



# OpenShift Container Platform 4.11

## 네트워킹

클러스터 네트워킹 구성 및 관리





## 법적 공지

Copyright © 2023 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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.

## 초록

이 문서는 DNS, 인그레스 및 Pod 네트워크를 포함하여 OpenShift Container Platform 클러스터 네트워크를 구성 및 관리하기 위한 지침을 제공합니다.

## 차례

<b>1장. 네트워킹 이해</b> .....	<b>7</b>
1.1. OPENSIFT CONTAINER PLATFORM DNS	7
1.2. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	7
1.3. OPENSIFT CONTAINER PLATFORM 네트워킹에 대한 일반 용어집	8
<b>2장. 호스트에 액세스</b> .....	<b>11</b>
2.1. 설치 관리자 프로비저닝 인프라 클러스터에서 AMAZON WEB SERVICES의 호스트에 액세스	11
<b>3장. 네트워킹 OPERATOR 개요</b> .....	<b>12</b>
3.1. CNO(CLUSTER NETWORK OPERATOR)	12
3.2. DNS OPERATOR	12
3.3. INGRESS OPERATOR	12
3.4. 외부 DNS OPERATOR	12
<b>4장. OPENSIFT 컨테이너 플랫폼의 CLUSTER NETWORK OPERATOR</b> .....	<b>13</b>
4.1. CNO(CLUSTER NETWORK OPERATOR)	13
4.2. 클러스터 네트워크 구성 보기	13
4.3. CNO(CLUSTER NETWORK OPERATOR) 상태 보기	14
4.4. CNO(CLUSTER NETWORK OPERATOR) 로그 보기	14
4.5. CNO(CLUSTER NETWORK OPERATOR) 구성	14
4.6. 추가 리소스	20
<b>5장. OPENSIFT CONTAINER PLATFORM에서의 DNS OPERATOR</b> .....	<b>21</b>
5.1. DNS OPERATOR	21
5.2. DNS OPERATOR MANAGEMENTSTATE 변경	21
5.3. DNS POD 배치 제어	22
5.4. 기본 DNS보기	23
5.5. DNS 전달 사용	23
5.6. DNS OPERATOR 상태	27
5.7. DNS OPERATOR 로그	27
5.8. COREDNS 로그 수준 설정	28
5.9. COREDNS OPERATOR 로그 수준 설정	28
<b>6장. OPENSIFT CONTAINER PLATFORM에서의 INGRESS OPERATOR</b> .....	<b>30</b>
6.1. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	30
6.2. INGRESS 구성 자산	30
6.3. INGRESS 컨트롤러 구성 매개변수	30
6.4. 기본 INGRESS 컨트롤러 보기	43
6.5. INGRESS OPERATOR 상태 보기	44
6.6. INGRESS 컨트롤러 로그 보기	44
6.7. INGRESS 컨트롤러 상태 보기	44
6.8. INGRESS 컨트롤러 구성	44
6.9. 추가 리소스	65
<b>7장. OPENSIFT CONTAINER PLATFORM의 INGRESS 분할</b> .....	<b>66</b>
7.1. INGRESS 컨트롤러 분할	66
7.2. INGRESS 컨트롤러 분할을 위한 경로 생성	73
<b>8장. INGRESS 컨트롤러 끝점 게시 전략 구성</b> .....	<b>78</b>
8.1. INGRESS 컨트롤러 끝점 게시 전략	78
8.2. 추가 리소스	81
<b>9장. 끝점에 대한 연결 확인</b> .....	<b>82</b>

9.1. 연결 상태 점검 수행	82
9.2. 연결 상태 점검 구현	82
9.3. PODNETWORKCONNECTIVITYCHECK 오브젝트 필드	83
9.4. 끝점에 대한 네트워크 연결 확인	86
<b>10장. 클러스터 네트워크의 MTU 변경</b>	<b>91</b>
10.1. 클러스터 MTU 정보	91
10.2. 클러스터 MTU 변경	93
10.3. 추가 리소스	106
<b>11장. 노드 포트 서비스 범위 구성</b>	<b>108</b>
11.1. 사전 요구 사항	108
11.2. 노드 포트 범위 확장	108
11.3. 추가 리소스	109
<b>12장. IP 페일오버 구성</b>	<b>111</b>
12.1. IP 페일오버 환경 변수	112
12.2. IP 페일오버 구성	113
12.3. 가상 IP 주소 정보	118
12.4. 검사 구성 및 스크립트 알람	118
12.5. VRRP 선점 구성	121
12.6. VRRP ID 오프셋 정보	122
12.7. 254개 이상의 주소에 대한 IP 페일오버 구성	123
12.8. INGRESSIP의 고가용성	124
12.9. IP 페일오버 제거	125
<b>13장. 인터페이스 수준 네트워크 SYSCTL 구성</b>	<b>128</b>
13.1. 튜닝 CNI 구성	128
13.2. 추가 리소스	132
<b>14장. 베어 메탈 클러스터에서 SCTP(STREAM CONTROL TRANSMISSION PROTOCOL) 사용</b>	<b>133</b>
14.1. OPENSIFT CONTAINER PLATFORM에서의 SCTP(스트림 제어 전송 프로토콜)	133
14.2. SCTP(스트림 제어 전송 프로토콜) 활성화	134
14.3. SCTP(STREAM CONTROL TRANSMISSION PROTOCOL)의 활성화 여부 확인	135
<b>15장. PTP 하드웨어 사용</b>	<b>139</b>
15.1. PTP 하드웨어 정보	139
15.2. PTP 정보	139
15.3. CLI를 사용하여 PTP OPERATOR 설치	141
15.4. 웹 콘솔을 사용하여 PTP OPERATOR 설치	143
15.5. PTP 장치 구성	144
15.6. 일반적인 PTP OPERATOR 문제 해결	163
15.7. PTP 하드웨어 빠른 이벤트 알람 프레임워크	167
<b>16장. 외부 DNS OPERATOR</b>	<b>186</b>
16.1. OPENSIFT CONTAINER PLATFORM의 외부 DNS OPERATOR	186
16.2. 클라우드 공급자에 외부 DNS OPERATOR 설치	188
16.3. 외부 DNS OPERATOR 구성 매개변수	190
16.4. AWS에서 DNS 레코드 생성	192
16.5. AZURE에서 DNS 레코드 생성	195
16.6. GCP에서 DNS 레코드 생성	198
16.7. INFOBLOX에서 DNS 레코드 생성	201
16.8. 외부 DNS OPERATOR에서 클러스터 전체 프록시 구성	203
<b>17장. 네트워크 정책</b>	<b>205</b>

17.1. 네트워크 정책 정의	205
17.2. 네트워크 정책 이벤트 로깅	209
17.3. 네트워크 정책 생성	220
17.4. 네트워크 정책 보기	223
17.5. 네트워크 정책 편집	226
17.6. 네트워크 정책 삭제	229
17.7. 프로젝트의 기본 네트워크 정책 정의	231
17.8. 네트워크 정책으로 다중 테넌트 격리 구성	235
<b>18장. AWS LOAD BALANCER OPERATOR</b>	<b>239</b>
18.1. OPENSIFT CONTAINER PLATFORM의 AWS LOAD BALANCER OPERATOR	239
18.2. AWS LOAD BALANCER OPERATOR 이해	240
18.3. AWS 로드 밸런서 컨트롤러 인스턴스 생성	242
18.4. 여러 INGRESS 생성	246
18.5. TLS 종료 추가	250
<b>19장. 다중 네트워크</b>	<b>253</b>
19.1. 다중 네트워크 이해하기	253
19.2. 추가 네트워크 구성	254
19.3. 가상 라우팅 및 전달 정보	269
19.4. 다중 네트워크 정책 구성	269
19.5. 추가 네트워크에 POD 연결	279
19.6. 추가 네트워크에서 POD 제거	287
19.7. 추가 네트워크 편집	287
19.8. 추가 네트워크 제거	289
19.9. VRF에 보조 네트워크 할당	290
<b>20장. 하드웨어 네트워크</b>	<b>295</b>
20.1. SR-IOV(SINGLE ROOT I/O VIRTUALIZATION) 하드웨어 네트워크 정보	295
20.2. SR-IOV NETWORK OPERATOR 설치	304
20.3. SR-IOV NETWORK OPERATOR 구성	308
20.4. SR-IOV 네트워크 장치 구성	315
20.5. SR-IOV 이더넷 네트워크 연결 구성	328
20.6. SR-IOV INFINIBAND 네트워크 연결 구성	336
20.7. SR-IOV 추가 네트워크에 POD 추가	343
20.8. SR-IOV 네트워크의 인터페이스 수준 네트워크 SYSCTL 설정 구성	352
20.9. 고성능 멀티 캐스트 사용	370
20.10. DPDK 및 RDMA 사용	372
20.11. POD 수준 본딩 사용	396
20.12. 하드웨어 오프로드 구성	401
20.13. SR-IOV NETWORK OPERATOR 설치 제거	408
<b>21장. OPENSIFT SDN 기본 CNI 네트워크 공급자</b>	<b>411</b>
21.1. OPENSIFT SDN 기본 CNI 네트워크 공급자 정보	411
21.2. 프로젝트의 송신 IP 구성	412
21.3. 프로젝트에 대한 송신 방화벽 구성	422
21.4. 프로젝트의 송신 방화벽 편집	430
21.5. 프로젝트의 송신 방화벽 편집	431
21.6. 프로젝트에서 송신 방화벽 제거	432
21.7. 송신 라우터 POD 사용에 대한 고려 사항	433
21.8. 리디렉션 모드에서 송신 라우터 POD 배포	436
21.9. HTTP 프록시 모드에서 송신 라우터 POD 배포	440
21.10. DNS 프록시 모드에서 송신 라우터 POD 배포	444
21.11. 구성 맵에서 송신 라우터 POD 대상 목록 구성	448

21.12. 프로젝트에 멀티 캐스트 사용	451
21.13. 프로젝트에 대한 멀티 캐스트 비활성화	454
21.14. OPENSIFT SDN을 사용하여 네트워크 격리 구성	454
21.15. KUBE-PROXY 설정	457
<b>22장. OVN-KUBERNETES 기본 CNI 네트워크 공급자</b>	<b>461</b>
22.1. OVN-KUBERNETES 기본 CNI(CONTAINER NETWORK INTERFACE) 네트워크 공급자 정보	461
22.2. OPENSIFT SDN 클러스터 네트워크 공급자에서 마이그레이션	464
22.3. OPENSIFT SDN 네트워크 공급자로 롤백	477
22.4. IPV4/IPV6 듀얼 스택 네트워킹으로 변환	484
22.5. IPSEC 암호화 구성	487
22.6. 프로젝트에 대한 송신 방화벽 구성	492
22.7. 프로젝트의 송신 방화벽 보기	500
22.8. 프로젝트의 송신 방화벽 편집	501
22.9. 프로젝트에서 송신 방화벽 제거	503
22.10. 송신 IP 주소 구성	503
22.11. 송신 IP 주소 할당	514
22.12. 송신 라우터 POD 사용에 대한 고려 사항	516
22.13. 리디렉션 모드에서 송신 라우터 POD 배포	519
22.14. 프로젝트에 멀티 캐스트 사용	526
22.15. 프로젝트에 대한 멀티 캐스트 비활성화	529
22.16. 네트워크 흐름 추적	530
22.17. 하이브리드 네트워킹 구성	535
<b>23장. 경로 구성</b>	<b>539</b>
23.1. 경로 구성	539
23.2. 보안 경로	574
<b>24장. 수신 클러스터 트래픽 구성</b>	<b>580</b>
24.1. 수신 클러스터 트래픽 구성 개요	580
24.2. 서비스의 EXTERNALIP 구성	581
24.3. INGRESS 컨트롤러를 사용한 수신 클러스터 트래픽 구성	590
24.4. 로드 밸런서를 사용하여 수신 클러스터 트래픽 구성	601
24.5. AWS에서 수신 클러스터 트래픽 구성	607
24.6. 서비스 외부 IP에 대한 수신 클러스터 트래픽 구성	614
24.7. NODEPORT를 사용하여 수신 클러스터 트래픽 구성	617
<b>25장. KUBERNETES NMSTATE</b>	<b>621</b>
25.1. KUBERNETES NMSTATE OPERATOR 정보	621
25.2. 노드 네트워크 상태 관찰	624
25.3. 노드 네트워크 구성 업데이트	627
25.4. 노드 네트워크 구성 문제 해결	646
<b>26장. 클러스터 전체 프록시 구성</b>	<b>652</b>
26.1. 사전 요구 사항	652
26.2. 클러스터 전체 프록시 사용	652
26.3. 클러스터 전체 프록시 제거	655
<b>27장. 사용자 정의 PKI 구성</b>	<b>657</b>
27.1. 설치 중 클러스터 단위 프록시 구성	657
27.2. 클러스터 전체 프록시 사용	659
27.3. OPERATOR를 사용한 인증서 주입	662
<b>28장. RHOSP의 로드 밸런싱</b>	<b>665</b>
28.1. KURYR SDN으로 OCTAVIA OVN 로드 밸런서 공급자 드라이버 사용	665



28.2. OCTAVIA를 사용하여 애플리케이션 트래픽의 클러스터 확장	667
28.3. RHOSP OCTAVIA를 사용하여 수신 트래픽 스케일링	670
28.4. 외부 로드 밸런서 구성	673
<b>29장. METALLB로 로드 밸런싱</b> .....	<b>677</b>
29.1. METALLB 및 METALLB OPERATOR 정보	677
29.2. METALLB OPERATOR 설치	687
29.3. METALLB OPERATOR 업그레이드	695
29.4. METALLB 주소 풀 구성	699
29.5. IP 주소 풀 광고 정보	703
29.6. METALLB BGP 피어 구성	713
29.7. 커뮤니티 별칭 구성	720
29.8. METALLB BFD 프로필 구성	723
29.9. METALLB를 사용하도록 서비스 구성	726
29.10. METALLB 로깅, 문제 해결 및 지원	732
<b>30장. 보조 인터페이스 지표와 네트워크 연결 연관 짓기</b> .....	<b>745</b>
30.1. 모니터링을 위한 보조 네트워크 메트릭 확장	745
<b>31장. 네트워크 관찰 기능</b> .....	<b>748</b>
31.1. NETWORK OBSERVABILITY OPERATOR 릴리스 노트	748
31.2. 네트워크 관찰 정보	748
31.3. NETWORK OBSERVABILITY OPERATOR 설치	750
31.4. OPENSIFT CONTAINER PLATFORM의 NETWORK OBSERVABILITY OPERATOR	758
31.5. NETWORK OBSERVABILITY OPERATOR 구성	761
31.6. 네트워크 트래픽 관찰	766
31.7. FLOWCOLLECTOR 구성 매개변수	771
31.8. 네트워크 관찰 문제 해결	802



# 1장. 네트워킹 이해

클러스터 관리자는 클러스터 내에서 외부 트래픽에 실행되는 애플리케이션을 노출하고 네트워크 연결을 보호하는 몇 가지 옵션이 있습니다.

- 노드 포트 또는 로드 밸런서와 같은 서비스 유형
- **Ingress** 및 **Route**와 같은 API 리소스

기본적으로 Kubernetes는 pod 내에서 실행되는 애플리케이션의 내부 IP 주소를 각 pod에 할당합니다. pod와 해당 컨테이너에 네트워크를 지정할 수 있지만 클러스터 외부의 클라이언트에는 네트워킹 액세스 권한이 없습니다. 애플리케이션을 외부 트래픽에 노출할 때 각 pod에 고유 IP 주소를 부여하면 포트 할당, 네트워킹, 이름 지정, 서비스 검색, 로드 밸런싱, 애플리케이션 구성 및 마이그레이션 등 다양한 업무를 할 때 pod를 물리적 호스트 또는 가상 머신처럼 취급할 수 있습니다.



## 참고

일부 클라우드 플랫폼은 IPv4 **169.254.0.0/16** CIDR 블록의 링크 로컬 IP 주소인 169.254.169.254 IP 주소에서 수신 대기하는 메타데이터 API를 제공합니다.

Pod 네트워크에서는 이 CIDR 블록에 접근할 수 없습니다. 이러한 IP 주소에 액세스해야 하는 pod의 경우 pod 사양의 **spec.hostNetwork** 필드를 **true**로 설정하여 호스트 네트워크 액세스 권한을 부여해야 합니다.

Pod의 호스트 네트워크 액세스를 허용하면 해당 pod에 기본 네트워크 인프라에 대한 액세스 권한이 부여됩니다.

## 1.1. OPENSIFT CONTAINER PLATFORM DNS

여러 Pod에 사용하기 위해 프론트엔드 및 백엔드 서비스와 같은 여러 서비스를 실행하는 경우 사용자 이름, 서비스 IP 등에 대한 환경 변수를 생성하여 프론트엔드 Pod가 백엔드 서비스와 통신하도록 할 수 있습니다. 서비스를 삭제하고 다시 생성하면 새 IP 주소를 서비스에 할당할 수 있으며 서비스 IP 환경 변수의 업데이트된 값을 가져오기 위해 프론트엔드 Pod를 다시 생성해야 합니다. 또한 백엔드 서비스를 생성한 후 프론트엔드 Pod를 생성해야 서비스 IP가 올바르게 생성되고 프론트엔드 Pod에 환경 변수로 제공할 수 있습니다.

이러한 이유로 서비스 DNS는 물론 서비스 IP/포트를 통해서도 서비스를 이용할 수 있도록 OpenShift Container Platform에 DNS를 내장했습니다.

## 1.2. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR

OpenShift Container Platform 클러스터를 생성할 때 클러스터에서 실행되는 Pod 및 서비스에는 각각 자체 IP 주소가 할당됩니다. IP 주소는 내부에서 실행되지만 외부 클라이언트가 액세스할 수 없는 다른 pod 및 서비스에 액세스할 수 있습니다. Ingress Operator는 **IngressController** API를 구현하며 OpenShift Container Platform 클러스터 서비스에 대한 외부 액세스를 활성화하는 구성 요소입니다.

Ingress Operator를 사용하면 라우팅을 처리하기 위해 하나 이상의 HAProxy 기반 **Ingress 컨트롤러**를 배포하고 관리하여 외부 클라이언트가 서비스에 액세스할 수 있습니다. Ingress Operator를 사용하여 OpenShift 컨테이너 플랫폼 **Route** 및 Kubernetes **Ingress** 리소스를 지정하면 수신 트래픽을 라우팅할 수 있습니다. **endpointPublishingStrategy** 유형 및 내부 로드 밸런싱을 정의하는 기능과 같은 Ingress 컨트롤러 내 구성은 Ingress 컨트롤러 끝점을 게시하는 방법을 제공합니다.

### 1.2.1. 경로와 Ingress 비교

OpenShift Container Platform의 Kubernetes Ingress 리소스는 클러스터 내에서 Pod로 실행되는 공유 라우터 서비스를 사용하여 Ingress 컨트롤러를 구현합니다. Ingress 트래픽을 관리하는 가장 일반적인 방법은 Ingress 컨트롤러를 사용하는 것입니다. 다른 일반 Pod와 마찬가지로 이 Pod를 확장하고 복제할 수 있습니다. 이 라우터 서비스는 오픈 소스 로드 밸런서 솔루션인 [HAProxy](#)를 기반으로 합니다.

OpenShift Container Platform 경로는 클러스터의 서비스에 대한 Ingress 트래픽을 제공합니다. 경로는 TLS 재암호화, TLS 패스스루, 블루-그린 배포를 위한 분할 트래픽 등 표준 Kubernetes Ingress 컨트롤러에서 지원하지 않는 고급 기능을 제공합니다.

Ingress 트래픽은 경로를 통해 클러스터의 서비스에 액세스합니다. 경로 및 Ingress는 Ingress 트래픽을 처리하는 데 필요한 주요 리소스입니다. Ingress는 외부 요청을 수락하고 경로를 기반으로 위임하는 것과 같은 경로와 유사한 기능을 제공합니다. 그러나 Ingress를 사용하면 HTTP/2, HTTPS, SNI(서버 이름 식별) 및 인증서가 있는 TLS와 같은 특정 유형의 연결만 허용할 수 있습니다. OpenShift Container Platform에서는 Ingress 리소스에서 지정하는 조건을 충족하기 위해 경로가 생성됩니다.

### 1.3. OPENSIFT CONTAINER PLATFORM 네트워킹에 대한 일반 용어집

이 용어집은 네트워킹 콘텐츠에 사용되는 일반적인 용어를 정의합니다.

#### 인증

OpenShift Container Platform 클러스터에 대한 액세스를 제어하기 위해 클러스터 관리자는 사용자 인증을 구성하고 승인된 사용자만 클러스터에 액세스할 수 있는지 확인할 수 있습니다. OpenShift Container Platform 클러스터와 상호 작용하려면 OpenShift Container Platform API에 인증해야 합니다. OpenShift Container Platform API에 대한 요청에 OAuth 액세스 토큰 또는 X.509 클라이언트 인증서를 제공하여 인증할 수 있습니다.

#### AWS Load Balancer Operator

ALB(AWS Load Balancer) Operator는 **aws-load-balancer-controller**의 인스턴스를 배포 및 관리합니다.

#### CNO(Cluster Network Operator)

CNO(Cluster Network Operator)는 OpenShift Container Platform 클러스터에서 클러스터 네트워크 구성 요소를 배포하고 관리합니다. 여기에는 설치 중에 클러스터에 선택된 CNI(Container Network Interface) 기본 네트워크 공급자 플러그인의 배포가 포함됩니다.

#### 구성 맵

구성 맵에서는 구성 데이터를 Pod에 삽입하는 방법을 제공합니다. 구성 맵에 저장된 데이터를 **ConfigMap** 유형의 볼륨에서 참조할 수 있습니다. Pod에서 실행되는 애플리케이션에서는 이 데이터를 사용할 수 있습니다.

#### CR(사용자 정의 리소스)

CR은 Kubernetes API의 확장입니다. 사용자 정의 리소스를 생성할 수 있습니다.

#### DNS

클러스터 DNS는 Kubernetes 서비스에 대한 DNS 레코드를 제공하는 DNS 서버입니다. Kubernetes에서 시작하는 컨테이너는 DNS 검색에 이 DNS 서버를 자동으로 포함합니다.

#### DNS Operator

DNS Operator는 CoreDNS를 배포하고 관리하여 Pod에 이름 확인 서비스를 제공합니다. 이를 통해 OpenShift Container Platform에서 DNS 기반 Kubernetes 서비스 검색이 가능합니다.

#### Deployment

애플리케이션의 라이프사이클을 유지 관리하는 Kubernetes 리소스 오브젝트입니다.

#### domain

domain은 Ingress 컨트롤러에서 제공하는 DNS 이름입니다.

#### egress

Pod에서 네트워크의 아웃바운드 트래픽을 통해 외부적으로 공유하는 데이터 프로세스입니다.

### 외부 DNS Operator

외부 DNS Operator는 ExternalDNS를 배포하고 관리하여 외부 DNS 공급자에서 OpenShift Container Platform으로 서비스 및 경로에 대한 이름 확인을 제공합니다.

### HTTP 기반 경로

HTTP 기반 경로는 기본 HTTP 라우팅 프로토콜을 사용하고 비보안 애플리케이션 포트에 서비스를 노출하는 비보안 경로입니다.

### Ingress

OpenShift Container Platform의 Kubernetes Ingress 리소스는 클러스터 내에서 Pod로 실행되는 공유 라우터 서비스를 사용하여 Ingress 컨트롤러를 구현합니다.

### Ingress 컨트롤러

Ingress Operator는 Ingress 컨트롤러를 관리합니다. OpenShift Container Platform 클러스터에 대한 외부 액세스를 허용하는 가장 일반적인 방법은 Ingress 컨트롤러를 사용하는 것입니다.

### 설치 프로그램에서 제공하는 인프라

설치 프로그램은 클러스터가 실행되는 인프라를 배포하고 구성합니다.

### kubelet

Pod에서 컨테이너가 실행 중인지 확인하기 위해 클러스터의 각 노드에서 실행되는 기본 노드 에이전트입니다.

### Kubernetes NMState Operator

Kubernetes NMState Operator는 OpenShift Container Platform 클러스터 노드에서 NMState를 사용하여 상태 중심 네트워크 구성을 수행하는 데 필요한 Kubernetes API를 제공합니다.

### kube-proxy

kube-proxy는 각 노드에서 실행되며 외부 호스트에서 서비스를 사용할 수 있도록 지원하는 프록시 서비스입니다. 컨테이너를 수정하도록 요청을 전달하는 데 도움이 되며 기본 로드 밸런싱을 수행할 수 있습니다.

### 로드 밸런서

OpenShift Container Platform은 로드 밸런서를 사용하여 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신합니다.

### MetalLB Operator

클러스터 관리자는 **LoadBalancer** 유형의 서비스가 클러스터에 추가되면 MetalLB가 서비스에 대한 외부 IP 주소를 추가하도록 클러스터에 MetalLB Operator를 추가할 수 있습니다.

### 멀티 캐스트

IP 멀티 캐스트를 사용하면 데이터가 여러 IP 주소로 동시에 브로드캐스트됩니다.

### 네임스페이스

네임스페이스는 모든 프로세스에 표시되는 특정 시스템 리소스를 격리합니다. 네임스페이스 내에서 해당 네임스페이스의 멤버인 프로세스만 해당 리소스를 볼 수 있습니다.

### networking

OpenShift Container Platform 클러스터의 네트워크 정보입니다.

### node

OpenShift Container Platform 클러스터의 작업자 시스템입니다. 노드는 VM(가상 머신) 또는 물리적 머신입니다.

### OpenShift Container Platform Ingress Operator

Ingress Operator는 **IngressController** API를 구현하며 OpenShift Container Platform 서비스에 대한 외부 액세스를 가능하게 하는 구성 요소입니다.

### Pod

OpenShift Container Platform 클러스터에서 실행되는 볼륨 및 IP 주소와 같은 공유 리소스가 있는 하나 이상의 컨테이너입니다. Pod는 정의, 배포 및 관리되는 최소 컴퓨팅 단위입니다.

### PTP Operator

PTP Operator는 **linuxptp** 서비스를 생성하고 관리합니다.

### Route

OpenShift Container Platform 경로는 클러스터의 서비스에 대한 Ingress 트래픽을 제공합니다. 경로는 TLS 재암호화, TLS 패스스루, 블루-그린 배포를 위한 분할 트래픽 등 표준 Kubernetes Ingress 컨트롤러에서 지원하지 않는 고급 기능을 제공합니다.

### 스케일링

리소스 용량을 늘리거나 줄입니다.

### service

Pod 세트에 실행 중인 애플리케이션을 노출합니다.

### SR-IOV(Single Root I/O Virtualization) Network Operator

SR-IOV(Single Root I/O Virtualization) Network Operator는 클러스터의 SR-IOV 네트워크 장치 및 네트워크 첨부 파일을 관리합니다.

### 소프트웨어 정의 네트워킹(SDN)

OpenShift Container Platform에서는 소프트웨어 정의 네트워킹(SDN) 접근법을 사용하여 OpenShift Container Platform 클러스터 전체의 pod 간 통신이 가능한 통합 클러스터 네트워크를 제공합니다.

### SCTP(스트림 제어 전송 프로토콜)

SCTP는 IP 네트워크에서 실행되는 안정적인 메시지 기반 프로토콜입니다.

### taint

테인트 및 톨러레이션은 Pod가 적절한 노드에 예약되도록 합니다. 노드에 하나 이상의 테인트를 적용할 수 있습니다.

### 허용 오차

Pod에 허용 오차를 적용할 수 있습니다. 허용 오차를 사용하면 스케줄러에서 일치하는 테인트를 사용하여 Pod를 예약할 수 있습니다.

### 웹 콘솔

OpenShift Container Platform을 관리할 UI(사용자 인터페이스)입니다.

## 2장. 호스트에 액세스

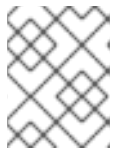
베스천 호스트(Bastion Host)를 생성하여 OpenShift Container Platform 인스턴스에 액세스하고 SSH(Secure Shell) 액세스 권한으로 컨트롤 플레인 노드에 액세스하는 방법을 알아봅니다.

