.1"
  ]
}]
```

```
],
  "mac": "20:04:0f:f1:88:01",
  "mtu": 1500,
  "dns": {},
  "device-info": {
    "type": "pci",
    "version": "1.1.0",
    "pci": {
      "pci-address": "0000:86:01.3"
    }
  }
}]"
```

### 17.3.6. Configuring parallel node draining during SR-IOV network policy updates

By default, the SR-IOV Network Operator drains workloads from a node before every policy change. The Operator performs this action, one node at a time, to ensure that no workloads are affected by the reconfiguration.

In large clusters, draining nodes sequentially can be time-consuming, taking hours or even days. In time-sensitive environments, you can enable parallel node draining in an **SriovNetworkPoolConfig** custom resource (CR) for faster rollouts of SR-IOV network configurations.

To configure parallel draining, use the **SriovNetworkPoolConfig** CR to create a node pool. You can then add nodes to the pool and define the maximum number of nodes in the pool that the Operator can drain in parallel. With this approach, you can enable parallel draining for faster reconfiguration while ensuring you still have enough nodes remaining in the pool to handle any running workloads.



#### NOTE

A node can only belong to one SR-IOV network pool configuration. If a node is not part of a pool, it is added to a virtual, default, pool that is configured to drain one node at a time only.

The node might restart during the draining process.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the SR-IOV Network Operator.
- Nodes have hardware that support SR-IOV.

#### Procedure

1. Create a **SriovNetworkPoolConfig** resource:
  - a. Create a YAML file that defines the **SriovNetworkPoolConfig** resource:

#### Example sriov-nw-pool.yaml file

```
apiVersion: v1
```

```

kind: SriovNetworkPoolConfig
metadata:
  name: pool-1 1
  namespace: openshift-sriov-network-operator 2
spec:
  maxUnavailable: 2 3
  nodeSelector: 4
  matchLabels:
    node-role.kubernetes.io/worker: ""

```

- 1** Specify the name of the **SriovNetworkPoolConfig** object.
- 2** Specify namespace where the SR-IOV Network Operator is installed.
- 3** Specify an integer number, or percentage value, for nodes that can be unavailable in the pool during an update. For example, if you have 10 nodes and you set the maximum unavailable to 2, then only 2 nodes can be drained in parallel at any time, leaving 8 nodes for handling workloads.
- 4** Specify the nodes to add the pool by using the node selector. This example adds all nodes with the **worker** role to the pool.

- b. Create the **SriovNetworkPoolConfig** resource by running the following command:

```
$ oc create -f sriov-nw-pool.yaml
```

2. Create the **sriov-test** namespace by running the following command:

```
$ oc create namespace sriov-test
```

3. Create a **SriovNetworkNodePolicy** resource:

- a. Create a YAML file that defines the **SriovNetworkNodePolicy** resource:

#### Example sriov-node-policy.yaml file

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    pfNames: ["ens1"]
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  numVfs: 5
  priority: 99
  resourceName: sriov_nic_1

```

- b. Create the **SriovNetworkNodePolicy** resource by running the following command:

```
$ oc create -f sriov-node-policy.yaml
```

4. Create a **SriovNetwork** resource:

- Create a YAML file that defines the **SriovNetwork** resource:

**Example sriov-network.yaml file**

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nic-1
  namespace: openshift-sriov-network-operator
spec:
  linkState: auto
  networkNamespace: sriov-test
  resourceName: sriov_nic_1
  capabilities: '{ "mac": true, "ips": true }'
  ipam: '{ "type": "static" }'
```

- Create the **SriovNetwork** resource by running the following command:

```
$ oc create -f sriov-network.yaml
```

## Verification

- View the node pool you created by running the following command:

```
$ oc get sriovNetworkpoolConfig -n openshift-sriov-network-operator
```

**Example output**

NAME	AGE
pool-1	67s <b>①</b>

- ①** In this example, **pool-1** contains all the nodes with the **worker** role.

To demonstrate the node draining process using the example scenario from the above procedure, complete the following steps:

- Update the number of virtual functions in the **SriovNetworkNodePolicy** resource to trigger workload draining in the cluster:

```
$ oc patch SriovNetworkNodePolicy sriov-nic-1 -n openshift-sriov-network-operator --type merge -p '{"spec": {"numVfs": 4}}'
```

- Monitor the draining status on the target cluster by running the following command:

```
$ oc get sriovNetworkNodeState -n openshift-sriov-network-operator
```

**Example output**

NAMESPACE	NAME	SYNC STATUS	DESIRED SYNC STATE
CURRENT SYNC STATE	AGE		
openshift-sriov-network-operator	worker-0	InProgress	Drain_Required
DrainComplete	3d10h		
openshift-sriov-network-operator	worker-1	InProgress	Drain_Required
DrainComplete	3d10h		

When the draining process is complete, the **SYNC STATUS** changes to **Succeeded**, and the **DESIRED SYNC STATE** and **CURRENT SYNC STATE** values return to **IDLE**.

### Example output

NAMESPACE	NAME	SYNC STATUS	DESIRED SYNC STATE
CURRENT SYNC STATE	AGE		
openshift-sriov-network-operator	worker-0	Succeeded	Idle
3d10h			
openshift-sriov-network-operator	worker-1	Succeeded	Idle
3d10h			

### 17.3.7. Excluding the SR-IOV network topology for NUMA-aware scheduling

To exclude advertising the SR-IOV network resource's Non-Uniform Memory Access (NUMA) node to the Topology Manager, you can configure the **excludeTopology** specification in the **SriovNetworkNodePolicy** custom resource. Use this configuration for more flexible SR-IOV network deployments during NUMA-aware pod scheduling.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have configured the CPU Manager policy to **static**. For more information about CPU Manager, see the *Additional resources* section.
- You have configured the Topology Manager policy to **single-numa-node**.
- You have installed the SR-IOV Network Operator.

#### Procedure

1. Create the **SriovNetworkNodePolicy** CR:
  - a. Save the following YAML in the **sriov-network-node-policy.yaml** file, replacing values in the YAML to match your environment:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <policy_name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 ①
  nodeSelector:
    kubernetes.io/hostname: <node_name>
  numVfs: <number_of_Vfs>
```

```

nicSelector: ②
  vendor: "<vendor_ID>"
  deviceID: "<device_ID>"
deviceType: netdevice
excludeTopology: true ③

```

- ① The resource name of the SR-IOV network device plugin. This YAML uses a sample **resourceName** value.
- ② Identify the device for the Operator to configure by using the NIC selector.
- ③ To exclude advertising the NUMA node for the SR-IOV network resource to the Topology Manager, set the value to **true**. The default value is **false**.



#### NOTE

If multiple **SriovNetworkNodePolicy** resources target the same SR-IOV network resource, the **SriovNetworkNodePolicy** resources must have the same value as the **excludeTopology** specification. Otherwise, the conflicting policy is rejected.

- b. Create the **SriovNetworkNodePolicy** resource by running the following command:

```
$ oc create -f sriov-network-node-policy.yaml
```

#### Example output

```
sriovnetworknodepolicy.sriovnetwork.openshift.io/policy-for-numa-0 created
```

2. Create the **SriovNetwork** CR:

- a. Save the following YAML in the **sriov-network.yaml** file, replacing values in the YAML to match your environment:

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-numa-0-network ①
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 ②
  networkNamespace: <namespace> ③
  ipam: |- ④
  {
    "type": "<ipam_type>",
  }

```

- ① Replace **sriov-numa-0-network** with the name for the SR-IOV network resource.
- ② Specify the resource name for the **SriovNetworkNodePolicy** CR from the previous step. This YAML uses a sample **resourceName** value.

- 3** Enter the namespace for your SR-IOV network resource.
- 4** Enter the IP address management configuration for the SR-IOV network.

b. Create the **SriovNetwork** resource by running the following command:

```
$ oc create -f sriov-network.yaml
```

#### Example output

```
sriovnetwork.sriovnetwork.openshift.io/sriov-numa-0-network created
```

3. Create a pod and assign the SR-IOV network resource from the previous step:

- a. Save the following YAML in the **sriov-network-pod.yaml** file, replacing values in the YAML to match your environment:

```
apiVersion: v1
kind: Pod
metadata:
  name: <pod_name>
  annotations:
    k8s.v1.cni.cncf.io/networks: |->
      [
        {
          "name": "sriov-numa-0-network", 1
        }
      ]
spec:
  containers:
    - name: <container_name>
      image: <image>
      imagePullPolicy: IfNotPresent
      command: ["sleep", "infinity"]
```

- 1** This is the name of the **SriovNetwork** resource that uses the **SriovNetworkNodePolicy** resource.

b. Create the **Pod** resource by running the following command:

```
$ oc create -f sriov-network-pod.yaml
```

#### Example output

```
pod/example-pod created
```

### Verification

1. Verify the status of the pod by running the following command, replacing **<pod\_name>** with the name of the pod:

```
$ oc get pod <pod_name>
```

## Example output

NAME	READY	STATUS	RESTARTS	AGE
test-deployment-sriov-76cbbf4756-k9v72	1/1	Running	0	45h

2. Open a debug session with the target pod to verify that the SR-IOV network resources are deployed to a different node than the memory and CPU resources.

- a. Open a debug session with the pod by running the following command, replacing <pod\_name> with the target pod name.

```
$ oc debug pod/<pod_name>
```

- b. Set **/host** as the root directory within the debug shell. The debug pod mounts the root file system from the host in **/host** within the pod. By changing the root directory to **/host**, you can run binaries from the host file system:

```
$ chroot /host
```

- c. View information about the CPU allocation by running the following commands:

```
$ lscpu | grep NUMA
```

## Example output

```
NUMA node(s): 2
NUMA node0 CPU(s): 0,2,4,6,8,10,12,14,16,18, ...
NUMA node1 CPU(s): 1,3,5,7,9,11,13,15,17,19, ...
```

```
$ cat /proc/self/status | grep Cpus
```

## Example output

```
Cpus_allowed: aa
Cpus_allowed_list: 1,3,5,7
```

```
$ cat /sys/class/net/net1/device numa_node
```

## Example output

```
0
```

In this example, CPUs 1,3,5, and 7 are allocated to **NUMA node1** but the SR-IOV network resource can use the NIC in **NUMA node0**.



### NOTE

If the **excludeTopology** specification is set to **True**, it is possible that the required resources exist in the same NUMA node.

### 17.3.8. Additional resources

- [Configuring an SR-IOV network device](#)
- [Using CPU Manager](#)

## 17.4. CONFIGURING AN SR-IOV INFINIBAND NETWORK ATTACHMENT

You can configure an InfiniBand (IB) network attachment for an Single Root I/O Virtualization (SR-IOV) device in the cluster.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.4.1. InfiniBand device configuration object

You can configure an InfiniBand (IB) network device by defining an **SriovIBNetwork** object.

The following YAML describes an **SriovIBNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: <name> ①
  namespace: openshift-sriov-network-operator ②
spec:
  resourceName: <sriov_resource_name> ③
  networkNamespace: <target_namespace> ④
  ipam: |- ⑤
    {}
  linkState: <link_state> ⑥
  capabilities: <capabilities> ⑦
```

- ① A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.
- ② The namespace where the SR-IOV Operator is installed.
- ③ The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.
- ④ The target namespace for the **SriovIBNetwork** object. Only pods in the target namespace can attach to the network device.
- ⑤ Optional: A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.
- ⑥ Optional: The link state of virtual function (VF). Allowed values are **enable**, **disable** and **auto**.
- ⑦ Optional: The capabilities to configure for this network. You can specify '{ "ips": true }' to enable IP address support or '{ "infinibandGUID": true }' to enable IB Global Unique Identifier (GUID) support.

### 17.4.1.1. Creating a configuration for assignment of dual-stack IP addresses dynamically

Dual-stack IP address assignment can be configured with the **ipRanges** parameter for:

- IPv4 addresses
- IPv6 addresses
- multiple IP address assignment

#### Procedure

1. Set **type** to **whereabouts**.
2. Use **ipRanges** to allocate IP addresses as shown in the following example:

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
    - name: whereabouts-shim
      namespace: default
      type: Raw
      rawCNIConfig: |-  
        {
          "name": "whereabouts-dual-stack",
          "cniVersion": "0.3.1",
          "type": "bridge",
          "ipam": {
              "type": "whereabouts",
              "ipRanges": [
                  {"range": "192.168.10.0/24"},
                  {"range": "2001:db8::/64"}
              ]
          }
      }
  }
```

3. Attach network to a pod. For more information, see "Adding a pod to a secondary network".
4. Verify that all IP addresses are assigned.
5. Run the following command to ensure the IP addresses are assigned as metadata.

```
$ oc exec -it mypod -- ip a
```

### 17.4.1.2. Configuration of IP address assignment for a network attachment

For secondary networks, IP addresses can be assigned using an IP Address Management (IPAM) CNI plugin, which supports various assignment methods, including Dynamic Host Configuration Protocol (DHCP) and static assignment.

The DHCP IPAM CNI plugin responsible for dynamic assignment of IP addresses operates with two distinct components:

- **CNI Plugin**: Responsible for integrating with the Kubernetes networking stack to request and release IP addresses.
- **DHCP IPAM CNI Daemon**: A listener for DHCP events that coordinates with existing DHCP servers in the environment to handle IP address assignment requests. This daemon is *not* a DHCP server itself.

For networks requiring **type: dhcp** in their IPAM configuration, ensure the following:

- A DHCP server is available and running in the environment. The DHCP server is external to the cluster and is expected to be part of the customer's existing network infrastructure.
- The DHCP server is appropriately configured to serve IP addresses to the nodes.

In cases where a DHCP server is unavailable in the environment, it is recommended to use the Whereabouts IPAM CNI plugin instead. The Whereabouts CNI provides similar IP address management capabilities without the need for an external DHCP server.



### NOTE

Use the Whereabouts CNI plugin when there is no external DHCP server or where static IP address management is preferred. The Whereabouts plugin includes a reconciler daemon to manage stale IP address allocations.

A DHCP lease must be periodically renewed throughout the container's lifetime, so a separate daemon, the DHCP IPAM CNI Daemon, is required. To deploy the DHCP IPAM CNI daemon, modify the Cluster Network Operator (CNO) configuration to trigger the deployment of this daemon as part of the secondary network setup.

#### 17.4.1.2.1. Static IP address assignment configuration

The following table describes the configuration for static IP address assignment:

**Table 17.8. ipam static configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>static</b> is required.
<b>addresses</b>	<b>array</b>	An array of objects specifying IP addresses to assign to the virtual interface. Both IPv4 and IPv6 IP addresses are supported.
<b>routes</b>	<b>array</b>	An array of objects specifying routes to configure inside the pod.
<b>dns</b>	<b>array</b>	Optional: An array of objects specifying the DNS configuration.

The **addresses** array requires objects with the following fields:

**Table 17.9. ipam.addresses[] array**

Field	Type	Description
<b>address</b>	<b>string</b>	An IP address and network prefix that you specify. For example, if you specify <b>10.10.21.10/24</b> , then the secondary network is assigned an IP address of <b>10.10.21.10</b> and the netmask is <b>255.255.255.0</b> .
<b>gateway</b>	<b>string</b>	The default gateway to route egress network traffic to.

Table 17.10. ipam.routes[] array

Field	Type	Description
<b>dst</b>	<b>string</b>	The IP address range in CIDR format, such as <b>192.168.17.0/24</b> or <b>0.0.0.0/0</b> for the default route.
<b>gw</b>	<b>string</b>	The gateway where network traffic is routed.

Table 17.11. ipam.dns object

Field	Type	Description
<b>nameservers</b>	<b>array</b>	An array of one or more IP addresses for to send DNS queries to.
<b>domain</b>	<b>array</b>	The default domain to append to a hostname. For example, if the domain is set to <b>example.com</b> , a DNS lookup query for <b>example-host</b> is rewritten as <b>example-host.example.com</b> .
<b>search</b>	<b>array</b>	An array of domain names to append to an unqualified hostname, such as <b>example-host</b> , during a DNS lookup query.

### Static IP address assignment configuration example

```
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}
```

#### 17.4.1.2.2. Dynamic IP address (DHCP) assignment configuration

A pod obtains its original DHCP lease when it is created. The lease must be periodically renewed by a minimal DHCP server deployment running on the cluster.



## IMPORTANT

For an Ethernet network attachment, the SR-IOV Network Operator does not create a DHCP server deployment; the Cluster Network Operator is responsible for creating the minimal DHCP server deployment.

To trigger the deployment of the DHCP server, you must create a shim network attachment by editing the Cluster Network Operator configuration, as in the following example:

### Example shim network attachment definition

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
    - name: dhcp-shim
      namespace: default
      type: Raw
      rawCNIConfig: |-
        {
          "name": "dhcp-shim",
          "cniVersion": "0.3.1",
          "type": "bridge",
          "ipam": {
            "type": "dhcp"
          }
        }
      # ...
```

The following table describes the configuration parameters for dynamic IP address assignment with DHCP.

**Table 17.12. ipam DHCP configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>dhcp</b> is required.

The following JSON example describes the configuration p for dynamic IP address assignment with DHCP.

### Dynamic IP address (DHCP) assignment configuration example

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

#### 17.4.1.2.3. Dynamic IP address assignment configuration with Whereabouts

The Whereabouts CNI plugin allows the dynamic assignment of an IP address to a secondary network without the use of a DHCP server.

The Whereabouts CNI plugin also supports overlapping IP address ranges and configuration of the same CIDR range multiple times within separate **NetworkAttachmentDefinition** CRDs. This provides greater flexibility and management capabilities in multi-tenant environments.

#### 17.4.1.2.3.1. Dynamic IP address configuration objects

The following table describes the configuration objects for dynamic IP address assignment with Whereabouts:

**Table 17.13. ipam whereabouts configuration object**

Field	Type	Description
<b>type</b>	<b>string</b>	The IPAM address type. The value <b>whereabouts</b> is required.
<b>range</b>	<b>string</b>	An IP address and range in CIDR notation. IP addresses are assigned from within this range of addresses.
<b>exclude</b>	<b>array</b>	Optional: A list of zero or more IP addresses and ranges in CIDR notation. IP addresses within an excluded address range are not assigned.
<b>network_name</b>	<b>string</b>	Optional: Helps ensure that each group or domain of pods gets its own set of IP addresses, even if they share the same range of IP addresses. Setting this field is important for keeping networks separate and organized, notably in multi-tenant environments.

#### 17.4.1.2.3.2. Dynamic IP address assignment configuration that uses Whereabouts

The following example shows a dynamic address assignment configuration that uses Whereabouts:

##### Whereabouts dynamic IP address assignment

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

#### 17.4.1.2.3.3. Dynamic IP address assignment that uses Whereabouts with overlapping IP address ranges

The following example shows a dynamic IP address assignment that uses overlapping IP address ranges for multi-tenant networks.

## NetworkAttachmentDefinition 1

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/29",
    "network_name": "example_net_common", ①
  }
}
```

- ① Optional. If set, must match the **network\_name** of NetworkAttachmentDefinition 2.

## NetworkAttachmentDefinition 2

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/24",
    "network_name": "example_net_common", ①
  }
}
```

- ① Optional. If set, must match the **network\_name** of NetworkAttachmentDefinition 1.

### 17.4.2. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating an **SriovIBNetwork** object. When you create an **SriovIBNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.



#### NOTE

Do not modify or delete an **SriovIBNetwork** object if it is attached to any pods in a **running** state.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create a **SriovIBNetwork** object, and then save the YAML in the **<name>.yaml** file, where **<name>** is a name for this additional network. The object specification might resemble the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
```

```

spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |->
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
  
```

2. To create the object, enter the following command:

```
$ oc create -f <name>.yaml
```

where **<name>** specifies the name of the additional network.

3. Optional: To confirm that the **NetworkAttachmentDefinition** object that is associated with the **SriovIBNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the networkNamespace you specified in the **SriovIBNetwork** object.

```
$ oc get net-attach-def -n <namespace>
```

#### 17.4.3. Runtime configuration for an InfiniBand-based SR-IOV attachment

When attaching a pod to an additional network, you can specify a runtime configuration to make specific customizations for the pod. For example, you can request a specific MAC hardware address.

You specify the runtime configuration by setting an annotation in the pod specification. The annotation key is **k8s.v1.cni.cncf.io/networks**, and it accepts a JSON object that describes the runtime configuration.

The following JSON describes the runtime configuration options for an InfiniBand-based SR-IOV network attachment.

```

[>
  {
    "name": "<network_attachment>", ①
    "infiniband-guid": "<guid>", ②
    "ips": ["<cidr_range>"] ③
  }
]
  
```

- 1 The name of the SR-IOV network attachment definition CR.
- 2 The InfiniBand GUID for the SR-IOV device. To use this feature, you also must specify **{"infinibandGUID": true}** in the **SriovIBNetwork** object.
- 3 The IP addresses for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{"ips": true}** in the **SriovIBNetwork** object.

## Example runtime configuration

```

apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |->
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
    - name: sample-container
      image: <image>
      imagePullPolicy: IfNotPresent
      command: ["sleep", "infinity"]

```

### 17.4.4. Adding a pod to a secondary network

You can add a pod to a secondary network. The pod continues to send normal cluster-related network traffic over the default network.

When a pod is created, a secondary networks is attached to the pod. However, if a pod already exists, you cannot attach a secondary network to it.

The pod must be in the same namespace as the secondary network.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster.

#### Procedure

- 1 Add an annotation to the **Pod** object. Only one of the following annotation formats can be used:

- To attach a secondary network without any customization, add an annotation with the following format. Replace **<network>** with the name of the secondary network to associate with the pod:

```

  metadata:
    annotations:
      k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] ①

```

- 1 To specify more than one secondary network, separate each network with a comma. Do not include whitespace between the comma. If you specify the same secondary network multiple times, that pod will have multiple network interfaces attached to that network.

- b. To attach a secondary network with customizations, add an annotation with the following format:

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-  

    [  

      {  

        "name": "<network>", ①  

        "namespace": "<namespace>", ②  

        "default-route": ["<default-route>"] ③  

      }  

    ]
```

- ① Specify the name of the secondary network defined by a **NetworkAttachmentDefinition** object.
- ② Specify the namespace where the **NetworkAttachmentDefinition** object is defined.
- ③ Optional: Specify an override for the default route, such as **192.168.17.1**.

2. To create the pod, enter the following command. Replace **<name>** with the name of the pod.

```
$ oc create -f <name>.yaml
```

3. Optional: To Confirm that the annotation exists in the **Pod** CR, enter the following command, replacing **<name>** with the name of the pod.

```
$ oc get pod <name> -o yaml
```

In the following example, the **example-pod** pod is attached to the **net1** secondary network:

```
$ oc get pod example-pod -o yaml  

apiVersion: v1  

kind: Pod  

metadata:  

  annotations:  

    k8s.v1.cni.cncf.io/networks: macvlan-bridge  

    k8s.v1.cni.cncf.io/network-status: |- ①  

    [{  

      "name": "ovn-kubernetes",  

      "interface": "eth0",  

      "ips": [  

        "10.128.2.14"  

      ],  

      "default": true,  

      "dns": {}  

    },  

    {  

      "name": "macvlan-bridge",  

      "interface": "net1",  

      "ips": [  

        "20.2.2.100"  

      ],  

      "mac": "22:2f:60:a5:f8:00",  

      "status": {}  

    }]
```

```

        "dns": {}
    }]
name: example-pod
namespace: default
spec:
...
status:
...

```

- 1 The **k8s.v1.cni.cncf.io/network-status** parameter is a JSON array of objects. Each object describes the status of a secondary network attached to the pod. The annotation value is stored as a plain text value.

#### 17.4.4.1. Exposing MTU for vfio-pci SR-IOV devices to pod

After adding a pod to an additional network, you can check that the MTU is available for the SR-IOV network.

##### Procedure

- Check that the pod annotation includes MTU by running the following command:

```
$ oc describe pod example-pod
```

The following example shows the sample output:

```

"mac": "20:04:0f:f1:88:01",
"mtu": 1500,
"dns": {},
"device-info": {
    "type": "pci",
    "version": "1.1.0",
    "pci": {
        "pci-address": "0000:86:01.3"
    }
}

```

- Verify that the MTU is available in **/etc/podnetinfo** inside the pod by running the following command:

```
$ oc exec example-pod -n sriov-tests -- cat /etc/podnetinfo/annotations | grep mtu
```

The following example shows the sample output:

```

k8s.v1.cni.cncf.io/network-status="[{
    "name": "ovn-kubernetes",
    "interface": "eth0",
    "ips": [
        "10.131.0.67"
    ],
    "mac": "0a:58:0a:83:00:43",
    "default": true,
    "dns": {}
}
]

```

```

    },
    "name": "sriov-tests/sriov-nic-1",
    "interface": "net1",
    "ips": [
        "192.168.10.1"
    ],
    "mac": "20:04:0f:f1:88:01",
    "mtu": 1500,
    "dns": {},
    "device-info": {
        "type": "pci",
        "version": "1.1.0",
        "pci": {
            "pci-address": "0000:86:01.3"
        }
    }
}
]]"

```

#### 17.4.5. Additional resources

- [Configuring an SR-IOV network device](#)
- [Using CPU Manager](#)
- [Exclude SR-IOV network topology for NUMA-aware scheduling](#)

### 17.5. CONFIGURING AN RDMA SUBSYSTEM FOR SR-IOV

Remote Direct Memory Access (RDMA) allows direct memory access between two systems without involving the operating system of either system. You can configure an RDMA Container Network Interface (CNI) on Single Root I/O Virtualization (SR-IOV) to enable high-performance, low-latency communication between containers. When you combine RDMA with SR-IOV, you provide a mechanism to expose hardware counters of Mellanox Ethernet devices for use inside Data Plane Development Kit (DPDK) applications.

#### 17.5.1. Configuring SR-IOV RDMA CNI

Configure an RDMA CNI on SR-IOV.



#### NOTE

This procedure applies only to Mellanox devices.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.

#### Procedure

- Create an **SriovNetworkPoolConfig** CR and save it as **sriov-nw-pool.yaml**, as shown in the following example:

#### Example SriovNetworkPoolConfig CR

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: worker
  namespace: openshift-sriov-network-operator
spec:
  maxUnavailable: 1
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker: ""
  rdmaMode: exclusive ①
```

- Set RDMA network namespace mode to **exclusive**.
- Create the **SriovNetworkPoolConfig** resource by running the following command:

```
$ oc create -f sriov-nw-pool.yaml
```

- Create an **SriovNetworkNodePolicy** CR and save it as **sriov-node-policy.yaml**, as shown in the following example:

#### Example SriovNetworkNodePolicy CR

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-nic-pf1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true ①
  nicSelector:
    pfNames: ["ens3f0np0"]
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  numVfs: 4
  priority: 99
  resourceName: sriov_nic_pf1
```

- Activate RDMA mode.
- Create the **SriovNetworkNodePolicy** resource by running the following command:

```
$ oc create -f sriov-node-policy.yaml
```

- Create an **SriovNetwork** CR and save it as **sriov-network.yaml**, as shown in the following example:

## Example SriovNetwork CR

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-nic-pf1
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: sriov-tests
  resourceName: sriov_nic_pf1
  ipam: |-  
    metaPlugins: |  
      {  
        "type": "rdma" ①  
      }  
    }
```

- ① Create the RDMA plugin.

6. Create the **SriovNetwork** resource by running the following command:

```
$ oc create -f sriov-network.yaml
```

## Verification

1. Create a **Pod** CR and save it as **sriov-test-pod.yaml**, as shown in the following example:

### Example runtime configuration

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
annotations:
  k8s.v1.cni.cncf.io/networks: |-  
    [  
      {  
        "name": "net1",  
        "mac": "20:04:0f:f1:88:01",  
        "ips": ["192.168.10.1/24", "2001::1/64"]  
      }  
    ]  
spec:  
  containers:  
    - name: sample-container  
      image: <image>  
      imagePullPolicy: IfNotPresent  
      command: ["sleep", "infinity"]
```

2. Create the test pod by running the following command:

```
$ oc create -f sriov-test-pod.yaml
```

3. Log in to the test pod by running the following command:

```
$ oc rsh testpod1 -n sriov-tests
```

- Verify that the path to the **hw-counters** directory exists by running the following command:

```
$ ls
/sys/bus/pci/devices/${PCIDEVICE_OPENSHIFT_IO_SRIOV_NIC_PF1}/infiniband/*/ports/1/hw
_counters/
```

### Example output

```
duplicate_request    out_of_buffer req_cqe_flush_error      resp_cqe_flush_error
roce_adp_retrans    roce_slow_restart_trans
implied_nak_seq_err out_of_sequence req_remote_access_errors
resp_local_length_error  roce_adp_retrans_to rx_atomic_requests
lifespan            packet_seq_err req_remote_invalid_request resp_remote_access_errors
roce_slow_restart   rx_read_requests
local_ack_timeout_err req_cqe_error resp_cqe_error      rnr_nak_retry_err
roce_slow_restart_cnps rx_write_requests
```

## 17.6. CONFIGURING INTERFACE-LEVEL NETWORK SYSCTL SETTINGS AND ALL-MULTICAST MODE FOR SR-IOV NETWORKS

As a cluster administrator, you can change interface-level network sysctls and several interface attributes such as promiscuous mode, all-multicast mode, MTU, and MAC address by using the tuning Container Network Interface (CNI) meta plugin for a pod connected to a SR-IOV network device.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.6.1. Labeling nodes with an SR-IOV enabled NIC

If you want to enable SR-IOV on only SR-IOV capable nodes there are a couple of ways to do this:

- Install the Node Feature Discovery (NFD) Operator. NFD detects the presence of SR-IOV enabled NICs and labels the nodes with **node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = true**.
- Examine the **SriovNetworkNodeState** CR for each node. The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the SR-IOV Network Operator on the worker node. Label each node with **feature.node.kubernetes.io/network-sriov.capable: "true"** by using the following command:

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```



#### NOTE

You can label the nodes with whatever name you want.

### 17.6.2. Setting one sysctl flag

You can set interface-level network **sysctl** settings for a pod connected to a SR-IOV network device.

In this example, `net.ipv4.conf.IFNAME.accept_redirects` is set to **1** on the created virtual interfaces.

The **sysctl-tuning-test** is a namespace used in this example.

- Use the following command to create the **sysctl-tuning-test** namespace:

```
$ oc create namespace sysctl-tuning-test
```

### 17.6.2.1. Setting one sysctl flag on nodes with SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** custom resource definition (CRD) to OpenShift Container Platform. You can configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** custom resource (CR).



#### NOTE

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain and reboot the nodes.

It can take several minutes for a configuration change to apply.

Follow this procedure to create a **SriovNetworkNodePolicy** custom resource (CR).

#### Procedure

- 1 Create an **SriovNetworkNodePolicy** custom resource (CR). For example, save the following YAML as the file **policyoneflag-sriov-node-network.yaml**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyoneflag 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyoneflag 3
  nodeSelector: 4
    feature.node.kubernetes.io/network-sriov.capable="true"
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
    pfNames: ["ens5"] 8
  deviceType: "netdevice" 9
  isRdma: false 10
```

- 1** The name for the custom resource object.
- 2** The namespace where the SR-IOV Network Operator is installed.
- 3** The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.
- 4**

The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and

- 5 Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.
- 6 The number of the virtual functions (VFs) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.
- 7 The NIC selector identifies the device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.
- 8 Optional: An array of one or more physical function (PF) names for the device.
- 9 Optional: The driver type for the virtual functions. The only allowed value is **netdevice**. For a Mellanox NIC to work in DPDK mode on bare metal nodes, set **isRdma** to **true**.
- 10 Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**. If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode. Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.



#### NOTE

The **vfio-pci** driver type is not supported.

## 2. Create the **SriovNetworkNodePolicy** object:

```
$ oc create -f policyoneflag-sriov-node-network.yaml
```

After applying the configuration update, all the pods in **sriov-network-operator** namespace change to the **Running** status.

## 3. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node\_name>** with the name of a node with the SR-IOV network device that you just configured.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

#### Example output

```
Succeeded
```

### 17.6.2.2. Configuring sysctl on a SR-IOV network

You can set interface specific **sysctl** settings on virtual interfaces created by SR-IOV by adding the tuning configuration to the optional **metaPlugins** parameter of the **SriovNetwork** resource.

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.



## NOTE

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To change the interface-level network **net.ipv4.conf.IFNAME.accept\_redirects sysctl** settings, create an additional SR-IOV network with the Container Network Interface (CNI) tuning plugin.

## Prerequisites

- Install the OpenShift Container Platform CLI (oc).
- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

## Procedure

- 1 Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR. Save the YAML as the file **sriov-network-interface-sysctl.yaml**.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: onevalidflag ①
  namespace: openshift-sriov-network-operator ②
spec:
  resourceName: policyoneflag ③
  networkNamespace: sysctl-tuning-test ④
  ipam: '{ "type": "static" }' ⑤
  capabilities: '{ "mac": true, "ips": true }' ⑥
  metaPlugins : | ⑦
    {
      "type": "tuning",
      "capabilities":{
        "mac":true
      },
      "sysctl":{
        "net.ipv4.conf.IFNAME.accept_redirects": "1"
      }
    }
```

- 1 A name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.
- 2 The namespace where the SR-IOV Network Operator is installed.
- 3

The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.

- ④ The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.
- ⑤ A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.
- ⑥ Optional: Set capabilities for the additional network. You can specify "`{"ips": true}`" to enable IP address support or "`{"mac": true}`" to enable MAC address support.
- ⑦ Optional: The metaPlugins parameter is used to add additional capabilities to the device. In this use case set the **type** field to **tuning**. Specify the interface-level network **sysctl** you want to set in the **sysctl** field.

## 2. Create the **SriovNetwork** resource:

```
$ oc create -f sriov-network-interface-sysctl.yaml
```

### Verifying that the **NetworkAttachmentDefinition** CR is successfully created

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

```
$ oc get network-attachment-definitions -n <namespace> ①
```

- Replace **<namespace>** with the value for **networkNamespace** that you specified in the **SriovNetwork** object. For example, **sysctl-tuning-test**.

### Example output

NAME	AGE
onevalidflag	14m



### NOTE

There might be a delay before the SR-IOV Network Operator creates the CR.

### Verifying that the additional SR-IOV network attachment is successful

To verify that the tuning CNI is correctly configured and the additional SR-IOV network attachment is attached, do the following:

#### 1. Create a **Pod** CR. Save the following YAML as the file **examplepod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
```

```
k8s.v1.cni.cncf.io/networks: |-
[
  {
    "name": "onevalidflag", 1
    "mac": "0a:56:0a:83:04:0c", 2
    "ips": ["10.100.100.200/24"] 3
  }
]
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
```

- 1** The name of the SR-IOV network attachment definition CR.
- 2** Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify `{ "mac": true }` in the **SriovNetwork** object.
- 3** Optional: IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify `{ "ips": true }` in the **SriovNetwork** object.

## 2. Create the **Pod** CR:

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

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

```
$ oc get pod -n sysctl-tuning-test
```

### Example output

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

## 4. Log in to the pod by running the following command:

```
$ oc rsh -n sysctl-tuning-test tunepod
```

- Verify the values of the configured sysctl flag. Find the value **net.ipv4.conf.*IFNAME*.accept\_redirects** by running the following command:

```
$ sysctl net.ipv4.conf.net1.accept_redirects
```

#### Example output

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

### 17.6.3. Configuring sysctl settings for pods associated with bonded SR-IOV interface flag

You can set interface-level network **sysctl** settings for a pod connected to a bonded SR-IOV network device.

In this example, the specific network interface-level **sysctl** settings that can be configured are set on the bonded interface.

The **sysctl-tuning-test** is a namespace used in this example.

- Use the following command to create the **sysctl-tuning-test** namespace:

```
$ oc create namespace sysctl-tuning-test
```

#### 17.6.3.1. Setting all sysctl flag on nodes with bonded SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** custom resource definition (CRD) to OpenShift Container Platform. You can configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** custom resource (CR).



#### NOTE

When applying the configuration specified in a SriovNetworkNodePolicy object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes.

It might take several minutes for a configuration change to apply.

Follow this procedure to create a **SriovNetworkNodePolicy** custom resource (CR).

#### Procedure

- Create an **SriovNetworkNodePolicy** custom resource (CR). Save the following YAML as the file **policyallflags-sriov-node-network.yaml**. Replace **policyallflags** with the name for the configuration.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyallflags 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyallflags 3
```

```

nodeSelector: ④
  node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = `true`
priority: 10 ⑤
numVfs: 5 ⑥
nicSelector: ⑦
  pfNames: ["ens1f0"] ⑧
deviceType: "netdevice" ⑨
isRdma: false ⑩

```

- ① The name for the custom resource object.
- ② The namespace where the SR-IOV Network Operator is installed.
- ③ The resource name of the SR-IOV network device plugin. You can create multiple SR-IOV network node policies for a resource name.
- ④ The node selector specifies the nodes to configure. Only SR-IOV network devices on the selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed on selected nodes only.
- ⑤ Optional: The priority is an integer value between **0** and **99**. A smaller value receives higher priority. For example, a priority of **10** is a higher priority than **99**. The default value is **99**.
- ⑥ The number of virtual functions (VFs) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.
- ⑦ The NIC selector identifies the device for the Operator to configure. You do not have to specify values for all the parameters. It is recommended to identify the network device with enough precision to avoid selecting a device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceId**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they refer to the same device. If you specify a value for **netFilter**, then you do not need to specify any other parameter because a network ID is unique.
- ⑧ Optional: An array of one or more physical function (PF) names for the device.
- ⑨ Optional: The driver type for the virtual functions. The only allowed value is **netdevice**. For a Mellanox NIC to work in DPDK mode on bare metal nodes, set **isRdma** to **true**.
- ⑩ Optional: Configures whether to enable remote direct memory access (RDMA) mode. The default value is **false**. If the **isRdma** parameter is set to **true**, you can continue to use the RDMA-enabled VF as a normal network device. A device can be used in either mode. Set **isRdma** to **true** and additionally set **needVhostNet** to **true** to configure a Mellanox NIC for use with Fast Datapath DPDK applications.



#### NOTE

The **vfio-pci** driver type is not supported.

2. Create the SriovNetworkNodePolicy object:

```
$ oc create -f policyallflags-sriov-node-network.yaml
```

After applying the configuration update, all the pods in sriov-network-operator namespace change to the **Running** status.

- To verify that the SR-IOV network device is configured, enter the following command. Replace **<node\_name>** with the name of a node with the SR-IOV network device that you just configured.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

#### Example output

```
Succeeded
```

### 17.6.3.2. Configuring sysctl on a bonded SR-IOV network

You can set interface specific **sysctl** settings on a bonded interface created from two SR-IOV interfaces. Do this by adding the tuning configuration to the optional **Plugins** parameter of the bond network attachment definition.



#### NOTE

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

To change specific interface-level network **sysctl** settings create the **SriovNetwork** custom resource (CR) with the Container Network Interface (CNI) tuning plugin by using the following procedure.

#### Prerequisites

- Install the OpenShift Container Platform CLI (oc).
- Log in to the OpenShift Container Platform cluster as a user with cluster-admin privileges.

#### Procedure

- Create the **SriovNetwork** custom resource (CR) for the bonded interface as in the following example CR. Save the YAML as the file **sriov-network-attachment.yaml**.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: allvalidflags 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyallflags 3
  networkNamespace: sysctl-tuning-test 4
  capabilities: '{ "mac": true, "ips": true }' 5
```

**1** A name for the object. The SR-IOV Network Operator creates a NetworkAttachmentDefinition object with same name.

- 2 The namespace where the SR-IOV Network Operator is installed.
- 3 The value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.
- 4 The target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.
- 5 Optional: The capabilities to configure for this additional network. You can specify "{ "ips": true }" to enable IP address support or "{ "mac": true }" to enable MAC address support.

2. Create the **SriovNetwork** resource:

```
$ oc create -f sriov-network-attachment.yaml
```

3. Create a bond network attachment definition as in the following example CR. Save the YAML as the file **sriov-bond-network-interface.yaml**.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-sysctl-network
  namespace: sysctl-tuning-test
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "bound-net",
    "plugins": [
      {
        "type": "bond", ①
        "mode": "active-backup", ②
        "failOverMac": 1, ③
        "linksInContainer": true, ④
        "miimon": "100",
        "links": [ ⑤
          {"name": "net1"},
          {"name": "net2"}
        ],
        "ipam": { ⑥
          "type": "static"
        }
      },
      {
        "type": "tuning", ⑦
        "capabilities": {
          "mac": true
        },
        "sysctl": {
          "net.ipv4.conf.IFNAME.accept_redirects": "0",
          "net.ipv4.conf.IFNAME.accept_source_route": "0",
          "net.ipv4.conf.IFNAME.disable_policy": "1",
          "net.ipv4.conf.IFNAME.secure_redirects": "0",
          "net.ipv4.conf.IFNAME.send_redirects": "0",
        }
      }
    ]
  }'
```

```

    "net.ipv6.conf.IFNAME.accept_redirects": "0",
    "net.ipv6.conf.IFNAME.accept_source_route": "1",
    "net.ipv6.neigh.IFNAME.base_reachable_time_ms": "20000",
    "net.ipv6.neigh.IFNAME.retrans_time_ms": "2000"
  }
}
]
'

```

- 1 The type is **bond**.
- 2 The **mode** attribute specifies the bonding mode. The bonding modes supported are:
  - **balance-rr** - 0
  - **active-backup** - 1
  - **balance-xor** - 2
 For **balance-rr** or **balance-xor** modes, you must set the **trust** mode to **on** for the SR-IOV virtual function.
- 3 The **failover** attribute is mandatory for active-backup mode.
- 4 The **linksInContainer=true** flag informs the Bond CNI that the required interfaces are to be found inside the container. By default, Bond CNI looks for these interfaces on the host which does not work for integration with SRIOV and Multus.
- 5 The **links** section defines which interfaces will be used to create the bond. By default, Multus names the attached interfaces as: "net", plus a consecutive number, starting with one.
- 6 A configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition. In this pod example IP addresses are configured manually, so in this case, **ipam** is set to static.
- 7 Add additional capabilities to the device. For example, set the **type** field to **tuning**. Specify the interface-level network **sysctl** you want to set in the sysctl field. This example sets all interface-level network **sysctl** settings that can be set.

#### 4. Create the bond network attachment resource:

```
$ oc create -f sriov-bond-network-interface.yaml
```

#### Verifying that the **NetworkAttachmentDefinition** CR is successfully created

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:

```
$ oc get network-attachment-definitions -n <namespace> ①
```

- 1 Replace **<namespace>** with the networkNamespace that you specified when configuring the network attachment, for example, **sysctl-tuning-test**.

#### Example output

NAME	AGE
bond-sysctl-network	22m
allvalidflags	47m

**NOTE**

There might be a delay before the SR-IOV Network Operator creates the CR.

## Verifying that the additional SR-IOV network resource is successful

To verify that the tuning CNI is correctly configured and the additional SR-IOV network attachment is attached, do the following:

- 1 Create a **Pod** CR. For example, save the following YAML as the file **examplepod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |->
      [
        {"name": "allvalidflags"}, ①
        {"name": "allvalidflags"},
        {
          "name": "bond-sysctl-network",
          "interface": "bond0",
          "mac": "0a:56:0a:83:04:0c", ②
          "ips": ["10.100.100.200/24"] ③
        }
      ]
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
```

- ① The name of the SR-IOV network attachment definition CR.
- ② Optional: The MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify `{ "mac": true }` in the `SriovNetwork` object.

- 3** Optional: IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses

2. Apply the YAML:

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

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

```
$ oc get pod -n sysctl-tuning-test
```

#### Example output

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

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

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5. Verify the values of the configured **sysctl** flag. Find the value **net.ipv6.neigh.*IFNAME*.base\_reachable\_time\_ms** by running the following command::

```
$ sysctl net.ipv6.neigh.bond0.base_reachable_time_ms
```

#### Example output

```
net.ipv6.neigh.bond0.base_reachable_time_ms = 20000
```

### 17.6.4. About all-multicast mode

Enabling all-multicast mode, particularly in the context of rootless applications, is critical. If you do not enable this mode, you would be required to grant the **NET\_ADMIN** capability to the pod's Security Context Constraints (SCC). If you were to allow the **NET\_ADMIN** capability to grant the pod privileges to make changes that extend beyond its specific requirements, you could potentially expose security vulnerabilities.

The tuning CNI plugin supports changing several interface attributes, including all-multicast mode. By enabling this mode, you can allow applications running on Virtual Functions (VFs) that are configured on a SR-IOV network device to receive multicast traffic from applications on other VFs, whether attached to the same or different physical functions.

#### 17.6.4.1. Enabling the all-multicast mode on an SR-IOV network

You can enable the all-multicast mode on an SR-IOV interface by:

- Adding the tuning configuration to the **metaPlugins** parameter of the **SriovNetwork** resource
- Setting the **allmulti** field to **true** in the tuning configuration

**NOTE**

Ensure that you create the virtual function (VF) with trust enabled.

The SR-IOV Network Operator manages additional network definitions. When you specify an additional SR-IOV network to create, the SR-IOV Network Operator creates the **NetworkAttachmentDefinition** custom resource (CR) automatically.

**NOTE**

Do not edit **NetworkAttachmentDefinition** custom resources that the SR-IOV Network Operator manages. Doing so might disrupt network traffic on your additional network.

Enable the all-multicast mode on a SR-IOV network by following this guidance.

## Prerequisites

- You have installed the OpenShift Container Platform CLI (oc).
- You are logged in to the OpenShift Container Platform cluster as a user with **cluster-admin** privileges.
- You have installed the SR-IOV Network Operator.
- You have configured an appropriate **SriovNetworkNodePolicy** object.

## Procedure

1. Create a YAML file with the following settings that defines a **SriovNetworkNodePolicy** object for a Mellanox ConnectX-5 device. Save the YAML file as **sriovnetpolicy-mlx.yaml**.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnetpolicy-mlx
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    deviceID: "1017"
    pfNames:
      - ens8f0np0#0-9
    rootDevices:
      - 0000:d8:00.0
    vendor: "15b3"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 10
  priority: 99
  resourceName: resourcexmlx
```

2. Optional: If the SR-IOV capable cluster nodes are not already labeled, add the **SriovNetworkNodePolicy.Spec.NodeSelector** label. For more information about labeling nodes, see "Understanding how to update labels on nodes".

3. Create the **SriovNetworkNodePolicy** object by running the following command:

```
$ oc create -f sriovnetpolicy-mlx.yaml
```

After applying the configuration update, all the pods in the **sriov-network-operator** namespace automatically move to a **Running** status.

4. Create the **enable-allmulti-test** namespace by running the following command:

```
$ oc create namespace enable-allmulti-test
```

5. Create the **SriovNetwork** custom resource (CR) for the additional SR-IOV network attachment and insert the **metaPlugins** configuration, as in the following example CR YAML, and save the file as **sriov-enable-all-multicast.yaml**.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: enableallmulti 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: enableallmulti 3
  networkNamespace: enable-allmulti-test 4
  ipam: '{ "type": "static" }' 5
  capabilities: '{ "mac": true, "ips": true }' 6
  trust: "on" 7
  metaPlugins : | 8
    {
      "type": "tuning",
      "capabilities":{
        "mac":true
      },
      "allmulti": true
    }
}
```

- 1** Specify a name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with the same name.
- 2** Specify the namespace where the SR-IOV Network Operator is installed.
- 3** Specify a value for the **spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.
- 4** Specify the target namespace for the **SriovNetwork** object. Only pods in the target namespace can attach to the additional network.
- 5** Specify a configuration object for the IPAM CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.
- 6** Optional: Set capabilities for the additional network. You can specify "{ "ips": true }" to enable IP address support or "{ "mac": true }" to enable MAC address support.

- 7 Specify the trust mode of the virtual function. This must be set to "on".
- 8 Add more capabilities to the device by using the **metaPlugins** parameter. In this use case, set the **type** field to **tuning**, and add the **allmulti** field and set it to **true**.

6. Create the **SriovNetwork** resource by running the following command:

```
$ oc create -f sriov-enable-all-multicast.yaml
```

### Verification of the NetworkAttachmentDefinition CR

- Confirm that the SR-IOV Network Operator created the **NetworkAttachmentDefinition** CR by running the following command:
 

```
$ oc get network-attachment-definitions -n <namespace> 1
```
- 1 Replace **<namespace>** with the value for **networkNamespace** that you specified in the **SriovNetwork** object. For this example, that is **enable-allmulti-test**.

### Example output

NAME	AGE
enableallmulti	14m



#### NOTE

There might be a delay before the SR-IOV Network Operator creates the CR.

1. Display information about the SR-IOV network resources by running the following command:

```
$ oc get sriovnetwork -n openshift-sriov-network-operator
```

### Verification of the additional SR-IOV network attachment

To verify that the tuning CNI is correctly configured and that the additional SR-IOV network attachment is attached, follow these steps:

1. Create a **Pod** CR. Save the following sample YAML in a file named **examplepod.yaml**:

```
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: enable-allmulti-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "enableallmulti", 1
          "mac": "0a:56:0a:83:04:0c", 2
        }
      ]
```

```

    "ips": ["10.100.100.200/24"] 3
}
]
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

```

- 1** Specify the name of the SR-IOV network attachment definition CR.
- 2** Optional: Specify the MAC address for the SR-IOV device that is allocated from the resource type defined in the SR-IOV network attachment definition CR. To use this feature, you also must specify **{"mac": true}** in the **SriovNetwork** object.
- 3** Optional: Specify the IP addresses for the SR-IOV device that are allocated from the resource type defined in the SR-IOV network attachment definition CR. Both IPv4 and IPv6 addresses are supported. To use this feature, you also must specify **{"ips": true}** in the **SriovNetwork** object.

2. Create the **Pod** CR by running the following command:

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

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

```
$ oc get pod -n enable-allmulti-test
```

#### Example output

```

NAME      READY  STATUS   RESTARTS  AGE
samplepod 1/1   Running  0        47s

```

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

```
$ oc rsh -n enable-allmulti-test samplepod
```

5. 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 ①
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 ②

```

- ① **eth0@if22** is the primary interface
- ② **net1@if24** is the secondary interface configured with the network-attachment-definition that supports the all-multicast mode (**ALLMULTI** flag)

## 17.7. CONFIGURING QINQ SUPPORT FOR SR-IOV ENABLED WORKLOADS

QinQ, formally known as 802.1Q-in-802.1Q, is a networking technique defined by IEEE 802.1ad. IEEE 802.1ad extends the IEEE 802.1Q-1998 standard and enriches VLAN capabilities by introducing an additional 802.1Q tag to packets already tagged with 802.1Q. This method is also referred to as VLAN stacking or double VLAN.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.7.1. About 802.1Q-in-802.1Q support

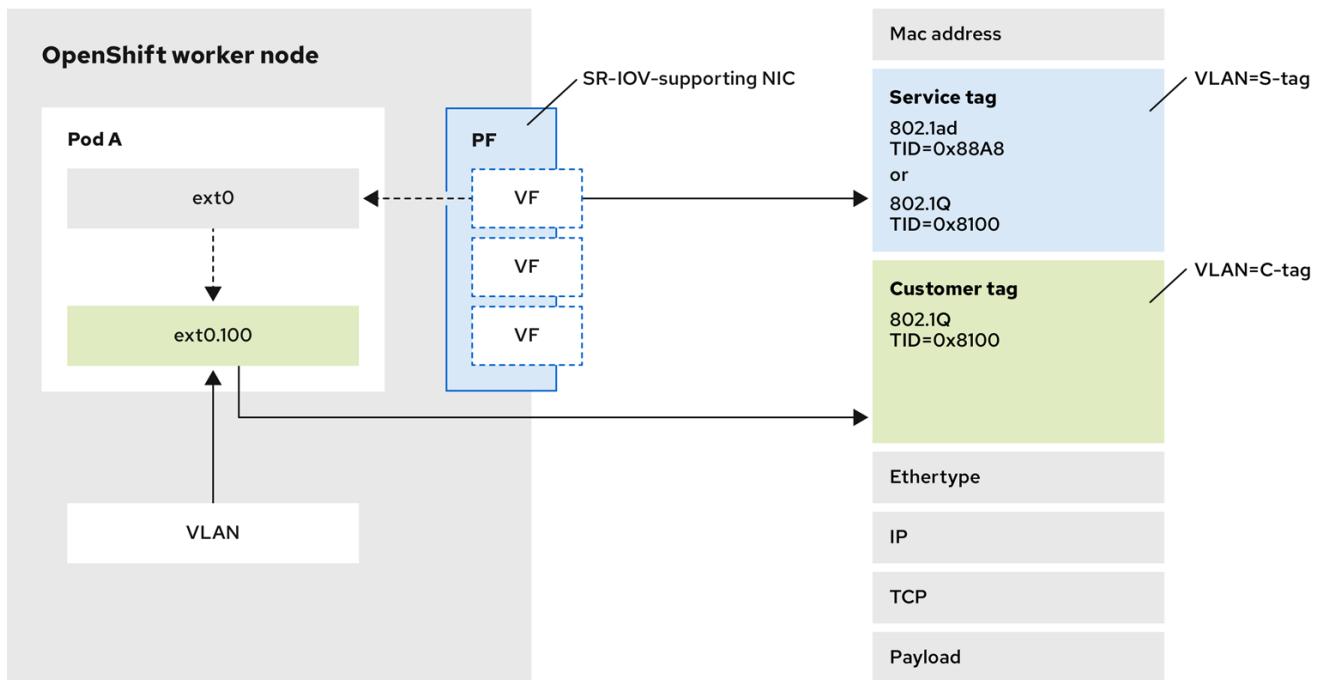
In traditional VLAN setups, frames typically contain a single VLAN tag, such as VLAN-100, as well as other metadata such as Quality of Service (QoS) bits and protocol information. QinQ introduces a second VLAN tag, where the service provider designates the outer tag for their use, offering them flexibility, while the inner tag remains dedicated to the customer's VLAN.

QinQ facilitates the creation of nested VLANs by using double VLAN tagging, enabling finer segmentation and isolation of traffic within a network environment. This approach is particularly valuable in service provider networks where you need to deliver VLAN-based services to multiple customers over a common infrastructure, while ensuring separation and isolation of traffic.

The following diagram illustrates how OpenShift Container Platform can use SR-IOV and QinQ to achieve advanced network segmentation and isolation for containerized workloads.

The diagram shows how double VLAN tagging (QinQ) works in a worker node with SR-IOV support. The SR-IOV virtual function (VF) located in the pod namespace, **ext0** is configured by the SR-IOV Container Network Interface (CNI) with a VLAN ID and VLAN protocol. This corresponds to the S-tag. Inside the pod, the VLAN CNI creates a subinterface using the primary interface **ext0**. This subinterface adds an internal VLAN ID using the 802.1Q protocol, which corresponds to the C-tag.

This demonstrates how QinQ enables finer traffic segmentation and isolation within the network. The Ethernet frame structure is detailed on the right, highlighting the inclusion of both VLAN tags, EtherType, IP, TCP, and Payload sections. QinQ facilitates the delivery of VLAN-based services to multiple customers over a shared infrastructure while ensuring traffic separation and isolation.



693\_OpenShift\_0624

The OpenShift Container Platform SR-IOV solution already supports setting the VLAN protocol on the **SriovNetwork** custom resource (CR). The virtual function (VF) can use this protocol to set the VLAN tag, also known as the outer tag. Pods can then use the VLAN CNI plugin to configure the inner tag.

**Table 17.14. Supported network interface cards**

NIC	802.1ad/802.1Q	802.1Q/802.1Q
Intel X710	No	Supported
Intel E810	Supported	Supported
Mellanox	No	Supported

## Additional resources

[Configuration for an VLAN additional network](#)

### 17.7.2. Configuring QinQ support for SR-IOV enabled workloads

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.

#### Procedure

- Create a file named **sriovnetpolicy-810-sriov-node-network.yaml** by using the following content:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnetpolicy-810
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    pfNames:
      - ens5f0#0-9
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: resource810
```

- Create the **SriovNetworkNodePolicy** object by running the following command:

```
$ oc create -f sriovnetpolicy-810-sriov-node-network.yaml
```

- Open a separate terminal window and monitor the synchronization status of the SR-IOV network node state for the node specified in the **openshift-sriov-network-operator** namespace by running the following command:

```
$ watch -n 1 'oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath=".status.syncStatus"}'
```

The synchronization status indicates a change from **InProgress** to **Succeeded**.

- Create a **SriovNetwork** object, and set the outer VLAN called the S-tag, or **Service Tag**, as it belongs to the infrastructure.



### IMPORTANT

You must configure the VLAN on the trunk interface of the switch. In addition, you might need to further configure some switches to support QinQ tagging.

- Create a file named **nad-sriovnetwork-1ad-810.yaml** by using the following content:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriovnetwork-1ad-810
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{}'
  vlan: 171 1
  vlanProto: "802.1ad" 2
  networkNamespace: default
  resourceName: resource810
```

- 1 Sets the S-tag VLAN tag to **171**.
- 2 Specifies the VLAN protocol to assign to the virtual function (VF). Supported values are **802.1ad** and **802.1q**. The default value is **802.1q**.

b. Create the object by running the following command:

```
$ oc create -f nad-sriovnetwork-1ad-810.yaml
```

5. Create a **NetworkAttachmentDefinition** object with an inner VLAN. The inner VLAN is often referred to as the C-tag, or **Customer Tag**, as it belongs to the Network Function:

a. Create a file named **nad-cvlan100.yaml** by using the following content:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: nad-cvlan100
  namespace: default
spec:
  config: {
    "name": "vlan-100",
    "cniVersion": "0.3.1",
    "type": "vlan",
    "linkInContainer": true,
    "master": "net1", 1
    "vlanId": 100,
    "ipam": {"type": "static"}
  }
```

- 1 Specifies the VF interface inside the pod. The default name is **net1** as the name is not set in the pod annotation.

b. Apply the YAML file by running the following command:

```
$ oc apply -f nad-cvlan100.yaml
```

## Verification

- Verify QinQ is active on the node by following this procedure:

1. Create a file named **test-qinq-pod.yaml** by using the following content:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: sriovnetwork-1ad-810, nad-cvlan100
spec:
  containers:
    - name: test-container
      image: quay.io/ocp-edge-qe/cnf-gotests-client:v4.10
```

```
imagePullPolicy: Always
securityContext:
  privileged: true
```

2. Create the test pod by running the following command:

```
$ oc create -f test-qinq-pod.yaml
```

3. Enter into a debug session on the target node where the pod is present and display information about the network interface **ens5f0** by running the following command:

```
$ oc debug node/my-cluster-node -- bash -c "ip link show ens5f0"
```

### Example output

```
6: ens5f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP
  mode DEFAULT group default qlen 1000
  link/ether b4:96:91:a5:22:10 brd ff:ff:ff:ff:ff:ff
    vf 0 link/ether a2:81:ba:d0:6f:f3 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
    vf 1 link/ether 8a:bb:0a:36:f2:ed brd ff:ff:ff:ff:ff:ff, vlan 171, vlan protocol 802.1ad, spoof checking on, link-state auto, trust off
    vf 2 link/ether ca:0e:e1:5b:0c:d2 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
    vf 3 link/ether ee:6c:e2:f5:2c:70 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
    vf 4 link/ether 0a:d6:b7:66:5e:e8 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
    vf 5 link/ether da:d5:e7:14:4f:aa brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
    vf 6 link/ether d6:8e:85:75:12:5c brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
    vf 7 link/ether d6:eb:ce:9c:ea:78 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
    vf 8 link/ether 5e:c5:cc:05:93:3c brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust on
    vf 9 link/ether a6:5a:7c:1c:2a:16 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
```

The **vlan protocol 802.1ad** ID in the output indicates that the interface supports VLAN tagging with protocol 802.1ad (QinQ). The VLAN ID is 171.

## 17.8. USING HIGH PERFORMANCE MULTICAST

You can use multicast on your Single Root I/O Virtualization (SR-IOV) hardware network.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.8.1. High performance multicast

The OVN-Kubernetes network plugin supports multicast between pods on the default network. This is best used for low-bandwidth coordination or service discovery, and not high-bandwidth applications. For applications such as streaming media, like Internet Protocol television (IPTV) and multipoint

videoconferencing, you can utilize Single Root I/O Virtualization (SR-IOV) hardware to provide near-native performance.

When using additional SR-IOV interfaces for multicast:

- Multicast packages must be sent or received by a pod through the additional SR-IOV interface.
- The physical network which connects the SR-IOV interfaces decides the multicast routing and topology, which is not controlled by OpenShift Container Platform.

### 17.8.2. Configuring an SR-IOV interface for multicast

The follow procedure creates an example SR-IOV interface for multicast.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

#### Procedure

1. Create a **SriovNetworkNodePolicy** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ['ens803f0']
    rootDevices: ['0000:86:00.0']
```

2. Create a **SriovNetwork** object:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | ①
  {
    "type": "host-local", ②
    "subnet": "10.56.217.0/24",
    "rangeStart": "10.56.217.171",
    "rangeEnd": "10.56.217.181",
    "routes": [
```

```

        {"dst": "224.0.0.0/5"},  

        {"dst": "232.0.0.0/5"}  

    ],  

    "gateway": "10.56.217.1"  

}  

resourceName: example

```

- 1** **2** If you choose to configure DHCP as IPAM, ensure that you provision the following default routes through your DHCP server: **224.0.0.0/5** and **232.0.0.0/5**. This is to override the static multicast route set by the default network provider.

3. Create a pod with multicast application:

```

apiVersion: v1  

kind: Pod  

metadata:  

  name: testpm  

  namespace: default  

  annotations:  

    k8s.v1.cni.cncf.io/networks: nic1  

spec:  

  containers:  

  - name: example  

    image: rhel7:latest  

    securityContext:  

      capabilities:  

        add: ["NET_ADMIN"] 1  

    command: [ "sleep", "infinity"]

```

- 1** The **NET\_ADMIN** capability is required only if your application needs to assign the multicast IP address to the SR-IOV interface. Otherwise, it can be omitted.

## 17.9. USING DPDK AND RDMA

The containerized Data Plane Development Kit (DPDK) application is supported on OpenShift Container Platform. You can use Single Root I/O Virtualization (SR-IOV) network hardware with the Data Plane Development Kit (DPDK) and with remote direct memory access (RDMA).

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.9.1. Example use of a virtual function in a pod

You can run a remote direct memory access (RDMA) or a Data Plane Development Kit (DPDK) application in a pod with SR-IOV VF attached.

This example shows a pod using a virtual function (VF) in RDMA mode:

#### Pod spec that uses RDMA mode

```

apiVersion: v1  

kind: Pod  

metadata:

```

```

name: rdma-app
annotations:
  k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
    - name: testpmd
      image: <RDMA_image>
      imagePullPolicy: IfNotPresent
      securityContext:
        runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
      command: ["sleep", "infinity"]

```

The following example shows a pod with a VF in DPDK mode:

#### Pod spec that uses DPDK mode

```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
    - name: testpmd
      image: <DPDK_image>
      securityContext:
        runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"]
  volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
  resources:
    limits:
      memory: "1Gi"
      cpu: "2"
      hugepages-1Gi: "4Gi"
    requests:
      memory: "1Gi"
      cpu: "2"
      hugepages-1Gi: "4Gi"
  command: ["sleep", "infinity"]
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

#### 17.9.2. Using a virtual function in DPDK mode with an Intel NIC

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the SR-IOV Network Operator.
- Log in as a user with **cluster-admin** privileges.

## Procedure

1. Create the following **SriovNetworkNodePolicy** object, and then save the YAML in the **intel-dpdk-node-policy.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: vfio-pci ①
```

- 1 Specify the driver type for the virtual functions to **vfio-pci**.



### NOTE

See the **Configuring SR-IOV network devices** section for a detailed explanation on each option in **SriovNetworkNodePolicy**.

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

After the configuration update is applied, all the pods in **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

```
$ oc create -f intel-dpdk-node-policy.yaml
```

3. Create the following **SriovNetwork** object, and then save the YAML in the **intel-dpdk-network.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
```

```

kind: SriosNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |-  

# ... ①
  vlan: <vlan>
  resourceName: intelnics

```

- ① Specify a configuration object for the ipam CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.



### NOTE

See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriosNetwork**.

An optional library, app-netutil, provides several API methods for gathering network information about a container's parent pod.

4. Create the **SriosNetwork** object by running the following command:

```
$ oc create -f intel-dpdk-network.yaml
```

5. Create the following **Pod** spec, and then save the YAML in the **intel-dpdk-pod.yaml** file.

```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> ①
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
    - name: testpmd
      image: <DPDK_image> ②
      securityContext:
        runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] ③
  volumeMounts:
    - mountPath: /mnt/huge ④
      name: hugepage
  resources:
    limits:
      openshift.io/intelnics: "1" ⑤
      memory: "1Gi"
      cpu: "4" ⑥
      hugepages-1Gi: "4Gi" ⑦
    requests:

```

```

openshift.io/intelnics: "1"
memory: "1Gi"
cpu: "4"
hugepages-1Gi: "4Gi"
command: ["sleep", "infinity"]
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages

```

- 1** Specify the same **target\_namespace** where the **SriovNetwork** object **intel-dpdk-network** is created. If you would like to create the pod in a different namespace, change **target\_namespace** in both the **Pod** spec and the **SriovNetwork** object.
- 2** Specify the DPDK image which includes your application and the DPDK library used by application.
- 3** Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.
- 4** Mount a hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.
- 5** Optional: Specify the number of DPDK devices allocated to DPDK pod. This resource request and limit, if not explicitly specified, will be automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by the SR-IOV Operator. It is enabled by default and can be disabled by setting **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.
- 6** Specify the number of CPUs. The DPDK pod usually requires exclusive CPUs to be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and creating a pod with **Guaranteed** QoS.
- 7** Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes. For example, adding kernel arguments **default\_hugepagesz=1GB**, **hugepagesz=1G** and **hugepages=16** will result in **16\*1Gi** hugepages be allocated during system boot.

6. Create the DPDK pod by running the following command:

```
$ oc create -f intel-dpdk-pod.yaml
```

### 17.9.3. Using a virtual function in DPDK mode with a Mellanox NIC

You can create a network node policy and create a Data Plane Development Kit (DPDK) pod using a virtual function in DPDK mode with a Mellanox NIC.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the Single Root I/O Virtualization (SR-IOV) Network Operator.
- You have logged in as a user with **cluster-admin** privileges.

## Procedure

- Save the following **SriovNetworkNodePolicy** YAML configuration to an **mlx-dpdk-node-policy.yaml** file:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" 1
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice 2
  isRdma: true 3
```

- 1** Specify the device hex code of the SR-IOV network device.
- 2** Specify the driver type for the virtual functions to **netdevice**. A Mellanox SR-IOV Virtual Function (VF) can work in DPDK mode without using the **vfio-pci** device type. The VF device appears as a kernel network interface inside a container.
- 3** Enable Remote Direct Memory Access (RDMA) mode. This is required for Mellanox cards to work in DPDK mode.



### NOTE

See [Configuring an SR-IOV network device](#) for a detailed explanation of each option in the **SriovNetworkNodePolicy** object.

When applying the configuration specified in an **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes. It might take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

After the configuration update is applied, all the pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

- Create the **SriovNetworkNodePolicy** object by running the following command:

```
$ oc create -f mlx-dpdk-node-policy.yaml
```

- Save the following **SriovNetwork** YAML configuration to an **mlx-dpdk-network.yaml** file:

```
apiVersion: sriovnetwork.openshift.io/v1
```

```

kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- ①
...
  vlan: <vlan>
  resourceName: mlxnic

```

- Specify a configuration object for the IP Address Management (IPAM) Container Network Interface (CNI) plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.



#### NOTE

See [Configuring an SR-IOV network device](#) for a detailed explanation on each option in the **SriovNetwork** object.

The **app-netutil** option library provides several API methods for gathering network information about the parent pod of a container.

- Create the **SriovNetwork** object by running the following command:

```
$ oc create -f mlx-dpdk-network.yaml
```

- Save the following **Pod** YAML configuration to an **mlx-dpdk-pod.yaml** file:

```

apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> ①
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
    - name: testpm
      image: <DPDK_image> ②
      securityContext:
        runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] ③
      volumeMounts:
        - mountPath: /mnt/huge ④
          name: hugepage
      resources:
        limits:
          openshift.io/mlxnic: "1" ⑤
          memory: "1Gi"
          cpu: "4" ⑥
          hugepages-1Gi: "4Gi" ⑦

```

```

requests:
  openshift.io/mlxnic: "1"
  memory: "1Gi"
  cpu: "4"
  hugepages-1Gi: "4Gi"
  command: ["sleep", "infinity"]
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages

```

- 1 Specify the same **target\_namespace** where **SriovNetwork** object **mlx-dpdk-network** is created. To create the pod in a different namespace, change **target\_namespace** in both the **Pod** spec and **SriovNetwork** object.
- 2 Specify the DPDK image which includes your application and the DPDK library used by the application.
- 3 Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.
- 4 Mount the hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the **emptyDir** volume type with the medium being **Hugepages**.
- 5 Optional: Specify the number of DPDK devices allocated for the DPDK pod. If not explicitly specified, this resource request and limit is automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by SR-IOV Operator. It is enabled by default and can be disabled by setting the **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.
- 6 Specify the number of CPUs. The DPDK pod usually requires that exclusive CPUs be allocated from the kubelet. To do this, set the CPU Manager policy to **static** and create a pod with **Guaranteed** Quality of Service (QoS).
- 7 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepages requires adding kernel arguments to Nodes.

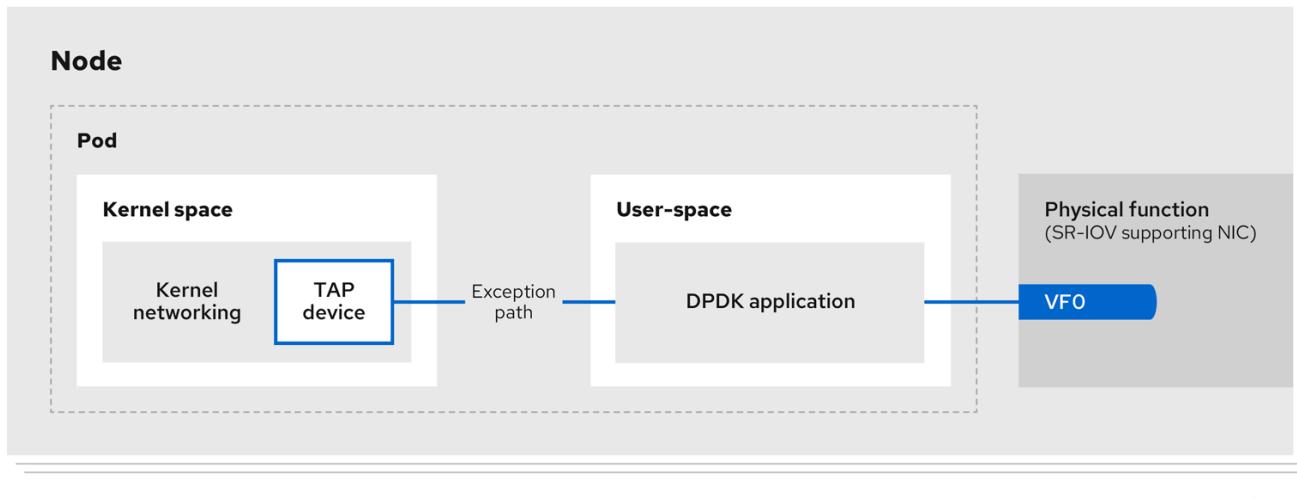
6. Create the DPDK pod by running the following command:

```
$ oc create -f mlx-dpdk-pod.yaml
```

#### 17.9.4. Using the TAP CNI to run a rootless DPDK workload with kernel access

DPDK applications can use **virtio-user** as an exception path to inject certain types of packets, such as log messages, into the kernel for processing. For more information about this feature, see [Virtio\\_user as Exception Path](#).

In OpenShift Container Platform version 4.14 and later, you can use non-privileged pods to run DPDK applications alongside the tap CNI plugin. To enable this functionality, you need to mount the **vhost-net** device by setting the **needVhostNet** parameter to **true** within the **SriovNetworkNodePolicy** object.

**Figure 17.1. DPDK and TAP example configuration**

348\_OpenShift\_0923

## Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the SR-IOV Network Operator.
- You are logged in as a user with **cluster-admin** privileges.
- Ensure that **setsebools container\_use\_devices=on** is set as root on all nodes.



### NOTE

Use the Machine Config Operator to set this SELinux boolean.

## Procedure

1. Create a file, such as **test-namespace.yaml**, with content like the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
```

2. Create the new **Namespace** object by running the following command:

```
$ oc apply -f test-namespace.yaml
```

3. Create a file, such as **sriov-node-network-policy.yaml**, with content like the following example::

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
```

```

metadata:
  name: sriovnic
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice 1
  isRdma: true 2
  needVhostNet: true 3
  nicSelector:
    vendor: "15b3" 4
    deviceID: "101b" 5
    rootDevices: ["00:05.0"]
  numVfs: 10
  priority: 99
  resourceName: sriovnic
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"

```

- 1** This indicates that the profile is tailored specifically for Mellanox Network Interface Controllers (NICs).
- 2** Setting **isRdma** to **true** is only required for a Mellanox NIC.
- 3** This mounts the **/dev/net/tun** and **/dev/vhost-net** devices into the container so the application can create a tap device and connect the tap device to the DPDK workload.
- 4** The vendor hexadecimal code of the SR-IOV network device. The value 15b3 is associated with a Mellanox NIC.
- 5** The device hexadecimal code of the SR-IOV network device.

4. Create the **SriovNetworkNodePolicy** object by running the following command:

```
$ oc create -f sriov-node-network-policy.yaml
```

5. Create the following **SriovNetwork** object, and then save the YAML in the **sriov-network-attachment.yaml** file:

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: test-namespace
  resourceName: sriovnic
  spoofChk: "off"
  trust: "on"

```



#### NOTE

See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriovNetwork**.

An optional library, **app-netutil**, provides several API methods for gathering network information about a container's parent pod.

6. Create the **SriovNetwork** object by running the following command:

```
$ oc create -f sriov-network-attachment.yaml
```

7. Create a file, such as **tap-example.yaml**, that defines a network attachment definition, with content like the following example:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: tap-one
  namespace: test-namespace 1
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "tap",
    "plugins": [
      {
        "type": "tap",
        "multiQueue": true,
        "selinuxcontext": "system_u:system_r:container_t:s0"
      },
      {
        "type": "tuning",
        "capabilities": {
          "mac": true
        }
      }
    ]
}'
```

- 1** Specify the same **target\_namespace** where the **SriovNetwork** object is created.

8. Create the **NetworkAttachmentDefinition** object by running the following command:

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

9. Create a file, such as **dpgk-pod-rootless.yaml**, with content like the following example:

```
apiVersion: v1
kind: Pod
metadata:
  name: dpgk-app
  namespace: test-namespace 1
  annotations:
    k8s.v1.cni.cncf.io/networks: '[{"name": "sriov-network", "namespace": "test-namespace"}, {"name": "tap-one", "interface": "ext0", "namespace": "test-namespace"}]'
spec:
  nodeSelector:
    kubernetes.io/hostname: "worker-0"
```

```

securityContext:
  fsGroup: 1001 ②
  runAsGroup: 1001 ③
  seccompProfile:
    type: RuntimeDefault
containers:
- name: testpmd
  image: <DPDK_image> ④
  securityContext:
    capabilities:
      drop: ["ALL"] ⑤
      add: ⑥
        - IPC_LOCK
        - NET_RAW #for mlx only ⑦
  runAsUser: 1001 ⑧
  privileged: false ⑨
  allowPrivilegeEscalation: true ⑩
  runAsNonRoot: true ⑪
volumeMounts:
- mountPath: /mnt/huge ⑫
  name: hugepages
resources:
limits:
  openshift.io/sriovnic: "1" ⑬
  memory: "1Gi"
  cpu: "4" ⑭
  hugepages-1Gi: "4Gi" ⑮
requests:
  openshift.io/sriovnic: "1"
  memory: "1Gi"
  cpu: "4"
  hugepages-1Gi: "4Gi"
  command: ["sleep", "infinity"]
runtimeClassName: performance-cnf-performanceprofile ⑯
volumes:
- name: hugepages
  emptyDir:
    medium: HugePages

```

- ① Specify the same **target\_namespace** in which the **SriovNetwork** object is created. If you want to create the pod in a different namespace, change **target\_namespace** in both the **Pod** spec and the **SriovNetwork** object.
- ② Sets the group ownership of volume-mounted directories and files created in those volumes.
- ③ Specify the primary group ID used for running the container.
- ④ Specify the DPDK image that contains your application and the DPDK library used by application.
- ⑤ Removing all capabilities (**ALL**) from the container's securityContext means that the container has no special privileges beyond what is necessary for normal operation.

- 6 Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access. These
- 7 Mellanox network interface controller (NIC) requires the **NET\_RAW** capability.
- 8 Specify the user ID used for running the container.
- 9 This setting indicates that the container or containers within the pod should not be granted privileged access to the host system.
- 10 This setting allows a container to escalate its privileges beyond the initial non-root privileges it might have been assigned.
- 11 This setting ensures that the container runs with a non-root user. This helps enforce the principle of least privilege, limiting the potential impact of compromising the container and reducing the attack surface.
- 12 Mount a hugepage volume to the DPDK pod under **/mnt/huge**. The hugepage volume is backed by the emptyDir volume type with the medium being **Hugepages**.
- 13 Optional: Specify the number of DPDK devices allocated for the DPDK pod. If not explicitly specified, this resource request and limit is automatically added by the SR-IOV network resource injector. The SR-IOV network resource injector is an admission controller component managed by SR-IOV Operator. It is enabled by default and can be disabled by setting the **enableInjector** option to **false** in the default **SriovOperatorConfig** CR.
- 14 Specify the number of CPUs. The DPDK pod usually requires exclusive CPUs to be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and creating a pod with **Guaranteed** QoS.
- 15 Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the DPDK pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes. For example, adding kernel arguments **default\_hugepagesz=1GB**, **hugepagesz=1G** and **hugepages=16** will result in **16\*1Gi** hugepages be allocated during system boot.
- 16 If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

10. Create the DPDK pod by running the following command:

```
$ oc create -f dpdk-pod-rootless.yaml
```

## Additional resources

- [Creating a performance profile](#)
- [Configuring an SR-IOV network device](#)

### 17.9.5. Overview of achieving a specific DPDK line rate

To achieve a specific Data Plane Development Kit (DPDK) line rate, deploy a Node Tuning Operator and configure Single Root I/O Virtualization (SR-IOV). You must also tune the DPDK settings for the following resources:

- Isolated CPUs
- Hugepages
- The topology scheduler

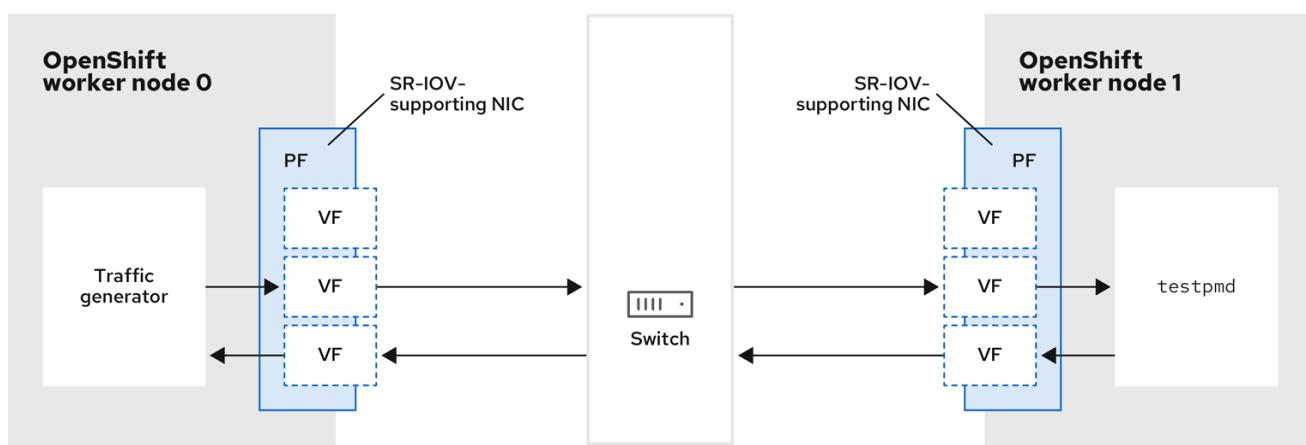


## NOTE

In previous versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift Container Platform applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

## DPDK test environment

The following diagram shows the components of a traffic-testing environment:



261\_OpenShift\_0722

- **Traffic generator:** An application that can generate high-volume packet traffic.
- **SR-IOV-supporting NIC:** A network interface card compatible with SR-IOV. The card runs a number of virtual functions on a physical interface.
- **Physical Function (PF):** A PCI Express (PCIe) function of a network adapter that supports the SR-IOV interface.
- **Virtual Function (VF):** A lightweight PCIe function on a network adapter that supports SR-IOV. The VF is associated with the PCIe PF on the network adapter. The VF represents a virtualized instance of the network adapter.
- **Switch:** A network switch. Nodes can also be connected back-to-back.
- **testpmd:** An example application included with DPDK. The **testpmd** application can be used to test the DPDK in a packet-forwarding mode. The **testpmd** application is also an example of how to build a fully-fledged application using the DPDK Software Development Kit (SDK).
- **worker 0 and worker 1:** OpenShift Container Platform nodes.

### 17.9.6. Using SR-IOV and the Node Tuning Operator to achieve a DPDK line rate

You can use the Node Tuning Operator to configure isolated CPUs, hugepages, and a topology scheduler. You can then use the Node Tuning Operator with Single Root I/O Virtualization (SR-IOV) to achieve a specific Data Plane Development Kit (DPDK) line rate.

## Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the SR-IOV Network Operator.
- You have logged in as a user with **cluster-admin** privileges.
- You have deployed a standalone Node Tuning Operator.



### NOTE

In previous versions of OpenShift Container Platform, the Performance Addon Operator was used to implement automatic tuning to achieve low latency performance for OpenShift applications. In OpenShift Container Platform 4.11 and later, this functionality is part of the Node Tuning Operator.

## Procedure

- 1 Create a **PerformanceProfile** object based on the following example:

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  globallyDisableIrqLoadBalancing: true
  cpu:
    isolated: 21-51,73-103 ①
    reserved: 0-20,52-72 ②
  hugepages:
    defaultHugepagesSize: 1G ③
    pages:
      - count: 32
        size: 1G
  net:
    userLevelNetworking: true
  numa:
    topologyPolicy: "single-numa-node"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
```

- 1 If hyperthreading is enabled on the system, allocate the relevant symbolic links to the **isolated** and **reserved** CPU groups. If the system contains multiple non-uniform memory access nodes (NUMAs), allocate CPUs from both NUMAs to both groups. You can also use the Performance Profile Creator for this task. For more information, see *Creating a performance profile*.
- 2 You can also specify a list of devices that will have their queues set to the reserved CPU count. For more information, see *Reducing NIC queues using the Node Tuning Operator*.

- 3** Allocate the number and size of hugepages needed. You can specify the NUMA configuration for the hugepages. By default, the system allocates an even number to every

2. Save the **yaml** file as **mlx-dpdk-perfprofile-policy.yaml**.
3. Apply the performance profile using the following command:

```
$ oc create -f mlx-dpdk-perfprofile-policy.yaml
```

### 17.9.6.1. DPDK library for use with container applications

An [optional library](#), **app-netutil**, provides several API methods for gathering network information about a pod from within a container running within that pod.

This library can assist with integrating SR-IOV virtual functions (VFs) in Data Plane Development Kit (DPDK) mode into the container. The library provides both a Golang API and a C API.

Currently there are three API methods implemented:

#### **GetCPUInfo()**

This function determines which CPUs are available to the container and returns the list.

#### **GetHugepages()**

This function determines the amount of huge page memory requested in the **Pod** spec for each container and returns the values.

#### **GetInterfaces()**

This function determines the set of interfaces in the container and returns the list. The return value includes the interface type and type-specific data for each interface.

The repository for the library includes a sample Dockerfile to build a container image, **dpdk-app-centos**. The container image can run one of the following DPDK sample applications, depending on an environment variable in the pod specification: **I2fwd**, **I3wd** or **testpmd**. The container image provides an example of integrating the **app-netutil** library into the container image itself. The library can also integrate into an init container. The init container can collect the required data and pass the data to an existing DPDK workload.

### 17.9.6.2. Example SR-IOV Network Operator for virtual functions

You can use the Single Root I/O Virtualization (SR-IOV) Network Operator to allocate and configure Virtual Functions (VFs) from SR-IOV-supporting Physical Function NICs on the nodes.

For more information on deploying the Operator, see *Installing the SR-IOV Network Operator*. For more information on configuring an SR-IOV network device, see *Configuring an SR-IOV network device*.

There are some differences between running Data Plane Development Kit (DPDK) workloads on Intel VFs and Mellanox VFs. This section provides object configuration examples for both VF types. The following is an example of an **sriovNetworkNodePolicy** object used to run DPDK applications on Intel NICs:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
```

```

spec:
  deviceType: vfio-pci ①
  needVhostNet: true ②
  nicSelector:
    pfNames: ["ens3f0"]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  needVhostNet: true
  nicSelector:
    pfNames: ["ens3f1"]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_2

```

① For Intel NICs, **deviceType** must be **vfio-pci**.

② If kernel communication with DPDK workloads is required, add **needVhostNet: true**. This mounts the **/dev/net/tun** and **/dev/vhost-net** devices into the container so the application can create a tap device and connect the tap device to the DPDK workload.

The following is an example of an **sriovNetworkNodePolicy** object for Mellanox NICs:

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice ①
  isRdma: true ②
  nicSelector:
    rootDevices:
      - "0000:5e:00.1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:

```

```

name: dpdk-nic-2
namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    rootDevices:
      - "0000:5e:00.0"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_2

```

- 1 For Mellanox devices the **deviceType** must be **netdevice**.
- 2 For Mellanox devices **isRdma** must be **true**. Mellanox cards are connected to DPDK applications using Flow Bifurcation. This mechanism splits traffic between Linux user space and kernel space, and can enhance line rate processing capability.

#### 17.9.6.3. Example SR-IOV network operator

The following is an example definition of an **sriovNetwork** object. In this case, Intel and Mellanox configurations are identical:

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-1
  namespace: openshift-sriov-network-operator
spec:
  ipam: {"type": "host-local", "ranges": [{"subnet": "10.0.1.0/24"}], "dataDir": "/run/my-orchestrator/container-ipam-state-1"} 1
  networkNamespace: dpdk-test 2
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_1 3
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-2
  namespace: openshift-sriov-network-operator
spec:
  ipam: {"type": "host-local", "ranges": [{"subnet": "10.0.2.0/24"}], "dataDir": "/run/my-orchestrator/container-ipam-state-1"}
  networkNamespace: dpdk-test
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_2

```

- 1 You can use a different IP Address Management (IPAM) implementation, such as Whereabouts. For more information, see *Dynamic IP address assignment configuration with Whereabouts* .

- 2 You must request the **networkNamespace** where the network attachment definition will be created. You must create the **sriovNetwork** CR under the **openshift-sriov-network-operator**
- 3 The **resourceName** value must match that of the **resourceName** created under the **sriovNetworkNodePolicy**.

#### 17.9.6.4. Example DPDK base workload

The following is an example of a Data Plane Development Kit (DPDK) container:

```

apiVersion: v1
kind: Namespace
metadata:
  name: dpdk-test
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[ ①
      {
        "name": "dpdk-network-1",
        "namespace": "dpdk-test"
      },
      {
        "name": "dpdk-network-2",
        "namespace": "dpdk-test"
      }
    ]'
    irq-load-balancing.crio.io: "disable" ②
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
  labels:
    app: dpdk
    name: testpmd
    namespace: dpdk-test
spec:
  runtimeClassName: performance-performance ③
  containers:
    - command:
        - /bin/bash
        - -c
        - sleep INF
      image: registry.redhat.io/openshift4/dpdk-base-rhel8
      imagePullPolicy: Always
      name: dpdk
      resources: ④
        limits:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 2Gi
        requests:
          cpu: "16"
          hugepages-1Gi: 8Gi

```

```

memory: 2Gi
securityContext:
capabilities:
  add:
    - IPC_LOCK
    - SYS_RESOURCE
    - NET_RAW
    - NET_ADMIN
  runAsUser: 0
volumeMounts:
  - mountPath: /mnt/huge
    name: hugepages
terminationGracePeriodSeconds: 5
volumes:
  - emptyDir:
    medium: HugePages
    name: hugepages

```

- 1 Request the SR-IOV networks you need. Resources for the devices will be injected automatically.
- 2 Disable the CPU and IRQ load balancing base. See *Disabling interrupt processing for individual pods* for more information.
- 3 Set the **runtimeClass** to **performance-performance**. Do not set the **runtimeClass** to **HostNetwork** or **privileged**.
- 4 Request an equal number of resources for requests and limits to start the pod with **Guaranteed** Quality of Service (QoS).



#### NOTE

Do not start the pod with **SLEEP** and then exec into the pod to start the testpmd or the DPDK workload. This can add additional interrupts as the **exec** process is not pinned to any CPU.

#### 17.9.6.5. Example testpmd script

The following is an example script for running **testpmd**:

```

#!/bin/bash
set -ex
export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
echo ${CPU}

dpdk-testpmd -I ${CPU} -a ${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_1} -a
${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_2} -n 4 -- -i --nb-cores=15 --rxq=4096 --txd=4096 --
rxq=7 --txq=7 --forward-mode=mac --eth-peer=0,50:00:00:00:01 --eth-peer=1,50:00:00:00:02

```

This example uses two different **sriovNetwork** CRs. The environment variable contains the Virtual Function (VF) PCI address that was allocated for the pod. If you use the same network in the pod definition, you must split the **pciAddress**. It is important to configure the correct MAC addresses of the traffic generator. This example uses custom MAC addresses.

#### 17.9.7. Using a virtual function in RDMA mode with a Mellanox NIC



## IMPORTANT

RDMA over Converged Ethernet (RoCE) 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](#).

RDMA over Converged Ethernet (RoCE) is the only supported mode when using RDMA on OpenShift Container Platform.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the SR-IOV Network Operator.
- Log in as a user with **cluster-admin** privileges.

### Procedure

- 1 Create the following **SriovNetworkNodePolicy** object, and then save the YAML in the **mlx-rdma-node-policy.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" ①
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice ②
  isRdma: true ③
```

- ① Specify the device hex code of the SR-IOV network device.
- ② Specify the driver type for the virtual functions to **netdevice**.
- ③ Enable RDMA mode.



## NOTE

See the **Configuring SR-IOV network devices** section for a detailed explanation on each option in **SriovNetworkNodePolicy**.

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator may drain the nodes, and in some cases, reboot nodes. It may take several minutes for a configuration change to apply. Ensure that there are enough available nodes in your cluster to handle the evicted workload beforehand.

After the configuration update is applied, all the pods in the **openshift-sriov-network-operator** namespace will change to a **Running** status.

2. Create the **SriovNetworkNodePolicy** object by running the following command:

```
$ oc create -f mlx-rdma-node-policy.yaml
```

3. Create the following **SriovNetwork** object, and then save the YAML in the **mlx-rdma-network.yaml** file.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-rdma-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- ①
# ...
  vlan: <vlan>
  resourceName: mlxnic
```

- 1 Specify a configuration object for the ipam CNI plugin as a YAML block scalar. The plugin manages IP address assignment for the attachment definition.



## NOTE

See the "Configuring SR-IOV additional network" section for a detailed explanation on each option in **SriovNetwork**.

An optional library, app-netutil, provides several API methods for gathering network information about a container's parent pod.

4. Create the **SriovNetworkNodePolicy** object by running the following command:

```
$ oc create -f mlx-rdma-network.yaml
```

5. Create the following **Pod** spec, and then save the YAML in the **mlx-rdma-pod.yaml** file.

```
apiVersion: v1
kind: Pod
metadata:
```

```

name: rdma-app
namespace: <target_namespace> ①
annotations:
  k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
    - name: testpmd
      image: <RDMA_image> ②
      securityContext:
        runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] ③
      volumeMounts:
        - mountPath: /mnt/huge ④
          name: hugepage
      resources:
        limits:
          memory: "1Gi"
          cpu: "4" ⑤
        hugepages-1Gi: "4Gi" ⑥
      requests:
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
      command: ["sleep", "infinity"]
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

- ① Specify the same **target\_namespace** where **SriovNetwork** object **mlx-rdma-network** is created. If you would like to create the pod in a different namespace, change **target\_namespace** in both **Pod** spec and **SriovNetwork** object.
- ② Specify the RDMA image which includes your application and RDMA library used by application.
- ③ Specify additional capabilities required by the application inside the container for hugepage allocation, system resource allocation, and network interface access.
- ④ Mount the hugepage volume to RDMA pod under **/mnt/huge**. The hugepage volume is backed by the **emptyDir** volume type with the medium being **Hugepages**.
- ⑤ Specify number of CPUs. The RDMA pod usually requires exclusive CPUs be allocated from the kubelet. This is achieved by setting CPU Manager policy to **static** and create pod with **Guaranteed** QoS.
- ⑥ Specify hugepage size **hugepages-1Gi** or **hugepages-2Mi** and the quantity of hugepages that will be allocated to the RDMA pod. Configure **2Mi** and **1Gi** hugepages separately. Configuring **1Gi** hugepage requires adding kernel arguments to Nodes.

6. Create the RDMA pod by running the following command:

```
$ oc create -f mlx-rdma-pod.yaml
```

### 17.9.8. A test pod template for clusters that use OVS-DPDK on OpenStack

The following **testpmd** pod demonstrates container creation with huge pages, reserved CPUs, and the SR-IOV port.

#### An example testpmd pod

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-dpdk
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
  # ...
spec:
  containers:
    - name: testpmd
      command: ["sleep", "99999"]
      image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
      securityContext:
        capabilities:
          add: ["IPC_LOCK","SYS_ADMIN"]
        privileged: true
        runAsUser: 0
      resources:
        requests:
          memory: 1000Mi
          hugepages-1Gi: 1Gi
          cpu: '2'
          openshift.io/dpdk1: 1 ①
        limits:
          hugepages-1Gi: 1Gi
          cpu: '2'
          memory: 1000Mi
          openshift.io/dpdk1: 1
      volumeMounts:
        - mountPath: /mnt/huge
          name: hugepage
          readOnly: False
    runtimeClassName: performance-cnf-performanceprofile ②
  volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages
```

- ① The name **dpdk1** in this example is a user-created **SriovNetworkNodePolicy** resource. You can substitute this name for that of a resource that you create.
- ② If your performance profile is not named **cnf-performance profile**, replace that string with the correct performance profile name.

### 17.9.9. Additional resources

- Supported devices
- Creating a performance profile
- Adjusting the NIC queues with the performance profile
- Provisioning real-time and low latency workloads
- Installing the SR-IOV Network Operator
- Configuring an SR-IOV network device
- Dynamic IP address assignment configuration with Whereabouts
- Disabling interrupt processing for individual pods
- Configuring an SR-IOV Ethernet network attachment

## 17.10. USING POD-LEVEL BONDING

Bonding at the pod level is vital to enable workloads inside pods that require high availability and more throughput. With pod-level bonding, you can create a bond interface from multiple single root I/O virtualization (SR-IOV) virtual function interfaces in a kernel mode interface. The SR-IOV virtual functions are passed into the pod and attached to a kernel driver.

One scenario where pod level bonding is required is creating a bond interface from multiple SR-IOV virtual functions on different physical functions. Creating a bond interface from two different physical functions on the host can be used to achieve high availability and throughput at pod level.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

For guidance on tasks such as creating a SR-IOV network, network policies, network attachment definitions and pods, see [Configuring an SR-IOV network device](#).

### 17.10.1. Configuring a bond interface from two SR-IOV interfaces

Bonding enables multiple network interfaces to be aggregated into a single logical "bonded" interface. Bond Container Network Interface (Bond-CNI) brings bond capability into containers.

Bond-CNI can be created using Single Root I/O Virtualization (SR-IOV) virtual functions and placing them in the container network namespace.

OpenShift Container Platform only supports Bond-CNI using SR-IOV virtual functions. The SR-IOV Network Operator provides the SR-IOV CNI plugin needed to manage the virtual functions. Other CNIs or types of interfaces are not supported.

#### Prerequisites

- The SR-IOV Network Operator must be installed and configured to obtain virtual functions in a container.
- To configure SR-IOV interfaces, an SR-IOV network and policy must be created for each interface.

- The SR-IOV Network Operator creates a network attachment definition for each SR-IOV interface, based on the SR-IOV network and policy defined.
- The **linkState** is set to the default value **auto** for the SR-IOV virtual function.

### 17.10.1.1. Creating a bond network attachment definition

Now that the SR-IOV virtual functions are available, you can create a bond network attachment definition.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-net1
  namespace: demo
spec:
  config: '{
    "type": "bond", 1
    "cniVersion": "0.3.1",
    "name": "bond-net1",
    "mode": "active-backup", 2
    "failOverMac": 1, 3
    "linksInContainer": true, 4
    "miimon": 100,
    "mtu": 1500,
    "links": [ 5
      {"name": "net1"},
      {"name": "net2"}
    ],
    "ipam": {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "routes": [
        {
          "dst": "0.0.0.0/0"
        }
      ],
      "gateway": "10.56.217.1"
    }
  }'
```

- 1** The cni-type is always set to **bond**.
- 2** The **mode** attribute specifies the bonding mode.



#### NOTE

The bonding modes supported are:

- **balance-rr** - 0
- **active-backup** - 1
- **balance-xor** - 2

For **balance-rr** or **balance-xor** modes, you must set the **trust** mode to **on** for the SR-IOV virtual function.

- 3 The **failover** attribute is mandatory for active-backup mode and must be set to 1.
- 4 The **linksInContainer=true** flag informs the Bond CNI that the required interfaces are to be found inside the container. By default, Bond CNI looks for these interfaces on the host which does not work for integration with SRIOV and Multus.
- 5 The **links** section defines which interfaces will be used to create the bond. By default, Multus names the attached interfaces as: "net", plus a consecutive number, starting with one.

### 17.10.1.2. Creating a pod using a bond interface

- 1 Test the setup by creating a pod with a YAML file named for example **podbonding.yaml** with content similar to the following:

```
apiVersion: v1
kind: Pod
metadata:
  name: bondpod1
  namespace: demo
  annotations:
    k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1 ①
spec:
  containers:
    - name: podexample
      image: quay.io/openshift/origin-network-interface-bond-cni:4.11.0
      command: ["/bin/bash", "-c", "sleep INF"]
```

- 1 Note the network annotation: it contains two SR-IOV network attachments, and one bond network attachment. The bond attachment uses the two SR-IOV interfaces as bonded port interfaces.

- 2 Apply the yaml by running the following command:

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

- 3 Inspect the pod interfaces with the following command:

```
$ oc rsh -n demo bondpod1
sh-4.4#
sh-4.4# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
  valid_lft forever preferred_lft forever
3: eth0@if150: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue state UP
  link/ether 62:b1:b5:c8:fb:7a brd ff:ff:ff:ff:ff:ff
  inet 10.244.1.122/24 brd 10.244.1.255 scope global eth0
    valid_lft forever preferred_lft forever
4: net3: <BROADCAST,MULTICAST,UP,LOWER_UP400> mtu 1500 qdisc noqueue state UP qlen 1000
  link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff ①
  inet 10.56.217.66/24 scope global bond0
```

```

valid_lft forever preferred_lft forever
43: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master
    bond0 state UP qlen 1000
    link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff ②
44: net2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master
    bond0 state UP qlen 1000
    link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff ③

```

- ① The bond interface is automatically named **net3**. To set a specific interface name add **@name** suffix to the pod's **k8s.v1.cni.cncf.io/networks** annotation.
- ② The **net1** interface is based on an SR-IOV virtual function.
- ③ The **net2** interface is based on an SR-IOV virtual function.



#### NOTE

If no interface names are configured in the pod annotation, interface names are assigned automatically as **net<n>**, with **<n>** starting at 1.

4. Optional: If you want to set a specific interface name for example **bond0**, edit the **k8s.v1.cni.cncf.io/networks** annotation and set **bond0** as the interface name as follows:

annotations:

```
k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1@bond0
```

## 17.11. CONFIGURING HARDWARE OFFLOADING

As a cluster administrator, you can configure hardware offloading on compatible nodes to increase data processing performance and reduce load on host CPUs.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.11.1. About hardware offloading

Open vSwitch hardware offloading is a method of processing network tasks by diverting them away from the CPU and offloading them to a dedicated processor on a network interface controller. As a result, clusters can benefit from faster data transfer speeds, reduced CPU workloads, and lower computing costs.

The key element for this feature is a modern class of network interface controllers known as SmartNICs. A SmartNIC is a network interface controller that is able to handle computationally-heavy network processing tasks. In the same way that a dedicated graphics card can improve graphics performance, a SmartNIC can improve network performance. In each case, a dedicated processor improves performance for a specific type of processing task.

In OpenShift Container Platform, you can configure hardware offloading for bare metal nodes that have a compatible SmartNIC. Hardware offloading is configured and enabled by the SR-IOV Network Operator.

Hardware offloading is not compatible with all workloads or application types. Only the following two communication types are supported:

- pod-to-pod
- pod-to-service, where the service is a ClusterIP service backed by a regular pod

In all cases, hardware offloading takes place only when those pods and services are assigned to nodes that have a compatible SmartNIC. Suppose, for example, that a pod on a node with hardware offloading tries to communicate with a service on a regular node. On the regular node, all the processing takes place in the kernel, so the overall performance of the pod-to-service communication is limited to the maximum performance of that regular node. Hardware offloading is not compatible with DPDK applications.

Enabling hardware offloading on a node, but not configuring pods to use it, can result in decreased throughput performance for pod traffic. You cannot configure hardware offloading for pods that are managed by OpenShift Container Platform.

### 17.11.2. Supported devices

Hardware offloading is supported on the following network interface controllers:

**Table 17.15. Supported network interface controllers**

Manufacturer	Model	Vendor ID	Device ID
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f
Mellanox	MT42822 BlueField-2 in ConnectX-6 NIC mode	15b3	a2d6

### 17.11.3. Prerequisites

- Your cluster has at least one bare metal machine with a network interface controller that is supported for hardware offloading.
- You [installed the SR-IOV Network Operator](#).
- Your cluster uses the [OVN-Kubernetes network plugin](#).
- In your [OVN-Kubernetes network plugin configuration](#), the **gatewayConfig.routingViaHost** field is set to **false**.

### 17.11.4. Setting the SR-IOV Network Operator into `systemd` mode

To support hardware offloading, you must first set the SR-IOV Network Operator into **systemd** mode.

#### Prerequisites

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

- You have access to the cluster as a user that has the **cluster-admin** role.

## Procedure

- 1 Create a **SriovOperatorConfig** custom resource (CR) to deploy all the SR-IOV Operator components:

- a. Create a file named **sriovOperatorConfig.yaml** that contains the following YAML:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default 1
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: true
  enableOperatorWebhook: true
  configurationMode: "systemd" 2
  logLevel: 2
```

- 1** The only valid name for the **SriovOperatorConfig** resource is **default** and it must be in the namespace where the Operator is deployed.
- 2** Setting the SR-IOV Network Operator into **systemd** mode is only relevant for Open vSwitch hardware offloading.

- b. Create the resource by running the following command:

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

### 17.11.5. Configuring a machine config pool for hardware offloading

To enable hardware offloading, you now create a dedicated machine config pool and configure it to work with the SR-IOV Network Operator.

## Prerequisites

- 1 SR-IOV Network Operator installed and set into **systemd** mode.

## Procedure

- 1 Create a machine config pool for machines you want to use hardware offloading on.
  - a. Create a file, such as **mcp-offloading.yaml**, with content like the following example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: mcp-offloading 1
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,mcp-]
```

```
    offloading]} 2
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/mcp-offloading: "" 3
```

**1** **2** The name of your machine config pool for hardware offloading.

**3** This node role label is used to add nodes to the machine config pool.

- Apply the configuration for the machine config pool:

```
$ oc create -f mcp-offloading.yaml
```

- Add nodes to the machine config pool. Label each node with the node role label of your pool:

```
$ oc label node worker-2 node-role.kubernetes.io/mcp-offloading=""
```

- Optional: To verify that the new pool is created, run the following command:

```
$ oc get nodes
```

### Example output

NAME	STATUS	ROLES	AGE	VERSION
master-0	Ready	master	2d	v1.31.3
master-1	Ready	master	2d	v1.31.3
master-2	Ready	master	2d	v1.31.3
worker-0	Ready	worker	2d	v1.31.3
worker-1	Ready	worker	2d	v1.31.3
worker-2	Ready	mcp-offloading,worker	47h	v1.31.3
worker-3	Ready	mcp-offloading,worker	47h	v1.31.3

- Add this machine config pool to the **SriovNetworkPoolConfig** custom resource:

- Create a file, such as **sriov-pool-config.yaml**, with content like the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading 1
```

**1** The name of your machine config pool for hardware offloading.

- Apply the configuration:

```
$ oc create -f <SriovNetworkPoolConfig_name>.yaml
```

**NOTE**

When you apply the configuration specified in a **SriovNetworkPoolConfig** object, the SR-IOV Operator drains and restarts the nodes in the machine config pool.

It might take several minutes for a configuration changes to apply.

### 17.11.6. Configuring the SR-IOV network node policy

You can create an SR-IOV network device configuration for a node by creating an SR-IOV network node policy. To enable hardware offloading, you must define the `.spec.eSwitchMode` field with the value `"switchdev"`.

The following procedure creates an SR-IOV interface for a network interface controller with hardware offloading.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

#### Procedure

- 1 Create a file, such as **sriov-node-policy.yaml**, with content like the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy ①
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice ②
  eSwitchMode: "switchdev" ③
  nicSelector:
    deviceID: "1019"
    rootDevices:
      - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
      - ens8f0
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 6
  priority: 5
  resourceName: mlxnic
```

- 1 The name for the custom resource object.
- 2 Required. Hardware offloading is not supported with **vfio-pci**.
- 3 Required.

2. Apply the configuration for the policy:

```
$ oc create -f sriov-node-policy.yaml
```



#### NOTE

When you apply the configuration specified in a **SriovNetworkPoolConfig** object, the SR-IOV Operator drains and restarts the nodes in the machine config pool.

It might take several minutes for a configuration change to apply.

### 17.11.6.1. An example SR-IOV network node policy for OpenStack

The following example describes an SR-IOV interface for a network interface controller (NIC) with hardware offloading on Red Hat OpenStack Platform (RHOSP).

#### An SR-IOV interface for a NIC with hardware offloading on RHOSP

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ${name}
  namespace: openshift-sriov-network-operator
spec:
  deviceType: switchdev
  isRdma: true
  nicSelector:
    netFilter: openstack/NetworkID:${net_id}
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: ${name}
```

### 17.11.7. Improving network traffic performance using a virtual function

Follow this procedure to assign a virtual function to the OVN-Kubernetes management port and increase its network traffic performance.

This procedure results in the creation of two pools: the first has a virtual function used by OVN-Kubernetes, and the second comprises the remaining virtual functions.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

#### Procedure

1. Add the **network.operator.openshift.io/smart-nic** label to each worker node with a SmartNIC present by running the following command:

```
$ oc label node <node-name> network.operator.openshift.io/smart-nic=
```

Use the **oc get nodes** command to get a list of the available nodes.

2. Create a policy named **sriov-node-mgmt-vf-policy.yaml** for the management port with content such as the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-mgmt-vf-policy
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  eSwitchMode: "switchdev"
  nicSelector:
    devicID: "1019"
    rootDevices:
      - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
      - ens8f0#0-0 ①
  nodeSelector:
    network.operator.openshift.io/smart-nic: ""
  numVfs: 6 ②
  priority: 5
  resourceName: mgmtvf
```

- ① Replace this device with the appropriate network device for your use case. The #0-0 part of the **pfNames** value reserves a single virtual function used by OVN-Kubernetes.
- ② The value provided here is an example. Replace this value with one that meets your requirements. For more information, see *SR-IOV network node configuration object* in the *Additional resources* section.

3. Create a policy named **sriov-node-policy.yaml** with content such as the following example:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  eSwitchMode: "switchdev"
  nicSelector:
    devicID: "1019"
    rootDevices:
      - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
      - ens8f0#1-5 ①
  nodeSelector:
    network.operator.openshift.io/smart-nic: ""
```

```
numVfs: 6 2
priority: 5
resourceName: mlxnic
```

- 1** Replace this device with the appropriate network device for your use case.
- 2** The value provided here is an example. Replace this value with the value specified in the **sriov-node-mgmt-vf-policy.yaml** file. For more information, see *SR-IOV network node configuration object* in the *Additional resources* section.

**NOTE**

The **sriov-node-mgmt-vf-policy.yaml** file has different values for the **pfNames** and **resourceName** keys than the **sriov-node-policy.yaml** file.

4. Apply the configuration for both policies:

```
$ oc create -f sriov-node-policy.yaml
$ oc create -f sriov-node-mgmt-vf-policy.yaml
```

5. Create a Cluster Network Operator (CNO) ConfigMap in the cluster for the management configuration:

- a. Create a ConfigMap named **hardware-offload-config.yaml** with the following contents:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: hardware-offload-config
  namespace: openshift-network-operator
data:
  mgmt-port-resource-name: openshift.io/mgmtvf
```

- b. Apply the configuration for the ConfigMap:

```
$ oc create -f hardware-offload-config.yaml
```

**Additional resources**

- [SR-IOV network node configuration object](#)

**17.11.8. Creating a network attachment definition**

After you define the machine config pool and the SR-IOV network node policy, you can create a network attachment definition for the network interface card you specified.

**Prerequisites**

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

## Procedure

1. Create a file, such as **net-attach-def.yaml**, with content like the following example:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net-attach-def ①
  namespace: net-attach-def ②
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/mlxnic ③
spec:
  config: '[{"cniVersion":"0.3.1","name":"ovn-kubernetes","type":"ovn-k8s-cni-overlay","ipam":{},"dns":{}}'
```

- ① The name for your network attachment definition.
- ② The namespace for your network attachment definition.
- ③ This is the value of the **spec.resourceName** field you specified in the **SriovNetworkNodePolicy** object.

2. Apply the configuration for the network attachment definition:

```
$ oc create -f net-attach-def.yaml
```

## Verification

- Run the following command to see whether the new definition is present:

```
$ oc get net-attach-def -A
```

### Example output

NAMESPACE	NAME	AGE
net-attach-def	net-attach-def	43h

## 17.11.9. Adding the network attachment definition to your pods

After you create the machine config pool, the **SriovNetworkPoolConfig** and **SriovNetworkNodePolicy** custom resources, and the network attachment definition, you can apply these configurations to your pods by adding the network attachment definition to your pod specifications.

## Procedure

- In the pod specification, add the **.metadata.annotations.k8s.v1.cni.cncf.io/networks** field and specify the network attachment definition you created for hardware offloading:

```
....  
  metadata:  
    annotations:  
      v1.multus-cni.io/default-network: net-attach-def/net-attach-def ①
```

- 1 The value must be the name and namespace of the network attachment definition you created for hardware offloading.

## 17.12. SWITCHING BLUEFIELD-2 FROM DPU TO NIC

You can switch the Bluefield-2 network device from data processing unit (DPU) mode to network interface controller (NIC) mode.

Before you perform any tasks in the following documentation, ensure that you [installed the SR-IOV Network Operator](#).

### 17.12.1. Switching Bluefield-2 from DPU mode to NIC mode

Use the following procedure to switch Bluefield-2 from data processing units (DPU) mode to network interface controller (NIC) mode.



#### IMPORTANT

Currently, only switching Bluefield-2 from DPU to NIC mode is supported. Switching from NIC mode to DPU mode is unsupported.

#### Prerequisites

- You have installed the SR-IOV Network Operator. For more information, see "Installing SR-IOV Network Operator".
- You have updated Bluefield-2 to the latest firmware. For more information, see [Firmware for NVIDIA BlueField-2](#).

#### Procedure

1. Add the following labels to each of your worker nodes by entering the following commands:

```
$ oc label node <example_node_name_one> node-role.kubernetes.io/sriov=
```

```
$ oc label node <example_node_name_two> node-role.kubernetes.io/sriov=
```

2. Create a machine config pool for the SR-IOV Network Operator, for example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: sriov
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,sriov]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/sriov: ""
```

3. Apply the following **machineconfig.yaml** file to the worker nodes:

```

apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: sriov
  name: 99-bf2-dpu
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-
8;base64,ZmluZF9jb250YWluZXIoKSB7CiAgY3JpY3RsIHBzIC1vIGpb24gfCBqcSATciAnLmNv
bnRhaW5lcnNbXSB8IHNIbGVjdCgubWV0YWRhdGEubmFtZT09InNyaW92LW5ldHdvcnstY29
uZmlnLWRhZW1vbilpIHwgLmlkJwp9CnVudGlsIG91dHB1dD0kKGZpbmRfY29udGFpbmVyKT
sgW1sgLW4gliRvdXRwdXQiF1dOyBkbwogIGVjaG8gIndhaXRpbmcgZm9yIGNvbnRhaW5lciB
0byBjB21IIHVwlgojIHNsZVVwIDE7CmRvbmUKISBzdWRvIGNyaWN0bCBleGVjICRvdXRwdX
QgL2JpbmRhdGEvc2NyaXB0cy9iZjltc3dpdGNolW1vZGUuc2ggliRAlg0=
          mode: 0755
          overwrite: true
          path: /etc/default/switch_in_sriov_config_daemon.sh
  systemd:
    units:
      - name: dpu-switch.service
        enabled: true
        contents: |
          [Unit]
          Description=Switch BlueField2 card to NIC/DPU mode
          RequiresMountsFor=%t/containers
          Wants=network.target
          After=network-online.target kubelet.service
          [Service]
          SuccessExitStatus=0 120
          RemainAfterExit=True
          ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic || shutdown
-r now' ①
          Type=oneshot
          [Install]
          WantedBy=multi-user.target

```

- ① Optional: The PCI address of a specific card can optionally be specified, for example **ExecStart=/bin/bash -c '/etc/default/switch\_in\_sriov\_config\_daemon.sh nic 0000:5e:00.0 || echo done'**. By default, the first device is selected. If there is more than one device, you must specify which PCI address to be used. The PCI address must be the same on all nodes that are switching Bluefield-2 from DPU mode to NIC mode.

4. Wait for the worker nodes to restart. After restarting, the Bluefield-2 network device on the worker nodes is switched into NIC mode.
5. Optional: You might need to restart the host hardware because most recent Bluefield-2 firmware releases require a hardware restart to switch into NIC mode.

## Additional resources

- [Installing SR-IOV Network Operator](#)

# CHAPTER 18. OVN-KUBERNETES NETWORK PLUGIN

## 18.1. ABOUT THE OVN-KUBERNETES NETWORK PLUGIN

The OpenShift Container Platform cluster uses a virtualized network for pod and service networks.

Part of Red Hat OpenShift Networking, the OVN-Kubernetes network plugin is the default network provider for OpenShift Container Platform. OVN-Kubernetes is based on Open Virtual Network (OVN) and provides an overlay-based networking implementation. A cluster that uses the OVN-Kubernetes plugin also runs Open vSwitch (OVS) on each node. OVN configures OVS on each node to implement the declared network configuration.



### NOTE

OVN-Kubernetes is the default networking solution for OpenShift Container Platform and single-node OpenShift deployments.

OVN-Kubernetes, which arose from the OVS project, uses many of the same constructs, such as open flow rules, to determine how packets travel through the network. For more information, see the [Open Virtual Network website](#).

OVN-Kubernetes is a series of daemons for OVS that translate virtual network configurations into **OpenFlow** rules. **OpenFlow** is a protocol for communicating with network switches and routers, providing a means for remotely controlling the flow of network traffic on a network device so that network administrators can configure, manage, and monitor the flow of network traffic.

OVN-Kubernetes provides more of the advanced functionality not available with **OpenFlow**. OVN supports distributed virtual routing, distributed logical switches, access control, Dynamic Host Configuration Protocol (DHCP), and DNS. OVN implements distributed virtual routing within logic flows that equate to open flows. For example, if you have a pod that sends out a DHCP request to the DHCP server on the network, a logic flow rule in the request helps the OVN-Kubernetes handle the packet so that the server can respond with gateway, DNS server, IP address, and other information.

OVN-Kubernetes runs a daemon on each node. There are daemon sets for the databases and for the OVN controller that run on every node. The OVN controller programs the Open vSwitch daemon on the nodes to support the network provider features: egress IPs, firewalls, routers, hybrid networking, IPSEC encryption, IPv6, network policy, network policy logs, hardware offloading, and multicast.

### 18.1.1. OVN-Kubernetes purpose

The OVN-Kubernetes network plugin is an open-source, fully-featured Kubernetes CNI plugin that uses Open Virtual Network (OVN) to manage network traffic flows. OVN is a community developed, vendor-agnostic network virtualization solution. The OVN-Kubernetes network plugin uses the following technologies:

- OVN to manage network traffic flows.
- Kubernetes network policy support and logs, including ingress and egress rules.
- The Generic Network Virtualization Encapsulation (Geneve) protocol, rather than Virtual Extensible LAN (VXLAN), to create an overlay network between nodes.

The OVN-Kubernetes network plugin supports the following capabilities:

- Hybrid clusters that can run both Linux and Microsoft Windows workloads. This environment is known as *hybrid networking*.
- Offloading of network data processing from the host central processing unit (CPU) to compatible network cards and data processing units (DPUs). This is known as *hardware offloading*.
- IPv4-primary dual-stack networking on bare-metal, VMware vSphere, IBM Power®, IBM Z®, and Red Hat OpenStack Platform (RHOSP) platforms.
- IPv6 single-stack networking on RHOSP and bare metal platforms.
- IPv6-primary dual-stack networking for a cluster running on a bare-metal, a VMware vSphere, or an RHOSP platform.
- Egress firewall devices and egress IP addresses.
- Egress router devices that operate in redirect mode.
- IPsec encryption of intracluster communications.

### 18.1.2. OVN-Kubernetes IPv6 and dual-stack limitations

The OVN-Kubernetes network plugin has the following limitations:

- For clusters configured for dual-stack networking, both IPv4 and IPv6 traffic must use the same network interface as the default gateway. If this requirement is not met, pods on the host in the **ovnkube-node** daemon set enter the **CrashLoopBackOff** state. If you display a pod with a command such as **oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml**, the **status** field contains more than one message about the default gateway, as shown in the following output:

```
I1006 16:09:50.985852 60651 helper_linux.go:73] Found default gateway interface br-ex
192.168.127.1
I1006 16:09:50.985923 60651 helper_linux.go:73] Found default gateway interface ens4
fe80::5054:ff:febe:bcd4
F1006 16:09:50.985939 60651 ovnkube.go:130] multiple gateway interfaces detected: br-ex
ens4
```

The only resolution is to reconfigure the host networking so that both IP families use the same network interface for the default gateway.

- For clusters configured for dual-stack networking, both the IPv4 and IPv6 routing tables must contain the default gateway. If this requirement is not met, pods on the host in the **ovnkube-node** daemon set enter the **CrashLoopBackOff** state. If you display a pod with a command such as **oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml**, the **status** field contains more than one message about the default gateway, as shown in the following output:

```
I0512 19:07:17.589083 108432 helper_linux.go:74] Found default gateway interface br-ex
192.168.123.1
F0512 19:07:17.589141 108432 ovnkube.go:133] failed to get default gateway interface
```

The only resolution is to reconfigure the host networking so that both IP families contain the default gateway.

### 18.1.3. Session affinity

Session affinity is a feature that applies to Kubernetes **Service** objects. You can use *session affinity* if you want to ensure that each time you connect to a <service\_VIP>:<Port>, the traffic is always load balanced to the same back end. For more information, including how to set session affinity based on a client's IP address, see [Session affinity](#).

#### Stickiness timeout for session affinity

The OVN-Kubernetes network plugin for OpenShift Container Platform calculates the stickiness timeout for a session from a client based on the last packet. For example, if you run a **curl** command 10 times, the sticky session timer starts from the tenth packet not the first. As a result, if the client is continuously contacting the service, then the session never times out. The timeout starts when the service has not received a packet for the amount of time set by the **timeoutSeconds** parameter.

#### Additional resources

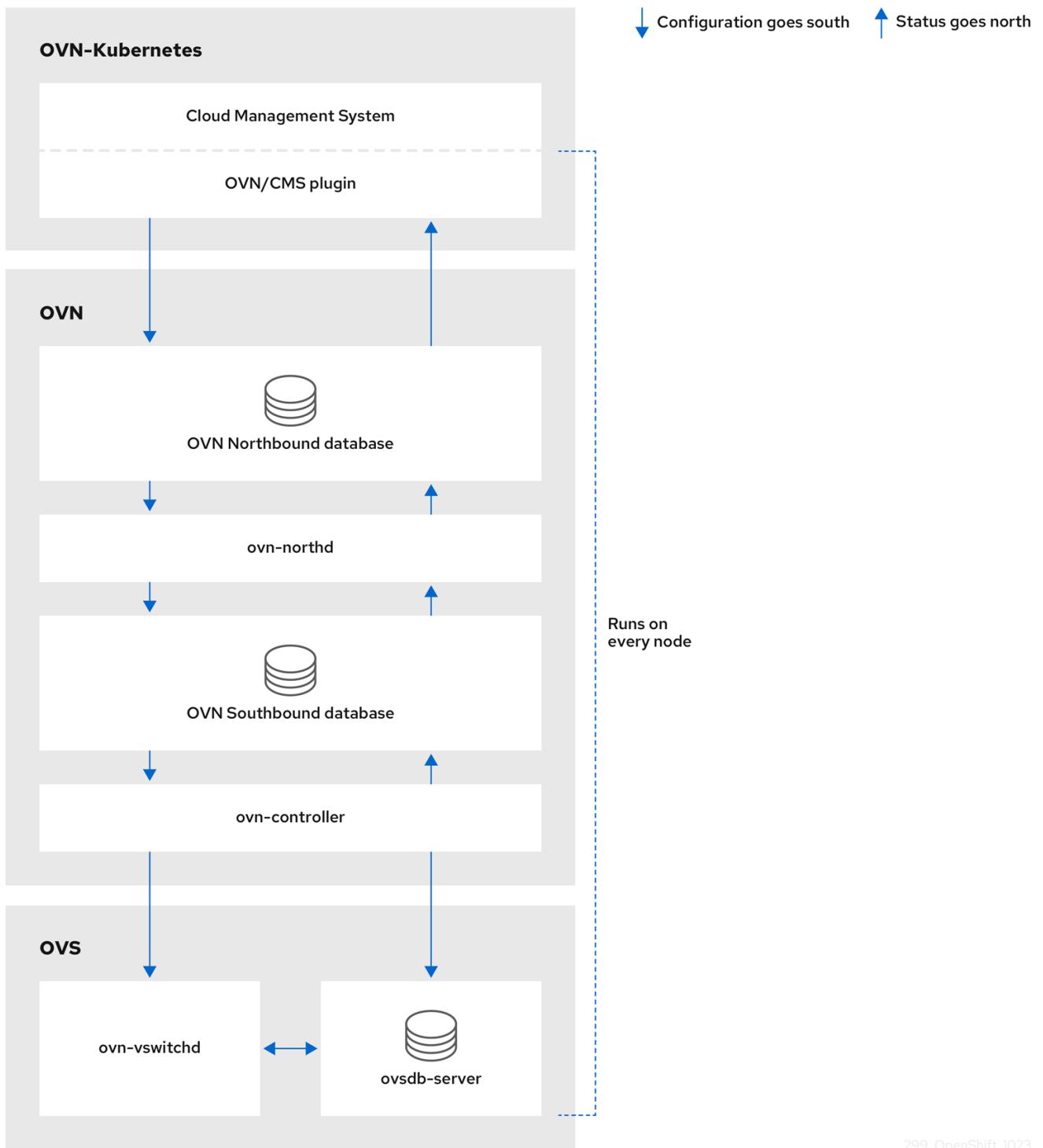
- [Configuring an egress firewall for a project](#)
- [About network policy](#)
- [Logging network policy events](#)
- [Enabling multicast for a project](#)
- [Configuring IPsec encryption](#)
- [Network \[operator.openshift.io/v1\]](#)

## 18.2. OVN-KUBERNETES ARCHITECTURE

### 18.2.1. Introduction to OVN-Kubernetes architecture

The following diagram shows the OVN-Kubernetes architecture.

Figure 18.1. OVK-Kubernetes architecture



299\_OpenShift\_1023

The key components are:

- **Cloud Management System (CMS)** - A platform specific client for OVN that provides a CMS specific plugin for OVN integration. The plugin translates the cloud management system's concept of the logical network configuration, stored in the CMS configuration database in a CMS-specific format, into an intermediate representation understood by OVN.
- **OVN Northbound database (**nbdb**) container** - Stores the logical network configuration passed by the CMS plugin.

- **OVN Southbound database (**sbdb**) container** - Stores the physical and logical network configuration state for Open vSwitch (OVS) system on each node, including tables that bind them.
- **OVN north daemon (**ovn-northd**)** - This is the intermediary client between **nbdb** container and **sbdb** container. It translates the logical network configuration in terms of conventional network concepts, taken from the **nbdb** container, into logical data path flows in the **sbdb** container. The container name for **ovn-northd** daemon is **northd** and it runs in the **ovnkube-node** pods.
- **ovn-controller** - This is the OVN agent that interacts with OVS and hypervisors, for any information or update that is needed for **sbdb** container. The **ovn-controller** reads logical flows from the **sbdb** container, translates them into **OpenFlow** flows and sends them to the node's OVS daemon. The container name is **ovn-controller** and it runs in the **ovnkube-node** pods.

The OVN northd, northbound database, and southbound database run on each node in the cluster and mostly contain and process information that is local to that node.

The OVN northbound database has the logical network configuration passed down to it by the cloud management system (CMS). The OVN northbound database contains the current desired state of the network, presented as a collection of logical ports, logical switches, logical routers, and more. The **ovn-northd** (**northd** container) connects to the OVN northbound database and the OVN southbound database. It translates the logical network configuration in terms of conventional network concepts, taken from the OVN northbound database, into logical data path flows in the OVN southbound database.

The OVN southbound database has physical and logical representations of the network and binding tables that link them together. It contains the chassis information of the node and other constructs like remote transit switch ports that are required to connect to the other nodes in the cluster. The OVN southbound database also contains all the logic flows. The logic flows are shared with the **ovn-controller** process that runs on each node and the **ovn-controller** turns those into **OpenFlow** rules to program **Open vSwitch**(OVS).

The Kubernetes control plane nodes contain two **ovnkube-control-plane** pods on separate nodes, which perform the central IP address management (IPAM) allocation for each node in the cluster. At any given time, a single **ovnkube-control-plane** pod is the leader.

### 18.2.2. Listing all resources in the OVN-Kubernetes project

Finding the resources and containers that run in the OVN-Kubernetes project is important to help you understand the OVN-Kubernetes networking implementation.

#### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- The OpenShift CLI (**oc**) installed.

#### Procedure

1. Run the following command to get all resources, endpoints, and **ConfigMaps** in the OVN-Kubernetes project:

```
$ oc get all,ep,cm -n openshift-ovn-kubernetes
```

#### Example output

■

```

Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in
v4.10000+
NAME                  READY  STATUS   RESTARTS  AGE
pod/ovnkube-control-plane-65c6f55656-6d55h  2/2   Running  0          114m
pod/ovnkube-control-plane-65c6f55656-fd7vw  2/2   Running  2 (104m ago)  114m
pod/ovnkube-node-bcvts                8/8    Running  0          113m
pod/ovnkube-node-drgvv                8/8    Running  0          113m
pod/ovnkube-node-f2pxt                8/8    Running  0          113m
pod/ovnkube-node-frqsb                8/8    Running  0          105m
pod/ovnkube-node-lbxkk                8/8    Running  0          105m
pod/ovnkube-node-tt7bx                8/8    Running  1 (102m ago)  105m

NAME           TYPE      CLUSTER-IP   EXTERNAL-IP PORT(S)      AGE
service/ovn-kubernetes-control-plane ClusterIP  None       <none>     9108/TCP
114m
service/ovn-kubernetes-node        ClusterIP  None       <none>     9103/TCP,9105/TCP
114m

NAME           DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE
NODE SELECTOR          AGE
daemonset.apps/ovnkube-node  6        6        6        6
beta.kubernetes.io/os=linux  114m

NAME           READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/ovnkube-control-plane  3/3   3        3        114m

NAME           DESIRED  CURRENT  READY  AGE
replicaset.apps/ovnkube-control-plane-65c6f55656  3        3        3        114m

NAME           ENDPOINTS          AGE
endpoints/ovn-kubernetes-control-plane  10.0.0.3:9108,10.0.0.4:9108,10.0.0.5:9108
114m
endpoints/ovn-kubernetes-node        10.0.0.3:9105,10.0.0.4:9105,10.0.0.5:9105 + 9 more...
114m

NAME           DATA  AGE
configmap/control-plane-status  1    113m
configmap/kube-root-ca.crt     1    114m
configmap/openshift-service-ca.crt  1    114m
configmap/ovn-ca               1    114m
configmap/ovnkube-config       1    114m
configmap/signer-ca           1    114m

```

There is one **ovnkube-node** pod for each node in the cluster. The **ovnkube-config** config map has the OpenShift Container Platform OVN-Kubernetes configurations.

2. List all of the containers in the **ovnkube-node** pods by running the following command:

```
$ oc get pods ovnkube-node-bcvts -o jsonpath='{.spec.containers[*].name}' -n openshift-ovn-
kubernetes
```

### Expected output

```
ovn-controller ovn-acl-logging kube-rbac-proxy-node kube-rbac-proxy-ovn-metrics northd
nbdb sbdb ovnkube-controller
```

The **ovnkube-node** pod is made up of several containers. It is responsible for hosting the northbound database (**nbdb** container), the southbound database (**sbdb** container), the north daemon (**northd** container), **ovn-controller** and the **ovnkube-controller** container. The **ovnkube-controller** container watches for API objects like pods, egress IPs, namespaces, services, endpoints, egress firewall, and network policies. It is also responsible for allocating pod IP from the available subnet pool for that node.

3. List all the containers in the **ovnkube-control-plane** pods by running the following command:

```
$ oc get pods ovnkube-control-plane-65c6f55656-6d55h -o
jsonpath='{.spec.containers[*].name}' -n openshift-ovn-kubernetes
```

#### Expected output

```
kube-rbac-proxy ovnkube-cluster-manager
```

The **ovnkube-control-plane** pod has a container (**ovnkube-cluster-manager**) that resides on each OpenShift Container Platform node. The **ovnkube-cluster-manager** container allocates pod subnet, transit switch subnet IP and join switch subnet IP to each node in the cluster. The **kube-rbac-proxy** container monitors metrics for the **ovnkube-cluster-manager** container.

### 18.2.3. Listing the OVN-Kubernetes northbound database contents

Each node is controlled by the **ovnkube-controller** container running in the **ovnkube-node** pod on that node. To understand the OVN logical networking entities you need to examine the northbound database that is running as a container inside the **ovnkube-node** pod on that node to see what objects are in the node you wish to see.

#### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- The OpenShift CLI (**oc**) installed.



#### PROCEDURE

To run ovn **nbctl** or **sbctl** commands in a cluster you must open a remote shell into the **nbdb** or **sbdb** containers on the relevant node

1. List pods by running the following command:

```
$ oc get po -n openshift-ovn-kubernetes
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
ovnkube-control-plane-8444dff7f9-4lh9k	2/2	Running	0	27m
ovnkube-control-plane-8444dff7f9-5rjh9	2/2	Running	0	27m
ovnkube-node-55xs2	8/8	Running	0	26m
ovnkube-node-7r84r	8/8	Running	0	16m
ovnkube-node-bqq8p	8/8	Running	0	17m

ovnkube-node-mkj4f	8/8	Running	0	26m
ovnkube-node-mlr8k	8/8	Running	0	26m
ovnkube-node-wqn2m	8/8	Running	0	16m

2. Optional: To list the pods with node information, run the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
<b>NOMINATED NODE</b> READINESS GATES						
ovnkube-control-plane-8444dff7f9-4lh9k	2/2	Running	0	27m	10.0.0.3	ci-In-t487nnb-72292-mdcnq-master-1
	<none>	<none>				
ovnkube-control-plane-8444dff7f9-5rjh9	2/2	Running	0	27m	10.0.0.4	ci-In-t487nnb-72292-mdcnq-master-2
	<none>	<none>				
ovnkube-node-55xs2	8/8	Running	0	26m	10.0.0.4	ci-In-t487nnb-72292-mdcnq-master-2
	<none>	<none>				
ovnkube-node-7r84r	8/8	Running	0	17m	10.0.128.3	ci-In-t487nnb-72292-mdcnq-worker-b-wbz7z
	<none>	<none>				
ovnkube-node-bqq8p	8/8	Running	0	17m	10.0.128.2	ci-In-t487nnb-72292-mdcnq-worker-a-lh7ms
	<none>	<none>				
ovnkube-node-mkj4f	8/8	Running	0	27m	10.0.0.5	ci-In-t487nnb-72292-mdcnq-master-0
	<none>	<none>				
ovnkube-node-mlr8k	8/8	Running	0	27m	10.0.0.3	ci-In-t487nnb-72292-mdcnq-master-1
	<none>	<none>				
ovnkube-node-wqn2m	8/8	Running	0	17m	10.0.128.4	ci-In-t487nnb-72292-mdcnq-worker-c-przlm
	<none>	<none>				

3. Navigate into a pod to look at the northbound database by running the following command:

```
$ oc rsh -c nbdb -n openshift-ovn-kubernetes ovnkube-node-55xs2
```

4. Run the following command to show all the objects in the northbound database:

```
$ ovn-nbctl show
```

The output is too long to list here. The list includes the NAT rules, logical switches, load balancers and so on.