### 2.1. 설치 관리자 프로비저닝 인프라 클러스터에서 AMAZON WEB SERVICES의 호스트에 액세스

OpenShift Container Platform 설치 관리자는 OpenShift Container Platform 클러스터에 프로비저닝된 Amazon EC2(Amazon Elastic Compute Cloud) 인스턴스에 대한 퍼블릭 IP 주소를 생성하지 않습니다. OpenShift Container Platform 호스트에 SSH를 사용하려면 다음 절차를 따라야 합니다.

#### 프로세스

1. **openshift-install** 명령으로 생성된 가상 프라이빗 클라우드(VPC)에 SSH로 액세스할 수 있는 보안 그룹을 만듭니다.
2. 설치 관리자가 생성한 퍼블릭 서브넷 중 하나에 Amazon EC2 인스턴스를 생성합니다.
3. 생성한 Amazon EC2 인스턴스와 퍼블릭 IP 주소를 연결합니다.  
OpenShift Container Platform 설치와는 달리, 생성한 Amazon EC2 인스턴스를 SSH 키 쌍과 연결해야 합니다. 이 인스턴스에서 사용되는 운영 체제는 중요하지 않습니다. 그저 인터넷을 OpenShift Container Platform 클러스터의 VPC에 연결하는 SSH 베스천의 역할을 수행하기 때문입니다. 사용하는 AMI(Amazon 머신 이미지)는 중요합니다. 예를 들어, RHCOS(Red Hat Enterprise Linux CoreOS)를 사용하면 설치 프로그램과 마찬가지로 Ignition을 통해 키를 제공할 수 있습니다.
4. Amazon EC2 인스턴스를 프로비저닝한 후 SSH로 연결할 수 있는 경우 OpenShift Container Platform 설치와 연결된 SSH 키를 추가해야 합니다. 이 키는 베스천 인스턴스의 키와 다를 수 있지만 반드시 달라야 하는 것은 아닙니다.



#### 참고

SSH 직접 액세스는 재해 복구 시에만 권장됩니다. Kubernetes API가 응답할 때는 권한 있는 Pod를 대신 실행합니다.

5. **oc get nodes**를 실행하고 출력을 확인한 후 마스터 노드 중 하나를 선택합니다. 호스트 이름은 **ip-10-0-1-163.ec2.internal**과 유사합니다.
6. Amazon EC2에 수동으로 배포한 베스천 SSH 호스트에서 해당 컨트롤 플레인 호스트에 SSH로 연결합니다. 설치 중 지정한 것과 동일한 SSH 키를 사용해야 합니다.

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

## 3장. 네트워킹 OPERATOR 개요

OpenShift Container Platform은 여러 유형의 네트워킹 Operator를 지원합니다. 이러한 네트워킹 Operator를 사용하여 클러스터 네트워킹을 관리할 수 있습니다.

### 3.1. CNO(CLUSTER NETWORK OPERATOR)

CNO(Cluster Network Operator)는 OpenShift Container Platform 클러스터에서 클러스터 네트워크 구성 요소를 배포하고 관리합니다. 여기에는 설치 중에 클러스터에 대해 선택된 CNI(Container Network Interface) 기본 네트워크 공급자 플러그인 배포가 포함됩니다. 자세한 내용은 [OpenShift Container Platform의 Cluster Network Operator](#) 를 참조하십시오.

### 3.2. DNS OPERATOR

DNS Operator는 CoreDNS를 배포하고 관리하여 Pod에 이름 확인 서비스를 제공합니다. 이를 통해 OpenShift Container Platform에서 DNS 기반 Kubernetes 서비스 검색이 가능합니다. 자세한 내용은 [OpenShift Container Platform의 DNS Operator](#) 를 참조하십시오.

### 3.3. INGRESS OPERATOR

OpenShift Container Platform 클러스터를 생성할 때 클러스터에서 실행되는 Pod 및 서비스는 각각 할당된 IP 주소입니다. IP 주소는 근처에 있는 다른 포트 및 서비스에서 액세스할 수 있지만 외부 클라이언트는 액세스할 수 없습니다. Ingress Operator는 Ingress 컨트롤러 API를 구현하고 OpenShift Container Platform 클러스터 서비스에 대한 외부 액세스를 활성화해야 합니다. 자세한 내용은 [OpenShift Container Platform의 Ingress Operator](#) 를 참조하십시오.

### 3.4. 외부 DNS OPERATOR

외부 DNS Operator는 ExternalDNS를 배포하고 관리하여 외부 DNS 공급자에서 OpenShift Container Platform으로 서비스 및 경로에 대한 이름 확인을 제공합니다. 자세한 내용은 [외부 DNS Operator 이해](#) 를 참조하십시오.



## 4장. OPENSIFT 컨테이너 플랫폼의 CLUSTER NETWORK OPERATOR

CNO(Cluster Network Operator)는 설치 중에 클러스터에 대해 선택한 CNI(Container Network Interface) 기본 네트워크 공급자 플러그인을 포함하여 OpenShift Container Platform 클러스터에 클러스터 네트워크 구성 요소를 배포하고 관리합니다.

### 4.1. CNO(CLUSTER NETWORK OPERATOR)

Cluster Network Operator는 **operator.openshift.io** API 그룹에서 **네트워크** API를 구현합니다. Operator는 데몬 세트를 사용하여 OpenShift SDN 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인 또는 클러스터 설치 중에 선택한 기본 네트워크 공급자 플러그인을 배포합니다.

#### 프로세스

Cluster Network Operator는 설치 중에 Kubernetes **Deployment**로 배포됩니다.

1. 다음 명령을 실행하여 배포 상태를 확인합니다.

```
$ oc get -n openshift-network-operator deployment/network-operator
```

#### 출력 예

```
NAME          READY  UP-TO-DATE  AVAILABLE  AGE
network-operator  1/1    1            1          56m
```

2. 다음 명령을 실행하여 Cluster Network Operator의 상태를 확인합니다.

```
$ oc get clusteroperator/network
```

#### 출력 예

```
NAME    VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
network 4.5.4    True       False        False     50m
```

**AVAILABLE**, **PROGRESSING** 및 **DEGRADED** 필드에서 Operator 상태에 대한 정보를 볼 수 있습니다. Cluster Network Operator가 사용 가능한 상태 조건을 보고하는 경우 **AVAILABLE** 필드는 **True**로 설정됩니다.

### 4.2. 클러스터 네트워크 구성 보기

모든 새로운 OpenShift Container Platform 설치에는 이름이 **cluster**인 **network.config** 오브젝트가 있습니다.

#### 프로세스

- **oc describe** 명령을 사용하여 클러스터 네트워크 구성을 확인합니다.

```
$ oc describe network.config/cluster
```

#### 출력 예

```
■
```

```

Name:      cluster
Namespace:
Labels:    <none>
Annotations: <none>
API Version: config.openshift.io/v1
Kind:      Network
Metadata:
  Self Link:      /apis/config.openshift.io/v1/networks/cluster
Spec: ①
  Cluster Network:
    Cidr:      10.128.0.0/14
    Host Prefix: 23
    Network Type: OpenShiftSDN
  Service Network:
    172.30.0.0/16
Status: ②
  Cluster Network:
    Cidr:      10.128.0.0/14
    Host Prefix: 23
    Cluster Network MTU: 8951
    Network Type: OpenShiftSDN
  Service Network:
    172.30.0.0/16
Events: <none>

```

- ① **Spec** 필드에는 클러스터 네트워크의 구성 상태가 표시됩니다.
- ② **Status** 필드에는 클러스터 네트워크 구성의 현재 상태가 표시됩니다.

### 4.3. CNO(CLUSTER NETWORK OPERATOR) 상태 보기

**oc describe** 명령을 사용하여 상태를 조사하고 Cluster Network Operator의 세부 사항을 볼 수 있습니다.

#### 프로세스

- 다음 명령을 실행하여 Cluster Network Operator의 상태를 확인합니다.

```
$ oc describe clusteroperators/network
```

### 4.4. CNO(CLUSTER NETWORK OPERATOR) 로그 보기

**oc logs** 명령을 사용하여 Cluster Network Operator 로그를 확인할 수 있습니다.

#### 프로세스

- 다음 명령을 실행하여 Cluster Network Operator의 로그를 확인합니다.

```
$ oc logs --namespace=openshift-network-operator deployment/network-operator
```

### 4.5. CNO(CLUSTER NETWORK OPERATOR) 구성

클러스터 네트워크의 구성은 CNO(Cluster Network Operator) 구성의 일부로 지정되며 **cluster**라는 이름의 CR(사용자 정의 리소스) 오브젝트에 저장됩니다. CR은 **operator.openshift.io** API 그룹에서 **Network** API의 필드를 지정합니다.

CNO 구성은 **Network.config.openshift.io** API 그룹의 **Network** API에서 클러스터 설치 중에 다음 필드를 상속하며 이러한 필드는 변경할 수 없습니다.

#### clusterNetwork

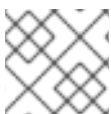
Pod IP 주소가 할당되는 IP 주소 풀입니다.

#### serviceNetwork

서비스를 위한 IP 주소 풀입니다.

#### defaultNetwork.type

OpenShift SDN 또는 OVN-Kubernetes와 같은 클러스터 네트워크 공급자입니다.



#### 참고

클러스터를 설치한 후에는 이전 섹션에 나열된 필드를 수정할 수 없습니다.

**cluster**라는 CNO 오브젝트에서 **defaultNetwork** 오브젝트의 필드를 설정하여 클러스터의 클러스터 네트워크 공급자 구성을 지정할 수 있습니다.

### 4.5.1. CNO(Cluster Network Operator) 구성 오브젝트

CNO(Cluster Network Operator)의 필드는 다음 표에 설명되어 있습니다.

표 4.1. CNO(Cluster Network Operator) 구성 오브젝트


필드	유형	설명
<b>metadata.name</b>	<b>string</b>	CNO 개체 이름입니다. 이 이름은 항상 <b>cluster</b> 입니다.
<b>spec.clusterNetwork</b>	<b>array</b>	Pod IP 주소가 할당되는 IP 주소 블록과 클러스터의 각 개별 노드에 할당된 서브넷 접두사 길이를 지정하는 목록입니다. 예를 들어 다음과 같습니다.  <pre>spec:   clusterNetwork:     - cidr: 10.128.0.0/19       hostPrefix: 23     - cidr: 10.128.32.0/19       hostPrefix: 23</pre> <p>이 값은 준비 전용이며 클러스터 설치 중에 <b>cluster</b> 라는 <b>Network.config.openshift.io</b> 개체에서 상속됩니다.</p>

필드	유형	설명
<b>spec.serviceNetwork</b>	<b>array</b>	<p>서비스의 IP 주소 블록입니다. OpenShift SDN 및 OVN-Kubernetes CNI(Container Network Interface) 네트워크 공급자는 서비스 네트워크에 대한 단일 IP 주소 블록만 지원합니다. 예를 들어 다음과 같습니다.</p> <pre>spec:   serviceNetwork:   - 172.30.0.0/14</pre> <p>이 값은 준비 전용이며 클러스터 설치 중에 <b>cluster</b> 라는 <b>Network.config.openshift.io</b> 개체에서 상속됩니다.</p>
<b>spec.defaultNetwork</b>	<b>object</b>	클러스터 네트워크의 CNI(Container Network Interface) 클러스터 네트워크 공급자를 구성합니다.
<b>spec.kubeProxyConfig</b>	<b>object</b>	이 개체의 필드는 kube-proxy 구성을 지정합니다. OVN-Kubernetes 클러스터 네트워크 공급자를 사용하는 경우 kube-proxy 구성이 적용되지 않습니다.

**defaultNetwork** 오브젝트 구성

**defaultNetwork** 오브젝트의 값은 다음 표에 정의되어 있습니다.

표 4.2. **defaultNetwork** 오브젝트

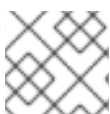
필드	유형	설명
<b>type</b>	<b>string</b>	<p><b>OpenShiftSDN</b> 또는 <b>OVNKubernetes</b> 중 하나이며, 클러스터 네트워크 공급자가 설치 중에 선택됩니다. 클러스터를 설치한 후에는 이 값을 변경할 수 없습니다.</p> <div style="display: flex; align-items: center;">  <div> <p><b>참고</b></p> <p>OpenShift Container Platform은 기본적으로 OpenShift SDN CNI(Container Network Interface) 클러스터 네트워크 공급자를 사용합니다.</p> </div> </div>
<b>openshiftSDNConfig</b>	<b>object</b>	이 오브젝트는 OpenShift SDN 클러스터 네트워크 공급자에만 유효합니다.
<b>ovnKubernetesConfig</b>	<b>object</b>	이 오브젝트는 OVN-Kubernetes 클러스터 네트워크 공급자에만 유효합니다.

OpenShift SDN CNI 네트워크 공급자에 대한 구성

다음 표에서는 OpenShift SDN Container Network Interface (CNI) 클러스터 네트워크 공급자의 구성 필드를 설명합니다.

표 4.3. openshiftSDNConfig 오브젝트

필드	유형	설명
<b>mode</b>	<b>string</b>	OpenShift SDN의 네트워크 격리 모드입니다.
<b>mtu</b>	<b>integer</b>	VXLAN 오버레이 네트워크의 최대 전송 단위(MTU)입니다. 이 값은 일반적으로 자동 구성됩니다.
<b>vxlanPort</b>	<b>integer</b>	모든 VXLAN 패킷에 사용할 포트입니다. 기본값은 <b>4789</b> 입니다.



#### 참고

클러스터 설치 중 클러스터 네트워크 공급자에 대한 구성만 변경할 수 있습니다.

### OpenShift SDN 구성 예

```
defaultNetwork:
  type: OpenShiftSDN
  openshiftSDNConfig:
    mode: NetworkPolicy
    mtu: 1450
    vxlanPort: 4789
```

### OVN-Kubernetes CNI 클러스터 네트워크 공급자에 대한 구성

다음 표에서는 OVN-Kubernetes CNI 클러스터 네트워크 공급자의 구성 필드를 설명합니다.

표 4.4. ovnKubernetesConfig object

필드	유형	설명
<b>mtu</b>	<b>integer</b>	Geneve(Generic Network Virtualization Encapsulation) 오버레이 네트워크의 MTU(최대 전송 단위)입니다. 이 값은 일반적으로 자동 구성됩니다.
<b>genevePort</b>	<b>integer</b>	Geneve 오버레이 네트워크용 UDP 포트입니다.
<b>ipsecConfig</b>	<b>object</b>	필드가 있으면 클러스터에 IPsec이 활성화됩니다.
<b>policyAuditConfig</b>	<b>object</b>	네트워크 정책 감사 로깅을 사용자 정의할 구성 오브젝트를 지정합니다. 설정되지 않으면 기본값 감사 로그 설정이 사용됩니다.


필드	유형	설명
<b>gatewayConfig</b>	<b>object</b>	<p>선택 사항: 송신 트래픽이 노드 게이트웨이로 전송되는 방법을 사용자 정의할 구성 오브젝트를 지정합니다.</p> <div style="display: flex; align-items: flex-start;">  <div> <p><b>참고</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> </div> </div>

표 4.5. policyAuditConfig 오브젝트

필드	유형	설명
<b>rateLimit</b>	integer	노드당 1초마다 생성할 최대 메시지 수입니다. 기본값은 초당 <b>20</b> 개의 메시지입니다.
<b>maxFileSize</b>	integer	감사 로그의 최대 크기(바이트)입니다. 기본값은 <b>50000000</b> 또는 50MB입니다.
<b>대상</b>	string	<p>다음 추가 감사 로그 대상 중 하나입니다.</p> <p><b>libc</b> 호스트에서 journald 프로세스의 libc <b>syslog()</b> 함수입니다.</p> <p><b>udp:&lt;host&gt;:&lt;port&gt;</b> syslog 서버입니다. <b>&lt;host&gt;:&lt;port&gt;</b>를 syslog 서버의 호스트 및 포트로 바꿉니다.</p> <p><b>unix:&lt;file&gt;</b> <b>&lt;file&gt;</b>로 지정된 Unix Domain Socket 파일입니다.</p> <p><b>null</b> 감사 로그를 추가 대상으로 보내지 마십시오.</p>
<b>syslogFacility</b>	string	RFC5424에 정의된 <b>kern</b> 과 같은 syslog 기능입니다. 기본값은 <b>local0</b> 입니다.

표 4.6. gatewayConfig 오브젝트

필드	유형	설명
----	----	----

필드	유형	설명
<b>routingViaHost</b>	<b>boolean</b>	<p>Pod에서 호스트 네트워킹 스택으로 송신 트래픽을 보내려면 이 필드를 <b>true</b>로 설정합니다. 커널 라우팅 테이블에 수동으로 구성된 경로를 사용하는 고도의 전문 설치 및 애플리케이션의 경우 송신 트래픽을 호스트 네트워킹 스택으로 라우팅해야 할 수 있습니다. 기본적으로 송신 트래픽은 클러스터를 종료하기 위해 OVN에서 처리되며 커널 라우팅 테이블의 특수 경로의 영향을 받지 않습니다. 기본값은 <b>false</b>입니다.</p> <p>이 필드는 Open vSwitch 하드웨어 오프로드 기능과 상호 작용합니다. 이 필드를 <b>true</b>로 설정하면 송신 트래픽이 호스트 네트워킹 스택에서 처리되므로 오프로드의 성능 이점이 제공되지 않습니다.</p>



#### 참고

설치 후 활동으로 런타임 시 변경할 수 있는 **gatewayConfig** 필드를 제외하고 클러스터 네트워크 공급자의 구성만 변경할 수 있습니다.


### IPSec가 활성화된 OVN-Kubernetes 구성의 예

```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
    mtu: 1400
    genevePort: 6081
    ipsecConfig: {}
```

#### kubeProxyConfig 오브젝트 구성

**kubeProxyConfig** 오브젝트의 값은 다음 표에 정의되어 있습니다.

표 4.7. kubeProxyConfig object

필드	유형	설명
<b>iptablesSyncPeriod</b>	<b>string</b>	<p><b>iptables</b> 규칙의 새로 고침 간격입니다. 기본값은 <b>30s</b>입니다. 유효 접미사로 <b>s, m, h</b>가 있으며, 자세한 설명은 <a href="#">Go time 패키지</a> 문서를 참조하십시오.</p> <p> <b>참고</b></p> <p>OpenShift Container Platform 4.3 이상에서는 성능이 개선되어 더 이상 <b>iptablesSyncPeriod</b> 매개변수를 조정할 필요가 없습니다.</p>

필드	유형	설명
<code>proxyArguments.iptables-min-sync-period</code>	array	<p><b>iptables</b> 규칙을 새로 고치기 전 최소 기간입니다. 이 필드를 통해 새로 고침 간격이 너무 짧지 않도록 조정할 수 있습니다. 유효 접미사로 <b>s, m, h</b>가 있으며, 자세한 설명은 <a href="#">Go time 패키지</a>를 참조하십시오. 기본값은 다음과 같습니다.</p> <pre>kubeProxyConfig:   proxyArguments:     iptables-min-sync-period:       - 0s</pre>

### 4.5.2. CNO(Cluster Network Operator) 구성 예시

다음 예에서는 전체 CNO 구성이 지정됩니다.

#### CNO(Cluster Network Operator) 개체 예시

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork: 1
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork: 2
  - 172.30.0.0/16
  defaultNetwork: 3
  type: OpenShiftSDN
  openshiftSDNConfig:
    mode: NetworkPolicy
    mtu: 1450
    vxlanPort: 4789
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period:
        - 0s
```

1 2 3 클러스터 설치 중에만 구성됩니다.

## 4.6. 추가 리소스

- [operator.openshift.io API 그룹의 네트워크 API](#)



## 5장. OPENSIFT CONTAINER PLATFORM에서의 DNS OPERATOR

DNS Operator는 CoreDNS를 배포 및 관리하여 Pod에 이름 확인 서비스를 제공하여 OpenShift Container Platform에서 DNS 기반 Kubernetes 서비스 검색을 활성화합니다.

### 5.1. DNS OPERATOR

DNS Operator는 **operator.openshift.io** API 그룹에서 **dns** API를 구현합니다. Operator는 데몬 세트를 사용하여 CoreDNS를 배포하고 데몬 세트에 대한 서비스를 생성하며 이름 확인에서 CoreDNS 서비스 IP 주소를 사용하기 위해 Pod에 명령을 내리도록 kubelet을 구성합니다.

#### 프로세스

DNS Operator는 설치 중에 **Deployment** 오브젝트로 배포됩니다.

1. **oc get** 명령을 사용하여 배포 상태를 확인합니다.

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

#### 출력 예

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

2. **oc get** 명령을 사용하여 DNS Operator의 상태를 확인합니다.

```
$ oc get clusteroperator/dns
```

#### 출력 예

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED	SINCE
dns	4.1.0-0.11	True	False	False	92m

**AVAILABLE**, **PROGRESSING** 및 **DEGRADED**는 Operator의 상태에 대한 정보를 제공합니다. **AVAILABLE**은 CoreDNS 데몬 세트에서 1개 이상의 포트가 **Available** 상태 조건을 보고할 때 **True**입니다.

### 5.2. DNS OPERATOR MANAGEMENTSTATE 변경

DNS는 CoreDNS 구성 요소를 관리하여 클러스터의 pod 및 서비스에 대한 이름 확인 서비스를 제공합니다. DNS Operator의 **managementState**는 기본적으로 **Managed**로 설정되어 있으며 이는 DNS Operator가 리소스를 적극적으로 관리하고 있음을 의미합니다. **Unmanaged**로 변경할 수 있습니다. 이는 DNS Operator가 해당 리소스를 관리하지 않음을 의미합니다.

다음은 DNS Operator **managementState**를 변경하는 사용 사례입니다.

- 사용자가 개발자이며 구성 변경을 테스트하여 CoreDNS의 문제가 해결되었는지 확인하려고 합니다. **managementState**를 **Unmanaged**로 설정하여 DNS Operator가 수정 사항을 덮어쓰지 않도록 할 수 있습니다.

- 클러스터 관리자이며 CoreDNS 관련 문제를 보고했지만 문제가 해결될 때까지 해결 방법을 적용해야 합니다. DNS Operator의 **managementState** 필드를 **Unmanaged**로 설정하여 해결 방법을 적용할 수 있습니다.

## 절차

- managementState** DNS Operator 변경:

```
oc patch dns.operator.openshift.io default --type merge --patch '{"spec": {"managementState": "Unmanaged"}}'
```

## 5.3. DNS POD 배치 제어

DNS Operator에는 2개의 데몬 세트(CoreDNS 및 **/etc/hosts** 파일 관리용)가 있습니다. 이미지 가져오기를 지원할 클러스터 이미지 레지스트리의 항목을 추가하려면 모든 노드 호스트에서 **/etc/hosts**의 데몬 세트를 실행해야 합니다. 보안 정책은 CoreDNS에 대한 데몬 세트가 모든 노드에서 실행되지 않도록 하는 노드 쌍 간 통신을 금지할 수 있습니다.

클러스터 관리자는 사용자 정의 노드 선택기를 사용하여 특정 노드에서 CoreDNS를 실행하거나 실행하지 않도록 데몬 세트를 구성할 수 있습니다.

### 사전 요구 사항

- oc** CLI를 설치했습니다.
- cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

### 프로세스

- 특정 노드 간 통신을 방지하려면 **spec.nodePlacement.nodeSelector** API 필드를 구성합니다.

- 이름이 **default**인 DNS Operator 오브젝트를 수정합니다.

```
$ oc edit dns.operator/default
```

- spec.nodePlacement.nodeSelector** API 필드에 컨트롤 플레인 노드만 포함하는 노드 선택기를 지정합니다.

```
spec:
  nodePlacement:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
```

- CoreDNS의 데몬 세트가 노드에서 실행되도록 테인트 및 허용 오차를 구성합니다.

- 이름이 **default**인 DNS Operator 오브젝트를 수정합니다.

```
$ oc edit dns.operator/default
```

- 테인트 키와 테인트에 대한 허용 오차를 지정합니다.

```
spec:
  nodePlacement:
    tolerations:
```

```
- effect: NoExecute
  key: "dns-only"
  operators: Equal
  value: abc
  tolerationSeconds: 3600 1
```

- 1 테인트가 **dns-only**인 경우 무기한 허용될 수 있습니다. **tolerationSeconds**를 생략할 수 있습니다.

## 5.4. 기본 DNS보기

모든 새로운 OpenShift Container Platform 설치에서는 **dns.operator**의 이름이 **default**로 지정됩니다.

### 프로세스

1. **oc describe** 명령을 사용하여 기본 **dns**를 확인합니다.

```
$ oc describe dns.operator/default
```

### 출력 예

```
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 Cluster Domain 필드는 정규화된 pod 및 service 도메인 이름을 구성하는 데 사용되는 기본 DNS 도메인입니다.
- 2 Cluster IP는 이름을 확인하기 위한 주소 Pod 쿼리입니다. IP는 service CIDR 범위에서 10번째 주소로 정의됩니다.

2. 클러스터의 service CIDR을 찾으려면 **oc get** 명령을 사용합니다.

```
$ oc get networks.config/cluster -o jsonpath='{$.status.serviceNetwork}'
```

### 출력 예

```
[172.30.0.0/16]
```

## 5.5. DNS 전달 사용

DNS 전달을 사용하여 **/etc/resolv.conf** 파일의 기본 전달 구성을 다음과 같은 방법으로 덮어쓸 수 있습니다.

- 모든 영역에 대해 이름 서버를 지정합니다. 전달된 영역이 OpenShift Container Platform에서 관리하는 Ingress 도메인인 경우 도메인에 대한 업스트림 이름 서버를 승인해야 합니다.
- 업스트림 DNS 서버 목록을 제공합니다.
- 기본 전달 정책을 변경합니다.



**참고**

기본 도메인의 DNS 전달 구성에는 **/etc/resolv.conf** 파일과 업스트림 DNS 서버에 지정된 기본 서버가 모두 있을 수 있습니다.

**절차**

1. 이름이 **default**인 DNS Operator 오브젝트를 수정합니다.

```
$ oc edit dns.operator/default
```

이를 통해 Operator는 **Server** 를 기반으로 추가 서버 구성 블록으로 **dns-default** 라는 구성 맵을 생성하고 업데이트할 수 있습니다. 서버에 쿼리와 일치하는 영역이 없는 경우 이름 확인은 업스트림 DNS 서버로 대체됩니다.

**DNS 전달 구성**

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
    - name: example-server 1
      zones: 2
      - example.com
    forwardPlugin:
      policy: Random 3
      upstreams: 4
      - 1.1.1.1
      - 2.2.2.2:5353
    upstreamResolvers: 5
      policy: Random 6
      upstreams: 7
      - type: SystemResolvConf 8
      - type: Network
        address: 1.2.3.4 9
        port: 53 10
```

- 1 **rfc6335** 서비스 이름 구문을 준수해야 합니다.
- 2 **rfc1123** 서비스 이름 구문의 하위 도메인 정의를 준수해야 합니다. cluster domain, **cluster.local.local**은 **zones** 필드에 유효하지 않은 하위 도메인입니다.

- 3 업스트림 리졸버를 선택할 정책을 정의합니다. 기본값은 **Random** 입니다. 또한 **RoundRobin** 및 **Sequential** 값을 사용할 수도 있습니다.
- 4 **forwardPlugin**당 최대 15개의 업스트림이 허용됩니다.
- 5 선택 사항: 이를 사용하여 기본 정책을 재정의하고 DNS 확인을 기본 도메인의 지정된 DNS 확인자(업스트림 확인자)로 전달할 수 있습니다. 업스트림 리졸버를 제공하지 않으면 DNS 이름 쿼리는 **/etc/resolv.conf** 의 서버로 이동합니다.
- 6 조회를 위해 업스트림 서버가 선택된 순서를 결정합니다. 다음 값 중 하나를 지정할 수 있습니다: **Random, RoundRobin**, 또는 **Sequential**. 기본값은 **Sequential** 입니다.
- 7 선택 사항: 이를 사용하여 업스트림 확인자를 제공할 수 있습니다.
- 8 두 가지 유형의 업스트림 (**SystemResolvConf** 및 **Network**)을 지정할 수 있습니다. **SystemResolvConf** 는 **/etc/resolv.conf** 및 **Network** 를 사용하도록 업스트림을 구성하여 **Networkresolver** 를 정의합니다. 하나 또는 둘 다를 지정할 수 있습니다.
- 9 지정된 유형이 **네트워크** 이면 IP 주소를 제공해야 합니다. **address** 필드는 유효한 IPv4 또는 IPv6 주소여야 합니다.
- 10 지정된 유형이 **네트워크** 이면 선택적으로 포트를 제공할 수 있습니다. **port** 필드는 **1** 에서 **65535** 사이의 값을 가져야 합니다. 업스트림 포트를 지정하지 않으면 기본적으로 포트 853 이 시도됩니다.

고도로 규제된 환경에서 작업하는 경우 추가 DNS 트래픽 및 데이터 개인 정보를 확보할 수 있도록 요청을 업스트림 리졸버에 전달할 때 DNS 트래픽을 보호할 수 있는 기능이 필요할 수 있습니다. 클러스터 관리자는 전달된 DNS 쿼리에 대해 TLS(Transport Layer Security)를 구성할 수 있습니다.

### TLS를 사용하여 DNS 전달 구성

```

apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
    - name: example-server 1
  zones: 2
    - example.com
  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** **rfc6335** 서비스 이름 구문을 준수해야 합니다.
- 2** **rfc1123** 서비스 이름 구문의 하위 도메인 정의를 준수해야 합니다. cluster domain, **cluster.local.local**은 **zones** 필드에 유효하지 않은 하위 도메인입니다. 클러스터 도메인에 해당하는 **cluster.local**은 영역에 유효하지 않은 하위 도메인입니다.
- 3** 전달된 DNS 쿼리를 위해 TLS를 구성할 때 값 **TLS** 를 갖도록 **transport** 필드를 설정합니다. 기본적으로 CoreDNS는 10초 동안 연결을 전달했습니다. 요청이 발행되지 않은 경우 CoreDNS는 10초 동안 열려 있는 TCP 연결을 유지합니다. 대규모 클러스터를 사용하면 노드당 연결을 시작할 수 있으므로 DNS 서버에서 많은 새 연결이 열려 있을 수 있음을 확인합니다. 성능 문제를 방지하기 위해 적절하게 DNS 계층을 설정합니다.
- 4** 전달된 DNS 쿼리에 대해 TLS를 구성할 때 이는 업스트림 TLS 서버 인증서를 검증하기 위해 SNI(서버 이름 표시)의 일부로 사용되는 필수 서버 이름입니다.
- 5** 업스트림 리졸버를 선택할 정책을 정의합니다. 기본값은 **Random** 입니다. 또한 **RoundRobin** 및 **Sequential** 값을 사용할 수도 있습니다.
- 6** 필수 항목입니다. 이를 사용하여 업스트림 확인자를 제공할 수 있습니다. **forwardPlugin** 항목당 최대 15 개의 업스트림 항목을 사용할 수 있습니다.
- 7** 선택 사항: 이를 사용하여 기본 정책을 재정의하고 DNS 확인을 기본 도메인의 지정된 DNS 확인자(업스트림 확인자)로 전달할 수 있습니다. 업스트림 리졸버를 제공하지 않으면 DNS 이름 쿼리는 **/etc/resolv.conf** 의 서버로 이동합니다.
- 8** **네트워크** 유형은 이 업스트림 확인자가 **/etc/resolv.conf** 에 나열된 업스트림 확인자와 별도로 전달된 요청을 처리해야 함을 나타냅니다. TLS를 사용하는 경우 **네트워크** 유형만 허용되며 IP 주소를 제공해야 합니다.
- 9** **address** 필드는 유효한 IPv4 또는 IPv6 주소여야 합니다.
- 10** 선택적으로 포트를 제공할 수 있습니다. **포트** 는 1 에서 **65535** 사이의 값을 가져야합니다. 업스트림 포트를 지정하지 않으면 기본적으로 포트 853이 시도됩니다.



**참고**

**servers** 가 정의되지 않았거나 유효하지 않은 경우 구성 맵에는 기본 서버만 포함됩니다.

2. 구성 맵을 표시합니다.

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

이전 샘플 DNS를 기반으로 하는 샘플 DNS ConfigMap

```

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 ❶
    }
    .: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

```

❶ **forwardPlugin**을 변경하면 CoreDNS 데몬 세트의 롤링 업데이트가 트리거됩니다.

### 추가 리소스

- DNS 전달에 대한 자세한 내용은 [CoreDNS 전달 설명서](#)를 참조하십시오.

## 5.6. DNS OPERATOR 상태

**oc describe** 명령을 사용하여 상태를 확인하고 DNS Operator의 세부 사항을 볼 수 있습니다.

### 프로세스

DNS Operator의 상태를 확인하려면 다음을 실행합니다.

```
$ oc describe clusteroperators/dns
```

## 5.7. DNS OPERATOR 로그

**oc logs** 명령을 사용하여 DNS Operator 로그를 확인할 수 있습니다.

### 프로세스

DNS Operator의 로그를 확인합니다.

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

## 5.8. COREDNS 로그 수준 설정

CoreDNS 로그 수준을 구성하여 로깅된 오류 메시지에 대한 세부 정보 양을 확인할 수 있습니다. CoreDNS 로그 수준에 유효한 값은 **Normal, Debug, Trace** 입니다. 기본 **logLevel** 은 **Normal** 입니다.



### 참고

오류 플러그인은 항상 활성화되어 있습니다. 다음 **logLevel** 설정은 다른 오류 응답을 보고합니다.

- **logLevel:Normal** 는 "errors" 클래스를 활성화합니다. **log . { class error } }**.
- **logLevel:Debug** 는 "denial" 클래스: **log . { 클래스 거부 오류 }**를 활성화합니다.
- **logLevel:Trace** 를 사용하면 "모든" 클래스: **log . { 클래스 all }**.

### 절차

- **logLevel** 을 **Debug** 로 설정하려면 다음 명령을 입력합니다.

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Debug"}}' --type=merge
```

- **logLevel** 을 **Trace** 로 설정하려면 다음 명령을 입력합니다.

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"logLevel":"Trace"}}' --type=merge
```

### 검증

- 원하는 로그 수준을 설정하려면 구성 맵을 확인합니다.

```
$ oc get configmap/dns-default -n openshift-dns -o yaml
```

## 5.9. COREDNS OPERATOR 로그 수준 설정

클러스터 관리자는 OpenShift DNS 문제를 더 빠르게 추적하도록 Operator 로그 수준을 구성할 수 있습니다. **operatorLogLevel** 에 유효한 값은 **Normal, Debug** 및 **Trace** 입니다. 추적은 가장 자세한 정보가 있습니다. 기본 **operatorLogLevel** 은 **Normal** 입니다. 여기에는 추적, 디버그, 정보, 경고, 오류, Fatal 및 Panic 의 7 가지 로깅 수준이 있습니다. 로깅 수준이 설정되면 해당 심각도 또는 그 이상의 모든 항목이 있는 로 그 항목이 기록됩니다.

- **operatorLogLevel: "Normal"** 은 **logrus.SetLogLevel("Info")** 을 설정합니다.
- **operatorLogLevel: "Debug"** 는 **logrus.SetLogLevel("Debug")** 을 설정합니다.
- **operatorLogLevel: "Trace"** 는 **logrus.SetLogLevel("Trace")** 을 설정합니다.

### 절차



- **operatorLogLevel** 을 디버그 로 설정 하려면 다음 명령을 입력합니다.

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Debug"}}' --  
type=merge
```

- **operatorLogLevel** 을 **Trace** 로 설정 하려면 다음 명령을 입력합니다.

```
$ oc patch dnses.operator.openshift.io/default -p '{"spec":{"operatorLogLevel":"Trace"}}' --  
type=merge
```

## 6장. OPENSIFT CONTAINER PLATFORM에서의 INGRESS OPERATOR

### 6.1. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR

OpenShift Container Platform 클러스터를 생성할 때 클러스터에서 실행되는 Pod 및 서비스에는 각각 자체 IP 주소가 할당됩니다. IP 주소는 내부에서 실행되지만 외부 클라이언트가 액세스할 수 없는 다른 pod 및 서비스에 액세스할 수 있습니다. Ingress Operator는 **IngressController** API를 구현하며 OpenShift Container Platform 클러스터 서비스에 대한 외부 액세스를 활성화하는 구성 요소입니다.

Ingress Operator를 사용하면 라우팅을 처리하기 위해 하나 이상의 HAProxy 기반 **Ingress 컨트롤러**를 배포하고 관리하여 외부 클라이언트가 서비스에 액세스할 수 있습니다. Ingress Operator를 사용하여 OpenShift 컨테이너 플랫폼 **Route** 및 Kubernetes **Ingress** 리소스를 지정하면 수신 트래픽을 라우팅할 수 있습니다. **endpointPublishingStrategy** 유형 및 내부 로드 밸런싱을 정의하는 기능과 같은 Ingress 컨트롤러 내 구성은 Ingress 컨트롤러 끝점을 게시하는 방법을 제공합니다.

### 6.2. INGRESS 구성 자산

설치 프로그램은 **config.openshift.io** API 그룹인 **cluster-ingress-02-config.yml**에 **Ingress** 리소스가 포함된 자산을 생성합니다.

#### Ingress 리소스의 YAML 정의

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.openshift demos.com
```

설치 프로그램은 이 자산을 **manifests / 디렉터리**의 **cluster-ingress-02-config.yml** 파일에 저장합니다. 이 **Ingress** 리소스는 Ingress와 관련된 전체 클러스터 구성을 정의합니다. 이 Ingress 구성은 다음과 같이 사용됩니다.


- Ingress Operator는 클러스터 Ingress 구성에 설정된 도메인을 기본 Ingress 컨트롤러의 도메인으로 사용합니다.
- OpenShift API Server Operator는 클러스터 Ingress 구성의 도메인을 사용합니다. 이 도메인은 명시적 호스트를 지정하지 않는 **Route** 리소스에 대한 기본 호스트를 생성할 수도 있습니다.

### 6.3. INGRESS 컨트롤러 구성 매개변수

**ingresscontrollers.operator.openshift.io** 리소스에서 제공되는 구성 매개변수는 다음과 같습니다.

매개변수	설명
------	----

매개변수	설명
<b>domain</b>	<p><b>domain</b>은 Ingress 컨트롤러에서 제공하는 DNS 이름이며 여러 기능을 구성하는데 사용됩니다.</p> <ul style="list-style-type: none"> <li>● <b>LoadBalancerService</b> 끝점 게시 방식에서는 <b>domain</b>을 사용하여 DNS 레코드를 구성합니다. <b>endpointPublishingStrategy</b>를 참조하십시오.</li> <li>● 생성된 기본 인증서를 사용하는 경우, 인증서는 <b>domain</b> 및 해당 <b>subdomains</b>에 유효합니다. <b>defaultCertificate</b>를 참조하십시오.</li> <li>● 사용자가 외부 DNS 레코드의 대상 위치를 확인할 수 있도록 이 값이 개별 경로 상태에 게시됩니다.</li> </ul> <p><b>domain</b> 값은 모든 Ingress 컨트롤러에서 고유해야 하며 업데이트할 수 없습니다.</p> <p>비어 있는 경우 기본값은 <b>ingress.config.openshift.io/cluster.spec.domain</b>입니다.</p>
<b>replicas</b>	<p><b>replicas</b>는 원하는 개수의 Ingress 컨트롤러 복제본입니다. 설정되지 않은 경우, 기본값은 <b>2</b>입니다.</p>
<b>endpointPublishingStrategy</b>	<p><b>endpointPublishingStrategy</b>는 Ingress 컨트롤러 끝점을 다른 네트워크에 게시하고 로드 밸런서 통합을 활성화하며 다른 시스템에 대한 액세스를 제공하는 데 사용됩니다.</p> <p>설정되지 않은 경우, 기본값은 <b>infrastructure.config.openshift.io/cluster.status.platform</b>을 기반으로 다음과 같습니다.</p> <ul style="list-style-type: none"> <li>● Amazon Web Services (AWS): <b>LoadBalancerService</b> (외부 범위 포함)</li> <li>● Azure: <b>LoadBalancerService</b> (외부 범위 포함)</li> <li>● GCP(Google Cloud Platform): <b>LoadBalancerService</b> (외부 범위 포함)</li> <li>● 베어 메탈: <b>NodePortService</b></li> <li>● 기타: <b>HostNetwork</b></li> </ul>

매개변수	설명	참고
		<p><b>HostNetwork</b>에는 선택적 바인딩 포트에 대해 다음과 같은 기본값이 있는 <b>hostNetwork</b> 필드가 있습니다. <b>httpPort: 80, httpsPort: 443, statsPort: 1936</b>. 바인딩 포트를 사용하면 <b>HostNetwork</b> 전략의 동일한 노드에 여러 Ingress 컨트롤러를 배포할 수 있습니다.</p> <p><b>예제</b></p> <pre> 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                 </pre> <p><b>참고</b></p> <p>RHOSP(Red Hat OpenStack Platform)에서 <b>LoadBalancerService</b> 끝점 게시 전략은 클라우드 공급자가 상태 모니터를 생성하도록 구성된 경우에만 지원됩니다. RHOSP 16.1 및 16.2의 경우 이 전략은 Amphora Octavia 공급자를 사용하는 경우에만 가능합니다.</p> <p>자세한 내용은 RHOSP 설치 설명서의 "클라우드 공급자 옵션 설정" 섹션을 참조하십시오.</p>
<p><b>defaultCertificate</b></p>	<p><b>defaultCertificate</b> 값은 Ingress 컨트롤러가 제공하는 기본 인증서가 포함된 보안을 위한 참조입니다. 경로는 고유한 인증서를 지정하지 않으면 대부분의 플랫폼의 경우 사용됩니다. <b>defaultCertificate</b> 필드를 구성할 수 있습니다. <b>endpointPublishingStrategy</b> 필드를 구성할 수 있습니다. <b>endpointPublishingStrategy</b> 필드는 구성할 수 있습니다.</p> <p>보안에는 키와 데이터, 즉 <b>*tls.crt</b>: 인증서 파일 내용 <b>*tls.key</b>: 키 파일 내용이 포함되어야 합니다.</p> <ul style="list-style-type: none"> <li>● <b>loadBalancer.scope</b></li> <li>● <b>loadBalancer.providerParameters.gcpClientAccess</b></li> <li>● <b>hostNetwork.protocol</b></li> <li>● <b>nodePort.protocol</b></li> </ul> <p>설정하지 않으면 인위적으로 인증서가 자동으로 생성되어 사용됩니다. 인증서는 Ingress 컨트롤러 도메인 및 하위 도메인에 유효하며 생성된 인증서의 CA는 클러스터의 인위적으로 생성된 인증서와 통합됩니다.</p> <p>생성된 인증서 또는 사용자 정의 인증서는 OpenShift Container Platform 내장 OAuth 서버와 자동으로 통합됩니다.</p>	
<p><b>namespaceSelector</b></p>	<p><b>namespaceSelector</b>는 Ingress 컨트롤러가 서비스를 제공하는 네임스페이스 집합을 필터링하는 데 사용됩니다. 이는 분할을 구현하는 데 유용합니다.</p>	
<p><b>routeSelector</b></p>	<p><b>routeSelector</b>는 Ingress 컨트롤러가 서비스를 제공하는 경로 집합을 필터링하는 데 사용됩니다. 이는 분할을 구현하는 데 유용합니다.</p>	

매개변수	설명
<p><b>nodePlacement</b></p>	<p><b>nodePlacement</b>를 사용하면 Ingress 컨트롤러의 스케줄링을 명시적으로 제어할 수 있습니다.</p> <p>설정하지 않으면 기본값이 사용됩니다.</p> <div data-bbox="517 405 625 875" style="float: left; width: 60px; height: 210px; background: repeating-linear-gradient(45deg, transparent, transparent 2px, #ccc 2px, #ccc 4px); border: 1px solid #ccc; margin-bottom: 10px;"></div> <p><b>참고</b></p> <p><b>nodePlacement</b> 매개변수는 <b>nodeSelector</b> 및 <b>tolerations</b>의 두 부분으로 구성됩니다. 예를 들어 다음과 같습니다.</p> <pre style="border-left: 2px solid #ccc; padding-left: 10px; margin-left: 20px;">nodePlacement: nodeSelector:   matchLabels:     kubernetes.io/os: linux tolerations:   - effect: NoSchedule     operator: Exists</pre>
<p><b>tlsSecurityProfile</b></p>	<p><b>tlsSecurityProfile</b>은 Ingress 컨트롤러의 TLS 연결 설정을 지정합니다.</p> <p>설정되지 않으면, 기본값은 <b>apiservers.config.openshift.io/cluster</b> 리소스를 기반으로 설정됩니다.</p> <p><b>Old, Intermediate</b> 및 <b>Modern</b> 프로파일 유형을 사용하는 경우 유효한 프로파일 구성은 릴리스마다 변경될 수 있습니다. 예를 들어, 릴리스 <b>X.Y.Z</b>에 배포된 <b>Intermediate</b> 프로파일을 사용하도록 설정한 경우 <b>X.Y.Z+1</b> 릴리스로 업그레이드하면 새 프로파일 구성이 Ingress 컨트롤러에 적용되어 롤아웃이 발생할 수 있습니다.</p> <p>Ingress 컨트롤러의 최소 TLS 버전은 <b>1.1</b>이며 최대 TLS 버전은 <b>1.3</b>입니다.</p> <div data-bbox="517 1756 625 1890" style="float: left; width: 60px; height: 60px; background: repeating-linear-gradient(45deg, transparent, transparent 2px, #ccc 2px, #ccc 4px); border: 1px solid #ccc; margin-bottom: 10px;"></div> <p><b>참고</b></p> <p>구성된 보안 프로파일의 암호 및 최소 TLS 버전은 <b>TLSProfile</b> 상태에 반영됩니다.</p> <div data-bbox="517 1935 625 2069" style="float: left; width: 60px; height: 60px; background: repeating-linear-gradient(45deg, transparent, transparent 2px, #ccc 2px, #ccc 4px); border: 1px solid #ccc; margin-bottom: 10px;"></div> <p><b>중요</b></p> <p>Ingress Operator는 <b>Old</b> 또는 <b>Custom</b> 프로파일의 TLS<b>1.0</b> 을 <b>1.1</b> 로 변환합니다.</p>

매개변수	설명
<p><b>clientTLS</b></p>	<p><b>clientTLS</b>는 클러스터 및 서비스에 대한 클라이언트 액세스를 인증하므로 상호 TLS 인증이 활성화됩니다. 설정되지 않은 경우 클라이언트 TLS가 활성화되지 않습니다.</p> <p><b>clientTLS</b>에는 필수 하위 필드인 <b>spec.clientTLS.clientCertificatePolicy</b> 및 <b>spec.clientTLS.ClientCA</b>가 있습니다.</p> <p><b>ClientCertificatePolicy</b> 하위 필드는 <b>Required</b> 또는 <b>Optional</b> 이라는 두 값 중 하나를 허용합니다. <b>ClientCA</b> 하위 필드는 openshift-config 네임스페이스에 있는 구성 맵을 지정합니다. 구성 맵에는 CA 인증서 번들이 포함되어야 합니다. <b>AllowedSubjectPatterns</b>는 요청을 필터링할 유효한 클라이언트 인증서의 고유 이름과 일치하는 정규식 목록을 지정하는 선택적 값입니다. 정규 표현식은 PCRE 구문을 사용해야 합니다. 클라이언트 인증서의 고유 이름과 일치하는 하나 이상의 패턴이 있어야 합니다. 그러지 않으면 Ingress 컨트롤러에서 인증서를 거부하고 연결을 거부합니다. 지정하지 않으면 Ingress 컨트롤러에서 고유 이름을 기반으로 인증서를 거부하지 않습니다.</p>
<p><b>routeAdmission</b></p>	<p><b>routeAdmission</b>은 네임스페이스에서 클레임을 허용 또는 거부하는 등 새로운 경로 클레임을 처리하기 위한 정책을 정의합니다.</p> <p><b>namespaceOwnership</b>은 네임스페이스에서 호스트 이름 클레임을 처리하는 방법을 설명합니다. 기본값은 <b>Strict</b>입니다.</p> <ul style="list-style-type: none"> <li>● <b>Strict</b>: 경로가 네임스페이스에서 동일한 호스트 이름을 요청하는 것을 허용하지 않습니다.</li> <li>● <b>InterNamespaceAllowed</b>: 경로가 네임스페이스에서 동일한 호스트 이름의 다른 경로를 요청하도록 허용합니다.</li> </ul> <p><b>wildcardPolicy</b>는 Ingress 컨트롤러에서 와일드카드 정책이 포함된 경로를 처리하는 방법을 설명합니다.</p> <ul style="list-style-type: none"> <li>● <b>WildcardsAllowed</b>: 와일드카드 정책이 포함된 경로가 Ingress 컨트롤러에 의해 허용됨을 나타냅니다.</li> <li>● <b>WildcardsDisallowed</b>: 와일드카드 정책이 <b>None</b>인 경로만 Ingress 컨트롤러에 의해 허용됨을 나타냅니다. <b>WildcardsAllowed</b>에서 <b>WildcardsDisallowed</b>로 <b>wildcardPolicy</b>를 업데이트하면 와일드카드 정책이 <b>Subdomain</b>인 허용되는 경로의 작동이 중지됩니다. Ingress 컨트롤러에서 이러한 경로를 다시 허용하려면 이 경로를 설정이 <b>None</b>인 와일드카드 정책으로 다시 생성해야 합니다. 기본 설정은 <b>WildcardsDisallowed</b>입니다.</li> </ul>

매개변수	설명
<b>IngressControllerLogging</b>	<p><b>logging</b>은 어디에서 무엇이 기록되는지에 대한 매개변수를 정의합니다. 이 필드가 비어 있으면 작동 로그는 활성화되지만 액세스 로그는 비활성화됩니다.</p> <ul style="list-style-type: none"> <li>● <b>access</b>는 클라이언트 요청이 기록되는 방법을 설명합니다. 이 필드가 비어 있으면 액세스 로깅이 비활성화됩니다. <ul style="list-style-type: none"> <li>○ <b>destination</b>은 로그 메시지의 대상을 설명합니다. <ul style="list-style-type: none"> <li>■ <b>type</b>은 로그 대상의 유형입니다. <ul style="list-style-type: none"> <li>● <b>Container</b>는 로그가 사이드카 컨테이너로 이동하도록 지정합니다. Ingress Operator는 Ingress 컨트롤러 pod에서 <b>logs</b> 라는 컨테이너를 구성하고 컨테이너에 로그를 작성하도록 Ingress 컨트롤러를 구성합니다. 관리자는 이 컨테이너에서 로그를 읽는 사용자 정의 로깅 솔루션을 구성해야 합니다. 컨테이너 로그를 사용한다는 것은 로그 비율이 컨테이너 런타임 용량 또는 사용자 정의 로깅 솔루션 용량을 초과하면 로그가 삭제될 수 있음을 의미합니다.</li> <li>● <b>Syslog</b>는 로그가 Syslog 끝점으로 전송되도록 지정합니다. 관리자는 Syslog 메시지를 수신할 수 있는 끝점을 지정해야 합니다. 관리자가 사용자 정의 Syslog 인스턴스를 구성하는 것이 좋습니다.</li> </ul> </li> </ul> </li> <li>■ <b>컨테이너</b>는 <b>Container</b> 로깅 대상 유형의 매개변수를 설명합니다. 현재는 컨테이너 로깅에 대한 매개변수가 없으므로 이 필드는 비어 있어야 합니다.</li> <li>■ <b>syslog</b>는 <b>Syslog</b> 로깅 대상 유형의 매개변수를 설명합니다. <ul style="list-style-type: none"> <li>● <b>address</b>는 로그 메시지를 수신하는 syslog 끝점의 IP 주소입니다.</li> <li>● <b>port</b>는 로그 메시지를 수신하는 syslog 끝점의 UDP 포트 번호입니다.</li> <li>● <b>maxLength</b>는 syslog 메시지의 최대 길이입니다. <b>480</b>에서 <b>4096</b> 바이트 사이여야 합니다. 이 필드가 비어 있으면 최대 길이는 기본값인 <b>1024</b> 바이트로 설정됩니다.</li> <li>● <b>facility</b>는 로그 메시지의 syslog 기능을 지정합니다. 이 필드가 비어 있으면 장치가 <b>local1</b>이 됩니다. 아니면 <b>kern, user, mail, daemon, auth, syslog, lpr, news, uucp, cron, auth2, ftp, ntp, audit, alert, cron2, local0, local1, local2, local3, local4, local5, local6</b> 또는 <b>local7</b> 중에서 유효한 syslog 장치를 지정해야 합니다.</li> </ul> </li> <li>○ <b>httpLogFormat</b>은 HTTP 요청에 대한 로그 메시지의 형식을 지정합니다. 이 필드가 비어 있으면 로그 메시지는 구현의 기본 HTTP 로그 형식을 사용합니다. HAProxy의 기본 HTTP 로그 형식과 관련된 내용은 <a href="#">HAProxy 문서</a>를 참조하십시오.</li> </ul> </li> </ul>

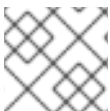
매개변수	설명
<p><b>httpHeaders</b></p>	<p><b>httpHeaders</b>는 HTTP 헤더에 대한 정책을 정의합니다.</p> <p><b>IngressControllerHTTPHeaders</b> 에 <b>forwardedHeaderPolicy</b> 를 설정하면 Ingress 컨트롤러가 <b>Forwarded,X-Forwarded-For,X-Forwarded-Port,X-Forwarded-Proto -Version</b> HTTP 헤더를 설정하는 시기와 방법을 지정합니다.</p> <p>기본적으로 정책은 <b>Append</b>로 설정됩니다.</p> <ul style="list-style-type: none"> <li>● <b>Append</b>는 Ingress 컨트롤러에서 기존 헤더를 유지하면서 헤더를 추가하도록 지정합니다.</li> <li>● <b>Replace</b>는 Ingress 컨트롤러에서 헤더를 설정하고 기존 헤더를 제거하도록 지정합니다.</li> <li>● <b>IfNone</b>은 헤더가 아직 설정되지 않은 경우 Ingress 컨트롤러에서 헤더를 설정하도록 지정합니다.</li> <li>● <b>Never</b>는 Ingress 컨트롤러에서 헤더를 설정하지 않고 기존 헤더를 보존하도록 지정합니다.</li> </ul> <p><b>headerNameCaseAdjustments</b>를 설정하여 HTTP 헤더 이름에 적용할 수 있는 대/소문자 조정을 지정할 수 있습니다. 각 조정은 원하는 대문자를 사용하여 HTTP 헤더 이름으로 지정됩니다. 예를 들어 <b>X-Forwarded-For</b>를 지정하면 지정된 대문자를 사용하도록 <b>x-forwarded-for</b> HTTP 헤더를 조정해야 합니다.</p> <p>이러한 조정은 HTTP/1을 사용하는 경우에만 일반 텍스트, 에지 종료 및 재암호화 경로에 적용됩니다.</p> <p>요청 헤더의 경우 이러한 조정은 <b>haproxy.router.openshift.io/h1-adjust-case=true</b> 주석이 있는 경로에만 적용됩니다. 응답 헤더의 경우 이러한 조정이 모든 HTTP 응답에 적용됩니다. 이 필드가 비어 있으면 요청 헤더가 조정되지 않습니다.</p>
<p><b>httpCompression</b></p>	<p><b>httpCompression</b> 은 HTTP 트래픽 압축에 대한 정책을 정의합니다.</p> <ul style="list-style-type: none"> <li>● <b>mime types</b>은 압축을 적용할 MIME 유형 목록을 정의합니다. 예를 들어 <b>text/css; charset=utf-8,text/html, image/svg+xml,application/octet-stream,X-custom/customsub, type/subtype; [;attribute=value]</b>. 유형은 다음과 같습니다. <b>application, image, message, multipart, text, video, or a custom type prefaced by X-</b>; 예를 들어 <b>MIME</b> 유형 및 하위 유형에 대한 전체 표기법을 보려면 <b>RFC1341</b>을 참조하십시오.</li> </ul>
<p><b>httpErrorCodePages</b></p>	<p><b>httpErrorCodePages</b>는 사용자 정의 HTTP 오류 코드 응답 페이지를 지정합니다. 기본적으로 IngressController는 IngressController 이미지에 빌드된 오류 페이지를 사용합니다.</p>