You can narrow down and focus on specific components by using some of the following optional commands:

- a. Run the following command to show the list of logical routers:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c northd -- ovn-nbctl lr-list
```

### Example output

```
45339f4f-7d0b-41d0-b5f9-9fcfa9ce40ce6 (GR_ci-In-t487nnb-72292-mdcnq-master-2)
96a0a0f0-e7ed-4fec-8393-3195563de1b8 (ovn_cluster_router)
```

**NOTE**

From this output you can see there is router on each node plus an **ovn\_cluster\_router**.

- b. Run the following command to show the list of logical switches:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb -- ovn-nbctl ls-list
```

**Example output**

```
bdd7dc3d-d848-4a74-b293-cc15128ea614 (ci-In-t487nnb-72292-mdcnq-master-2)
b349292d-ee03-4914-935f-1940b6cb91e5 (ext_ci-In-t487nnb-72292-mdcnq-master-2)
0aac0754-ea32-4e33-b086-35eeabf0a140 (join)
992509d7-2c3f-4432-88db-c179e43592e5 (transit_switch)
```

**NOTE**

From this output you can see there is an ext switch for each node plus switches with the node name itself and a join switch.

- c. Run the following command to show the list of load balancers:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb -- ovn-nbctl lb-list
```

**Example output**

UUID	LB	PROTO	VIP	IPs
7c84c673-ed2a-4436-9a1f-9bc5dd181eea	Service_default/	tcp	172.30.0.1:443	10.0.0.3:6443,169.254.169.2:6443,10.0.0.5:6443
4d663fd9-ddc8-4271-b333-4c0e279e20bb	Service_default/	tcp	172.30.0.1:443	10.0.0.3:6443,10.0.0.4:6443,10.0.0.5:6443
292eb07f-b82f-4962-868a-4f541d250bca	Service_Openshift	tcp	172.30.105.247:443	10.129.0.12:8443
034b5a7f-bb6a-45e9-8e6d-573a82dc5ee3	Service_Openshift	tcp	172.30.192.38:443	10.0.0.3:10259,10.0.0.4:10259,10.0.0.5:10259
a68bb53e-be84-48df-bd38-bdd82fc4026	Service_Openshift	tcp	172.30.161.125:8443	10.129.0.32:8443
6cc21b3d-2c54-4c94-8ff5-d8e017269c2e	Service_Openshift	tcp	172.30.3.144:443	10.129.0.22:8443
37996ffd-7268-4862-a27f-61cd62e09c32	Service_Openshift	tcp	172.30.181.107:443	10.129.0.18:8443
81d4da3c-f811-411f-ae0c-bc6713d0861d	Service_Openshift	tcp	172.30.228.23:443	10.129.0.29:8443
ac5a4f3b-b6ba-4ceb-82d0-d84f2c41306e	Service_Openshift	tcp	172.30.14.240:9443	10.129.0.36:9443
c88979fb-1ef5-414b-90ac-43b579351ac9	Service_Openshift	tcp	172.30.231.192:9001	10.128.0.5:9001,10.128.2.5:9001,10.129.0.5:9001,10.129.2.4:9001,10.130.0.3:9001,10.131.0.3:9001
fcb0a3fb-4a77-4230-a84a-be45dce757e8	Service_Openshift	tcp		

```

172.30.189.92:443 10.130.0.17:8440
67ef3e7b-ceb9-4bf0-8d96-b43bde4c9151 Service_openshif tcp
172.30.67.218:443 10.129.0.9:8443
d0032fba-7d5e-424a-af25-4ab9b5d46e81 Service_openshif tcp
172.30.102.137:2379 10.0.0.3:2379,10.0.0.4:2379,10.0.0.5:2379
tcp 172.30.102.137:9979
10.0.0.3:9979,10.0.0.4:9979,10.0.0.5:9979
7361c537-3eec-4e6c-bc0c-0522d182abd4 Service_openshif tcp
172.30.198.215:9001
10.0.0.3:9001,10.0.0.4:9001,10.0.0.5:9001,10.0.128.2:9001,10.0.128.3:9001,10.0.128.4:9
001
0296c437-1259-410b-a6fd-81c310ad0af5 Service_openshif tcp
172.30.198.215:9001
10.0.0.3:9001,169.254.169.2:9001,10.0.0.5:9001,10.0.128.2:9001,10.0.128.3:9001,10.0.1
28.4:9001
5d5679f5-45b8-479d-9f7c-08b123c688b8 Service_openshif tcp
172.30.38.253:17698 10.128.0.52:17698,10.129.0.84:17698,10.130.0.60:17698
2adcbab4-d1c9-447d-9573-b5dc9f2efbfa Service_openshif tcp
172.30.148.52:443 10.0.0.4:9202,10.0.0.5:9202
tcp 172.30.148.52:444
10.0.0.4:9203,10.0.0.5:9203
tcp 172.30.148.52:445
10.0.0.4:9204,10.0.0.5:9204
tcp 172.30.148.52:446
10.0.0.4:9205,10.0.0.5:9205
2a33a6d7-af1b-4892-87cc-326a380b809b Service_openshif tcp
172.30.67.219:9091 10.129.2.16:9091,10.131.0.16:9091
tcp 172.30.67.219:9092
10.129.2.16:9092,10.131.0.16:9092
tcp 172.30.67.219:9093
10.129.2.16:9093,10.131.0.16:9093
tcp 172.30.67.219:9094
10.129.2.16:9094,10.131.0.16:9094
f56f59d7-231a-4974-99b3-792e2741ec8d Service_openshif tcp
172.30.89.212:443 10.128.0.41:8443,10.129.0.68:8443,10.130.0.44:8443
08c2c6d7-d217-4b96-b5d8-c80c4e258116 Service_openshif tcp
172.30.102.137:2379 10.0.0.3:2379,169.254.169.2:2379,10.0.0.5:2379
tcp 172.30.102.137:9979
10.0.0.3:9979,169.254.169.2:9979,10.0.0.5:9979
60a69c56-fc6a-4de6-bd88-3f2af5ba5665 Service_openshif tcp
172.30.10.193:443 10.129.0.25:8443
ab1ef694-0826-4671-a22c-565fc2d282ec Service_openshif tcp
172.30.196.123:443 10.128.0.33:8443,10.129.0.64:8443,10.130.0.37:8443
b1fb34d3-0944-4770-9ee3-2683e7a630e2 Service_openshif tcp
172.30.158.93:8443 10.129.0.13:8443
95811c11-56e2-4877-be1e-c78ccb3a82a9 Service_openshif tcp
172.30.46.85:9001 10.130.0.16:9001
4bab1d1-b873-4535-884c-3f6fc07a50fd Service_openshif tcp 172.30.28.87:443
10.129.0.26:8443
6c2e1c90-f0ca-484e-8a8e-40e71442110a Service_openshif udp 172.30.0.10:53
10.128.0.13:5353,10.128.2.6:5353,10.129.0.39:5353,10.129.2.6:5353,10.130.0.11:5353,1
0.131.0.9:5353

```

**NOTE**

From this truncated output you can see there are many OVN-Kubernetes load balancers. Load balancers in OVN-Kubernetes are representations of services.

- Run the following command to display the options available with the command **ovn-nbctl**:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb ovn-nbctl --help
```

#### 18.2.4. Command-line arguments for ovn-nbctl to examine northbound database contents

The following table describes the command-line arguments that can be used with **ovn-nbctl** to examine the contents of the northbound database.

**NOTE**

Open a remote shell in the pod you want to view the contents of and then run the **ovn-nbctl** commands.

**Table 18.1. Command-line arguments to examine northbound database contents**

Argument	Description
<b>ovn-nbctl show</b>	An overview of the northbound database contents as seen from a specific node.
<b>ovn-nbctl show &lt;switch_or_router&gt;</b>	Show the details associated with the specified switch or router.
<b>ovn-nbctl lr-list</b>	Show the logical routers.
<b>ovn-nbctl lrp-list &lt;router&gt;</b>	Using the router information from <b>ovn-nbctl lr-list</b> to show the router ports.
<b>ovn-nbctl lr-nat-list &lt;router&gt;</b>	Show network address translation details for the specified router.
<b>ovn-nbctl ls-list</b>	Show the logical switches
<b>ovn-nbctl lsp-list &lt;switch&gt;</b>	Using the switch information from <b>ovn-nbctl ls-list</b> to show the switch port.
<b>ovn-nbctl lsp-get-type &lt;port&gt;</b>	Get the type for the logical port.
<b>ovn-nbctl lb-list</b>	Show the load balancers.

## 18.2.5. Listing the OVN-Kubernetes southbound database contents

Each node is controlled by the **ovnkube-controller** container running in the **ovnkube-node** pod on that node. To understand the OVN logical networking entities you need to examine the northbound database that is running as a container inside the **ovnkube-node** pod on that node to see what objects are in the node you wish to see.

### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- The OpenShift CLI (**oc**) installed.



### PROCEDURE

To run ovn **nbctl** or **sbctl** commands in a cluster you must open a remote shell into the **nbdb** or **sbdb** containers on the relevant node

1. List the pods by running the following command:

```
$ oc get po -n openshift-ovn-kubernetes
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE
ovnkube-control-plane-8444dff7f9-4lh9k	2/2	Running	0	27m
ovnkube-control-plane-8444dff7f9-5rjh9	2/2	Running	0	27m
ovnkube-node-55xs2	8/8	Running	0	26m
ovnkube-node-7r84r	8/8	Running	0	16m
ovnkube-node-bqq8p	8/8	Running	0	17m
ovnkube-node-mkj4f	8/8	Running	0	26m
ovnkube-node-mlr8k	8/8	Running	0	26m
ovnkube-node-wqn2m	8/8	Running	0	16m

2. Optional: To list the pods with node information, run the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
<b>NOMINATED NODE READINESS GATES</b>						
ovnkube-control-plane-8444dff7f9-4lh9k	2/2	Running	0	27m	10.0.0.3	ci-In-t487nnb-72292-mdcnq-master-1
ovnkube-control-plane-8444dff7f9-5rjh9	2/2	Running	0	27m	10.0.0.4	ci-In-t487nnb-72292-mdcnq-master-2
ovnkube-node-55xs2	8/8	Running	0	26m	10.0.0.4	ci-In-t487nnb-72292-mdcnq-master-2
ovnkube-node-7r84r	8/8	Running	0	17m	10.0.128.3	ci-In-t487nnb-72292-mdcnq-worker-b-wbz7z
ovnkube-node-bqq8p	8/8	Running	0	17m	10.0.128.2	ci-In-t487nnb-72292-mdcnq-worker-a-lh7ms
ovnkube-node-mkj4f	8/8	Running	0	27m	10.0.0.5	ci-In-t487nnb-72292-mdcnq-worker-a-lh7ms

```

72292-mdcnq-master-0      <none>      <none>
ovnkube-node-mlr8k          8/8   Running  0       27m  10.0.0.3  ci-In-t487nnb-
72292-mdcnq-master-1      <none>      <none>
ovnkube-node-wqn2m          8/8   Running  0       17m  10.0.128.4 ci-In-
t487nnb-72292-mdcnq-worker-c-przlm <none>      <none>

```

3. Navigate into a pod to look at the southbound database:

```
$ oc rsh -c sbdb -n openshift-ovn-kubernetes ovnkube-node-55xs2
```

4. Run the following command to show all the objects in the southbound database:

```
$ ovn-sbctl show
```

### Example output

```

Chassis "5db31703-35e9-413b-8cdf-69e7eecb41f7"
  hostname: ci-In-9gp362t-72292-v2p94-worker-a-8bmwz
  Encap geneve
    ip: "10.0.128.4"
    options: {csum="true"}
  Port_Binding tstor-ci-In-9gp362t-72292-v2p94-worker-a-8bmwz
Chassis "070debed-99b7-4bce-b17d-17e720b7f8bc"
  hostname: ci-In-9gp362t-72292-v2p94-worker-b-svmp6
  Encap geneve
    ip: "10.0.128.2"
    options: {csum="true"}
  Port_Binding k8s-ci-In-9gp362t-72292-v2p94-worker-b-svmp6
  Port_Binding rtoe-GR_ci-In-9gp362t-72292-v2p94-worker-b-svmp6
  Port_Binding openshift-monitoring_alertmanager-main-1
  Port_Binding rtoj-GR_ci-In-9gp362t-72292-v2p94-worker-b-svmp6
  Port_Binding etor-GR_ci-In-9gp362t-72292-v2p94-worker-b-svmp6
  Port_Binding cr-rtos-ci-In-9gp362t-72292-v2p94-worker-b-svmp6
  Port_Binding openshift-e2e-loki_loki-promtail-qcrcz
  Port_Binding jtor-GR_ci-In-9gp362t-72292-v2p94-worker-b-svmp6
  Port_Binding openshift-multus_network-metrics-daemon-mkd4t
  Port_Binding openshift-ingress-canary_ingress-canary-xtvj4
  Port_Binding openshift-ingress_router-default-6c76cbc498-pvlqk
  Port_Binding openshift-dns_dns-default-zz582
  Port_Binding openshift-monitoring_thanos-querier-57585899f5-lbf4f
  Port_Binding openshift-network-diagnostics_network-check-target-tn228
  Port_Binding openshift-monitoring_prometheus-k8s-0
  Port_Binding openshift-image-registry_image-registry-68899bd877-xqxjj
Chassis "179ba069-0af1-401c-b044-e5ba90f60fea"
  hostname: ci-In-9gp362t-72292-v2p94-master-0
  Encap geneve
    ip: "10.0.0.5"
    options: {csum="true"}
  Port_Binding tstor-ci-In-9gp362t-72292-v2p94-master-0
Chassis "68c954f2-5a76-47be-9e84-1cb13bd9dab9"
  hostname: ci-In-9gp362t-72292-v2p94-worker-c-mjf9w
  Encap geneve
    ip: "10.0.128.3"
    options: {csum="true"}
  Port_Binding tstor-ci-In-9gp362t-72292-v2p94-worker-c-mjf9w

```

```

Chassis "2de65d9e-9abf-4b6e-a51d-a1e038b4d8af"
  hostname: ci-In-9gp362t-72292-v2p94-master-2
  Encap geneve
    ip: "10.0.0.4"
    options: {csum="true"}
  Port_Binding tstor-ci-In-9gp362t-72292-v2p94-master-2
Chassis "1d371cb8-5e21-44fd-9025-c4b162cc4247"
  hostname: ci-In-9gp362t-72292-v2p94-master-1
  Encap geneve
    ip: "10.0.0.3"
    options: {csum="true"}
  Port_Binding tstor-ci-In-9gp362t-72292-v2p94-master-1

```

This detailed output shows the chassis and the ports that are attached to the chassis which in this case are all of the router ports and anything that runs like host networking. Any pods communicate out to the wider network using source network address translation (SNAT). Their IP address is translated into the IP address of the node that the pod is running on and then sent out into the network.

In addition to the chassis information the southbound database has all the logic flows and those logic flows are then sent to the **ovn-controller** running on each of the nodes. The **ovn-controller** translates the logic flows into open flow rules and ultimately programs **OpenvSwitch** so that your pods can then follow open flow rules and make it out of the network.

- Run the following command to display the options available with the command **ovn-sbctl**:

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c sbdb ovn-sbctl --help
```

### 18.2.6. Command-line arguments for ovn-sbctl to examine southbound database contents

The following table describes the command-line arguments that can be used with **ovn-sbctl** to examine the contents of the southbound database.



#### NOTE

Open a remote shell in the pod you wish to view the contents of and then run the **ovn-sbctl** commands.

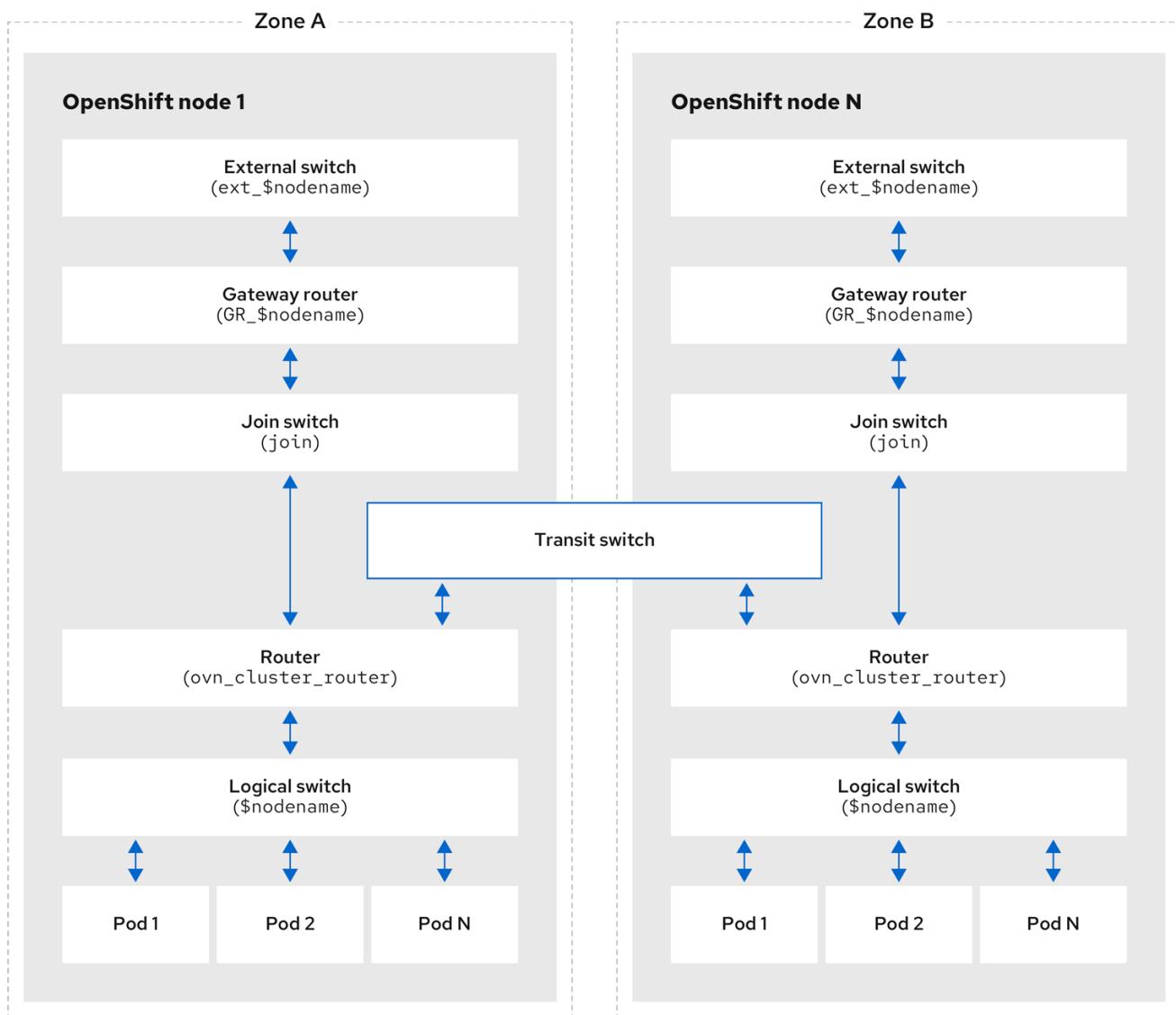
**Table 18.2. Command-line arguments to examine southbound database contents**

Argument	Description
<b>ovn-sbctl show</b>	An overview of the southbound database contents as seen from a specific node.
<b>ovn-sbctl list Port_Binding &lt;port&gt;</b>	List the contents of southbound database for a the specified port .
<b>ovn-sbctl dump-flows</b>	List the logical flows.

### 18.2.7. OVN-Kubernetes logical architecture

OVN is a network virtualization solution. It creates logical switches and routers. These switches and routers are interconnected to create any network topologies. When you run **ovnkube-trace** with the log level set to 2 or 5 the OVN-Kubernetes logical components are exposed. The following diagram shows how the routers and switches are connected in OpenShift Container Platform.

Figure 18.2. OVN-Kubernetes router and switch components



299\_OpenShift\_1023

The key components involved in packet processing are:

#### Gateway routers

Gateway routers sometimes called L3 gateway routers, are typically used between the distributed routers and the physical network. Gateway routers including their logical patch ports are bound to a physical location (not distributed), or chassis. The patch ports on this router are known as l3gateway ports in the ovn-southbound database (**ovn-sbdb**).

#### Distributed logical routers

Distributed logical routers and the logical switches behind them, to which virtual machines and containers attach, effectively reside on each hypervisor.

#### Join local switch

Join local switches are used to connect the distributed router and gateway routers. It reduces the number of IP addresses needed on the distributed router.

### Logical switches with patch ports

Logical switches with patch ports are used to virtualize the network stack. They connect remote logical ports through tunnels.

### Logical switches with localnet ports

Logical switches with localnet ports are used to connect OVN to the physical network. They connect remote logical ports by bridging the packets to directly connected physical L2 segments using localnet ports.

### Patch ports

Patch ports represent connectivity between logical switches and logical routers and between peer logical routers. A single connection has a pair of patch ports at each such point of connectivity, one on each side.

### I3gateway ports

I3gateway ports are the port binding entries in the **ovn-sbdb** for logical patch ports used in the gateway routers. They are called I3gateway ports rather than patch ports just to portray the fact that these ports are bound to a chassis just like the gateway router itself.

### localnet ports

localnet ports are present on the bridged logical switches that allows a connection to a locally accessible network from each **ovn-controller** instance. This helps model the direct connectivity to the physical network from the logical switches. A logical switch can only have a single localnet port attached to it.

## 18.2.7.1. Installing network-tools on local host

Install **network-tools** on your local host to make a collection of tools available for debugging OpenShift Container Platform cluster network issues.

### Procedure

1. Clone the **network-tools** repository onto your workstation with the following command:

```
$ git clone git@github.com:openshift/network-tools.git
```

2. Change into the directory for the repository you just cloned:

```
$ cd network-tools
```

3. Optional: List all available commands:

```
$ ./debug-scripts/network-tools -h
```

## 18.2.7.2. Running network-tools

Get information about the logical switches and routers by running **network-tools**.

### Prerequisites

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

- You are logged in to the cluster as a user with **cluster-admin** privileges.
- You have installed **network-tools** on local host.

## Procedure

1. List the routers by running the following command:

```
$ ./debug-scripts/network-tools ovn-db-run-command ovn-nbctl lr-list
```

### Example output

```
944a7b53-7948-4ad2-a494-82b55eeccf87 (GR_ci-In-54932yb-72292-kd676-worker-c-rzj99)
84bd4a4c-4b0b-4a47-b0cf-a2c32709fc53 (ovn_cluster_router)
```

2. List the localnet ports by running the following command:

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=localnet
```

### Example output

```
_uuid          : d05298f5-805b-4838-9224-1211afc2f199
additional_chassis : []
additional_encap  : []
chassis         : []
datapath        : f3c2c959-743b-4037-854d-26627902597c
encap           : []
external_ids    : {}
gateway_chassis : []
ha_chassis_group : []
logical_port    : br-ex_ci-In-54932yb-72292-kd676-worker-c-rzj99
mac             : [unknown]
mirror_rules    : []
nat_addresses   : []
options         : {network_name=physnet}
parent_port     : []
port_security   : []
requested_additional_chassis: []
requested_chassis : []
tag              : []
tunnel_key      : 2
type            : localnet
up              : false
virtual_parent  : []

[...]
```

3. List the **I3gateway** ports by running the following command:

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=I3gateway
```

## Example output

```

_uuid          : 5207a1f3-1cf3-42f1-83e9-387bbb06b03c
additional_chassis : []
additional_encap  : []
chassis        : ca6eb600-3a10-4372-a83e-e0d957c4cd92
datapath       : f3c2c959-743b-4037-854d-26627902597c
encap          : []
external_ids   : {}
gateway_chassis : []
ha_chassis_group : []
logical_port   : etor-GR_ci-In-54932yb-72292-kd676-worker-c-rzj99
mac            : ["42:01:0a:00:80:04"]
mirror_rules   : []
nat_addresses  : ["42:01:0a:00:80:04 10.0.128.4"]
options        : {l3gateway-chassis="84737c36-b383-4c83-92c5-2bd5b3c7e772", peer=rtoe-
GR_ci-In-54932yb-72292-kd676-worker-c-rzj99}
parent_port    : []
port_security  : []
requested_additional_chassis: []
requested_chassis : []
tag             : []
tunnel_key     : 1
type           : l3gateway
up             : true
virtual_parent : []

_uuid          : 6088d647-84f2-43f2-b53f-c9d379042679
additional_chassis : []
additional_encap  : []
chassis        : ca6eb600-3a10-4372-a83e-e0d957c4cd92
datapath       : dc9cea00-d94a-41b8-bdb0-89d42d13aa2e
encap          : []
external_ids   : {}
gateway_chassis : []
ha_chassis_group : []
logical_port   : jtor-GR_ci-In-54932yb-72292-kd676-worker-c-rzj99
mac            : [router]
mirror_rules   : []
nat_addresses  : []
options        : {l3gateway-chassis="84737c36-b383-4c83-92c5-2bd5b3c7e772", peer=rtoj-
GR_ci-In-54932yb-72292-kd676-worker-c-rzj99}
parent_port    : []
port_security  : []
requested_additional_chassis: []
requested_chassis : []
tag             : []
tunnel_key     : 2
type           : l3gateway
up             : true
virtual_parent : []

[...]

```

4. List the patch ports by running the following command:

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=patch
```

### Example output

```
_uuid          : 785fb8b6-ee5a-4792-a415-5b1cb855dac2
additional_chassis : []
additional_encap   : []
chassis         : []
datapath        : f1ddd1cc-dc0d-43b4-90ca-12651305acec
encap           : []
external_ids    : {}
gateway_chassis : []
ha_chassis_group : []
logical_port    : stor-ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac             : [router]
mirror_rules    : []
nat_addresses   : ["0a:58:0a:80:02:01 10.128.2.1 is_chassis_resident(\"cr-rtos-ci-ln-54932yb-72292-kd676-worker-c-rzj99\")"]
options         : {peer=rto
s-ci-ln-54932yb-72292-kd676-worker-c-rzj99}
parent_port     : []
port_security   : []
requested_additional_chassis: []
requested_chassis : []
tag             : []
tunnel_key      : 1
type            : patch
up              : false
virtual_parent  : []

_uuid          : c01ff587-21a5-40b4-8244-4cd0425e5d9a
additional_chassis : []
additional_encap   : []
chassis         : []
datapath        : f6795586-bf92-4f84-9222-efe4ac6a7734
encap           : []
external_ids    : {}
gateway_chassis : []
ha_chassis_group : []
logical_port    : rtoj-ovn_cluster_router
mac             : ["0a:58:64:40:00:01 100.64.0.1/16"]
mirror_rules    : []
nat_addresses   : []
options         : {peer=jtor-ovn_cluster_router}
parent_port     : []
port_security   : []
requested_additional_chassis: []
requested_chassis : []
tag             : []
tunnel_key      : 1
type            : patch
up              : false
virtual_parent  : []
[...]
```

## 18.2.8. Additional resources

- [Tracing Openflow with ovnkube-trace](#)
- [OVN architecture](#)
- [ovn-nbctl linux manual page](#)
- [ovn-sbctl linux manual page](#)

## 18.3. TROUBLESHOOTING OVN-KUBERNETES

OVN-Kubernetes has many sources of built-in health checks and logs. Follow the instructions in these sections to examine your cluster. If a support case is necessary, follow the [support guide](#) to collect additional information through a **must-gather**. Only use the `--gather-network-logs` when instructed by support.

### 18.3.1. Monitoring OVN-Kubernetes health by using readiness probes

The **ovnkube-control-plane** and **ovnkube-node** pods have containers configured with readiness probes.

#### Prerequisites

- Access to the OpenShift CLI (**oc**).
- You have access to the cluster with **cluster-admin** privileges.
- You have installed **jq**.

#### Procedure

1. Review the details of the **ovnkube-node** readiness probe by running the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node \
-o json | jq '.items[0].spec.containers[] | .name,.readinessProbe'
```

The readiness probe for the northbound and southbound database containers in the **ovnkube-node** pod checks for the health of the databases and the **ovnkube-controller** container.

The **ovnkube-controller** container in the **ovnkube-node** pod has a readiness probe to verify the presence of the OVN-Kubernetes CNI configuration file, the absence of which would indicate that the pod is not running or is not ready to accept requests to configure pods.

2. Show all events including the probe failures, for the namespace by using the following command:

```
$ oc get events -n openshift-ovn-kubernetes
```

3. Show the events for just a specific pod:

```
$ oc describe pod ovnkube-node-9lqfk -n openshift-ovn-kubernetes
```

4. Show the messages and statuses from the cluster network operator:

```
$ oc get co/network -o json | jq '.status.conditions[]'
```

5. Show the **ready** status of each container in **ovnkube-node** pods by running the following script:

```
$ for p in $(oc get pods --selector app=ovnkube-node -n openshift-ovn-kubernetes \
-o jsonpath='{range.items[*]}{" "}{.metadata.name}'); do echo === $p ===; \
oc get pods -n openshift-ovn-kubernetes $p -o json | jq '.status.containerStatuses[] | .name, \
.ready'; \
done
```



#### NOTE

The expectation is all container statuses are reporting as **true**. Failure of a readiness probe sets the status to **false**.

#### Additional resources

- [Monitoring application health by using health checks](#)

### 18.3.2. Viewing OVN-Kubernetes alerts in the console

The Alerting UI provides detailed information about alerts and their governing alerting rules and silences.

#### Prerequisites

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.

#### Procedure (UI)

1. In the **Administrator** perspective, select **Observe → Alerting**. The three main pages in the Alerting UI in this perspective are the **Alerts**, **Silences**, and **Alerting Rules** pages.
2. View the rules for OVN-Kubernetes alerts by selecting **Observe → Alerting → Alerting Rules**.

### 18.3.3. Viewing OVN-Kubernetes alerts in the CLI

You can get information about alerts and their governing alerting rules and silences from the command line.

#### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- The OpenShift CLI (**oc**) installed.
- You have installed **jq**.

#### Procedure

1. View active or firing alerts by running the following commands.
  - a. Set the alert manager route environment variable by running the following command:

```
$ ALERT_MANAGER=$(oc get route alertmanager-main -n openshift-monitoring \
-o jsonpath='{@.spec.host}'')
```

- b. Issue a **curl** request to the alert manager route API by running the following command, replacing **\$ALERT\_MANAGER** with the URL of your **Alertmanager** instance:

```
$ curl -s -k -H "Authorization: Bearer $(oc create token prometheus-k8s -n openshift-monitoring)" https://$ALERT_MANAGER/api/v1/alerts | jq '.data[] | "\(.labels.severity) \(.labels.alertname) \(.labels.pod) \(.labels.container) \(.labels.endpoint) \(.labels.instance)"'
```

2. View alerting rules by running the following command:

```
$ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -s \
'http://localhost:9090/api/v1/rules' | jq '.data.groups[].rules[] | select((.name|contains("ovn")) \
or (.name|contains("OVN")) or (.name|contains("Ovn")) or (.name|contains("North")) or \
(.name|contains("South")))) and .type=="alerting"'
```

#### 18.3.4. Viewing the OVN-Kubernetes logs using the CLI

You can view the logs for each of the pods in the **ovnkube-master** and **ovnkube-node** pods using the OpenShift CLI (**oc**).

##### Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- Access to the OpenShift CLI (**oc**).
- You have installed **jq**.

##### Procedure

1. View the log for a specific pod:

```
$ oc logs -f <pod_name> -c <container_name> -n <namespace>
```

where:

**-f**

Optional: Specifies that the output follows what is being written into the logs.

**<pod\_name>**

Specifies the name of the pod.

**<container\_name>**

Optional: Specifies the name of a container. When a pod has more than one container, you must specify the container name.

**<namespace>**

Specify the namespace the pod is running in.

For example:

```
$ oc logs ovnkube-node-5dx44 -n openshift-ovn-kubernetes
$ oc logs -f ovnkube-node-5dx44 -c ovnkube-controller -n openshift-ovn-kubernetes
```

The contents of log files are printed out.

- Examine the most recent entries in all the containers in the **ovnkube-node** pods:

```
$ for p in $(oc get pods --selector app=ovnkube-node -n openshift-ovn-kubernetes \
-o jsonpath='{range.items[*]}{" "}{.metadata.name}'); \
do echo === $p ===; for container in $(oc get pods -n openshift-ovn-kubernetes $p \
-o json | jq -r '.status.containerStatuses[] | .name'); do echo ---$container---; \
oc logs -c $container $p -n openshift-ovn-kubernetes --tail=5; done; done
```

- View the last 5 lines of every log in every container in an **ovnkube-node** pod using the following command:

```
$ oc logs -l app=ovnkube-node -n openshift-ovn-kubernetes --all-containers --tail 5
```

### 18.3.5. Viewing the OVN-Kubernetes logs using the web console

You can view the logs for each of the pods in the **ovnkube-master** and **ovnkube-node** pods in the web console.

#### Prerequisites

- Access to the OpenShift CLI (**oc**).

#### Procedure

- In the OpenShift Container Platform console, navigate to **Workloads → Pods** or navigate to the pod through the resource you want to investigate.
- Select the **openshift-ovn-kubernetes** project from the drop-down menu.
- Click the name of the pod you want to investigate.
- Click **Logs**. By default for the **ovnkube-master** the logs associated with the **northd** container are displayed.
- Use the down-down menu to select logs for each container in turn.

#### 18.3.5.1. Changing the OVN-Kubernetes log levels

The default log level for OVN-Kubernetes is 4. To debug OVN-Kubernetes, set the log level to 5. Follow this procedure to increase the log level of the OVN-Kubernetes to help you debug an issue.

#### Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.

## Procedure

- Run the following command to get detailed information for all pods in the OVN-Kubernetes project:

```
$ oc get po -o wide -n openshift-ovn-kubernetes
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
<b>NOMINATED NODE</b> READINESS GATES						
ovnkube-control-plane-65497d4548-9ptdr	2/2	Running	2 (128m ago)	147m	10.0.0.3	
ci-In-3njdr9b-72292-5nwkp-master-0	<none>	<none>				
ovnkube-control-plane-65497d4548-j6zfk	2/2	Running	0	147m	10.0.0.5	ci-In-3njdr9b-72292-5nwkp-master-2
ovnkube-node-5dx44	8/8	Running	0	146m	10.0.0.3	ci-In-3njdr9b-72292-5nwkp-master-0
ovnkube-node-dpfn4	8/8	Running	0	146m	10.0.0.4	ci-In-3njdr9b-72292-5nwkp-master-1
ovnkube-node-kwc9l	8/8	Running	0	134m	10.0.128.2	ci-In-3njdr9b-72292-5nwkp-worker-a-2fjcj
ovnkube-node-mcrhl	8/8	Running	0	134m	10.0.128.4	ci-In-3njdr9b-72292-5nwkp-worker-c-v9x5v
ovnkube-node-nsct4	8/8	Running	0	146m	10.0.0.5	ci-In-3njdr9b-72292-5nwkp-master-2
ovnkube-node-zrj9f	8/8	Running	0	134m	10.0.128.3	ci-In-3njdr9b-72292-5nwkp-worker-b-v78h7

- Create a **ConfigMap** file similar to the following example and use a filename such as **env-overrides.yaml**:

### Example ConfigMap file

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: env-overrides
  namespace: openshift-ovn-kubernetes
data:
  ci-In-3njdr9b-72292-5nwkp-master-0: | 1
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
  ci-In-3njdr9b-72292-5nwkp-master-2: |
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
    _master: | 2
      # This sets the log level for the ovn-kubernetes master process as well as the ovn-dbchecker:
      OVN_KUBE_LOG_LEVEL=5
```

```
# You might also/instead want to enable debug logging for northd, nbdb and sbdb on all masters:
OVN_LOG_LEVEL=dbg
```

- 1 Specify the name of the node you want to set the debug log level on.
- 2 Specify **\_master** to set the log levels of **ovnkube-master** components.

3. Apply the **ConfigMap** file by using the following command:

```
$ oc apply -n openshift-ovn-kubernetes -f env-overrides.yaml
```

#### Example output

```
configmap/env-overrides.yaml created
```

4. Restart the **ovnkube** pods to apply the new log level by using the following commands:

```
$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ci-ln-3njdr9b-72292-5nwkp-master-0 -l app=ovnkube-node

$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ci-ln-3njdr9b-72292-5nwkp-master-2 -l app=ovnkube-node

$ oc delete pod -n openshift-ovn-kubernetes -l app=ovnkube-node
```

5. To verify that the `ConfigMap` file has been applied to all nodes for a specific pod, run the following command:

```
$ oc logs -n openshift-ovn-kubernetes --all-containers --prefix ovnkube-node-<xxxx> | grep -E -m 10 '(Logging config:[|vconsole|DBG])'
```

where:

**<XXXX>**

Specifies the random sequence of letters for a pod from the previous step.

#### Example output

```
[pod/ovnkube-node-2cpjc/sbdb] + exec /usr/share/ovn/scripts/ovn-ctl --no-monitor '--ovn-sb-log=-vconsole:info -vfile:off -vPATTERN:console:%D{%Y-%m-%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' run_sb_ovsdb
[pod/ovnkube-node-2cpjc/ovnkube-controller] I1012 14:39:59.984506 35767 config.go:2247] Logging config: {File: CNIFile:/var/log/ovn-kubernetes/ovn-k8s-cni-overlay.log LibovsdbFile:/var/log/ovnkube/libovsdb.log Level:5 LogFileSize:100 LogFileMaxBackups:5 LogFileMaxAge:0 ACLLoggingRateLimit:20}
[pod/ovnkube-node-2cpjc/northd] + exec ovn-northd --no-chdir -vconsole:info -vfile:off -vPATTERN:console:%D{%Y-%m-%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' --pidfile /var/run/ovn/ovn-northd.pid --n-threads=1
[pod/ovnkube-node-2cpjc/nbdb] + exec /usr/share/ovn/scripts/ovn-ctl --no-monitor '--ovn-nb-log=-vconsole:info -vfile:off -vPATTERN:console:%D{%Y-%m-%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' run_nb_ovsdb
```

```
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.552Z|00002|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes, including 1
bucket with 6 nodes (32 nodes total across 32 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00003|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes, including 1
bucket with 6 nodes (64 nodes total across 64 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00004|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes, including 1
bucket with 7 nodes (32 nodes total across 32 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00005|reconnect|DBG|unix:/var/run/openvswitch/db.sock: entering
BACKOFF
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00007|reconnect|DBG|unix:/var/run/openvswitch/db.sock: entering
CONNECTING
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00008|ovsdb_cs|DBG|unix:/var/run/openvswitch/db.sock:
SERVER_SCHEMA_REQUESTED -> SERVER_SCHEMA_REQUESTED at lib/ovsdb-
cs.c:423
```

6. Optional: Check the **ConfigMap** file has been applied by running the following command:

```
for f in $(oc -n openshift-ovn-kubernetes get po -l 'app=ovnkube-node' --no-headers -o
custom-columns=N:.metadata.name) ; do echo "---- $f ----" ; oc -n openshift-ovn-kubernetes
exec -c ovnkube-controller $f -- pgrep -a -f init-ovnkube-controller | grep -P -o
'^.*loglevel\s+\d' ; done
```

### Example output

```
---- ovnkube-node-2dt57 ----
60981 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-c-vmh5n.c.openshift-
qe.internal --init-node xpst8-worker-c-vmh5n.c.openshift-qe.internal --config-
file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-4zznh ----
178034 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-2.c.openshift-qe.internal --
init-node xpst8-master-2.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-548sx ----
77499 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-a-fjtnb.c.openshift-qe.internal -
-init-node xpst8-worker-a-fjtnb.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-6btrf ----
73781 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-b-p8rww.c.openshift-
qe.internal --init-node xpst8-worker-b-p8rww.c.openshift-qe.internal --config-
file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-fkc9r ----
130707 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-0.c.openshift-qe.internal --
init-node xpst8-master-0.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 5
---- ovnkube-node-tk9l4 ----
181328 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-1.c.openshift-qe.internal --
init-node xpst8-master-1.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
```

### 18.3.6. Checking the OVN-Kubernetes pod network connectivity

The connectivity check controller, in OpenShift Container Platform 4.10 and later, orchestrates connection verification checks in your cluster. These include Kubernetes API, OpenShift API and individual nodes. The results for the connection tests are stored in **PodNetworkConnectivity** objects in the **openshift-network-diagnostics** namespace. Connection tests are performed every minute in parallel.

#### Prerequisites

- Access to the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.
- You have installed **jq**.

#### Procedure

1. To list the current **PodNetworkConnectivityCheck** objects, enter the following command:

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics
```

2. View the most recent success for each connection object by using the following command:

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]|.spec.targetEndpoint,.status.successes[0]'
```

3. View the most recent failures for each connection object by using the following command:

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]|.spec.targetEndpoint,.status.failures[0]'
```

4. View the most recent outages for each connection object by using the following command:

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]|.spec.targetEndpoint,.status.outages[0]'
```

The connectivity check controller also logs metrics from these checks into Prometheus.

5. View all the metrics by running the following command:

```
$ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
promtool query instant http://localhost:9090 \
'{component="openshift-network-diagnostics"}'
```

6. View the latency between the source pod and the openshift api service for the last 5 minutes:

```
$ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
promtool query instant http://localhost:9090 \
'{component="openshift-network-diagnostics"}'
```

### 18.3.7. Checking OVN-Kubernetes network traffic with OVS sampling using the CLI



## IMPORTANT

Checking OVN-Kubernetes network traffic with OVS sampling 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](#).

OVN-Kubernetes network traffic can be viewed with OVS sampling via the CLI for the following network APIs:

- **NetworkPolicy**
- **AdminNetworkPolicy**
- **BaselineNetworkPolicy**
- **UserDefinedNetwork** isolation
- **EgressFirewall**
- Multicast ACLs.

Scripts for these networking events are found in the **/usr/bin/ovnkube-observ** path of each OVN-Kubernetes node.

Although both the Network Observability Operator and checking OVN-Kubernetes network traffic with OVS sampling are good for debuggability, the Network Observability Operator is intended for observing network events. Alternatively, checking OVN-Kubernetes network traffic with OVS sampling using the CLI is intended to help with packet tracing; it can also be used while the Network Observability Operator is installed, however that is not a requirement.

Administrators can add the **--add-ovs-collect** option to view network traffic across the node, or pass in additional flags to filter result for specific pods. Additional flags can be found in the "OVN-Kubernetes network traffic with OVS sampling flags" section.

Use the following procedure to view OVN-Kubernetes network traffic using the CLI.

### Prerequisites

- You are logged in to the cluster as a user with **cluster-admin** privileges.
- You have created a source pod and a destination pod and ran traffic between them.
- You have created at least one of the following network APIs: **NetworkPolicy**, **AdminNetworkPolicy**, **BaselineNetworkPolicy**, **UserDefinedNetwork** isolation, multicast, or egress firewalls.

### Procedure

1. To enable the **OVNObservability** with OVS sampling feature, enable **TechPreviewNoUpgrade** feature set in the **FeatureGate** CR named **cluster** by entering the following command:

—

```
$ oc patch --type=merge --patch '{"spec": {"featureSet": "TechPreviewNoUpgrade"}}' featuregate/cluster
```

### Example output

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

2. Confirm that the **OVNObservability** feature is enabled by entering the following command:

```
$ oc get featuregate cluster -o yaml
```

### Example output

```
featureGates:  
# ...  
enabled:  
- name: OVNObservability
```

3. Obtain a list of the pods inside of the namespace in which you have created one of the relevant network APIs by entering the following command. Note the **NODE** name of the pods, as they are used in the following step.

```
$ oc get pods -n <namespace> -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE		READINESS GATES				
destination-pod	1/1	Running	0	53s	10.131.0.23	ci-1n-1gqp7b2-72292-bb9dv-
worker-a-gtmpc	<none>	<none>				
source-pod	1/1	Running	0	56s	10.131.0.22	ci-1n-1gqp7b2-72292-bb9dv-
worker-a-gtmpc	<none>	<none>				

4. Obtain a list of OVN-Kubernetes pods and locate the pod that shares the same **NODE** as the pods from the previous step by entering the following command:

```
$ oc get pods -n openshift-ovn-kubernetes -o wide
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE		READINESS GATES				
ovnkube-node-jzn5b	8/8	Running	1 (34m ago)	37m	10.0.128.2	ci-1n-1gqp7b2-72292-bb9dv-worker-a-gtmpc <none>
...						

5. Open a bash shell inside of the **ovnkube-node** pod by entering the following command:

```
$ oc exec -it <pod_name> -n openshift-ovn-kubernetes -- bash
```

6. While inside of the **ovnkube-node** pod, you can run the **ovnkube-observ -add-ovs-collector** script to show network events using the OVS collector. For example:

```
# /usr/bin/ovnkube-observ -add-ovs-collector
```

#### Example output

```
...
2024/12/02 19:41:41.327584 OVN-K message: Allowed by default allow from local node
policy, direction ingress
2024/12/02 19:41:41.327593 src=10.131.0.2, dst=10.131.0.6

2024/12/02 19:41:41.327692 OVN-K message: Allowed by default allow from local node
policy, direction ingress
2024/12/02 19:41:41.327715 src=10.131.0.6, dst=10.131.0.2
...
...
```

7. You can filter the content by type, such as source pods, by entering the following command with the **-filter-src-ip** flag and your pod's IP address. For example:

```
# /usr/bin/ovnkube-observ -add-ovs-collector -filter-src-ip <pod_ip_address>
```

#### Example output

```
...
Found group packets, id 14
2024/12/10 16:27:12.456473 OVN-K message: Allowed by admin network policy allow-
egress-group1, direction Egress
2024/12/10 16:27:12.456570 src=10.131.0.22, dst=10.131.0.23

2024/12/10 16:27:14.484421 OVN-K message: Allowed by admin network policy allow-
egress-group1, direction Egress
2024/12/10 16:27:14.484428 src=10.131.0.22, dst=10.131.0.23

2024/12/10 16:27:12.457222 OVN-K message: Allowed by network policy test:allow-ingress-
from-specific-pod, direction Ingress
2024/12/10 16:27:12.457228 src=10.131.0.22, dst=10.131.0.23

2024/12/10 16:27:12.457288 OVN-K message: Allowed by network policy test:allow-ingress-
from-specific-pod, direction Ingress
2024/12/10 16:27:12.457299 src=10.131.0.22, dst=10.131.0.23
...
...
```

For a full list of flags that can be passed in with **/usr/bin/ovnkube-observ**, see "OVN-Kubernetes network traffic with OVS sampling flags".

### 18.3.7.1. OVN-Kubernetes network traffic with OVS sampling flags

The following flags are available to view OVN-Kubernetes network traffic by using the CLI. Append these flags to the following syntax in your terminal after you have opened a bash shell inside of the **ovnkube-node** pod:

#### Command syntax

```
# /usr/bin/ovnkube-observ <flag>
```

Flag	Description
<b>-h</b>	Returns a complete list flags that can be used with the <b>usr/bin/ovnkube-observ</b> command.
<b>-add-ovs-collector</b>	Add OVS collector to enable sampling. Use with caution. Make sure no one else is using observability.
<b>-enable-enrichment</b>	Enrich samples with NBDB data. Defaults to <b>true</b> .
<b>-filter-dst-ip</b>	Filter only packets to a given destination IP.
<b>-filter-src-ip</b>	Filters only packets from a given source IP.
<b>-log-cookie</b>	Print raw sample cookie with psample group_id.
<b>-output-file</b>	Output file to write the samples to.
<b>-print-full-packet</b>	Print full received packet. When false, only source and destination IPs are printed with every sample.

### 18.3.8. Additional resources

- [Gathering data about your cluster for Red Hat Support](#)
- [Implementation of connection health checks](#)
- [Verifying network connectivity for an endpoint](#)

## 18.4. TRACING OPENFLOW WITH OVNKUBE-TRACE

OVN and OVS traffic flows can be simulated in a single utility called **ovnkube-trace**. The **ovnkube-trace** utility runs **ovn-trace**, **ovs-appctl ofproto/trace** and **ovn-detrace** and correlates that information in a single output.

You can execute the **ovnkube-trace** binary from a dedicated container. For releases after OpenShift Container Platform 4.7, you can also copy the binary to a local host and execute it from that host.

### 18.4.1. Installing the ovnkube-trace on local host

The **ovnkube-trace** tool traces packet simulations for arbitrary UDP or TCP traffic between points in an OVN-Kubernetes driven OpenShift Container Platform cluster. Copy the **ovnkube-trace** binary to your local host making it available to run against the cluster.

#### Prerequisites

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

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

## Procedure

1. Create a pod variable by using the following command:

```
$ POD=$(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-control-plane -o name | head -1 | awk -F '/' '{print $NF}')
```

2. Run the following command on your local host to copy the binary from the **ovnkube-control-plane** pods:

```
$ oc cp -n openshift-ovn-kubernetes $POD:/usr/bin/ovnkube-trace -c ovnkube-cluster-manager ovnkube-trace
```



### NOTE

If you are using Red Hat Enterprise Linux (RHEL) 8 to run the **ovnkube-trace** tool, you must copy the file **/usr/lib/rhel8/ovnkube-trace** to your local host.

3. Make **ovnkube-trace** executable by running the following command:

```
$ chmod +x ovnkube-trace
```

4. Display the options available with **ovnkube-trace** by running the following command:

```
$ ./ovnkube-trace -help
```

## Expected output

```
Usage of ./ovnkube-trace:
-addr-family string
    Address family (ip4 or ip6) to be used for tracing (default "ip4")
-dst string
    dest: destination pod name
-dst-ip string
    destination IP address (meant for tests to external targets)
-dst-namespace string
    k8s namespace of dest pod (default "default")
-dst-port string
    dst-port: destination port (default "80")
-kubeconfig string
    absolute path to the kubeconfig file
-loglevel string
    loglevel: klog level (default "0")
-ovn-config-namespace string
    namespace used by ovn-config itself
-service string
    service: destination service name
-skip-detrace
    skip ovn-detrace command
-src string
    src: source pod name
```

```

-src-namespace string
  k8s namespace of source pod (default "default")
-tcp
  use tcp transport protocol
-udp
  use udp transport protocol

```

The command-line arguments supported are familiar Kubernetes constructs, such as namespaces, pods, services so you do not need to find the MAC address, the IP address of the destination nodes, or the ICMP type.

The log levels are:

- 0 (minimal output)
- 2 (more verbose output showing results of trace commands)
- 5 (debug output)

### 18.4.2. Running ovnkube-trace

Run **ovn-trace** to simulate packet forwarding within an OVN logical network.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- You have installed **ovnkube-trace** on local host

#### Example: Testing that DNS resolution works from a deployed pod

This example illustrates how to test the DNS resolution from a deployed pod to the core DNS pod that runs in the cluster.

#### Procedure

1. Start a web service in the default namespace by entering the following command:

```
$ oc run web --namespace=default --image=quay.io/openshifttest/nginx --labels="app=web" -expose --port=80
```

2. List the pods running in the **openshift-dns** namespace:

```
oc get pods -n openshift-dns
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
dns-default-8s42x	2/2	Running	0	5h8m
dns-default-mdw6r	2/2	Running	0	4h58m
dns-default-p8t5h	2/2	Running	0	4h58m
dns-default-rl6nk	2/2	Running	0	5h8m
dns-default-xbgqx	2/2	Running	0	5h8m

```

dns-default-zv8f6  2/2  Running  0      4h58m
node-resolver-62jjb 1/1  Running  0      5h8m
node-resolver-8z4cj 1/1  Running  0      4h59m
node-resolver-bq244 1/1  Running  0      5h8m
node-resolver-hc58n 1/1  Running  0      4h59m
node-resolver-lm6z4 1/1  Running  0      5h8m
node-resolver-zfx5k 1/1  Running  0      5h

```

3. Run the following **ovnkube-trace** command to verify DNS resolution is working:

```

$ ./ovnkube-trace \
-src-namespace default \ ①
-src web \ ②
-dst-namespace openshift-dns \ ③
-dst dns-default-p8t5h \ ④
-udp -dst-port 53 \ ⑤
-loglevel 0 ⑥

```

- ① Namespace of the source pod
- ② Source pod name
- ③ Namespace of destination pod
- ④ Destination pod name
- ⑤ Use the **udp** transport protocol. Port 53 is the port the DNS service uses.
- ⑥ Set the log level to 0 (0 is minimal and 5 is debug)

#### Example output if the src&dst pod lands on the same node

```

ovn-trace source pod to destination pod indicates success from web to dns-default-p8t5h
ovn-trace destination pod to source pod indicates success from dns-default-p8t5h to web
ovs-appctl ofproto/trace source pod to destination pod indicates success from web to dns-
default-p8t5h
ovs-appctl ofproto/trace destination pod to source pod indicates success from dns-default-
p8t5h to web
ovn-detrace source pod to destination pod indicates success from web to dns-default-p8t5h
ovn-detrace destination pod to source pod indicates success from dns-default-p8t5h to web

```

#### Example output if the src&dst pod lands on a different node

```

ovn-trace source pod to destination pod indicates success from web to dns-default-8s42x
ovn-trace (remote) source pod to destination pod indicates success from web to dns-default-
8s42x
ovn-trace destination pod to source pod indicates success from dns-default-8s42x to web
ovn-trace (remote) destination pod to source pod indicates success from dns-default-8s42x to
web
ovs-appctl ofproto/trace source pod to destination pod indicates success from web to dns-
default-8s42x
ovs-appctl ofproto/trace destination pod to source pod indicates success from dns-default-

```

8s42x to web

ovn-detrace source pod to destination pod indicates success from web to dns-default-8s42x  
ovn-detrace destination pod to source pod indicates success from dns-default-8s42x to web

The output indicates success from the deployed pod to the DNS port and also indicates that it is successful going back in the other direction. So you know bi-directional traffic is supported on UDP port 53 if my web pod wants to do dns resolution from core DNS.

If for example that did not work and you wanted to get the **ovn-trace**, the **ovs-appctl** of **proto/trace** and **ovn-detrace**, and more debug type information increase the log level to 2 and run the command again as follows:

```
$ ./ovnkube-trace \
-src-namespace default \
-src web \
-dst-namespace openshift-dns \
-dst dns-default-467qw \
-udp -dst-port 53 \
-loglevel 2
```

The output from this increased log level is too much to list here. In a failure situation the output of this command shows which flow is dropping that traffic. For example an egress or ingress network policy may be configured on the cluster that does not allow that traffic.

### Example: Verifying by using debug output a configured default deny

This example illustrates how to identify by using the debug output that an ingress default deny policy blocks traffic.

#### Procedure

1. Create the following YAML that defines a **deny-by-default** policy to deny ingress from all pods in all namespaces. Save the YAML in the **deny-by-default.yaml** file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: default
spec:
  podSelector: {}
  ingress: []
```

2. Apply the policy by entering the following command:

```
$ oc apply -f deny-by-default.yaml
```

#### Example output

```
networkpolicy.networking.k8s.io/deny-by-default created
```

3. Start a web service in the **default** namespace by entering the following command:

```
$ oc run web --namespace=default --image=quay.io/openshifttest/nginx --labels="app=web" -  
-expose --port=80
```

4. Run the following command to create the **prod** namespace:

```
$ oc create namespace prod
```

5. Run the following command to label the **prod** namespace:

```
$ oc label namespace/prod purpose=production
```

6. Run the following command to deploy an **alpine** image in the **prod** namespace and start a shell:

```
$ oc run test-6459 --namespace=prod --rm -i -t --image=alpine -- sh
```

7. Open another terminal session.

8. In this new terminal session run **ovn-trace** to verify the failure in communication between the source pod **test-6459** running in namespace **prod** and destination pod running in the **default** namespace:

```
$ ./ovnkube-trace \  
-src-namespace prod \  
-src test-6459 \  
-dst-namespace default \  
-dst web \  
-tcp -dst-port 80 \  
-loglevel 0
```

### Example output

```
ovn-trace source pod to destination pod indicates failure from test-6459 to web
```

9. Increase the log level to 2 to expose the reason for the failure by running the following command:

```
$ ./ovnkube-trace \  
-src-namespace prod \  
-src test-6459 \  
-dst-namespace default \  
-dst web \  
-tcp -dst-port 80 \  
-loglevel 2
```

### Example output

```
...  
-----  
3. Is_out_acl_hint (northd.c:7454): !ct.new && ct.est && !ct.rpl && ct_mark.blocked == 0,  
priority 4, uuid 12efc456  
    reg0[8] = 1;  
    reg0[10] = 1;
```

```

    next;
5. ls_out_acl_action (northd.c:7835): reg8[30..31] == 0, priority 500, uuid 69372c5d
    reg8[30..31] = 1;
    next(4);
5. ls_out_acl_action (northd.c:7835): reg8[30..31] == 1, priority 500, uuid 2fa0af89
    reg8[30..31] = 2;
    next(4);
4. ls_out_acl_eval (northd.c:7691): reg8[30..31] == 2 && reg0[10] == 1 && (outport ==
@a16982411286042166782_ingressDefaultDeny), priority 2000, uuid 447d0dab
    reg8[17] = 1;
    ct_commit { ct_mark.blocked = 1; }; ①
    next;
...

```

① Ingress traffic is blocked due to the default deny policy being in place.

10. Create a policy that allows traffic from all pods in a particular namespaces with a label **purpose=production**. Save the YAML in the **web-allow-prod.yaml** file:

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              purpose: production

```

11. Apply the policy by entering the following command:

```
$ oc apply -f web-allow-prod.yaml
```

12. Run **ovnkube-trace** to verify that traffic is now allowed by entering the following command:

```

$ ./ovnkube-trace \
-src-namespace prod \
-src test-6459 \
-dst-namespace default \
-dst web \
-tcp -dst-port 80 \
-loglevel 0

```

#### Expected output

```
ovn-trace source pod to destination pod indicates success from test-6459 to web
```

ovn-trace destination pod to source pod indicates success from web to test-6459  
 ovs-appctl ofproto/trace source pod to destination pod indicates success from test-6459 to web  
 ovs-appctl ofproto/trace destination pod to source pod indicates success from web to test-6459  
 ovn-detrace source pod to destination pod indicates success from test-6459 to web  
 ovn-detrace destination pod to source pod indicates success from web to test-6459

13. Run the following command in the shell that was opened in step six to connect nginx to the web-server:

```
wget -qO- --timeout=2 http://web.default
```

#### Expected output

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
body {
  width: 35em;
  margin: 0 auto;
  font-family: Tahoma, Verdana, Arial, sans-serif;
}
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

#### 18.4.3. Additional resources

- [Tracing Openflow with ovnkube-trace utility](#)
- [ovnkube-trace](#)

## 18.5. CONVERTING TO IPV4/IPV6 DUAL-STACK NETWORKING

As a cluster administrator, you can convert your IPv4 single-stack cluster to a dual-network cluster network that supports IPv4 and IPv6 address families. After converting to dual-stack networking, new and existing pods have dual-stack networking enabled.



## IMPORTANT

When using dual-stack networking where IPv6 is required, you cannot use IPv4-mapped IPv6 addresses, such as `::FFFF:198.51.100.1`.

### Additional resources

- For more information about platform-specific support for dual-stack networking, see [OVN-Kubernetes purpose](#)

#### 18.5.1. Converting to a dual-stack cluster network

As a cluster administrator, you can convert your single-stack cluster network to a dual-stack cluster network.



## IMPORTANT

After converting your cluster to use dual-stack networking, you must re-create any existing pods for them to receive IPv6 addresses, because only new pods are assigned IPv6 addresses.

Converting a single-stack cluster network to a dual-stack cluster network consists of creating patches and applying them to the network and infrastructure of the cluster. You can convert to a dual-stack cluster network for a cluster that runs on either installer-provisioned infrastructure or user-provisioned infrastructure.



## NOTE

Each patch operation that changes **clusterNetwork**, **serviceNetwork**, **apiServerInternalIPs**, and **ingressIP** objects triggers a restart of the cluster. Changing the **MachineNetworks** object does not cause a reboot of the cluster.

On installer-provisioned infrastructure only, if you need to add IPv6 virtual IPs (VIPs) for API and Ingress services to an existing dual-stack-configured cluster, you need to patch only the infrastructure and not the network for the cluster.



## IMPORTANT

If you already upgraded your cluster to OpenShift Container Platform 4.16 or later and you need to convert the single-stack cluster network to a dual-stack cluster network, you must specify an existing IPv4 **machineNetwork** network configuration from the **install-config.yaml** file for API and Ingress services in the YAML configuration patch file. This configuration ensures that IPv4 traffic exists in the same network interface as the default gateway.

### Example YAML configuration file with an added IPv4 address block for the machineNetwork network

```
- op: add
  path: /spec/platformSpec/baremetal/machineNetworks/- ①
  value: 192.168.1.0/24
  # ...
```

- ① Ensure that you specify an address block for the **machineNetwork** network where your machines operate. You must select both API and Ingress IP addresses for the machine network.

## Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- Your cluster uses the OVN-Kubernetes network plugin.
- The cluster nodes have IPv6 addresses.
- You have configured an IPv6-enabled router based on your infrastructure.

## Procedure

1. To specify IPv6 address blocks for cluster and service networks, create a YAML configuration patch file that has a similar configuration to the following example:

```
- op: add
  path: /spec/clusterNetwork/
  value: ①
    cidr: fd01::/48
    hostPrefix: 64
- op: add
  path: /spec/serviceNetwork/
  value: fd02::/112 ②
```

- ① Specify an object with the **cidr** and **hostPrefix** parameters. The host prefix must be **64** or greater. The IPv6 Classless Inter-Domain Routing (CIDR) prefix must be large enough to accommodate the specified host prefix.
- ② Specify an IPv6 CIDR with a prefix of **112**. Kubernetes uses only the lowest 16 bits. For a prefix of **112**, IP addresses are assigned from **112** to **128** bits.

- Patch the cluster network configuration by entering the following command in your CLI:

```
$ oc patch network.config.openshift.io cluster \①
--type='json' --patch-file <file>.yaml
```

- Where **file** specifies the name of your created YAML file.

### Example output

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

- On installer-provisioned infrastructure where you added IPv6 VIPs for API and Ingress services, complete the following steps:

- Specify IPv6 VIPs for API and Ingress services for your cluster. Create a YAML configuration patch file that has a similar configuration to the following example:

```
- op: add
path: /spec/platformSpec/baremetal/machineNetworks/- ①
value: fd2e:6f44:5dd8::64
- op: add
path: /spec/platformSpec/baremetal/apiServerInternalIPs/- ②
value: fd2e:6f44:5dd8::4
- op: add
path: /spec/platformSpec/baremetal/ingressIPs/
value: fd2e:6f44:5dd8::5
```

- Ensure that you specify an address block for the **machineNetwork** network where your machines operate. You must select both API and Ingress IP addresses for the machine network.
- Ensure that you specify each file path according to your platform. The example demonstrates a file path on a bare-metal platform.

- Patch the infrastructure by entering the following command in your CLI:

```
$ oc patch infrastructure cluster \
--type='json' --patch-file <file>.yaml
```

Where:

**<file>**

Specifies the name of your created YAML file.

### Example output

```
infrastructure/cluster patched
```

## Verification

- Show the cluster network configuration by entering the following command in your CLI:

```
$ oc describe network
```

- Verify the successful installation of the patch on the network configuration by checking that the cluster network configuration recognizes the IPv6 address blocks that you specified in the YAML file.

#### Example output

```
# ...
Status:
Cluster Network:
  Cidr:          10.128.0.0/14
  Host Prefix:   23
  Cidr:          fd01::/48
  Host Prefix:   64
Cluster Network MTU: 1400
Network Type:    OVNKubernetes
Service Network:
  172.30.0.0/16
  fd02::/112
# ...
```

- Complete the following additional tasks for a cluster that runs on installer-provisioned infrastructure:

- Show the cluster infrastructure configuration by entering the following command in your CLI:

```
$ oc describe infrastructure
```

- Verify the successful installation of the patch on the cluster infrastructure by checking that the infrastructure recognizes the IPv6 address blocks that you specified in the YAML file.

#### Example output

```
# ...
spec:
# ...
platformSpec:
baremetal:
  apiServerInternalIPs:
    - 192.168.123.5
    - fd2e:6f44:5dd8::4
  ingressIPs:
    - 192.168.123.10
    - fd2e:6f44:5dd8::5
status:
# ...
platformStatus:
baremetal:
  apiServerInternalIP: 192.168.123.5
  apiServerInternalIPs:
    - 192.168.123.5
    - fd2e:6f44:5dd8::4
```

```
ingressIP: 192.168.123.10
ingressIPs:
- 192.168.123.10
- fd2e:6f44:5dd8::5
# ...
```

## 18.5.2. Converting to a single-stack cluster network

As a cluster administrator, you can convert your dual-stack cluster network to a single-stack cluster network.



### IMPORTANT

If you originally converted your IPv4 single-stack cluster network to a dual-stack cluster, you can convert only back to the IPv4 single-stack cluster and not an IPv6 single-stack cluster network. The same restriction applies for converting back to an IPv6 single-stack cluster network.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in to the cluster with a user with **cluster-admin** privileges.
- Your cluster uses the OVN-Kubernetes network plugin.
- The cluster nodes have IPv6 addresses.
- You have enabled dual-stack networking.

#### Procedure

1. Edit the **networks.config.openshift.io** custom resource (CR) by running the following command:
 

```
$ oc edit networks.config.openshift.io
```
2. Remove the IPv4 or IPv6 configuration that you added to the **cidr** and the **hostPrefix** parameters from completing the "Converting to a dual-stack cluster network" procedure steps.

## 18.6. CONFIGURING OVN-KUBERNETES INTERNAL IP ADDRESS SUBNETS

As a cluster administrator, you can change the IP address ranges that the OVN-Kubernetes network plugin uses for the join and transit subnets.

### 18.6.1. Configuring the OVN-Kubernetes join subnet

You can change the join subnet used by OVN-Kubernetes to avoid conflicting with any existing subnets already in use in your environment.

#### 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

- To change the OVN-Kubernetes join subnet, enter the following command:

```
$ oc patch network.operator.openshift.io cluster --type='merge' \
-p='{"spec":{"defaultNetwork":{"ovnKubernetesConfig": \
{"ipv4":{"internalJoinSubnet": "<join_subnet>"}, \
"ipv6":{"internalJoinSubnet": "<join_subnet>"}}}}'
```

where:

### <join\_subnet>

Specifies an IP address subnet for internal use by OVN-Kubernetes. The subnet must be larger than the number of nodes in the cluster and it must be large enough to accommodate one IP address per node in the cluster. This subnet cannot overlap with any other subnets used by OpenShift Container Platform or on the host itself. The default value for IPv4 is **100.64.0.0/16** and the default value for IPv6 is **fd98::/64**.

## Example output

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

## Verification

- To confirm that the configuration is active, enter the following command:

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

The command operation can take up to 30 minutes for this change to take effect.

## Example output

```
{
  "ovnKubernetesConfig": {
    "ipv4": {
      "internalJoinSubnet": "100.64.1.0/16"
    },
    "type": "OVNKubernetes"
  }
}
```

## 18.6.2. Configuring the OVN-Kubernetes masquerade subnet as a post-installation operation

You can change the masquerade subnet used by OVN-Kubernetes as a post-installation operation to avoid conflicts with any existing subnets that are already in use in your environment.

## Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster as a user with **cluster-admin** privileges.

## Procedure

- Change your cluster's masquerade subnet:
  - For dualstack clusters using IPv6, run the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge -p '{"spec": {"defaultNetwork":{"ovnKubernetesConfig":{"gatewayConfig":{"ipv4": {"internalMasqueradeSubnet": "<ipv4_masquerade_subnet>"}, "ipv6": {"internalMasqueradeSubnet": "<ipv6_masquerade_subnet>"}}}}}'
```

where:

### **ipv4\_masquerade\_subnet**

Specifies an IP address to be used as the IPv4 masquerade subnet. This range cannot overlap with any other subnets used by OpenShift Container Platform or on the host itself. In versions of OpenShift Container Platform earlier than 4.17, the default value for IPv4 was **169.254.169.0/29**, and clusters that were upgraded to version 4.17 maintain this value. For new clusters starting from version 4.17, the default value is **169.254.0.0/17**.