매개변수	설명
<p><b>httpCaptureCookies</b></p>	<p><b>httpCapECDHECookies</b> 는 액세스 로그에서 캡처하려는 HTTP 쿠키를 지정합니다. <b>httpCapECDHECookies</b> 필드가 비어 있으면 액세스 로그에서 쿠키를 캡처하지 않습니다.</p> <p>캡처하려는 모든 쿠키의 경우 다음 매개변수는 <b>IngressController</b> 구성에 있어야 합니다.</p> <ul style="list-style-type: none"> <li>● <b>name</b> 은 쿠키 이름을 지정합니다.</li> <li>● <b>MaxLength</b> 는 쿠키의 최대 길이를 지정합니다.</li> <li>● <b>matchType</b> 은 쿠키의 필드 이름이 캡처 쿠키 설정과 정확히 일치하는지 또는 캡처 쿠키 설정의 접두사인지 지정합니다. <b>matchType</b> 필드는 <b>Exact</b> 및 <b>Prefix</b> 매개변수를 사용합니다.</li> </ul> <p>예를 들어 다음과 같습니다.</p> <pre>httpCaptureCookies: - matchType: Exact   maxLength: 128   name: MYCOOKIE</pre>
<p><b>httpCaptureHeaders</b></p>	<p><b>httpCapECDHEHeaders</b> 는 액세스 로그에서 캡처할 HTTP 헤더를 지정합니다. <b>httpCapECDHEHeaders</b> 필드가 비어 있으면 액세스 로그에서 헤더를 캡처하지 않습니다.</p> <p><b>httpCapECDHEHeaders</b> 에는 액세스 로그에서 캡처할 두 개의 헤더 목록이 포함되어 있습니다. 두 헤더 필드 목록은 <b>request</b> 및 <b>response</b> 입니다. 두 목록 모두에서 <b>name</b> 필드는 헤더 이름을 지정해야 하며 <b>maxlength</b> 필드는 헤더의 최대 길이를 지정해야 합니다. 예를 들어 다음과 같습니다.</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>
<p><b>tuningOptions</b></p>	<p><b>tuningOptions</b> 는 Ingress 컨트롤러 Pod의 성능을 조정하는 옵션을 지정합니다.</p> <ul style="list-style-type: none"> <li>● <b>clientFinTimeout</b> 은 연결을 닫는 서버에 대한 클라이언트 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 시간 제한은 <b>1s</b> 입니다.</li> <li>● <b>ClientTimeout</b> 은 클라이언트 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 <b>30s</b> 입니다.</li> <li>● <b>headerBufferBytes</b> 는 Ingress 컨트롤러 연결 세션에 대해 예약된 메모리 양을 바이트 단위로 지정합니다. Ingress 컨트롤러에 HTTP/2가 활성화된 경우 이 값은 <b>16384</b> 이상이어야 합니다. 설정되지 않은 경우, 기</li> </ul>

매개 변수	설명
	<p>본 값은 <b>32768</b> 바이트입니다. <b>headerBufferBytes</b> 값이 너무 작으면 Ingress 컨트롤러가 손상될 수 있으며 <b>headerBufferBytes</b> 값이 너무 크면 Ingress 컨트롤러가 필요 이상으로 많은 메모리를 사용할 수 있기 때문에 이 필드를 설정하지 않는 것이 좋습니다.</p> <ul style="list-style-type: none"> <li>● <b>headerBufferMaxRewriteBytes</b>는 HTTP 헤더 재작성 및 Ingress 컨트롤러 연결 세션에 대한 <b>headerBufferBytes</b>의 바이트 단위로 예약해야 하는 메모리 양을 지정합니다. <b>headerBufferMaxRewriteBytes</b>의 최소 값은 <b>4096</b>입니다. <b>headerBufferBytes</b>는 들어오는 HTTP 요청에 대해 <b>headerBufferMaxRewriteBytes</b>보다 커야 합니다. 설정되지 않은 경우, 기본값은 <b>8192</b> 바이트입니다. <b>headerBufferMaxRewriteBytes</b> 값이 너무 작으면 Ingress 컨트롤러가 손상될 수 있으며 <b>headerBufferMaxRewriteBytes</b> 값이 너무 크면 Ingress 컨트롤러가 필요 이상으로 많은 메모리를 사용할 수 있기 때문에 이 필드를 설정하지 않는 것이 좋습니다.</li> <li>● <b>healthCheckInterval</b> 은 라우터가 상태 점검 간에 대기하는 시간을 지정합니다. 기본값은 <b>5s</b>입니다.</li> <li>● <b>serverFinTimeout</b>은 연결을 종료하는 클라이언트에 대한 서버 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 시간 제한은 <b>1s</b>입니다.</li> <li>● <b>ServerTimeout</b>은 서버 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 <b>30s</b>입니다.</li> <li>● <b>threadCount</b>는 HAProxy 프로세스별로 생성할 스레드 수를 지정합니다. 더 많은 스레드를 생성하면 각 Ingress 컨트롤러 Pod가 더 많은 시스템 리소스 비용으로 더 많은 연결을 처리할 수 있습니다. HAProxy는 최대 <b>64</b>개의 스레드를 지원합니다. 이 필드가 비어 있으면 Ingress 컨트롤러는 기본값 <b>4</b> 스레드를 사용합니다. 기본값은 향후 릴리스에서 변경될 수 있습니다. HAProxy 스레드 수를 늘리면 Ingress 컨트롤러 Pod에서 부하에 더 많은 CPU 시간을 사용할 수 있고 다른 Pod에서 수행해야 하는 CPU 리소스를 수신하지 못하므로 이 필드를 설정하지 않는 것이 좋습니다. 스레드 수를 줄이면 Ingress 컨트롤러가 제대로 작동하지 않을 수 있습니다.</li> <li>● <b>tlsInspectDelay</b>는 라우터에서 일치하는 경로를 찾기 위해 데이터를 보유할 수 있는 시간을 지정합니다. 이 값을 너무 짧게 설정하면 일치하는 인증서를 사용하는 경우에도 라우터가 에지 종료, 재암호화 또는 패스스루 경로의 기본 인증서로 대체될 수 있습니다. 기본 검사 지연은 <b>5s</b>입니다.</li> <li>● <b>tunnelTimeout</b>은 터널이 유틸리티 상태일 때 웹소켓을 포함한 터널 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 <b>1h</b>입니다.</li> <li>● <b>maxConnections</b> 는 HAProxy 프로세스별로 설정할 수 있는 최대 동시 연결 수를 지정합니다. 이 값을 늘리면 각 Ingress 컨트롤러 Pod가 추가 시스템 리소스 비용으로 더 많은 연결을 처리할 수 있습니다. 허용되는 값은 <b>0, 1, 2000</b> 및 <b>2000000</b> 범위 내의 모든 값 또는 필드를 비워 둘 수 있습니다.             <ul style="list-style-type: none"> <li>○ 이 필드를 비워 두거나 값이 <b>0</b> 인 경우 Ingress 컨트롤러에서 기본 platform 을 <b>사용합니다</b>. 이 값은 향후 릴리스에서 변경될 수 있습니다.</li> <li>○ 필드에 값이 <b>-1</b> 인 경우 HAProxy는 실행 중인 컨테이너에서 사용 가능한 <b>ulimits</b> 에 따라 최대 값을 동적으로 계산합니다. 이 프로세스에서는 현재 기본 의 default value에 비해 상당한 메모리 사용량을 발생시키는 큰 계산된 값을 <b>생성합니다</b>.</li> <li>○ 필드에 현재 운영 체제 제한보다 큰 값이 있는 경우 HAProxy 프로세스가 시작되지 않습니다.</li> <li>○ 별도의 값을 선택하고 라우터 Pod가 새 노드로 마이그레이션되면 새 노드에 동일한 <b>ulimit</b> 가 구성되어 있지 않을 수 있습니다. 이러한 경우 Pod가 시작되지 않습니다.</li> </ul> </li> </ul>

매개변수	설명
	<ul style="list-style-type: none"> <li>다른 <b>ulimits</b> 가 구성된 노드가 있고 별도의 값을 선택하는 경우 런타임 시 최대 연결 수를 계산하도록 이 필드에 <b>-1</b> 값을 사용하는 것이 좋습니다.</li> </ul>
<b>logEmptyRequests</b>	<p><b>logEmptyRequests</b>는 요청이 수신 및 기록되지 않은 연결을 지정합니다. 이러한 빈 요청은 로드 밸런서 상태 프로브 또는 웹 브라우저 추측 연결(사전 연결)에서 발생하며 이러한 요청을 로깅하는 것은 바람직하지 않을 수 있습니다. 이러한 빈 요청은 로드 밸런서 상태 프로브 또는 웹 브라우저 추측 연결(사전 연결)에서 발생하며 이러한 요청을 로깅하는 것은 바람직하지 않을 수 있습니다. 이러한 요청은 포트 검색으로 인해 발생할 수 있으며 빈 요청을 로깅하면 침입 시도를 감지하는데 도움이 될 수 있습니다. 이 필드에 허용되는 값은 <b>Log</b> 및 <b>Ignore</b> 입니다. 기본값은 <b>Log</b> 입니다.</p> <p><b>LoggingPolicy</b> 유형은 다음 두 값 중 하나를 허용합니다.</p> <ul style="list-style-type: none"> <li>● <b>Log</b>: 이 값을 <b>Log</b>로 설정하면 이벤트가 로깅되어야 함을 나타냅니다.</li> <li>● <b>ignore</b>: 이 값을 <b>Ignore</b> 로 설정하면 HAProxy 구성에서 <b>dontlognull</b> 옵션이 설정됩니다.</li> </ul>
<b>HTTPEmptyRequestsPolicy</b>	<p><b>HTTPEmptyRequestsPolicy</b>는 요청을 수신하기 전에 연결 시간이 초과된 경우 HTTP 연결이 처리되는 방법을 설명합니다.. 이 필드에 허용되는 값은 <b>Respond</b> 및 <b>Ignore</b>입니다. 기본값은 <b>Respond</b>입니다.</p> <p><b>HTTPEmptyRequestsPolicy</b> 유형은 다음 두 값 중 하나를 허용합니다.</p> <ul style="list-style-type: none"> <li>● <b>Response</b>: 필드가 <b>Respond</b>로 설정된 경우 Ingress 컨트롤러는 HTTP <b>400</b> 또는 <b>408</b> 응답을 전송하고 액세스 로깅이 활성화된 경우 연결을 로깅한 다음 적절한 메트릭의 연결을 계산합니다.</li> <li>● <b>ignore</b>: 이 옵션을 <b>Ignore</b>로 설정하면 HAProxy 구성에 <b>http-ignore-probes</b> 매개변수가 추가됩니다. 필드가 <b>Ignore</b> 로 설정된 경우 Ingress 컨트롤러는 응답을 전송하지 않고 연결을 종료한 다음 연결을 기록하거나 메트릭을 늘립니다.</li> </ul> <p>이러한 연결은 로드 밸런서 상태 프로브 또는 웹 브라우저 추측 연결(preconnect)에서 제공되며 무시해도 됩니다. 그러나 이러한 요청은 네트워크 오류로 인해 발생할 수 있으므로 이 필드를 <b>Ignore</b>로 설정하면 문제를 탐지하고 진단할 수 있습니다. 이러한 요청은 포트 검색으로 인해 발생할 수 있으며, 이 경우 빈 요청을 로깅하면 침입 시도를 탐지하는데 도움이 될 수 있습니다.</p>



## 참고

모든 매개변수는 선택 사항입니다.

### 6.3.1. Ingress 컨트롤러 TLS 보안 프로파일



TLS 보안 프로파일은 서버가 서버에 연결할 때 연결 클라이언트가 사용할 수 있는 암호를 규제하는 방법을 제공합니다.

#### 6.3.1.1. TLS 보안 프로파일 이해

TLS(Transport Layer Security) 보안 프로필을 사용하여 다양한 OpenShift Container Platform 구성 요소에 필요한 TLS 암호를 정의할 수 있습니다. OpenShift Container Platform TLS 보안 프로필은 [Mozilla 권장 구성](#)을 기반으로 합니다.

각 구성 요소에 대해 다음 TLS 보안 프로필 중 하나를 지정할 수 있습니다.

표 6.1. TLS 보안 프로필

Profile	설명
<b>Old</b>	<p>이 프로필은 레거시 클라이언트 또는 라이브러리와 함께 사용하기 위한 것입니다. 프로필은 <a href="#">이전 버전과의 호환성</a> 권장 구성을 기반으로 합니다.</p> <p><b>Old</b> 프로파일에는 최소 TLS 버전 1.0이 필요합니다.</p> <div style="display: flex; align-items: center;">  <div> <p><b>참고</b></p> <p>Ingress 컨트롤러의 경우 최소 TLS 버전이 1.0에서 1.1로 변환됩니다.</p> </div> </div>
<b>Intermediate</b>	<p>이 프로필은 대부분의 클라이언트에서 권장되는 구성입니다. Ingress 컨트롤러, kubelet 및 컨트롤 플레인의 기본 TLS 보안 프로파일입니다. 프로필은 <a href="#">중간 호환성</a> 권장 구성을 기반으로 합니다.</p> <p><b>Intermediate</b> 프로파일에는 최소 TLS 버전이 1.2가 필요합니다.</p>
<b>Modern</b>	<p>이 프로필은 이전 버전과의 호환성이 필요하지 않은 최신 클라이언트와 사용하기 위한 것입니다. 이 프로필은 <a href="#">최신 호환성</a> 권장 구성을 기반으로 합니다.</p> <p><b>Modern</b> 프로파일에는 최소 TLS 버전 1.3이 필요합니다.</p>
<b>사용자 지정</b>	<p>이 프로필을 사용하면 사용할 TLS 버전과 암호를 정의할 수 있습니다.</p> <div style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;"> <div style="display: flex; align-items: center;">  <div> <p><b>주의</b></p> <p><b>Custom</b> 프로파일을 사용할 때는 잘못된 구성으로 인해 문제가 발생할 수 있으므로 주의해야 합니다.</p> </div> </div> </div>



**참고**

미리 정의된 프로파일 유형 중 하나를 사용하는 경우 유효한 프로파일 구성은 릴리스마다 변경될 수 있습니다. 예를 들어 릴리스 X.Y.Z에 배포된 중간 프로필을 사용하는 사양이 있는 경우 릴리스 X.Y.Z+1로 업그레이드하면 새 프로파일 구성이 적용되어 롤아웃이 발생할 수 있습니다.

6.3.1.2. Ingress 컨트롤러의 TLS 보안 프로필 구성

Ingress 컨트롤러에 대한 TLS 보안 프로필을 구성하려면 **IngressController** CR(사용자 정의 리소스)을 편집하여 사전 정의된 또는 사용자 지정 TLS 보안 프로필을 지정합니다. TLS 보안 프로필이 구성되지 않은 경우 기본값은 API 서버에 설정된 TLS 보안 프로필을 기반으로 합니다.

### Old TLS 보안 프로파일을 구성하는 샘플 IngressController CR

```
apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
...
```

TLS 보안 프로필은 Ingress 컨트롤러의 TLS 연결에 대한 최소 TLS 버전과 TLS 암호를 정의합니다.

**Status.Tls Profile** 아래의 **IngressController** CR(사용자 정의 리소스) 및 **Spec.Tls Security Profile** 아래 구성된 TLS 보안 프로필에서 구성된 TLS 보안 프로필의 암호 및 최소 TLS 버전을 확인할 수 있습니다. **Custom** TLS 보안 프로필의 경우 특정 암호 및 최소 TLS 버전이 두 매개변수 아래에 나열됩니다.



#### 참고

HAProxy Ingress 컨트롤러 이미지는 TLS **1.3** 및 **Modern** 프로필을 지원합니다.

Ingress Operator는 **Old** 또는 **Custom** 프로파일의 TLS **1.0**을 **1.1**로 변환합니다.

#### 사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 프로세스

1. **openshift-ingress-operator** 프로젝트에서 **IngressController** CR을 편집하여 TLS 보안 프로필을 구성합니다.

```
$ oc edit IngressController default -n openshift-ingress-operator
```

2. **spec.tlsSecurityProfile** 필드를 추가합니다.

#### Custom 프로필에 대한 IngressController CR 샘플

```
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 TLS 보안 프로파일 유형(**Old**, **Intermediate** 또는 **Custom**)을 지정합니다. 기본값은 **Intermediate**입니다.
- 2 선택한 유형의 적절한 필드를 지정합니다.
  - **old:** {}
  - **intermediate:** {}
  - **custom:**
- 3 **custom** 유형의 경우 TLS 암호화 목록 및 최소 허용된 TLS 버전을 지정합니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

검증

- **IngressController** CR에 프로파일이 설정되어 있는지 확인합니다.

```
$ oc describe IngressController default -n openshift-ingress-operator
```

출력 예

```
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
...
```

6.3.1.3. 상호 TLS 인증 구성

**spec.clientTLS** 값을 설정하여 mTLS(mTLS) 인증을 사용하도록 Ingress 컨트롤러를 구성할 수 있습니다. **clientTLS** 값은 클라이언트 인증서를 확인하도록 Ingress 컨트롤러를 구성합니다. 이 구성에는 구성 맵에 대한 참조인 **clientCA** 값 설정이 포함됩니다. 구성 맵에는 클라이언트의 인증서를 확인하는 데 사용되는 PEM 인코딩 CA 인증서 번들이 포함되어 있습니다. 필요한 경우 인증서 제목 필터 목록을 구성할 수 있습니다.

**clientCA** 값이 X509v3 인증서 취소 목록(CRL) 배포 지점을 지정하는 경우 Ingress Operator는 CRL을 다운로드하고 이를 승인하도록 Ingress 컨트롤러를 구성합니다. 유효한 인증서를 제공하지 않는 요청은 거부됩니다.

#### 사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 절차

1. **openshift-config** 네임스페이스에 있는 구성 맵을 생성합니다.

```
$ oc create configmap router-ca-certs-default --from-file=ca-bundle.pem=client-ca.crt -n openshift-config
```



#### 참고

구성 맵 데이터 키는 **ca-bundle.pem** 이어야 하며 데이터 값은 PEM 형식의 CA 인증서여야 합니다.

2. **openshift-ingress-operator** 프로젝트에서 **IngressController** 리소스를 편집합니다.

```
$ oc edit IngressController default -n openshift-ingress-operator
```

3. spec.clientTLS 필드 및 하위 필드를 추가하여 상호 TLS를 구성합니다.

#### 패턴 필터링을 지정하는 clientTLS 프로필에 대한 IngressController CR 샘플

```
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$"
```

## 6.4. 기본 INGRESS 컨트롤러 보기

Ingress Operator는 OpenShift Container Platform의 핵심 기능이며 즉시 사용이 가능합니다.

모든 새로운 OpenShift Container Platform 설치에는 이름이 **ingresscontroller**로 기본으로 지정됩니다. 추가 Ingress 컨트롤러를 추가할 수 있습니다. 기본 **ingresscontroller**가 삭제되면 Ingress Operator가 1분 이내에 자동으로 다시 생성합니다.

#### 프로세스

- 기본 Ingress 컨트롤러를 확인합니다.

■

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/default
```

## 6.5. INGRESS OPERATOR 상태 보기

Ingress Operator의 상태를 확인 및 조사할 수 있습니다.

### 프로세스

- Ingress Operator 상태를 확인합니다.

```
$ oc describe clusteroperators/ingress
```

## 6.6. INGRESS 컨트롤러 로그 보기

Ingress 컨트롤러의 로그를 확인할 수 있습니다.

### 프로세스

- Ingress 컨트롤러 로그를 확인합니다.

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator -c <container_name>
```

## 6.7. INGRESS 컨트롤러 상태 보기

특정 Ingress 컨트롤러의 상태를 확인할 수 있습니다.

### 프로세스

- Ingress 컨트롤러의 상태를 확인합니다.

```
$ oc describe --namespace=openshift-ingress-operator ingresscontroller/<name>
```

## 6.8. INGRESS 컨트롤러 구성

### 6.8.1. 사용자 정의 기본 인증서 설정

관리자는 Secret 리소스를 생성하고 **IngressController** CR(사용자 정의 리소스)을 편집하여 사용자 정의 인증서를 사용하도록 Ingress 컨트롤러를 구성할 수 있습니다.

#### 사전 요구 사항

- PEM 인코딩 파일에 인증서/키 쌍이 있어야 합니다. 이때 인증서는 신뢰할 수 있는 인증 기관 또는 사용자 정의 PKI에서 구성한 신뢰할 수 있는 개인 인증 기관의 서명을 받은 인증서입니다.
- 인증서가 다음 요구 사항을 충족합니다.
  - 인증서가 Ingress 도메인에 유효해야 합니다.
  - 인증서는 **subjectAltName** 확장자를 사용하여 **\*.apps.ocp4.example.com**과 같은 와일드카드 도메인을 지정합니다.



- **IngressController** CR이 있어야 합니다. 기본 설정을 사용할 수 있어야 합니다.

```
$ oc --namespace openshift-ingress-operator get ingresscontrollers
```

#### 출력 예

```
NAME    AGE
default 10m
```



#### 참고

임시 인증서가 있는 경우 사용자 정의 기본 인증서가 포함 된 보안의 **tls.crt** 파일에 인증서가 포함되어 있어야 합니다. 인증서를 지정하는 경우에는 순서가 중요합니다. 서버 인증서 다음에 임시 인증서를 나열해야 합니다.

#### 프로세스

아래에서는 사용자 정의 인증서 및 키 쌍이 현재 작업 디렉터리의 **tls.crt** 및 **tls.key** 파일에 있다고 가정합니다. 그리고 **tls.crt** 및 **tls.key**의 실제 경로 이름으로 변경합니다. Secret 리소스를 생성하고 IngressController CR에서 참조하는 경우 **custom-certs-default**를 다른 이름으로 변경할 수도 있습니다.



#### 참고

이 작업을 수행하면 롤링 배포 전략에 따라 Ingress 컨트롤러가 재배포됩니다.

1. **tls.crt** 및 **tls.key** 파일을 사용하여 **openshift-ingress** 네임스페이스에 사용자 정의 인증서를 포함하는 Secret 리소스를 만듭니다.

```
$ oc --namespace openshift-ingress create secret tls custom-certs-default --cert=tls.crt --key=tls.key
```

2. 새 인증서 보안 키를 참조하도록 IngressController CR을 업데이트합니다.

```
$ oc patch --type=merge --namespace openshift-ingress-operator ingresscontrollers/default \
--patch '{"spec":{"defaultCertificate":{"name":"custom-certs-default"}}}'
```

3. 업데이트가 적용되었는지 확인합니다.

```
$ echo Q |\
openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null |\
openssl x509 -noout -subject -issuer -enddate
```

다음과 같습니다.

#### <domain>

클러스터의 기본 도메인 이름을 지정합니다.

#### 출력 예

```
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
```

### 작은 정보

다음 YAML을 적용하여 사용자 지정 기본 인증서를 설정할 수 있습니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  defaultCertificate:
    name: custom-certs-default
```

인증서 보안 이름은 CR을 업데이트하는 데 사용된 값과 일치해야 합니다.

IngressController CR이 수정되면 Ingress Operator는 사용자 정의 인증서를 사용하도록 Ingress 컨트롤러의 배포를 업데이트합니다.

## 6.8.2. 사용자 정의 기본 인증서 제거

관리자는 사용할 Ingress 컨트롤러를 구성한 사용자 정의 인증서를 제거할 수 있습니다.

### 사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.
- 이전에 Ingress 컨트롤러에 대한 사용자 정의 기본 인증서를 구성했습니다.

### 프로세스

- 사용자 정의 인증서를 제거하고 OpenShift Container Platform과 함께 제공되는 인증서를 복원하려면 다음 명령을 입력합니다.

```
$ oc patch -n openshift-ingress-operator ingresscontrollers/default \
  --type json -p '$- op: remove\n path: /spec/defaultCertificate'
```

클러스터가 새 인증서 구성을 조정하는 동안 지연이 발생할 수 있습니다.

### 검증

- 원래 클러스터 인증서가 복원되었는지 확인하려면 다음 명령을 입력합니다.

```
$ echo Q | \
  openssl s_client -connect console-openshift-console.apps.<domain>:443 -showcerts
2>/dev/null | \
  openssl x509 -noout -subject -issuer -enddate
```

다음과 같습니다.

#### <domain>

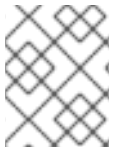
클러스터의 기본 도메인 이름을 지정합니다.

#### 출력 예

```
subject=CN = *.apps.<domain>
issuer=CN = ingress-operator@1620633373
notAfter=May 10 10:44:36 2023 GMT
```

### 6.8.3. Ingress 컨트롤러 확장

처리량 증가 요구 등 라우팅 성능 또는 가용성 요구 사항을 충족하도록 Ingress 컨트롤러를 수동으로 확장할 수 있습니다. **IngressController** 리소스를 확장하려면 **oc** 명령을 사용합니다. 다음 절차는 기본 **IngressController**를 확장하는 예제입니다.



#### 참고

원하는 수의 복제본을 만드는 데에는 시간이 걸리기 때문에 확장은 즉시 적용되지 않습니다.

#### 프로세스

1. 기본 **IngressController**의 현재 사용 가능한 복제본 개수를 살펴봅니다.

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

#### 출력 예

```
2
```

2. **oc patch** 명령을 사용하여 기본 **IngressController**의 복제본 수를 원하는 대로 조정합니다. 다음 예제는 기본 **IngressController**를 3개의 복제본으로 조정합니다.

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec":{"replicas":
3}}' --type=merge
```

#### 출력 예

```
ingresscontroller.operator.openshift.io/default patched
```

3. 기본 **IngressController**가 지정한 복제본 수에 맞게 조정되었는지 확인합니다.

```
$ oc get -n openshift-ingress-operator ingresscontrollers/default -o
jsonpath='{$.status.availableReplicas}'
```

#### 출력 예

```
3
```

## 작은 정보

또는 다음 YAML을 적용하여 Ingress 컨트롤러를 세 개의 복제본으로 확장할 수 있습니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3
```

- 1 다른 양의 복제본이 필요한 경우 **replicas** 값을 변경합니다.

### 6.8.4. 수신 액세스 로깅 구성

Ingress 컨트롤러가 로그에 액세스하도록 구성할 수 있습니다. 수신 트래픽이 많지 않은 클러스터의 경우 사이트카에 로그를 기록할 수 있습니다. 트래픽이 많은 클러스터가 있는 경우 로깅 스택의 용량을 초과하지 않거나 OpenShift Container Platform 외부의 로깅 인프라와 통합하기 위해 사용자 정의 syslog 끝점으로 로그를 전달할 수 있습니다. 액세스 로그의 형식을 지정할 수도 있습니다.

컨테이너 로깅은 기존 Syslog 로깅 인프라가 없는 경우 트래픽이 적은 클러스터에서 액세스 로그를 활성화하거나 Ingress 컨트롤러의 문제를 진단하는 동안 단기적으로 사용하는 데 유용합니다.

액세스 로그가 OpenShift 로깅 스택 용량을 초과할 수 있는 트래픽이 많은 클러스터 또는 로깅 솔루션이 기존 Syslog 로깅 인프라와 통합되어야 하는 환경에는 Syslog가 필요합니다. Syslog 사용 사례는 중첩될 수 있습니다.

#### 사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

사이트카에 Ingress 액세스 로깅을 구성합니다.

- 수신 액세스 로깅을 구성하려면 **spec.logging.access.destination**을 사용하여 대상을 지정해야 합니다. 사이트카 컨테이너에 로깅을 지정하려면 **Container** **spec.logging.access.destination.type**을 지정해야 합니다. 다음 예제는 **Container** 대상에 로그를 기록하는 Ingress 컨트롤러 정의입니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
```

- 사이드카에 로그를 기록하도록 Ingress 컨트롤러를 구성하면 Operator는 Ingress 컨트롤러 Pod에 **logs** 라는 컨테이너를 만듭니다.

```
$ oc -n openshift-ingress logs deployment.apps/router-default -c logs
```

#### 출력 예

```
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"
```

Syslog 끝점에 대한 Ingress 액세스 로깅을 구성합니다.

- 수신 액세스 로깅을 구성하려면 **spec.logging.access.destination**을 사용하여 대상을 지정해야 합니다. Syslog 끝점 대상에 로깅을 지정하려면 **spec.logging.access.destination.type**에 대한 **Syslog**를 지정해야 합니다. 대상 유형이 **Syslog**인 경우, **spec.logging.access.destination.syslog.endpoint**를 사용하여 대상 끝점을 지정해야 하며 **spec.logging.access.destination.syslog.facility**를 사용하여 장치를 지정할 수 있습니다. 다음 예제는 **Syslog** 대상에 로그를 기록하는 Ingress 컨트롤러 정의입니다.

```
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
```



#### 참고

**syslog** 대상 포트는 UDP여야 합니다.

특정 로그 형식으로 Ingress 액세스 로깅을 구성합니다.

- **spec.logging.access.httpLogFormat**을 지정하여 로그 형식을 사용자 정의할 수 있습니다. 다음 예제는 IP 주소 1.2.3.4 및 포트 10514를 사용하여 **syslog** 끝점에 로그하는 Ingress 컨트롤러 정의입니다.

```
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'

```

Ingress 액세스 로깅을 비활성화합니다.

- Ingress 액세스 로깅을 비활성화하려면 **spec.logging** 또는 **spec.logging.access**를 비워 둡니다.

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access: null

```

### 6.8.5. Ingress 컨트롤러 스레드 수 설정

클러스터 관리자는 클러스터에서 처리할 수 있는 들어오는 연결의 양을 늘리기 위해 스레드 수를 설정할 수 있습니다. 기존 Ingress 컨트롤러에 패치하여 스레드의 양을 늘릴 수 있습니다.

#### 사전 요구 사항

- 다음은 Ingress 컨트롤러를 이미 생성했다고 가정합니다.

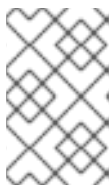
#### 프로세스

- 스레드 수를 늘리도록 Ingress 컨트롤러를 업데이트합니다.

```

$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec": {"tuningOptions": {"threadCount": 8}}}'

```



#### 참고

많은 리소스를 실행할 수 있는 노드가 있는 경우 원하는 노드의 용량과 일치하는 라벨을 사용하여 **spec.nodePlacement.nodeSelector**를 구성하고 **spec.tuningOptions.threadCount**를 적절하게 높은 값으로 구성할 수 있습니다.

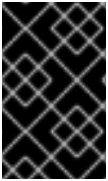
### 6.8.6. 내부 로드 밸런서를 사용하도록 Ingress 컨트롤러 구성

클라우드 플랫폼에서 Ingress 컨트롤러를 생성할 때 Ingress 컨트롤러는 기본적으로 퍼블릭 클라우드 로드 밸런서에 의해 게시됩니다. 관리자는 내부 클라우드 로드 밸런서를 사용하는 Ingress 컨트롤러를 생성할 수 있습니다.



### 주의

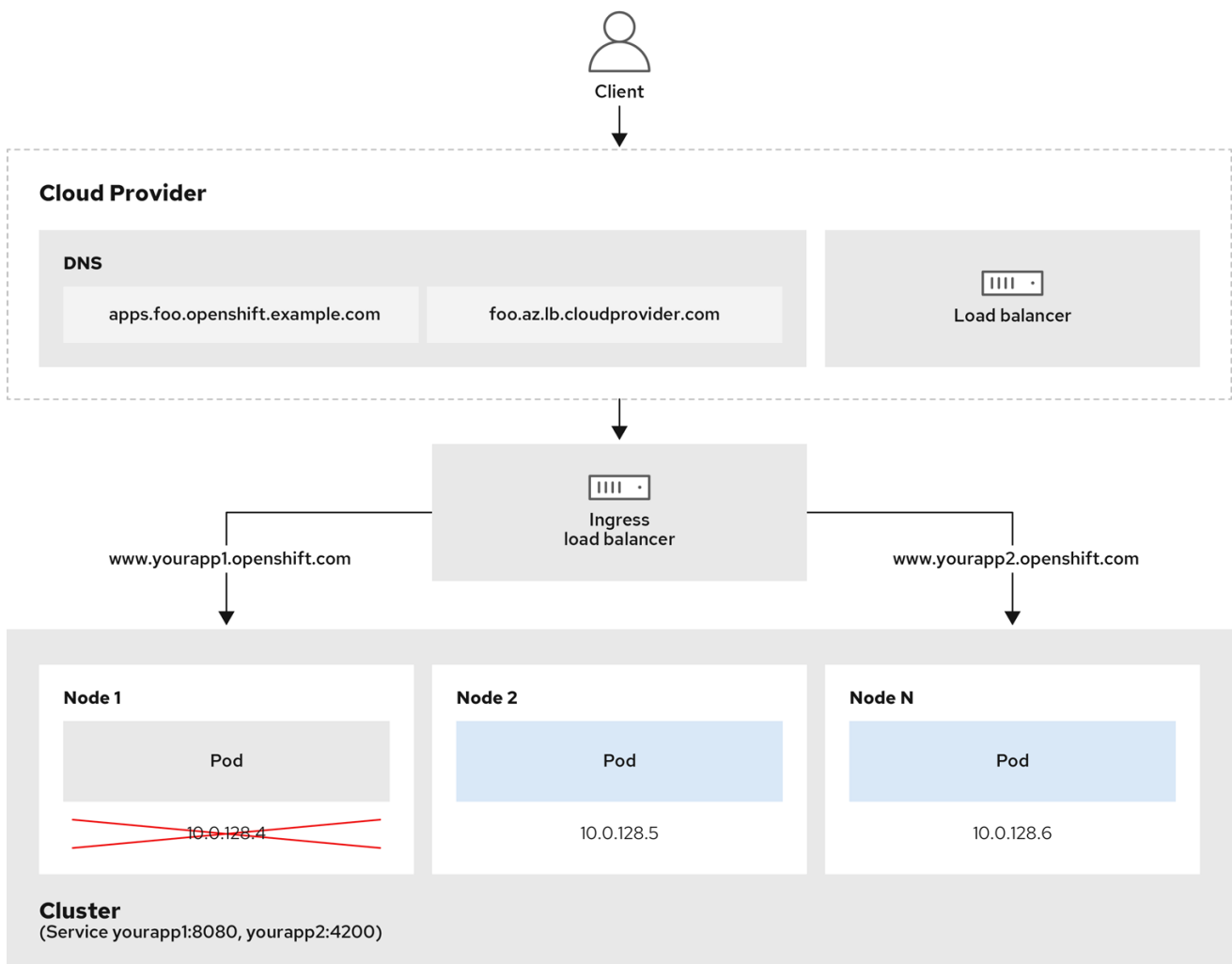
클라우드 공급자가 Microsoft Azure인 경우 노드를 가리키는 퍼블릭 로드 밸런서가 하나 이상 있어야 합니다. 그렇지 않으면 모든 노드의 인터넷 연결이 끊어집니다.



### 중요

**IngressController**의 범위를 변경하려면 CR(사용자 정의 리소스)이 생성된 후 **.spec.endpointPublishingStrategy.loadBalancer.scope** 매개변수를 변경할 수 있습니다.

그림 6.1. LoadBalancer 다이어그램



202\_OpenShift\_0222

이전 그래픽에서는 OpenShift Container Platform Ingress LoadBalancerService 끝점 게시 전략에 대한 다음 개념을 보여줍니다.

- OpenShift Ingress 컨트롤러 로드 밸런서를 사용하여 클라우드 공급자 로드 밸런서 또는 내부적으로 로드 밸런싱을 외부적으로 로드할 수 있습니다.

- 그래픽에 표시된 것처럼 로드 밸런서의 단일 IP 주소 및 더 친숙한 포트(예: 8080 및 4200)를 사용할 수 있습니다.
- 외부 로드 밸런서의 트래픽은 Pod에서 지시하며 down 노드의 인스턴스에 표시된 대로 로드 밸런서에서 관리합니다. 구현 세부 사항은 [Kubernetes 서비스 설명서](#)를 참조하십시오.

#### 사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

1. 다음 예제와 같이 **<name>-ingress-controller.yam** 파일에 **IngressController** CR(사용자 정의 리소스)을 생성합니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> ❶
spec:
  domain: <domain> ❷
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: Internal ❸
```

- ❶ **<name>**을 **IngressController** 오브젝트의 이름으로 변경합니다.
- ❷ 컨트롤러가 게시한 애플리케이션의 **domain**을 지정합니다.
- ❸ 내부 로드 밸런서를 사용하려면 **Internal** 값을 지정합니다.

2. 다음 명령을 실행하여 이전 단계에서 정의된 Ingress 컨트롤러를 생성합니다.

```
$ oc create -f <name>-ingress-controller.yaml ❶
```

- ❶ **<name>**을 **IngressController** 오브젝트의 이름으로 변경합니다.

3. 선택 사항: Ingress 컨트롤러가 생성되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc --all-namespaces=true get ingresscontrollers
```

#### 6.8.7. GCP에서 Ingress 컨트롤러에 대한 글로벌 액세스 구성

내부 로드 밸런서가 있는 GCP에서 생성된 Ingress 컨트롤러는 서비스의 내부 IP 주소를 생성합니다. 클러스터 관리자는 로드 밸런서와 동일한 VPC 네트워크 및 컴퓨팅 리전 내의 모든 리전의 클라이언트가 클러스터에서 실행되는 워크로드에 도달할 수 있도록 하는 글로벌 액세스 옵션을 지정할 수 있습니다.

자세한 내용은 [글로벌 액세스](#)에 대한 GCP 설명서를 참조하십시오.



## 사전 요구 사항

- GCP 인프라에 OpenShift Container Platform 클러스터를 배포했습니다.
- 내부 로드 밸런서를 사용하도록 Ingress 컨트롤러 구성
- OpenShift CLI(**oc**)를 설치합니다.

## 프로세스

1. 글로벌 액세스를 허용하도록 Ingress 컨트롤러 리소스를 구성합니다.



### 참고

Ingress 컨트롤러를 생성하고 글로벌 액세스 옵션을 지정할 수도 있습니다.

- a. Ingress 컨트롤러 리소스를 구성합니다.

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

- b. YAML 파일을 편집합니다.

### Global에 대한 clientAccess 구성 샘플

```
spec:
  endpointPublishingStrategy:
    loadBalancer:
      providerParameters:
        gcp:
          clientAccess: Global 1
          type: GCP
        scope: Internal
        type: LoadBalancerService
```

- 1** **gcp.clientAccess**를 **Global**로 설정합니다.

- c. 파일을 저장하여 변경 사항을 적용합니다.

2. 다음 명령을 실행하여 서비스가 글로벌 액세스를 허용하는지 확인합니다.

```
$ oc -n openshift-ingress edit svc/router-default -o yaml
```

출력에서 주석 **networking.gke.io/internal-load-balancer-allow-global-access**가 있는 GCP에 글로벌 액세스가 활성화되어 있음을 보여줍니다.

## 6.8.8. Ingress 컨트롤러 상태 점검 간격 설정

클러스터 관리자는 상태 점검 간격을 설정하여 두 번 연속된 상태 점검 간에 라우터가 대기하는 시간을 정의할 수 있습니다. 이 값은 모든 경로에 대한 기본값으로 전역적으로 적용됩니다. 기본값은 5초입니다.

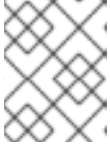
## 사전 요구 사항

- 다음은 Ingress 컨트롤러를 이미 생성했다고 가정합니다.

프로세스

- 백엔드 상태 점검 간격을 변경하도록 Ingress 컨트롤러를 업데이트합니다.

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec": {"tuningOptions": {"healthCheckInterval": "8s"}}}'
```



참고

단일 경로에 대해 **healthCheckInterval** 을 재정의하려면 경로 주석 **router.openshift.io/haproxy.health.check.interval** 을 사용하십시오.

6.8.9. 클러스터의 기본 Ingress 컨트롤러를 내부로 구성

클러스터를 삭제하고 다시 생성하여 클러스터의 **default** Ingress 컨트롤러를 내부용으로 구성할 수 있습니다.



주의

클라우드 공급자가 Microsoft Azure인 경우 노드를 가리키는 퍼블릭 로드 밸런서가 하나 이상 있어야 합니다. 그렇지 않으면 모든 노드의 인터넷 연결이 끊어집니다.



중요

**IngressController** 의 범위를 변경하려면 CR(사용자 정의 리소스)이 생성된 후 **.spec.endpointPublishingStrategy.loadBalancer.scope** 매개변수를 변경할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

프로세스

1. 클러스터의 기본 Ingress 컨트롤러를 삭제하고 다시 생성하여 내부용으로 구성합니다.

```
$ 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
```

### 6.8.10. 경로 허용 정책 구성

관리자 및 애플리케이션 개발자는 도메인 이름이 동일한 여러 네임스페이스에서 애플리케이션을 실행할 수 있습니다. 이는 여러 팀이 동일한 호스트 이름에 노출되는 마이크로 서비스를 개발하는 조직을 위한 것입니다.



#### 주의

네임스페이스 간 클레임은 네임스페이스 간 신뢰가 있는 클러스터에 대해서만 허용해야 합니다. 그렇지 않으면 악의적인 사용자가 호스트 이름을 인수할 수 있습니다. 따라서 기본 승인 정책에서는 네임스페이스 간에 호스트 이름 클레임을 허용하지 않습니다.

#### 사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.

#### 프로세스

- 다음 명령을 사용하여 **ingresscontroller** 리소스 변수의 **.spec.routeAdmission** 필드를 편집합니다.

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec": {"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --type=merge
```

#### 샘플 Ingress 컨트롤러 구성

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
  ...
```

#### 작은 정보

다음 YAML을 적용하여 경로 승인 정책을 구성할 수 있습니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

### 6.8.11. 와일드카드 경로 사용

HAProxy Ingress 컨트롤러는 와일드카드 경로를 지원합니다. Ingress Operator는 **wildcardPolicy**를 사용하여 Ingress 컨트롤러의 **ROUTER\_ALLOW\_WILDCARD\_ROUTES** 환경 변수를 구성합니다.

Ingress 컨트롤러의 기본 동작은 와일드카드 정책이 **None**인 경로를 허용하고, 이는 기존 **IngressController** 리소스의 이전 버전과 호환됩니다.

#### 프로세스

1. 와일드카드 정책을 구성합니다.

- a. 다음 명령을 사용하여 **IngressController** 리소스를 편집합니다.

```
$ oc edit IngressController
```

- b. **spec**에서 **wildcardPolicy** 필드를 **WildcardsDisallowed** 또는 **WildcardsAllowed**로 설정합니다.

```
spec:
  routeAdmission:
    wildcardPolicy: WildcardsDisallowed # or WildcardsAllowed
```

### 6.8.12. X-Forwarded 헤더 사용

HAProxy Ingress 컨트롤러를 구성하여 **Forwarded** 및 **X-Forwarded-For**를 포함한 HTTP 헤더 처리 방법에 대한 정책을 지정합니다. Ingress Operator는 **HTTPHeaders** 필드를 사용하여 Ingress 컨트롤러의 **ROUTER\_SET\_FORWARDED\_HEADERS** 환경 변수를 구성합니다.

#### 프로세스

1. Ingress 컨트롤러에 대한 **HTTPHeaders** 필드를 구성합니다.

- a. 다음 명령을 사용하여 **IngressController** 리소스를 편집합니다.

```
$ oc edit IngressController
```

- b. **spec**에서 **HTTPHeaders** 정책 필드를 **Append, Replace, IfNone** 또는 **Never**로 설정합니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    forwardedHeaderPolicy: Append
```

#### 사용 사례 예

클러스터 관리자는 다음을 수행할 수 있습니다.

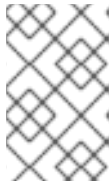
- Ingress 컨트롤러로 전달하기 전에 **X-Forwarded-For** 헤더를 각 요청에 삽입하는 외부 프록시를 구성합니다.

헤더를 수정하지 않은 상태로 전달하도록 Ingress 컨트롤러를 구성하려면 **never** 정책을 지정합니다. 그러면 Ingress 컨트롤러에서 헤더를 설정하지 않으며 애플리케이션은 외부 프록시에서 제공하는 헤더만 수신합니다.

- 외부 프록시에서 외부 클러스터 요청에 설정한 **X-Forwarded-For** 헤더를 수정하지 않은 상태로 전달하도록 Ingress 컨트롤러를 구성합니다.  
외부 프록시를 통과하지 않는 내부 클러스터 요청에 **X-Forwarded-For** 헤더를 설정하도록 Ingress 컨트롤러를 구성하려면 **if-none** 정책을 지정합니다. HTTP 요청에 이미 외부 프록시를 통해 설정된 헤더가 있는 경우 Ingress 컨트롤러에서 해당 헤더를 보존합니다. 요청이 프록시를 통해 제공되지 않아 헤더가 없는 경우에는 Ingress 컨트롤러에서 헤더를 추가합니다.

애플리케이션 개발자는 다음을 수행할 수 있습니다.

- **X-Forwarded-For** 헤더를 삽입하는 애플리케이션별 외부 프록시를 구성합니다.  
다른 경로에 대한 정책에 영향을 주지 않으면서 애플리케이션 경로에 대한 헤더를 수정하지 않은 상태로 전달하도록 Ingress 컨트롤러를 구성하려면 애플리케이션 경로에 주석 **haproxy.router.openshift.io/set-forwarded-headers: if-none** 또는 **haproxy.router.openshift.io/set-forwarded-headers: never**를 추가하십시오.



#### 참고

Ingress 컨트롤러에 전역적으로 설정된 값과 관계없이 경로별로 **haproxy.router.openshift.io/set-forwarded-headers** 주석을 설정할 수 있습니다.

### 6.8.13. HTTP/2 수신 연결 사용

이제 HAProxy에서 투명한 엔드 투 엔드 HTTP/2 연결을 활성화할 수 있습니다. 애플리케이션 소유자는 이를 통해 단일 연결, 헤더 압축, 바이너리 스트림 등 HTTP/2 프로토콜 기능을 활용할 수 있습니다.

개별 Ingress 컨트롤러 또는 전체 클러스터에 대해 HAProxy에서 HTTP/2 연결을 활성화할 수 있습니다.

클라이언트에서 HAProxy로의 연결에 HTTP/2 사용을 활성화하려면 경로에서 사용자 정의 인증서를 지정해야 합니다. 기본 인증서를 사용하는 경로에서는 HTTP/2를 사용할 수 없습니다. 이것은 동일한 인증서를 사용하는 다른 경로의 연결을 클라이언트가 재사용하는 등 동시 연결로 인한 문제를 방지하기 위한 제한입니다.

HAProxy에서 애플리케이션 pod로의 연결은 re-encrypt 라우팅에만 HTTP/2를 사용할 수 있으며 Edge termination 또는 비보안 라우팅에는 사용할 수 없습니다. 이 제한은 백엔드와 HTTP/2 사용을 협상할 때 HAProxy가 TLS의 확장인 ALPN(Application-Level Protocol Negotiation)을 사용하기 때문에 필요합니다. 이는 엔드 투 엔드 HTTP/2가 패스스루(passthrough) 및 re-encrypt 라우팅에는 적합하지만 비보안 또는 Edge termination 라우팅에는 적합하지 않음을 의미합니다.



#### 주의

Ingress 컨트롤러에서 재암호화 경로와 HTTP/2가 활성화된 WebSockets를 사용하려면 HTTP/2를 통해 WebSocket 지원이 필요합니다. WebSockets over HTTP/2는 현재 OpenShift Container Platform에서 지원되지 않는 HAProxy 2.4의 기능입니다.



## 중요

패스스루(passthrough)가 아닌 경로의 경우 Ingress 컨트롤러는 클라이언트와의 연결과 관계없이 애플리케이션에 대한 연결을 협상합니다. 다시 말해 클라이언트가 Ingress 컨트롤러에 연결하여 HTTP/1.1을 협상하고, Ingress 컨트롤러가 애플리케이션에 연결하여 HTTP/2를 협상하고, 클라이언트 HTTP/1.1 연결에서 받은 요청을 HTTP/2 연결을 사용하여 애플리케이션에 전달할 수 있습니다. Ingress 컨트롤러는 WebSocket을 HTTP/2로 전달할 수 없고 HTTP/2 연결을 WebSocket으로 업그레이드할 수 없기 때문에 나중에 클라이언트가 HTTP/1.1 연결을 WebSocket 프로토콜로 업그레이드하려고 하면 문제가 발생하게 됩니다. 결과적으로, WebSocket 연결을 허용하는 애플리케이션이 있는 경우 HTTP/2 프로토콜 협상을 허용하지 않아야 합니다. 그러지 않으면 클라이언트가 WebSocket 프로토콜로 업그레이드할 수 없게 됩니다.

## 프로세스

단일 Ingress 컨트롤러에서 HTTP/2를 활성화합니다.

- Ingress 컨트롤러에서 HTTP/2를 사용하려면 다음과 같이 **oc annotate** 명령을 입력합니다.

```
$ oc -n openshift-ingress-operator annotate ingresscontrollers/<ingresscontroller_name>
ingress.operator.openshift.io/default-enable-http2=true
```

**<ingresscontroller\_name>**을 주석 처리할 Ingress 컨트롤러의 이름으로 변경합니다.

전체 클러스터에서 HTTP/2를 활성화합니다.

- 전체 클러스터에 HTTP/2를 사용하려면 **oc annotate** 명령을 입력합니다.

```
$ oc annotate ingresses.config/cluster ingress.operator.openshift.io/default-enable-http2=true
```

## 작은 정보

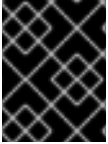
다음 YAML을 적용하여 주석을 추가할 수도 있습니다.

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
annotations:
  ingress.operator.openshift.io/default-enable-http2: "true"
```

### 6.8.14. Ingress 컨트롤러에 대한 PROXY 프로토콜 구성

클러스터 관리자는 Ingress 컨트롤러에서 **HostNetwork** 또는 **NodePortService** 엔드포인트 게시 전략 유형을 사용하는 경우 **PROXY 프로토콜**을 구성할 수 있습니다. PROXY 프로토콜을 사용하면 로드 밸런서에서 Ingress 컨트롤러가 수신하는 연결에 대한 원래 클라이언트 주소를 유지할 수 있습니다. 원래 클라이언트 주소는 HTTP 헤더를 로깅, 필터링 및 삽입하는 데 유용합니다. 기본 구성에서 Ingress 컨트롤러가 수신하는 연결에는 로드 밸런서와 연결된 소스 주소만 포함됩니다.

이 기능은 클라우드 배포에서 지원되지 않습니다. 이 제한 사항은 OpenShift Container Platform이 클라우드 플랫폼에서 실행되고 IngressController에서 서비스 로드 밸런서를 사용해야 함을 지정하기 때문에 Ingress Operator는 로드 밸런서 서비스를 구성하고 소스 주소를 유지하기 위한 플랫폼 요구 사항에 따라 PROXY 프로토콜을 활성화하기 때문입니다.



### 중요

PROXY 프로토콜을 사용하거나 TCP를 사용하려면 OpenShift Container Platform과 외부 로드 밸런서를 모두 구성해야 합니다.



### 주의

PROXY 프로토콜은 Keepalived Ingress VIP를 사용하는 비클라우드 플랫폼에 설치 관리자 프로비저닝 클러스터가 있는 기본 Ingress 컨트롤러에 지원되지 않습니다.

### 사전 요구 사항

- Ingress 컨트롤러가 생성되어 있습니다.

### 프로세스

1. Ingress 컨트롤러 리소스를 편집합니다.

```
$ oc -n openshift-ingress-operator edit ingresscontroller/default
```

2. PROXY 구성을 설정합니다.

- Ingress 컨트롤러에서 hostNetwork 엔드포인트 게시 전략 유형을 사용하는 경우 **spec.endpointPublishingStrategy.hostNetwork.protocol** 하위 필드를 **PROXY**로 설정합니다.

#### PROXY에 대한 hostNetwork 구성 샘플

```
spec:
  endpointPublishingStrategy:
    hostNetwork:
      protocol: PROXY
      type: HostNetwork
```

- Ingress 컨트롤러에서 NodePortService 엔드포인트 게시 전략 유형을 사용하는 경우 **spec.endpointPublishingStrategy.nodePort.protocol** 하위 필드를 **PROXY**로 설정합니다.

#### PROXY에 대한 nodePort 구성 샘플

```
spec:
  endpointPublishingStrategy:
    nodePort:
      protocol: PROXY
      type: NodePortService
```

### 6.8.15. appsDomain 옵션을 사용하여 대체 클러스터 도메인 지정

클러스터 관리자는 **appsDomain** 필드를 구성하여 사용자가 생성한 경로의 기본 클러스터 도메인에 대한 대안을 지정할 수 있습니다. **appsDomain** 필드는 **domain** 필드에 지정된 기본값 대신 사용할 OpenShift

Container Platform의 선택적 **도메인**입니다. 대체 도메인을 지정하면 새 경로의 기본 호스트를 결정하기 위해 기본 클러스터 도메인을 덮어씁니다.

예를 들어, 회사의 DNS 도메인을 클러스터에서 실행되는 애플리케이션의 경로 및 인그레스의 기본 도메인으로 사용할 수 있습니다.

**사전 요구 사항**

- OpenShift Container Platform 클러스터를 배포했습니다.
- **oc** 명령줄 인터페이스를 설치했습니다.

**프로세스**

1. 사용자 생성 경로에 대한 대체 기본 도메인을 지정하여 **appsDomain** 필드를 구성합니다.
  - a. Ingress 클러스터 리소스를 편집합니다.

```
$ oc edit ingresses.config/cluster -o yaml
```

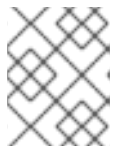
- b. YAML 파일을 편집합니다.

**test.example.com에 대한 샘플 appsDomain 구성**

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: apps.example.com
  appsDomain: <test.example.com>
```

- 1 기본 도메인을 지정합니다. 설치 후에는 기본 도메인을 수정할 수 없습니다.
- 2 선택사항: 애플리케이션 경로에 사용할 OpenShift Container Platform 인프라의 도메인입니다. **앱**의 기본 접두사 대신 **test**와 같은 대체 접두사를 사용할 수 있습니다.

2. 경로를 노출하고 경로 도메인 변경을 확인하여 기존 경로에 **appsDomain** 필드에 지정된 도메인 이름이 포함되어 있는지 확인합니다.



**참고**

경로를 노출하기 전에 **openshift-apiserver**가 롤링 업데이트를 완료할 때까지 기다립니다.

- a. 경로를 노출합니다.

```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

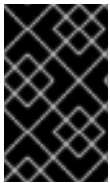
출력 예:



```
$ oc get routes
NAME          HOST/PORT          PATH SERVICES          PORT
TERMINATION  WILDCARD
hello-openshift hello_openshift-<my_project>.test.example.com
hello-openshift 8080-tcp          None
```

### 6.8.16. HTTP 헤더 대소문자 변환

HAProxy 2.2는 기본적으로 HTTP 헤더 이름을 소문자로 (예: **Host: xyz.com**을 **host: xyz.com**으로) 변경합니다. 기존 애플리케이션이 HTTP 헤더 이름의 대문자에 민감한 경우 Ingress Controller **spec.httpHeaders.headerNameCaseAdjustments** API 필드를 사용하여 기존 애플리케이션을 수정할 때 까지 지원합니다.



#### 중요

OpenShift Container Platform에는 HAProxy 2.2가 포함되어 있으므로 업그레이드하기 전에 **spec.httpHeaders.headerNameCaseAdjustments** 를 사용하여 필요한 구성을 추가하십시오.

#### 사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 프로세스

클러스터 관리자는 **oc patch** 명령을 입력하거나 Ingress 컨트롤러 YAML 파일에서 **HeaderNameCaseAdjustments** 필드를 설정하여 HTTP 헤더 케이스를 변환할 수 있습니다.

- **oc patch** 명령을 입력하여 대문자로 작성할 HTTP 헤더를 지정합니다.
  1. **oc patch** 명령을 입력하여 HTTP **host** 헤더를 **Host**로 변경합니다.

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{\"spec\":{\"httpHeaders\":{\"headerNameCaseAdjustments\":{\"Host\"}}}'
```

2. 애플리케이션 경로에 주석을 추가합니다.

```
$ oc annotate routes/my-application haproxy.router.openshift.io/h1-adjust-case=true
```

그런 다음 Ingress 컨트롤러는 지정된 대로 **host** 요청 헤더를 조정합니다.