### **ipv6\_masquerade\_subnet**

Specifies an IP address to be used as the IPv6 masquerade subnet. This range cannot overlap with any other subnets used by OpenShift Container Platform or on the host itself. The default value for IPv6 is **fd69::/125**.

- For clusters using IPv4, run the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge -p '{"spec": {"defaultNetwork":{"ovnKubernetesConfig":{"gatewayConfig":{"ipv4": {"internalMasqueradeSubnet": "<ipv4_masquerade_subnet>"}}}}}'
```

where:

**ipv4\_masquerade\_subnet**: Specifies an IP address to be used as the IPv4 masquerade subnet. This range cannot overlap with any other subnets used by OpenShift Container Platform or on the host itself. In versions of OpenShift Container Platform earlier than 4.17, the default value for IPv4 was **169.254.169.0/29**, and clusters that were upgraded to version 4.17 maintain this value. For new clusters starting from version 4.17, the default value is **169.254.0.0/17**.

### 18.6.3. Configuring the OVN-Kubernetes transit subnet

You can change the transit subnet used by OVN-Kubernetes to avoid conflicting with any existing subnets already in use in your environment.

## 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

- To change the OVN-Kubernetes transit subnet, enter the following command:

```
$ oc patch network.operator.openshift.io cluster --type='merge' \
-p='{"spec":{"defaultNetwork":{"ovnKubernetesConfig": \
{"ipv4":{"internalTransitSwitchSubnet": "<transit_subnet>"}, \
"ipv6":{"internalTransitSwitchSubnet": "<transit_subnet>"}}}}'
```

where:

### <transit\_subnet>

Specifies an IP address subnet for the distributed transit switch that enables east-west traffic. This subnet cannot overlap with any other subnets used by OVN-Kubernetes or on the host itself. The default value for IPv4 is **100.88.0.0/16** and the default value for IPv6 is **fd97::/64**.

## Example output

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

## Verification

- To confirm that the configuration is active, enter the following command:

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

It can take up to 30 minutes for this change to take effect.

## Example output

```
{
  "ovnKubernetesConfig": {
    "ipv4": {
      "internalTransitSwitchSubnet": "100.88.1.0/16"
    },
    "type": "OVNKubernetes"
  }
}
```

## 18.7. CONFIGURING GATEWAY MODE

As a cluster administrator you can configure the **gatewayConfig** object to manage how external traffic leaves the cluster. You do so by setting the **routingViaHost** spec to **true** for local mode or **false** for shared mode.

In local gateway mode, traffic is routed through the host and is consequently applied to the routing table of the host. In shared gateway mode, traffic is not routed through the host. Instead, traffic the Open vSwitch (OVS) outputs traffic directly to the node IP interface.

### 18.7.1. Setting local and shared gateway modes

As a cluster administrator you can configure the gateway mode using the **gatewayConfig** spec in the Cluster Network Operator. The following procedure can be used to set the **routingViaHost** field to **true** for local mode or **false** for shared mode.

You can follow the optional step 4 to enable IP forwarding alongside local gateway mode if you need the host network of the node to act as a router for traffic not related to OVN-Kubernetes. For example, possible use cases for combining local gateway mode with IP forwarding include:

- Configuring all pod egress traffic to be forwarded via the node's IP
- Integrating OVN-Kubernetes CNI with external network address translation (NAT) devices
- Configuring OVN-Kubernetes CNI to use a kernel routing table

#### Prerequisites

- You are logged in as a user with admin privileges.

#### Procedure

1. Back up the existing network configuration by running the following command:

```
$ oc get network.operator cluster -o yaml > network-config-backup.yaml
```

2. Set the **routingViaHost** parameter to **true** for local gateway mode by running the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"gatewayConfig":{"routingViaHost": true}}}}}'
```

3. Verify that local gateway mode has been set by running the following command:

```
$ oc get networks.operator.openshift.io cluster -o yaml | grep -A 5 "gatewayConfig"
```

#### Example output

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
# ...
gatewayConfig:
  ipv4: {}
  ipv6: {}
  routingViaHost: true 1
  genevePort: 6081
  ipsecConfig:
# ...
```

- 1** A value of **true** sets local gateway mode and a value of **false** sets shared gateway mode. In local gateway mode, traffic is routed through the host. In shared gateway mode, traffic is not routed through the host.

4. Optional: Enable IP forwarding globally by running the following command:

```
$ oc patch network.operator cluster --type=merge -p '{"spec":{"defaultNetwork": "ovnKubernetesConfig":{"gatewayConfig":{"ipForwarding": "Global"}}}}'
```

- a. Verify that the **ipForwarding** spec has been set to **Global** by running the following command:

```
$ oc get networks.operator.openshift.io cluster -o yaml | grep -A 5 "gatewayConfig"
```

### Example output

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
# ...
gatewayConfig:
  ipForwarding: Global
  ipv4: {}
  ipv6: {}
  routingViaHost: true
  genevePort: 6081
# ...
```

## 18.8. CONFIGURE AN EXTERNAL GATEWAY ON THE DEFAULT NETWORK

As a cluster administrator, you can configure an external gateway on the default network.

This feature offers the following benefits:

- Granular control over egress traffic on a per-namespace basis
- Flexible configuration of static and dynamic external gateway IP addresses
- Support for both IPv4 and IPv6 address families

### 18.8.1. Prerequisites

- Your cluster uses the OVN-Kubernetes network plugin.
- Your infrastructure is configured to route traffic from the secondary external gateway.

### 18.8.2. How OpenShift Container Platform determines the external gateway IP address

You configure a secondary external gateway with the **AdminPolicyBasedExternalRoute** custom resource (CR) from the **k8s.ovn.org** API group. The CR supports static and dynamic approaches to specifying an external gateway's IP address.

Each namespace that a **AdminPolicyBasedExternalRoute** CR targets cannot be selected by any other **AdminPolicyBasedExternalRoute** CR. A namespace cannot have concurrent secondary external gateways.

Changes to policies are isolated in the controller. If a policy fails to apply, changes to other policies do not trigger a retry of other policies. Policies are only re-evaluated, applying any differences that might have occurred by the change, when updates to the policy itself or related objects to the policy such as target namespaces, pod gateways, or namespaces hosting them from dynamic hops are made.

### Static assignment

You specify an IP address directly.

### Dynamic assignment

You specify an IP address indirectly, with namespace and pod selectors, and an optional network attachment definition.

- If the name of a network attachment definition is provided, the external gateway IP address of the network attachment is used.
- If the name of a network attachment definition is not provided, the external gateway IP address for the pod itself is used. However, this approach works only if the pod is configured with **hostNetwork** set to **true**.

### 18.8.3. AdminPolicyBasedExternalRoute object configuration

You can define an **AdminPolicyBasedExternalRoute** object, which is cluster scoped, with the following properties. A namespace can be selected by only one **AdminPolicyBasedExternalRoute** CR at a time.

Table 18.3. **AdminPolicyBasedExternalRoute** object

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	Specifies the name of the <b>AdminPolicyBasedExternalRoute</b> object.

Field	Type	Description
<b>spec.from</b>	<b>string</b>	<p>Specifies a namespace selector that the routing policies apply to. Only <b>namespaceSelector</b> is supported for external traffic. For example:</p> <pre>from:   namespaceSelector:     matchLabels:       kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059</pre> <p>A namespace can only be targeted by one <b>AdminPolicyBasedExternalRoute</b> CR. If a namespace is selected by more than one <b>AdminPolicyBasedExternalRoute</b> CR, a <b>failed</b> error status occurs on the second and subsequent CRs that target the same namespace. To apply updates, you must change the policy itself or related objects to the policy such as target namespaces, pod gateways, or namespaces hosting them from dynamic hops in order for the policy to be re-evaluated and your changes to be applied.</p>
<b>spec.nextHops</b>	<b>object</b>	<p>Specifies the destinations where the packets are forwarded to. Must be either or both of <b>static</b> and <b>dynamic</b>. You must have at least one next hop defined.</p>

Table 18.4. **nextHops** object

Field	Type	Description
<b>static</b>	<b>array</b>	Specifies an array of static IP addresses.
<b>dynamic</b>	<b>array</b>	Specifies an array of pod selectors corresponding to pods configured with a network attachment definition to use as the external gateway target.

Table 18.5. **nextHops.static** object

Field	Type	Description
<b>ip</b>	<b>string</b>	Specifies either an IPv4 or IPv6 address of the next destination hop.
<b>bfdEnabled</b>	<b>boolean</b>	Optional: Specifies whether Bi-Directional Forwarding Detection (BFD) is supported by the network. The default value is <b>false</b> .

Table 18.6. `nextHops.dynamic` object

Field	Type	Description
<code>podSelector</code>	<code>string</code>	Specifies a [set-based] ( <a href="https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#set-based-requirement">https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#set-based-requirement</a> ) label selector to filter the pods in the namespace that match this network configuration.
<code>namespaceSelector</code>	<code>string</code>	Specifies a <b>set-based</b> selector to filter the namespaces that the <code>podSelector</code> applies to. You must specify a value for this field.
<code>bfdEnabled</code>	<code>boolean</code>	Optional: Specifies whether Bi-Directional Forwarding Detection (BFD) is supported by the network. The default value is <b>false</b> .
<code>networkAttachmentName</code>	<code>string</code>	Optional: Specifies the name of a network attachment definition. The name must match the list of logical networks associated with the pod. If this field is not specified, the host network of the pod is used. However, the pod must be configured as a host network pod to use the host network.

#### 18.8.3.1. Example secondary external gateway configurations

In the following example, the **AdminPolicyBasedExternalRoute** object configures two static IP addresses as external gateways for pods in namespaces with the `kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059` label.

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: default-route-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059
  nextHops:
    static:
      - ip: "172.18.0.8"
      - ip: "172.18.0.9"
```

In the following example, the **AdminPolicyBasedExternalRoute** object configures a dynamic external gateway. The IP addresses used for the external gateway are derived from the additional network attachments associated with each of the selected pods.

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: shadow-traffic-policy
```

```

spec:
from:
  namespaceSelector:
    matchLabels:
      externalTraffic: ""
nextHops:
  dynamic:
    - podSelector:
        matchLabels:
          gatewayPod: ""
  namespaceSelector:
    matchLabels:
      shadowTraffic: ""
    networkAttachmentName: shadow-gateway
  - podSelector:
      matchLabels:
        gigabyteGW: ""
  namespaceSelector:
    matchLabels:
      gatewayNamespace: ""
  networkAttachmentName: gateway

```

In the following example, the **AdminPolicyBasedExternalRoute** object configures both static and dynamic external gateways.

```

apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: multi-hop-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
        trafficType: "egress"
  nextHops:
    static:
      - ip: "172.18.0.8"
      - ip: "172.18.0.9"
    dynamic:
      - podSelector:
          matchLabels:
            gatewayPod: ""
      namespaceSelector:
        matchLabels:
          egressTraffic: ""
    networkAttachmentName: gigabyte

```

#### 18.8.4. Configure a secondary external gateway

You can configure an external gateway on the default network for a namespace in your cluster.

##### Prerequisites

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

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

## Procedure

1. Create a YAML file that contains an **AdminPolicyBasedExternalRoute** object.
2. To create an admin policy based external route, enter the following command:

```
$ oc create -f <file>.yaml
```

where:

**<file>**

Specifies the name of the YAML file that you created in the previous step.

## Example output

```
adminpolicybasedexternalroute.k8s.ovn.org/default-route-policy created
```

3. To confirm that the admin policy based external route was created, enter the following command:

```
$ oc describe apbexternalroute <name> | tail -n 6
```

where:

**<name>**

Specifies the name of the **AdminPolicyBasedExternalRoute** object.

## Example output

```
Status:  
Last Transition Time: 2023-04-24T15:09:01Z  
Messages:  
Configured external gateway IPs: 172.18.0.8  
Status: Success  
Events: <none>
```

### 18.8.5. Additional resources

- For more information about additional network attachments, see [Understanding multiple networks](#)

## 18.9. CONFIGURING AN EGRESS IP ADDRESS

As a cluster administrator, you can configure the OVN-Kubernetes Container Network Interface (CNI) network plugin to assign one or more egress IP addresses to a namespace, or to specific pods in a namespace.

### 18.9.1. Egress IP address architectural design and implementation

The OpenShift Container Platform egress IP address functionality allows you to ensure that the traffic from one or more pods in one or more namespaces has a consistent source IP address for services outside the cluster network.

For example, you might have a pod that periodically queries a database that is hosted on a server outside of your cluster. To enforce access requirements for the server, a packet filtering device is configured to allow traffic only from specific IP addresses. To ensure that you can reliably allow access to the server from only that specific pod, you can configure a specific egress IP address for the pod that makes the requests to the server.

An egress IP address assigned to a namespace is different from an egress router, which is used to send traffic to specific destinations.

In some cluster configurations, application pods and ingress router pods run on the same node. If you configure an egress IP address for an application project in this scenario, the IP address is not used when you send a request to a route from the application project.



### IMPORTANT

Egress IP addresses must not be configured in any Linux network configuration files, such as **ifcfg-eth0**.

#### 18.9.1.1. Platform support

Support for the egress IP address functionality on various platforms is summarized in the following table:

Platform	Supported
Bare metal	Yes
VMware vSphere	Yes
Red Hat OpenStack Platform (RHOSP)	Yes
Amazon Web Services (AWS)	Yes
Google Cloud Platform (GCP)	Yes
Microsoft Azure	Yes
IBM Z® and IBM® LinuxONE	Yes
IBM Z® and IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM	Yes
IBM Power®	Yes
Nutanix	Yes



## IMPORTANT

The assignment of egress IP addresses to control plane nodes with the EgressIP feature is not supported on a cluster provisioned on Amazon Web Services (AWS). ([BZ#2039656](#)).

### 18.9.1.2. Public cloud platform considerations

For clusters provisioned on public cloud infrastructure, there is a constraint on the absolute number of assignable IP addresses per node. The maximum number of assignable IP addresses per node, or the *IP capacity*, can be described in the following formula:

$$\text{IP capacity} = \text{public cloud default capacity} - \text{sum(current IP assignments)}$$

While the Egress IPs capability manages the IP address capacity per node, it is important to plan for this constraint in your deployments. For example, for a cluster installed on bare-metal infrastructure with 8 nodes you can configure 150 egress IP addresses. However, if a public cloud provider limits IP address capacity to 10 IP addresses per node, the total number of assignable IP addresses is only 80. To achieve the same IP address capacity in this example cloud provider, you would need to allocate 7 additional nodes.

To confirm the IP capacity and subnets for any node in your public cloud environment, you can enter the **oc get node <node\_name> -o yaml** command. The **cloud.network.openshift.io/egress-ipconfig** annotation includes capacity and subnet information for the node.

The annotation value is an array with a single object with fields that provide the following information for the primary network interface:

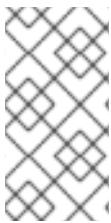
- **interface**: Specifies the interface ID on AWS and Azure and the interface name on GCP.
- **ifaddr**: Specifies the subnet mask for one or both IP address families.
- **capacity**: Specifies the IP address capacity for the node. On AWS, the IP address capacity is provided per IP address family. On Azure and GCP, the IP address capacity includes both IPv4 and IPv6 addresses.

Automatic attachment and detachment of egress IP addresses for traffic between nodes are available. This allows for traffic from many pods in namespaces to have a consistent source IP address to locations outside of the cluster. This also supports OpenShift SDN and OVN-Kubernetes, which is the default networking plugin in Red Hat OpenShift Networking in OpenShift Container Platform 4.18.



## NOTE

The RHOSP egress IP address feature creates a Neutron reservation port called **egressip-<IP address>**. Using the same RHOSP user as the one used for the OpenShift Container Platform cluster installation, you can assign a floating IP address to this reservation port to have a predictable SNAT address for egress traffic. When an egress IP address on an RHOSP network is moved from one node to another, because of a node failover, for example, the Neutron reservation port is removed and recreated. This means that the floating IP association is lost and you need to manually reassign the floating IP address to the new reservation port.

**NOTE**

When an RHOSP cluster administrator assigns a floating IP to the reservation port, OpenShift Container Platform cannot delete the reservation port. The **CloudPrivateIPConfig** object cannot perform delete and move operations until an RHOSP cluster administrator unassigns the floating IP from the reservation port.

The following examples illustrate the annotation from nodes on several public cloud providers. The annotations are indented for readability.

**Example `cloud.network.openshift.io/egress-ipconfig` annotation on AWS**

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface": "eni-078d267045138e436",
    "ifaddr": {"ipv4": "10.0.128.0/18"},
    "capacity": {"ipv4": 14, "ipv6": 15}
  }
]
```

**Example `cloud.network.openshift.io/egress-ipconfig` annotation on GCP**

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface": "nic0",
    "ifaddr": {"ipv4": "10.0.128.0/18"},
    "capacity": {"ip": 14}
  }
]
```

The following sections describe the IP address capacity for supported public cloud environments for use in your capacity calculation.

**18.9.1.2.1. Amazon Web Services (AWS) IP address capacity limits**

On AWS, constraints on IP address assignments depend on the instance type configured. For more information, see [IP addresses per network interface per instance type](#)

**18.9.1.2.2. Google Cloud Platform (GCP) IP address capacity limits**

On GCP, the networking model implements additional node IP addresses through IP address aliasing, rather than IP address assignments. However, IP address capacity maps directly to IP aliasing capacity.

The following capacity limits exist for IP aliasing assignment:

- Per node, the maximum number of IP aliases, both IPv4 and IPv6, is 100.
- Per VPC, the maximum number of IP aliases is unspecified, but OpenShift Container Platform scalability testing reveals the maximum to be approximately 15,000.

For more information, see [Per instance quotas](#) and [Alias IP ranges overview](#).

**18.9.1.2.3. Microsoft Azure IP address capacity limits**

On Azure, the following capacity limits exist for IP address assignment:

- Per NIC, the maximum number of assignable IP addresses, for both IPv4 and IPv6, is 256.
- Per virtual network, the maximum number of assigned IP addresses cannot exceed 65,536.

For more information, see [Networking limits](#).

### 18.9.1.3. Considerations for using an egress IP on additional network interfaces

In OpenShift Container Platform, egress IPs provide administrators a way to control network traffic. Egress IPs can be used with the **br-ex**, or primary, network interface, which is a Linux bridge interface associated with Open vSwitch, or they can be used with additional network interfaces.

You can inspect your network interface type by running the following command:

```
$ ip -details link show
```

The primary network interface is assigned a node IP address which also contains a subnet mask. Information for this node IP address can be retrieved from the Kubernetes node object for each node within your cluster by inspecting the **k8s.ovn.org/node-primary-ifaddr** annotation. In an IPv4 cluster, this annotation is similar to the following example: "**k8s.ovn.org/node-primary-ifaddr: {"ipv4":"192.168.111.23/24"}**".

If the egress IP is not within the subnet of the primary network interface subnet, you can use an egress IP on another Linux network interface that is not of the primary network interface type. By doing so, OpenShift Container Platform administrators are provided with a greater level of control over networking aspects such as routing, addressing, segmentation, and security policies. This feature provides users with the option to route workload traffic over specific network interfaces for purposes such as traffic segmentation or meeting specialized requirements.

If the egress IP is not within the subnet of the primary network interface, then the selection of another network interface for egress traffic might occur if they are present on a node.

You can determine which other network interfaces might support egress IPs by inspecting the **k8s.ovn.org/host-cidrs** Kubernetes node annotation. This annotation contains the addresses and subnet mask found for the primary network interface. It also contains additional network interface addresses and subnet mask information. These addresses and subnet masks are assigned to network interfaces that use the [longest prefix match routing](#) mechanism to determine which network interface supports the egress IP.



#### NOTE

OVN-Kubernetes provides a mechanism to control and direct outbound network traffic from specific namespaces and pods. This ensures that it exits the cluster through a particular network interface and with a specific egress IP address.

#### Requirements for assigning an egress IP to a network interface that is not the primary network interface

For users who want an egress IP and traffic to be routed over a particular interface that is not the primary network interface, the following conditions must be met:

- OpenShift Container Platform is installed on a bare metal cluster. This feature is disabled within cloud or hypervisor environments.

- Your OpenShift Container Platform pods are not configured as host-networked.
- If a network interface is removed or if the IP address and subnet mask which allows the egress IP to be hosted on the interface is removed, then the egress IP is reconfigured. Consequently, it could be assigned to another node and interface.
- IP forwarding must be enabled for the network interface. To enable IP forwarding, you can use the **oc edit network.operator** command and edit the object like the following example:

```
# ...
spec:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      gatewayConfig:
        ipForwarding: Global
# ...
```

#### 18.9.1.4. Assignment of egress IPs to pods

To assign one or more egress IPs to a namespace or specific pods in a namespace, the following conditions must be satisfied:

- At least one node in your cluster must have the **k8s.ovn.org/egress assignable: ""** label.
- An **EgressIP** object exists that defines one or more egress IP addresses to use as the source IP address for traffic leaving the cluster from pods in a namespace.



#### IMPORTANT

If you create **EgressIP** objects prior to labeling any nodes in your cluster for egress IP assignment, OpenShift Container Platform might assign every egress IP address to the first node with the **k8s.ovn.org/egress assignable: ""** label.

To ensure that egress IP addresses are widely distributed across nodes in the cluster, always apply the label to the nodes you intent to host the egress IP addresses before creating any **EgressIP** objects.

#### 18.9.1.5. Assignment of egress IPs to nodes

When creating an **EgressIP** object, the following conditions apply to nodes that are labeled with the **k8s.ovn.org/egress assignable: ""** label:

- An egress IP address is never assigned to more than one node at a time.
- An egress IP address is equally balanced between available nodes that can host the egress IP address.
- If the **spec.EgressIPs** array in an **EgressIP** object specifies more than one IP address, the following conditions apply:
  - No node will ever host more than one of the specified IP addresses.

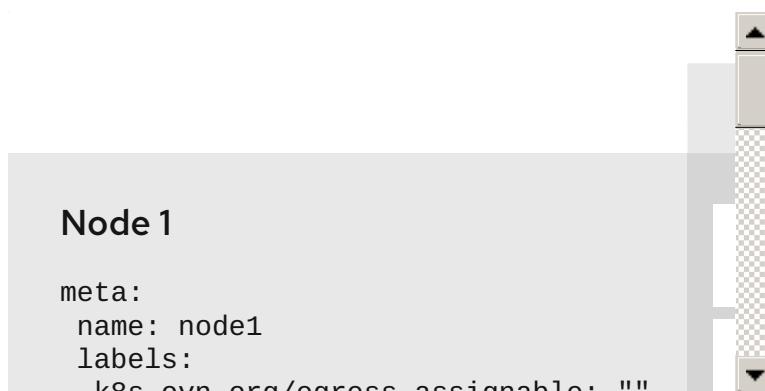
- Traffic is balanced roughly equally between the specified IP addresses for a given namespace.
- If a node becomes unavailable, any egress IP addresses assigned to it are automatically reassigned, subject to the previously described conditions.

When a pod matches the selector for multiple **EgressIP** objects, there is no guarantee which of the egress IP addresses that are specified in the **EgressIP** objects is assigned as the egress IP address for the pod.

Additionally, if an **EgressIP** object specifies multiple egress IP addresses, there is no guarantee which of the egress IP addresses might be used. For example, if a pod matches a selector for an **EgressIP** object with two egress IP addresses, **10.10.20.1** and **10.10.20.2**, either might be used for each TCP connection or UDP conversation.

#### 18.9.1.6. Architectural diagram of an egress IP address configuration

The following diagram depicts an egress IP address configuration. The diagram describes four pods in two different namespaces running on three nodes in a cluster. The nodes are assigned IP addresses from the **192.168.126.0/18** CIDR block on the host network.



Both Node 1 and Node 3 are labeled with **k8s.ovn.org/egress-assignable: ""** and thus available for the assignment of egress IP addresses.

The dashed lines in the diagram depict the traffic flow from pod1, pod2, and pod3 traveling through the pod network to egress the cluster from Node 1 and Node 3. When an external service receives traffic from any of the pods selected by the example **EgressIP** object, the source IP address is either **192.168.126.10** or **192.168.126.102**. The traffic is balanced roughly equally between these two nodes.

The following resources from the diagram are illustrated in detail:

#### Namespace objects

The namespaces are defined in the following manifest:

#### Namespace objects

```

apiVersion: v1
kind: Namespace
metadata:
  name: namespace1
  labels:
    env: prod
---
apiVersion: v1

```

```

kind: Namespace
metadata:
  name: namespace2
  labels:
    env: prod

```

## EgressIP object

The following **EgressIP** object describes a configuration that selects all pods in any namespace with the **env** label set to **prod**. The egress IP addresses for the selected pods are **192.168.126.10** and **192.168.126.102**.

## EgressIP object

```

apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egressips-prod
spec:
  egressIPs:
    - 192.168.126.10
    - 192.168.126.102
  namespaceSelector:
    matchLabels:
      env: prod
status:
  items:
    - node: node1
      egressIP: 192.168.126.10
    - node: node3
      egressIP: 192.168.126.102

```

For the configuration in the previous example, OpenShift Container Platform assigns both egress IP addresses to the available nodes. The **status** field reflects whether and where the egress IP addresses are assigned.

### 18.9.2. EgressIP object

The following YAML describes the API for the **EgressIP** object. The scope of the object is cluster-wide; it is not created in a namespace.

```

apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: <name> ①
spec:
  egressIPs: ②
    - <ip_address>
  namespaceSelector: ③
  ...
  podSelector: ④
  ...

```

- 1 The name for the **EgressIPs** object.
- 2 An array of one or more IP addresses.
- 3 One or more selectors for the namespaces to associate the egress IP addresses with.
- 4 Optional: One or more selectors for pods in the specified namespaces to associate egress IP addresses with. Applying these selectors allows for the selection of a subset of pods within a namespace.

The following YAML describes the stanza for the namespace selector:

### Namespace selector stanza

```
namespaceSelector: ①
matchLabels:
  <label_name>: <label_value>
```

- 1 One or more matching rules for namespaces. If more than one match rule is provided, all matching namespaces are selected.

The following YAML describes the optional stanza for the pod selector:

### Pod selector stanza

```
podSelector: ①
matchLabels:
  <label_name>: <label_value>
```

- 1 Optional: One or more matching rules for pods in the namespaces that match the specified **namespaceSelector** rules. If specified, only pods that match are selected. Others pods in the namespace are not selected.

In the following example, the **EgressIP** object associates the **192.168.126.11** and **192.168.126.102** egress IP addresses with pods that have the **app** label set to **web** and are in the namespaces that have the **env** label set to **prod**:

### Example EgressIP object

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group1
spec:
  egressIPs:
    - 192.168.126.11
    - 192.168.126.102
  podSelector:
    matchLabels:
      app: web
```

```
namespaceSelector:
  matchLabels:
    env: prod
```

In the following example, the **EgressIP** object associates the **192.168.127.30** and **192.168.127.40** egress IP addresses with any pods that do not have the **environment** label set to **development**:

### Example EgressIP object

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group2
spec:
  egressIPs:
    - 192.168.127.30
    - 192.168.127.40
  namespaceSelector:
    matchExpressions:
      - key: environment
        operator: NotIn
        values:
          - development
```

### 18.9.3. The egressIPConfig object

As a feature of egress IP, the **reachabilityTotalTimeoutSeconds** parameter configures the EgressIP node reachability check total timeout in seconds. If the EgressIP node cannot be reached within this timeout, the node is declared down.

You can set a value for the **reachabilityTotalTimeoutSeconds** in the configuration file for the **egressIPConfig** object. Setting a large value might cause the EgressIP implementation to react slowly to node changes. The implementation reacts slowly for EgressIP nodes that have an issue and are unreachable.

If you omit the **reachabilityTotalTimeoutSeconds** parameter from the **egressIPConfig** object, the platform chooses a reasonable default value, which is subject to change over time. The current default is **1** second. A value of **0** disables the reachability check for the EgressIP node.

The following **egressIPConfig** object describes changing the **reachabilityTotalTimeoutSeconds** from the default **1** second probes to **5** second probes:

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      egressIPConfig: ①
        reachabilityTotalTimeoutSeconds: 5 ②
```

```
gatewayConfig:
  routingViaHost: false
  genevePort: 6081
```

- 1 The **egressIPConfig** holds the configurations for the options of the **EgressIP** object. By changing these configurations, you can extend the **EgressIP** object.
- 2 The value for **reachabilityTotalTimeoutSeconds** accepts integer values from **0** to **60**. A value of **0** disables the reachability check of the egressIP node. Setting a value from **1** to **60** corresponds to the timeout in seconds for a probe to send the reachability check to the node.

#### 18.9.4. Labeling a node to host egress IP addresses

You can apply the **k8s.ovn.org/egress-assignable=""** label to a node in your cluster so that OpenShift Container Platform can assign one or more egress IP addresses to the node.

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster as a cluster administrator.

##### Procedure

- To label a node so that it can host one or more egress IP addresses, enter the following command:

```
$ oc label nodes <node_name> k8s.ovn.org/egress-assignable="" 1
```

- 1 The name of the node to label.

##### TIP

You can alternatively apply the following YAML to add the label to a node:

```
apiVersion: v1
kind: Node
metadata:
  labels:
    k8s.ovn.org/egress-assignable: ""
  name: <node_name>
```

#### 18.9.5. Next steps

- [Assigning egress IPs](#)

#### 18.9.6. Additional resources

- [LabelSelector meta/v1](#)
- [LabelSelectorRequirement meta/v1](#)

## 18.10. ASSIGNING AN EGRESS IP ADDRESS

As a cluster administrator, you can assign an egress IP address for traffic leaving the cluster from a namespace or from specific pods in a namespace.

### 18.10.1. Assigning an egress IP address to a namespace

You can assign one or more egress IP addresses to a namespace or to specific pods in a namespace.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster as a cluster administrator.
- Configure at least one node to host an egress IP address.

#### Procedure

1. Create an **EgressIP** object:
  - a. Create a **<egressips\_name>.yaml** file where **<egressips\_name>** is the name of the object.
  - b. In the file that you created, define an **EgressIP** object, as in the following example:

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-project1
spec:
  egressIPs:
    - 192.168.127.10
    - 192.168.127.11
  namespaceSelector:
    matchLabels:
      env: qa
```

2. To create the object, enter the following command.

```
$ oc apply -f <egressips_name>.yaml ①
```

- 1 Replace **<egressips\_name>** with the name of the object.

#### Example output

```
egressips.k8s.ovn.org/<egressips_name> created
```

3. Optional: Store the **<egressips\_name>.yaml** file so that you can make changes later.
4. Add labels to the namespace that requires egress IP addresses. To add a label to the namespace of an **EgressIP** object defined in step 1, run the following command:

```
$ oc label ns <namespace> env=qa ①
```

- Replace **<namespace>** with the namespace that requires egress IP addresses.

## Verification

- To show all egress IPs that are in use in your cluster, enter the following command:

```
$ oc get egressip -o yaml
```



### NOTE

The command **oc get egressip** only returns one egress IP address regardless of how many are configured. This is not a bug and is a limitation of Kubernetes. As a workaround, you can pass in the **-o yaml** or **-o json** flags to return all egress IPs addresses in use.

### Example output

```
# ...
spec:
  egressIPs:
    - 192.168.127.10
    - 192.168.127.11
# ...
```

## 18.10.2. Additional resources

- [Configuring egress IP addresses](#)

## 18.11. CONFIGURING AN EGRESS SERVICE

As a cluster administrator, you can configure egress traffic for pods behind a load balancer service by using an egress service.



### IMPORTANT

Egress service 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](#).

You can use the **EgressService** custom resource (CR) to manage egress traffic in the following ways:

- Assign a load balancer service IP address as the source IP address for egress traffic for pods behind the load balancer service.  
Assigning the load balancer IP address as the source IP address in this context is useful to

present a single point of egress and ingress. For example, in some scenarios, an external system communicating with an application behind a load balancer service can expect the source and destination IP address for the application to be the same.



## NOTE

When you assign the load balancer service IP address to egress traffic for pods behind the service, OVN-Kubernetes restricts the ingress and egress point to a single node. This limits the load balancing of traffic that MetallLB typically provides.

- Assign the egress traffic for pods behind a load balancer to a different network than the default node network.

This is useful to assign the egress traffic for applications behind a load balancer to a different network than the default network. Typically, the different network is implemented by using a VRF instance associated with a network interface.

### 18.11.1. Egress service custom resource

Define the configuration for an egress service in an **EgressService** custom resource. The following YAML describes the fields for the configuration of an egress service:

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: <egress_service_name> ①
  namespace: <namespace> ②
spec:
  sourceIPBy: <egress_traffic_ip> ③
  nodeSelector: ④
    matchLabels:
      node-role.kubernetes.io/<role>: ""
  network: <egress_traffic_network> ⑤
```

- ① Specify the name for the egress service. The name of the **EgressService** resource must match the name of the load-balancer service that you want to modify.
- ② Specify the namespace for the egress service. The namespace for the **EgressService** must match the namespace of the load-balancer service that you want to modify. The egress service is namespace-scoped.
- ③ Specify the source IP address of egress traffic for pods behind a service. Valid values are **LoadBalancerIP** or **Network**. Use the **LoadBalancerIP** value to assign the **LoadBalancer** service ingress IP address as the source IP address for egress traffic. Specify **Network** to assign the network interface IP address as the source IP address for egress traffic.
- ④ Optional: If you use the **LoadBalancerIP** value for the **sourceIPBy** specification, a single node handles the **LoadBalancer** service traffic. Use the **nodeSelector** field to limit which node can be assigned this task. When a node is selected to handle the service traffic, OVN-Kubernetes labels the node in the following format: **egress-service.k8s.ovn.org/<svc-namespace>-<svc-name>: ""**. When the **nodeSelector** field is not specified, any node can manage the **LoadBalancer** service traffic.
- ⑤

Optional: Specify the routing table ID for egress traffic. Ensure that the value matches the **route-table-id** ID defined in the **NodeNetworkConfigurationPolicy** resource. If you do not include the

### Example egress service specification

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: test-egress-service
  namespace: test-namespace
spec:
  sourceIPBy: "LoadBalancerIP"
  nodeSelector:
    matchLabels:
      vrf: "true"
  network: "2"
```

### 18.11.2. Deploying an egress service

You can deploy an egress service to manage egress traffic for pods behind a **LoadBalancer** service.

The following example configures the egress traffic to have the same source IP address as the ingress IP address of the **LoadBalancer** service.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- You configured MetalLB **BGPPeer** resources.

#### Procedure

1. Create an **IPAddressPool** CR with the desired IP for the service:
  - a. Create a file, such as **ip-addr-pool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example-pool
  namespace: metallb-system
spec:
  addresses:
  - 172.19.0.100/32
```

- b. Apply the configuration for the IP address pool by running the following command:

```
$ oc apply -f ip-addr-pool.yaml
```

2. Create **Service** and **EgressService** CRs:

- a. Create a file, such as **service-egress-service.yaml**, with content like the following example:

```

apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
  annotations:
    metallb.io/address-pool: example-pool ①
spec:
  selector:
    app: example
  ports:
    - name: http
      protocol: TCP
      port: 8080
      targetPort: 8080
  type: LoadBalancer
---
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: example-service
  namespace: example-namespace
spec:
  sourceIPBy: "LoadBalancerIP" ②
  nodeSelector: ③
  matchLabels:
    node-role.kubernetes.io/worker: ""

```

- ① The **LoadBalancer** service uses the IP address assigned by MetalLB from the **example-pool** IP address pool.
- ② This example uses the **LoadBalancerIP** value to assign the ingress IP address of the **LoadBalancer** service as the source IP address of egress traffic.
- ③ When you specify the **LoadBalancerIP** value, a single node handles the **LoadBalancer** service's traffic. In this example, only nodes with the **worker** label can be selected to handle the traffic. When a node is selected, OVN-Kubernetes labels the node in the following format **egress-service.k8s.ovn.org/<svc-namespace>-<svc-name>: ""**.



#### NOTE

If you use the **sourceIPBy: "LoadBalancerIP"** setting, you must specify the load-balancer node in the **BGPAdvertisement** custom resource (CR).

- b. Apply the configuration for the service and egress service by running the following command:

```
$ oc apply -f service-egress-service.yaml
```

3. Create a **BGPAdvertisement** CR to advertise the service:

- a. Create a file, such as **service-bgp-advertisement.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta1
kind: BGPAAdvertisement
metadata:
  name: example-bgp-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - example-pool
  nodeSelectors:
  - matchLabels:
      egress-service.k8s.ovn.org/example-namespace-example-service: ""

```

- 1** In this example, the **EgressService** CR configures the source IP address for egress traffic to use the load-balancer service IP address. Therefore, you must specify the load-balancer node for return traffic to use the same return path for the traffic originating from the pod.

## Verification

- 1 Verify that you can access the application endpoint of the pods running behind the MetalLB service by running the following command:
 

```
$ curl <external_ip_address>:<port_number>
```

**1**

**1** Update the external IP address and port number to suit your application endpoint.
- 2 If you assigned the **LoadBalancer** service's ingress IP address as the source IP address for egress traffic, verify this configuration by using tools such as **tcpdump** to analyze packets received at the external client.

## Additional resources

- [Exposing a service through a network VRF](#)
- [Example: Network interface with a VRF instance node network configuration policy](#)
- [Managing symmetric routing with MetalLB](#)
- [About virtual routing and forwarding](#)

## 18.12. CONSIDERATIONS FOR THE USE OF AN EGRESS ROUTER POD

### 18.12.1. About an egress router pod

The OpenShift Container Platform egress router pod redirects traffic to a specified remote server from a private source IP address that is not used for any other purpose. An egress router pod can send network traffic to servers that are set up to allow access only from specific IP addresses.



## NOTE

The egress router pod is not intended for every outgoing connection. Creating large numbers of egress router pods can exceed the limits of your network hardware. For example, creating an egress router pod for every project or application could exceed the number of local MAC addresses that the network interface can handle before reverting to filtering MAC addresses in software.



## IMPORTANT

The egress router image is not compatible with Amazon AWS, Azure Cloud, or any other cloud platform that does not support layer 2 manipulations due to their incompatibility with macvlan traffic.

### 18.12.1.1. Egress router modes

In **redirect mode**, an egress router pod configures **iptables** rules to redirect traffic from its own IP address to one or more destination IP addresses. Client pods that need to use the reserved source IP address must be configured to access the service for the egress router rather than connecting directly to the destination IP. You can access the destination service and port from the application pod by using the **curl** command. For example:

```
$ curl <router_service_IP> <port>
```



## NOTE

The egress router CNI plugin supports redirect mode only. The egress router CNI plugin does not support HTTP proxy mode or DNS proxy mode.

### 18.12.1.2. Egress router pod implementation

The egress router implementation uses the egress router Container Network Interface (CNI) plugin. The plugin adds a secondary network interface to a pod.

An egress router is a pod that has two network interfaces. For example, the pod can have **eth0** and **net1** network interfaces. The **eth0** interface is on the cluster network and the pod continues to use the interface for ordinary cluster-related network traffic. The **net1** interface is on a secondary network and has an IP address and gateway for that network. Other pods in the OpenShift Container Platform cluster can access the egress router service and the service enables the pods to access external services. The egress router acts as a bridge between pods and an external system.

Traffic that leaves the egress router exits through a node, but the packets have the MAC address of the **net1** interface from the egress router pod.

When you add an egress router custom resource, the Cluster Network Operator creates the following objects:

- The network attachment definition for the **net1** secondary network interface of the pod.
- A deployment for the egress router.

If you delete an egress router custom resource, the Operator deletes the two objects in the preceding list that are associated with the egress router.

### 18.12.1.3. Deployment considerations

An egress router pod adds an additional IP address and MAC address to the primary network interface of the node. As a result, you might need to configure your hypervisor or cloud provider to allow the additional address.

#### Red Hat OpenStack Platform (RHOSP)

If you deploy OpenShift Container Platform on RHOSP, you must allow traffic from the IP and MAC addresses of the egress router pod on your OpenStack environment. If you do not allow the traffic, then [communication will fail](#):

```
$ openstack port set --allowed-address \
    ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

#### VMware vSphere

If you are using VMware vSphere, see the [VMware documentation for securing vSphere standard switches](#). View and change VMware vSphere default settings by selecting the host virtual switch from the vSphere Web Client.

Specifically, ensure that the following are enabled:

- [MAC Address Changes](#)
- [Forged Transits](#)
- [Promiscuous Mode Operation](#)

### 18.12.1.4. Failover configuration

To avoid downtime, the Cluster Network Operator deploys the egress router pod as a deployment resource. The deployment name is **egress-router-cni-deployment**. The pod that corresponds to the deployment has a label of **app=egress-router-cni**.

To create a new service for the deployment, use the **oc expose deployment/egress-router-cni-deployment --port <port\_number>** command or create a file like the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: app-egress
spec:
  ports:
    - name: tcp-8080
      protocol: TCP
      port: 8080
    - name: tcp-8443
      protocol: TCP
      port: 8443
    - name: udp-80
      protocol: UDP
      port: 80
  type: ClusterIP
  selector:
    app: egress-router-cni
```

## 18.12.2. Additional resources

- [Deploying an egress router in redirection mode](#)

## 18.13. DEPLOYING AN EGRESS ROUTER POD IN REDIRECT MODE

As a cluster administrator, you can deploy an egress router pod to redirect traffic to specified destination IP addresses from a reserved source IP address.

The egress router implementation uses the egress router Container Network Interface (CNI) plugin.

### 18.13.1. Egress router custom resource

Define the configuration for an egress router pod in an egress router custom resource. The following YAML describes the fields for the configuration of an egress router in redirect mode:

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: <egress_router_name>
  namespace: <namespace> ①
spec:
  addresses: [ ②
    {
      ip: "<egress_router>", ③
      gateway: "<egress_gateway>" ④
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [ ⑤
      {
        destinationIP: "<egress_destination>",
        port: <egress_router_port>,
        targetPort: <target_port>, ⑥
        protocol: <network_protocol> ⑦
      },
      ...
    ],
    fallbackIP: "<egress_destination>" ⑧
  }
}
```

- ① Optional: The **namespace** field specifies the namespace to create the egress router in. If you do not specify a value in the file or on the command line, the **default** namespace is used.
- ② The **addresses** field specifies the IP addresses to configure on the secondary network interface.
- ③ The **ip** field specifies the reserved source IP address and netmask from the physical network that the node is on to use with egress router pod. Use CIDR notation to specify the IP address and netmask.
- ④ The **gateway** field specifies the IP address of the network gateway.
- ⑤ Optional: The **redirectRules** field specifies a combination of egress destination IP address, egress router port, and protocol. Incoming connections to the egress router on the specified port and

router port, and protocol. Incoming connections to the egress router on the specified port and protocol are routed to the destination IP address.

- 6 Optional: The **targetPort** field specifies the network port on the destination IP address. If this field is not specified, traffic is routed to the same network port that it arrived on.
- 7 The **protocol** field supports TCP, UDP, or SCTP.
- 8 Optional: The **fallbackIP** field specifies a destination IP address. If you do not specify any redirect rules, the egress router sends all traffic to this fallback IP address. If you specify redirect rules, any connections to network ports that are not defined in the rules are sent by the egress router to this fallback IP address. If you do not specify this field, the egress router rejects connections to network ports that are not defined in the rules.

## Example egress router specification

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: egress-router-redirect
spec:
  networkInterface: {
    macvlan: {
      mode: "Bridge"
    }
  }
  addresses: [
    {
      ip: "192.168.12.99/24",
      gateway: "192.168.12.1"
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [
      {
        destinationIP: "10.0.0.99",
        port: 80,
        protocol: UDP
      },
      {
        destinationIP: "203.0.113.26",
        port: 8080,
        targetPort: 80,
        protocol: TCP
      },
      {
        destinationIP: "203.0.113.27",
        port: 8443,
        targetPort: 443,
        protocol: TCP
      }
    ]
  }
}
```

### 18.13.2. Deploying an egress router in redirect mode

You can deploy an egress router to redirect traffic from its own reserved source IP address to one or more destination IP addresses.

After you add an egress router, the client pods that need to use the reserved source IP address must be modified to connect to the egress router rather than connecting directly to the destination IP.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create an egress router definition.
2. To ensure that other pods can find the IP address of the egress router pod, create a service that uses the egress router, as in the following example:

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: web-app
      protocol: TCP
      port: 8080
      targetPort: 8080
      type: ClusterIP
  selector:
    app: egress-router-cni ①
```

- 1 Specify the label for the egress router. The value shown is added by the Cluster Network Operator and is not configurable.

After you create the service, your pods can connect to the service. The egress router pod redirects traffic to the corresponding port on the destination IP address. The connections originate from the reserved source IP address.

#### Verification

To verify that the Cluster Network Operator started the egress router, complete the following procedure:

- 1 View the network attachment definition that the Operator created for the egress router:

```
$ oc get network-attachment-definition egress-router-cni-nad
```

The name of the network attachment definition is not configurable.

#### Example output

NAME	AGE
egress-router-cni-nad	18m

- View the deployment for the egress router pod:

```
$ oc get deployment egress-router-cni-deployment
```

The name of the deployment is not configurable.

#### Example output

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
egress-router-cni-deployment	1/1	1	1	18m

- View the status of the egress router pod:

```
$ oc get pods -l app=egress-router-cni
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
egress-router-cni-deployment-575465c75c-qkq6m	1/1	Running	0	18m

- View the logs and the routing table for the egress router pod.

- Get the node name for the egress router pod:

```
$ POD_NODENAME=$(oc get pod -l app=egress-router-cni -o jsonpath=".items[0].spec.nodeName")
```

- Enter into a debug session on the target node. This step instantiates a debug pod called **<node\_name>-debug**:

```
$ oc debug node/$POD_NODENAME
```

- Set **/host** as the root directory within the debug shell. The debug pod mounts the root file system of the host in **/host** within the pod. By changing the root directory to **/host**, you can run binaries from the executable paths of the host:

```
# chroot /host
```

- From within the **chroot** environment console, display the egress router logs:

```
# cat /tmp/egress-router-log
```

#### Example output

```
2021-04-26T12:27:20Z [debug] Called CNI ADD
2021-04-26T12:27:20Z [debug] Gateway: 192.168.12.1
2021-04-26T12:27:20Z [debug] IP Source Addresses: [192.168.12.99/24]
2021-04-26T12:27:20Z [debug] IP Destinations: [80 UDP 10.0.0.99/30 8080 TCP
203.0.113.26/30 80 8443 TCP 203.0.113.27/30 443]
```

```

2021-04-26T12:27:20Z [debug] Created macvlan interface
2021-04-26T12:27:20Z [debug] Renamed macvlan to "net1"
2021-04-26T12:27:20Z [debug] Adding route to gateway 192.168.12.1 on macvlan interface
2021-04-26T12:27:20Z [debug] deleted default route {Ifindex: 3 Dst: <nil> Src: <nil> Gw: 10.128.10.1 Flags: [] Table: 254}
2021-04-26T12:27:20Z [debug] Added new default route with gateway 192.168.12.1
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p UDP --dport 80 -j DNAT --to-destination 10.0.0.99
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p TCP --dport 8080 -j DNAT --to-destination 203.0.113.26:80
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p TCP --dport 8443 -j DNAT --to-destination 203.0.113.27:443
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat -o net1 -j SNAT --to-source 192.168.12.99

```

The logging file location and logging level are not configurable when you start the egress router by creating an **EgressRouter** object as described in this procedure.

- From within the **chroot** environment console, get the container ID:

```
# crictl ps --name egress-router-cni-pod | awk '{print $1}'
```

#### Example output

```

CONTAINER
bac9fae69ddb6

```

- Determine the process ID of the container. In this example, the container ID is **bac9fae69ddb6**:

```
# crictl inspect -o yaml bac9fae69ddb6 | grep 'pid:' | awk '{print $2}'
```

#### Example output

```
68857
```

- Enter the network namespace of the container:

```
# nsenter -n -t 68857
```

- Display the routing table:

```
# ip route
```

In the following example output, the **net1** network interface is the default route. Traffic for the cluster network uses the **eth0** network interface. Traffic for the **192.168.12.0/24** network uses the **net1** network interface and originates from the reserved source IP address **192.168.12.99**. The pod routes all other traffic to the gateway at IP address **192.168.12.1**. Routing for the service network is not shown.

#### Example output

```

default via 192.168.12.1 dev net1
10.128.10.0/23 dev eth0 proto kernel scope link src 10.128.10.18

```

```
192.168.12.0/24 dev net1 proto kernel scope link src 192.168.12.99
192.168.12.1 dev net1
```

## 18.14. ENABLING MULTICAST FOR A PROJECT

### 18.14.1. About multicast

With IP multicast, data is broadcast to many IP addresses simultaneously.



#### IMPORTANT

- At this time, multicast is best used for low-bandwidth coordination or service discovery and not a high-bandwidth solution.
- By default, network policies affect all connections in a namespace. However, multicast is unaffected by network policies. If multicast is enabled in the same namespace as your network policies, it is always allowed, even if there is a **deny-all** network policy. Cluster administrators should consider the implications to the exemption of multicast from network policies before enabling it.

Multicast traffic between OpenShift Container Platform pods is disabled by default. If you are using the OVN-Kubernetes network plugin, you can enable multicast on a per-project basis.

### 18.14.2. Enabling multicast between pods

You can enable multicast between pods for your project.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

#### Procedure

- Run the following command to enable multicast for a project. Replace **<namespace>** with the namespace for the project you want to enable multicast for.

```
$ oc annotate namespace <namespace> \
  k8s.ovn.org/multicast-enabled=true
```

#### TIP

You can alternatively apply the following YAML to add the annotation:

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: "true"
```

## Verification

To verify that multicast is enabled for a project, complete the following procedure:

1. Change your current project to the project that you enabled multicast for. Replace <project> with the project name.

```
$ oc project <project>
```

2. Create a pod to act as a multicast receiver:

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
    - name: mlistener
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
    ports:
      - containerPort: 30102
        name: mlistener
        protocol: UDP
EOF
```

3. Create a pod to act as a multicast sender:

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. In a new terminal window or tab, start the multicast listener.

- a. Get the IP address for the Pod:

```
$ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
```

- b. Start the multicast listener by entering the following command:

```
$ oc exec mlistener -i -t -- \
socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
EXEC:hostname
```

5. Start the multicast transmitter.

- a. Get the pod network IP address range:

```
$ CIDR=$(oc get Network.config.openshift.io cluster \
-o jsonpath='{.status.clusterNetwork[0].cidr}')
```

- b. To send a multicast message, enter the following command:

```
$ oc exec msender -i -t -- \
/bin/bash -c "echo | socat STDIO UDP4-
DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
```

If multicast is working, the previous command returns the following output:

```
mlistener
```

## 18.15. DISABLING MULTICAST FOR A PROJECT

### 18.15.1. Disabling multicast between pods

You can disable multicast between pods for your project.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- You must log in to the cluster with a user that has the **cluster-admin** role.

#### Procedure

- Disable multicast by running the following command:

```
$ oc annotate namespace <namespace> \ ①
k8s.ovn.org/multicast-enabled-
```

- ① The **namespace** for the project you want to disable multicast for.

**TIP**

You can alternatively apply the following YAML to delete the annotation:

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: null
```

## 18.16. TRACKING NETWORK FLOWS

As a cluster administrator, you can collect information about pod network flows from your cluster to assist with the following areas:

- Monitor ingress and egress traffic on the pod network.
- Troubleshoot performance issues.
- Gather data for capacity planning and security audits.

When you enable the collection of the network flows, only the metadata about the traffic is collected. For example, packet data is not collected, but the protocol, source address, destination address, port numbers, number of bytes, and other packet-level information is collected.

The data is collected in one or more of the following record formats:

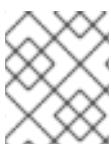
- NetFlow
- sFlow
- IPFIX

When you configure the Cluster Network Operator (CNO) with one or more collector IP addresses and port numbers, the Operator configures Open vSwitch (OVS) on each node to send the network flows records to each collector.

You can configure the Operator to send records to more than one type of network flow collector. For example, you can send records to NetFlow collectors and also send records to sFlow collectors.

When OVS sends data to the collectors, each type of collector receives identical records. For example, if you configure two NetFlow collectors, OVS on a node sends identical records to the two collectors. If you also configure two sFlow collectors, the two sFlow collectors receive identical records. However, each collector type has a unique record format.

Collecting the network flows data and sending the records to collectors affects performance. Nodes process packets at a slower rate. If the performance impact is too great, you can delete the destinations for collectors to disable collecting network flows data and restore performance.

**NOTE**

Enabling network flow collectors might have an impact on the overall performance of the cluster network.

### 18.16.1. Network object configuration for tracking network flows

The fields for configuring network flows collectors in the Cluster Network Operator (CNO) are shown in the following table:

**Table 18.7. Network flows configuration**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	The name of the CNO object. This name is always <b>cluster</b> .
<b>spec.exportNetworkFlows</b>	<b>object</b>	One or more of <b>netFlow</b> , <b>sFlow</b> , or <b>ipfix</b> .
<b>spec.exportNetworkFlows.netFlow.collectors</b>	<b>array</b>	A list of IP address and network port pairs for up to 10 collectors.
<b>spec.exportNetworkFlows.sFlow.collectors</b>	<b>array</b>	A list of IP address and network port pairs for up to 10 collectors.
<b>spec.exportNetworkFlows.ipfix.collectors</b>	<b>array</b>	A list of IP address and network port pairs for up to 10 collectors.

After applying the following manifest to the CNO, the Operator configures Open vSwitch (OVS) on each node in the cluster to send network flows records to the NetFlow collector that is listening at **192.168.1.99:2056**.

#### Example configuration for tracking network flows

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

### 18.16.2. Adding destinations for network flows collectors

As a cluster administrator, you can configure the Cluster Network Operator (CNO) to send network flows metadata about the pod network to a network flows collector.

#### Prerequisites

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

- You have a network flows collector and know the IP address and port that it listens on.

## Procedure

1. Create a patch file that specifies the network flows collector type and the IP address and port information of the collectors:

```
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

2. Configure the CNO with the network flows collectors:

```
$ oc patch network.operator cluster --type merge -p "$(cat <file_name>.yaml)"
```

### Example output

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

## Verification

Verification is not typically necessary. You can run the following command to confirm that Open vSwitch (OVS) on each node is configured to send network flows records to one or more collectors.

1. View the Operator configuration to confirm that the **exportNetworkFlows** field is configured:

```
$ oc get network.operator cluster -o jsonpath=".spec.exportNetworkFlows"
```

### Example output

```
{"netFlow":{"collectors":["192.168.1.99:2056"]}}
```

2. View the network flows configuration in OVS from each node:

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node -o jsonpath='{range@.items[*]}.metadata.name}{\n}{end}'; do
  echo;
  echo $pod;
  oc -n openshift-ovn-kubernetes exec -c ovnkube-controller $pod \
    -- bash -c 'for type in ipfix sflow netflow ; do ovs-vsctl find $type ; done';
done
```

### Example output

```
ovnkube-node-xrn4p
_uuid          : a4d2aaca-5023-4f3d-9400-7275f92611f9
active_timeout   : 60
add_id_to_interface : false
engine_id       : []
engine_type     : []
```

```

external_ids      : {}
targets          : ["192.168.1.99:2056"]

ovnkube-node-z4vq9
_uuid           : 61d02fdb-9228-4993-8ff5-b27f01a29bd6
active_timeout   : 60
add_id_to_interface : false
engine_id        : []
engine_type      : []
external_ids     : {}
targets          : ["192.168.1.99:2056"]-
...
```

### 18.16.3. Deleting all destinations for network flows collectors

As a cluster administrator, you can configure the Cluster Network Operator (CNO) to stop sending network flows metadata to a network flows collector.

#### Prerequisites

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

#### Procedure

1. Remove all network flows collectors:

```
$ oc patch network.operator cluster --type='json' \
-p='[{"op":"remove", "path":"/spec/exportNetworkFlows"}]'
```

#### Example output

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

### 18.16.4. Additional resources

- [Network \[operator.openshift.io/v1\]](#)

## 18.17. CONFIGURING HYBRID NETWORKING

As a cluster administrator, you can configure the Red Hat OpenShift Networking OVN-Kubernetes network plugin to allow Linux and Windows nodes to host Linux and Windows workloads, respectively.

### 18.17.1. Configuring hybrid networking with OVN-Kubernetes

You can configure your cluster to use hybrid networking with the OVN-Kubernetes network plugin. This allows a hybrid cluster that supports different node networking configurations.

**NOTE**

This configuration is necessary to run both Linux and Windows nodes in the same cluster.

**Prerequisites**

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

**Procedure**

1. To configure the OVN-Kubernetes hybrid network overlay, enter the following command:

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{
  "spec": {
    "defaultNetwork": {
      "ovnKubernetesConfig": {
        "hybridOverlayConfig": {
          "hybridClusterNetwork": [
            {
              "cidr": "<cidr>",
              "hostPrefix": <prefix>
            }
          ],
          "hybridOverlayVXLANPort": <overlay_port>
        }
      }
    }
  }
}'
```

where:

**cidr**

Specify the CIDR configuration used for nodes on the additional overlay network. This CIDR must not overlap with the cluster network CIDR.

**hostPrefix**

Specifies the subnet prefix length to assign to each individual node. For example, if **hostPrefix** is set to **23**, then each node is assigned a /23 subnet out of the given **cidr**, which allows for 510 (2^(32 - 23) - 2) pod IP addresses. If you are required to provide access to nodes from an external network, configure load balancers and routers to manage the traffic.

**hybridOverlayVXLANPort**

Specify a custom VXLAN port for the additional overlay network. This is required for running Windows nodes in a cluster installed on vSphere, and must not be configured for any other cloud provider. The custom port can be any open port excluding the default **4789** port. For more information on this requirement, see the Microsoft documentation on [Pod-to-pod connectivity between hosts is broken](#).

**Example output**

network.operator.openshift.io/cluster patched

2. To confirm that the configuration is active, enter the following command. It can take several minutes for the update to apply.

```
$ oc get network.operator.openshift.io -o jsonpath=".items[0].spec.defaultNetwork.ovnKubernetesConfig"
```

### 18.17.2. Additional resources

- [Understanding Windows container workloads](#)
- [Enabling Windows container workloads](#)
- [Installing a cluster on AWS with network customizations](#)
- [Installing a cluster on Azure with network customizations](#)

# CHAPTER 19. CONFIGURING ROUTES

## 19.1. ROUTE CONFIGURATION

### 19.1.1. Creating an HTTP-based route

Create a route to host your application at a public URL. The route can either be secure or unsecured, depending on the network security configuration of your application. An HTTP-based route is an unsecured route that uses the basic HTTP routing protocol and exposes a service on an unsecured application port.

The following procedure describes how to create a simple HTTP-based route to a web application, using the **hello-openshift** application as an example.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as an administrator.
- You have a web application that exposes a port and a TCP endpoint listening for traffic on the port.

#### Procedure

1. Create a project called **hello-openshift** by running the following command:

```
$ oc new-project hello-openshift
```

2. Create a pod in the project by running the following command:

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. Create a service called **hello-openshift** by running the following command:

```
$ oc expose pod/hello-openshift
```

4. Create an unsecured route to the **hello-openshift** application by running the following command:

```
$ oc expose svc hello-openshift
```

#### Verification

- To verify that the **route** resource that you created, run the following command:

```
$ oc get routes -o yaml <name of resource> ①
```

- ① In this example, the route is named **hello-openshift**.

## Sample YAML definition of the created unsecured route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: www.example.com 1
  port:
    targetPort: 8080 2
  to:
    kind: Service
    name: hello-openshift
```

- 1** The **host** field is an alias DNS record that points to the service. This field can be any valid DNS name, such as **www.example.com**. The DNS name must follow DNS952 subdomain conventions. If not specified, a route name is automatically generated.
- 2** The **targetPort** field is the target port on pods that is selected by the service that this route points to.



### NOTE

To display your default ingress domain, run the following command:

```
$ oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

### 19.1.2. Creating a route for Ingress Controller sharding

A route allows you to host your application at a URL. In this case, the hostname is not set and the route uses a subdomain instead. When you specify a subdomain, you automatically use the domain of the Ingress Controller that exposes the route. For situations where a route is exposed by multiple Ingress Controllers, the route is hosted at multiple URLs.

The following procedure describes how to create a route for Ingress Controller sharding, using the **hello-openshift** application as an example.

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as a project administrator.
- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.
- You have configured the Ingress Controller for sharding.

## Procedure

1. Create a project called **hello-openshift** by running the following command:

```
$ oc new-project hello-openshift
```

2. Create a pod in the project by running the following command:

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/openshift/hello-openshift/hello-pod.json
```

3. Create a service called **hello-openshift** by running the following command:

```
$ oc expose pod/hello-openshift
```

4. Create a route definition called **hello-openshift-route.yaml**:

### YAML definition of the created route for sharding

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded 1
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift 2
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

- 1** Both the label key and its corresponding label value must match the ones specified in the Ingress Controller. In this example, the Ingress Controller has the label key and value **type: sharded**.
- 2** The route will be exposed using the value of the **subdomain** field. When you specify the **subdomain** field, you must leave the hostname unset. If you specify both the **host** and **subdomain** fields, then the route will use the value of the **host** field, and ignore the **subdomain** field.

5. Use **hello-openshift-route.yaml** to create a route to the **hello-openshift** application by running the following command:

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

## Verification

- Get the status of the route with the following command:

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

The resulting **Route** resource should look similar to the following:

### Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello Openshift Edge
  namespace: hello Openshift
spec:
  subdomain: hello Openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello Openshift
status:
  ingress:
    - host: hello Openshift.<apps-sharded.baseDomain.example.net> ①
      routerCanonicalHostname: router-sharded.<apps-sharded.baseDomain.example.net> ②
      routerName: sharded ③
```

- ① The hostname the Ingress Controller, or router, uses to expose the route. The value of the **host** field is automatically determined by the Ingress Controller, and uses its domain. In this example, the domain of the Ingress Controller is **<apps-sharded.baseDomain.example.net>**.
- ② The hostname of the Ingress Controller.
- ③ The name of the Ingress Controller. In this example, the Ingress Controller has the name **sharded**.