- Ingress 컨트롤러 YAML 파일을 구성하여 **HeaderNameCaseAdjustments** 필드를 사용하여 조정합니다.
  1. 다음 예제 Ingress 컨트롤러 YAML은 적절하게 주석이 달린 경로의 HTTP/1 요청에 대해 **host** 헤더를 **Host**로 조정합니다.

#### Ingress 컨트롤러 YAML 예시

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
```

```
namespace: openshift-ingress-operator
spec:
  httpHeaders:
    headerNameCaseAdjustments:
      - Host
```

- 다음 예제 경로에서는 **haproxy.router.openshift.io/h1-adjust-case** 주석을 사용하여 HTTP 응답 헤더 이름 대소문자 조정을 활성화합니다.

경로 YAML의 예

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true ❶
  name: my-application
  namespace: my-application
spec:
  to:
    kind: Service
    name: my-application
```

❶ **haproxy.router.openshift.io/h1-adjust-case**를 true로 설정합니다.

### 6.8.17. 라우터 압축 사용

특정 MIME 유형에 대해 전역적으로 라우터 압축을 지정하도록 HAProxy Ingress 컨트롤러를 구성합니다. **mimeTypes** 변수를 사용하여 압축이 적용되는 MIME 유형의 형식을 정의할 수 있습니다. 유형은 애플리케이션, 이미지, 메시지, 다중 파트, 텍스트, 비디오 또는 "X-"로 선행되는 사용자 정의 유형입니다. MIME 유형 및 하위 유형에 대한 전체 표기법을 보려면 [RFC1341](#) 을 참조하십시오.



참고

압축을 위해 할당된 메모리는 최대 연결에 영향을 미칠 수 있습니다. 또한 대규모 버퍼 압축으로 인해 regex 또는 긴 regex 목록과 같은 대기 시간이 발생할 수 있습니다.

모든 MIME 유형이 압축의 이점을 제공하는 것은 아니지만, HAProxy는 여전히 리소스를 사용하여 예 지시된 경우 압축하려고 합니다. 일반적으로 html, css 및 js와 같은 텍스트 형식은 압축의 이점을 활용하지만 이미지, 오디오 및 비디오와 같은 형식이 압축되는 시간과 리소스를 교환하는 것은 거의 없습니다.

절차

- Ingress 컨트롤러의 **httpCompression** 필드를 구성합니다.
  - 다음 명령을 사용하여 **IngressController** 리소스를 편집합니다.

```
$ oc edit -n openshift-ingress-operator ingresscontrollers/default
```

- 사양에서 **httpCompression** 정책 필드를 **mimeTypes** 로 설정하고 압축이 적용되어야 하는 MIME 유형 목록을 지정합니다.

```

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"
    ...

```

### 6.8.18. HAProxy 오류 코드 응답 페이지 사용자 정의

클러스터 관리자는 503, 404 또는 두 오류 페이지에 대한 사용자 지정 오류 코드 응답 페이지를 지정할 수 있습니다. HAProxy 라우터는 애플리케이션 pod가 실행 중이 아닌 경우 503 오류 페이지 또는 요청된 URL이 없는 경우 404 오류 페이지를 제공합니다. 예를 들어 503 오류 코드 응답 페이지를 사용자 지정하면 애플리케이션 pod가 실행되지 않을 때 페이지가 제공되며 HAProxy 라우터에서 잘못된 경로 또는 존재하지 않는 경로에 대해 기본 404 오류 코드 HTTP 응답 페이지가 제공됩니다.

사용자 정의 오류 코드 응답 페이지가 구성 맵에 지정되고 Ingress 컨트롤러에 패치됩니다. 구성 맵 키의 사용 가능한 파일 이름은 **error-page-503.http** 및 **error-page-404.http** 입니다.

사용자 지정 HTTP 오류 코드 응답 페이지는 [HAProxy HTTP 오류 페이지 구성 지침](#) 을 따라야 합니다. 다음은 기본 OpenShift Container Platform HAProxy 라우터 [http 503 오류 코드 응답 페이지](#) 의 예입니다. 기본 콘텐츠를 고유한 사용자 지정 페이지를 생성하기 위한 템플릿으로 사용할 수 있습니다.

기본적으로 HAProxy 라우터는 애플리케이션이 실행 중이 아니거나 경로가 올바르지 않거나 존재하지 않는 경우 503 오류 페이지만 제공합니다. 이 기본 동작은 OpenShift Container Platform 4.8 및 이전 버전의 동작과 동일합니다. HTTP 오류 코드 응답 사용자 정의에 대한 구성 맵이 제공되지 않고 사용자 정의 HTTP 오류 코드 응답 페이지를 사용하는 경우 라우터는 기본 404 또는 503 오류 코드 응답 페이지를 제공합니다.



#### 참고

OpenShift Container Platform 기본 503 오류 코드 페이지를 사용자 정의를 위한 템플릿으로 사용하는 경우 파일의 헤더에 CRLF 줄 종료를 사용할 수 있는 편집기가 필요합니다.

#### 절차

1. **openshift-config** 네임스페이스에 **my-custom-error-code-pages** 라는 구성 맵을 생성합니다.

```

$ oc -n openshift-config create configmap my-custom-error-code-pages \
--from-file=error-page-503.http \
--from-file=error-page-404.http

```



#### 중요

사용자 정의 오류 코드 응답 페이지에 올바른 형식을 지정하지 않으면 라우터 Pod 중단이 발생합니다. 이 중단을 해결하려면 구성 맵을 삭제하거나 수정하고 영향을 받는 라우터 Pod를 삭제하여 올바른 정보로 다시 생성해야 합니다.

- 이름별로 **my-custom-error-code-pages** 구성 맵을 참조하도록 Ingress 컨트롤러를 패치합니다.

```
$ oc patch -n openshift-ingress-operator ingresscontroller/default --patch '{"spec": {"httpErrorPages":{"name":"my-custom-error-code-pages"}}}' --type=merge
```

Ingress Operator는 **my-custom-error-code-pages** 구성 맵을 **openshift-config** 네임스페이스에서 **openshift-ingress** 네임스페이스로 복사합니다. Operator는 **openshift-ingress** 네임스페이스에서 **<your\_ingresscontroller\_name>-errorpages** 패턴에 따라 구성 맵의 이름을 지정합니다.

- 복사본을 표시합니다.

```
$ oc get cm default-errorpages -n openshift-ingress
```

#### 출력 예

NAME	DATA	AGE
default-errorpages	2	25s <b>1</b>

- 1** **default** Ingress 컨트롤러 CR(사용자 정의 리소스)이 패치되었기 때문에 구성 맵 이름은 **default-errorpages**입니다.

- 사용자 정의 오류 응답 페이지가 포함된 구성 맵이 라우터 볼륨에 마운트되는지 확인합니다. 여기서 구성 맵 키는 사용자 정의 HTTP 오류 코드 응답이 있는 파일 이름입니다.

- 503 사용자 지정 HTTP 사용자 정의 오류 코드 응답의 경우:

```
$ oc -n openshift-ingress rsh <router_pod> cat /var/lib/haproxy/conf/error_code_pages/error-page-503.http
```

- 404 사용자 지정 HTTP 사용자 정의 오류 코드 응답의 경우:

```
$ oc -n openshift-ingress rsh <router_pod> cat /var/lib/haproxy/conf/error_code_pages/error-page-404.http
```

## 검증

사용자 정의 오류 코드 HTTP 응답을 확인합니다.

- 테스트 프로젝트 및 애플리케이션을 생성합니다.

```
$ oc new-project test-ingress
```

```
$ oc new-app django-psql-example
```

- 503 사용자 정의 http 오류 코드 응답의 경우:

- 애플리케이션의 모든 pod를 중지합니다.

- 다음 curl 명령을 실행하거나 브라우저에서 경로 호스트 이름을 방문합니다.

```
$ curl -vk <route_hostname>
```

- 404 사용자 정의 http 오류 코드 응답의 경우:

- a. 존재하지 않는 경로 또는 잘못된 경로를 방문합니다.
- b. 다음 curl 명령을 실행하거나 브라우저에서 경로 호스트 이름을 방문합니다.

```
$ curl -vk <route_hostname>
```

4. **errorfile** 속성이 **haproxy.config** 파일에 제대로 있는지 확인합니다.

```
$ oc -n openshift-ingress rsh <router> cat /var/lib/haproxy/conf/haproxy.config | grep errorfile
```

### 6.8.19. Ingress 컨트롤러 최대 연결 설정

클러스터 관리자는 OpenShift 라우터 배포에 대한 최대 동시 연결 수를 설정할 수 있습니다. 기존 Ingress 컨트롤러를 패치하여 최대 연결 수를 늘릴 수 있습니다.

#### 사전 요구 사항

- 다음은 Ingress 컨트롤러를 이미 생성했다고 가정합니다.

#### 절차

- HAProxy에 대한 최대 연결 수를 변경하려면 Ingress 컨트롤러를 업데이트합니다.

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --type=merge -p '{"spec": {"tuningOptions": {"maxConnections": 7500}}}'
```



#### 주의

**spec.tuningOptions.maxConnections** 값을 현재 운영 체제 제한보다 크게 설정하면 HAProxy 프로세스가 시작되지 않습니다. 이 매개변수에 대한 자세한 내용은 "Ingress Controller 구성 매개변수" 섹션의 표를 참조하십시오.

## 6.9. 추가 리소스

- [사용자 정의 PKI 구성](#)

## 7장. OPENSIFT CONTAINER PLATFORM의 INGRESS 분할

OpenShift Container Platform에서 Ingress 컨트롤러는 모든 경로를 제공하거나 경로 서브 세트를 제공할 수 있습니다. 기본적으로 Ingress 컨트롤러는 클러스터의 모든 네임스페이스에서 생성된 모든 경로를 제공합니다. 선택한 특성을 기반으로 경로의 하위 집합인 *shard* 를 생성하여 라우팅을 최적화하도록 클러스터에 Ingress 컨트롤러를 추가할 수 있습니다. 경로를 *shard*의 멤버로 표시하려면 경로 또는 네임스페이스 메타데이터 필드에 라벨을 사용합니다. Ingress 컨트롤러는 *선택 표현식* 이라고도 하는 *선택기* 를 사용하여 제공할 전체 경로 풀에서 경로 서브 세트를 선택합니다.

Ingress 분할은 트래픽을 특정 Ingress 컨트롤러로 라우팅하거나 다음 섹션에 설명된 다양한 이유로 트래픽을 분리하려는 경우 여러 Ingress 컨트롤러에서 들어오는 트래픽을 로드 밸런싱하려는 경우에 유용합니다.

기본적으로 각 경로는 클러스터의 기본 도메인을 사용합니다. 그러나 라우터 도메인을 대신 사용하도록 경로를 구성할 수 있습니다. 자세한 내용은 [Ingress 컨트롤러 Sharding의 경로 생성](#) 을 참조하십시오.

### 7.1. INGRESS 컨트롤러 분할

라우터 샤딩이라고도 하는 Ingress 샤딩을 사용하여 경로, 네임스페이스 또는 둘 다에 라벨을 추가하여 여러 라우터에 경로 집합을 배포할 수 있습니다. Ingress 컨트롤러는 해당 선택기 세트를 사용하여 지정된 라벨이 있는 경로만 허용합니다. 각 Ingress shard는 지정된 선택 표현식을 사용하여 필터링되는 경로로 구성됩니다.

트래픽이 클러스터로 유입되는 기본 메커니즘으로 Ingress 컨트롤러의 요구 사항이 중요할 수 있습니다. 클러스터 관리자는 다음을 위해 경로를 분할할 수 있습니다.

- 여러 경로를 통해 Ingress 컨트롤러 또는 라우터를 로드 밸런싱하여 변경에 대한 응답 속도 향상
- 특정 경로가 나머지 경로와 다른 수준의 신뢰성을 가지도록 할당
- 특정 Ingress 컨트롤러에 다른 정책을 정의할 수 있도록 허용
- 특정 경로만 추가 기능을 사용하도록 허용
- 예를 들어, 내부 및 외부 사용자가 다른 경로를 볼 수 있도록 다른 주소에 다른 경로를 노출
- 녹색 배포 중에 애플리케이션의 한 버전에서 다른 버전으로 트래픽을 전송합니다.

Ingress 컨트롤러가 분할되면 지정된 경로가 그룹의 0개 이상의 Ingress 컨트롤러에 허용됩니다. 경로 상태는 Ingress 컨트롤러가 승인했는지 여부를 나타냅니다. Ingress 컨트롤러는 해당 shard에 고유한 경우에만 경로를 허용합니다.

Ingress 컨트롤러는 세 가지 분할 방법을 사용할 수 있습니다.

- 네임스페이스 선택기와 일치하는 라벨이 있는 네임스페이스의 모든 경로가 Ingress shard에 있도록 Ingress 컨트롤러에 네임스페이스 선택기만 추가합니다.
- 경로 선택기와 일치하는 레이블이 있는 모든 경로가 Ingress shard에 있도록 Ingress 컨트롤러에 경로 선택기만 추가합니다.
- 네임스페이스 선택기와 경로 선택기를 모두 Ingress 컨트롤러에 추가하여 네임스페이스 선택기와 일치하는 라벨과 일치하는 레이블이 있는 경로가 Ingress shard에 있도록 합니다.

샤딩을 사용하면 여러 Ingress 컨트롤러에 경로 서브 세트를 배포할 수 있습니다. 이러한 서브셋은 오버라이프(over-overla)이거나, 기존 샤딩(Sampling)이라고도 하며, 중복되는 분할이라고 할 수 있습니다.

### 7.1.1. 기존 분할 예

Ingress 컨트롤러 **finops-router** 는 label selector **spec.namespaceSelector.matchLabels.name** 을 Thanos 및 **ops** 로 설정하여 구성됩니다.

#### **finops-router**에 대한 YAML 정의의 예

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: finops-router
    namespace: openshift-ingress-operator
  spec:
    namespaceSelector:
      matchLabels:
        name:
          - finance
          - ops
```

두 번째 Ingress 컨트롤러 **dev-router** 는 라벨 선택기 **spec.namespaceSelector.matchLabels.name** 을 **dev** 로 설정하여 구성됩니다.

#### **dev-router**YAML 정의의 예

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: dev-router
    namespace: openshift-ingress-operator
  spec:
    namespaceSelector:
      matchLabels:
        name: dev
```

모든 애플리케이션 경로가 별도의 네임스페이스에 있는 경우 각각 **name:finance**, **name:ops**, **name:dev** 로 레이블이 지정된 경우 이 구성은 두 Ingress 컨트롤러 간에 경로를 효과적으로 배포합니다. 콘솔, 인증 및 기타 목적을 위한 OpenShift Container Platform 경로는 처리해서는 안 됩니다.

위의 시나리오에서는 분할이 중복된 하위 집합이 없는 특별한 파티션 케이스가 됩니다. 경로는 라우터 shard 간에 나뉩니다.



### 주의

**namespaceSelector** 또는 **routeSelector** 필드에 제외를 위한 경로가 포함되지 않는 한 기본 Ingress 컨트롤러는 모든 경로를 계속 제공합니다. 기본 Ingress 컨트롤러에서 경로를 제외하는 방법에 대한 자세한 내용은 이 [Red Hat Knowledgebase 솔루션](#) 및 "기본 Ingress 컨트롤러 삭제" 섹션을 참조하십시오.

## 7.1.2. 중복된 분할 예

위의 예에서 **finops-router** 및 **dev-router** 외에도, **dev** 및 **ops** 로 설정된 라벨 선택기 **spec.namespaceSelector.matchLabels.name** 으로 구성됩니다.

### ECDHE -router에 대한 YAML 정의의 예

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: devops-router
    namespace: openshift-ingress-operator
  spec:
    namespaceSelector:
      matchLabels:
        name:
          - dev
          - ops
```

**name:dev** 및 **name:ops** 레이블이 지정된 네임스페이스의 경로는 이제 서로 다른 두 개의 Ingress 컨트롤러에서 서비스를 제공합니다. 이 구성을 사용하면 경로가 중복되는 하위 집합이 있습니다.

경로의 중복된 하위 집합을 사용하면 더 복잡한 라우팅 규칙을 생성할 수 있습니다. 예를 들어 전용 **finops-router** 로 더 높은 우선 순위 트래픽을 전환하면서 더 낮은 우선 순위 트래픽을 **ECDHE -router** 로 전송할 수 있습니다.

## 7.1.3. 기본 Ingress 컨트롤러 분할

새 Ingress shard를 생성한 후 기본 Ingress 컨트롤러에도 적용되는 새 Ingress shard에 적용되는 경



로가 있을 수 있습니다. 이는 기본 **Ingress** 컨트롤러에 선택기가 없고 기본적으로 모든 경로를 허용하기 때문입니다.

네임스페이스 선택기 또는 경로 선택기를 사용하여 특정 라벨을 사용하여 **Ingress** 컨트롤러를 서비스 경로에서 제한할 수 있습니다. 다음 절차에서는 기본 **Ingress** 컨트롤러가 네임스페이스 선택기를 사용하여 새로 분할된 **Thanos**, **ops**, **dev** 경로를 서비스하지 못하도록 제한합니다. 이렇게 하면 **Ingress shard**에 더 많은 격리가 추가되었습니다.



#### 중요

모든 **OpenShift Container Platform** 관리 경로를 동일한 **Ingress** 컨트롤러에 보관해야 합니다. 따라서 이러한 필수 경로를 제외하는 기본 **Ingress** 컨트롤러에 추가 선택기를 추가하지 마십시오.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- 프로젝트 관리자로 로그인되어 있습니다.

#### 절차

1. 다음 명령을 실행하여 기본 **Ingress** 컨트롤러를 수정합니다.

```
$ oc edit ingresscontroller -n openshift-ingress-operator default
```

2. **jaeger**, **ops**, **dev** 라벨이 있는 경로를 제외하는 **namespaceSelector** 를 포함 하도록 **Ingress** 컨트롤러를 편집합니다.

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: default
    namespace: openshift-ingress-operator
  spec:
    namespaceSelector:
      matchExpressions:
      - key: type
        operator: NotIn
      values:
```

- **finance**
- **ops**
- **dev**

기본 **Ingress** 컨트롤러는 더 이상 **name:finance, name:ops, name:dev** 로 레이블이 지정된 네임스페이스를 제공하지 않습니다.

#### 7.1.4. Ingress 분할 및 DNS

클러스터 관리자는 프로젝트의 각 라우터에 대해 별도의 **DNS** 항목을 수행해야 합니다. 라우터는 알 수 없는 경로를 다른 라우터로 전달하지 않습니다.

다음 예제를 고려하십시오.

- 라우터 **A**는 호스트 **192.168.0.5**에 존재하며 **\*.foo.com** 이 있는 경로가 있습니다.
- 라우터 **B**는 호스트 **192.168.1.9**에 있으며 **\*.example.com** 이 있는 경로가 있습니다.

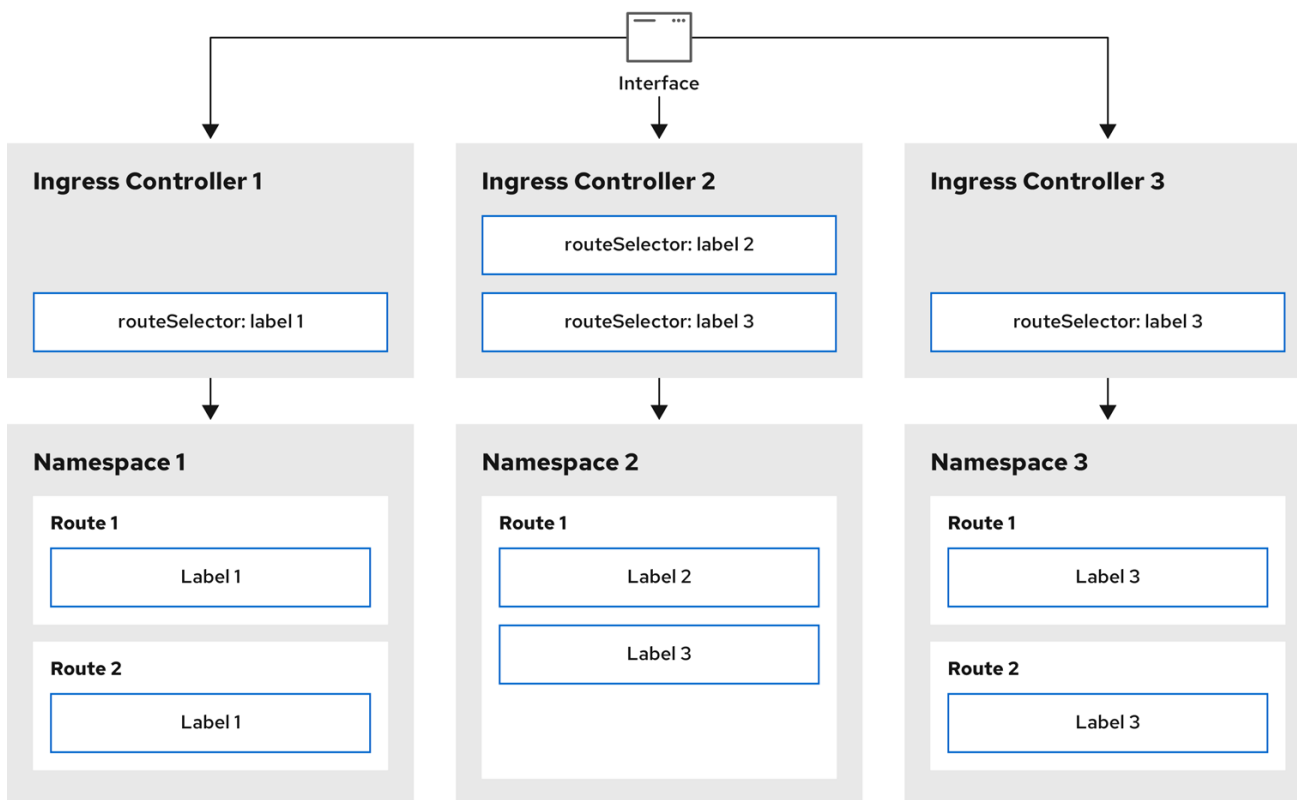
별도의 **DNS** 항목은 라우터 **A** 및 **\*.example.com** 을 라우터 **B**를 호스팅하는 노드에 **\*.foo.com** 으로 확인해야 합니다.

- **\*.foo.com A IN 192.168.0.5**
- **\*.example.com A IN 192.168.1.9**

#### 7.1.5. 경로 라벨을 사용하여 Ingress 컨트롤러 분할 구성

경로 라벨을 사용한 **Ingress** 컨트롤러 분할이란 **Ingress** 컨트롤러가 경로 선택기에서 선택한 모든 네임스페이스의 모든 경로를 제공한다는 뜻입니다.

그림 7.1. 경로 라벨을 사용한 Ingress 분할



301\_OpenShift\_0123

**Ingress** 컨트롤러 분할은 들어오는 트래픽 부하를 일련의 **Ingress** 컨트롤러에 균형 있게 분배하고 트래픽을 특정 **Ingress** 컨트롤러에 격리할 때 유용합니다. 예를 들어, 회사 A는 하나의 **Ingress** 컨트롤러로, 회사 B는 다른 **Ingress** 컨트롤러로 이동합니다.

#### 프로세스

1.

`router-internal.yaml` 파일을 다음과 같이 편집합니다.

```
# cat router-internal.yaml
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    routeSelector:
      matchLabels:
```

```

type: sharded
status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
    
```

2.

**Ingress** 컨트롤러 `router-internal.yaml` 파일을 적용합니다.

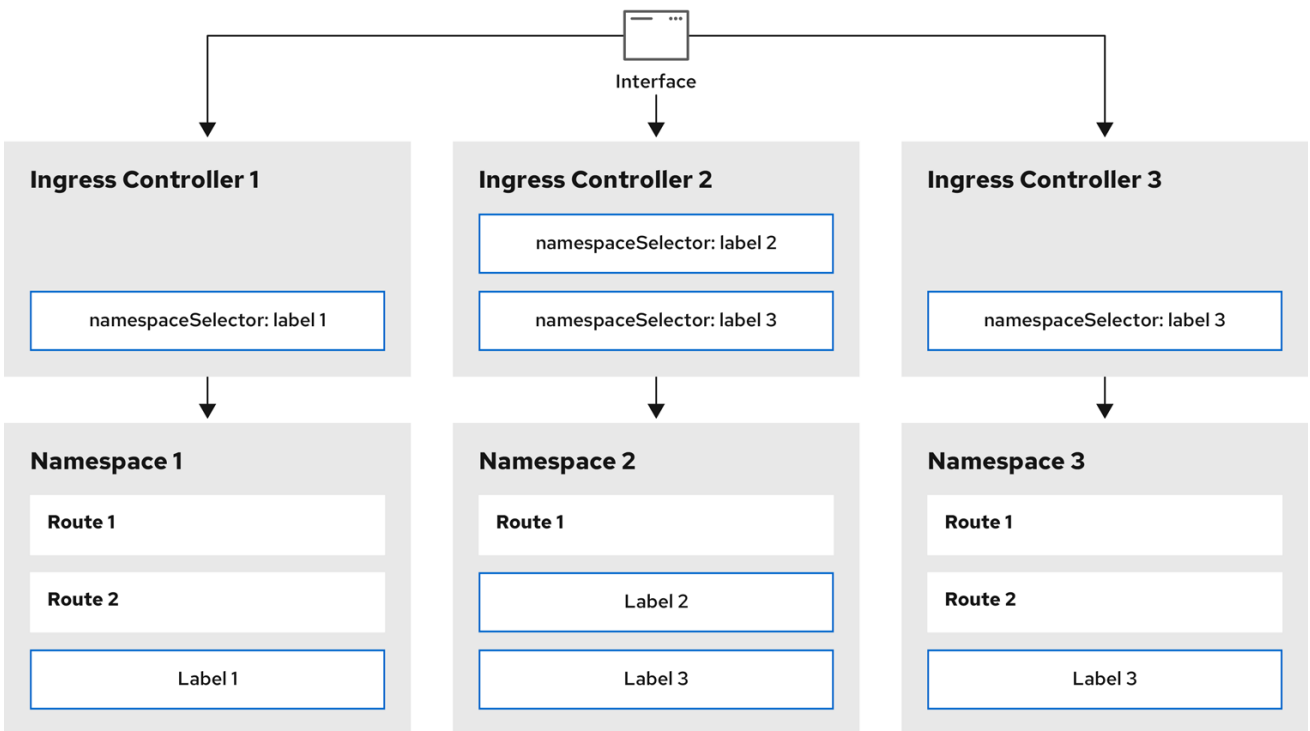
```
# oc apply -f router-internal.yaml
```

**Ingress** 컨트롤러는 `type: sharded` 라벨이 있는 네임스페이스에서 경로를 선택합니다.

### 7.1.6. 네임스페이스 라벨을 사용하여 **Ingress** 컨트롤러 분할 구성

네임스페이스 라벨을 사용한 **Ingress** 컨트롤러 분할이란 **Ingress** 컨트롤러가 네임스페이스 선택기에 서 선택한 모든 네임스페이스의 모든 경로를 제공한다는 뜻입니다.

그림 7.2. 네임스페이스 라벨을 사용한 **Ingress** 분할



301\_OpenShift\_0123

**Ingress** 컨트롤러 분할은 들어오는 트래픽 부하를 일련의 **Ingress** 컨트롤러에 균형 있게 분배하고 트래픽을 특정 **Ingress** 컨트롤러에 격리할 때 유용합니다. 예를 들어, 회사 A는 하나의 **Ingress** 컨트롤러로, 회사 B는 다른 **Ingress** 컨트롤러로 이동합니다.

## 프로세스

1.

`router-internal.yaml` 파일을 다음과 같이 편집합니다.

```
# cat router-internal.yaml
```

출력 예

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    namespaceSelector:
      matchLabels:
        type: sharded
    status: {}
  kind: List
  metadata:
    resourceVersion: ""
    selfLink: ""
```

2.

`Ingress` 컨트롤러 `router-internal.yaml` 파일을 적용합니다.

```
# oc apply -f router-internal.yaml
```

`Ingress` 컨트롤러는 네임스페이스 선택기에서 선택한 `type: sharded` 라벨이 있는 네임스페이스에서 경로를 선택합니다.

## 7.2. INGRESS 컨트롤러 분할을 위한 경로 생성

경로를 사용하면 URL에서 애플리케이션을 호스팅할 수 있습니다. 이 경우 호스트 이름이 설정되지 않고 경로는 대신 하위 도메인을 사용합니다. 하위 도메인을 지정하면 경로를 노출하는 `Ingress` 컨트롤러의

도메인을 자동으로 사용합니다. 여러 **Ingress** 컨트롤러에서 경로가 노출되는 경우 경로는 여러 **URL**에서 호스팅됩니다.

다음 절차에서는 **hello-openshift** 애플리케이션을 예제로 사용하여 **Ingress** 컨트롤러 분할에 대한 경로를 생성하는 방법을 설명합니다.

**Ingress** 컨트롤러 분할은 들어오는 트래픽 부하를 일련의 **Ingress** 컨트롤러에 균형 있게 분배하고 트래픽을 특정 **Ingress** 컨트롤러에 격리할 때 유용합니다. 예를 들어, 회사 **A**는 하나의 **Ingress** 컨트롤러로, 회사 **B**는 다른 **Ingress** 컨트롤러로 이동합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- 프로젝트 관리자로 로그인되어 있습니다.
- 포트에서 트래픽을 수신 대기하는 포트 및 **HTTP** 또는 **TLS** 엔드포인트를 노출하는 웹 애플리케이션이 있습니다.
- 분할을 위해 **Ingress** 컨트롤러가 구성되어 있습니다.

#### 프로세스

1. 다음 명령을 실행하여 **hello-openshift** 라는 프로젝트를 생성합니다.

```
$ oc new-project hello-openshift
```

2. 다음 명령을 실행하여 프로젝트에 **Pod**를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. 다음 명령을 실행하여 **hello-openshift** 라는 서비스를 생성합니다.

```
$ oc expose pod/hello-openshift
```

4.

`hello-openshift-route.yaml` 이라는 경로 정의를 생성합니다.

분할을 위해 생성된 경로의 **YAML** 정의:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded ❶
    name: hello-openshift-edge
    namespace: hello-openshift
spec:
  subdomain: hello-openshift ❷
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

❶

레이블 키와 해당 레이블 값은 **Ingress** 컨트롤러에 지정된 라벨과 일치해야 합니다. 이 예제에서 **Ingress** 컨트롤러에는 레이블 키와 값 `type: sharded` 가 있습니다.

❷

경로는 `subdomain` 필드의 값을 사용하여 노출됩니다. 하위 도메인 필드를 지정할 때 호스트 이름을 설정되지 않은 상태로 두어야 합니다. `host` 및 `subdomain` 필드를 모두 지정하면 경로는 호스트 필드의 값을 사용하고 `subdomain` 필드를 무시합니다.

5.

다음 명령을 실행하여 `hello-openshift-route.yaml` 을 사용하여 `hello-openshift` 애플리케이션에 대한 경로를 생성합니다.

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

검증

•

다음 명령을 사용하여 경로의 상태를 가져옵니다.

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

생성된 **Route** 리소스는 다음과 유사해야 합니다.

출력 예

```
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

**Ingress** 컨트롤러 또는 라우터에서 경로를 노출하는 데 사용하는 호스트 이름입니다. **host** 필드의 값은 **Ingress** 컨트롤러에 의해 자동으로 결정되며 도메인을 사용합니다. 이 예제에서 **Ingress** 컨트롤러의 도메인은 `< apps-sharded.basedomain.example.net >`입니다.

2

**Ingress** 컨트롤러의 호스트 이름입니다.

3

**Ingress** 컨트롤러의 이름입니다. 이 예제에서 **Ingress** 컨트롤러에는 이름 `sharded` 가 있습니다.



## 추가 리소스

- [기본 Ingress 컨트롤러\(라우터\) 성능](#)

## 8장. INGRESS 컨트롤러 끝점 게시 전략 구성

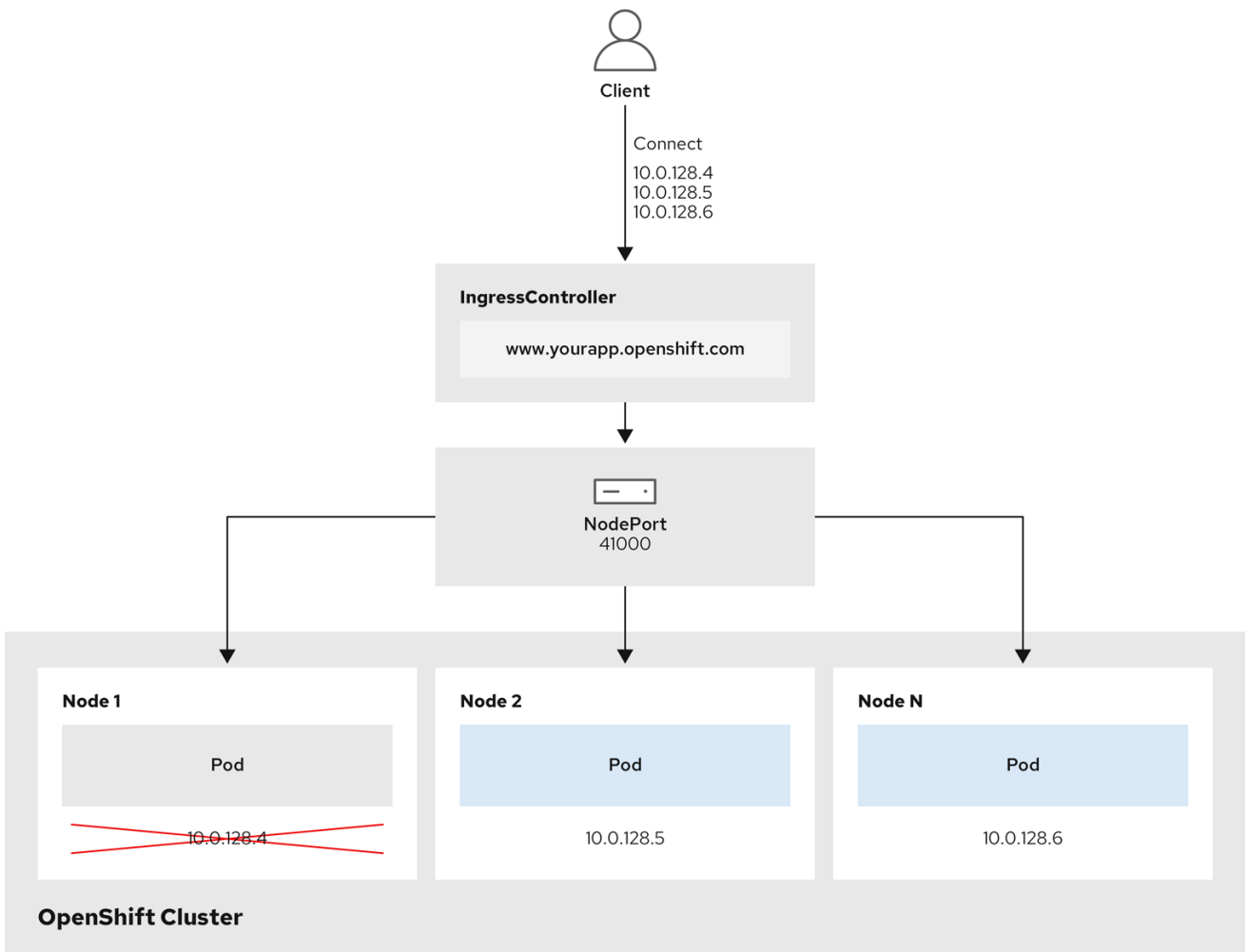
### 8.1. INGRESS 컨트롤러 끝점 게시 전략

#### NodePortService 끝점 게시 전략

**NodePortService** 끝점 게시 전략에서는 **Kubernetes NodePort** 서비스를 사용하여 **Ingress** 컨트롤러를 게시합니다.

이 구성에서는 **Ingress** 컨트롤러를 배포하기 위해 컨테이너 네트워킹을 사용합니다. 배포를 게시하기 위해 **NodePortService**가 생성됩니다. 특정 노드 포트는 **OpenShift Container Platform**에 의해 동적으로 할당됩니다. 그러나 정적 포트 할당을 지원하기 위해 관리형 **NodePortService**의 노드 포트 필드에 대한 변경 사항은 유지됩니다.

그림 8.1. Diagram of NodePortService



202\_OpenShift\_0222

이전 그래픽에서는 **OpenShift Container Platform Ingress NodePort** 끝점 게시 전략에 대한 다음 개

념을 보여줍니다.

- 클러스터에서 사용 가능한 모든 노드에는 외부에서 액세스할 수 있는 자체 IP 주소가 있습니다. 클러스터에서 실행 중인 서비스는 모든 노드의 고유한 **NodePort**에 바인딩됩니다.
- 클라이언트가 그래픽에 **10.0.128.4 IP** 주소를 연결하여 다운된 노드에 클라이언트가 연결되면 노드 포트는 서비스를 실행 중인 사용 가능한 노드에 클라이언트를 직접 연결합니다. 이 시나리오에서는 로드 밸런싱이 필요하지 않습니다. 이미지가 표시된 대로 **10.0.128.4** 주소가 다운되어 다른 IP 주소를 대신 사용해야 합니다.



#### 참고

**Ingress Operator**는 서비스의 `.spec.ports[].nodePort` 필드에 대한 업데이트를 무시합니다.

기본적으로 포트는 자동으로 할당되며 통합을 위해 포트 할당에 액세스할 수 있습니다. 그러나 동적 포트에 대한 응답으로 쉽게 재구성할 수 없는 기존 인프라와 통합하기 위해 정적 포트 할당이 필요한 경우가 있습니다. 정적 노드 포트와 통합하기 위해 관리 서비스 리소스를 직접 업데이트할 수 있습니다.

자세한 내용은 [NodePort에 대한 Kubernetes 서비스 설명서](#)를 참조하십시오.

### HostNetwork 끝점 게시 전략

**HostNetwork** 끝점 게시 전략에서는 **Ingress** 컨트롤러가 배포된 노드 포트에 **Ingress** 컨트롤러를 게시합니다.

**HostNetwork** 끝점 게시 전략이 있는 **Ingress** 컨트롤러는 노드당 하나의 **Pod** 복제본만 가질 수 있습니다. **n**개의 복제본이 필요한 경우에는 해당 복제본을 예약할 수 있는 **n**개 이상의 노드를 사용해야 합니다. 각 **pod** 복제본은 예약된 노드 호스트에서 포트 **80** 및 **443**을 요청하므로 동일한 노드의 다른 **pod**가 해당 포트를 사용하는 경우 복제본을 노드에 예약할 수 없습니다.

#### 8.1.1. Ingress 컨트롤러 끝점 게시 범위를 Internal에 구성

클러스터 관리자가 클러스터가 프라이빗으로 지정되도록 지정하지 않고 새 클러스터를 설치하면 기본 **Ingress** 컨트롤러가 **External** 로 설정된 범위를 사용하여 생성됩니다. 클러스터 관리자는 외부 범위

**Ingress** 컨트롤러를 **Internal** 로 변경할 수 있습니다.

사전 요구 사항

- **oc CLI**를 설치했습니다.

절차

- 외부 범위가 지정된 **Ingress** 컨트롤러를 **Internal** 로 변경하려면 다음 명령을 입력합니다.

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/default --type=merge --
patch='{"spec":{"endpointPublishingStrategy":
{"type":"LoadBalancerService","loadBalancer":{"scope":"Internal"}}}'
```

- **Ingress** 컨트롤러의 상태를 확인하려면 다음 명령을 입력합니다.

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

- **Progressing** 상태 조건은 추가 작업을 수행해야 하는지 여부를 나타냅니다. 예를 들어 상태 조건은 다음 명령을 입력하여 서비스를 삭제해야 함을 나타낼 수 있습니다.

```
$ oc -n openshift-ingress delete services/router-default
```

서비스를 삭제하면 **Ingress Operator**에서 내부로 다시 생성합니다.

### 8.1.2. Ingress 컨트롤러 끝점 게시 범위를 외부로 구성

클러스터 관리자가 클러스터가 프라이빗으로 지정되도록 지정하지 않고 새 클러스터를 설치하면 기본 **Ingress** 컨트롤러가 **External** 로 설정된 범위를 사용하여 생성됩니다.

설치 중 또는 이후에 **Ingress** 컨트롤러의 범위를 **Internal** 로 구성할 수 있으며 클러스터 관리자는 내부 **Ingress** 컨트롤러를 외부로 변경할 수 있습니다.

중요

일부 플랫폼에서는 서비스를 삭제하고 다시 생성해야 합니다.

범위를 변경하면 몇 분 동안 **Ingress** 트래픽이 중단될 수 있습니다. 이는 **OpenShift Container Platform**이 기존 서비스 로드 밸런서를 프로비저닝 해제하고 새 서비스를 프로비저닝하며 **DNS**를 업데이트할 수 있기 때문에 서비스를 삭제하고 다시 생성해야 하는 플랫폼에 적용됩니다.

## 사전 요구 사항

- **oc CLI**를 설치했습니다.

## 절차

- 내부 범위가 지정된 **Ingress** 컨트롤러를 외부로 변경하려면 다음 명령을 입력합니다.

```
$ oc -n openshift-ingress-operator patch ingresscontrollers/private --type=merge --
patch='{"spec":{"endpointPublishingStrategy":
{"type":"LoadBalancerService","loadBalancer":{"scope":"External"}}}'
```

- **Ingress** 컨트롤러의 상태를 확인하려면 다음 명령을 입력합니다.

```
$ oc -n openshift-ingress-operator get ingresscontrollers/default -o yaml
```

- **Progressing** 상태 조건은 추가 작업을 수행해야 하는지 여부를 나타냅니다. 예를 들어 상태 조건은 다음 명령을 입력하여 서비스를 삭제해야 함을 나타낼 수 있습니다.

```
$ oc -n openshift-ingress delete services/router-default
```

서비스를 삭제하면 **Ingress Operator**가 외부로 다시 생성합니다.

## 8.2. 추가 리소스

- 자세한 내용은 **Ingress** 컨트롤러 구성 매개변수를 참조하십시오.

## 9장. 끝점에 대한 연결 확인

**CNO(Cluster Network Operator)**는 클러스터 내 리소스 간에 연결 상태 검사를 수행하는 연결 확인 컨트롤러인 컨트롤러를 실행합니다. 상태 점검 결과를 검토하여 연결 문제를 진단하거나 현재 조사하고 있는 문제의 원인으로 네트워크 연결을 제거할 수 있습니다.

### 9.1. 연결 상태 점검 수행

클러스터 리소스에 도달할 수 있는지 확인하기 위해 다음 클러스터 API 서비스 각각에 TCP 연결이 수행됩니다.

- **Kubernetes API 서버 서비스**
- **Kubernetes API 서버 끝점**
- **OpenShift API 서버 서비스**
- **OpenShift API 서버 끝점**
- **로드 밸런서**

클러스터의 모든 노드에서 서비스 및 서비스 끝점에 도달할 수 있는지 확인하기 위해 다음 대상 각각에 TCP 연결이 수행됩니다.

- **상태 점검 대상 서비스**
- **상태 점검 대상 끝점**

### 9.2. 연결 상태 점검 구현

연결 검증 컨트롤러는 클러스터의 연결 확인 검사를 오케스트레이션합니다. 연결 테스트의 결과는 **openshift-network-diagnostics**의 **PodNetworkConnectivity** 오브젝트에 저장됩니다. 연결 테스트는 병렬로 1분마다 수행됩니다.

**CNO(Cluster Network Operator)**는 클러스터에 여러 리소스를 배포하여 연결 상태 점검을 전달하고 수신합니다.

#### 상태 점검 소스

이 프로그램은 **Deployment** 오브젝트에서 관리하는 단일 포드 복제본 세트에 배포됩니다. 프로그램은 **PodNetworkConnectivity** 오브젝트를 사용하고 각 오브젝트에 지정된 **spec.targetEndpoint**에 연결됩니다.

#### 상태 점검 대상

클러스터의 모든 노드에서 데몬 세트의 일부로 배포된 포드입니다. 포드는 인바운드 상태 점검을 수신 대기합니다. 모든 노드에 이 포드가 있으면 각 노드로의 연결을 테스트할 수 있습니다.

### 9.3. PODNETWORKCONNECTIVITYCHECK 오브젝트 필드

**PodNetworkConnectivityCheck** 오브젝트 필드는 다음 표에 설명되어 있습니다.

표 9.1. **PodNetworkConnectivityCheck** 오브젝트 필드

필드	유형	설명
<b>metadata.name</b>	<b>string</b>	다음과 같은 형식의 오브젝트 이름: <b>&lt;source&gt;-to- &lt;target&gt;</b> . <b>&lt;target&gt;</b> 에서 설명한 대상에는 다음 문자열 중 하나가 포함되어 있습니다. <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>	오브젝트와 연결된 네임스페이스입니다. 이 값은 항상 <b>openshift-network-diagnostics</b> 입니다.
<b>spec.sourcePod</b>	<b>string</b>	연결 확인이 시작된 포드의 이름입니다(예: <b>network-check-source-596b4c6566-rgh92</b> ).
<b>spec.targetEndpoint</b>	<b>string</b>	연결 검사의 대상입니다(예: <b>api.devcluster.example.com:6443</b> ).

필드	유형	설명
<b>spec.tlsClientCert</b>	<b>object</b>	사용할 TLS 인증서 설정입니다.
<b>spec.tlsClientCert.name</b>	<b>string</b>	해당하는 경우 사용되는 TLS 인증서의 이름입니다. 기본값은 빈 문자열입니다.
<b>status</b>	<b>object</b>	연결 테스트의 조건 및 최근 연결 성공 및 실패의 로그를 나타내는 오브젝트입니다.
<b>status.conditions</b>	<b>array</b>	연결 확인의 최신 상태 및 모든 이전 상태입니다.
<b>status.failures</b>	<b>array</b>	실패한 시도에서의 연결 테스트 로그입니다.
<b>status.outages</b>	<b>array</b>	중단 기간을 포함하는 테스트 로그를 연결합니다.
<b>status.successes</b>	<b>array</b>	성공적인 시도에서의 연결 테스트 로그입니다.

다음 표에서는 **status.conditions** 배열에서 오브젝트 필드를 설명합니다.

표 9.2. **status.conditions**

필드	유형	설명
<b>lastTransitionTime</b>	<b>string</b>	연결 조건이 하나의 상태에서 다른 상태로 전환된 시간입니다.
<b>message</b>	<b>string</b>	사람이 읽기 쉬운 형식으로 마지막 전환에 대한 세부 정보입니다.
<b>reason</b>	<b>string</b>	머신에서 읽을 수 있는 형식으로 전환의 마지막 상태입니다.
<b>status</b>	<b>string</b>	조건의 상태:
<b>type</b>	<b>string</b>	조건의 유형입니다.

다음 표에서는 **status.conditions** 배열에서 오브젝트 필드를 설명합니다.

표 9.3. **status.outages**

필드	유형	설명
----	----	----



필드	유형	설명
<b>end</b>	<b>string</b>	연결 오류가 해결될 때부터의 타임 스탬프입니다.
<b>endLogs</b>	<b>array</b>	서비스 중단에 성공적인 종료와 관련된 로그 항목을 포함한 연결 로그 항목입니다.
<b>message</b>	<b>string</b>	사람이 읽을 수 있는 형식의 중단 세부 정보에 대한 요약입니다.
<b>start</b>	<b>string</b>	연결 오류가 먼저 감지될 때부터의 타임 스탬프입니다.
<b>startLogs</b>	<b>array</b>	원래 오류를 포함한 연결 로그 항목입니다.

### 연결 로그 필드

연결 로그 항목의 필드는 다음 표에 설명되어 있습니다. 오브젝트는 다음 필드에서 사용됩니다.

- **`status.failures[]`**
- **`status.successes[]`**
- **`status.outages[].startLogs[]`**
- **`status.outages[].endLogs[]`**

표 9.4. 연결 로그 오브젝트

필드	유형	설명
<b>latency</b>	<b>string</b>	작업 기간을 기록합니다.
<b>message</b>	<b>string</b>	사람이 읽을 수 있는 형식으로 상태를 제공합니다.
<b>reason</b>	<b>string</b>	머신에서 읽을 수 있는 형식으로 상태의 이유를 제공합니다. 값은 <b>TCPConnect</b> , <b>TCPConnectError</b> , <b>DNSResolve</b> , <b>DNSError</b> 중 하나입니다.
<b>success</b>	<b>boolean</b>	로그 항목이 성공 또는 실패인지를 나타냅니다.
<b>time</b>	<b>string</b>	연결 확인 시작 시간입니다.

### 9.4. 끝점에 대한 네트워크 연결 확인

클러스터 관리자는 API 서버, 로드 밸런서, 서비스 또는 포드와 같은 끝점의 연결을 확인할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 프로세스

1. 현재 **PodNetworkConnectivityCheck** 오브젝트를 나열하려면 다음 명령을 입력합니다.

```
$ oc get podnetworkconnectivitycheck -n openshift-network-diagnostics
```

#### 출력 예

NAME	AGE
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rgrp-master-0	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rgrp-master-1	73m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rgrp-master-2	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-kubernetes-apiserver-service-cluster	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-kubernetes-default-service-cluster	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-load-balancer-api-external	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-load-balancer-api-internal	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rgrp-master-0	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rgrp-master-1	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rgrp-master-2	75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh	74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rgrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rgrp-worker-c-n8mbf	74m

```

network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-
target-ci-ln-x5sv9rb-f76d1-4rzrp-worker-d-4hnrz 74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-
target-service-cluster 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-
apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-
apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-
apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2 74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-
apiserver-service-cluster 75m

```

2.

연결 테스트 로그를 확인합니다.

a.

이전 명령의 출력에서 연결 로그를 검토할 끝점을 식별합니다.

b.

오브젝트를 확인하려면 다음 명령을 입력합니다:

```

$ oc get podnetworkconnectivitycheck <name> \
-n openshift-network-diagnostics -o yaml

```

여기서 <name>은 PodNetworkConnectivityCheck 오브젝트의 이름을 지정합니다.

출력 예

```

apiVersion: controlplane.operator.openshift.io/v1alpha1
kind: PodNetworkConnectivityCheck
metadata:
  name: network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-
kubernetes-apiserver-endpoint-ci-ln-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'
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-f76d1-4rgrp-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-f76d1-4rgrp-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-f76d1-4rgrp-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-f76d1-4rgrp-master-0:
tcp
    connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnect
    success: true
    time: "2021-01-13T21:05:34Z"
```

## 10장. 클러스터 네트워크의 MTU 변경

클러스터 관리자는 클러스터 설치 후 클러스터 네트워크의 MTU를 변경할 수 있습니다. MTU 변경을 종료하려면 클러스터 노드를 재부팅해야 하므로 이러한 변경이 중단됩니다. OVN-Kubernetes 또는 OpenShift SDN 클러스터 네트워크 공급자를 사용하여 클러스터의 MTU만 변경할 수 있습니다.

### 10.1. 클러스터 MTU 정보

클러스터 네트워크의 최대 전송 단위(MTU)를 설치하는 동안 클러스터에 있는 노드의 기본 네트워크 인터페이스 MTU를 기반으로 자동으로 탐지됩니다. 일반적으로 감지된 MTU를 재정의할 필요는 없습니다.

다음과 같은 몇 가지 이유로 클러스터 네트워크의 MTU를 변경할 수 있습니다.

- 클러스터 설치 중에 감지된 MTU가 인프라에 적합하지 않음
- 이제 클러스터 인프라에는 최적의 성능을 위해 다른 MTU가 필요한 노드 추가와 같이 다른 MTU가 필요합니다.

OVN-Kubernetes 및 OpenShift SDN 클러스터 네트워크 공급자에 대해서만 클러스터 MTU를 변경할 수 있습니다.

#### 10.1.1. 서비스 중단 고려 사항

클러스터에서 MTU 변경을 시작하면 다음과 같은 결과가 서비스 가용성에 영향을 미칠 수 있습니다.

- 새 MTU로 마이그레이션을 완료하려면 두 개 이상의 롤링 재부팅이 필요합니다. 이 기간 동안 일부 노드를 재시작하기 때문에 사용할 수 없습니다.
- 절대 TCP 시간 초과 간격보다 짧은 시간 제한 간격을 사용하여 클러스터에 배포된 특정 애플리케이션은 MTU를 변경하는 동안 중단될 수 있습니다.

#### 10.1.2. MTU 값 선택

MTU 마이그레이션을 계획할 때 다음과 같은 두 가지 MTU 값이 서로 다릅니다.

- **Hardware MTU:** 이 MTU 값은 네트워크 인프라의 세부 사항에 따라 설정됩니다.
  
- **클러스터 네트워크 MTU:** 이 MTU 값은 클러스터 네트워크 오버레이 오버헤드를 고려하기 위해 하드웨어 MTU보다 항상 적습니다. 특정 오버헤드는 클러스터 네트워크 공급자에 의해 결정됩니다.
  - **OVN-Kubernetes:** 100 바이트
  
  - **OpenShift SDN:** 50 바이트

클러스터에 다른 노드에 대한 다른 MTU 값이 필요한 경우 클러스터의 모든 노드에서 사용하는 가장 낮은 MTU 값에서 클러스터 네트워크 공급자의 오버헤드 값을 제거해야 합니다. 예를 들어, 클러스터의 일부 노드에 9001의 MTU가 있고 일부에는 1500의 MTU가 있는 경우 이 값을 1400으로 설정해야 합니다.

### 10.1.3. 마이그레이션 프로세스의 작동 방식

다음 표는 프로세스의 사용자 시작 단계와 마이그레이션이 수행하는 작업 간에 분할하여 마이그레이션 프로세스를 요약합니다.

표 10.1. 클러스터 MTU의 실시간 마이그레이션

사용자 시작 단계	OpenShift Container Platform 활동
-----------	---------------------------------



사용자 시작 단계	OpenShift Container Platform 활동
<p>Cluster Network Operator 구성에서 다음 값을 설정합니다.</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>CNO(Cluster Network Operator):</b> 각 필드가 유효한 값으로 설정되었는지 확인합니다.</p> <ul style="list-style-type: none"> <li>● <b>mtu.machine.to</b> 는 하드웨어의 MTU를 변경하지 않는 경우 새 하드웨어 MTU 또는 현재 하드웨어 MTU로 설정해야 합니다. 이 값은 일시적인 것이며 마이그레이션 프로세스의 일부로 사용됩니다. 기존 하드웨어 MTU 값과 다른 하드웨어 MTU 값과 다른 하드웨어 MTU를 별도로 지정하는 경우 머신 구성, DHCP 설정 또는 Linux 커널 명령줄과 같은 다른 방법으로 MTU를 수동으로 구성해야 합니다.</li> <li>● <b>mtu.network.from</b> 필드는 클러스터 네트워크의 현재 MTU인 <b>network.status.clusterNetworkMTU</b> 필드와 같아야 합니다.</li> <li>● <b>mtu.network.to</b> 필드는 대상 클러스터 네트워크 MTU로 설정해야 하며 클러스터 네트워크 공급자의 오버레이 오버헤드를 허용하려면 하드웨어 MTU보다 작아야 합니다. OVN-Kubernetes의 경우 오버헤드는 <b>100</b> 바이트이고 OpenShift SDN의 경우 오버헤드는 <b>50</b> 바이트입니다.</li> </ul> <p>제공된 값이 유효한 경우 CNO는 <b>mtu.network.to</b> 필드 값으로 설정된 클러스터 네트워크의 MTU를 사용하여 새 임시 구성을 기록합니다.</p> <p><b>MCO(Machine Config Operator):</b> 클러스터의 각 노드의 롤링 재부팅을 수행합니다.</p>
<p>클러스터에 있는 노드의 기본 네트워크 인터페이스의 MTU를 재구성합니다. 다음을 포함하여 다양한 방법을 사용하여 이를 수행할 수 있습니다.</p> <ul style="list-style-type: none"> <li>● MTU 변경 사항을 사용하여 새 NetworkManager 연결 프로필 배포</li> <li>● DHCP 서버 설정을 통해 MTU 변경</li> <li>● 부팅 매개 변수를 통해 MTU 변경</li> </ul>	<p>해당 없음</p>
<p>클러스터 네트워크 공급자의 CNO 구성에서 <b>mtu</b> 값을 설정하고 <b>spec.migration</b> 을 <b>null</b> 로 설정합니다.</p>	<p><b>MCO(Machine Config Operator):</b> 새 MTU 구성으로 클러스터의 각 노드가 롤링 재부팅을 수행합니다.</p>

## 10.2. 클러스터 MTU 변경

클러스터 관리자는 클러스터의 최대 전송 단위(MTU)를 변경할 수 있습니다. 마이그레이션이 중단되고 MTU 업데이트가 돌아오므로 클러스터의 노드를 일시적으로 사용할 수 없게 될 수 있습니다.