### 19.1.3. Configuring route timeouts

You can configure the default timeouts for an existing route when you have services in need of a low timeout, which is required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

#### Prerequisites

- You need a deployed Ingress Controller on a running cluster.

#### Procedure

1. Using the **oc annotate** command, add the timeout to the route:

```
$ oc annotate route <route_name> \
--overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> ①
```

- 1 Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).

The following example sets a timeout of two seconds on a route named **myroute**:

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

### 19.1.4. HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) policy is a security enhancement, which signals to the browser client that only HTTPS traffic is allowed on the route host. HSTS also optimizes web traffic by signaling HTTPS transport is required, without using HTTP redirects. HSTS is useful for speeding up interactions with websites.

When HSTS policy is enforced, HSTS adds a Strict Transport Security header to HTTP and HTTPS responses from the site. You can use the **insecureEdgeTerminationPolicy** value in a route to redirect HTTP to HTTPS. When HSTS is enforced, the client changes all requests from the HTTP URL to HTTPS before the request is sent, eliminating the need for a redirect.

Cluster administrators can configure HSTS to do the following:

- Enable HSTS per-route
- Disable HSTS per-route
- Enforce HSTS per-domain, for a set of domains, or use namespace labels in combination with domains



#### IMPORTANT

HSTS works only with secure routes, either edge-terminated or re-encrypt. The configuration is ineffective on HTTP or passthrough routes.

#### 19.1.4.1. Enabling HTTP Strict Transport Security per-route

HTTP strict transport security (HSTS) is implemented in the HAProxy template and applied to edge and re-encrypt routes that have the **haproxy.router.openshift.io/hsts\_header** annotation.

#### Prerequisites

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

#### Procedure

- To enable HSTS on a route, add the **haproxy.router.openshift.io/hsts\_header** value to the edge-terminated or re-encrypt route. You can use the **oc annotate** tool to do this by running the following command. To properly run the command, ensure that the semicolon (;) in the **haproxy.router.openshift.io/hsts\_header** route annotation is also surrounded by double quotation marks ("").

**Example annotate command that sets the maximum age to 31536000 ms (approximately 8.5 hours)**

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header=max-age=31536000;\\
includeSubDomains;preload"
```

### Example route configured with an annotation

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
  1 2 3
  # ...
spec:
  host: def.abc.com
  tls:
    termination: "reencrypt"
    ...
    wildcardPolicy: "Subdomain"
  # ...
```

- 1 Required. **max-age** measures the length of time, in seconds, that the HSTS policy is in effect. If set to **0**, it negates the policy.
- 2 Optional. When included, **includeSubDomains** tells the client that all subdomains of the host must have the same HSTS policy as the host.
- 3 Optional. When **max-age** is greater than 0, you can add **preload** in **haproxy.router.openshift.io/hsts\_header** to allow external services to include this site in their HSTS preload lists. For example, sites such as Google can construct a list of sites that have **preload** set. Browsers can then use these lists to determine which sites they can communicate with over HTTPS, even before they have interacted with the site. Without **preload** set, browsers must have interacted with the site over HTTPS, at least once, to get the header.

### Additional resources

- [Enabling HTTP/2 Ingress connectivity](#)

#### 19.1.4.2. Disabling HTTP Strict Transport Security per-route

To disable HTTP strict transport security (HSTS) per-route, you can set the **max-age** value in the route annotation to **0**.

### Prerequisites

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

### Procedure

- To disable HSTS, set the **max-age** value in the route annotation to **0**, by entering the following command:

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

### TIP

You can alternatively apply the following YAML to create the config map:

#### Example of disabling HSTS per-route

```
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=0
```

- To disable HSTS for every route in a namespace, enter the following command:

```
$ oc annotate route --all -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

### Verification

1. To query the annotation for all routes, enter the following command:

```
$ oc get route --all-namespaces -o go-template='{{range .items}}{{if .metadata.annotations}}
{{$a := index .metadata.annotations "haproxy.router.openshift.io/hsts_header"}}{{${n :=
.metadata.name}}{{with $a}}Name: {{$n}} HSTS: {{$a}}{{"\n}}{{else}}{{""}}{{end}}{{end}}
{{end}}'
```

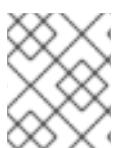
#### Example output

```
Name: routename HSTS: max-age=0
```

### 19.1.4.3. Enforcing HTTP Strict Transport Security per-domain

To enforce HTTP Strict Transport Security (HSTS) per-domain for secure routes, add a **requiredHSTSPolicies** record to the Ingress spec to capture the configuration of the HSTS policy.

If you configure a **requiredHSTSPolicy** to enforce HSTS, then any newly created route must be configured with a compliant HSTS policy annotation.



#### NOTE

To handle upgraded clusters with non-compliant HSTS routes, you can update the manifests at the source and apply the updates.



#### NOTE

You cannot use **oc expose route** or **oc create route** commands to add a route in a domain that enforces HSTS, because the API for these commands does not accept annotations.



## IMPORTANT

HSTS cannot be applied to insecure, or non-TLS routes, even if HSTS is requested for all routes globally.

### Prerequisites

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

### Procedure

1. Edit the Ingress configuration YAML by running the following command and updating fields as needed:

```
$ oc edit ingresses.config.openshift.io/cluster
```

### Example HSTS policy

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: 'hello-openshift-default.apps.username.devcluster.openshift.com'
  requiredHSTSPolicies: ①
    - domainPatterns: ②
      - '*hello-openshift-default.apps.username.devcluster.openshift.com'
      - '*hello-openshift-default2.apps.username.devcluster.openshift.com'
    namespaceSelector: ③
      matchLabels:
        myPolicy: strict
    maxAge: ④
      smallestMaxAge: 1
      largestMaxAge: 31536000
    preloadPolicy: RequirePreload ⑤
    includeSubDomainsPolicy: RequireIncludeSubDomains ⑥
    - domainPatterns:
      - 'abc.example.com'
      - '*xyz.example.com'
    namespaceSelector:
      matchLabels: {}
    maxAge: {}
    preloadPolicy: NoOpinion
    includeSubDomainsPolicy: RequireNoIncludeSubDomains
```

① Required. **requiredHSTSPolicies** are validated in order, and the first matching **domainPatterns** applies.

② Required. You must specify at least one **domainPatterns** hostname. Any number of domains can be listed. You can include multiple sections of enforcing options for different **domainPatterns**.

- 3 Optional. If you include **namespaceSelector**, it must match the labels of the project where the routes reside, to enforce the set HSTS policy on the routes. Routes that only match the
- 4 Required. **max-age** measures the length of time, in seconds, that the HSTS policy is in effect. This policy setting allows for a smallest and largest **max-age** to be enforced.
  - The **largestMaxAge** value must be between **0** and **2147483647**. It can be left unspecified, which means no upper limit is enforced.
  - The **smallestMaxAge** value must be between **0** and **2147483647**. Enter **0** to disable HSTS for troubleshooting, otherwise enter **1** if you never want HSTS to be disabled. It can be left unspecified, which means no lower limit is enforced.
- 5 Optional. Including **preload** in **haproxy.router.openshift.io/hsts\_header** allows external services to include this site in their HSTS preload lists. Browsers can then use these lists to determine which sites they can communicate with over HTTPS, before they have interacted with the site. Without **preload** set, browsers need to interact at least once with the site to get the header. **preload** can be set with one of the following:
  - **RequirePreload**: **preload** is required by the **RequiredHSTSPolicy**.
  - **RequireNoPreload**: **preload** is forbidden by the **RequiredHSTSPolicy**.
  - **NoOpinion**: **preload** does not matter to the **RequiredHSTSPolicy**.
- 6 Optional. **includeSubDomainsPolicy** can be set with one of the following:
  - **RequireIncludeSubDomains**: **includeSubDomains** is required by the **RequiredHSTSPolicy**.
  - **RequireNoIncludeSubDomains**: **includeSubDomains** is forbidden by the **RequiredHSTSPolicy**.
  - **NoOpinion**: **includeSubDomains** does not matter to the **RequiredHSTSPolicy**.

2. You can apply HSTS to all routes in the cluster or in a particular namespace by entering the **oc annotate command**.

- To apply HSTS to all routes in the cluster, enter the **oc annotate command**. For example:

```
$ oc annotate route --all --all-namespaces --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

- To apply HSTS to all routes in a particular namespace, enter the **oc annotate command**. For example:

```
$ oc annotate route --all -n my-namespace --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

## Verification

You can review the HSTS policy you configured. For example:

- To review the **maxAge** set for required HSTS policies, enter the following command:

```
$ oc get clusteroperator/ingress -n openshift-ingress-operator -o jsonpath='{range .spec.requiredHSTSPolicies[*]}.spec.requiredHSTSPolicies.maxAgePolicy.largestMaxAge} {"\n"}{end}'
```

- To review the HSTS annotations on all routes, enter the following command:

```
$ oc get route --all-namespaces -o go-template='{{range .items}}{{if .metadata.annotations}}{{{$a := index .metadata.annotations "haproxy.router.openshift.io/hsts_header"}}}{$n := .metadata.name}}{{with $a}}Name: {{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}{{end}}'
```

### Example output

```
Name: <_routename_> HSTS: max-age=31536000;preload;includeSubDomains
```

#### 19.1.5. Throughput issue troubleshooting methods

Sometimes applications deployed by using OpenShift Container Platform can cause network throughput issues, such as unusually high latency between specific services.

If pod logs do not reveal any cause of the problem, use the following methods to analyze performance issues:

- Use a packet analyzer, such as **ping** or **tcpdump** to analyze traffic between a pod and its node. For example, [run the \*\*tcpdump\*\* tool on each pod](#) while reproducing the behavior that led to the issue. Review the captures on both sides to compare send and receive timestamps to analyze the latency of traffic to and from a pod. Latency can occur in OpenShift Container Platform if a node interface is overloaded with traffic from other pods, storage devices, or the data plane.

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> ①
```

- ① **podip** is the IP address for the pod. Run the **oc get pod <pod\_name> -o wide** command to get the IP address of a pod.

The **tcpdump** command generates a file at **/tmp/dump.pcap** containing all traffic between these two pods. You can run the analyzer shortly before the issue is reproduced and stop the analyzer shortly after the issue is finished reproducing to minimize the size of the file. You can also [run a packet analyzer between the nodes](#) with:

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- Use a bandwidth measuring tool, such as **iperf**, to measure streaming throughput and UDP throughput. Locate any bottlenecks by running the tool from the pods first, and then running it from the nodes.
  - For information on installing and using **iperf**, see this [Red Hat Solution](#).
- In some cases, the cluster might mark the node with the router pod as unhealthy due to latency issues. Use worker latency profiles to adjust the frequency that the cluster waits for a status update from the node before taking action.
- If your cluster has designated lower-latency and higher-latency nodes, configure the **spec.nodePlacement** field in the Ingress Controller to control the placement of the router pod.

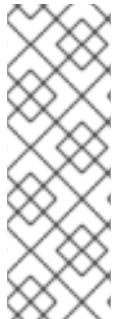
## Additional resources

- [Latency spikes or temporary reduction in throughput to remote workers](#)
- [Ingress Controller configuration parameters](#)

### 19.1.6. Using cookies to keep route statefulness

OpenShift Container Platform provides sticky sessions, which enables stateful application traffic by ensuring all traffic hits the same endpoint. However, if the endpoint pod terminates, whether through restart, scaling, or a change in configuration, this statefulness can disappear.

OpenShift Container Platform can use cookies to configure session persistence. The ingress controller selects an endpoint to handle any user requests, and creates a cookie for the session. The cookie is passed back in the response to the request and the user sends the cookie back with the next request in the session. The cookie tells the ingress controller which endpoint is handling the session, ensuring that client requests use the cookie so that they are routed to the same pod.



#### NOTE

Cookies cannot be set on passthrough routes, because the HTTP traffic cannot be seen. Instead, a number is calculated based on the source IP address, which determines the backend.

If backends change, the traffic can be directed to the wrong server, making it less sticky. If you are using a load balancer, which hides source IP, the same number is set for all connections and traffic is sent to the same pod.

#### 19.1.6.1. Annotating a route with a cookie

You can set a cookie name to overwrite the default, auto-generated one for the route. This allows the application receiving route traffic to know the cookie name. Deleting the cookie can force the next request to re-choose an endpoint. The result is that if a server is overloaded, that server tries to remove the requests from the client and redistribute them.

#### Procedure

1. Annotate the route with the specified cookie name:

```
$ oc annotate route <route_name> router.openshift.io/cookie_name=<cookie_name>"
```

where:

**<route\_name>**

Specifies the name of the route.

**<cookie\_name>**

Specifies the name for the cookie.

For example, to annotate the route **my\_route** with the cookie name **my\_cookie**:

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2. Capture the route hostname in a variable:

■

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

where:

#### <route\_name>

Specifies the name of the route.

3. Save the cookie, and then access the route:

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

Use the cookie saved by the previous command when connecting to the route:

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

### 19.1.7. Path-based routes

Path-based routes specify a path component that can be compared against a URL, which requires that the traffic for the route be HTTP based. Thus, multiple routes can be served using the same hostname, each with a different path. Routers should match routes based on the most specific path to the least.

The following table shows example routes and their accessibility:

**Table 19.1. Route availability**

Route	When Compared to	Accessible
<code>www.example.com/test</code>	<code>www.example.com/test</code>	Yes
	<code>www.example.com</code>	No
<code>www.example.com/test</code> and <code>www.example.com</code>	<code>www.example.com/test</code>	Yes
	<code>www.example.com</code>	Yes
<code>www.example.com</code>	<code>www.example.com/text</code>	Yes (Matched by the host, not the route)
	<code>www.example.com</code>	Yes

#### An unsecured route with a path

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
```

```
to:
kind: Service
name: service-name
```

- The path is the only added attribute for a path-based route.



#### NOTE

Path-based routing is not available when using passthrough TLS, as the router does not terminate TLS in that case and cannot read the contents of the request.

### 19.1.8. HTTP header configuration

OpenShift Container Platform provides different methods for working with HTTP headers. When setting or deleting headers, you can use specific fields in the Ingress Controller or an individual route to modify request and response headers. You can also set certain headers by using route annotations. The various ways of configuring headers can present challenges when working together.



#### NOTE

You can only set or delete headers within an **IngressController** or **Route** CR, you cannot append them. If an HTTP header is set with a value, that value must be complete and not require appending in the future. In situations where it makes sense to append a header, such as the X-Forwarded-For header, use the **spec.httpHeaders.forwardedHeaderPolicy** field, instead of **spec.httpHeaders.actions**.

#### 19.1.8.1. Order of precedence

When the same HTTP header is modified both in the Ingress Controller and in a route, HAProxy prioritizes the actions in certain ways depending on whether it is a request or response header.

- For HTTP response headers, actions specified in the Ingress Controller are executed after the actions specified in a route. This means that the actions specified in the Ingress Controller take precedence.
- For HTTP request headers, actions specified in a route are executed after the actions specified in the Ingress Controller. This means that the actions specified in the route take precedence.

For example, a cluster administrator sets the X-Frame-Options response header with the value **DENY** in the Ingress Controller using the following configuration:

#### Example IngressController spec

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
        - name: X-Frame-Options
          action:
```

```

type: Set
set:
  value: DENY

```

A route owner sets the same response header that the cluster administrator set in the Ingress Controller, but with the value **SAMEORIGIN** using the following configuration:

### Example Route spec

```

apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
        - name: X-Frame-Options
          action:
            type: Set
            set:
              value: SAMEORIGIN

```

When both the **IngressController** spec and **Route** spec are configuring the X-Frame-Options response header, then the value set for this header at the global level in the Ingress Controller takes precedence, even if a specific route allows frames. For a request header, the **Route** spec value overrides the **IngressController** spec value.

This prioritization occurs because the **haproxy.config** file uses the following logic, where the Ingress Controller is considered the front end and individual routes are considered the back end. The header value **DENY** applied to the front end configurations overrides the same header with the value **SAMEORIGIN** that is set in the back end:

```

frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'

```

Additionally, any actions defined in either the Ingress Controller or a route override values set using route annotations.

#### 19.1.8.2. Special case headers

The following headers are either prevented entirely from being set or deleted, or allowed under specific circumstances:

**Table 19.2. Special case header configuration options**

Header name	Configurable using <b>IngressController spec</b>	Configurable using <b>Route spec</b>	Reason for disallowment	Configurable using another method
<b>proxy</b>	No	No	The <b>proxy</b> HTTP request header can be used to exploit vulnerable CGI applications by injecting the header value into the <b>HTTP_PROXY</b> environment variable. The <b>proxy</b> HTTP request header is also non-standard and prone to error during configuration.	No
<b>host</b>	No	Yes	When the <b>host</b> HTTP request header is set using the <b>IngressController</b> CR, HAProxy can fail when looking up the correct route.	No
<b>strict-transport-security</b>	No	No	The <b>strict-transport-security</b> HTTP response header is already handled using route annotations and does not need a separate implementation.	Yes: the <b>haproxy.router.openshift.io/host_header</b> route annotation

Header name	Configurable using <b>IngressController spec</b>	Configurable using <b>Route spec</b>	Reason for disallowment	Configurable using another method
<b>cookie and set-cookie</b>	No	No	The cookies that HAProxy sets are used for session tracking to map client connections to particular backend servers. Allowing these headers to be set could interfere with HAProxy's session affinity and restrict HAProxy's ownership of a cookie.	Yes: <ul style="list-style-type: none"> <li>the <b>haproxy.router.openshift.io/disable_cookie</b> route annotation</li> <li>the <b>haproxy.router.openshift.io/cookie_name</b> route annotation</li> </ul>

### 19.1.9. Setting or deleting HTTP request and response headers in a route

You can set or delete certain HTTP request and response headers for compliance purposes or other reasons. You can set or delete these headers either for all routes served by an Ingress Controller or for specific routes.

For example, you might want to enable a web application to serve content in alternate locations for specific routes if that content is written in multiple languages, even if there is a default global location specified by the Ingress Controller serving the routes.

The following procedure creates a route that sets the Content-Location HTTP request header so that the URL associated with the application, <https://app.example.com>, directs to the location <https://app.example.com/lang/en-us>. Directing application traffic to this location means that anyone using that specific route is accessing web content written in American English.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You are logged into an OpenShift Container Platform cluster as a project administrator.
- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.

#### Procedure

1. Create a route definition and save it in a file called **app-example-route.yaml**:

## YAML definition of the created route with HTTP header directives

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  host: app.example.com
  tls:
    termination: edge
  to:
    kind: Service
    name: app-example
  httpHeaders:
    actions: ①
      response: ②
        - name: Content-Location ③
          action:
            type: Set ④
            set:
              value: /lang/en-us ⑤
```

- ① The list of actions you want to perform on the HTTP headers.
- ② The type of header you want to change. In this case, a response header.
- ③ The name of the header you want to change. For a list of available headers you can set or delete, see *HTTP header configuration*.
- ④ The type of action being taken on the header. This field can have the value **Set** or **Delete**.
- ⑤ When setting HTTP headers, you must provide a **value**. The value can be a string from a list of available directives for that header, for example **DENY**, or it can be a dynamic value that will be interpreted using HAProxy's dynamic value syntax. In this case, the value is set to the relative location of the content.

2. Create a route to your existing web application using the newly created route definition:

```
$ oc -n app-example create -f app-example-route.yaml
```

For HTTP request headers, the actions specified in the route definitions are executed after any actions performed on HTTP request headers in the Ingress Controller. This means that any values set for those request headers in a route will take precedence over the ones set in the Ingress Controller. For more information on the processing order of HTTP headers, see *HTTP header configuration*.

### 19.1.10. Route-specific annotations

The Ingress Controller can set the default options for all the routes it exposes. An individual route can override some of these defaults by providing specific configurations in its annotations. Red Hat does not support adding a route annotation to an operator-managed route.



## IMPORTANT

To create an allow list with multiple source IPs or subnets, use a space-delimited list. Any other delimiter type causes the list to be ignored without a warning or error message.

**Table 19.3. Route annotations**

Variable	Description	Environment variable used as default
<b>haproxy.router.openshift.io/balance</b>	Sets the load-balancing algorithm. Available options are <b>random</b> , <b>source</b> , <b>roundrobin</b> [ <sup>1</sup> ], and <b>leastconn</b> . The default value is <b>source</b> for TLS passthrough routes. For all other routes, the default is <b>random</b> .	<b>ROUTER_TCP_BALANCE_SCHEME</b> for passthrough routes. Otherwise, use <b>ROUTER_LOAD_BALANCE_ALGORITHM</b> .
<b>haproxy.router.openshift.io/disable_cookies</b>	Disables the use of cookies to track related connections. If set to ' <b>true</b> ' or ' <b>TRUE</b> ', the balance algorithm is used to choose which back-end serves connections for each incoming HTTP request.	
<b>router.openshift.io/cookie_name</b>	Specifies an optional cookie to use for this route. The name must consist of any combination of upper and lower case letters, digits, "_", and "-". The default is the hashed internal key name for the route.	
<b>haproxy.router.openshift.io/pod-concurrent-connections</b>	Sets the maximum number of connections that are allowed to a backing pod from a router. Note: If there are multiple pods, each can have this many connections. If you have multiple routers, there is no coordination among them, each may connect this many times. If not set, or set to 0, there is no limit.	
<b>haproxy.router.openshift.io/rate-limit-connections</b>	Setting ' <b>true</b> ' or ' <b>TRUE</b> ' enables rate limiting functionality which is implemented through stick-tables on the specific backend per route. Note: Using this annotation provides basic protection against denial-of-service attacks.	

Variable	Description	Environment variable used as default
<b>haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp</b>	<p>Limits the number of concurrent TCP connections made through the same source IP address. It accepts a numeric value.</p> <p>Note: Using this annotation provides basic protection against denial-of-service attacks.</p>	
<b>haproxy.router.openshift.io/rate-limit-connections.rate-http</b>	<p>Limits the rate at which a client with the same source IP address can make HTTP requests. It accepts a numeric value.</p> <p>Note: Using this annotation provides basic protection against denial-of-service attacks.</p>	
<b>haproxy.router.openshift.io/rate-limit-connections.rate-tcp</b>	<p>Limits the rate at which a client with the same source IP address can make TCP connections. It accepts a numeric value.</p> <p>Note: Using this annotation provides basic protection against denial-of-service attacks.</p>	
<b>haproxy.router.openshift.io/tmeout</b>	Sets a server-side timeout for the route. (TimeUnits)	<b>ROUTER_DEFAULT_SERVER_TIMEOUT</b>
<b>haproxy.router.openshift.io/tmeout-tunnel</b>	This timeout applies to a tunnel connection, for example, WebSocket over cleartext, edge, reencrypt, or passthrough routes. With cleartext, edge, or reencrypt route types, this annotation is applied as a timeout tunnel with the existing timeout value. For the passthrough route types, the annotation takes precedence over any existing timeout value set.	<b>ROUTER_DEFAULT_TUNNEL_TIMEOUT</b>
<b>ingresses.config/cluster ingress.operator.openshift.io/hard-stop-after</b>	You can set either an IngressController or the ingress config . This annotation redeploys the router and configures the HA proxy to emit the haproxy <b>hard-stop-after</b> global option, which defines the maximum time allowed to perform a clean soft-stop.	<b>ROUTER_HARD_STOP_AFTER</b>

Variable	Description	Environment variable used as default
<b>router.openshift.io/haproxy.health.check.interval</b>	Sets the interval for the back-end health checks. (TimeUnits)	<b>ROUTER_BACKEND_CHECK_INTERVAL</b>
<b>haproxy.router.openshift.io/ipp_allowlist</b>	<p>Sets an allowlist for the route. The allowlist is a space-separated list of IP addresses and CIDR ranges for the approved source addresses. Requests from IP addresses that are not in the allowlist are dropped.</p> <p>The maximum number of IP addresses and CIDR ranges directly visible in the <b>haproxy.config</b> file is 61. [2]</p>	
<b>haproxy.router.openshift.io/hsts_header</b>	Sets a Strict-Transport-Security header for the edge terminated or re-encrypt route.	
<b>haproxy.router.openshift.io/rewrite-target</b>	Sets the rewrite path of the request on the backend.	
<b>router.openshift.io/cookie-same-site</b>	<p>Sets a value to restrict cookies. The values are:</p> <p><b>Lax</b>: the browser does not send cookies on cross-site requests, but does send cookies when users navigate to the origin site from an external site. This is the default browser behavior when the <b>SameSite</b> value is not specified.</p> <p><b>Strict</b>: the browser sends cookies only for same-site requests.</p> <p><b>None</b>: the browser sends cookies for both cross-site and same-site requests.</p> <p>This value is applicable to re-encrypt and edge routes only. For more information, see the <a href="#">SameSite cookies documentation</a>.</p>	

Variable	Description	Environment variable used as default
<b>haproxy.router.openshift.io/set-forwarded-headers</b>	<p>Sets the policy for handling the <b>Forwarded</b> and <b>X-Forwarded-For</b> HTTP headers per route. The values are:</p> <ul style="list-style-type: none"> <li><b>append</b>: appends the header, preserving any existing header. This is the default value.</li> <li><b>replace</b>: sets the header, removing any existing header.</li> <li><b>never</b>: never sets the header, but preserves any existing header.</li> <li><b>if-none</b>: sets the header if it is not already set.</li> </ul>	<b>ROUTER_SET_FORWARDERD_HEADERS</b>

1. By default, the router reloads every 5 s which resets the balancing connection across pods from the beginning. As a result, the **roundrobin** state is not preserved across reloads. This algorithm works best when pods have nearly identical computing capabilities and storage capacity. If your application or service has continuously changing endpoints, for example, due to the use of a CI/CD pipeline, uneven balancing can result. In this case, use a different algorithm.
2. If the number of IP addresses and CIDR ranges in an allowlist exceeds 61, they are written into a separate file that is then referenced from the **haproxy.config** file. This file is stored in the **/var/lib/haproxy/router/allowlists** folder.



#### NOTE

To ensure that the addresses are written to the allowlist, check that the full list of CIDR ranges are listed in the Ingress Controller configuration file. The etcd object size limit restricts how large a route annotation can be. Because of this, it creates a threshold for the maximum number of IP addresses and CIDR ranges that you can include in an allowlist.



#### NOTE

Environment variables cannot be edited.

### Router timeout variables

**TimeUnits** are represented by a number followed by the unit: **us** \*(microseconds), **ms** (milliseconds, default), **s** (seconds), **m** (minutes), **h** \*(hours), **d** (days).

The regular expression is: [1-9][0-9]\*(**us\|ms\|s\|m\|h\|d**).

Variable	Default	Description
<b>ROUTER_BACKEND_CHECK_INTERVAL</b>	<b>5000ms</b>	Length of time between subsequent liveness checks on back ends.
<b>ROUTER_CLIENT_FIN_TIMEOUT</b>	<b>1s</b>	Controls the TCP FIN timeout period for the client connecting to the route. If the FIN sent to close the connection does not answer within the given time, HAProxy closes the connection. This is harmless if set to a low value and uses fewer resources on the router.
<b>ROUTER_DEFAULT_CLIENT_TIMEOUT</b>	<b>30s</b>	Length of time that a client has to acknowledge or send data.
<b>ROUTER_DEFAULT_CONNECT_TIMEOUT</b>	<b>5s</b>	The maximum connection time.
<b>ROUTER_DEFAULT_SERVER_FIN_TIMEOUT</b>	<b>1s</b>	Controls the TCP FIN timeout from the router to the pod backing the route.
<b>ROUTER_DEFAULT_SERVER_TIMEOUT</b>	<b>30s</b>	Length of time that a server has to acknowledge or send data.
<b>ROUTER_DEFAULT_TUNNEL_TIMEOUT</b>	<b>1h</b>	Length of time for TCP or WebSocket connections to remain open. This timeout period resets whenever HAProxy reloads.
<b>ROUTER_SLOWLORIS_HTTP_KEEPALIVE</b>	<b>300s</b>	<p>Set the maximum time to wait for a new HTTP request to appear. If this is set too low, it can cause problems with browsers and applications not expecting a small <b>keepalive</b> value.</p> <p>Some effective timeout values can be the sum of certain variables, rather than the specific expected timeout. For example, <b>ROUTER_SLOWLORIS_HTTP_KEEPALIVE</b> adjusts <b>timeout http-keep-alive</b>. It is set to <b>300s</b> by default, but HAProxy also waits on <b>tcp-request inspect-delay</b>, which is set to <b>5s</b>. In this case, the overall timeout would be <b>300s</b> plus <b>5s</b>.</p>
<b>ROUTER_SLOWLORIS_TIMEOUT</b>	<b>10s</b>	Length of time the transmission of an HTTP request can take.

Variable	Default	Description
<b>RELOAD_INTERVAL</b>	<b>5s</b>	Allows the minimum frequency for the router to reload and accept new changes.
<b>ROUTER_METRICS_HAPROXY_TIMEOUT</b>	<b>5s</b>	Timeout for the gathering of HAProxy metrics.

### A route setting custom timeout

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms ①
...

```

- ① Specifies the new timeout with HAProxy supported units (**us**, **ms**, **s**, **m**, **h**, **d**). If the unit is not provided, **ms** is the default.



#### NOTE

Setting a server-side timeout value for passthrough routes too low can cause WebSocket connections to timeout frequently on that route.

### A route that allows only one specific IP address

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.10
```

### A route that allows several IP addresses

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.10 192.168.1.11 192.168.1.12
```

### A route that allows an IP address CIDR network

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 192.168.1.0/24
```

### A route that allows both IP an address and IP address CIDR networks

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_allowlist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

## A route specifying a rewrite target

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / ①
...

```

- ① Sets / as rewrite path of the request on the backend.

Setting the **haproxy.router.openshift.io/rewrite-target** annotation on a route specifies that the Ingress Controller should rewrite paths in HTTP requests using this route before forwarding the requests to the backend application. The part of the request path that matches the path specified in **spec.path** is replaced with the rewrite target specified in the annotation.

The following table provides examples of the path rewriting behavior for various combinations of **spec.path**, request path, and rewrite target.

**Table 19.4. rewrite-target examples**

Route.spec.path	Request path	Rewrite target	Forwarded request path
/foo	/foo	/	/
/foo	/foo/	/	/
/foo	/foo/bar	/	/bar
/foo	/foo/bar/	/	/bar/
/foo	/foo	/bar	/bar
/foo	/foo/	/bar	/bar/
/foo	/foo/bar	/baz	/baz/bar
/foo	/foo/bar/	/baz	/baz/bar/
/foo/	/foo	/	N/A (request path does not match route path)
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

Certain special characters in **haproxy.router.openshift.io/rewrite-target** require special handling because they must be escaped properly. Refer to the following table to understand how these characters are handled.

**Table 19.5. Special character handling**

For character	Use characters	Notes
#	\#	Avoid # because it terminates the rewrite expression
%	% or %%	Avoid odd sequences such as %%%
'	\'	Avoid ' because it is ignored

All other valid URL characters can be used without escaping.

### 19.1.11. Configuring the route admission policy

Administrators and application developers can run applications in multiple namespaces with the same domain name. This is for organizations where multiple teams develop microservices that are exposed on the same hostname.



#### WARNING

Allowing claims across namespaces should only be enabled for clusters with trust between namespaces, otherwise a malicious user could take over a hostname. For this reason, the default admission policy disallows hostname claims across namespaces.

#### Prerequisites

- Cluster administrator privileges.

#### Procedure

- Edit the **.spec.routeAdmission** field of the **ingresscontroller** resource variable using the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec": {"routeAdmission": {"namespaceOwnership": "InterNamespaceAllowed"}}}' --type=merge
```

#### Sample Ingress Controller configuration

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
...
...
```

**TIP**

You can alternatively apply the following YAML to configure the route admission policy:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

### 19.1.12. Creating a route through an Ingress object

Some ecosystem components have an integration with Ingress resources but not with route resources. To cover this case, OpenShift Container Platform automatically creates managed route objects when an Ingress object is created. These route objects are deleted when the corresponding Ingress objects are deleted.

#### Procedure

- 1 Define an Ingress object in the OpenShift Container Platform console or by entering the **oc create** command:

#### YAML Definition of an Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt" ①
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert ②
spec:
  rules:
    - host: www.example.com ③
      http:
        paths:
          - backend:
              service:
                name: frontend
                port:
                  number: 443
              path: /
              pathType: Prefix
            tls:
              - hosts:
                  - www.example.com
                secretName: example-com-tls-certificate
```

① The **route.openshift.io/termination** annotation can be used to configure the **spec.tls.termination** field of the **Route** as **Ingress** has no field for this. The accepted values are **edge**, **passthrough** and **reencrypt**. All other values are silently ignored. When

the annotation value is unset, **edge** is the default route. The TLS certificate details must be defined in the template file to implement the default edge route.

- 3 When working with an **Ingress** object, you must specify an explicit hostname, unlike when working with routes. You can use the `<host_name>.<cluster_ingress_domain>` syntax, for example `apps.openshiftdemos.com`, to take advantage of the `*`. `<cluster_ingress_domain>` wildcard DNS record and serving certificate for the cluster. Otherwise, you must ensure that there is a DNS record for the chosen hostname.
  - a. If you specify the **passthrough** value in the `route.openshift.io/termination` annotation, set **path** to `"` and **pathType** to **ImplementationSpecific** in the spec:

```
spec:
  rules:
    - host: www.example.com
      http:
        paths:
          - path: "
            pathType: ImplementationSpecific
            backend:
              service:
                name: frontend
                port:
                  number: 443
```

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

- 2 The `route.openshift.io/destination-ca-certificate-secret` can be used on an Ingress object to define a route with a custom destination certificate (CA). The annotation references a Kubernetes secret, **secret-ca-cert** that will be inserted into the generated route.
  - a. To specify a route object with a destination CA from an ingress object, you must create a **kubernetes.io/tls** or **Opaque** type secret with a certificate in PEM-encoded format in the `data.tls.crt` specifier of the secret.

2. List your routes:

```
$ oc get routes
```

The result includes an autogenerated route whose name starts with **frontend-**:

NAME	HOST/PORT	PATH	SERVICES	PORT	TERMINATION
WILDCARD					
frontend-gnztq	www.example.com		frontend	443	reencrypt/Redirect None

If you inspect this route, it looks this:

### YAML Definition of an autogenerated route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-gnztq
```

```

ownerReferences:
- apiVersion: networking.k8s.io/v1
  controller: true
  kind: Ingress
  name: frontend
  uid: 4e6c59cc-704d-4f44-b390-617d879033b6
spec:
  host: www.example.com
  path: /
  port:
    targetPort: https
  tls:
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
  insecureEdgeTerminationPolicy: Redirect
  key: |
    -----BEGIN RSA PRIVATE KEY-----
    [...]
    -----END RSA PRIVATE KEY-----
  termination: reencrypt
  destinationCACertificate: |
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
to:
  kind: Service
  name: frontend

```

### 19.1.13. Creating a route using the default certificate through an Ingress object

If you create an Ingress object without specifying any TLS configuration, OpenShift Container Platform generates an insecure route. To create an Ingress object that generates a secure, edge-terminated route using the default ingress certificate, you can specify an empty TLS configuration as follows.

#### Prerequisites

- You have a service that you want to expose.
- You have access to the OpenShift CLI (**oc**).

#### Procedure

1. Create a YAML file for the Ingress object. In this example, the file is called **example-ingress.yaml**:

#### YAML definition of an Ingress object

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  ...
spec:

```

```
rules:
```

```
...
```

```
tls:
```

```
- {} ①
```

- ① Use this exact syntax to specify TLS without specifying a custom certificate.

2. Create the Ingress object by running the following command:

```
$ oc create -f example-ingress.yaml
```

## Verification

- Verify that OpenShift Container Platform has created the expected route for the Ingress object by running the following command:

```
$ oc get routes -o yaml
```

## Example output

```
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend-j9sdd ①
    ...
  spec:
    ...
    tls: ②
      insecureEdgeTerminationPolicy: Redirect
      termination: edge ③
    ...

```

- ① The name of the route includes the name of the Ingress object followed by a random suffix.
- ② In order to use the default certificate, the route should not specify **spec.certificate**.
- ③ The route should specify the **edge** termination policy.

### 19.1.14. Creating a route using the destination CA certificate in the Ingress annotation

The **route.openshift.io/destination-ca-certificate-secret** annotation can be used on an Ingress object to define a route with a custom destination CA certificate.

#### Prerequisites

- You may have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.

- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.
- You must have a separate destination CA certificate in a PEM-encoded file.
- You must have a service that you want to expose.

## Procedure

1. Create a secret for the destination CA certificate by entering the following command:

```
$ oc create secret generic dest-ca-cert --from-file=tls.crt=<file_path>
```

For example:

```
$ oc -n test-ns create secret generic dest-ca-cert --from-file=tls.crt=tls.crt
```

## Example output

```
secret/dest-ca-cert created
```

2. Add the **route.openshift.io/destination-ca-certificate-secret** to the Ingress annotations:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt"
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert ①
...
...
```

① The annotation references a Kubernetes secret.

3. The secret referenced in this annotation will be inserted into the generated route.

## Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
tls:
  insecureEdgeTerminationPolicy: Redirect
  termination: reencrypt
  destinationCACertificate: |
    -----BEGIN CERTIFICATE-----
```

```
[...]
-----END CERTIFICATE-----
...
```

### 19.1.15. Configuring the OpenShift Container Platform Ingress Controller for dual-stack networking

If your OpenShift Container Platform cluster is configured for IPv4 and IPv6 dual-stack networking, your cluster is externally reachable by OpenShift Container Platform routes.

The Ingress Controller automatically serves services that have both IPv4 and IPv6 endpoints, but you can configure the Ingress Controller for single-stack or dual-stack services.

#### Prerequisites

- You deployed an OpenShift Container Platform cluster on bare metal.
- You installed the OpenShift CLI (**oc**).

#### Procedure

1. To have the Ingress Controller serve traffic over IPv4/IPv6 to a workload, you can create a service YAML file or modify an existing service YAML file by setting the **ipFamilies** and **ipFamilyPolicy** fields. For example:

##### Sample service YAML file

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: yyyy-mm-ddT00:00:00Z
  labels:
    name: <service_name>
    manager: kubectl-create
    operation: Update
    time: yyyy-mm-ddT00:00:00Z
    name: <service_name>
    namespace: <namespace_name>
    resourceVersion: "<resource_version_number>"
    selfLink: "/api/v1/namespaces/<namespace_name>/services/<service_name>"
    uid: <uid_number>
spec:
  clusterIP: 172.30.0.0/16
  clusterIPs: 1
    - 172.30.0.0/16
    - <second_IP_address>
  ipFamilies: 2
    - IPv4
    - IPv6
  ipFamilyPolicy: RequireDualStack 3
  ports:
    - port: 8080
      protocol: TCP
      targetport: 8080
```

```

selector:
  name: <namespace_name>
sessionAffinity: None
type: ClusterIP
status:
  loadbalancer: {}

```

- 1 In a dual-stack instance, there are two different **clusterIPs** provided.
- 2 For a single-stack instance, enter **IPv4** or **IPv6**. For a dual-stack instance, enter both **IPv4** and **IPv6**.
- 3 For a single-stack instance, enter **SingleStack**. For a dual-stack instance, enter **RequireDualStack**.

These resources generate corresponding **endpoints**. The Ingress Controller now watches **endpointslices**.

2. To view **endpoints**, enter the following command:

```
$ oc get endpoints
```

3. To view **endpointslices**, enter the following command:

```
$ oc get endpointslices
```

#### Additional resources

- [Specifying an alternative cluster domain using the appsDomain option](#)

## 19.2. SECURED ROUTES

Secure routes provide the ability to use several types of TLS termination to serve certificates to the client. The following sections describe how to create re-encrypt, edge, and passthrough routes with custom certificates.



### IMPORTANT

If you create routes in Microsoft Azure through public endpoints, the resource names are subject to restriction. You cannot create resources that use certain terms. For a list of terms that Azure restricts, see [Resolve reserved resource name errors](#) in the Azure documentation.

### 19.2.1. Creating a re-encrypt route with a custom certificate

You can configure a secure route using reencrypt TLS termination with a custom certificate by using the **oc create route** command.

#### Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.

- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.
- You must have a separate destination CA certificate in a PEM-encoded file.
- You must have a service that you want to expose.



### NOTE

Password protected key files are not supported. To remove a passphrase from a key file, use the following command:

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

## Procedure

This procedure creates a **Route** resource with a custom certificate and reencrypt TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You must also specify a destination CA certificate to enable the Ingress Controller to trust the service's certificate. You may also specify a CA certificate if needed to complete the certificate chain. Substitute the actual path names for **tls.crt**, **tls.key**, **cacert.crt**, and (optionally) **ca.crt**. Substitute the name of the **Service** resource that you want to expose for **frontend**. Substitute the appropriate hostname for **www.example.com**.

- Create a secure **Route** resource using reencrypt TLS termination and a custom certificate:

```
$ oc create route reencrypt --service=frontend --cert=tls.crt --key=tls.key --dest-ca-cert=destca.crt --ca-cert=ca.crt --hostname=www.example.com
```

If you examine the resulting **Route** resource, it should look similar to the following:

### YAML Definition of the Secure Route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
```

```
-----END CERTIFICATE-----
destinationCACertificate: |-
-----BEGIN CERTIFICATE-----
[...]
-----END CERTIFICATE-----
```

See **oc create route reencrypt --help** for more options.

### 19.2.2. Creating an edge route with a custom certificate

You can configure a secure route using edge TLS termination with a custom certificate by using the **oc create route** command. With an edge route, the Ingress Controller terminates TLS encryption before forwarding traffic to the destination pod. The route specifies the TLS certificate and key that the Ingress Controller uses for the route.

#### Prerequisites

- You must have a certificate/key pair in PEM-encoded files, where the certificate is valid for the route host.
- You may have a separate CA certificate in a PEM-encoded file that completes the certificate chain.
- You must have a service that you want to expose.



#### NOTE

Password protected key files are not supported. To remove a passphrase from a key file, use the following command:

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

#### Procedure

This procedure creates a **Route** resource with a custom certificate and edge TLS termination. The following assumes that the certificate/key pair are in the **tls.crt** and **tls.key** files in the current working directory. You may also specify a CA certificate if needed to complete the certificate chain. Substitute the actual path names for **tls.crt**, **tls.key**, and (optionally) **ca.crt**. Substitute the name of the service that you want to expose for **frontend**. Substitute the appropriate hostname for **www.example.com**.

- Create a secure **Route** resource using edge TLS termination and a custom certificate.

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=www.example.com
```

If you examine the resulting **Route** resource, it should look similar to the following:

#### YAML Definition of the Secure Route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
```

```

host: www.example.com
to:
  kind: Service
  name: frontend
tls:
  termination: edge
  key: |-
    -----BEGIN PRIVATE KEY-----
    [...]
    -----END PRIVATE KEY-----
  certificate: |-
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
  caCertificate: |-
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----

```

See **oc create route edge --help** for more options.

### 19.2.3. Creating a passthrough route

You can configure a secure route using passthrough termination by using the **oc create route** command. With passthrough termination, encrypted traffic is sent straight to the destination without the router providing TLS termination. Therefore no key or certificate is required on the route.

#### Prerequisites

- You must have a service that you want to expose.

#### Procedure

- Create a **Route** resource:

```
$ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
```

If you examine the resulting **Route** resource, it should look similar to the following:

#### A Secured Route Using Passthrough Termination

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured ①
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough ②
    insecureEdgeTerminationPolicy: None ③

```

```
to:
kind: Service
name: frontend
```

- 1** The name of the object, which is limited to 63 characters.
- 2** The **termination** field is set to **passthrough**. This is the only required **tls** field.
- 3** Optional **insecureEdgeTerminationPolicy**. The only valid values are **None**, **Redirect**, or empty for disabled.

The destination pod is responsible for serving certificates for the traffic at the endpoint. This is currently the only method that can support requiring client certificates, also known as two-way authentication.

#### 19.2.4. Creating a route with externally managed certificate



##### IMPORTANT

Securing route with external certificates in TLS secrets 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](#).

You can configure OpenShift Container Platform routes with third-party certificate management solutions by using the **.spec.tls.externalCertificate** field of the route API. You can reference externally managed TLS certificates via secrets, eliminating the need for manual certificate management. Using the externally managed certificate reduces errors ensuring a smoother rollout of certificate updates, enabling the OpenShift router to serve renewed certificates promptly.



##### NOTE

This feature applies to both edge routes and re-encrypt routes.

#### Prerequisites

- You must enable the **RouteExternalCertificate** feature gate.
- You must have the **create** and **update** permissions on the **routes/custom-host**.
- You must have a secret containing a valid certificate/key pair in PEM-encoded format of type **kubernetes.io/tls**, which includes both **tls.key** and **tls.crt** keys.
- You must place the referenced secret in the same namespace as the route you want to secure.

#### Procedure

1. Create a **role** in the same namespace as the secret to allow the router service account read access by running the following command:

```
$ oc create role secret-reader --verb=get,list,watch --resource=secrets --resource-name=<secret-name> \ ①
--namespace=<current-namespace> ②
```

- ① Specify the actual name of your secret.
- ② Specify the namespace where both your secret and route reside.

2. Create a **rolebinding** in the same namespace as the secret and bind the router service account to the newly created role by running the following command:

```
$ oc create rolebinding secret-reader-binding --role=secret-reader --
serviceaccount=openshift-ingress:router --namespace=<current-namespace> ①
```

- ① Specify the namespace where both your secret and route reside.

3. Create a YAML file that defines the **route** and specifies the secret containing your certificate using the following example.

#### YAML definition of the secure route

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: myedge
  namespace: test
spec:
  host: myedge-test.apps.example.com
  tls:
    externalCertificate:
      name: <secret-name> ①
    termination: edge
  [...]
  [...]
```

- ① Specify the actual name of your secret.

4. Create a **route** resource by running the following command:

```
$ oc apply -f <route.yaml> ①
```

- ① Specify the generated YAML filename.

If the secret exists and has a certificate/key pair, the router will serve the generated certificate if all prerequisites are met.



## NOTE

If **.spec.tls.externalCertificate** is not provided, the router will use default generated certificates.

You cannot provide the **.spec.tls.certificate** field or the **.spec.tls.key** field when using the **.spec.tls.externalCertificate** field.

## Additional resources

- For troubleshooting routes with externally managed certificates, check the OpenShift Container Platform router pod logs for errors, see [Investigating pod issues](#).

# CHAPTER 20. CONFIGURING INGRESS CLUSTER TRAFFIC

## 20.1. CONFIGURING INGRESS CLUSTER TRAFFIC OVERVIEW

OpenShift Container Platform provides the following methods for communicating from outside the cluster with services running in the cluster.

The methods are recommended, in order of preference:

- If you have HTTP/HTTPS, use an Ingress Controller.
- If you have a TLS-encrypted protocol other than HTTPS. For example, for TLS with the SNI header, use an Ingress Controller.
- Otherwise, use a Load Balancer, an External IP, or a **NodePort**.

Method	Purpose
<a href="#">Use an Ingress Controller</a>	Allows access to HTTP/HTTPS traffic and TLS-encrypted protocols other than HTTPS (for example, TLS with the SNI header).
<a href="#">Automatically assign an external IP using a load balancer service</a>	Allows traffic to non-standard ports through an IP address assigned from a pool. Most cloud platforms offer a method to start a service with a load-balancer IP address.
<a href="#">About MetalLB and the MetalLB Operator</a>	Allows traffic to a specific IP address or address from a pool on the machine network. For bare-metal installations or platforms that are like bare metal, MetalLB provides a way to start a service with a load-balancer IP address.
<a href="#">Manually assign an external IP to a service</a>	Allows traffic to non-standard ports through a specific IP address.
<a href="#">Configure a NodePort</a>	Expose a service on all nodes in the cluster.

### 20.1.1. Comparison: Fault tolerant access to external IP addresses

For the communication methods that provide access to an external IP address, fault tolerant access to the IP address is another consideration. The following features provide fault tolerant access to an external IP address.

#### IP failover

IP failover manages a pool of virtual IP address for a set of nodes. It is implemented with Keepalived and Virtual Router Redundancy Protocol (VRRP). IP failover is a layer 2 mechanism only and relies on multicast. Multicast can have disadvantages for some networks.

#### MetalLB

MetalLB has a layer 2 mode, but it does not use multicast. Layer 2 mode has a disadvantage that it transfers all traffic for an external IP address through one node.

## Manually assigning external IP addresses

You can configure your cluster with an IP address block that is used to assign external IP addresses to services. By default, this feature is disabled. This feature is flexible, but places the largest burden on the cluster or network administrator. The cluster is prepared to receive traffic that is destined for the external IP, but each customer has to decide how they want to route traffic to nodes.

## 20.2. CONFIGURING EXTERNALIPS FOR SERVICES

As a cluster administrator, you can designate an IP address block that is external to the cluster that can send traffic to services in the cluster.

This functionality is generally most useful for clusters installed on bare-metal hardware.

### 20.2.1. Prerequisites

- Your network infrastructure must route traffic for the external IP addresses to your cluster.

### 20.2.2. About ExternalIP

For non-cloud environments, OpenShift Container Platform supports the use of the `ExternalIP` facility to specify external IP addresses in the `spec.externalIPs[]` parameter of the `Service` object. A service configured with an `ExternalIP` functions similarly to a service with `type=NodePort`, whereby you traffic directs to a local node for load balancing.



#### IMPORTANT

For cloud environments, use the load balancer services for automatic deployment of a cloud load balancer to target the endpoints of a service.

After you specify a value for the parameter, OpenShift Container Platform assigns an additional virtual IP address to the service. The IP address can exist outside of the service network that you defined for your cluster.



#### WARNING

Because `ExternalIP` is disabled by default, enabling the `ExternalIP` functionality might introduce security risks for the service, because in-cluster traffic to an external IP address is directed to that service. This configuration means that cluster users could intercept sensitive traffic destined for external resources.

You can use either a MetalLB implementation or an IP failover deployment to attach an `ExternalIP` resource to a service in the following ways:

#### Automatic assignment of an external IP

OpenShift Container Platform automatically assigns an IP address from the `autoAssignCIDRs` CIDR block to the `spec.externalIPs[]` array when you create a `Service` object with `spec.type=LoadBalancer` set. For this configuration, OpenShift Container Platform implements a

cloud version of the load balancer service type and assigns IP addresses to the services. Automatic assignment is disabled by default and must be configured by a cluster administrator as described in the "Configuration for ExternalIP" section.

### Manual assignment of an external IP

OpenShift Container Platform uses the IP addresses assigned to the **spec.externalIPs[]** array when you create a **Service** object. You cannot specify an IP address that is already in use by another service.

After using either the MetalLB implementation or an IP failover deployment to host external IP address blocks, you must configure your networking infrastructure to ensure that the external IP address blocks are routed to your cluster. This configuration means that the IP address is not configured in the network interfaces from nodes. To handle the traffic, you must configure the routing and access to the external IP by using a method, such as static Address Resolution Protocol (ARP) entries.

OpenShift Container Platform extends the ExternalIP functionality in Kubernetes by adding the following capabilities:

- Restrictions on the use of external IP addresses by users through a configurable policy
- Allocation of an external IP address automatically to a service upon request

#### 20.2.3. Additional resources

- [Configuring IP failover](#)
- [About MetalLB and the MetalLB Operator](#)

#### 20.2.4. Configuration for ExternalIP

Use of an external IP address in OpenShift Container Platform is governed by the following parameters in the **Network.config.openshift.io** custom resource (CR) that is named **cluster**:

- **spec.externalIP.autoAssignCIDRs** defines an IP address block used by the load balancer when choosing an external IP address for the service. OpenShift Container Platform supports only a single IP address block for automatic assignment. This configuration requires less steps than manually assigning ExternalIPs to services, which requires managing the port space of a limited number of shared IP addresses. If you enable automatic assignment, a **Service** object with **spec.type=LoadBalancer** is allocated an external IP address.
- **spec.externalIP.policy** defines the permissible IP address blocks when manually specifying an IP address. OpenShift Container Platform does not apply policy rules to IP address blocks that you defined in the **spec.externalIP.autoAssignCIDRs** parameter.

If routed correctly, external traffic from the configured external IP address block can reach service endpoints through any TCP or UDP port that the service exposes.



#### IMPORTANT

As a cluster administrator, you must configure routing to externalIPs. You must also ensure that the IP address block you assign terminates at one or more nodes in your cluster. For more information, see [Kubernetes External IPs](#).

OpenShift Container Platform supports both the automatic and manual assignment of IP addresses, where each address is guaranteed to be assigned to a maximum of one service. This configuration ensures that each service can expose its chosen ports regardless of the ports exposed by other services.



### NOTE

To use IP address blocks defined by **autoAssignCIDRs** in OpenShift Container Platform, you must configure the necessary IP address assignment and routing for your host network.

The following YAML describes a service with an external IP address configured:

#### Example Service object with `spec.externalIPs[]` set

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: highport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
# ...
```

#### 20.2.5. Restrictions on the assignment of an external IP address

As a cluster administrator, you can specify IP address blocks to allow and to reject IP addresses for a service. Restrictions apply only to users without **cluster-admin** privileges. A cluster administrator can always set the service **spec.externalIPs[]** field to any IP address.

You configure an IP address policy by specifying Classless Inter-Domain Routing (CIDR) address blocks for the **spec.ExternalIP.policy** parameter in the **policy** object.

#### Example in JSON form of a `policy` object and its CIDR parameters

```
{
  "policy": {
    "allowedCIDRs": []}
```

```

        "rejectedCIDRs": []
    }
}

```

When configuring policy restrictions, the following rules apply:

- If **policy** is set to `{}`, creating a **Service** object with **spec.ExternalIPs[]** results in a failed service. This setting is the default for OpenShift Container Platform. The same behavior exists for **policy: null**.
- If **policy** is set and either **policy.allowedCIDRs[]** or **policy.rejectedCIDRs[]** is set, the following rules apply:
  - If **allowedCIDRs[]** and **rejectedCIDRs[]** are both set, **rejectedCIDRs[]** has precedence over **allowedCIDRs[]**.
  - If **allowedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** succeeds only if the specified IP addresses are allowed.
  - If **rejectedCIDRs[]** is set, creating a **Service** object with **spec.ExternalIPs[]** succeeds only if the specified IP addresses are not rejected.

## 20.2.6. Example policy objects

The examples in this section show different **spec.externalIP.policy** configurations.

- In the following example, the policy prevents OpenShift Container Platform from creating any service with a specified external IP address.

### Example policy to reject any value specified for Service object spec.externalIPs[]

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: {}
# ...

```

- In the following example, both the **allowedCIDRs** and **rejectedCIDRs** fields are set.

### Example policy that includes both allowed and rejected CIDR blocks

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy:
      allowedCIDRs:
        - 172.16.66.10/23

```

```

    rejectedCIDRs:
      - 172.16.66.10/24
    # ...
  
```

- In the following example, **policy** is set to `{}`. With this configuration, using the `oc get networks.config.openshift.io -o yaml` command to view the configuration means **policy** parameter does not show on the command output. The same behavior exists for **policy: null**.

#### Example policy to allow any value specified for Service object spec.externalIPs[]

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
    - cidr: 10.128.0.0/14
      hostPrefix: 23
  externalIP:
    policy: {}
  # ...
  
```

### 20.2.7. ExternalIP address block configuration

The configuration for ExternalIP address blocks is defined by a Network custom resource (CR) named **cluster**. The Network CR is part of the **config.openshift.io** API group.



#### IMPORTANT

During cluster installation, the Cluster Version Operator (CVO) automatically creates a Network CR named **cluster**. Creating any other CR objects of this type is not supported.

The following YAML describes the ExternalIP configuration:

#### Network.config.openshift.io CR named **cluster**

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    autoAssignCIDRs: [] 1
    policy: 2
    ...
  
```

- 1** Defines the IP address block in CIDR format that is available for automatic assignment of external IP addresses to a service. Only a single IP address range is allowed.
- 2** Defines restrictions on manual assignment of an IP address to a service. If no restrictions are defined, specifying the **spec.externalIP** field in a **Service** object is not allowed. By default, no restrictions are defined.

The following YAML describes the fields for the **policy** stanza:

### Network.config.openshift.io policy stanza

```
policy:
  allowedCIDRs: [] 1
  rejectedCIDRs: [] 2
```

- 1** A list of allowed IP address ranges in CIDR format.
- 2** A list of rejected IP address ranges in CIDR format.

### Example external IP configurations

Several possible configurations for external IP address pools are displayed in the following examples:

- The following YAML describes a configuration that enables automatically assigned external IP addresses:

#### Example configuration with **spec.externalIP.autoAssignCIDRs** set

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    autoAssignCIDRs:
      - 192.168.132.254/29
```

- The following YAML configures policy rules for the allowed and rejected CIDR ranges:

#### Example configuration with **spec.externalIP.policy** set

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    policy:
      allowedCIDRs:
        - 192.168.132.0/29
        - 192.168.132.8/29
      rejectedCIDRs:
        - 192.168.132.7/32
```

### 20.2.8. Configure external IP address blocks for your cluster

As a cluster administrator, you can configure the following ExternalIP settings:

- An ExternalIP address block used by OpenShift Container Platform to automatically populate the **spec.clusterIP** field for a **Service** object.
- A policy object to restrict what IP addresses may be manually assigned to the **spec.clusterIP** array of a **Service** object.

## Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

## Procedure

1. Optional: To display the current external IP configuration, enter the following command:

```
$ oc describe networks.config cluster
```

2. To edit the configuration, enter the following command:

```
$ oc edit networks.config cluster
```

3. Modify the ExternalIP configuration, as in the following example:

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: ①
  ...
```

- 1 Specify the configuration for the **externalIP** stanza.

4. To confirm the updated ExternalIP configuration, enter the following command:

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{{"\n"}}'
```

### 20.2.9. Next steps

- [Configuring ingress cluster traffic for a service external IP](#)

## 20.3. CONFIGURING INGRESS CLUSTER TRAFFIC USING AN INGRESS CONTROLLER

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses an Ingress Controller.

### 20.3.1. Using Ingress Controllers and routes

The Ingress Operator manages Ingress Controllers and wildcard DNS.

Using an Ingress Controller is the most common way to allow external access to an OpenShift Container Platform cluster.

An Ingress Controller is configured to accept external requests and proxy them based on the configured routes. This is limited to HTTP, HTTPS using SNI, and TLS using SNI, which is sufficient for web applications and services that work over TLS with SNI.

Work with your administrator to configure an Ingress Controller to accept external requests and proxy them based on the configured routes.

The administrator can create a wildcard DNS entry and then set up an Ingress Controller. Then, you can work with the edge Ingress Controller without having to contact the administrators.

By default, every Ingress Controller in the cluster can admit any route created in any project in the cluster.

The Ingress Controller:

- Has two replicas by default, which means it should be running on two worker nodes.
- Can be scaled up to have more replicas on more nodes.



#### NOTE

The procedures in this section require prerequisites performed by the cluster administrator.

### 20.3.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- You have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

### 20.3.3. Creating a project and service

If the project and service that you want to expose does not exist, create the project and then create the service.

If the project and service already exists, skip to the procedure on exposing the service to create a route.

#### Prerequisites

- Install the OpenShift CLI (**oc**) and log in as a cluster administrator.

## Procedure

1. Create a new project for your service by running the **oc new-project** command:

```
$ oc new-project <project_name>
```

2. Use the **oc new-app** command to create your service:

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n <project_name>
```

## Example output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nodejs-ex	ClusterIP	172.30.197.157	<none>	8080/TCP	70s



### NOTE

By default, the new service does not have an external IP address.

## 20.3.4. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

## Prerequisites

- You logged into OpenShift Container Platform.

## Procedure

1. Log in to the project where the service you want to expose is located:

```
$ oc project <project_name>
```

2. Run the **oc expose service** command to expose the route:

```
$ oc expose service nodejs-ex
```

## Example output

```
route.route.openshift.io/nodejs-ex exposed
```

3. To verify that the service is exposed, you can use a tool, such as **curl** to check that the service is accessible from outside the cluster.

- a. To find the hostname of the route, enter the following command:

```
$ oc get route
```

## Example output

NAME	HOST/PORT	PATH	SERVICES	PORT	TERMINATION
	WILDCARD				
nodejs-ex	nodejs-ex-myproject.example.com		nodejs-ex	8080-tcp	None

- b. To check that the host responds to a GET request, enter the following command:

## Example curl command

```
$ curl --head nodejs-ex-myproject.example.com
```

## Example output

```
HTTP/1.1 200 OK
```

```
...
```

### 20.3.5. Ingress sharding in OpenShift Container Platform

In OpenShift Container Platform, an Ingress Controller can serve all routes, or it can serve a subset of routes. By default, the Ingress Controller serves any route created in any namespace in the cluster. You can add additional Ingress Controllers to your cluster to optimize routing by creating *shards*, which are subsets of routes based on selected characteristics. To mark a route as a member of a shard, use labels in the route or namespace **metadata** field. The Ingress Controller uses *selectors*, also known as a *selection expression*, to select a subset of routes from the entire pool of routes to serve.

Ingress sharding is useful in cases where you want to load balance incoming traffic across multiple Ingress Controllers, when you want to isolate traffic to be routed to a specific Ingress Controller, or for a variety of other reasons described in the next section.

By default, each route uses the default domain of the cluster. However, routes can be configured to use the domain of the router instead.

### 20.3.6. Ingress Controller sharding

You can use Ingress sharding, also known as router sharding, to distribute a set of routes across multiple routers by adding labels to routes, namespaces, or both. The Ingress Controller uses a corresponding set of selectors to admit only the routes that have a specified label. Each Ingress shard comprises the routes that are filtered by using a given selection expression.

As the primary mechanism for traffic to enter the cluster, the demands on the Ingress Controller can be significant. As a cluster administrator, you can shard the routes to:

- Balance Ingress Controllers, or routers, with several routes to accelerate responses to changes.
- Assign certain routes to have different reliability guarantees than other routes.
- Allow certain Ingress Controllers to have different policies defined.
- Allow only specific routes to use additional features.
- Expose different routes on different addresses so that internal and external users can see different routes, for example.

- Transfer traffic from one version of an application to another during a blue-green deployment.

When Ingress Controllers are sharded, a given route is admitted to zero or more Ingress Controllers in the group. The status of a route describes whether an Ingress Controller has admitted the route. An Ingress Controller only admits a route if the route is unique to a shard.

With sharding, you can distribute subsets of routes over multiple Ingress Controllers. These subsets can be nonoverlapping, also called *traditional* sharding, or overlapping, otherwise known as *overlapped* sharding.

The following table outlines three sharding methods:

Sharding method	Description
Namespace selector	After you add a namespace selector to the Ingress Controller, all routes in a namespace that have matching labels for the namespace selector are included in the Ingress shard. Consider this method when an Ingress Controller serves all routes created in a namespace.
Route selector	After you add a route selector to the Ingress Controller, all routes with labels that match the route selector are included in the Ingress shard. Consider this method when you want an Ingress Controller to serve only a subset of routes or a specific route in a namespace.
Namespace and route selectors	Provides your Ingress Controller scope for both namespace selector and route selector methods. Consider this method when you want the flexibility of both the namespace selector and the route selector methods.

#### 20.3.6.1. Traditional sharding example

An example of a configured Ingress Controller **finops-router** that has the label selector **spec.namespaceSelector.matchExpressions** with key values set to **finance** and **ops**:

##### Example YAML definition for **finops-router**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: finops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchExpressions:
      - key: name
        operator: In
        values:
          - finance
          - ops
```

An example of a configured Ingress Controller **dev-router** that has the label selector **spec.namespaceSelector.matchLabels.name** with the key value set to **dev**:

##### Example YAML definition for **dev-router**

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: dev-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchLabels:
      name: dev

```

If all application routes are in separate namespaces, such as each labeled with **name:finance**, **name:ops**, and **name:dev**, the configuration effectively distributes your routes between the two Ingress Controllers. OpenShift Container Platform routes for console, authentication, and other purposes should not be handled.

In the previous scenario, sharding becomes a special case of partitioning, with no overlapping subsets. Routes are divided between router shards.



#### WARNING

The **default** Ingress Controller continues to serve all routes unless the **namespaceSelector** or **routeSelector** fields contain routes that are meant for exclusion. See this [Red Hat Knowledgebase solution](#) and the section "Sharding the default Ingress Controller" for more information on how to exclude routes from the default Ingress Controller.

#### 20.3.6.2. Overlapped sharding example

An example of a configured Ingress Controller **devops-router** that has the label selector **spec.namespaceSelector.matchExpressions** with key values set to **dev** and **ops**:

##### Example YAML definition for **devops-router**

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: devops-router
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchExpressions:
    - key: name
      operator: In
      values:
      - dev
      - ops

```

The routes in the namespaces labeled **name:dev** and **name:ops** are now serviced by two different Ingress Controllers. With this configuration, you have overlapping subsets of routes.

With overlapping subsets of routes you can create more complex routing rules. For example, you can divert higher priority traffic to the dedicated **finops-router** while sending lower priority traffic to **devops-router**.

### 20.3.6.3. Sharding the default Ingress Controller

After creating a new Ingress shard, there might be routes that are admitted to your new Ingress shard that are also admitted by the default Ingress Controller. This is because the default Ingress Controller has no selectors and admits all routes by default.

You can restrict an Ingress Controller from servicing routes with specific labels using either namespace selectors or route selectors. The following procedure restricts the default Ingress Controller from servicing your newly sharded **finance**, **ops**, and **dev**, routes using a namespace selector. This adds further isolation to Ingress shards.



#### IMPORTANT

You must keep all of OpenShift Container Platform's administration routes on the same Ingress Controller. Therefore, avoid adding additional selectors to the default Ingress Controller that exclude these essential routes.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as a project administrator.

#### Procedure

1. Modify the default Ingress Controller by running the following command:

```
$ oc edit ingresscontroller -n openshift-ingress-operator default
```

2. Edit the Ingress Controller to contain a **namespaceSelector** that excludes the routes with any of the **finance**, **ops**, and **dev** labels:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  namespaceSelector:
    matchExpressions:
      - key: name
        operator: NotIn
        values:
          - finance
          - ops
          - dev
```

The default Ingress Controller will no longer serve the namespaces labeled **name:finance**, **name:ops**, and **name:dev**.

#### 20.3.6.4. Ingress sharding and DNS

The cluster administrator is responsible for making a separate DNS entry for each router in a project. A router will not forward unknown routes to another router.

Consider the following example:

- Router A lives on host 192.168.0.5 and has routes with **\*.foo.com**.
- Router B lives on host 192.168.1.9 and has routes with **\*.example.com**.

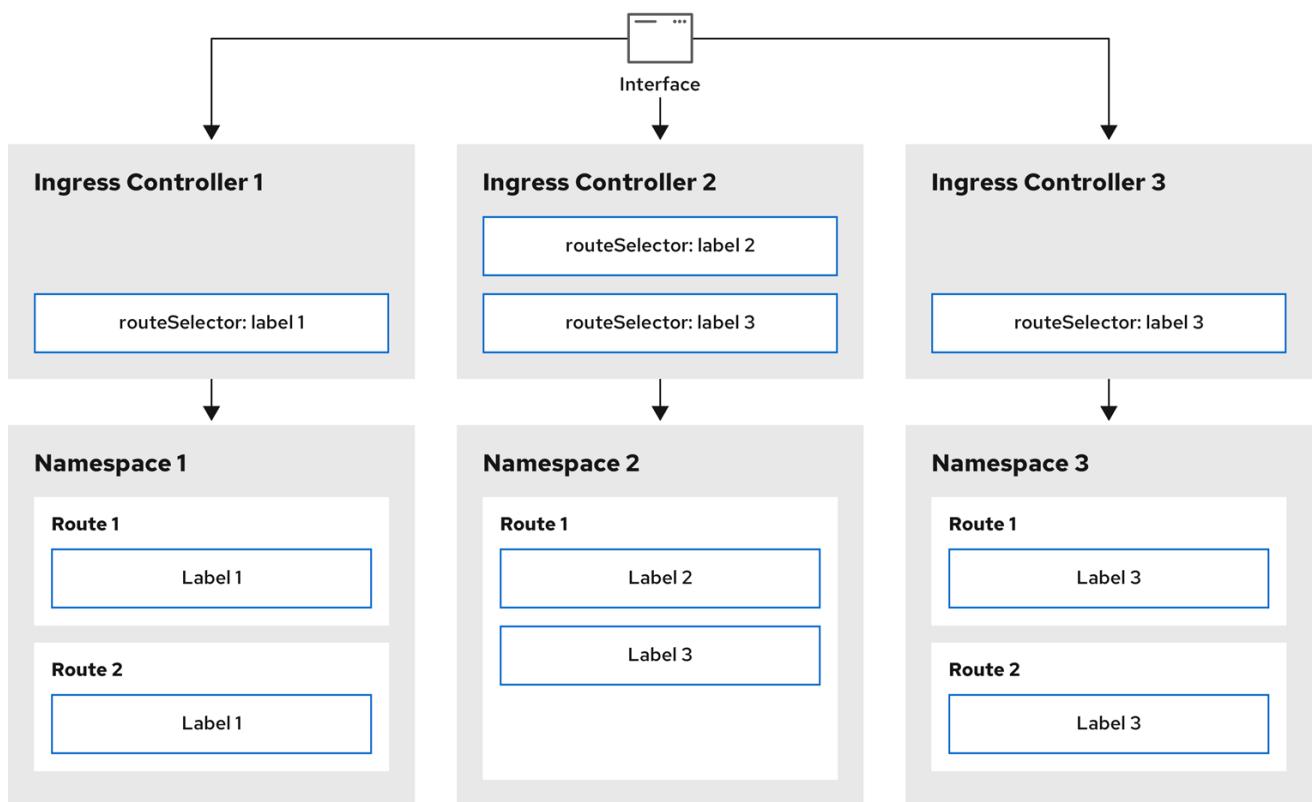
Separate DNS entries must resolve **\*.foo.com** to the node hosting Router A and **\*.example.com** to the node hosting Router B:

- **\*.foo.com A IN 192.168.0.5**
- **\*.example.com A IN 192.168.1.9**

#### 20.3.6.5. Configuring Ingress Controller sharding by using route labels

Ingress Controller sharding by using route labels means that the Ingress Controller serves any route in any namespace that is selected by the route selector.

Figure 20.1. Ingress sharding using route labels



301\_OpenShift\_0123

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

#### Procedure

1. Edit the **router-internal.yaml** file:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  routeSelector:
    matchLabels:
      type: sharded
```

- 1** Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

2. Apply the Ingress Controller **router-internal.yaml** file:

```
# oc apply -f router-internal.yaml
```

The Ingress Controller selects routes in any namespace that have the label **type: sharded**.

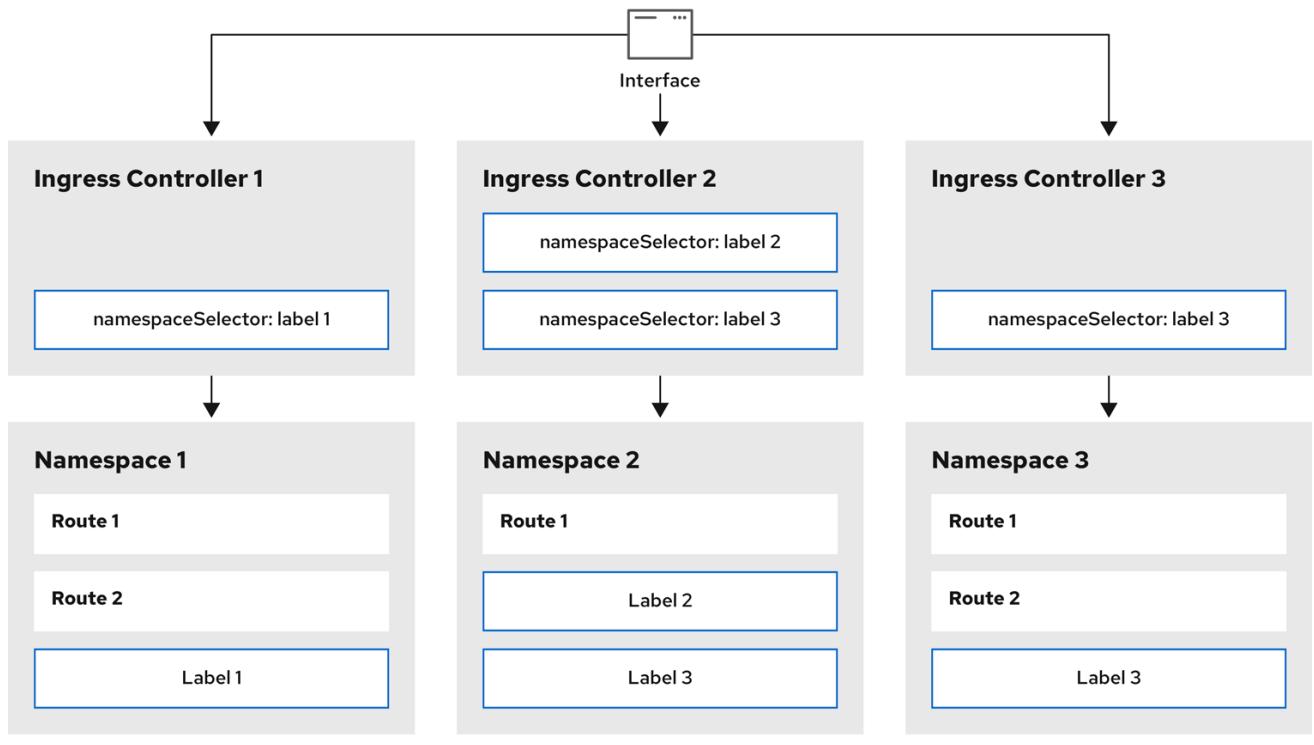
3. Create a new route using the domain configured in the **router-internal.yaml**:

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

#### 20.3.6.6. Configuring Ingress Controller sharding by using namespace labels

Ingress Controller sharding by using namespace labels means that the Ingress Controller serves any route in any namespace that is selected by the namespace selector.

Figure 20.2. Ingress sharding using namespace labels



301\_OpenShift\_0123

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

## Procedure

1. Edit the **router-internal.yaml** file:

```
$ cat router-internal.yaml
```

### Example output

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> ①
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
  
```

- 1 Specify a domain to be used by the Ingress Controller. This domain must be different from the default Ingress Controller domain.

2. Apply the Ingress Controller **router-internal.yaml** file:

```
$ oc apply -f router-internal.yaml
```

The Ingress Controller selects routes in any namespace that is selected by the namespace selector that have the label **type: sharded**.

3. Create a new route using the domain configured in the **router-internal.yaml**:

```
$ oc expose svc <service-name> --hostname <route-name>.apps-sharded.basedomain.example.net
```

### 20.3.6.7. Creating a route for Ingress Controller sharding

A route allows you to host your application at a URL. In this case, the hostname is not set and the route uses a subdomain instead. When you specify a subdomain, you automatically use the domain of the Ingress Controller that exposes the route. For situations where a route is exposed by multiple Ingress Controllers, the route is hosted at multiple URLs.

The following procedure describes how to create a route for Ingress Controller sharding, using the **hello-openshift** application as an example.

Ingress Controller sharding is useful when balancing incoming traffic load among a set of Ingress Controllers and when isolating traffic to a specific Ingress Controller. For example, company A goes to one Ingress Controller and company B to another.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- You are logged in as a project administrator.
- You have a web application that exposes a port and an HTTP or TLS endpoint listening for traffic on the port.
- You have configured the Ingress Controller for sharding.

#### Procedure

1. Create a project called **hello-openshift** by running the following command:

```
$ oc new-project hello-openshift
```

2. Create a pod in the project by running the following command:

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. Create a service called **hello-openshift** by running the following command:

```
$ oc expose pod/hello-openshift
```

4. Create a route definition called **hello-openshift-route.yaml**:

## YAML definition of the created route for sharding

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded 1
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift 2
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

- 1** Both the label key and its corresponding label value must match the ones specified in the Ingress Controller. In this example, the Ingress Controller has the label key and value **type: sharded**.
- 2** The route will be exposed using the value of the **subdomain** field. When you specify the **subdomain** field, you must leave the hostname unset. If you specify both the **host** and **subdomain** fields, then the route will use the value of the **host** field, and ignore the **subdomain** field.

5. Use **hello-openshift-route.yaml** to create a route to the **hello-openshift** application by running the following command:

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

### Verification

- Get the status of the route with the following command:

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

The resulting **Route** resource should look similar to the following:

### Example output

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
```

```

kind: Service
name: hello-openshift
status:
ingress:
- host: hello-openshift.<apps-sharded.baseDomain.example.net> 1
  routerCanonicalHostname: router-sharded.<apps-sharded.baseDomain.example.net> 2
  routerName: sharded 3

```

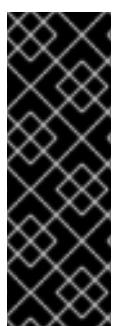
- 1** The hostname the Ingress Controller, or router, uses to expose the route. The value of the **host** field is automatically determined by the Ingress Controller, and uses its domain. In this example, the domain of the Ingress Controller is **<apps-sharded.baseDomain.example.net>**.
- 2** The hostname of the Ingress Controller.
- 3** The name of the Ingress Controller. In this example, the Ingress Controller has the name **sharded**.

## Additional resources

- [Baseline Ingress Controller \(router\) performance](#)
- [Configuring the Ingress Controller](#)
- [Installing a cluster on bare metal](#)
- [Installing a cluster on vSphere](#)
- [About network policy](#)

## 20.4. CONFIGURING THE INGRESS CONTROLLER ENDPOINT PUBLISHING STRATEGY

The **endpointPublishingStrategy** is used to publish the Ingress Controller endpoints to other networks, enable load balancer integrations, and provide access to other systems.



### IMPORTANT

On Red Hat OpenStack Platform (RHOSP), the **LoadBalancerService** endpoint publishing strategy is supported only if a cloud provider is configured to create health monitors. For RHOSP 16.2, this strategy is possible only if you use the Amphora Octavia provider.

For more information, see the "Setting RHOSP Cloud Controller Manager options" section of the RHOSP installation documentation.

### 20.4.1. Ingress Controller endpoint publishing strategy

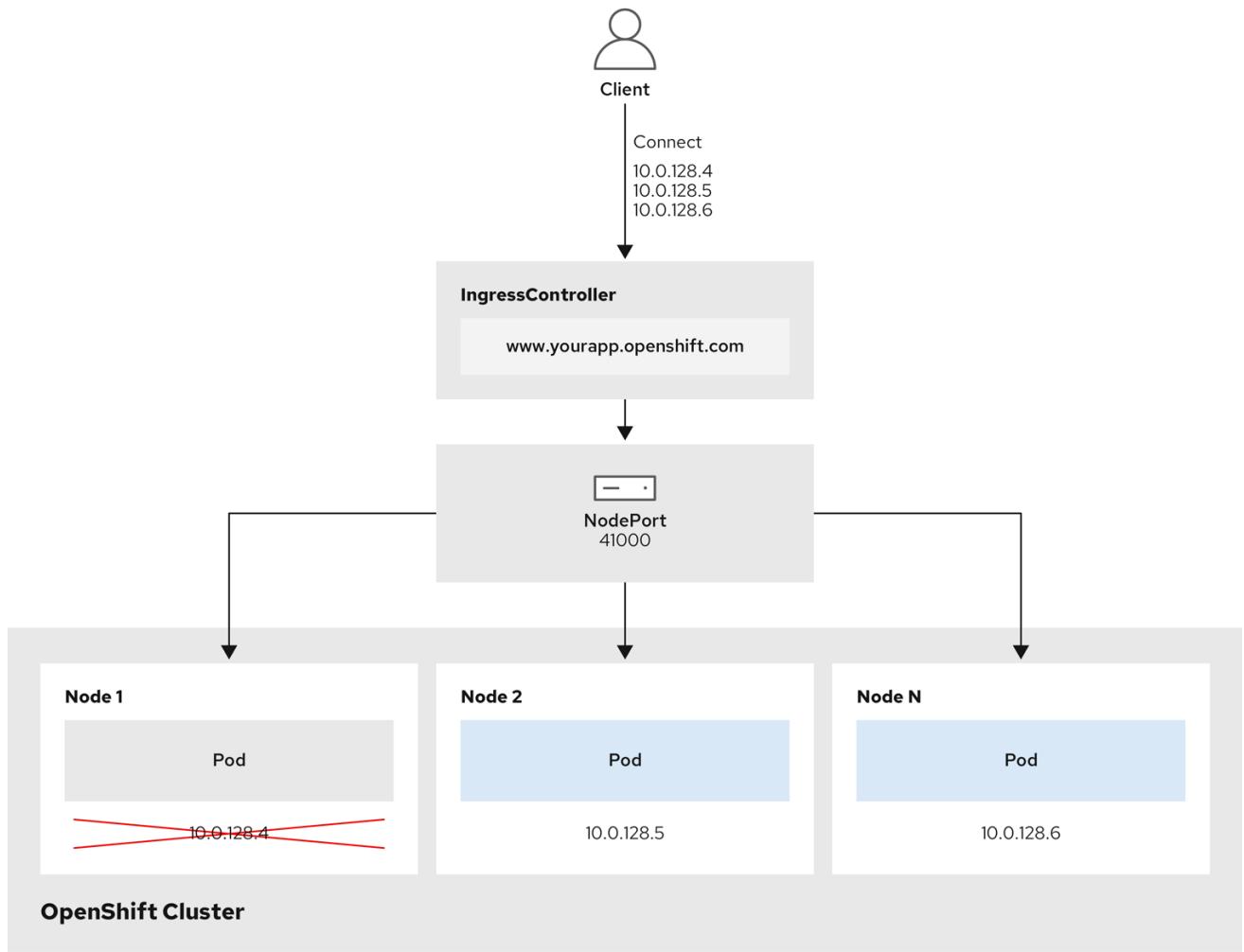
#### NodePortService endpoint publishing strategy

The **NodePortService** endpoint publishing strategy publishes the Ingress Controller using a Kubernetes NodePort service.

In this configuration, the Ingress Controller deployment uses container networking. A **NodePortService**

is created to publish the deployment. The specific node ports are dynamically allocated by OpenShift Container Platform; however, to support static port allocations, your changes to the node port field of the managed **NodePortService** are preserved.

Figure 20.3. Diagram of NodePortService



202\_OpenShift\_0222

The preceding graphic shows the following concepts pertaining to OpenShift Container Platform Ingress NodePort endpoint publishing strategy:

- All the available nodes in the cluster have their own, externally accessible IP addresses. The service running in the cluster is bound to the unique NodePort for all the nodes.
- When the client connects to a node that is down, for example, by connecting the **10.0.128.4** IP address in the graphic, the node port directly connects the client to an available node that is running the service. In this scenario, no load balancing is required. As the image shows, the **10.0.128.4** address is down and another IP address must be used instead.



## NOTE

The Ingress Operator ignores any updates to `.spec.ports[].nodePort` fields of the service.

By default, ports are allocated automatically and you can access the port allocations for integrations. However, sometimes static port allocations are necessary to integrate with existing infrastructure which may not be easily reconfigured in response to dynamic ports. To achieve integrations with static node ports, you can update the managed service resource directly.

For more information, see the [Kubernetes Services documentation on NodePort](#).

### HostNetwork endpoint publishing strategy

The **HostNetwork** endpoint publishing strategy publishes the Ingress Controller on node ports where the Ingress Controller is deployed.

An Ingress Controller with the **HostNetwork** endpoint publishing strategy can have only one pod replica per node. If you want  $n$  replicas, you must use at least  $n$  nodes where those replicas can be scheduled. Because each pod replica requests ports **80** and **443** on the node host where it is scheduled, a replica cannot be scheduled to a node if another pod on the same node is using those ports.

The **HostNetwork** object has a **hostNetwork** field with the following default values for the optional binding ports: **httpPort: 80**, **httpsPort: 443**, and **statsPort: 1936**. By specifying different binding ports for your network, you can deploy multiple Ingress Controllers on the same node for the **HostNetwork** strategy.

### Example

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: internal
  namespace: openshift-ingress-operator
spec:
  domain: example.com
  endpointPublishingStrategy:
    type: HostNetwork
    hostNetwork:
      httpPort: 80
      httpsPort: 443
      statsPort: 1936
```

#### 20.4.1.1. Configuring the Ingress Controller endpoint publishing scope to Internal

When a cluster administrator installs a new cluster without specifying that the cluster is private, the default Ingress Controller is created with a **scope** set to **External**. Cluster administrators can change an **External** scoped Ingress Controller to **Internal**.

### Prerequisites

- You installed the **oc** CLI.

### Procedure

- To change an **External** scoped Ingress Controller to **Internal**, enter the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":{"scope":"Internal"}}}}'
```

- To check the status of the Ingress Controller, enter the following command:

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

- The **Progressing** status condition indicates whether you must take further action. For example, the status condition can indicate that you need to delete the service by entering the following command:

```
$ oc -n openshift-ingress delete services/router-default
```

If you delete the service, the Ingress Operator recreates it as **Internal**.

#### 20.4.1.2. Configuring the Ingress Controller endpoint publishing scope to External

When a cluster administrator installs a new cluster without specifying that the cluster is private, the default Ingress Controller is created with a **scope** set to **External**.

The Ingress Controller's scope can be configured to be **Internal** during installation or after, and cluster administrators can change an **Internal** Ingress Controller to **External**.



#### IMPORTANT

On some platforms, it is necessary to delete and recreate the service.

Changing the scope can cause disruption to Ingress traffic, potentially for several minutes. This applies to platforms where it is necessary to delete and recreate the service, because the procedure can cause OpenShift Container Platform to deprovision the existing service load balancer, provision a new one, and update DNS.

#### Prerequisites

- You installed the **oc** CLI.

#### Procedure

- To change an **Internal** scoped Ingress Controller to **External**, enter the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/private --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":{"scope":"External"}}}}'
```

- To check the status of the Ingress Controller, enter the following command:

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

- The **Progressing** status condition indicates whether you must take further action. For example, the status condition can indicate that you need to delete the service by entering the following command:

```
$ oc -n openshift-ingress delete services/router-default
```

If you delete the service, the Ingress Operator recreates it as **External**.

#### 20.4.1.3. Adding a single NodePort service to an Ingress Controller

Instead of creating a **NodePort**-type **Service** for each project, you can create a custom Ingress Controller to use the **NodePortService** endpoint publishing strategy. To prevent port conflicts, consider this configuration for your Ingress Controller when you want to apply a set of routes, through Ingress sharding, to nodes that might already have a **HostNetwork** Ingress Controller.

Before you set a **NodePort**-type **Service** for each project, read the following considerations:

- You must create a wildcard DNS record for the Nodeport Ingress Controller domain. A Nodeport Ingress Controller route can be reached from the address of a worker node. For more information about the required DNS records for routes, see "User-provisioned DNS requirements".
- You must expose a route for your service and specify the **--hostname** argument for your custom Ingress Controller domain.
- You must append the port that is assigned to the **NodePort**-type **Service** in the route so that you can access application pods.

#### Prerequisites

- You installed the OpenShift CLI (**oc**).
- Logged in as a user with **cluster-admin** privileges.
- You created a wildcard DNS record.

#### Procedure

1. Create a custom resource (CR) file for the Ingress Controller:

##### Example of a CR file that defines information for the **IngressController** object

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: <custom_ic_name> ①
    namespace: openshift-ingress-operator
  spec:
    replicas: 1
    domain: <custom_ic_domain_name> ②
    nodePlacement:
      nodeSelector:
        matchLabels:
```

```

<key>: <value> ③
namespaceSelector:
  matchLabels:
    <key>: <value> ④
  endpointPublishingStrategy:
    type: NodePortService
# ...

```

- 1 Specify the a custom **name** for the **IngressController** CR.
- 2 The DNS name that the Ingress Controller services. As an example, the default ingresscontroller domain is **apps.ipi-cluster.example.com**, so you would specify the **<custom\_ic\_domain\_name>** as **nodeportsvc.ipi-cluster.example.com**.
- 3 Specify the label for the nodes that include the custom Ingress Controller.
- 4 Specify the label for a set of namespaces. Substitute **<key>:<value>** with a map of key-value pairs where **<key>** is a unique name for the new label and **<value>** is its value. For example: **ingresscontroller: custom-ic**.

2. Add a label to a node by using the **oc label node** command:

```
$ oc label node <node_name> <key>=<value> ①
```

- 1 Where **<value>** must match the key-value pair specified in the **nodePlacement** section of your **IngressController** CR.

3. Create the **IngressController** object:

```
$ oc create -f <ingress_controller_cr>.yaml
```

4. Find the port for the service created for the **IngressController** CR:

```
$ oc get svc -n openshift-ingress
```

**Example output that shows port 80:32432/TCP for the router-nodeport-custom-ic3 service**

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
router-internal-default	ClusterIP	172.30.195.74	<none>	
80/TCP,443/TCP,1936/TCP			223d	
router-nodeport-custom-ic3	NodePort	172.30.109.219	<none>	
80:32432/TCP,443:31366/TCP,1936:30499/TCP			155m	

5. To create a new project, enter the following command:

```
$ oc new-project <project_name>
```

6. To label the new namespace, enter the following command:

```
$ oc label namespace <project_name> <key>=<value> ①
```

- ① Where **<key>=<value>** must match the value in the **namespaceSelector** section of your Ingress Controller CR.

7. Create a new application in your cluster:

```
$ oc new-app --image=<image_name> ①
```

- ① An example of **<image\_name>** is [quay.io/openshifttest/hello-openshift:multiarch](https://quay.io/repository/quay.io/openshifttest/hello-openshift:multiarch).

8. Create a **Route** object for a service, so that the pod can use the service to expose the application external to the cluster.

```
$ oc expose svc/<service_name> --hostname=<svc_name>-<project_name>.<custom_ic_domain_name> ①
```



#### NOTE

You must specify the domain name of your custom Ingress Controller in the **--hostname** argument. If you do not do this, the Ingress Operator uses the default Ingress Controller to serve all the routes for your cluster.

9. Check that the route has the **Admitted** status and that it includes metadata for the custom Ingress Controller:

```
$ oc get route/hello-openshift -o json | jq '.status.ingress'
```

#### Example output

```
# ...
{
  "conditions": [
    {
      "lastTransitionTime": "2024-05-17T18:25:41Z",
      "status": "True",
      "type": "Admitted"
    }
  ],
  [
    {
      "host": "hello-openshift.nodeportsvc.ipi-cluster.example.com",
      "routerCanonicalHostname": "router-nodeportsvc.nodeportsvc.ipi-cluster.example.com",
      "routerName": "nodeportsvc", "wildcardPolicy": "None"
    }
  ],
}
```

10. Update the default **IngressController** CR to prevent the default Ingress Controller from managing the **NodePort**-type **Service**. The default Ingress Controller will continue to monitor all other cluster traffic.

```
$ oc patch --type=merge -n openshift-ingress-operator ingresscontroller/default --patch
'{"spec":{"namespaceSelector":{"matchExpressions":[{"key":"<key>","operator":"NotIn","values":["<value>"]}]}}'
```

## Verification

- Verify that the DNS entry can route inside and outside of your cluster by entering the following command. The command outputs the IP address of the node that received the label from running the **oc label node** command earlier in the procedure.

```
$ dig +short <svc_name>-<project_name>.<custom_ic_domain_name>
```

- To verify that your cluster uses the IP addresses from external DNS servers for DNS resolution, check the connection of your cluster by entering the following command:

```
$ curl <svc_name>-<project_name>.<custom_ic_domain_name>:<port> ①
```

① Where **<port>** is the node port from the **NodePort**-type **Service**. Based on example output from the **oc get svc -n openshift-ingress** command, the **80:32432/TCP** HTTP route means that **32432** is the node port.

## Output example

```
Hello OpenShift!
```

### 20.4.2. Additional resources

- [Ingress Controller configuration parameters](#)
- [Setting RHOSP Cloud Controller Manager options](#)
- [User-provisioned DNS requirements](#)

## 20.5. CONFIGURING INGRESS CLUSTER TRAFFIC USING A LOAD BALANCER

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a load balancer.

### 20.5.1. Using a load balancer to get traffic into the cluster

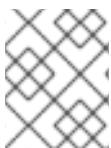
If you do not need a specific external IP address, you can configure a load balancer service to allow external access to an OpenShift Container Platform cluster.

A load balancer service allocates a unique IP. The load balancer has a single edge router IP, which can be a virtual IP (VIP), but is still a single machine for initial load balancing.



#### NOTE

If a pool is configured, it is done at the infrastructure level, not by a cluster administrator.

**NOTE**

The procedures in this section require prerequisites performed by the cluster administrator.

### 20.5.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

### 20.5.3. Creating a project and service

If the project and service that you want to expose does not exist, create the project and then create the service.

If the project and service already exists, skip to the procedure on exposing the service to create a route.

#### Prerequisites

- Install the OpenShift CLI (**oc**) and log in as a cluster administrator.

#### Procedure

1. Create a new project for your service by running the **oc new-project** command:

```
$ oc new-project <project_name>
```

2. Use the **oc new-app** command to create your service:

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n <project_name>
```

#### Example output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nodejs-ex	ClusterIP	172.30.197.157	<none>	8080/TCP	70s

**NOTE**

By default, the new service does not have an external IP address.

#### 20.5.4. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

##### Prerequisites

- You logged into OpenShift Container Platform.

##### Procedure

1. Log in to the project where the service you want to expose is located:

```
$ oc project <project_name>
```

2. Run the **oc expose service** command to expose the route:

```
$ oc expose service nodejs-ex
```

##### Example output

```
route.route.openshift.io/nodejs-ex exposed
```

3. To verify that the service is exposed, you can use a tool, such as **curl** to check that the service is accessible from outside the cluster.

- a. To find the hostname of the route, enter the following command:

```
$ oc get route
```

##### Example output

NAME	HOST/PORT	PATH	SERVICES	PORT	TERMINATION
WILDCARD					
nodejs-ex	nodejs-ex-myproject.example.com		nodejs-ex	8080-tcp	None

- b. To check that the host responds to a GET request, enter the following command:

##### Example curl command

```
$ curl --head nodejs-ex-myproject.example.com
```

##### Example output

```
HTTP/1.1 200 OK  
...
```

#### 20.5.5. Creating a load balancer service

Use the following procedure to create a load balancer service.

## Prerequisites

- Make sure that the project and service you want to expose exist.
- Your cloud provider supports load balancers.

## Procedure

To create a load balancer service:

1. Log in to OpenShift Container Platform.
2. Load the project where the service you want to expose is located.

```
$ oc project project1
```

3. Open a text file on the control plane node and paste the following text, editing the file as needed:

### Sample load balancer configuration file

```
apiVersion: v1
kind: Service
metadata:
  name: egress-2 ①
spec:
  ports:
    - name: db
      port: 3306 ②
  loadBalancerIP:
  loadBalancerSourceRanges: ③
    - 10.0.0.0/8
    - 192.168.0.0/16
  type: LoadBalancer ④
  selector:
    name: mysql ⑤
```

- ① Enter a descriptive name for the load balancer service.
- ② Enter the same port that the service you want to expose is listening on.
- ③ Enter a list of specific IP addresses to restrict traffic through the load balancer. This field is ignored if the cloud-provider does not support the feature.
- ④ Enter **Loadbalancer** as the type.
- ⑤ Enter the name of the service.



## NOTE

To restrict the traffic through the load balancer to specific IP addresses, it is recommended to use the Ingress Controller field **spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges**. Do not set the **loadBalancerSourceRanges** field.

4. Save and exit the file.
5. Run the following command to create the service:

```
$ oc create -f <file-name>
```

For example:

```
$ oc create -f mysql-lb.yaml
```

6. Execute the following command to view the new service:

```
$ oc get svc
```

### Example output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
egress-2	LoadBalancer	172.30.22.226	ad42f5d8b303045-487804948.example.com	3306:30357/TCP 15m

The service has an external IP address automatically assigned if there is a cloud provider enabled.

7. On the master, use a tool, such as cURL, to make sure you can reach the service using the public IP address:

```
$ curl <public-ip>:<port>
```

For example:

```
$ curl 172.29.121.74:3306
```

The examples in this section use a MySQL service, which requires a client application. If you get a string of characters with the **Got packets out of order** message, you are connecting with the service:

If you have a MySQL client, log in with the standard CLI command:

```
$ mysql -h 172.30.131.89 -u admin -p
```

### Example output

```
Enter password:  
Welcome to the MariaDB monitor. Commands end with ; or \g.
```

MySQL [(none)]>

## 20.6. CONFIGURING INGRESS CLUSTER TRAFFIC ON AWS

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses load balancers on AWS, specifically a Network Load Balancer (NLB) or a Classic Load Balancer (CLB). Both types of load balancers can forward the client's IP address to the node, but a CLB requires proxy protocol support, which OpenShift Container Platform automatically enables.

There are two ways to configure an Ingress Controller to use an NLB:

1. By force replacing the Ingress Controller that is currently using a CLB. This deletes the **IngressController** object and an outage will occur while the new DNS records propagate and the NLB is being provisioned.
2. By editing an existing Ingress Controller that uses a CLB to use an NLB. This changes the load balancer without having to delete and recreate the **IngressController** object.

Both methods can be used to switch from an NLB to a CLB.

You can configure these load balancers on a new or existing AWS cluster.

### 20.6.1. Configuring Classic Load Balancer timeouts on AWS

OpenShift Container Platform provides a method for setting a custom timeout period for a specific route or Ingress Controller. Additionally, an AWS Classic Load Balancer (CLB) has its own timeout period with a default time of 60 seconds.

If the timeout period of the CLB is shorter than the route timeout or Ingress Controller timeout, the load balancer can prematurely terminate the connection. You can prevent this problem by increasing both the timeout period of the route and CLB.

#### 20.6.1.1. Configuring route timeouts

You can configure the default timeouts for an existing route when you have services in need of a low timeout, which is required for Service Level Availability (SLA) purposes, or a high timeout, for cases with a slow back end.

#### Prerequisites

- You need a deployed Ingress Controller on a running cluster.

#### Procedure

1. Using the **oc annotate** command, add the timeout to the route:

```
$ oc annotate route <route_name> \
--overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> ①
```

① Supported time units are microseconds (us), milliseconds (ms), seconds (s), minutes (m), hours (h), or days (d).

The following example sets a timeout of two seconds on a route named **myroute**:

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

### 20.6.1.2. Configuring Classic Load Balancer timeouts

You can configure the default timeouts for a Classic Load Balancer (CLB) to extend idle connections.

#### Prerequisites

- You must have a deployed Ingress Controller on a running cluster.

#### Procedure

1. Set an AWS connection idle timeout of five minutes for the default **ingresscontroller** by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
--type=merge --patch='{"spec":{"endpointPublishingStrategy": \
{"type":"LoadBalancerService", "loadBalancer": \
{"scope":"External", "providerParameters":{"type":"AWS", "aws": \
{"type":"Classic", "classicLoadBalancer": \
{"connectionIdleTimeout":"5m"}}}}}}'
```

2. Optional: Restore the default value of the timeout by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
--type=merge --patch='{"spec":{"endpointPublishingStrategy": \
{"loadBalancer":{"providerParameters":{"aws":{"classicLoadBalancer": \
{"connectionIdleTimeout":null}}}}}}}'
```



#### NOTE

You must specify the **scope** field when you change the connection timeout value unless the current scope is already set. When you set the **scope** field, you do not need to do so again if you restore the default timeout value.

### 20.6.2. Configuring ingress cluster traffic on AWS using a Network Load Balancer

OpenShift Container Platform provides methods for communicating from outside the cluster with services that run in the cluster. One such method uses a Network Load Balancer (NLB). You can configure an NLB on a new or existing AWS cluster.

#### 20.6.2.1. Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer

You can switch the Ingress Controller that is using a Classic Load Balancer (CLB) to one that uses a Network Load Balancer (NLB) on AWS.

Switching between these load balancers will not delete the **IngressController** object.

**WARNING**

This procedure might cause the following issues:

- An outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.
- Leaked load balancer resources due to a change in the annotation of the service.

**Procedure**

1. Modify the existing Ingress Controller that you want to switch to using an NLB. This example assumes that your default Ingress Controller has an **External** scope and no other customizations:

**Example ingresscontroller.yaml file**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
  type: LoadBalancerService
```

**NOTE**

If you do not specify a value for the **spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type** field, the Ingress Controller uses the **spec.loadBalancer.platform.aws.type** value from the cluster **Ingress** configuration that was set during installation.

**TIP**

If your Ingress Controller has other customizations that you want to update, such as changing the domain, consider force replacing the Ingress Controller definition file instead.

2. Apply the changes to the Ingress Controller YAML file by running the command:

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

Expect several minutes of outages while the Ingress Controller updates.

### 20.6.2.2. Switching the Ingress Controller from using a Network Load Balancer to a Classic Load Balancer

You can switch the Ingress Controller that is using a Network Load Balancer (NLB) to one that uses a Classic Load Balancer (CLB) on AWS.

Switching between these load balancers will not delete the **IngressController** object.



#### WARNING

This procedure might cause an outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.

#### Procedure

1. Modify the existing Ingress Controller that you want to switch to using a CLB. This example assumes that your default Ingress Controller has an **External** scope and no other customizations:

#### Example ingresscontroller.yaml file

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: Classic
  type: LoadBalancerService
```



#### NOTE

If you do not specify a value for the **spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type** field, the Ingress Controller uses the **spec.loadBalancer.platform.aws.type** value from the cluster **Ingress** configuration that was set during installation.

**TIP**

If your Ingress Controller has other customizations that you want to update, such as changing the domain, consider force replacing the Ingress Controller definition file instead.

2. Apply the changes to the Ingress Controller YAML file by running the command:

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

Expect several minutes of outages while the Ingress Controller updates.

#### 20.6.2.3. Replacing Ingress Controller Classic Load Balancer with Network Load Balancer

You can replace an Ingress Controller that is using a Classic Load Balancer (CLB) with one that uses a Network Load Balancer (NLB) on AWS.

**WARNING**

This procedure might cause the following issues:

- An outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.
- Leaked load balancer resources due to a change in the annotation of the service.

#### Procedure

1. Create a file with a new default Ingress Controller. The following example assumes that your default Ingress Controller has an **External** scope and no other customizations:

##### Example `ingresscontroller.yaml` file

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
  type: LoadBalancerService
```

If your default Ingress Controller has other customizations, ensure that you modify the file accordingly.

## TIP

If your Ingress Controller has no other customizations and you are only updating the load balancer type, consider following the procedure detailed in "Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer".

- Force replace the Ingress Controller YAML file:

```
$ oc replace --force --wait -f ingresscontroller.yaml
```

Wait until the Ingress Controller is replaced. Expect several of minutes of outages.

### 20.6.2.4. Configuring an Ingress Controller Network Load Balancer on an existing AWS cluster

You can create an Ingress Controller backed by an AWS Network Load Balancer (NLB) on an existing cluster.

#### Prerequisites

- You must have an installed AWS cluster.
- PlatformStatus** of the infrastructure resource must be AWS.
  - To verify that the **PlatformStatus** is AWS, run:

```
$ oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.type}'  
AWS
```

#### Procedure

Create an Ingress Controller backed by an AWS NLB on an existing cluster.

- Create the Ingress Controller manifest:

```
$ cat ingresscontroller-aws-nlb.yaml
```

#### Example output

```
apiVersion: operator.openshift.io/v1  
kind: IngressController  
metadata:  
  name: $my_ingress_controller①  
  namespace: openshift-ingress-operator  
spec:  
  domain: $my_unique_ingress_domain②  
  endpointPublishingStrategy:  
    type: LoadBalancerService  
  loadBalancer:  
    scope: External③  
  providerParameters:
```

```
type: AWS
aws:
  type: NLB
```

- 1** Replace **\$my\_ingress\_controller** with a unique name for the Ingress Controller.
- 2** Replace **\$my\_unique\_ingress\_domain** with a domain name that is unique among all Ingress Controllers in the cluster. This variable must be a subdomain of the DNS name **<clusternamespace>. <domain>**.
- 3** You can replace **External** with **Internal** to use an internal NLB.

2. Create the resource in the cluster:

```
$ oc create -f ingresscontroller-aws-nlb.yaml
```



### IMPORTANT

Before you can configure an Ingress Controller NLB on a new AWS cluster, you must complete the [Creating the installation configuration file](#) procedure.

#### 20.6.2.5. Configuring an Ingress Controller Network Load Balancer on a new AWS cluster

You can create an Ingress Controller backed by an AWS Network Load Balancer (NLB) on a new cluster.

##### Prerequisites

- Create the **install-config.yaml** file and complete any modifications to it.

##### Procedure

Create an Ingress Controller backed by an AWS NLB on a new cluster.

1. Change to the directory that contains the installation program and create the manifests:

```
$ ./openshift-install create manifests --dir <installation_directory> 1
```

- 1** For **<installation\_directory>**, specify the name of the directory that contains the **install-config.yaml** file for your cluster.

2. Create a file that is named **cluster-ingress-default-ingresscontroller.yaml** in the **<installation\_directory>/manifests** directory:

```
$ touch <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml 1
```

- 1** For **<installation\_directory>**, specify the directory name that contains the **manifests** directory for your cluster.

After creating the file, several network configuration files are in the **manifests**/ directory, as shown:

```
$ ls <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml
```

## Example output

```
cluster-ingress-default-ingresscontroller.yaml
```

3. Open the **cluster-ingress-default-ingresscontroller.yaml** file in an editor and enter a custom resource (CR) that describes the Operator configuration you want:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
        aws:
          type: NLB
    type: LoadBalancerService
```

4. Save the **cluster-ingress-default-ingresscontroller.yaml** file and quit the text editor.
5. Optional: Back up the **manifests/cluster-ingress-default-ingresscontroller.yaml** file. The installation program deletes the **manifests**/ directory when creating the cluster.

### 20.6.2.6. Choosing subnets while creating a LoadBalancerService Ingress Controller

You can manually specify load balancer subnets for Ingress Controllers in an existing cluster. By default, the load balancer subnets are automatically discovered by AWS, but specifying them in the Ingress Controller overrides this, allowing for manual control.

#### Prerequisites

- You must have an installed AWS cluster.
- You must know the names or IDs of the subnets to which you intend to map your **IngressController**.

#### Procedure

1. Create a custom resource (CR) file.

Create a YAML file (e.g., **sample-ingress.yaml**) with the following content:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name>
spec:
  domain: <domain>
```

```
endpointPublishingStrategy:
  type: LoadBalancerService
  loadBalancer:
    scope: External
  dnsManagementPolicy: Managed
```

2. Create a custom resource (CR) file.

Add subnets to your YAML file:

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: <name> ①
  namespace: openshift-ingress-operator
spec:
  domain: <domain> ②
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: External
  providerParameters:
    type: AWS
    aws:
      type: Classic
    classicLoadBalancer: ③
      subnets:
        ids: ④
        - <subnet> ⑤
        - <subnet>
        - <subnet>
  dnsManagementPolicy: Managed
```

- 1 Replace **<name>** with a name for the **IngressController**.
- 2 Replace **<domain>** with the DNS name serviced by the **IngressController**.
- 3 You can also use the **networkLoadBalancer** field if using an NLB.
- 4 You can optionally specify a subnet by name using the **names** field instead of specifying the subnet by ID.
- 5 Specify subnet IDs (or names if you using **names**).



### IMPORTANT

You can specify a maximum of one subnet per availability zone. Only provide public subnets for external Ingress Controllers and private subnets for internal Ingress Controllers.

3. Apply the CR file.

- a. Save the file and apply it using the OpenShift CLI (**oc**).

```
$ oc apply -f sample-ingress.yaml
```

- b. Confirm the load balancer was provisioned successfully by checking the **IngressController** conditions.

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath="{.status.conditions}" | yq -PC
```

#### 20.6.2.7. Updating the subnets on an existing Ingress Controller

You can update an **IngressController** with manually specified load balancer subnets in OpenShift Container Platform to avoid any disruptions, to maintain the stability of your services, and to ensure that your network configuration aligns with your specific requirements. The following procedures show you how to select and apply new subnets, verify the configuration changes, and confirm successful load balancer provisioning.



#### WARNING

This procedure may cause an outage that can last several minutes due to new DNS records propagation, new load balancers provisioning, and other factors. IP addresses and canonical names of the Ingress Controller load balancer might change after applying this procedure.

#### Procedure

To update an **IngressController** with manually specified load balancer subnets, you can follow these steps:

1. Modify the existing IngressController to update to the new subnets.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: <name> ①
  namespace: openshift-ingress-operator
spec:
  domain: <domain> ②
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: External
    providerParameters:
      type: AWS
      aws:
        type: Classic ③
        classicLoadBalancer: ④
        subnets:
          ids: ⑤
```

- <updated\_subnet> ⑥  
 - <updated\_subnet>  
 - <updated\_subnet>

- 1 Replace <name> with a name for the **IngressController**.
- 2 Replace <domain> with the DNS name serviced by the **IngressController**.
- 3 Specify updated subnet IDs (or names if you using **names**).
- 4 You can also use the **networkLoadBalancer** field if using an NLB.
- 5 You can optionally specify a subnet by name using the **names** field instead of specifying the subnet by ID.
- 6 Update subnet IDs (or names if you are using **names**).



### IMPORTANT

You can specify a maximum of one subnet per availability zone. Only provide public subnets for external Ingress Controllers and private subnets for internal Ingress Controllers.

2. Examine the **Progressing** condition on the **IngressController** for instructions on how to apply the subnet updates by running the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator subnets -o jsonpath=".status.conditions[?(@.type==\"Progressing\")]{}" | yq -PC
```

### Example output

```
lastTransitionTime: "2024-11-25T20:19:31Z"
message: 'One or more status conditions indicate progressing:
LoadBalancerProgressing=True (OperandsProgressing: One or more managed resources
are progressing: The IngressController subnets were changed from [...] to [...]. To effectuate
this change, you must delete the service: `oc -n openshift-ingress delete svc/router-<name>` ;
the service load-balancer will then be deprovisioned and a new one created. This will most
likely cause the new load-balancer to have a different host name and IP address and cause
disruption. To return to the previous state, you can revert the change to the
IngressController: [...]')
reason: IngressControllerProgressing
status: "True"
type: Progressing
```

3. To apply the update, delete the service associated with the Ingress controller by running the following command:

```
$ oc -n openshift-ingress delete svc/router-<name>
```

### Verification

- To confirm that the load balancer was provisioned successfully, check the **IngressController** conditions by running the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath=".status.conditions" | yq -PC
```

### 20.6.2.8. Configuring AWS Elastic IP (EIP) addresses for a Network Load Balancer (NLB)

You can specify static IPs, otherwise known as elastic IPs, for your network load balancer (NLB) in the Ingress Controller. This is useful in situations where you want to configure appropriate firewall rules for your cluster network.

#### Prerequisites

- You must have an installed AWS cluster.
- You must know the names or IDs of the subnets to which you intend to map your **IngressController**.

#### Procedure

- 1 Create a YAML file that contains the following content:

##### **sample-ingress.yaml**

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> ①
spec:
  domain: <domain> ②
  endpointPublishingStrategy:
    loadBalancer:
      scope: External ③
      type: LoadBalancerService
      providerParameters:
        type: AWS
        aws:
          type: NLB
          networkLoadBalancer:
            subnets: ④
              ids:
                - <subnet_ID>
              names:
                - <subnet_A>
                - <subnet_B>
            eipAllocations: ⑤
              - <eipalloc_A>
              - <eipalloc_B>
              - <eipalloc_C>
```

- 1 Replace the **<name>** placeholder with a name for the Ingress Controller.
- 2 Replace the **<domain>** placeholder with the DNS name serviced by the Ingress Controller.

- 3 The scope must be set to the value **External** and be Internet-facing in order to allocate EIPs.
- 4 Specify the IDs and names for your subnets. The total number of IDs and names must be equal to your allocated EIPs.
- 5 Specify the EIP addresses.



### IMPORTANT

You can specify a maximum of one subnet per availability zone. Only provide public subnets for external Ingress Controllers. You can associate one EIP address per subnet.

- 2 Save and apply the CR file by entering the following command:

```
$ oc apply -f sample-ingress.yaml
```

### Verification

- 1 Confirm the load balancer was provisioned successfully by checking the **IngressController** conditions by running the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath=".status.conditions" | yq -PC
```

### 20.6.3. Additional resources

- [Installing a cluster on AWS with network customizations](#) .
- For more information on support for NLBs, see [Network Load Balancer support on AWS](#).
- For more information on proxy protocol support for CLBs, see [Configure proxy protocol support for your Classic Load Balancer](#)

## 20.7. CONFIGURING INGRESS CLUSTER TRAFFIC FOR A SERVICE EXTERNAL IP

You can use either a MetalLB implementation or an IP failover deployment to attach an ExternalIP resource to a service so that the service is available to traffic outside your OpenShift Container Platform cluster. Hosting an external IP address in this way is only applicable for a cluster installed on bare-metal hardware.

You must ensure that you correctly configure the external network infrastructure to route traffic to the service.

### 20.7.1. Prerequisites

- Your cluster is configured with ExternalIPs enabled. For more information, read [Configuring ExternalIPs for services](#).

**NOTE**

Do not use the same ExternalIP for the egress IP.

### 20.7.2. Attaching an ExternalIP to a service

You can attach an ExternalIP resource to a service. If you configured your cluster to automatically attach the resource to a service, you might not need to manually attach an ExternalIP to the service.

The examples in the procedure use a scenario that manually attaches an ExternalIP resource to a service in a cluster with an IP failover configuration.

#### Procedure

1. Confirm compatible IP address ranges for the ExternalIP resource by entering the following command in your CLI:

```
$ oc get networks.config cluster -o jsonpath='{.spec.externalIPs}'
```

**NOTE**

If **autoAssignCIDRs** is set and you did not specify a value for **spec.externalIPs** in the ExternalIP resource, OpenShift Container Platform automatically assigns ExternalIP to a new **Service** object.

2. Choose one of the following options to attach an ExternalIP resource to the service:

- a. If you are creating a new service, specify a value in the **spec.externalIPs** field and array of one or more valid IP addresses in the **allowedCIDRs** parameter.

#### Example of service YAML configuration file that supports an ExternalIP resource

```
apiVersion: v1
kind: Service
metadata:
  name: svc-with-externalip
spec:
  externalIPs:
    policy:
      allowedCIDRs:
        - 192.168.123.0/28
```

- b. If you are attaching an ExternalIP to an existing service, enter the following command. Replace **<name>** with the service name. Replace **<ip\_address>** with a valid ExternalIP address. You can provide multiple IP addresses separated by commas.

```
$ oc patch svc <name> -p \
'{
  "spec": {
    "externalIPs": [ "<ip_address>" ]
  }
}'
```

For example:

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
```

**Example output**

```
"mysql-55-rhel7" patched
```

3. To confirm that an ExternalIP address is attached to the service, enter the following command. If you specified an ExternalIP for a new service, you must create the service first.

```
$ oc get svc
```

**Example output**

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
mysql-55-rhel7	172.30.131.89	192.174.120.10	3306/TCP	13m

### 20.7.3. Additional resources

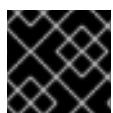
- [About MetalLB and the MetalLB Operator](#)
- [Configuring IP failover](#)
- [Configuring ExternalIPs for services](#)

## 20.8. CONFIGURING INGRESS CLUSTER TRAFFIC BY USING A NODEPORT

OpenShift Container Platform provides methods for communicating from outside the cluster with services running in the cluster. This method uses a **NodePort**.

### 20.8.1. Using a NodePort to get traffic into the cluster

Use a **NodePort**-type **Service** resource to expose a service on a specific port on all nodes in the cluster. The port is specified in the **Service** resource's **.spec.ports[\*].nodePort** field.



#### IMPORTANT

Using a node port requires additional port resources.

A **NodePort** exposes the service on a static port on the node's IP address. **NodePorts** are in the **30000** to **32767** range by default, which means a **NodePort** is unlikely to match a service's intended port. For example, port **8080** may be exposed as port **31020** on the node.

The administrator must ensure the external IP addresses are routed to the nodes.

**NodePorts** and external IPs are independent and both can be used concurrently.

**NOTE**

The procedures in this section require prerequisites performed by the cluster administrator.

### 20.8.2. Prerequisites

Before starting the following procedures, the administrator must:

- Set up the external port to the cluster networking environment so that requests can reach the cluster.
- Make sure there is at least one user with cluster admin role. To add this role to a user, run the following command:

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
```

- Have an OpenShift Container Platform cluster with at least one master and at least one node and a system outside the cluster that has network access to the cluster. This procedure assumes that the external system is on the same subnet as the cluster. The additional networking required for external systems on a different subnet is out-of-scope for this topic.

### 20.8.3. Creating a project and service

If the project and service that you want to expose does not exist, create the project and then create the service.

If the project and service already exists, skip to the procedure on exposing the service to create a route.

#### Prerequisites

- Install the OpenShift CLI (**oc**) and log in as a cluster administrator.

#### Procedure

1. Create a new project for your service by running the **oc new-project** command:

```
$ oc new-project <project_name>
```

2. Use the **oc new-app** command to create your service:

```
$ oc new-app nodejs:12~https://github.com/sclorg/nodejs-ex.git
```

3. To verify that the service was created, run the following command:

```
$ oc get svc -n <project_name>
```

#### Example output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nodejs-ex	ClusterIP	172.30.197.157	<none>	8080/TCP	70s

**NOTE**

By default, the new service does not have an external IP address.

#### 20.8.4. Exposing the service by creating a route

You can expose the service as a route by using the **oc expose** command.

##### Prerequisites

- You logged into OpenShift Container Platform.

##### Procedure

1. Log in to the project where the service you want to expose is located:

```
$ oc project <project_name>
```

2. To expose a node port for the application, modify the custom resource definition (CRD) of a service by entering the following command:

```
$ oc edit svc <service_name>
```

##### Example output

```
spec:
  ports:
    - name: 8443-tcp
      nodePort: 30327 1
      port: 8443
      protocol: TCP
      targetPort: 8443
    sessionAffinity: None
    type: NodePort 2
```

- 1** Optional: Specify the node port range for the application. By default, OpenShift Container Platform selects an available port in the **30000-32767** range.
- 2** Define the service type.

3. Optional: To confirm the service is available with a node port exposed, enter the following command:

```
$ oc get svc -n myproject
```

##### Example output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nodejs-ex	ClusterIP	172.30.217.127	<none>	3306/TCP	9m44s
nodejs-ex-ingress	NodePort	172.30.107.72	<none>	3306:31345/TCP	39s

4. Optional: To remove the service created automatically by the `oc new-app` command, enter the following command:

```
$ oc delete svc nodejs-ex
```

## Verification

- To check that the service node port is updated with a port in the **30000-32767** range, enter the following command:

```
$ oc get svc
```

In the following example output, the updated port is **30327**:

### Example output

```
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
httpd    NodePort   172.xx.xx.xx  <none>        8443:30327/TCP  109s
```

## 20.8.5. Additional resources

- [Configuring the node port service range](#)
- [Adding a single NodePort service to an Ingress Controller](#)

## 20.9. CONFIGURING INGRESS CLUSTER TRAFFIC USING LOAD BALANCER ALLOWED SOURCE RANGES

You can specify a list of IP address ranges for the **IngressController**. This restricts access to the load balancer service when the **endpointPublishingStrategy** is **LoadBalancerService**.

### 20.9.1. Configuring load balancer allowed source ranges

You can enable and configure the **spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges** field. By configuring load balancer allowed source ranges, you can limit the access to the load balancer for the Ingress Controller to a specified list of IP address ranges. The Ingress Operator reconciles the load balancer Service and sets the **spec.loadBalancerSourceRanges** field based on **AllowedSourceRanges**.



#### NOTE

If you have already set the **spec.loadBalancerSourceRanges** field or the load balancer service annotation **service.beta.kubernetes.io/load-balancer-source-ranges** in a previous version of OpenShift Container Platform, Ingress Controller starts reporting **Progressing=True** after an upgrade. To fix this, set **AllowedSourceRanges** that overwrites the **spec.loadBalancerSourceRanges** field and clears the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation. Ingress Controller starts reporting **Progressing=False** again.

## Prerequisites

- You have a deployed Ingress Controller on a running cluster.

## Procedure

- Set the allowed source ranges API for the Ingress Controller by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
--type=merge --patch='{"spec":{"endpointPublishingStrategy": \
{"type":"LoadBalancerService", "loadbalancer": \
{"scope":"External", "allowedSourceRanges":["0.0.0.0/0"]}}}' ①
```

- The example value **0.0.0.0/0** specifies the allowed source range.

### 20.9.2. Migrating to load balancer allowed source ranges

If you have already set the annotation **service.beta.kubernetes.io/load-balancer-source-ranges**, you can migrate to load balancer allowed source ranges. When you set the **AllowedSourceRanges**, the Ingress Controller sets the **spec.loadBalancerSourceRanges** field based on the **AllowedSourceRanges** value and unsets the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation.



#### NOTE

If you have already set the **spec.loadBalancerSourceRanges** field or the load balancer service annotation **service.beta.kubernetes.io/load-balancer-source-ranges** in a previous version of OpenShift Container Platform, the Ingress Controller starts reporting **Progressing=True** after an upgrade. To fix this, set **AllowedSourceRanges** that overwrites the **spec.loadBalancerSourceRanges** field and clears the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation. The Ingress Controller starts reporting **Progressing=False** again.

## Prerequisites

- You have set the **service.beta.kubernetes.io/load-balancer-source-ranges** annotation.

## Procedure

- Ensure that the **service.beta.kubernetes.io/load-balancer-source-ranges** is set:

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

#### Example output

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/load-balancer-source-ranges: 192.168.0.1/32
```

- Ensure that the **spec.loadBalancerSourceRanges** field is unset:

```
$ oc get svc router-default -n openshift-ingress -o yaml
```

#### Example output

```

...
spec:
  loadBalancerSourceRanges:
    - 0.0.0.0/0
...

```

3. Update your cluster to OpenShift Container Platform 4.18.
4. Set the allowed source ranges API for the **ingresscontroller** by running the following command:

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
--type=merge --patch='{"spec":{"endpointPublishingStrategy": \
{"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}' 1
```

- 1 The example value **0.0.0.0/0** specifies the allowed source range.

### 20.9.3. Additional resources

- [Introduction to OpenShift updates](#)

## 20.10. PATCHING EXISTING INGRESS OBJECTS

You can update or modify the following fields of existing **Ingress** objects without recreating the objects or disrupting services to them:

- Specifications
- Host
- Path
- Backend services
- SSL/TLS settings
- Annotations

### 20.10.1. Patching Ingress objects to resolve an **ingressWithoutClassName** alert

The **ingressClassName** field specifies the name of the **IngressClass** object. You must define the **ingressClassName** field for each **Ingress** object.

If you have not defined the **ingressClassName** field for an **Ingress** object, you could experience routing issues. After 24 hours, you will receive an **ingressWithoutClassName** alert to remind you to set the **ingressClassName** field.

#### Procedure

Patch the **Ingress** objects with a completed **ingressClassName** field to ensure proper routing and functionality.

1. List all **IngressClass** objects:

```
$ oc get ingressclass
```

2. List all **Ingress** objects in all namespaces:

```
$ oc get ingress -A
```

3. Patch the **Ingress** object:

```
$ oc patch ingress/<ingress_name> --type=merge --patch '{"spec": {"ingressClassName":"openshift-default"}}'
```

Replace **<ingress\_name>** with the name of the **Ingress** object. This command patches the **Ingress** object to include the desired ingress class name.

# CHAPTER 21. KUBERNETES NMSTATE

## 21.1. OBSERVING AND UPDATING THE NODE NETWORK STATE AND CONFIGURATION

After you install the Kubernetes NMState Operator, you can use the Operator to observe and update your cluster's node network state and network configuration.

For more information about how to install the NMState Operator, see [Kubernetes NMState Operator](#).



### IMPORTANT

You cannot provide any configuration that modifies the br-ex bridge, an OVN-Kubernetes-managed Open vSwitch 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.

### 21.1.1. 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 evaluate the freshness of the report.

### 21.1.2. Viewing the network state of a node (NNS) from the web console

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

#### Procedure

1. Navigate to **Networking** → **NodeNetworkState**.

In the **NodeNetworkState** page, you can 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.

2. To access the detailed information about **NodeNetworkState** resource, click the **NodeNetworkState** resource name listed in the **Name** column .
3. 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.

#### 21.1.2.1. Viewing a graphical representation of the NNS topology

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.

To open the topology view of the cluster, do the following:

1. In the **Administrator** view of the web console, navigate to **Networking** → **NodeNetworkState**.
2. In the upper-right corner of the page, click the **Topology** icon.  
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.

### 21.1.3. The `NodeNetworkConfigurationPolicy` manifest file

A **NodeNetworkConfigurationPolicy** (NNCP) manifest file defines policies that the Kubernetes NMState Operator uses to configure networking for nodes that exist in an OpenShift Container Platform cluster.

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.

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](#)

### 21.1.4. 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.

#### 21.1.4.1. 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.

#### 21.1.4.2. Creating a policy

You can create a policy by using either a form or YAML in the web console.

#### Procedure

1. Navigate to **Networking** → **NodeNetworkConfigurationPolicy**.
2. In the **NodeNetworkConfigurationPolicy** page, click **Create**, and select **From Form** option.  
In case there are no existing policies, you can alternatively click **Create NodeNetworkConfigurationPolicy** to create a policy using form.



#### NOTE

To create policy using YAML, click **Create**, and select **With YAML** option. The following steps are applicable to create a policy only by using form.

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. Optional: In the **Policy Interface(s)** section, a bridge interface is added by default with preset values in editable fields. Edit the values by executing the following steps:
  - a. Enter the name of the interface in **Interface name** field.
  - b. Select the network state from **Network state** dropdown. The default selected value is **Up**.
  - c. Select the type of interface from **Type** dropdown. The available values 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.

7. Click **Create** to complete policy creation.

## 21.1.5. Updating the policy

### 21.1.5.1. 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.

### 21.1.5.2. 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**.

### 21.1.5.3. 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**.

### 21.1.6. Managing policy by using the CLI

#### 21.1.6.1. Creating an interface on nodes

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](#)

### 21.1.6.2. 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
```

### 21.1.6.3. 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.

### 21.1.7. 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:

- When you need to apply a policy to more than one node, create a **NodeNetworkConfigurationPolicy** manifest for each target node. The Kubernetes NMState Operator applies the policy to each node with a defined NNCP in an unspecified order. Scoping a policy with this approach reduces the length of time for policy application but risks a cluster-wide outage if an error exists in the configuration of the cluster. To avoid this type of error, initially apply an NNCP to some nodes, confirm the NNCP is configured correctly for these nodes, and then proceed with applying the policy to the remaining nodes.
- 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.

### 21.1.7.1. Example: Ethernet interface node network configuration policy

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.

### 21.1.7.2. Example: Linux bridge interface node network configuration policy

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.

### 21.1.7.3. Example: VLAN interface node network configuration policy

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

#### 21.1.7.4. Example: Bond interface node network configuration policy

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:

- mode=1 active-backup
- mode=2 balance-xor
- mode=4 802.3ad

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:
            - 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 an active backup mode.
- 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.

#### 21.1.7.5. 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.

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.

#### 21.1.7.6. Example: Node network configuration policy for virtual functions

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 hardware 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.

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

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



#### IMPORTANT

Associating a VRF instance with a network interface 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](#).

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](#)

### 21.1.8. 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 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.



#### 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-ipob
spec:
# ...
  interfaces:
    - description: ""
      infiniband:
        mode: datagram 1
        pkey: "0xffff" 2
        ipv4:
          address:
            - ip: 100.125.3.4
              prefix-length: 16
            dhcp: false
            enabled: true
        ipv6:
          enabled: false
        name: ibp27s0
        state: up
        identifier: mac-address 3
        mac-address: 20:00:55:04:01:FE:80:00:00:00:00:00:02:C9:02:00:23:13:92 4
        type: infiniband 5
# ...
```

- 1 **datagram** is the default mode for an IPoIB interface, and this mode improves optimizes performance and latency. **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.
  - 2 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.
  - 3 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.
  - 4 Holds the MAC address of an interface. For an IP-over-InfiniBand (IPoIB) interface, the address is a 20-byte string.
  - 5 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> ①
```

- Replace **<nncp\_file\_name>** with the name of your NNCP file.

## 21.1.9. Capturing the static IP of a NIC attached to a bridge



### IMPORTANT

Capturing the static IP of a NIC 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](#).

### 21.1.9.1. 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.

## Additional resources

- [The NMPolicy project – Policy syntax](#)

### 21.1.10. 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.

#### 21.1.10.1. 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.

#### 21.1.10.2. 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.

#### 21.1.10.3. 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
# ...

```

#### 21.1.10.4. 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
# ...
```

#### 21.1.10.5. 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:
    search:
      - example.com
      - example.org
    server:
      - 2001:db8:f::1
      - 192.0.2.251
# ...

```

## 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:
    search:
      - example.com
      - example.org
    server:
      - 2001:db8:1::d1
      - 2001:db8:1::d2
      - 192.0.2.1
  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 dynamic DNS name servers for your network interface, define the dynamic 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 sets example.com and example.org static DNS search domains along with dynamic DNS name server settings**

```
# ...
dns-resolver:
  config:
    search:
      - example.com
      - example.org
    server: []
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
# ...
```

#### 21.1.10.6. Static routing

The following snippet configures a static route and a static IP 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 1
          prefix-length: 24
routes:
  config:
    - destination: 198.51.100.0/24
      metric: 150
      next-hop-address: 192.0.2.1 2
      next-hop-interface: eth1
      table-id: 254
# ...
```

**1** The static IP address for the Ethernet interface.

- 2 The next hop address for the node traffic. This must be in the same subnet as the IP address set for the Ethernet interface.



## 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.

## 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 \(User-provisioned infrastructure\)](#)

## 21.2. 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.

### 21.2.1. 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:

### Example output

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:

### Example output

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**:

#### Example output

```
- 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.

## 21.2.2. Troubleshooting DNS connectivity issues in a disconnected environment

If you experience DNS connectivity issues when configuring **nmstate** in a disconnected environment, you can configure the DNS server to resolve the list of name servers for the domain **root-servers.net**.



### 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.

### 21.2.2.1. 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 NS record. For example you can use the **/var/named/named.localhost** as a zone file that already matches this criteria.

#### Procedure

- 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
```

- Restart the **named** service by running the following command:

```
$ systemctl restart named
```

- 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
```

- 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.
```

### 21.2.2.2. 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.

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.
```

# CHAPTER 22. 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 Platform (GCP), 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.

## 22.1. 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.

## 22.2. 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**

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

**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.

## 22.3. 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.

## 22.4. 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.

## 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 23. CONFIGURING A CUSTOM PKI

Some platform components, such as the web console, use Routes for communication and must trust other components' certificates to interact with them. If you are using a custom public key infrastructure (PKI), you must configure it so its privately signed CA certificates are recognized across the cluster.

You can leverage the Proxy API to add cluster-wide trusted CA certificates. You must do this either during installation or at runtime.

- During *installation*, [configure the cluster-wide proxy](#). You must define your privately signed CA certificates in the **install-config.yaml** file's **additionalTrustBundle** setting. The installation program generates a ConfigMap that is named **user-ca-bundle** that contains the additional CA certificates you defined. The Cluster Network Operator then creates a **trusted-ca-bundle** ConfigMap that merges these CA certificates with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle; this ConfigMap is referenced in the Proxy object's **trustedCA** field.
- At *runtime*, [modify the default Proxy object to include your privately signed CA certificates](#) (part of cluster's proxy enablement workflow). This involves creating a ConfigMap that contains the privately signed CA certificates that should be trusted by the cluster, and then modifying the proxy resource with the **trustedCA** referencing the privately signed certificates' ConfigMap.



## NOTE

The installer configuration's **additionalTrustBundle** field and the proxy resource's **trustedCA** field are used to manage the cluster-wide trust bundle; **additionalTrustBundle** is used at install time and the proxy's **trustedCA** is used at runtime.

The **trustedCA** field is a reference to a **ConfigMap** containing the custom certificate and key pair used by the cluster component.

## 23.1. CONFIGURING THE CLUSTER-WIDE PROXY DURING INSTALLATION

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

### Prerequisites

- You reviewed the sites that your cluster requires access to and determined whether any of them need to bypass the proxy. By default, all cluster egress traffic is proxied, including calls to hosting cloud provider APIs. You added sites to the **Proxy** object's **spec.noProxy** field to bypass the proxy if necessary.



## NOTE

The **Proxy** object **status.noProxy** field is populated with the values of the **networking.machineNetwork[].cidr**, **networking.clusterNetwork[].cidr**, and **networking.serviceNetwork[]** fields from your installation configuration.

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

## Procedure

- 1 Edit your **install-config.yaml** file and add the proxy settings. For example:

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> ①
  httpsProxy: https://<username>:<pswd>@<ip>:<port> ②
  noProxy: ec2.<aws_region>.amazonaws.com,elasticloadbalancing.
<aws_region>.amazonaws.com,s3.<aws_region>.amazonaws.com ③
  additionalTrustBundle: | ④
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
additionalTrustBundlePolicy: <policy_to_add_additionalTrustBundle> 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.
- 3 A comma-separated list of destination domain names, IP addresses, or other network CIDRs to exclude from proxying. Preface a domain with . to match subdomains only. For example, **.y.com** matches **x.y.com**, but not **y.com**. Use \* to bypass the proxy for all destinations. If you have added the Amazon **EC2,Elastic Load Balancing**, and **S3** VPC endpoints to your VPC, you must add these endpoints to the **noProxy** field.
- 4 If provided, the installation program generates a config map that is named **user-ca-bundle** in the **openshift-config** namespace that contains one or more additional CA certificates that are required for proxying HTTPS connections. The Cluster Network Operator then creates a **trusted-ca-bundle** config map that merges these contents with the Red Hat Enterprise Linux CoreOS (RHCOS) trust bundle, and this config map is referenced in the **trustedCA** field of the **Proxy** object. The **additionalTrustBundle** field is required unless the proxy's identity certificate is signed by an authority from the RHCOS trust bundle.
- 5 Optional: The policy to determine the configuration of the **Proxy** object to reference the **user-ca-bundle** config map in the **trustedCA** field. The allowed values are **Proxyonly** and **Always**. Use **Proxyonly** to reference the **user-ca-bundle** config map only when **http/https** proxy is configured. Use **Always** to always reference the **user-ca-bundle** config map. The default value is **Proxyonly**.

**NOTE**

The installation program does not support the proxy **readinessEndpoints** field.

**NOTE**

If the installer times out, restart and then complete the deployment by using the **wait-for** command of the installer. For example:

```
$ ./openshift-install wait-for install-complete --log-level debug
```

2. Save the file and reference it when installing OpenShift Container Platform.

The installation program creates a cluster-wide proxy that is named **cluster** that uses the proxy settings in the provided **install-config.yaml** file. If no proxy settings are provided, a **cluster Proxy** object is still created, but it will have a nil **spec**.

**NOTE**

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

## 23.2. 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**

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

**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.

### 23.3. 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.

# CHAPTER 24. LOAD BALANCING ON RHOSP

## 24.1. LIMITATIONS OF LOAD BALANCER SERVICES

OpenShift Container Platform clusters on Red Hat OpenStack Platform (RHOSP) use Octavia to handle load balancer services. As a result of this choice, such clusters have a number of functional limitations.

RHOSP Octavia has two supported providers: Amphora and OVN. These providers differ in terms of available features as well as implementation details. These distinctions affect load balancer services that are created on your cluster.

### 24.1.1. Local external traffic policies

You can set the external traffic policy (ETP) parameter, `.spec.externalTrafficPolicy`, on a load balancer service to preserve the source IP address of incoming traffic when it reaches service endpoint pods. However, if your cluster uses the Amphora Octavia provider, the source IP of the traffic is replaced with the IP address of the Amphora VM. This behavior does not occur if your cluster uses the OVN Octavia provider.

Having the **ETP** option set to **Local** requires that health monitors be created for the load balancer. Without health monitors, traffic can be routed to a node that does not have a functional endpoint, which causes the connection to drop. To force Cloud Provider OpenStack to create health monitors, you must set the value of the **create-monitor** option in the cloud provider configuration to **true**.

In RHOSP 16.2, the OVN Octavia provider does not support health monitors. Therefore, setting the ETP to local is unsupported.

In RHOSP 16.2, the Amphora Octavia provider does not support HTTP monitors on UDP pools. As a result, UDP load balancer services have **UDP-CONNECT** monitors created instead. Due to implementation details, this configuration only functions properly with the OVN-Kubernetes CNI plugin.

## 24.2. SCALING CLUSTERS FOR APPLICATION TRAFFIC BY USING OCTAVIA

OpenShift Container Platform clusters that run on Red Hat OpenStack Platform (RHOSP) can use the Octavia load balancing service to distribute traffic across multiple virtual machines (VMs) or floating IP addresses. This feature mitigates the bottleneck that single machines or addresses create.

You must create your own Octavia load balancer to use it for application network scaling.

### 24.2.1. Scaling clusters by using Octavia

If you want to use multiple API load balancers, create an Octavia load balancer and then configure your cluster to use it.

#### Prerequisites

- Octavia is available on your Red Hat OpenStack Platform (RHOSP) deployment.

#### Procedure

- From a command line, create an Octavia load balancer that uses the Amphora driver:

```
$ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
<id_of_worker_vms_subnet>
```

You can use a name of your choice instead of **API\_OCP\_CLUSTER**.

- After the load balancer becomes active, create listeners:

```
$ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
HTTPS--protocol-port 6443 API_OCP_CLUSTER
```



#### NOTE

To view the status of the load balancer, enter **openstack loadbalancer list**.

- Create a pool that uses the round robin algorithm and has session persistence enabled:

```
$ openstack loadbalancer pool create --name API_OCP_CLUSTER_pool_6443 --lb-
algorithm ROUND_ROBIN --session-persistence type=<source_IP_address> --listener
API_OCP_CLUSTER_6443 --protocol HTTPS
```

- To ensure that control plane machines are available, create a health monitor:

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --type
TCP API_OCP_CLUSTER_pool_6443
```

- Add the control plane machines as members of the load balancer pool:

```
$ for SERVER in $(MASTER-0-IP MASTER-1-IP MASTER-2-IP)
do
  openstack loadbalancer member create --address $SERVER --protocol-port 6443
  API_OCP_CLUSTER_pool_6443
done
```

- Optional: To reuse the cluster API floating IP address, unset it:

```
$ openstack floating ip unset $API_FIP
```

- Add either the unset **API\_FIP** or a new address to the created load balancer VIP:

```
$ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value
API_OCP_CLUSTER) $API_FIP
```

Your cluster now uses Octavia for load balancing.

## 24.3. SERVICES FOR A USER-MANAGED LOAD BALANCER

You can configure an OpenShift Container Platform cluster on Red Hat OpenStack Platform (RHOSP) to use a user-managed load balancer in place of the default load balancer.



## IMPORTANT

Configuring a user-managed load balancer depends on your vendor's load balancer.

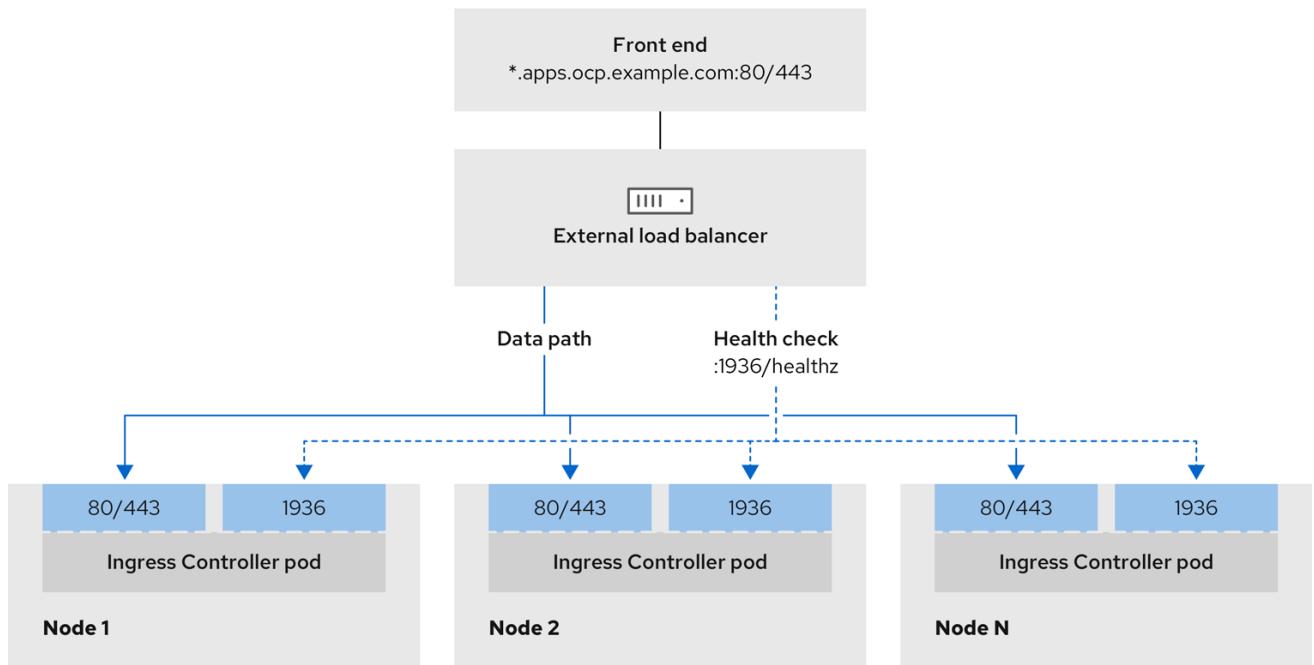
The information and examples in this section are for guideline purposes only. Consult the vendor documentation for more specific information about the vendor's load balancer.

Red Hat supports the following services for a user-managed load balancer:

- Ingress Controller
- OpenShift API
- OpenShift MachineConfig API

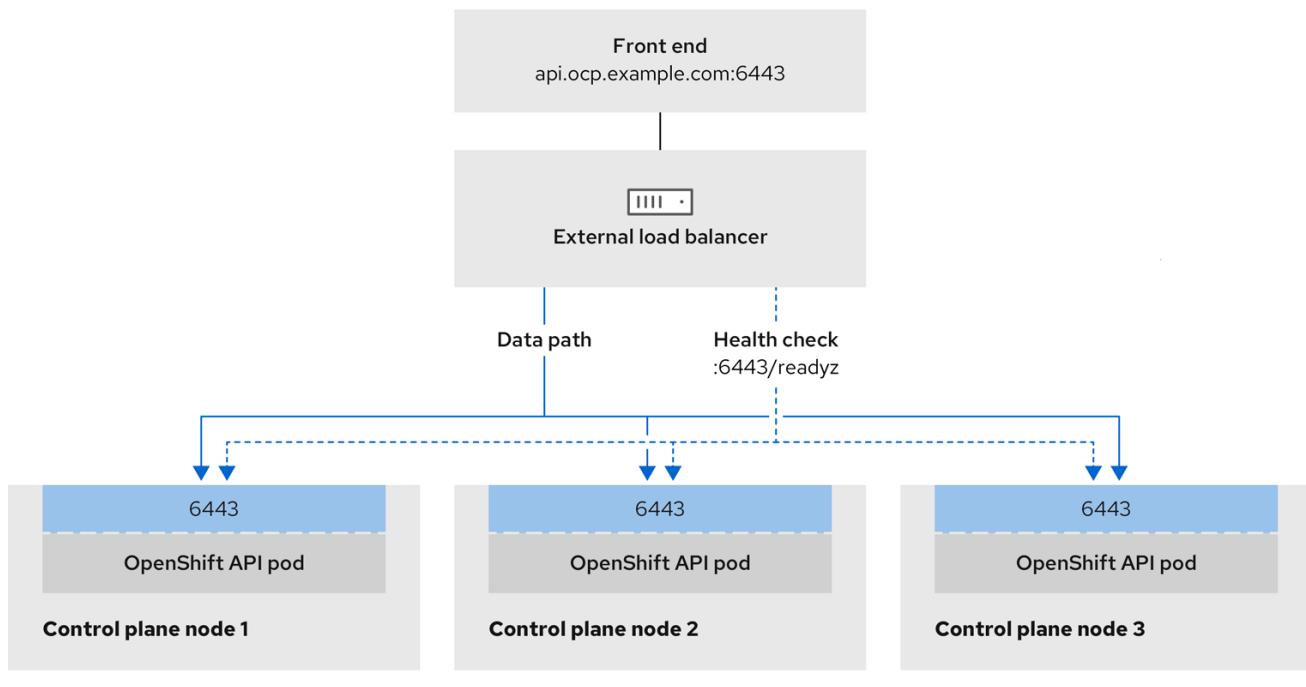
You can choose whether you want to configure one or all of these services for a user-managed load balancer. Configuring only the Ingress Controller service is a common configuration option. To better understand each service, view the following diagrams:

**Figure 24.1. Example network workflow that shows an Ingress Controller operating in an OpenShift Container Platform environment**



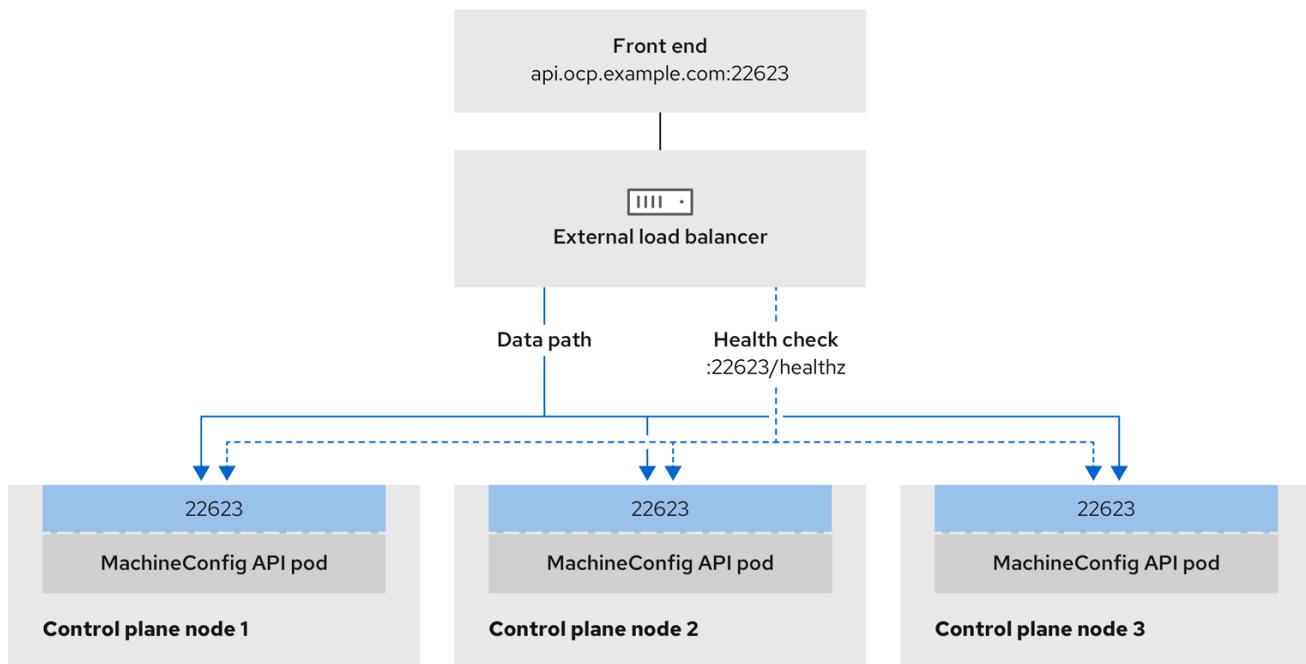
496\_OpenShift\_1223

Figure 24.2. Example network workflow that shows an OpenShift API operating in an OpenShift Container Platform environment



496\_OpenShift\_I223

Figure 24.3. Example network workflow that shows an OpenShift MachineConfig API operating in an OpenShift Container Platform environment



496\_OpenShift\_I223

The following configuration options are supported for user-managed load balancers:

- Use a node selector to map the Ingress Controller to a specific set of nodes. You must assign a static IP address to each node in this set, or configure each node to receive the same IP address from the Dynamic Host Configuration Protocol (DHCP). Infrastructure nodes commonly receive this type of configuration.

- Target all IP addresses on a subnet. This configuration can reduce maintenance overhead, because you can create and destroy nodes within those networks without reconfiguring the load balancer targets. If you deploy your ingress pods by using a machine set on a smaller network, such as a /27 or /28, you can simplify your load balancer targets.

### TIP

You can list all IP addresses that exist in a network by checking the machine config pool's resources.

Before you configure a user-managed load balancer for your OpenShift Container Platform cluster, consider the following information:

- For a front-end IP address, you can use the same IP address for the front-end IP address, the Ingress Controller's load balancer, and API load balancer. Check the vendor's documentation for this capability.
- For a back-end IP address, ensure that an IP address for an OpenShift Container Platform control plane node does not change during the lifetime of the user-managed load balancer. You can achieve this by completing one of the following actions:
  - Assign a static IP address to each control plane node.
  - Configure each node to receive the same IP address from the DHCP every time the node requests a DHCP lease. Depending on the vendor, the DHCP lease might be in the form of an IP reservation or a static DHCP assignment.
- Manually define each node that runs the Ingress Controller in the user-managed load balancer for the Ingress Controller back-end service. For example, if the Ingress Controller moves to an undefined node, a connection outage can occur.

#### 24.3.1. Configuring a user-managed load balancer

You can configure an OpenShift Container Platform cluster on Red Hat OpenStack Platform (RHOSP) to use a user-managed load balancer in place of the default load balancer.



### IMPORTANT

Before you configure a user-managed load balancer, ensure that you read the "Services for a user-managed load balancer" section.

Read the following prerequisites that apply to the service that you want to configure for your user-managed load balancer.



### NOTE

MetalLB, which runs on a cluster, functions as a user-managed load balancer.

#### OpenShift API prerequisites

- You defined a front-end IP address.
- TCP ports 6443 and 22623 are exposed on the front-end IP address of your load balancer. Check the following items:

- Port 6443 provides access to the OpenShift API service.
- Port 22623 can provide ignition startup configurations to nodes.
- The front-end IP address and port 6443 are reachable by all users of your system with a location external to your OpenShift Container Platform cluster.
- The front-end IP address and port 22623 are reachable only by OpenShift Container Platform nodes.
- The load balancer backend can communicate with OpenShift Container Platform control plane nodes on port 6443 and 22623.

## Ingress Controller prerequisites

- You defined a front-end IP address.
- TCP ports 443 and 80 are exposed on the front-end IP address of your load balancer.
- The front-end IP address, port 80 and port 443 are be reachable by all users of your system with a location external to your OpenShift Container Platform cluster.
- The front-end IP address, port 80 and port 443 are reachable to all nodes that operate in your OpenShift Container Platform cluster.
- The load balancer backend can communicate with OpenShift Container Platform nodes that run the Ingress Controller on ports 80, 443, and 1936.

## Prerequisite for health check URL specifications

You can configure most load balancers by setting health check URLs that determine if a service is available or unavailable. OpenShift Container Platform provides these health checks for the OpenShift API, Machine Configuration API, and Ingress Controller backend services.

The following examples show health check specifications for the previously listed backend services:

### Example of a Kubernetes API health check specification

```
Path: HTTPS:6443/readyz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

### Example of a Machine Config API health check specification

```
Path: HTTPS:22623/healthz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

### Example of an Ingress Controller health check specification

```
Path: HTTP:1936/healthz/ready
```

Healthy threshold: 2  
 Unhealthy threshold: 2  
 Timeout: 5  
 Interval: 10

## Procedure

- Configure the HAProxy Ingress Controller, so that you can enable access to the cluster from your load balancer on ports 6443, 22623, 443, and 80. Depending on your needs, you can specify the IP address of a single subnet or IP addresses from multiple subnets in your HAProxy configuration.

### Example HAProxy configuration with one listed subnet

```
# ...
listen my-cluster-api-6443
  bind 192.168.1.100:6443
  mode tcp
  balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /readyz
  http-check expect status 200
  server my-cluster-master-2 192.168.1.101:6443 check inter 10s rise 2 fall 2
  server my-cluster-master-0 192.168.1.102:6443 check inter 10s rise 2 fall 2
  server my-cluster-master-1 192.168.1.103:6443 check inter 10s rise 2 fall 2

listen my-cluster-machine-config-api-22623
  bind 192.168.1.100:22623
  mode tcp
  balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz
  http-check expect status 200
  server my-cluster-master-2 192.168.1.101:22623 check inter 10s rise 2 fall 2
  server my-cluster-master-0 192.168.1.102:22623 check inter 10s rise 2 fall 2
  server my-cluster-master-1 192.168.1.103:22623 check inter 10s rise 2 fall 2

listen my-cluster-apps-443
  bind 192.168.1.100:443
  mode tcp
  balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz/ready
  http-check expect status 200
  server my-cluster-worker-0 192.168.1.111:443 check port 1936 inter 10s rise 2 fall 2
  server my-cluster-worker-1 192.168.1.112:443 check port 1936 inter 10s rise 2 fall 2
  server my-cluster-worker-2 192.168.1.113:443 check port 1936 inter 10s rise 2 fall 2

listen my-cluster-apps-80
  bind 192.168.1.100:80
  mode tcp
  balance roundrobin
```

```

option httpchk
http-check connect
http-check send meth GET uri /healthz/ready
http-check expect status 200
server my-cluster-worker-0 192.168.1.111:80 check port 1936 inter 10s rise 2 fall 2
server my-cluster-worker-1 192.168.1.112:80 check port 1936 inter 10s rise 2 fall 2
server my-cluster-worker-2 192.168.1.113:80 check port 1936 inter 10s rise 2 fall 2
# ...

```

### Example HAProxy configuration with multiple listed subnets

```

# ...
listen api-server-6443
bind *:6443
mode tcp
server master-00 192.168.83.89:6443 check inter 1s
server master-01 192.168.84.90:6443 check inter 1s
server master-02 192.168.85.99:6443 check inter 1s
server bootstrap 192.168.80.89:6443 check inter 1s

listen machine-config-server-22623
bind *:22623
mode tcp
server master-00 192.168.83.89:22623 check inter 1s
server master-01 192.168.84.90:22623 check inter 1s
server master-02 192.168.85.99:22623 check inter 1s
server bootstrap 192.168.80.89:22623 check inter 1s

listen ingress-router-80
bind *:80
mode tcp
balance source
server worker-00 192.168.83.100:80 check inter 1s
server worker-01 192.168.83.101:80 check inter 1s

listen ingress-router-443
bind *:443
mode tcp
balance source
server worker-00 192.168.83.100:443 check inter 1s
server worker-01 192.168.83.101:443 check inter 1s

listen ironic-api-6385
bind *:6385
mode tcp
balance source
server master-00 192.168.83.89:6385 check inter 1s
server master-01 192.168.84.90:6385 check inter 1s
server master-02 192.168.85.99:6385 check inter 1s
server bootstrap 192.168.80.89:6385 check inter 1s

listen inspector-api-5050
bind *:5050
mode tcp
balance source
server master-00 192.168.83.89:5050 check inter 1s

```

```
server master-01 192.168.84.90:5050 check inter 1s
server master-02 192.168.85.99:5050 check inter 1s
server bootstrap 192.168.80.89:5050 check inter 1s
# ...
```

2. Use the **curl** CLI command to verify that the user-managed load balancer and its resources are operational:

- Verify that the cluster machine configuration API is accessible to the Kubernetes API server resource, by running the following command and observing the response:

```
$ curl https://<loadbalancer_ip_address>:6443/version --insecure
```

If the configuration is correct, you receive a JSON object in response:

```
{
  "major": "1",
  "minor": "11+",
  "gitVersion": "v1.11.0+ad103ed",
  "gitCommit": "ad103ed",
  "gitTreeState": "clean",
  "buildDate": "2019-01-09T06:44:10Z",
  "goVersion": "go1.10.3",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

- Verify that the cluster machine configuration API is accessible to the Machine config server resource, by running the following command and observing the output:

```
$ curl -v https://<loadbalancer_ip_address>:22623/healthz --insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
Content-Length: 0
```

- Verify that the controller is accessible to the Ingress Controller resource on port 80, by running the following command and observing the output:

```
$ curl -I -L -H "Host: console-openshift-console.apps.<cluster_name>.<base_domain>" http://<load_balancer_front_end_IP_address>
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.ocp4.private.opequon.net/
cache-control: no-cache
```

- Verify that the controller is accessible to the Ingress Controller resource on port 443, by running the following command and observing the output:

```
$ curl -I -L --insecure --resolve console-openshift-console.apps.<cluster_name>.<base_domain>:443:<Load Balancer Front End IP Address> https://console-openshift-console.apps.<cluster_name>.<base_domain>
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UIYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJFyQwWcGBsja261dG
LgaYO0nxzVERhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/
HttpOnly; Secure; SameSite=None
cache-control: private
```

3. Configure the DNS records for your cluster to target the front-end IP addresses of the user-managed load balancer. You must update records to your DNS server for the cluster API and applications over the load balancer.

### Examples of modified DNS records

```
<load_balancer_ip_address> A api.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```

```
<load_balancer_ip_address> A apps.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```



#### IMPORTANT

DNS propagation might take some time for each DNS record to become available. Ensure that each DNS record propagates before validating each record.

4. For your OpenShift Container Platform cluster to use the user-managed load balancer, you must specify the following configuration in your cluster's **install-config.yaml** file:

```
# ...
platform:
openstack:
loadBalancer:
type: UserManaged 1
apiVIPs:
- <api_ip> 2
```

```
    ingressVIPs:
      - <ingress_ip> ③
    # ...
```

- 1 Set **UserManaged** for the **type** parameter to specify a user-managed load balancer for your cluster. The parameter defaults to **OpenShiftManagedDefault**, which denotes the default internal load balancer. For services defined in an **openshift-kni-infra** namespace, a user-managed load balancer can deploy the **coredns** service to pods in your cluster but ignores **keepalived** and **haproxy** services.
- 2 Required parameter when you specify a user-managed load balancer. Specify the user-managed load balancer's public IP address, so that the Kubernetes API can communicate with the user-managed load balancer.
- 3 Required parameter when you specify a user-managed load balancer. Specify the user-managed load balancer's public IP address, so that the user-managed load balancer can manage ingress traffic for your cluster.

## Verification

1. Use the **curl** CLI command to verify that the user-managed load balancer and DNS record configuration are operational:
  - a. Verify that you can access the cluster API, by running the following command and observing the output:

```
$ curl https://api.<cluster_name>.<base_domain>:6443/version --insecure
```

If the configuration is correct, you receive a JSON object in response:

```
{
  "major": "1",
  "minor": "11+",
  "gitVersion": "v1.11.0+ad103ed",
  "gitCommit": "ad103ed",
  "gitTreeState": "clean",
  "buildDate": "2019-01-09T06:44:10Z",
  "goVersion": "go1.10.3",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

- b. Verify that you can access the cluster machine configuration, by running the following command and observing the output:

```
$ curl -v https://api.<cluster_name>.<base_domain>:22623/healthz --insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
Content-Length: 0
```

- c. Verify that you can access each cluster application on port, by running the following command and observing the output:

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.<cluster-name>.<base domain>/
cache-control: no-cache
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=39HoZgztDnzjJkq/JuLJMeoKNXIfiVv2YgZc09c3TBOBU4NI6kDXaJH1LdicNhN1UsQ
Wzon4Dor9GWGfopaTEQ==; Path=/; Secure
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Tue, 17 Nov 2020 08:42:10 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=9b714eb87e93cf34853e87a92d6894be; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

- d. Verify that you can access each cluster application on port 443, by running the following command and observing the output:

```
$ curl https://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --insecure
```

If the configuration is correct, the output from the command shows the following response:

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UIYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja261dGLgaYO0nxzVERhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

## 24.4. SPECIFYING A FLOATING IP ADDRESS IN THE INGRESS CONTROLLER

By default, a floating IP address gets randomly assigned to your OpenShift Container Platform cluster on Red Hat OpenStack Platform (RHOSP) upon deployment. This floating IP address is associated with your Ingress port.

You might want to pre-create a floating IP address before updating your DNS records and cluster deployment. In this situation, you can define a floating IP address to the Ingress Controller. You can do this regardless of whether you are using Octavia or a user-managed cluster.

## Procedure

- 1 Create the Ingress Controller custom resource (CR) file with the floating IPs:

### Example Ingress config `sample-ingress.yaml`

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> ①
spec:
  domain: <domain> ②
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: External ③
  providerParameters:
    type: OpenStack
    openstack:
      floatingIP: <ingress_port_IP> ④
```

- ① The name of your Ingress Controller. If you are using the default Ingress Controller, the value for this field is **default**.
- ② The DNS name serviced by the Ingress Controller.
- ③ You must set the scope to **External** to use a floating IP address.
- ④ The floating IP address associated with the port your Ingress Controller is listening on.

- 2 Apply the CR file by running the following command:

```
$ oc apply -f sample-ingress.yaml
```

- 3 Update your DNS records with the Ingress Controller endpoint:

```
*.apps.<name>.<domain>. IN A <ingress_port_IP>
```

- 4 Continue with creating your OpenShift Container Platform cluster.

## Verification

- Confirm that the load balancer was successfully provisioned by checking the **IngressController** conditions using the following command:

```
$ oc get ingresscontroller -n openshift-ingress-operator <name> -o jsonpath=".status.conditions" | yq -PC
```

# CHAPTER 25. LOAD BALANCING WITH METALLB

## 25.1. CONFIGURING METALLB ADDRESS POOLS

As a cluster administrator, you can add, modify, and delete address pools. The MetalLB Operator uses the address pool custom resources to set the IP addresses that MetalLB can assign to services. The namespace used in the examples assume the namespace is **metallb-system**.

For more information about how to install the MetalLB Operator, see [About MetalLB and the MetalLB Operator](#).

### 25.1.1. About the IPAddressPool custom resource

The fields for the **IPAddressPool** custom resource are described in the following tables.

**Table 25.1. MetalLB IPAddressPool pool custom resource**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	Specifies the name for the address pool. When you add a service, you can specify this pool name in the <b>metallb.io/address-pool</b> annotation to select an IP address from a specific pool. The names <b>doc-example</b> , <b>silver</b> , and <b>gold</b> are used throughout the documentation.
<b>metadata.namespace</b>	<b>string</b>	Specifies the namespace for the address pool. Specify the same namespace that the MetalLB Operator uses.
<b>metadata.label</b>	<b>string</b>	Optional: Specifies the key value pair assigned to the <b>IPAddressPool</b> . This can be referenced by the <b>ipAddressPoolSelectors</b> in the <b>BGPAdvertisement</b> and <b>L2Advertisement</b> CRD to associate the <b>IPAddressPool</b> with the advertisement
<b>spec.addresses</b>	<b>string</b>	Specifies a list of IP addresses for MetalLB Operator to assign to services. You can specify multiple ranges in a single pool; they will all share the same settings. Specify each range in CIDR notation or as starting and ending IP addresses separated with a hyphen.
<b>spec.autoAssign</b>	<b>boolean</b>	Optional: Specifies whether MetalLB automatically assigns IP addresses from this pool. Specify <b>false</b> if you want explicitly request an IP address from this pool with the <b>metallb.io/address-pool</b> annotation. The default value is <b>true</b> .
<b>spec.avoidBuggyIPs</b>	<b>boolean</b>	Optional: This ensures when enabled that IP addresses ending <b>.0</b> and <b>.255</b> are not allocated from the pool. The default value is <b>false</b> . Some older consumer network equipment mistakenly block IP addresses ending in <b>.0</b> and <b>.255</b> .

You can assign IP addresses from an **IPAddressPool** to services and namespaces by configuring the **spec.serviceAllocation** specification.

Table 25.2. MetallLB IPAddressPool custom resource spec.serviceAllocation subfields

Field	Type	Description
<b>priority</b>	<b>int</b>	Optional: Defines the priority between IP address pools when more than one IP address pool matches a service or namespace. A lower number indicates a higher priority.
<b>namespaces</b>	<b>array (string)</b>	Optional: Specifies a list of namespaces that you can assign to IP addresses in an IP address pool.
<b>namespaceSelectors</b>	<b>array (LabelSelector)</b>	Optional: Specifies namespace labels that you can assign to IP addresses from an IP address pool by using label selectors in a list format.
<b>serviceSelectors</b>	<b>array (LabelSelector)</b>	Optional: Specifies service labels that you can assign to IP addresses from an address pool by using label selectors in a list format.

### 25.1.2. Configuring an address pool

As a cluster administrator, you can add address pools to your cluster to control the IP addresses that MetallLB can assign to load-balancer services.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

- 1 Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example
  labels: ①
  zone: east
spec:
  addresses:
    - 203.0.113.1-203.0.113.10
    - 203.0.113.65-203.0.113.75
# ...
```

① This label assigned to the **IPAddressPool** can be referenced by the **ipAddressPoolSelectors** in the **BGPAdvertisement** CRD to associate the **IPAddressPool** with the advertisement.

2. Apply the configuration for the IP address pool:

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

## Verification

1. View the address pool by entering the following command:

```
$ oc describe -n metallb-system IPAddressPool doc-example
```

### Example output

```
Name:      doc-example
Namespace: metallb-system
Labels:    zone=east
Annotations: <none>
API Version: metallb.io/v1beta1
Kind:      IPAddressPool
Metadata:
...
Spec:
  Addresses:
    203.0.113.1-203.0.113.10
    203.0.113.65-203.0.113.75
  Auto Assign: true
  Events:     <none>
```

2. Confirm that the address pool name, such as **doc-example**, and the IP address ranges exist in the output.

### 25.1.3. Configure MetalLB address pool for VLAN

As a cluster administrator, you can add address pools to your cluster to control the IP addresses on a created VLAN that MetalLB can assign to load-balancer services

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Configure a separate VLAN.
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create a file, such as **ipaddresspool-vlan.yaml**, that is similar to the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-vlan
  labels:
```

```
zone: east ①
spec:
addresses:
- 192.168.100.1-192.168.100.254 ②
```

- ① This label assigned to the **IPAddressPool** can be referenced by the **ipAddressPoolSelectors** in the **BGPAdvertisement** CRD to associate the **IPAddressPool** with the advertisement.
- ② This IP range must match the subnet assigned to the VLAN on your network. To support layer 2 (L2) mode, the IP address range must be within the same subnet as the cluster nodes.

2. Apply the configuration for the IP address pool:

```
$ oc apply -f ipaddresspool-vlan.yaml
```

3. To ensure this configuration applies to the VLAN you need to set the **spec.gatewayConfig.ipForwarding** to **Global**.

a. Run the following command to edit the network configuration custom resource (CR):

```
$ oc edit network.operator.openshift/cluster
```

b. Update the **spec.defaultNetwork.ovnKubernetesConfig** section to include the **gatewayConfig.ipForwarding** set to **Global**. It should look something like this:

### Example

```
...
spec:
clusterNetwork:
- cidr: 10.128.0.0/14
  hostPrefix: 23
defaultNetwork:
type: OVNKubernetes
ovnKubernetesConfig:
  gatewayConfig:
    ipForwarding: Global
...
```

## 25.1.4. Example address pool configurations

The following examples show address pool configurations for specific scenarios.

### 25.1.4.1. Example: IPv4 and CIDR ranges

You can specify a range of IP addresses in classless inter-domain routing (CIDR) notation. You can combine CIDR notation with the notation that uses a hyphen to separate lower and upper bounds.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
```

```

name: doc-example-cidr
namespace: metallb-system
spec:
  addresses:
    - 192.168.100.0/24
    - 192.168.200.0/24
    - 192.168.255.1-192.168.255.5
# ...

```

#### 25.1.4.2. Example: Assign IP addresses

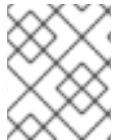
You can set the **autoAssign** field to **false** to prevent MetalLB from automatically assigning IP addresses from the address pool. You can then assign a single IP address or multiple IP addresses from an IP address pool. To assign an IP address, append the **/32** CIDR notation to the target IP address in the **spec.addresses** parameter. This setting ensures that only the specific IP address is available for assignment, leaving non-reserved IP addresses for application use.

#### Example IPAddressPool CR that assigns multiple IP addresses

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-reserved
  namespace: metallb-system
spec:
  addresses:
    - 192.168.100.1/32
    - 192.168.200.1/32
  autoAssign: false
# ...

```



#### NOTE

When you add a service, you can request a specific IP address from the address pool or you can specify the pool name in an annotation to request any IP address from the pool.

#### 25.1.4.3. Example: IPv4 and IPv6 addresses

You can add address pools that use IPv4 and IPv6. You can specify multiple ranges in the **addresses** list, just like several IPv4 examples.

Whether the service is assigned a single IPv4 address, a single IPv6 address, or both is determined by how you add the service. The **spec.ipFamilies** and **spec.ipFamilyPolicy** fields control how IP addresses are assigned to the service.

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-combined
  namespace: metallb-system
spec:
  addresses:

```

```

- 10.0.100.0/28 ①
- 2002:2:2::1-2002:2:2::100
# ...

```

- ① Where **10.0.100.0/28** is the local network IP address followed by the **/28** network prefix.

#### 25.1.4.4. Example: Assign IP address pools to services or namespaces

You can assign IP addresses from an **IPAddressPool** to services and namespaces that you specify.

If you assign a service or namespace to more than one IP address pool, MetalLB uses an available IP address from the higher-priority IP address pool. If no IP addresses are available from the assigned IP address pools with a high priority, MetalLB uses available IP addresses from an IP address pool with lower priority or no priority.



#### NOTE

You can use the **matchLabels** label selector, the **matchExpressions** label selector, or both, for the **namespaceSelectors** and **serviceSelectors** specifications. This example demonstrates one label selector for each specification.

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-service-allocation
  namespace: metallb-system
spec:
  addresses:
    - 192.168.20.0/24
  serviceAllocation:
    priority: 50 ①
    namespaces: ②
      - namespace-a
      - namespace-b
    namespaceSelectors: ③
      - matchLabels:
          zone: east
    serviceSelectors: ④
      - matchExpressions:
          - key: security
            operator: In
            values:
              - S1
# ...

```

- ① Assign a priority to the address pool. A lower number indicates a higher priority.
- ② Assign one or more namespaces to the IP address pool in a list format.
- ③ Assign one or more namespace labels to the IP address pool by using label selectors in a list format.
- ④ Assign one or more service labels to the IP address pool by using label selectors in a list format.

## 25.1.5. Next steps

- Configuring MetalLB with an L2 advertisement and label
- Configuring MetalLB BGP peers
- Configuring services to use MetalLB

## 25.2. ABOUT ADVERTISING FOR THE IP ADDRESS POOLS

You can configure MetalLB so that the IP address is advertised with layer 2 protocols, the BGP protocol, or both. With layer 2, MetalLB provides a fault-tolerant external IP address. With BGP, MetalLB provides fault-tolerance for the external IP address and load balancing.

MetalLB supports advertising using L2 and BGP for the same set of IP addresses.

MetalLB provides the flexibility to assign address pools to specific BGP peers effectively to a subset of nodes on the network. This allows for more complex configurations, for example facilitating the isolation of nodes or the segmentation of the network.

### 25.2.1. About the BGPAvertisements custom resource

The fields for the **BGPAvertisements** object are defined in the following table:

**Table 25.3. BGPAvertisements configuration**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	Specifies the name for the BGP advertisement.
<b>metadata.namespace</b>	<b>string</b>	Specifies the namespace for the BGP advertisement. Specify the same namespace that the MetalLB Operator uses.
<b>spec.aggregationLength</b>	<b>integer</b>	Optional: Specifies the number of bits to include in a 32-bit CIDR mask. To aggregate the routes that the speaker advertises to BGP peers, the mask is applied to the routes for several service IP addresses and the speaker advertises the aggregated route. For example, with an aggregation length of <b>24</b> , the speaker can aggregate several <b>10.0.1.x/32</b> service IP addresses and advertise a single <b>10.0.1.0/24</b> route.
<b>spec.aggregationLengthV6</b>	<b>integer</b>	Optional: Specifies the number of bits to include in a 128-bit CIDR mask. For example, with an aggregation length of <b>124</b> , the speaker can aggregate several <b>fc00:f853:0ccd:e799::x/128</b> service IP addresses and advertise a single <b>fc00:f853:0ccd:e799::0/124</b> route.

Field	Type	Description
<b>spec.communities</b>	<b>string</b>	<p>Optional: Specifies one or more BGP communities. Each community is specified as two 16-bit values separated by the colon character. Well-known communities must be specified as 16-bit values:</p> <ul style="list-style-type: none"> <li>• <b>NO_EXPORT:65535:65281</b></li> <li>• <b>NO_ADVERTISE:65535:65282</b></li> <li>• <b>NO_EXPORT_SUBCONFED:65535:65283</b></li> </ul> <div style="display: flex; align-items: center;">  <span style="margin-left: 10px;"><b>NOTE</b></span> </div> <p>You can also use community objects that are created along with the strings.</p>
<b>spec.localPref</b>	<b>integer</b>	Optional: Specifies the local preference for this advertisement. This BGP attribute applies to BGP sessions within the Autonomous System.
<b>spec.ipAddressPools</b>	<b>string</b>	Optional: The list of <b>IPAddressPools</b> to advertise with this advertisement, selected by name.
<b>spec.ipAddressPoolSelectors</b>	<b>string</b>	Optional: A selector for the <b>IPAddressPools</b> that gets advertised with this advertisement. This is for associating the <b>IPAddressPool</b> to the advertisement based on the label assigned to the <b>IPAddressPool</b> instead of the name itself. If no <b>IPAddressPool</b> is selected by this or by the list, the advertisement is applied to all the <b>IPAddressPools</b> .
<b>spec.nodeSelectors</b>	<b>string</b>	Optional: <b>NodeSelectors</b> allows to limit the nodes to announce as next hops for the load balancer IP. When empty, all the nodes are announced as next hops.
<b>spec.peers</b>	<b>string</b>	Optional: Use a list to specify the <b>metadata.name</b> values for each <b>BGPPeer</b> resource that receives advertisements for the MetalLB service IP address. The MetalLB service IP address is assigned from the IP address pool. By default, the MetalLB service IP address is advertised to all configured <b>BGPPeer</b> resources. Use this field to limit the advertisement to specific <b>BGPPeer</b> resources.

## 25.2.2. Configuring MetalLB with a BGP advertisement and a basic use case

Configure MetalLB as follows so that the peer BGP routers receive one **203.0.113.200/32** route and one **fc00:f853:ccd:e799::1/128** route for each load-balancer IP address that MetalLB assigns to a service. Because the **localPref** and **communities** fields are not specified, the routes are advertised with **localPref** set to zero and no BGP communities.

### 25.2.2.1. Example: Advertise a basic address pool configuration with BGP

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the BGP protocol.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create an IP address pool.
  - a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-basic
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::124
```

- b. Apply the configuration for the IP address pool:

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

2. Create a BGP advertisement.

- a. Create a file, such as **bgpadvertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAAdvertisement
metadata:
  name: bgpadvertisement-basic
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-basic
```

- b. Apply the configuration:

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

### 25.2.3. Configuring MetalLB with a BGP advertisement and an advanced use case

Configure MetalLB as follows so that MetalLB assigns IP addresses to load-balancer services in the ranges between **203.0.113.200** and **203.0.113.203** and between **fc00:f853:ccd:e799::0** and **fc00:f853:ccd:e799::f**.

To explain the two BGP advertisements, consider an instance when MetalLB assigns the IP address of **203.0.113.200** to a service. With that IP address as an example, the speaker advertises two routes to BGP peers:

- **203.0.113.200/32**, with **localPref** set to **100** and the community set to the numeric value of the **NO\_ADVERTISE** community. This specification indicates to the peer routers that they can use this route but they should not propagate information about this route to BGP peers.
- **203.0.113.200/30**, aggregates the load-balancer IP addresses assigned by MetalLB into a single route. MetalLB advertises the aggregated route to BGP peers with the community attribute set to **8000:800**. BGP peers propagate the **203.0.113.200/30** route to other BGP peers. When traffic is routed to a node with a speaker, the **203.0.113.200/32** route is used to forward the traffic into the cluster and to a pod that is associated with the service.

As you add more services and MetalLB assigns more load-balancer IP addresses from the pool, peer routers receive one local route, **203.0.113.20x/32**, for each service, as well as the **203.0.113.200/30** aggregate route. Each service that you add generates the **/30** route, but MetalLB deduplicates the routes to one BGP advertisement before communicating with peer routers.

### 25.2.3.1. Example: Advertise an advanced address pool configuration with BGP

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the BGP protocol.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create an IP address pool.
  - a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-adv
  labels:
    zone: east
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
  autoAssign: false
```

- b. Apply the configuration for the IP address pool:

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

2. Create a BGP advertisement.
  - a. Create a file, such as **bഗAdvertisement1.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta1
kind: BGPAAdvertisement
metadata:
  name: bgpadvertisement-adv-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 65535:65282
  aggregationLength: 32
  localPref: 100

```

- b. Apply the configuration:

```
$ oc apply -f bgpadvertisement1.yaml
```

- c. Create a file, such as **bgpadvertisement2.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta1
kind: BGPAAdvertisement
metadata:
  name: bgpadvertisement-adv-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 8000:800
  aggregationLength: 30
  aggregationLengthV6: 124

```

- d. Apply the configuration:

```
$ oc apply -f bgpadvertisement2.yaml
```

#### 25.2.4. Advertising an IP address pool from a subset of nodes

To advertise an IP address from an IP addresses pool, from a specific set of nodes only, use the **.spec.nodeSelector** specification in the BGPAAdvertisement custom resource. This specification associates a pool of IP addresses with a set of nodes in the cluster. This is useful when you have nodes on different subnets in a cluster and you want to advertise an IP addresses from an address pool from a specific subnet, for example a public-facing subnet only.

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

##### Procedure

1. Create an IP address pool by using a custom resource:

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400

```

- Control which nodes in the cluster the IP address from **pool1** advertises from by defining the **.spec.nodeSelector** value in the BGPAAdvertisement custom resource:

```

apiVersion: metallb.io/v1beta1
kind: BGPAAdvertisement
metadata:
  name: example
spec:
  ipAddressPools:
    - pool1
  nodeSelector:
    - matchLabels:
        kubernetes.io/hostname: NodeA
    - matchLabels:
        kubernetes.io/hostname: NodeB

```

In this example, the IP address from **pool1** advertises from **NodeA** and **NodeB** only.

### 25.2.5. About the L2Advertisement custom resource

The fields for the **L2Advertisements** object are defined in the following table:

**Table 25.4. L2 advertisements configuration**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	Specifies the name for the L2 advertisement.
<b>metadata.namespace</b>	<b>string</b>	Specifies the namespace for the L2 advertisement. Specify the same namespace that the MetalLB Operator uses.
<b>spec.ipAddressPools</b>	<b>string</b>	Optional: The list of <b>IPAddressPools</b> to advertise with this advertisement, selected by name.
<b>spec.ipAddressPoolSelectors</b>	<b>string</b>	Optional: A selector for the <b>IPAddressPools</b> that gets advertised with this advertisement. This is for associating the <b>IPAddressPool</b> to the advertisement based on the label assigned to the <b>IPAddressPool</b> instead of the name itself. If no <b>IPAddressPool</b> is selected by this or by the list, the advertisement is applied to all the <b>IPAddressPools</b> .

Field	Type	Description
<b>spec.nodeSelectors</b>	<b>string</b>	<p>Optional: <b>NodeSelectors</b> limits the nodes to announce as next hops for the load balancer IP. When empty, all the nodes are announced as next hops.</p>  <p><b>IMPORTANT</b></p> <p>Limiting the nodes to announce as next hops 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.</p> <p>For more information about the support scope of Red Hat Technology Preview features, see <a href="#">Technology Preview Features Support Scope</a></p>
<b>spec.interfaces</b>	<b>string</b>	<p>Optional: The list of <b>interfaces</b> that are used to announce the load balancer IP.</p>

## 25.2.6. Configuring MetalLB with an L2 advertisement

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the L2 protocol.

### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

### Procedure

1. Create an IP address pool.
  - a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
  addresses:
    - 4.4.4.0/24
  autoAssign: false
```

- b. Apply the configuration for the IP address pool:

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

2. Create a L2 advertisement.

a. Create a file, such as **l2advertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-l2
```

b. Apply the configuration:

```
$ oc apply -f l2advertisement.yaml
```

### 25.2.7. Configuring MetalLB with a L2 advertisement and label

The **ipAddressPoolSelectors** field in the **BGPAdvertisement** and **L2Advertisement** custom resource definitions is used to associate the **IPAddressPool** to the advertisement based on the label assigned to the **IPAddressPool** instead of the name itself.

This example shows how to configure MetalLB so that the **IPAddressPool** is advertised with the L2 protocol by configuring the **ipAddressPoolSelectors** field.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create an IP address pool.

a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2-label
  labels:
    zone: east
spec:
  addresses:
    - 172.31.249.87/32
```

b. Apply the configuration for the IP address pool:

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

2. Create a L2 advertisement advertising the IP using **ipAddressPoolSelectors**.

- Create a file, such as **l2advertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement-label
  namespace: metallb-system
spec:
  ipAddressPoolSelectors:
    - matchExpressions:
        - key: zone
          operator: In
          values:
            - east
```

- Apply the configuration:

```
$ oc apply -f l2advertisement.yaml
```

### 25.2.8. Configuring MetalLB with an L2 advertisement for selected interfaces

By default, the IP addresses from IP address pool that has been assigned to the service, is advertised from all the network interfaces. The **interfaces** field in the **L2Advertisement** custom resource definition is used to restrict those network interfaces that advertise the IP address pool.

This example shows how to configure MetalLB so that the IP address pool is advertised only from the network interfaces listed in the **interfaces** field of all nodes.

#### Prerequisites

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

#### Procedure

- Create an IP address pool.
- Create a file, such as **ipaddresspool.yaml**, and enter the configuration details like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
  addresses:
    - 4.4.4.0/24
  autoAssign: false
```

- Apply the configuration for the IP address pool like the following example:

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

2. Create a L2 advertisement advertising the IP with **interfaces** selector.

a. Create a YAML file, such as **l2advertisement.yaml**, and enter the configuration details like the following example:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-l2
  interfaces:
    - interfaceA
    - interfaceB
```

b. Apply the configuration for the advertisement like the following example:

```
$ oc apply -f l2advertisement.yaml
```



### IMPORTANT

The interface selector does not affect how MetalLB chooses the node to announce a given IP by using L2. The chosen node does not announce the service if the node does not have the selected interface.

## 25.2.9. Configuring MetalLB with secondary networks

From OpenShift Container Platform 4.14 the default network behavior is to not allow forwarding of IP packets between network interfaces. Therefore, when MetalLB is configured on a secondary interface, you need to add a machine configuration to enable IP forwarding for only the required interfaces.



### NOTE

OpenShift Container Platform clusters upgraded from 4.13 are not affected because a global parameter is set during upgrade to enable global IP forwarding.

To enable IP forwarding for the secondary interface, you have two options:

- Enable IP forwarding for a specific interface.
- Enable IP forwarding for all interfaces.



### NOTE

Enabling IP forwarding for a specific interface provides more granular control, while enabling it for all interfaces applies a global setting.

### 25.2.9.1. Enabling IP forwarding for a specific interface

#### Procedure

- Patch the Cluster Network Operator, setting the parameter **routingViaHost** to **true**, by running the following command:

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"gatewayConfig": {"routingViaHost": true}}}}}' --type=merge
```

- Enable forwarding for a specific secondary interface, such as **bridge-net** by creating and applying a **MachineConfig** CR:

- Base64-encode the string that is used to configure network kernel parameters by running the following command on your local machine:

```
$ echo -e "net.ipv4.conf.bridge-net.forwarding = 1\nnet.ipv6.conf.bridge-net.forwarding = 1\nnet.ipv4.conf.bridge-net.rp_filter = 0\nnet.ipv6.conf.bridge-net.rp_filter = 0" | base64 -w0
```

### Example output

```
bmV0LmlwdjQuY29uZi5icmlkZ2UtbmV0LmZvcndhcmRpbmcgPSAxCm5IdC5pcHY2LmNvb  
mYuYnJpZGdILW5IdC5mb3J3YXJkaW5nID0gMQpuZXQuaXB2NC5jb25mLmJyaWRnZS1  
uZXQucnBfZmlsdGVyID0gMApuZXQuaXB2Ni5jb25mLmJyaWRnZS1uZXQucnBfZmlsdGV  
yID0gMAo=
```

- Create the **MachineConfig** CR to enable IP forwarding for the specified secondary interface named **bridge-net**.

- Save the following YAML in the **enable-ip-forward.yaml** file:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: <node_role> 1
    name: 81-enable-global-forwarding
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - contents:
            source: data:text/plain;charset=utf-  
8;base64,bmV0LmlwdjQuY29uZi5icmlkZ2UtbmV0LmZvcndhcmRpbmcgPSAxCm5IdC5pcH  
Y2LmNvbYuYnJpZGdILW5IdC5mb3J3YXJkaW5nID0gMQpuZXQuaXB2NC5jb25mLmJy  
aWRnZS1uZXQucnBfZmlsdGVyID0gMApuZXQuaXB2Ni5jb25mLmJyaWRnZS1uZXQucn  
BfZmlsdGVyID0gMAo= 2
            verification: {}
          filesystem: root
          mode: 644
          path: /etc/sysctl.d/enable-global-forwarding.conf
        osImageURL: ""
```

1 Node role where you want to enable IP forwarding, for example, **worker**

- 2** Populate with the generated base64 string

- d. Apply the configuration by running the following command:

```
$ oc apply -f enable-ip-forward.yaml
```

## Verification

- After you apply the machine config, verify the changes by following this procedure:

- Enter into a debug session on the target node by running the following command:

```
$ oc debug node/<node-name>
```

This step instantiates a debug pod called **<node-name>-debug**.

- Set **/host** as the root directory within the debug shell by running the following command:

```
$ chroot /host
```

The debug pod mounts the host's root file system in **/host** within the pod. By changing the root directory to **/host**, you can run binaries contained in the host's executable paths.

- Verify that IP forwarding is enabled by running the following command:

```
$ cat /etc/sysctl.d/enable-global-forwarding.conf
```

## Expected output

```
net.ipv4.conf.bridge-net.forwarding = 1
net.ipv6.conf.bridge-net.forwarding = 1
net.ipv4.conf.bridge-net.rp_filter = 0
net.ipv6.conf.bridge-net.rp_filter = 0
```

The output indicates that IPv4 and IPv6 packet forwarding is enabled on the **bridge-net** interface.

### 25.2.9.2. Enabling IP forwarding globally

- Enable IP forwarding globally by running the following command:

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"gatewayConfig":{"ipForwarding": "Global"}}}}' --type=merge
```

### 25.2.10. Additional resources

- [Configuring a community alias](#).

## 25.3. CONFIGURING METALLB BGP PEERS

As a cluster administrator, you can add, modify, and delete Border Gateway Protocol (BGP) peers. The

MetalLB Operator uses the BGP peer custom resources to identify which peers that MetalLB **speaker** pods contact to start BGP sessions. The peers receive the route advertisements for the load-balancer IP addresses that MetalLB assigns to services.

### 25.3.1. About the BGP peer custom resource

The fields for the BGP peer custom resource are described in the following table.

**Table 25.5. MetalLB BGP peer custom resource**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	Specifies the name for the BGP peer custom resource.
<b>metadata.namespace</b>	<b>string</b>	Specifies the namespace for the BGP peer custom resource.
<b>spec.myASN</b>	<b>integer</b>	Specifies the Autonomous System Number (ASN) for the local end of the BGP session. In all BGP peer custom resources that you add, specify the same value . The range is <b>0</b> to <b>4294967295</b> .
<b>spec.peerASN</b>	<b>integer</b>	Specifies the ASN for the remote end of the BGP session. The range is <b>0</b> to <b>4294967295</b> . If you use this field, you cannot specify a value in the <b>spec.dynamicASN</b> field.
<b>spec.dynamicASN</b>	<b>string</b>	Detects the ASN to use for the remote end of the session without explicitly setting it. Specify <b>internal</b> for a peer with the same ASN, or <b>external</b> for a peer with a different ASN. If you use this field, you cannot specify a value in the <b>spec.peerASN</b> field.
<b>spec.peerAddress</b>	<b>string</b>	Specifies the IP address of the peer to contact for establishing the BGP session.
<b>spec.sourceAddress</b>	<b>string</b>	Optional: Specifies the IP address to use when establishing the BGP session. The value must be an IPv4 address.
<b>spec.peerPort</b>	<b>integer</b>	Optional: Specifies the network port of the peer to contact for establishing the BGP session. The range is <b>0</b> to <b>16384</b> .
<b>spec.holdTime</b>	<b>string</b>	Optional: Specifies the duration for the hold time to propose to the BGP peer. The minimum value is 3 seconds ( <b>3s</b> ). The common units are seconds and minutes, such as <b>3s</b> , <b>1m</b> , and <b>5m30s</b> . To detect path failures more quickly, also configure BFD.
<b>spec.keepaliveTime</b>	<b>string</b>	Optional: Specifies the maximum interval between sending keep-alive messages to the BGP peer. If you specify this field, you must also specify a value for the <b>holdTime</b> field. The specified value must be less than the value for the <b>holdTime</b> field.

Field	Type	Description
<b>spec.routerID</b>	<b>string</b>	Optional: Specifies the router ID to advertise to the BGP peer. If you specify this field, you must specify the same value in every BGP peer custom resource that you add.
<b>spec.password</b>	<b>string</b>	Optional: Specifies the MD5 password to send to the peer for routers that enforce TCP MD5 authenticated BGP sessions.
<b>spec.passwordSecret</b>	<b>string</b>	Optional: Specifies name of the authentication secret for the BGP Peer. The secret must live in the <b>metallb</b> namespace and be of type basic-auth.
<b>spec.bfdProfile</b>	<b>string</b>	Optional: Specifies the name of a BFD profile.
<b>spec.nodeSelectors</b>	<b>object[]</b>	Optional: Specifies a selector, using match expressions and match labels, to control which nodes can connect to the BGP peer.
<b>spec.ebgpMultiHop</b>	<b>boolean</b>	Optional: Specifies that the BGP peer is multiple network hops away. If the BGP peer is not directly connected to the same network, the speaker cannot establish a BGP session unless this field is set to <b>true</b> . This field applies to <i>external BGP</i> . External BGP is the term that is used to describe when a BGP peer belongs to a different Autonomous System.
<b>connectTime</b>	<b>duration</b>	Specifies how long BGP waits between connection attempts to a neighbor.



### NOTE

The **passwordSecret** field is mutually exclusive with the **password** field, and contains a reference to a secret containing the password to use. Setting both fields results in a failure of the parsing.

#### 25.3.2. Configuring a BGP peer

As a cluster administrator, you can add a BGP peer custom resource to exchange routing information with network routers and advertise the IP addresses for services.

##### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Configure MetallLB with a BGP advertisement.

##### Procedure

1. Create a file, such as **bgppeer.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10

```

2. Apply the configuration for the BGP peer:

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

### 25.3.3. Configure a specific set of BGP peers for a given address pool

This procedure illustrates how to:

- Configure a set of address pools (**pool1** and **pool2**).
- Configure a set of BGP peers (**peer1** and **peer2**).
- Configure BGP advertisement to assign **pool1** to **peer1** and **pool2** to **peer2**.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create address pool **pool1**.

- a. Create a file, such as **ipaddresspool1.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400

```

- b. Apply the configuration for the IP address pool **pool1**:

```
$ oc apply -f ipaddresspool1.yaml
```

2. Create address pool **pool2**.

- a. Create a file, such as **ipaddresspool2.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool2
spec:
  addresses:
    - 5.5.100-5.5.5.200
    - 2001:100:5::200-2001:100:5::400

```

- b. Apply the configuration for the IP address pool **pool2**:

```
$ oc apply -f ipaddresspool2.yaml
```

3. Create BGP **peer1**.

- a. Create a file, such as **bgppeer1.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer1
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10

```

- b. Apply the configuration for the BGP peer:

```
$ oc apply -f bgppeer1.yaml
```

4. Create BGP **peer2**.

- a. Create a file, such as **bgppeer2.yaml**, with content like the following example:

```

apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer2
spec:
  peerAddress: 10.0.0.2
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10

```

- b. Apply the configuration for the BGP peer2:

```
$ oc apply -f bgppeer2.yaml
```

5. Create BGP advertisement 1.

- a. Create a file, such as **bgpadvertisement1.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool1
  peers:
    - peer1
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
```

- b. Apply the configuration:

```
$ oc apply -f bgpadvertisement1.yaml
```

## 6. Create BGP advertisement 2.

- a. Create a file, such as **bgpadvertisement2.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool2
  peers:
    - peer2
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
```

- b. Apply the configuration:

```
$ oc apply -f bgpadvertisement2.yaml
```

### 25.3.4. Exposing a service through a network VRF

You can expose a service through a virtual routing and forwarding (VRF) instance by associating a VRF on a network interface with a BGP peer.



## IMPORTANT

Exposing a service through a VRF on a BGP peer 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](#).

By using a VRF on a network interface to expose a service through a BGP peer, you can segregate traffic to the service, configure independent routing decisions, and enable multi-tenancy support on a network interface.



## NOTE

By establishing a BGP session through an interface belonging to a network VRF, MetallLB can advertise services through that interface and enable external traffic to reach the service through this interface. However, the network VRF routing table is different from the default VRF routing table used by OVN-Kubernetes. Therefore, the traffic cannot reach the OVN-Kubernetes network infrastructure.

To enable the traffic directed to the service to reach the OVN-Kubernetes network infrastructure, you must configure routing rules to define the next hops for network traffic. See the **NodeNetworkConfigurationPolicy** resource in "Managing symmetric routing with MetallLB" in the *Additional resources* section for more information.

These are the high-level steps to expose a service through a network VRF with a BGP peer:

1. Define a BGP peer and add a network VRF instance.
2. Specify an IP address pool for MetallLB.
3. Configure a BGP route advertisement for MetallLB to advertise a route using the specified IP address pool and the BGP peer associated with the VRF instance.
4. Deploy a service to test the configuration.

## Prerequisites

- You installed the OpenShift CLI (**oc**).
- You logged in as a user with **cluster-admin** privileges.
- You defined a **NodeNetworkConfigurationPolicy** to associate a Virtual Routing and Forwarding (VRF) instance with a network interface. For more information about completing this prerequisite, see the *Additional resources* section.
- You installed MetallLB on your cluster.

## Procedure

1. Create a **BGPPeer** custom resources (CR):

- Create a file, such as **frrviavrf.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
  peerASN: 200
  peerAddress: 192.168.130.1
  vrf: ens4vrf ①
```

- ① Specifies the network VRF instance to associate with the BGP peer. MetalLB can advertise services and make routing decisions based on the routing information in the VRF.



### NOTE

You must configure this network VRF instance in a **NodeNetworkConfigurationPolicy** CR. See the *Additional resources* for more information.

- Apply the configuration for the BGP peer by running the following command:

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

- Create an **IPAddressPool** CR:

- Create a file, such as **first-pool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
    - 192.169.10.0/32
```

- Apply the configuration for the IP address pool by running the following command:

```
$ oc apply -f first-pool.yaml
```

- Create a **BGPAdvertisement** CR:

- Create a file, such as **first-adv.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: first-adv
  namespace: metallb-system
```

```

spec:
  ipAddressPools:
    - first-pool
  peers:
    - frrviavrf ①

```

- ① In this example, MetalLB advertises a range of IP addresses from the **first-pool** IP address pool to the **frrviavrf** BGP peer.

- b. Apply the configuration for the BGP advertisement by running the following command:

```
$ oc apply -f first-adv.yaml
```

4. Create a **Namespace**, **Deployment**, and **Service** CR:

- a. Create a file, such as **deploy-service.yaml**, with content like the following example:

```

apiVersion: v1
kind: Namespace
metadata:
  name: test
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server
  namespace: test
spec:
  selector:
    matchLabels:
      app: server
  template:
    metadata:
      labels:
        app: server
  spec:
    containers:
      - name: server
        image: registry.redhat.io/ubi9/ubi
        ports:
          - name: http
            containerPort: 30100
            command: ["/bin/sh", "-c"]
            args: ["sleep INF"]
---
apiVersion: v1
kind: Service
metadata:
  name: server1
  namespace: test
spec:
  ports:
    - name: http
      port: 30100
      protocol: TCP

```

```
targetPort: 30100
selector:
  app: server
type: LoadBalancer
```

- b. Apply the configuration for the namespace, deployment, and service by running the following command:

```
$ oc apply -f deploy-service.yaml
```

## Verification

1. Identify a MetalLB speaker pod by running the following command:

```
$ oc get -n metallb-system pods -l component=speaker
```

### Example output

```
NAME      READY  STATUS  RESTARTS  AGE
speaker-c6c5f  6/6   Running  0        69m
```

2. Verify that the state of the BGP session is **Established** in the speaker pod by running the following command, replacing the variables to match your configuration:

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf <vrf_name> neighbor"
```

### Example output

```
BGP neighbor is 192.168.30.1, remote AS 200, local AS 100, external link
BGP version 4, remote router ID 192.168.30.1, local router ID 192.168.30.71
BGP state = Established, up for 04:20:09
```

```
...
```

3. Verify that the service is advertised correctly by running the following command:

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf <vrf_name> ipv4"
```

## Additional resources

- [About virtual routing and forwarding](#)
- [Example: Network interface with a VRF instance node network configuration policy](#)
- [Configuring an egress service](#)
- [Managing symmetric routing with MetalLB](#)

### 25.3.5. Example BGP peer configurations

### 25.3.5.1. Example: Limit which nodes connect to a BGP peer

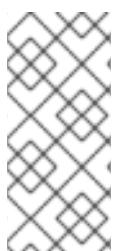
You can specify the node selectors field to control which nodes can connect to a BGP peer.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-nodesel
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  nodeSelectors:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values: [compute-1.example.com, compute-2.example.com]
```

### 25.3.5.2. Example: Specify a BFD profile for a BGP peer

You can specify a BFD profile to associate with BGP peers. BFD complements BGP by providing more rapid detection of communication failures between peers than BGP alone.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-peer-bfd
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  holdTime: "10s"
  bfdProfile: doc-example-bfd-profile-full
```



#### NOTE

Deleting the bidirectional forwarding detection (BFD) profile and removing the **bfdProfile** added to the border gateway protocol (BGP) peer resource does not disable the BFD. Instead, the BGP peer starts using the default BFD profile. To disable BFD from a BGP peer resource, delete the BGP peer configuration and recreate it without a BFD profile. For more information, see [BZ#2050824](#).

### 25.3.5.3. Example: Specify BGP peers for dual-stack networking

To support dual-stack networking, add one BGP peer custom resource for IPv4 and one BGP peer custom resource for IPv6.

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv4
  namespace: metallb-system
```

```

spec:
  peerAddress: 10.0.20.1
  peerASN: 64500
  myASN: 64500
---
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv6
  namespace: metallb-system
spec:
  peerAddress: 2620:52:0:88::104
  peerASN: 64500
  myASN: 64500

```

### 25.3.6. Next steps

- [Configuring services to use MetalLB](#)

## 25.4. CONFIGURING COMMUNITY ALIAS

As a cluster administrator, you can configure a community alias and use it across different advertisements.

### 25.4.1. About the community custom resource

The **community** custom resource is a collection of aliases for communities. Users can define named aliases to be used when advertising **ipAddressPools** using the **BGPAdvertisement**. The fields for the **community** custom resource are described in the following table.



#### NOTE

The **community** CRD applies only to BGPAdvertisement.

**Table 25.6. MetalLB community custom resource**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	Specifies the name for the <b>community</b> .
<b>metadata.namespace</b>	<b>string</b>	Specifies the namespace for the <b>community</b> . Specify the same namespace that the MetalLB Operator uses.
<b>spec.communities</b>	<b>string</b>	Specifies a list of BGP community aliases that can be used in BGPAdvertisements. A community alias consists of a pair of name (alias) and value (number:number). Link the BGPAdvertisement to a community alias by referring to the alias name in its <b>spec.communities</b> field.

**Table 25.7. CommunityAlias**

Field	Type	Description
<b>name</b>	<b>string</b>	The name of the alias for the <b>community</b> .
<b>value</b>	<b>string</b>	The BGP <b>community</b> value corresponding to the given name.

### 25.4.2. Configuring MetalLB with a BGP advertisement and community alias

Configure MetalLB as follows so that the **IPAddressPool** is advertised with the BGP protocol and the community alias set to the numeric value of the NO\_ADVERTISE community.

In the following example, the peer BGP router **doc-example-peer-community** receives one **203.0.113.200/32** route and one **fc00:f853:ccd:e799::1/128** route for each load-balancer IP address that MetalLB assigns to a service. A community alias is configured with the **NO\_ADVERTISE** community.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create an IP address pool.
  - a. Create a file, such as **ipaddresspool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-community
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::1/124
```

- b. Apply the configuration for the IP address pool:

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

2. Create a community alias named **community1**.

```
apiVersion: metallb.io/v1beta1
kind: Community
metadata:
  name: community1
  namespace: metallb-system
spec:
  communities:
    - name: NO_ADVERTISE
      value: '65535:65282'
```

3. Create a BGP peer named **doc-example-bgp-peer**.

- Create a file, such as **bgppeer.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-bgp-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

- Apply the configuration for the BGP peer:

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

4. Create a BGP advertisement with the community alias.

- Create a file, such as **bgpadvertisement.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAAdvertisement
metadata:
  name: bgp-community-sample
  namespace: metallb-system
spec:
  aggregationLength: 32
  aggregationLengthV6: 128
  communities:
    - NO_ADVERTISE ①
  ipAddressPools:
    - doc-example-bgp-community
  peers:
    - doc-example-peer
```

① Specify the **CommunityAlias.name** here and not the community custom resource (CR) name.

- Apply the configuration:

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

## 25.5. CONFIGURING METALLB BFD PROFILES

As a cluster administrator, you can add, modify, and delete Bidirectional Forwarding Detection (BFD) profiles. The MetallB Operator uses the BFD profile custom resources to identify which BGP sessions use BFD to provide faster path failure detection than BGP alone provides.

### 25.5.1. About the BFD profile custom resource

The fields for the BFD profile custom resource are described in the following table.

**Table 25.8. BFD profile custom resource**

Field	Type	Description
<b>metadata.name</b>	<b>string</b>	Specifies the name for the BFD profile custom resource.
<b>metadata.namespace</b>	<b>string</b>	Specifies the namespace for the BFD profile custom resource.
<b>spec.detectMultiplier</b>	<b>integer</b>	<p>Specifies the detection multiplier to determine packet loss. The remote transmission interval is multiplied by this value to determine the connection loss detection timer.</p> <p>For example, when the local system has the detect multiplier set to <b>3</b> and the remote system has the transmission interval set to <b>300</b>, the local system detects failures only after <b>900</b> ms without receiving packets.</p> <p>The range is <b>2</b> to <b>255</b>. The default value is <b>3</b>.</p>
<b>spec.echoMode</b>	<b>boolean</b>	<p>Specifies the echo transmission mode. If you are not using distributed BFD, echo transmission mode works only when the peer is also FRR. The default value is <b>false</b> and echo transmission mode is disabled.</p> <p>When echo transmission mode is enabled, consider increasing the transmission interval of control packets to reduce bandwidth usage. For example, consider increasing the transmit interval to <b>2000</b> ms.</p>
<b>spec.echoInterval</b>	<b>integer</b>	<p>Specifies the minimum transmission interval, less jitter, that this system uses to send and receive echo packets. The range is <b>10</b> to <b>60000</b>. The default value is <b>50</b> ms.</p>
<b>spec.minimumTtl</b>	<b>integer</b>	<p>Specifies the minimum expected TTL for an incoming control packet. This field applies to multi-hop sessions only.</p> <p>The purpose of setting a minimum TTL is to make the packet validation requirements more stringent and avoid receiving control packets from other sessions.</p> <p>The default value is <b>254</b> and indicates that the system expects only one hop between this system and the peer.</p>

Field	Type	Description
<b>spec.passiveMode</b>	<b>boolean</b>	<p>Specifies whether a session is marked as active or passive. A passive session does not attempt to start the connection. Instead, a passive session waits for control packets from a peer before it begins to reply.</p> <p>Marking a session as passive is useful when you have a router that acts as the central node of a star network and you want to avoid sending control packets that you do not need the system to send.</p> <p>The default value is <b>false</b> and marks the session as active.</p>
<b>spec.receiveInterval</b>	<b>integer</b>	Specifies the minimum interval that this system is capable of receiving control packets. The range is <b>10</b> to <b>60000</b> . The default value is <b>300</b> ms.
<b>spec.transmitInterval</b>	<b>integer</b>	Specifies the minimum transmission interval, less jitter, that this system uses to send control packets. The range is <b>10</b> to <b>60000</b> . The default value is <b>300</b> ms.

### 25.5.2. Configuring a BFD profile

As a cluster administrator, you can add a BFD profile and configure a BGP peer to use the profile. BFD provides faster path failure detection than BGP alone.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

#### Procedure

1. Create a file, such as **bfdprofile.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BFDProfile
metadata:
  name: doc-example-bfd-profile-full
  namespace: metallb-system
spec:
  receiveInterval: 300
  transmitInterval: 300
  detectMultiplier: 3
  echoMode: false
  passiveMode: true
  minimumTtl: 254
```

2. Apply the configuration for the BFD profile:

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

### 25.5.3. Next steps

- [Configure a BGP peer](#) to use the BFD profile.

## 25.6. CONFIGURING SERVICES TO USE METALLB

As a cluster administrator, when you add a service of type **LoadBalancer**, you can control how MetalLB assigns an IP address.

### 25.6.1. Request a specific IP address

Like some other load-balancer implementations, MetalLB accepts the **spec.loadBalancerIP** field in the service specification.

If the requested IP address is within a range from any address pool, MetalLB assigns the requested IP address. If the requested IP address is not within any range, MetalLB reports a warning.

#### Example service YAML for a specific IP address

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.io/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: <ip_address>
```

If MetalLB cannot assign the requested IP address, the **EXTERNAL-IP** for the service reports **<pending>** and running **oc describe service <service\_name>** includes an event like the following example.

#### Example event when MetalLB cannot assign a requested IP address

```
...
Events:
Type Reason Age From Message
---- ---- -- -----
Warning AllocationFailed 3m16s metallb-controller Failed to allocate IP for "default/invalid-request": "4.3.2.1" is not allowed in config
```

### 25.6.2. Request an IP address from a specific pool

To assign an IP address from a specific range, but you are not concerned with the specific IP address, then you can use the **metallb.io/address-pool** annotation to request an IP address from the specified address pool.

#### Example service YAML for an IP address from a specific pool

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.io/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

If the address pool that you specify for **<address\_pool\_name>** does not exist, MetalLB attempts to assign an IP address from any pool that permits automatic assignment.

#### 25.6.3. Accept any IP address

By default, address pools are configured to permit automatic assignment. MetalLB assigns an IP address from these address pools.

To accept any IP address from any pool that is configured for automatic assignment, no special annotation or configuration is required.

#### Example service YAML for accepting any IP address

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

#### 25.6.4. Share a specific IP address

By default, services do not share IP addresses. However, if you need to colocate services on a single IP address, you can enable selective IP sharing by adding the **metallb.io/allow-shared-ip** annotation to the services.

```
apiVersion: v1
```

```

kind: Service
metadata:
  name: service-http
  annotations:
    metallb.io/address-pool: doc-example
    metallb.io/allow-shared-ip: "web-server-svc" ①
spec:
  ports:
    - name: http
      port: 80 ②
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value> ③
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7 ④
---
apiVersion: v1
kind: Service
metadata:
  name: service-https
  annotations:
    metallb.io/address-pool: doc-example
    metallb.io/allow-shared-ip: "web-server-svc"
spec:
  ports:
    - name: https
      port: 443
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value>
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7

```

- ① Specify the same value for the **metallb.io/allow-shared-ip** annotation. This value is referred to as the *sharing key*.
- ② Specify different port numbers for the services.
- ③ Specify identical pod selectors if you must specify **externalTrafficPolicy: local** so the services send traffic to the same set of pods. If you use the **cluster** external traffic policy, then the pod selectors do not need to be identical.
- ④ Optional: If you specify the three preceding items, MetalLB might colocate the services on the same IP address. To ensure that services share an IP address, specify the IP address to share.

By default, Kubernetes does not allow multiprotocol load balancer services. This limitation would normally make it impossible to run a service like DNS that needs to listen on both TCP and UDP. To work around this limitation of Kubernetes with MetalLB, create two services:

- For one service, specify TCP and for the second service, specify UDP.
- In both services, specify the same pod selector.

- Specify the same sharing key and `spec.loadBalancerIP` value to colocate the TCP and UDP services on the same IP address.

## 25.6.5. Configuring a service with MetalLB

You can configure a load-balancing service to use an external IP address from an address pool.

### Prerequisites

- Install the OpenShift CLI (`oc`).
- Install the MetalLB Operator and start MetalLB.
- Configure at least one address pool.
- Configure your network to route traffic from the clients to the host network for the cluster.

### Procedure

- Create a `<service_name>.yaml` file. In the file, ensure that the `spec.type` field is set to **LoadBalancer**.  
Refer to the examples for information about how to request the external IP address that MetalLB assigns to the service.
- Create the service:

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

### Example output

```
service/<service_name> created
```

### Verification

- Describe the service:

```
$ oc describe service <service_name>
```

### Example output

```
Name:          <service_name>
Namespace:     default
Labels:        <none>
Annotations:   metallb.io/address-pool: doc-example ①
Selector:      app=service_name
Type:          LoadBalancer ②
IP Family Policy: SingleStack
IP Families:   IPv4
IP:            10.105.237.254
IPs:           10.105.237.254
LoadBalancer Ingress: 192.168.100.5 ③
Port:          <unset> 80/TCP
TargetPort:    8080/TCP
```

```

NodePort: <unset> 30550/TCP
Endpoints: 10.244.0.50:8080
Session Affinity: None
External Traffic Policy: Cluster
Events: 4
  Type Reason Age From Message
  ---- ---- -- -----
Normal nodeAssigned 32m (x2 over 32m) metallb-speaker announcing from node "
<node_name>"
```

- 1 The annotation is present if you request an IP address from a specific pool.
- 2 The service type must indicate **LoadBalancer**.
- 3 The load-balancer ingress field indicates the external IP address if the service is assigned correctly.
- 4 The events field indicates the node name that is assigned to announce the external IP address. If you experience an error, the events field indicates the reason for the error.

## 25.7. MANAGING SYMMETRIC ROUTING WITH METALLB

As a cluster administrator, you can effectively manage traffic for pods behind a MetalLB load-balancer service with multiple host interfaces by implementing features from MetalLB, NMState, and OVN-Kubernetes. By combining these features in this context, you can provide symmetric routing, traffic segregation, and support clients on different networks with overlapping CIDR addresses.

To achieve this functionality, learn how to implement virtual routing and forwarding (VRF) instances with MetalLB, and configure egress services.



### IMPORTANT

Configuring symmetric traffic by using a VRF instance with MetalLB and an egress service 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](#).

### 25.7.1. Challenges of managing symmetric routing with MetalLB

When you use MetalLB with multiple host interfaces, MetalLB exposes and announces a service through all available interfaces on the host. This can present challenges relating to network isolation, asymmetric return traffic and overlapping CIDR addresses.

One option to ensure that return traffic reaches the correct client is to use static routes. However, with this solution, MetalLB cannot isolate the services and then announce each service through a different interface. Additionally, static routing requires manual configuration and requires maintenance if remote sites are added.

A further challenge of symmetric routing when implementing a MetalLB service is scenarios where

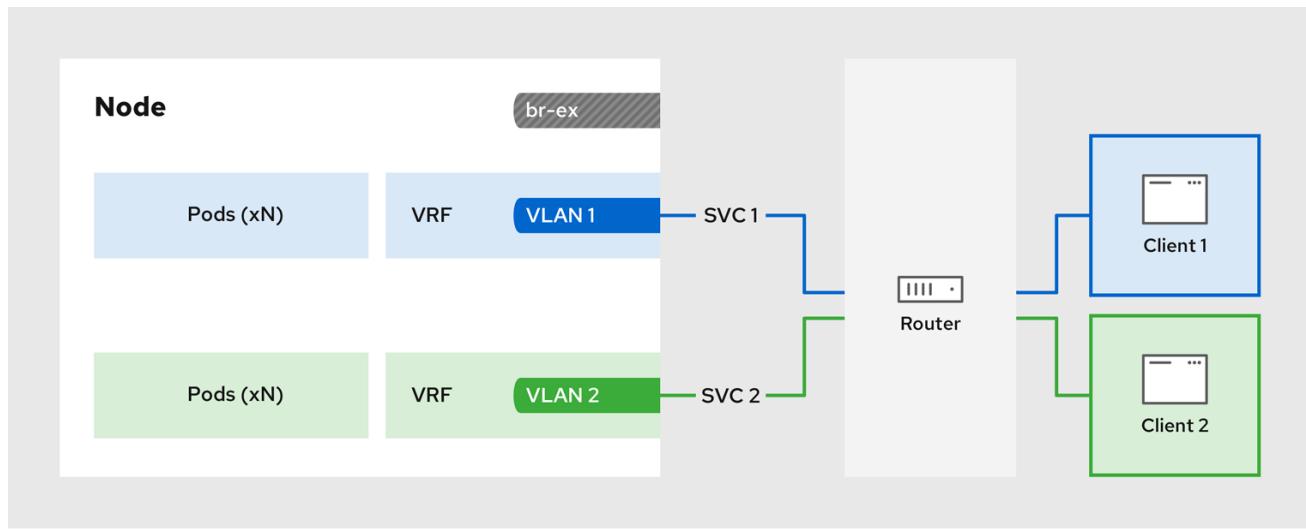
external systems expect the source and destination IP address for an application to be the same. The default behavior for OpenShift Container Platform is to assign the IP address of the host network interface as the source IP address for traffic originating from pods. This is problematic with multiple host interfaces.

You can overcome these challenges by implementing a configuration that combines features from MetalLB, NMState, and OVN-Kubernetes.

### 25.7.2. Overview of managing symmetric routing by using VRFs with MetalLB

You can overcome the challenges of implementing symmetric routing by using NMState to configure a VRF instance on a host, associating the VRF instance with a MetalLB **BGPPeer** resource, and configuring an egress service for egress traffic with OVN-Kubernetes.

**Figure 25.1. Network overview of managing symmetric routing by using VRFs with MetalLB**



357\_OpenShift\_0823

The configuration process involves three stages:

#### 1. Define a VRF and routing rules

- Configure a **NodeNetworkConfigurationPolicy** custom resource (CR) to associate a VRF instance with a network interface.
- Use the VRF routing table to direct ingress and egress traffic.

#### 2. Link the VRF to a MetalLB**BGPPeer**

- Configure a MetalLB **BGPPeer** resource to use the VRF instance on a network interface.
- By associating the **BGPPeer** resource with the VRF instance, the designated network interface becomes the primary interface for the BGP session, and MetalLB advertises the services through this interface.

#### 3. Configure an egress service

- Configure an egress service to choose the network associated with the VRF instance for egress traffic.

- Optional: Configure an egress service to use the IP address of the MetalLB load-balancer service as the source IP for egress traffic.

### 25.7.3. Configuring symmetric routing by using VRFs with MetalLB

You can configure symmetric network routing for applications behind a MetalLB service that require the same ingress and egress network paths.

This example associates a VRF routing table with MetalLB and an egress service to enable symmetric routing for ingress and egress traffic for pods behind a **LoadBalancer** service.



#### NOTE

- If you use the **sourceIPBy: "LoadBalancerIP"** setting in the **EgressService** CR, you must specify the load-balancer node in the **BGPAdvertisement** custom resource (CR).
- You can use the **sourceIPBy: "Network"** setting on clusters that use OVN-Kubernetes configured with the **gatewayConfig.routingViaHost** specification set to **true** only. Additionally, if you use the **sourceIPBy: "Network"** setting, you must schedule the application workload on nodes configured with the network VRF instance.

#### Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the Kubernetes NMState Operator.
- Install the MetalLB Operator.

#### Procedure

1. Create a **NodeNetworkConfigurationPolicy** CR to define the VRF instance:
  - a. Create a file, such as **node-network-vrf.yaml**, with content like the following example:

```
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
- name: ens4 ⑦
  type: ethernet
  state: up
  ipv4:
    address:
      - ip: 192.168.130.130
        prefix-length: 24
      dhcp: false
      enabled: true
  routes: ⑧
    config:
      - destination: 0.0.0.0/0
        metric: 150
        next-hop-address: 192.168.130.1
        next-hop-interface: ens4
        table-id: 2
  route-rules: ⑨
    config:
      - ip-to: 172.30.0.0/16
        priority: 998
        route-table: 254 ⑩
      - ip-to: 10.132.0.0/14
        priority: 998
        route-table: 254
      - ip-to: 169.254.0.0/17
        priority: 998
        route-table: 254

```

- 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 that the VRF attaches to.
- ⑥ The name of the route table ID for the VRF.
- ⑦ The IPv4 address of the interface associated with the VRF.
- ⑧ Defines the configuration for network routes. The **next-hop-address** field defines the IP address of the next hop for the route. The **next-hop-interface** field defines the outgoing interface for the route. In this example, the VRF routing table is **2**, which references the ID that you define in the **EgressService** CR.
- ⑨ Defines additional route rules. The **ip-to** fields must match the **Cluster Network** CIDR, **Service Network** CIDR, and **Internal Masquerade** subnet CIDR. You can view the values for these CIDR address specifications by running the following command: **oc describe network.operator/cluster**.
- ⑩ The main routing table that the Linux kernel uses when calculating routes has the ID **254**.

- b. Apply the policy by running the following command:

```
$ oc apply -f node-network-vrf.yaml
```

2. Create a **BGPPeer** custom resource (CR):

- a. Create a file, such as **frr-via-vrf.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
  peerASN: 200
  peerAddress: 192.168.130.1
  vrf: ens4vrf ①
```

① Specifies the VRF instance to associate with the BGP peer. MetalLB can advertise services and make routing decisions based on the routing information in the VRF.

- b. Apply the configuration for the BGP peer by running the following command:

```
$ oc apply -f frr-via-vrf.yaml
```

3. Create an **IPAddressPool** CR:

- a. Create a file, such as **first-pool.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
    - 192.169.10.0/32
```

- b. Apply the configuration for the IP address pool by running the following command:

```
$ oc apply -f first-pool.yaml
```

4. Create a **BGPAdvertisement** CR:

- a. Create a file, such as **first-adv.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: first-adv
  namespace: metallb-system
spec:
  ipAddressPools:
```

```

    - first-pool
  peers:
    - frrviavrf ①
  nodeSelectors:
    - matchLabels:
      egress-service.k8s.ovn.org/test-server1: "" ②

```

- ① In this example, MetalLB advertises a range of IP addresses from the **first-pool** IP address pool to the **frrviavrf** BGP peer.
- ② In this example, the **EgressService** CR configures the source IP address for egress traffic to use the load-balancer service IP address. Therefore, you must specify the load-balancer node for return traffic to use the same return path for the traffic originating from the pod.

b. Apply the configuration for the BGP advertisement by running the following command:

```
$ oc apply -f first-adv.yaml
```

5. Create an **EgressService** CR:

a. Create a file, such as **egress-service.yaml**, with content like the following example:

```

apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: server1 ①
  namespace: test ②
spec:
  sourceIPBy: "LoadBalancerIP" ③
  nodeSelector:
    matchLabels:
      vrf: "true" ④
  network: "2" ⑤

```

- ① Specify the name for the egress service. The name of the **EgressService** resource must match the name of the load-balancer service that you want to modify.
- ② Specify the namespace for the egress service. The namespace for the **EgressService** must match the namespace of the load-balancer service that you want to modify. The egress service is namespace-scoped.
- ③ This example assigns the **LoadBalancer** service ingress IP address as the source IP address for egress traffic.
- ④ If you specify **LoadBalancer** for the **sourceIPBy** specification, a single node handles the **LoadBalancer** service traffic. In this example, only a node with the label **vrf: "true"** can handle the service traffic. If you do not specify a node, OVN-Kubernetes selects a worker node to handle the service traffic. When a node is selected, OVN-Kubernetes labels the node in the following format: **egress-service.k8s.ovn.org/<svc\_namespace>-<svc\_name>: ""**.
- ⑤ Specify the routing table ID for egress traffic. Ensure that the value matches the

- b. Apply the configuration for the egress service by running the following command:

```
$ oc apply -f egress-service.yaml
```

## Verification

- Verify that you can access the application endpoint of the pods running behind the MetalLB service by running the following command:

```
$ curl <external_ip_address>:<port_number> ①
```

- Update the external IP address and port number to suit your application endpoint.

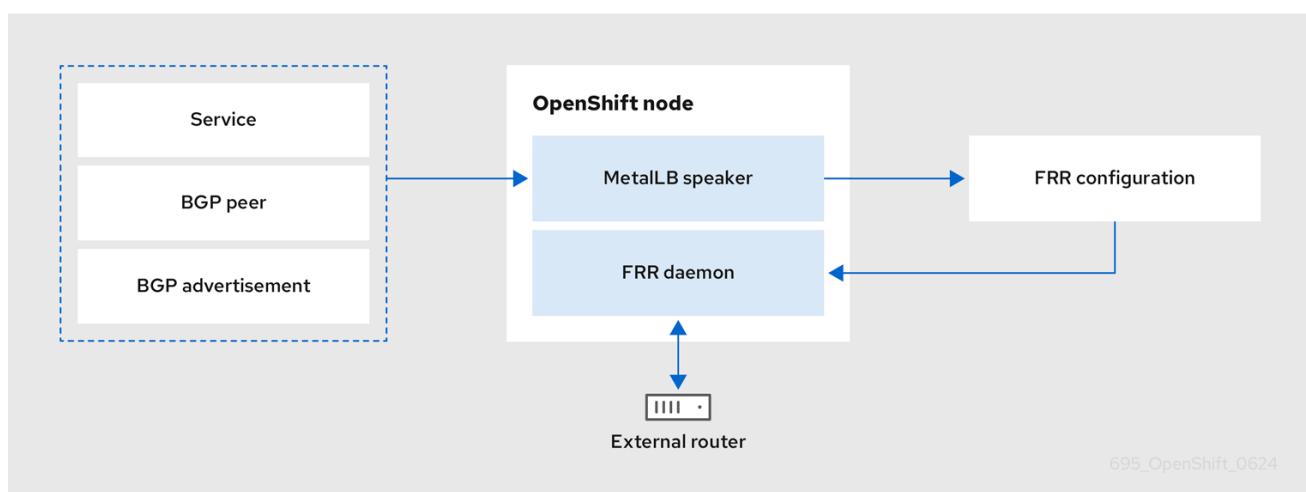
- Optional: If you assigned the **LoadBalancer** service ingress IP address as the source IP address for egress traffic, verify this configuration by using tools such as **tcpdump** to analyze packets received at the external client.

## Additional resources

- [About virtual routing and forwarding](#)
- [Exposing a service through a network VRF](#)
- [Example: Network interface with a VRF instance node network configuration policy](#)
- [Configuring an egress service](#)

## 25.8. CONFIGURING THE INTEGRATION OF METALLB AND FRR-K8S

FRRouting (FRR) is a free, open source internet routing protocol suite for Linux and UNIX platforms. **FRR-K8s** is a Kubernetes based DaemonSet that exposes a subset of the **FRR** API in a Kubernetes-compliant manner. As a cluster administrator, you can use the **FRRConfiguration** custom resource (CR) to access some of the FRR services not provided by MetalLB, for example, receiving routes. **MetalLB** generates the **FRR-K8s** configuration corresponding to the MetalLB configuration applied.



695\_OpenShift\_0624

**WARNING**

When configuring Virtual Route Forwarding (VRF) users must change their VRFs to a table ID lower than 1000 as higher than 1000 is reserved for OpenShift Container Platform.

### 25.8.1. FRR configurations

You can create multiple **FRRConfiguration** CRs to use **FRR** services in **MetalLB**. **MetalLB** generates an **FRRConfiguration** object which **FRR-K8s** merges with all other configurations that all users have created.

For example, you can configure **FRR-K8s** to receive all of the prefixes advertised by a given neighbor. The following example configures **FRR-K8s** to receive all of the prefixes advertised by a **BGPPeer** with host **172.18.0.5**:

#### Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: metallb-system
spec:
  bgp:
    routers:
      - asn: 64512
        neighbors:
          - address: 172.18.0.5
            asn: 64512
        toReceive:
          allowed:
            mode: all
```

You can also configure FRR-K8s to always block a set of prefixes, regardless of the configuration applied. This can be useful to avoid routes towards the pods or **ClusterIPs** CIDRs that might result in cluster malfunctions. The following example blocks the set of prefixes **192.168.1.0/24**:

#### Example MetalLB CR

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  bgpBackend: frr-k8s
  frrk8sConfig:
    alwaysBlock:
      - 192.168.1.0/24
```

You can set **FRR-K8s** to block the **Cluster Network** CIDR and **Service Network** CIDR. You can view the values for these CIDR address specifications by running the following command:

```
$ oc describe network.config/cluster
```

## 25.8.2. Configuring the FRRConfiguration CRD

The following section provides reference examples that use the **FRRConfiguration** custom resource (CR).

### 25.8.2.1. The routers field

You can use the **routers** field to configure multiple routers, one for each Virtual Routing and Forwarding (VRF) resource. For each router, you must define the Autonomous System Number (ASN).

You can also define a list of Border Gateway Protocol (BGP) neighbors to connect to, as in the following example:

#### Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
      - asn: 64512
        neighbors:
          - address: 172.30.0.3
            asn: 4200000000
            ebgpMultiHop: true
            port: 180
          - address: 172.18.0.6
            asn: 4200000000
            port: 179
```

### 25.8.2.2. The toAdvertise field

By default, **FRR-K8s** does not advertise the prefixes configured as part of a router configuration. In order to advertise them, you use the **toAdvertise** field.

You can advertise a subset of the prefixes, as in the following example:

#### Example FRRConfiguration CR

```
apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
```

```

routers:
- asn: 64512
neighbors:
- address: 172.30.0.3
asn: 4200000000
ebgpMultiHop: true
port: 180
toAdvertise:
allowed:
prefixes: ①
- 192.168.2.0/24
prefixes:
- 192.168.2.0/24
- 192.169.2.0/24

```

- ① Advertises a subset of prefixes.

The following example shows you how to advertise all of the prefixes:

#### Example FRRConfiguration CR

```

apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
name: test
namespace: frr-k8s-system
spec:
bgp:
routers:
- asn: 64512
neighbors:
- address: 172.30.0.3
asn: 4200000000
ebgpMultiHop: true
port: 180
toAdvertise:
allowed:
mode: all ①
prefixes:
- 192.168.2.0/24
- 192.169.2.0/24

```

- ① Advertises all prefixes.

#### 25.8.2.3. The toReceive field

By default, **FRR-K8s** does not process any prefixes advertised by a neighbor. You can use the **toReceive** field to process such addresses.

You can configure for a subset of the prefixes, as in this example:

#### Example FRRConfiguration CR

```

apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
      - asn: 64512
        neighbors:
          - address: 172.18.0.5
            asn: 64512
            port: 179
            toReceive:
              allowed:
                prefixes:
                  - prefix: 192.168.1.0/24
                  - prefix: 192.169.2.0/24
                    ge: 25 1
                    le: 28 2

```

- 1 2The prefix is applied if the prefix length is less than or equal to the **le** prefix length and greater than or equal to the **ge** prefix length.

The following example configures FRR to handle all the prefixes announced:

#### Example FRRConfiguration CR

```

apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
      - asn: 64512
        neighbors:
          - address: 172.18.0.5
            asn: 64512
            port: 179
            toReceive:
              allowed:
                mode: all

```

#### 25.8.2.4. The **bgp** field

You can use the **bgp** field to define various **BFD** profiles and associate them with a neighbor. In the following example, **BFD** backs up the **BGP** session and **FRR** can detect link failures:

#### Example FRRConfiguration CR

```

apiVersion: frrk8s.metallb.io/v1beta1

```

```

kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
      - asn: 64512
        neighbors:
          - address: 172.30.0.3
            asn: 64512
            port: 180
            bfdProfile: defaultprofile
        bfdProfiles:
          - name: defaultprofile

```

#### 25.8.2.5. The nodeSelector field

By default, **FRR-K8s** applies the configuration to all nodes where the daemon is running. You can use the **nodeSelector** field to specify the nodes to which you want to apply the configuration. For example:

#### Example FRRConfiguration CR

```

apiVersion: frrk8s.metallb.io/v1beta1
kind: FRRConfiguration
metadata:
  name: test
  namespace: frr-k8s-system
spec:
  bgp:
    routers:
      - asn: 64512
    nodeSelector:
      labelSelector:
        foo: "bar"

```

The fields for the **FRRConfiguration** custom resource are described in the following table:

**Table 25.9. MetalLB FRRConfiguration custom resource**

Field	Type	Description
<b>spec.bgp.routers</b>	<b>array</b>	Specifies the routers that FRR is to configure (one per VRF).
<b>spec.bgp.routers.asn</b>	<b>integer</b>	The Autonomous System Number (ASN) to use for the local end of the session.
<b>spec.bgp.routers.id</b>	<b>string</b>	Specifies the ID of the <b>bgp</b> router.
<b>spec.bgp.routers.vrf</b>	<b>string</b>	Specifies the host vrf used to establish sessions from this router.

Field	Type	Description
<code>spec.bgp.router.s.neighbors</code>	<code>array</code>	Specifies the neighbors to establish BGP sessions with.
<code>spec.bgp.router.s.neighbors.asn</code>	<code>integer</code>	Specifies the ASN to use for the remote end of the session. If you use this field, you cannot specify a value in the <code>spec.bgp.routers.neighbors.dynamicASN</code> field.
<code>spec.bgp.router.s.neighbors.dynamicASN</code>	<code>string</code>	Detects the ASN to use for the remote end of the session without explicitly setting it. Specify <b>internal</b> for a neighbor with the same ASN, or <b>external</b> for a neighbor with a different ASN. If you use this field, you cannot specify a value in the <code>spec.bgp.routers.neighbors.asn</code> field.
<code>spec.bgp.router.s.neighbors.address</code>	<code>string</code>	Specifies the IP address to establish the session with.
<code>spec.bgp.router.s.neighbors.port</code>	<code>integer</code>	Specifies the port to dial when establishing the session. Defaults to 179.
<code>spec.bgp.router.s.neighbors.password</code>	<code>string</code>	Specifies the password to use for establishing the BGP session. <b>Password</b> and <b>PasswordSecret</b> are mutually exclusive.
<code>spec.bgp.router.s.neighbors.passwordSecret</code>	<code>string</code>	Specifies the name of the authentication secret for the neighbor. The secret must be of type "kubernetes.io/basic-auth", and in the same namespace as the FRR-K8s daemon. The key "password" stores the password in the secret. <b>Password</b> and <b>PasswordSecret</b> are mutually exclusive.
<code>spec.bgp.router.s.neighbors.holdTime</code>	<code>duration</code>	Specifies the requested BGP hold time, per RFC4271. Defaults to 180s.
<code>spec.bgp.router.s.neighbors.keepaliveTime</code>	<code>duration</code>	Specifies the requested BGP keepalive time, per RFC4271. Defaults to <b>60s</b> .
<code>spec.bgp.router.s.neighbors.connectTime</code>	<code>duration</code>	Specifies how long BGP waits between connection attempts to a neighbor.
<code>spec.bgp.router.s.neighbors.ebgpMultiHop</code>	<code>boolean</code>	Indicates if the BGP Peer is multi-hops away.

Field	Type	Description
<b>spec.bgp.router.s.neighbors.bfdProfile</b>	<b>string</b>	Specifies the name of the BFD Profile to use for the BFD session associated with the BGP session. If not set, the BFD session is not set up.
<b>spec.bgp.router.s.neighbors.toAdvertiser.allowed</b>	<b>array</b>	Represents the list of prefixes to advertise to a neighbor, and the associated properties.
<b>spec.bgp.router.s.neighbors.toAdvertiser.allowed.prefixes</b>	<b>string array</b>	Specifies the list of prefixes to advertise to a neighbor. This list must match the prefixes that you define in the router.
<b>spec.bgp.router.s.neighbors.toAdvertiser.allowed.mode</b>	<b>string</b>	Specifies the mode to use when handling the prefixes. You can set to <b>filtered</b> to allow only the prefixes in the prefixes list. You can set to <b>all</b> to allow all the prefixes configured on the router.
<b>spec.bgp.router.s.neighbors.toAdvertiser.withLocalPref</b>	<b>array</b>	Specifies the prefixes associated with an advertised local preference. You must specify the prefixes associated with a local preference in the prefixes allowed to be advertised.
<b>spec.bgp.router.s.neighbors.toAdvertiser.withLocalPref.prefixes</b>	<b>string array</b>	Specifies the prefixes associated with the local preference.
<b>spec.bgp.router.s.neighbors.toAdvertiser.withLocalPref.localPref</b>	<b>integer</b>	Specifies the local preference associated with the prefixes.
<b>spec.bgp.router.s.neighbors.toAdvertiser.withCommunity</b>	<b>array</b>	Specifies the prefixes associated with an advertised BGP community. You must include the prefixes associated with a local preference in the list of prefixes that you want to advertise.
<b>spec.bgp.router.s.neighbors.toAdvertiser.withCommunity.prefixes</b>	<b>string array</b>	Specifies the prefixes associated with the community.

Field	Type	Description
<b>spec.bgp.router.s.neighbors.toAdvertisewithCommunity.community</b>	<b>string</b>	Specifies the community associated with the prefixes.
<b>spec.bgp.router.s.neighbors.toReceive</b>	<b>array</b>	Specifies the prefixes to receive from a neighbor.
<b>spec.bgp.router.s.neighbors.toReceive.allowed</b>	<b>array</b>	Specifies the information that you want to receive from a neighbor.
<b>spec.bgp.router.s.neighbors.toReceive.allowed.prefixes</b>	<b>array</b>	Specifies the prefixes allowed from a neighbor.
<b>spec.bgp.router.s.neighbors.toReceive.allowed.mode</b>	<b>string</b>	Specifies the mode to use when handling the prefixes. When set to <b>filtered</b> , only the prefixes in the <b>prefixes</b> list are allowed. When set to <b>all</b> , all the prefixes configured on the router are allowed.
<b>spec.bgp.router.s.neighbors.disableMP</b>	<b>boolean</b>	Disables MP BGP to prevent it from separating IPv4 and IPv6 route exchanges into distinct BGP sessions.
<b>spec.bgp.router.s.prefixes</b>	<b>string array</b>	Specifies all prefixes to advertise from this router instance.
<b>spec.bgp.bfdProfiles</b>	<b>array</b>	Specifies the list of bfd profiles to use when configuring the neighbors.
<b>spec.bgp.bfdProfiles.name</b>	<b>string</b>	The name of the BFD Profile to be referenced in other parts of the configuration.
<b>spec.bgp.bfdProfiles.receiveInterval</b>	<b>integer</b>	Specifies the minimum interval at which this system can receive control packets, in milliseconds. Defaults to <b>300ms</b> .
<b>spec.bgp.bfdProfiles.transmitInterval</b>	<b>integer</b>	Specifies the minimum transmission interval, excluding jitter, that this system wants to use to send BFD control packets, in milliseconds. Defaults to <b>300ms</b> .
<b>spec.bgp.bfdProfiles.detectMultiplier</b>	<b>integer</b>	Configures the detection multiplier to determine packet loss. To determine the connection loss-detection timer, multiply the remote transmission interval by this value.

Field	Type	Description
<b>spec.bgp.bfdProfiles.echoInterval</b>	<b>integer</b>	Configures the minimal echo receive transmission-interval that this system can handle, in milliseconds. Defaults to <b>50ms</b> .
<b>spec.bgp.bfdProfiles.echoMode</b>	<b>boolean</b>	Enables or disables the echo transmission mode. This mode is disabled by default, and not supported on multihop setups.
<b>spec.bgp.bfdProfiles.passiveMode</b>	<b>boolean</b>	Mark session as passive. A passive session does not attempt to start the connection and waits for control packets from peers before it begins replying.
<b>spec.bgp.bfdProfiles.MinimumTtl</b>	<b>integer</b>	For multihop sessions only. Configures the minimum expected TTL for an incoming BFD control packet.
<b>spec.nodeSelector</b>	<b>string</b>	Limits the nodes that attempt to apply this configuration. If specified, only those nodes whose labels match the specified selectors attempt to apply the configuration. If it is not specified, all nodes attempt to apply this configuration.
<b>status</b>	<b>string</b>	Defines the observed state of FRRConfiguration.

### 25.8.3. How FRR-K8s merges multiple configurations

In a case where multiple users add configurations that select the same node, **FRR-K8s** merges the configurations. Each configuration can only extend others. This means that it is possible to add a new neighbor to a router, or to advertise an additional prefix to a neighbor, but not possible to remove a component added by another configuration.

#### 25.8.3.1. Configuration conflicts

Certain configurations can cause conflicts, leading to errors, for example:

- different ASN for the same router (in the same VRF)
- different ASN for the same neighbor (with the same IP / port)
- multiple BFD profiles with the same name but different values

When the daemon finds an invalid configuration for a node, it reports the configuration as invalid and reverts to the previous valid **FRR** configuration.

#### 25.8.3.2. Merging

When merging, it is possible to do the following actions:

- Extend the set of IPs that you want to advertise to a neighbor.

- Add an extra neighbor with its set of IPs.
- Extend the set of IPs to which you want to associate a community.
- Allow incoming routes for a neighbor.

Each configuration must be self contained. This means, for example, that it is not possible to allow prefixes that are not defined in the router section by leveraging prefixes coming from another configuration.

If the configurations to be applied are compatible, merging works as follows:

- **FRR-K8s** combines all the routers.
- **FRR-K8s** merges all prefixes and neighbors for each router.
- **FRR-K8s** merges all filters for each neighbor.



### NOTE

A less restrictive filter has precedence over a stricter one. For example, a filter accepting some prefixes has precedence over a filter not accepting any, and a filter accepting all prefixes has precedence over one that accepts some.

## 25.9. METALLB LOGGING, TROUBLESHOOTING, AND SUPPORT

If you need to troubleshoot MetalLB configuration, see the following sections for commonly used commands.

### 25.9.1. Setting the MetalLB logging levels

MetalLB uses FRRouting (FRR) in a container with the default setting of **info** generates a lot of logging. You can control the verbosity of the logs generated by setting the **logLevel** as illustrated in this example.

Gain a deeper insight into MetalLB by setting the **logLevel** to **debug** as follows:

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

#### Procedure

1. Create a file, such as **setdebugloglevel.yaml**, with content like the following example:

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
```

```
logLevel: debug
nodeSelector:
  node-role.kubernetes.io/worker: ""
```

2. Apply the configuration:

```
$ oc replace -f setdebugloglevel.yaml
```



#### NOTE

Use **oc replace** as the understanding is the **metallb** CR is already created and here you are changing the log level.

3. Display the names of the **speaker** pods:

```
$ oc get -n metallb-system pods -l component=speaker
```

#### Example output

NAME	READY	STATUS	RESTARTS	AGE
speaker-2m9pm	4/4	Running	0	9m19s
speaker-7m4qw	3/4	Running	0	19s
speaker-szlmx	4/4	Running	0	9m19s



#### NOTE

Speaker and controller pods are recreated to ensure the updated logging level is applied. The logging level is modified for all the components of MetallLB.

4. View the **speaker** logs:

```
$ oc logs -n metallb-system speaker-7m4qw -c speaker
```

#### Example output

```
{"branch":"main","caller":"main.go:92","commit":"3d052535","goversion":"gc / go1.17.1 / amd64","level":"info","msg":"MetallLB speaker starting (commit 3d052535, branch main)","ts":"2022-05-17T09:55:05Z","version":""}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"ens4","level":"info","msg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
 {"caller":"announcer.go:119","event":"createNDPResponder","interface":"ens4","level":"info","msg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
 {"caller":"announcer.go:110","event":"createARPResponder","interface":"tun0","level":"info","msg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
 {"caller":"announcer.go:119","event":"createNDPResponder","interface":"tun0","level":"info","msg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
I0517 09:55:06.515686    95 request.go:665] Waited for 1.026500832s due to client-side throttling, not priority and fairness, request:
GET:https://172.30.0.1:443/apis/operators.coreos.com/v1alpha1?timeout=32s
 {"Starting Manager":"(MISSING)","caller":"k8s.go:389","level":"info","ts":"2022-05-17T09:55:08Z"}
 {"caller":"speakerlist.go:310","level":"info","msg":"node event - forcing sync","node
```

```

addr":"10.0.128.4","node event":"NodeJoin","node name":"ci-ln-qb8t3mb-72292-7s7rh-
worker-a-vvznj","ts":"2022-05-17T09:55:08Z"}
{"caller":"service_controller.go:113","controller":"ServiceReconciler","enqueueing":"openshift-
kube-controller-manager-operator/metrics","epslice": "{\"metadata\":{\"name\":\"metrics-
xtsxr\"},\"generateName\":\"metrics-\",\"namespace\":\"openshift-kube-controller-manager-
operator\"},\"uid\":\"ac6766d7-8504-492c-9d1e-
4ae8897990ad\"},\"resourceVersion\":\"9041\"},\"generation\":4,\"creationTimestamp\":\"2022-
05-17T07:16:53Z\"},\"labels\":{\"app\":\"kube-controller-manager-
operator\",\"endpointslice.kubernetes.io/managed-by\":\"endpointslice-
controller.k8s.io\",\"kubernetes.io/service-name\":\"metrics\"},\"annotations\":
{\"endpoints.kubernetes.io/last-change-trigger-time\":\"2022-05-
17T07:21:34Z\"},\"ownerReferences\":
[{\\"apiVersion\":\"v1\",\"kind\":\"Service\",\"name\":\"metrics\",\"uid\":\"0518eed3-6152-42be-
b566-0bd00a60faf8\",\"controller\":true,\"blockOwnerDeletion\":true}],\"managedFields\":
[{\\"manager\":\"kube-controller-
manager\",\"operation\":\"Update\",\"apiVersion\":\"discovery.k8s.io/v1\",\"time\":\"2022-05-
17T07:20:02Z\"},\"fieldsType\":\"FieldsV1\",\"fieldsV1\":{\"f:addressType\":{},\"f:endpoints\":
{}},\"f:metadata\":{\"f:annotations\":{},\"f:labels\":{},\"f:app\":
{}},\"f:endpointslice.kubernetes.io/managed-by\":{},\"f:kubernetes.io/service-name\":
{}},\"f:ownerReferences\":{},\"k:{\\\"uid\\\":\\\"0518eed3-6152-42be-b566-
0bd00a60faf8\\\"}\":{}},\"f:ports\":{}},\"addressType\":\"IPv4\",\"endpoints\":[{\\"addresses\":
[\"10.129.0.7\"],\"conditions\":{\"ready\":true,\"serving\":true,\"terminating\":false},\"targetRef\":
{\\"kind\":\"Pod\",\"name\":\"kube-controller-manager-operator-6b98b89ddd-
8d4nf\"},\"uid\":\"dd5139b8-e41c-4946-a31b-
1a629314e844\"},\"resourceVersion\":\"9038\"},\"nodeName\":\"ci-ln-qb8t3mb-72292-7s7rh-
master-0\"},\"zone\":\"us-central1-a\"]},\"ports\":
[{\\"name\":\"https\",\"protocol\":\"TCP\",\"port\":8443}]}},\"level\":\"debug\",\"ts\":\"2022-05-
17T09:55:08Z\"}

```

## 5. View the FRR logs:

```
$ oc logs -n metallb-system speaker-7m4qw -c frr
```

### Example output

```

Started watchfrr
2022/05/17 09:55:05 ZEBRA: client 16 says hello and bids fair to announce only bgp routes
vrf=0
2022/05/17 09:55:05 ZEBRA: client 31 says hello and bids fair to announce only vnc routes
vrf=0
2022/05/17 09:55:05 ZEBRA: client 38 says hello and bids fair to announce only static routes
vrf=0
2022/05/17 09:55:05 ZEBRA: client 43 says hello and bids fair to announce only bfd routes
vrf=0
2022/05/17 09:57:25.089 BGP: Creating Default VRF, AS 64500
2022/05/17 09:57:25.090 BGP: dup addr detect enable max_moves 5 time 180 freeze
 disable freeze_time 0
2022/05/17 09:57:25.090 BGP: bgp_get: Registering BGP instance (null) to zebra
2022/05/17 09:57:25.090 BGP: Registering VRF 0
2022/05/17 09:57:25.091 BGP: Rx Router Id update VRF 0 Id 10.131.0.1/32
2022/05/17 09:57:25.091 BGP: RID change : vrf VRF default(0), RTR ID 10.131.0.1
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF br0
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ens4

```

2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr 10.0.128.4/32  
 2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr fe80::c9d:84da:4d86:5618/64  
 2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF lo  
 2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ovs-system  
 2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF tun0  
 2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr 10.131.0.1/23  
 2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr fe80::40f1:d1ff:feb6:5322/64  
 2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2da49fed  
 2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2da49fed addr fe80::24bd:d1ff:fec1:d88/64  
 2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2fa08c8c  
 2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2fa08c8c addr fe80::6870:ff:fe96:efc8/64  
 2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth41e356b7  
 2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth41e356b7 addr fe80::48ff:37ff:fede:eb4b/64  
 2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth1295c6e2  
 2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth1295c6e2 addr fe80::b827:a2ff:feed:637/64  
 2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth9733c6dc  
 2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth9733c6dc addr fe80::3cf4:15ff:fe11:e541/64  
 2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth336680ea  
 2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth336680ea addr fe80::94b1:8bff:fe7e:488c/64  
 2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vetha0a907b7  
 2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vetha0a907b7 addr fe80::3855:a6ff:fe73:46c3/64  
 2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf35a4398  
 2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf35a4398 addr fe80::40ef:2fff:fe57:4c4d/64  
 2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf831b7f4  
 2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf831b7f4 addr fe80::f0d9:89ff:fe7c:1d32/64  
 2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vxlan\_sys\_4789  
 2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vxlan\_sys\_4789 addr fe80::80c1:82ff:fe4b:f078/64  
 2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Timer (start timer expire).  
 2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] BGP\_Start (Idle->Connect), fd -1  
 2022/05/17 09:57:26.094 BGP: Allocated bnc 10.0.0.1/32(0)(VRF default) peer 0x7f807f7631a0  
 2022/05/17 09:57:26.094 BGP: sendmsg\_zebra\_rnh: sending cmd ZEBRA\_NEXTHOP\_REGISTER for 10.0.0.1/32 (vrf VRF default)  
 2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Waiting for NHT  
 2022/05/17 09:57:26.094 BGP: bgp\_fsm\_change\_status : vrf default(0), Status: Connect established\_peers 0  
 2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Idle to Connect  
 2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] TCP\_connection\_open\_failed (Connect->Active), fd -1  
 2022/05/17 09:57:26.094 BGP: bgp\_fsm\_change\_status : vrf default(0), Status: Active established\_peers 0  
 2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Connect to Active  
 2022/05/17 09:57:26.094 ZEBRA: rnh\_register msg from client bgp: hdr->length=8, type=nexthop vrf=0

```

2022/05/17 09:57:26.094 ZEBRA: 0: Add RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: Evaluate RNH, type Nexthop (force)
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: NH has become unresolved
2022/05/17 09:57:26.094 ZEBRA: 0: Client bgp registers for RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 BGP: VRF default(0): Rcvd NH update 10.0.0.1/32(0) - metric 0/0
#nhops 0/0 flags 0x6
2022/05/17 09:57:26.094 BGP: NH update for 10.0.0.1/32(0)(VRF default) - flags 0x6
chgflags 0x0 - evaluate paths
2022/05/17 09:57:26.094 BGP: evaluate_paths: Updating peer (10.0.0.1(VRF default)) status
with NHT
2022/05/17 09:57:30.081 ZEBRA: Event driven route-map update triggered
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-out
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-in
2022/05/17 09:57:31.104 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type
RTM_NEWNEIGH(28), len=76, seq=0, pid=0
2022/05/17 09:57:31.104 ZEBRA: Neighbor Entry received is not on a VLAN or a BRIDGE,
ignoring
2022/05/17 09:57:31.105 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type
RTM_NEWNEIGH(28), len=76, seq=0, pid=0
2022/05/17 09:57:31.105 ZEBRA: Neighbor Entry received is not on a VLAN or a BRIDGE,
ignoring

```

### 25.9.1.1. FRRouting (FRR) log levels

The following table describes the FRR logging levels.

**Table 25.10. Log levels**

Log level	Description
<b>all</b>	Supplies all logging information for all logging levels.
<b>debug</b>	Information that is diagnostically helpful to people. Set to <b>debug</b> to give detailed troubleshooting information.
<b>info</b>	Provides information that always should be logged but under normal circumstances does not require user intervention. This is the default logging level.
<b>warn</b>	Anything that can potentially cause inconsistent <b>MetallB</b> behaviour. Usually <b>MetallB</b> automatically recovers from this type of error.
<b>error</b>	Any error that is fatal to the functioning of <b>MetallB</b> . These errors usually require administrator intervention to fix.
<b>none</b>	Turn off all logging.

### 25.9.2. Troubleshooting BGP issues

As a cluster administrator, if you need to troubleshoot BGP configuration issues, you need to run commands in the FRR container.

## Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

## Procedure

1. Display the names of the **frr-k8s** pods by running the following command:

```
$ oc -n metallb-system get pods -l component=frr-k8s
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE
frr-k8s-thsmw	6/6	Running	0	109m

2. Display the running configuration for FRR by running the following command:

```
$ oc exec -n metallb-system frr-k8s-thsmw -c frr -- vtysh -c "show running-config"
```

### Example output

Building configuration...

Current configuration:

```
!
frr version 8.5.3
frr defaults traditional
hostname some-hostname
log file /etc/frr/frr.log informational
log timestamp precision 3
no ip forwarding
no ipv6 forwarding
service integrated-vtysh-config
!
router bgp 64500 1
bgp router-id 10.0.1.2
no bgp ebgp-requires-policy
no bgp default ipv4-unicast
no bgp network import-check
neighbor 10.0.2.3 remote-as 64500 2
neighbor 10.0.2.3 bfd profile doc-example-bfd-profile-full 3
neighbor 10.0.2.3 timers 5 15
neighbor 10.0.2.4 remote-as 64500
neighbor 10.0.2.4 bfd profile doc-example-bfd-profile-full
neighbor 10.0.2.4 timers 5 15
!
address-family ipv4 unicast
network 203.0.113.200/30 4
neighbor 10.0.2.3 activate
neighbor 10.0.2.3 route-map 10.0.2.3-in in
neighbor 10.0.2.4 activate
neighbor 10.0.2.4 route-map 10.0.2.4-in in
```

```

exit-address-family
!
address-family ipv6 unicast
  network fc00:f853:ccd:e799::/124
  neighbor 10.0.2.3 activate
  neighbor 10.0.2.3 route-map 10.0.2.3-in in
  neighbor 10.0.2.4 activate
  neighbor 10.0.2.4 route-map 10.0.2.4-in in
exit-address-family
!
route-map 10.0.2.3-in deny 20
!
route-map 10.0.2.4-in deny 20
!
ip nht resolve-via-default
!
ipv6 nht resolve-via-default
!
line vty
!
bfd
  profile doc-example-bfd-profile-full
  transmit-interval 35
  receive-interval 35
  passive-mode
  echo-mode
  echo-interval 35
  minimum-ttl 10
!
!
end

```

- 1 The **router bgp** section indicates the ASN for MetalLB.
  - 2 Confirm that a **neighbor <ip-address> remote-as <peer-ASN>** line exists for each BGP peer custom resource that you added.
  - 3 If you configured BFD, confirm that the BFD profile is associated with the correct BGP peer and that the BFD profile appears in the command output.
  - 4 Confirm that the **network <ip-address-range>** lines match the IP address ranges that you specified in address pool custom resources that you added.
3. Display the BGP summary by running the following command:

```
$ oc exec -n metallb-system frr-k8s-thsmw -c frr -- vtysh -c "show bgp summary"
```

### Example output

```

IPv4 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

```

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
PfxSnt									
10.0.2.3	4	64500	387	389	0	0	0	00:32:02	0 1 1
10.0.2.4	4	64500	0	0	0	0	0	never	Active 0 2

Total number of neighbors 2

#### IPv6 Unicast Summary:

BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0

BGP table version 1

RIB entries 1, using 192 bytes of memory

Peers 2, using 29 KiB of memory

Neighbor	V	AS	MsgRcvd	MsgSent	TblVer	InQ	OutQ	Up/Down	State/PfxRcd
PfxSnt									
10.0.2.3	4	64500	387	389	0	0	0	00:32:02	NoNeg
10.0.2.4	4	64500	0	0	0	0	0	never	Active 0

Total number of neighbors 2

- 1 Confirm that the output includes a line for each BGP peer custom resource that you added.
- 2 Output that shows **0** messages received and messages sent indicates a BGP peer that does not have a BGP session. Check network connectivity and the BGP configuration of the BGP peer.

#### 4. Display the BGP peers that received an address pool by running the following command:

```
$ oc exec -n metallb-system frr-k8s-thsmw -c frr -- vtysh -c "show bgp ipv4 unicast 203.0.113.200/30"
```

Replace **ipv4** with **ipv6** to display the BGP peers that received an IPv6 address pool. Replace **203.0.113.200/30** with an IPv4 or IPv6 IP address range from an address pool.

#### Example output

```
BGP routing table entry for 203.0.113.200/30
Paths: (1 available, best #1, table default)
  Advertised to non peer-group peers:
    10.0.2.3 1
    Local
      0.0.0.0 from 0.0.0.0 (10.0.1.2)
        Origin IGP, metric 0, weight 32768, valid, sourced, local, best (First path received)
        Last update: Mon Jan 10 19:49:07 2022
```

- 1 Confirm that the output includes an IP address for a BGP peer.

### 25.9.3. Troubleshooting BFD issues

The Bidirectional Forwarding Detection (BFD) implementation that Red Hat supports uses FRRouting (FRR) in a container in the **speaker** pods. The BFD implementation relies on BFD peers also being

configured as BGP peers with an established BGP session. As a cluster administrator, if you need to troubleshoot BFD configuration issues, you need to run commands in the FRR container.

## Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

## Procedure

1. Display the names of the **speaker** pods:

```
$ oc get -n metallb-system pods -l component=speaker
```

### Example output

NAME	READY	STATUS	RESTARTS	AGE
speaker-66bth	4/4	Running	0	26m
speaker-gvfnf	4/4	Running	0	26m
...				

2. Display the BFD peers:

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bfd peers brief"
```

### Example output

Session count: 2			
SessionId	LocalAddress	PeerAddress	Status
=====	=====	=====	=====
3909139637	10.0.1.2	10.0.2.3	up <.>

<.> Confirm that the **PeerAddress** column includes each BFD peer. If the output does not list a BFD peer IP address that you expected the output to include, troubleshoot BGP connectivity with the peer. If the status field indicates **down**, check for connectivity on the links and equipment between the node and the peer. You can determine the node name for the speaker pod with a command like **oc get pods -n metallb-system speaker-66bth -o jsonpath='{.spec.nodeName}'**.

### 25.9.4. MetalLB metrics for BGP and BFD

OpenShift Container Platform captures the following Prometheus metrics for MetalLB that relate to BGP peers and BFD profiles.

Table 25.11. MetalLB BFD metrics

Name	Description
<b>frrk8s_bfd_control_packets_input</b>	Counts the number of BFD control packets received from each BFD peer.

Name	Description
<b>frrk8s_bfd_control_packet_output</b>	Counts the number of BFD control packets sent to each BFD peer.
<b>frrk8s_bfd_echo_packet_input</b>	Counts the number of BFD echo packets received from each BFD peer.
<b>frrk8s_bfd_echo_packet_output</b>	Counts the number of BFD echo packets sent to each BFD.
<b>frrk8s_bfd_session_down_events</b>	Counts the number of times the BFD session with a peer entered the <b>down</b> state.
<b>frrk8s_bfd_session_up</b>	Indicates the connection state with a BFD peer. <b>1</b> indicates the session is <b>up</b> and <b>0</b> indicates the session is <b>down</b> .
<b>frrk8s_bfd_session_up_events</b>	Counts the number of times the BFD session with a peer entered the <b>up</b> state.
<b>frrk8s_bfd_zebra_notifications</b>	Counts the number of BFD Zebra notifications for each BFD peer.

Table 25.12. MetalLB BGP metrics

Name	Description
<b>frrk8s_bgp_announced_prefixes_total</b>	Counts the number of load balancer IP address prefixes that are advertised to BGP peers. The terms <i>prefix</i> and <i>aggregated route</i> have the same meaning.
<b>frrk8s_bgp_session_up</b>	Indicates the connection state with a BGP peer. <b>1</b> indicates the session is <b>up</b> and <b>0</b> indicates the session is <b>down</b> .
<b>frrk8s_bgp_updates_total</b>	Counts the number of BGP update messages sent to each BGP peer.
<b>frrk8s_bgpOpens_sent</b>	Counts the number of BGP open messages sent to each BGP peer.
<b>frrk8s_bgpOpens_received</b>	Counts the number of BGP open messages received from each BGP peer.
<b>frrk8s_bgp_notifications_sent</b>	Counts the number of BGP notification messages sent to each BGP peer.
<b>frrk8s_bgp_updates_total_received</b>	Counts the number of BGP update messages received from each BGP peer.

Name	Description
<b>frrk8s_bgp_keepalives_sent</b>	Counts the number of BGP keepalive messages sent to each BGP peer.
<b>frrk8s_bgp_keepalives_received</b>	Counts the number of BGP keepalive messages received from each BGP peer.
<b>frrk8s_bgp_route_refresh_sent</b>	Counts the number of BGP route refresh messages sent to each BGP peer.
<b>frrk8s_bgp_total_sent</b>	Counts the number of total BGP messages sent to each BGP peer.
<b>frrk8s_bgp_total_received</b>	Counts the number of total BGP messages received from each BGP peer.

## Additional resources

- See [Querying metrics for all projects with the monitoring dashboard](#) for information about using the monitoring dashboard.

### 25.9.5. About collecting MetalLB data

You can use the **oc adm must-gather** CLI command to collect information about your cluster, your MetalLB configuration, and the MetalLB Operator. The following features and objects are associated with MetalLB and the MetalLB Operator:

- The namespace and child objects that the MetalLB Operator is deployed in
- All MetalLB Operator custom resource definitions (CRDs)

The **oc adm must-gather** CLI command collects the following information from FRRouting (FRR) that Red Hat uses to implement BGP and BFD:

- **/etc/frr/frr.conf**
- **/etc/frr/frr.log**
- **/etc/frr/daemons** configuration file
- **/etc/frr/vtysh.conf**

The log and configuration files in the preceding list are collected from the **frr** container in each **speaker** pod.

In addition to the log and configuration files, the **oc adm must-gather** CLI command collects the output from the following **vtysh** commands:

- **show running-config**
- **show bgp ipv4**

- **show bgp ipv6**
- **show bgp neighbor**
- **show bfd peer**

No additional configuration is required when you run the **oc adm must-gather** CLI command.

#### Additional resources

- [Gathering data about your cluster](#)

# CHAPTER 26. ASSOCIATING SECONDARY INTERFACES METRICS TO NETWORK ATTACHMENTS

## 26.1. EXTENDING SECONDARY NETWORK METRICS FOR MONITORING

Secondary devices, or interfaces, are used for different purposes. It is important to have a way to classify them to be able to aggregate the metrics for secondary devices with the same classification.

Exposed metrics contain the interface but do not specify where the interface originates. This is workable when there are no additional interfaces. However, if secondary interfaces are added, it can be difficult to use the metrics since it is hard to identify interfaces using only interface names.

When adding secondary interfaces, their names depend on the order in which they are added, and different secondary interfaces might belong to different networks and can be used for different purposes.

With **pod\_network\_name\_info** it is possible to extend the current metrics with additional information that identifies the interface type. In this way, it is possible to aggregate the metrics and to add specific alarms to specific interface types.

The network type is generated using the name of the related **NetworkAttachmentDefinition**, that in turn is used to differentiate different classes of secondary networks. For example, different interfaces belonging to different networks or using different CNIIs use different network attachment definition names.

### 26.1.1. Network Metrics Daemon

The Network Metrics Daemon is a daemon component that collects and publishes network related metrics.

The kubelet is already publishing network related metrics you can observe. These metrics are:

- **container\_network\_receive\_bytes\_total**
- **container\_network\_receive\_errors\_total**
- **container\_network\_receive\_packets\_total**
- **container\_network\_receive\_packets\_dropped\_total**
- **container\_network\_transmit\_bytes\_total**
- **container\_network\_transmit\_errors\_total**
- **container\_network\_transmit\_packets\_total**
- **container\_network\_transmit\_packets\_dropped\_total**

The labels in these metrics contain, among others:

- Pod name
- Pod namespace

- Interface name (such as **eth0**)

These metrics work well until new interfaces are added to the pod, for example via [Multus](#), as it is not clear what the interface names refer to.

The interface label refers to the interface name, but it is not clear what that interface is meant for. In case of many different interfaces, it would be impossible to understand what network the metrics you are monitoring refer to.

This is addressed by introducing the new **pod\_network\_name\_info** described in the following section.

### 26.1.2. Metrics with network name

This daemonset publishes a **pod\_network\_name\_info** gauge metric, with a fixed value of **0**:

```
pod_network_name_info{interface="net0",namespace="namespacename",network_name="hadname
space/firstNAD",pod="podname"} 0
```

The network name label is produced using the annotation added by Multus. It is the concatenation of the namespace the network attachment definition belongs to, plus the name of the network attachment definition.

The new metric alone does not provide much value, but combined with the network related **container\_network\_\*** metrics, it offers better support for monitoring secondary networks.

Using a **promql** query like the following ones, it is possible to get a new metric containing the value and the network name retrieved from the **k8s.v1.cni.cncf.io/network-status** annotation:

```
(container_network_receive_bytes_total) + on(namespace,pod,interface) group_left(network_name) (
pod_network_name_info )
(container_network_receive_errors_total) + on(namespace,pod,interface) group_left(network_name) (
pod_network_name_info )
(container_network_receive_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_receive_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_bytes_total) + on(namespace,pod,interface) group_left(network_name)
( pod_network_name_info )
(container_network_transmit_errors_total) + on(namespace,pod,interface) group_left(network_name)
( pod_network_name_info )
(container_network_transmit_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name)
```