다음 절차에서는 머신 구성, **DHCP** 또는 **ISO**를 사용하여 클러스터 **MTU**를 변경하는 방법을 설명합니다. **DHCP** 또는 **ISO** 접근법을 사용하는 경우 절차를 완료하기 위해 클러스터를 설치한 후 보관한 구성 아티팩트를 참조해야 합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 클러스터의 대상 **MTU**를 확인했습니다. 올바른 **MTU**는 클러스터가 사용하는 클러스터 네트워크 공급자에 따라 다릅니다.
  - **OVN-Kubernetes:** 클러스터 **MTU**는 클러스터의 가장 낮은 하드웨어 **MTU** 값보다 **100** 미만으로 설정되어야 합니다.
  - **OpenShift SDN:** 클러스터 **MTU**는 클러스터에서 가장 낮은 하드웨어 **MTU** 값보다 **50** 미만으로 설정되어야 합니다.

#### 절차

클러스터 네트워크의 **MTU**를 늘리거나 줄이려면 다음 절차를 완료합니다.

1. 클러스터 네트워크의 현재 **MTU**를 가져오려면 다음 명령을 입력합니다.

```
$ oc describe network.config cluster
```

출력 예

```
...
Status:
Cluster Network:
  Cidr:          10.217.0.0/22
  Host Prefix:   23
  Cluster Network MTU: 1400
```

**Network Type:** OpenShiftSDN

**Service Network:**

10.217.4.0/23

...

2.

하드웨어 MTU 설정을 준비합니다.

- DHCP를 사용하여 하드웨어 MTU를 지정하는 경우 다음 dnsmasq 구성과 같은 DHCP 구성을 업데이트합니다.

```
dhcp-option-force=26,<mtu>
```

다음과 같습니다.

<mtu>

알릴 DHCP 서버의 하드웨어 MTU를 지정합니다.

- PXE로 커널 명령줄을 사용하여 하드웨어 MTU를 지정하는 경우 그에 따라 해당 구성을 업데이트합니다.
- NetworkManager 연결 구성에 하드웨어 MTU를 지정하는 경우 다음 단계를 완료합니다. 이 방법은 DHCP, 커널 명령줄 또는 기타 방법으로 네트워크 구성을 명시적으로 지정하지 않은 경우 OpenShift Container Platform의 기본값입니다. 다음 절차에서는 클러스터 노드에서 모두 동일한 기본 네트워크 구성을 사용하여 수정되지 않은 상태로 작동해야 합니다.

i.

기본 네트워크 인터페이스를 찾습니다.

o

OpenShift SDN 클러스터 네트워크 공급자를 사용하는 경우 다음 명령을 입력합니다.

```
$ oc debug node/<node_name> -- chroot /host ip route list match 0.0.0.0/0 | awk '{print $5}'
```

다음과 같습니다.

**<node\_name>**

클러스터의 노드 이름을 지정합니다.

o

**OVN-Kubernetes** 클러스터 네트워크 공급자를 사용하는 경우 다음 명령을 입력합니다.

```
$ oc debug node/<node_name> -- chroot /host nmcli -g
connection.interface-name c show ovs-if-phys0
```

다음과 같습니다.

**<node\_name>**

클러스터의 노드 이름을 지정합니다.

ii.

이전 명령에서 반환된 인터페이스 이름에 대해 **NetworkManager**가 생성한 연결 프로필을 찾으려면 다음 명령을 입력합니다.

```
$ oc debug node/<node_name> -- chroot /host nmcli c | grep <interface>
```

다음과 같습니다.

**<interface>**

기본 네트워크 인터페이스의 이름을 지정합니다.

**OpenShift SDN**의 출력 예

```
Wired connection 1 46da4a6a-xxxx-xxxx-xxxx-ac0ca900f213 ethernet ens3
```

원래 연결 구성이 없는 **OVN-Kubernetes**의 출력 예

```

ovs-if-phys0    353774d3-0d3d-4ada-b14e-cd4d8824e2a8 ethernet    ens4
ovs-port-phys0  332ef950-b2e5-4991-a0dc-3158977c35ca ovs-port    ens4

```

**OVN-Kubernetes** 클러스터 네트워크 공급자의 경우 2개 또는 3개의 연결 관리자 프로필이 반환됩니다.

- 이전 명령에서 두 개의 프로필만 반환하는 경우 기본 **NetworkManager** 연결 구성을 템플릿으로 사용해야 합니다.
- 이전 명령에서 세 개의 프로필을 반환하는 경우 다음 수정 사항에 대한 템플릿으로 **ovs-if-phys0** 또는 **ovs-port-phys0** 이라는 프로필을 사용합니다.

iii.

기본 네트워크 인터페이스에 대한 **NetworkManager** 연결 구성의 파일 이름을 가져오려면 다음 명령을 입력합니다.

```

$ oc debug node/<node_name> -- chroot /host nmcli -g UUID,FILENAME c
show | grep <uuid> | cut -d: -f2

```

다음과 같습니다.

**<node\_name>**

클러스터의 노드 이름을 지정합니다.

**<uuid>**

**NetworkManager** 연결 프로필의 **UUID**를 지정합니다.

출력 예

```

/run/NetworkManager/system-connections/Wired connection 1.nmconnection

```

iv.

노드에서 **NetworkManager** 연결 구성을 복사하려면 다음 명령을 입력합니다.

```
$ oc debug node/<node_name> -- chroot /host cat "<profile_path>" >
config.nmconnection
```

다음과 같습니다.

**<node\_name>**

클러스터의 노드 이름을 지정합니다.

**<profile\_path>**

이전 단계에서 **NetworkManager** 연결의 파일 시스템 경로를 지정합니다.

**NetworkManager** 연결 구성 예

```
[connection]
id=Wired connection 1
uuid=3e96a02b-xxxx-xxxx-ad5d-61db28678130
type=ethernet
autoconnect-priority=-999
interface-name=enp1s0
permissions=
timestamp=1644109633

[ethernet]
mac-address-blacklist=

[ipv4]
dns-search=
method=auto

[ipv6]
addr-gen-mode=stable-privacy
dns-search=
method=auto

[proxy]

[.nmmeta]
nm-generated=true
```

v.

이전 단계에서 `config.nmconnection` 파일에 저장된 `NetworkManager` 구성 파일을 편집합니다.

○

다음 값을 설정합니다.

■

**802-3-Ethernet.mtu:** 시스템의 기본 네트워크 인터페이스에 대한 MTU를 지정합니다.

■

**connection.interface-name:** 선택 사항: 이 구성이 적용되는 네트워크 인터페이스 이름을 지정합니다.

■

**connection.autoconnect-priority:** 선택 사항: 0 이상의 정수 우선 순위 값을 지정하여 이 프로필이 동일한 인터페이스에 대해 다른 프로필에서 사용되는지 확인하는 것이 좋습니다. `OVN-Kubernetes` 클러스터 네트워크 공급자를 사용하는 경우 이 값은 100 보다 작아야 합니다.

○

`connection.uuid` 필드를 제거합니다.

○

다음 값을 변경합니다.

■

**connection.id:** 선택 사항: 다른 `NetworkManager` 연결 프로필 이름을 지정합니다.

### NetworkManager 연결 구성 예

```
[connection]
id=Primary network interface
type=ethernet
autoconnect-priority=10
interface-name=enp1s0
[802-3-ethernet]
mtu=8051
```

vi.

컨트롤 플레인 노드 및 클러스터의 작업자 노드에 대해 하나씩 두 개의 **MachineConfig** 오브젝트를 생성합니다.

A.

**control-plane-interface.bu** 파일에 다음 **Butane** 구성을 생성합니다.

```
variant: openshift
version: 4.11.0
metadata:
  name: 01-control-plane-interface
  labels:
    machineconfiguration.openshift.io/role: master
storage:
  files:
    - path: /etc/NetworkManager/system-connections/<connection_name>
      1
      contents:
        local: config.nmconnection 2
        mode: 0600
```

1

기본 네트워크 인터페이스에 대한 **NetworkManager** 연결 이름을 지정합니다.

2

이전 단계에서 업데이트된 **NetworkManager** 구성 파일의 로컬 파일 이름을 지정합니다.

B.

**worker-interface.bu** 파일에 다음 **Butane** 구성을 생성합니다.

```
variant: openshift
version: 4.11.0
metadata:
  name: 01-worker-interface
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/system-connections/<connection_name>
      1
      contents:
        local: config.nmconnection 2
        mode: 0644
```

1



2

이전 단계에서 업데이트된 **NetworkManager** 구성 파일의 로컬 파일 이름을 지정합니다.

C.

다음 명령을 실행하여 **Butane** 구성에서 **MachineConfig** 오브젝트를 생성합니다.

```
$ for manifest in control-plane-interface worker-interface; do
  butane --files-dir . $manifest.bu > $manifest.yaml
done
```

3.

**MTU** 마이그레이션을 시작하려면 다음 명령을 입력하여 마이그레이션 구성을 지정합니다. **Machine Config Operator**는 **MTU** 변경을 준비하기 위해 클러스터에서 노드 롤링 재부팅을 수행합니다.

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
'{"spec": {"migration": {"mtu": {"network": {"from": <overlay_from>, "to": <overlay_to> }, "machine": {"to": <machine_to> } } } }'
```

다음과 같습니다.

<overlay\_from>

현재 클러스터 네트워크 **MTU** 값을 지정합니다.

<overlay\_to>

클러스터 네트워크의 대상 **MTU**를 지정합니다. 이 값은 < machine\_to > 값을 기준으로 설정되며 **OVN-Kubernetes**의 값은 100 미만이어야 하며 **OpenShift SDN**은 50 미만이어야 합니다.

<machine\_to>

기본 호스트 네트워크의 기본 네트워크 인터페이스의 **MTU**를 지정합니다.

클러스터 **MTU**를 늘리는 예

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
'{"spec": {"migration": {"mtu": {"network": {"from": 1400, "to": 9000 }, "machine": {"to": 9100} } } }'
```

4.

**MCO**는 각 머신 구성 풀의 머신을 업데이트할 때 각 노드를 하나씩 재부팅합니다. 모든 노드가 업데이트될 때까지 기다려야 합니다. 다음 명령을 입력하여 머신 구성 풀 상태를 확인합니다.

```
$ oc get mcp
```

업데이트된 노드의 상태가 **UPDATED=true, UPDATING=false, DEGRADED=false**입니다.



참고

기본적으로 **MCO**는 풀당 한 번에 하나의 시스템을 업데이트하므로 클러스터 크기에 따라 마이그레이션에 걸리는 총 시간이 증가합니다.

5.

호스트의 새 머신 구성 상태를 확인합니다.

a.

머신 구성 상태 및 적용된 머신 구성 이름을 나열하려면 다음 명령을 입력합니다.

```
$ oc describe node | egrep "hostname|machineconfig"
```

출력 예

```
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
```

다음 구문이 올바른지 확인합니다.

- **machineconfiguration.openshift.io/state** 필드의 값은 **Done**입니다.
- **machineconfiguration.openshift.io/currentConfig** 필드의 값은 **machineconfiguration.openshift.io/desiredConfig** 필드의 값과 동일합니다.

b. 머신 구성이 올바른지 확인하려면 다음 명령을 입력합니다.

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

여기서 **<config\_name>**은 **machineconfiguration.openshift.io/currentConfig** 필드에서 머신 구성의 이름입니다.

머신 구성은 다음 업데이트를 **systemd** 구성에 포함해야 합니다.

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

6. 기본 네트워크 인터페이스 MTU 값을 업데이트합니다.

- **NetworkManager** 연결 구성으로 새 MTU를 지정하는 경우 다음 명령을 입력합니다. **MachineConfig Operator**는 클러스터의 노드 롤링 재부팅을 자동으로 수행합니다.

```
$ for manifest in control-plane-interface worker-interface; do
  oc create -f $manifest.yaml
done
```

- **DHCP** 서버 옵션 또는 커널 명령줄 및 **PXE**로 새 MTU를 지정하는 경우 인프라에 필요한 변경을 수행합니다.

7. **MCO**는 각 머신 구성 풀의 머신을 업데이트할 때 각 노드를 하나씩 재부팅합니다. 모든 노드가 업데이트될 때까지 기다려야 합니다. 다음 명령을 입력하여 머신 구성 풀 상태를 확인합니다.

```
$ oc get mcp
```

업데이트된 노드의 상태가 **UPDATED=true, UPDATING=false, DEGRADED=false**입니다.



참고

기본적으로 **MCO**는 풀당 한 번에 하나의 시스템을 업데이트하므로 클러스터 크기에 따라 마이그레이션에 걸리는 총 시간이 증가합니다.

8. 호스트의 새 머신 구성 상태를 확인합니다.

a. 머신 구성 상태 및 적용된 머신 구성 이름을 나열하려면 다음 명령을 입력합니다.

```
$ oc describe node | egrep "hostname|machineconfig"
```

출력 예

```
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
```

다음 구문이 올바른지 확인합니다.

- **machineconfiguration.openshift.io/state** 필드의 값은 **Done**입니다.
- **machineconfiguration.openshift.io/currentConfig** 필드의 값은 **machineconfiguration.openshift.io/desiredConfig** 필드의 값과 동일합니다.

b. 머신 구성이 올바른지 확인하려면 다음 명령을 입력합니다.

```
$ oc get machineconfig <config_name> -o yaml | grep path:
```

여기서 **<config\_name>**은 **machineconfiguration.openshift.io/currentConfig** 필드에

서 머신 구성의 이름입니다.

머신 구성이 성공적으로 배포된 경우 이전 출력에 `/etc/NetworkManager/system-connections/<connection_name >` 파일 경로가 포함됩니다.

머신 구성에는 `ExecStart=/usr/local/bin/mtu-migration.sh` 행이 포함되어 있지 않아야 합니다.

9.

MTU 마이그레이션을 완료하려면 다음 명령 중 하나를 입력합니다.

- 

**OVN-Kubernetes** 클러스터 네트워크 공급자를 사용하는 경우 다음을 수행합니다.

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": {"migration": null, "defaultNetwork":{"ovnKubernetesConfig": {"mtu":
  <mtu> }}}}'
```

다음과 같습니다.

<mtu>

< overlay\_to>로 지정한 새 클러스터 네트워크 MTU를 지정합니다.

- 

**OpenShift SDN** 클러스터 네트워크 공급자를 사용하는 경우:

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": {"migration": null, "defaultNetwork":{"openshiftSDNConfig": {"mtu":
  <mtu> }}}}'
```

다음과 같습니다.

<mtu>

< overlay\_to>로 지정한 새 클러스터 네트워크 MTU를 지정합니다.

검증

클러스터의 노드가 이전 프로세스에서 지정한 MTU를 사용하는지 확인할 수 있습니다.

1. 클러스터 네트워크의 현재 MTU를 가져오려면 다음 명령을 입력합니다.

```
$ oc describe network.config cluster
```

2. 노드의 기본 네트워크 인터페이스에 대한 현재 MTU를 가져옵니다.

- a. 클러스터의 노드를 나열하려면 다음 명령을 입력합니다.

```
$ oc get nodes
```

- b. 노드의 기본 네트워크 인터페이스의 현재 MTU 설정을 가져오려면 다음 명령을 입력합니다.

```
$ oc debug node/<node> -- chroot /host ip address show <interface>
```

다음과 같습니다.

**<node>**

이전 단계의 출력에서 노드를 지정합니다.

**<interface>**

노드의 기본 네트워크 인터페이스 이름을 지정합니다.

출력 예

```
ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8051
```

### 10.3. 추가 리소스

- [PXE 및 ISO 설치를 위한 고급 네트워크 옵션 사용](#)

- 키 파일 형식으로 **NetworkManager** 프로파일 수동 생성
- **nmcli**를 사용하여 동적 이더넷 연결 구성

## 11장. 노드 포트 서비스 범위 구성

클러스터 관리자는 사용 가능한 노드 포트 범위를 확장할 수 있습니다. 클러스터에서 많은 수의 노드 포트를 사용하는 경우 사용 가능한 포트 수를 늘려야 할 수 있습니다.

기본 포트 범위는 **30000~32767**입니다. 기본 범위 이상으로 확장한 경우에도 포트 범위는 축소할 수 없습니다.

### 11.1. 사전 요구 사항

- 클러스터 인프라는 확장된 범위 내에서 지정한 포트에 대한 액세스를 허용해야 합니다. 예를 들어, 노드 포트 범위를 **30000~32900**으로 확장하는 경우 방화벽 또는 패킷 필터링 구성에서 **32768~32900**의 포함 포트 범위를 허용해야 합니다.

### 11.2. 노드 포트 범위 확장

클러스터의 노드 포트 범위를 확장할 수 있습니다.

#### 사전 요구 사항

- OpenShift CLI(oc)**를 설치합니다.
- cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

#### 프로세스

- 노드 포트 범위를 확장하려면 다음 명령을 입력합니다. **<port>**를 새 범위에서 가장 큰 포트 번호로 변경합니다.

```
$ oc patch network.config.openshift.io cluster --type=merge -p '{
  "spec":
    { "serviceNodePortRange": "30000-<port>" }
}'
```



## 작은 정보

또는 다음 **YAML**을 적용하여 노드 포트 범위를 업데이트할 수 있습니다.

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  serviceNodePortRange: "30000-<port>"
```

## 출력 예

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

2.

구성이 활성 상태인지 확인하려면 다음 명령을 입력합니다. 업데이트가 적용되려면 몇 분 정도 걸릴 수 있습니다.

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

## 출력 예

```
"service-node-port-range":["30000-33000"]
```

## 11.3. 추가 리소스

- [NodePort](#)를 사용하여 수신 클러스터 트래픽 구성
- [Network \[config.openshift.io/v1\]](#)

- ***Service [core/v1]***

## 12장. IP 페일오버 구성

다음에서는 **OpenShift Container Platform** 클러스터의 **Pod** 및 서비스에 대한 **IP 페일오버** 구성에 대해 설명합니다.

**IP 페일오버**는 노드 집합에서 **VIP(가상 IP)** 주소 풀을 관리합니다. 세트의 모든 **VIP**는 세트에서 선택한 노드에서 서비스를 제공합니다. 단일 노드를 사용할 수 있는 경우 **VIP**가 제공됩니다. 노드에 **VIP**를 명시적으로 배포할 방법은 없으므로 **VIP**가 없는 노드와 많은 **VIP**가 많은 다른 노드가 있을 수 있습니다. 노드가 하나만 있는 경우 모든 **VIP**가 노드에 있습니다.



참고

**VIP**는 클러스터 외부에서 라우팅할 수 있어야 합니다.

**IP 페일오버**는 각 **VIP**의 포트를 모니터링하여 노드에서 포트에 연결할 수 있는지 확인합니다. 포트에 연결할 수 없는 경우 **VIP**가 노드에 할당되지 않습니다. 포트를 **0**으로 설정하면 이 검사가 비활성화됩니다. 검사 스크립트는 필요한 테스트를 수행합니다.

**IP 페일오버**는 **Keepalived**를 사용하여 호스트 집합에서 외부 액세스가 가능한 **VIP** 주소 집합을 호스팅합니다. 각 **VIP**는 한 번에 하나의 호스트에서만 서비스를 제공합니다. **keepalived**는 **VRRP(Virtual Router Redundancy Protocol: 가상 라우터 중복 프로토콜)**를 사용하여 호스트 집합에서 **VIP**를 대상으로 서비스를 결정합니다. 호스트를 사용할 수 없게 되거나 **Keepalived** 서비스가 응답하지 않는 경우 **VIP**가 세트의 다른 호스트로 전환됩니다. 즉, 호스트를 사용할 수 있는 한 **VIP**는 항상 서비스됩니다.

**Keepalived**를 실행하는 노드가 확인 스크립트를 통과하면 해당 노드의 **VIP**가 우선 순위와 현재 **master**의 우선 순위 및 선점 전략에 따라 마스터 상태를 입력할 수 있습니다.

클러스터 관리자는 **OPENSIFT\_HA\_NOTIFY\_SCRIPT** 변수를 통해 스크립트를 제공할 수 있으며 이 스크립트는 노드의 **VIP** 상태가 변경될 때마다 호출됩니다. **keepalived**는 **VIP**를 서비스하는 경우 **master** 상태를 사용하고, 다른 노드가 **VIP**를 서비스할 때 **backup** 상태를 사용하거나 검사 스크립트가 실패할 때 **fault** 상태를 사용합니다. 알림 스크립트는 상태가 변경될 때마다 새 상태로 호출됩니다.

**OpenShift Container Platform**에서 **IP 장애 조치 배포** 구성을 생성할 수 있습니다. **IP 장애 조치 배포** 구성은 **VIP** 주소 집합과 서비스할 노드 집합을 지정합니다. 클러스터에는 고유한 **VIP** 주소 집합을 관리할 때마다 여러 **IP 페일오버 배포** 구성이 있을 수 있습니다. **IP 장애 조치** 구성의 각 노드는 **IP 장애 조치 Pod**를 실행하며 이 **Pod**는 **Keepalived**를 실행합니다.

**VIP**를 사용하여 호스트 네트워킹이 있는 **pod**에 액세스하는 경우 애플리케이션 **pod**는 **IP 페일오버 pod**

를 실행하는 모든 노드에서 실행됩니다. 이를 통해 모든 IP 페일오버 노드가 마스터가 되고 필요한 경우 VIP에 서비스를 제공할 수 있습니다. IP 페일오버가 있는 모든 노드에서 애플리케이션 pod가 실행되지 않는 경우 일부 IP 페일오버 노드가 VIP를 서비스하지 않거나 일부 애플리케이션 pod는 트래픽을 수신하지 않습니다. 이러한 불일치를 방지하려면 IP 페일오버 및 애플리케이션 pod 모두에 동일한 선택기 및 복제 수를 사용합니다.

VIP를 사용하여 서비스에 액세스하는 동안 애플리케이션 pod가 실행 중인 위치와 상관없이 모든 노드에서 서비스에 연결할 수 있으므로 모든 노드가 IP 페일오버 노드 세트에 있을 수 있습니다. 언제든지 IP 페일오버 노드가 마스터가 될 수 있습니다. 서비스는 외부 IP와 서비스 포트를 사용하거나 NodePort를 사용할 수 있습니다.

서비스 정의에서 외부 IP를 사용하는 경우 VIP가 외부 IP로 설정되고 IP 페일오버 모니터링 포트가 서비스 포트에 설정됩니다. 노드 포트를 사용하면 포트는 클러스터의 모든 노드에서 열려 있으며, 서비스는 현재 VIP를 서비스하는 모든 노드에서 트래픽을 로드 밸런싱합니다. 이 경우 서비스 정의에서 IP 페일오버 모니터링 포트가 NodePort로 설정됩니다.



중요

NodePort 설정은 권한 있는 작업입니다.



중요

VIP 서비스의 가용성이 높더라도 성능은 여전히 영향을 받을 수 있습니다. keepalived는 각 VIP가 구성의 일부 노드에서 서비스되도록 하고, 다른 노드에 아무것도 없는 경우에도 여러 VIP가 동일한 노드에 배치될 수 있도록 합니다. IP 페일오버가 동일한 노드에 여러 VIP를 배치하면 일련의 VIP에 걸쳐 외부적으로 로드 밸런싱을 수행하는 전략이 좌절될 수 있습니다.

ingressIP를 사용하는 경우 ingressIP 범위와 동일한 VIP 범위를 갖도록 IP 페일오버를 설정할 수 있습니다. 모니터링 포트를 비활성화할 수도 있습니다. 이 경우 모든 VIP가 클러스터의 동일한 노드에 표시됩니다. 모든 사용자는 ingressIP를 사용하여 서비스를 설정하고 고가용성으로 설정할 수 있습니다.



중요

클러스터에는 최대 254개의 VIP가 있습니다.

## 12.1. IP 페일오버 환경 변수

다음 표에는 IP 페일오버를 구성하는 데 사용되는 변수가 표시되어 있습니다.

표 12.1. IP 페일오버 환경 변수

변수 이름	기본값	설명
<b>OPENSIFT_HA_MONITOR_PORT</b>	<b>80</b>	IP 페일오버 pod는 각 가상 IP(VIP)에서 이 포트에 대한 TCP 연결을 엽니다. 연결이 설정되면 서비스가 실행 중인 것으로 간주됩니다. 이 포트가 <b>0</b> 으로 설정되면 테스트가 항상 통과합니다.
<b>OPENSIFT_HA_NETWORK_INTERFACE</b>		IP 페일오버가 VRRP(Virtual Router Redundancy Protocol) 트래픽을 보내는 데 사용하는 인터페이스 이름입니다. 기본값은 <b>eth0</b> 입니다.
<b>OPENSIFT_HA_REPLICA_COUNT</b>	<b>2</b>	생성할 복제본 수입니다. 이는 IP 페일오버 배포 구성의 <b>spec.replicas</b> 값과 일치해야 합니다.
<b>OPENSIFT_HA_VIRTUAL_IPS</b>		복제할 IP 주소 범위 목록입니다. 이 정보를 제공해야 합니다. 예: <b>1.2.3.4-6,1.2.3.9</b> .
<b>OPENSIFT_HA_VRRP_ID_OFFSET</b>	<b>0</b>	가상 라우터 ID를 설정하는 데 사용되는 오프셋 값입니다. 다른 오프셋 값을 사용하면 동일한 클러스터 내에 여러 IP 페일오버 구성이 존재할 수 있습니다. 기본 오프셋은 <b>0</b> 이며 허용되는 범위는 <b>0</b> 에서 <b>255</b> 사이입니다.
<b>OPENSIFT_HA_VIP_GROUPS</b>		VRRP에 대해 생성할 그룹 수입니다. 설정하지 않으면 <b>OPENSIFT_HA_VIP_GROUPS</b> 변수로 지정된 각 가상 IP 범위에 대해 그룹이 생성됩니다.
<b>OPENSIFT_HA_IPTABLES_CHAIN</b>	INPUT	VRRP 트래픽을 허용하는 iptables 규칙을 자동으로 추가하는 <b>iptables</b> 체인의 이름입니다. 값을 설정하지 않으면 <b>iptables</b> 규칙이 추가되지 않습니다. 체인이 없으면 생성되지 않습니다.
<b>OPENSIFT_HA_CHECK_SCRIPT</b>		애플리케이션이 작동하는지 확인하기 위해 정기적으로 실행되는 스크립트의 Pod 파일 시스템에 있는 전체 경로 이름입니다.
<b>OPENSIFT_HA_CHECK_INTERVAL</b>	<b>2</b>	확인 스크립트가 실행되는 기간(초)입니다.
<b>OPENSIFT_HA_NOTIFY_SCRIPT</b>		상태가 변경될 때마다 실행되는 스크립트의 Pod 파일 시스템의 전체 경로 이름입니다.
<b>OPENSIFT_HA_PREEMPTION</b>	<b>preempt_nodelay 300</b>	더 높은 우선 순위의 호스트를 처리하는 전략입니다. <b>nopreempt</b> 전략에서는 더 낮은 우선 순위 호스트에서 더 높은 우선 순위 호스트로 마스터를 이동하지 않습니다.

## 12.2. IP 페일오버 구성

클러스터 관리자는 레이블 선택기에 정의된 대로 전체 클러스터 또는 노드의 하위 집합에서 IP 페일오버를 구성할 수 있습니다. 클러스터에서 여러 IP 페일오버 배포 구성을 설정할 수도 있습니다. 이 배포 구성은 서로 독립적입니다.

IP 페일오버 배포 구성을 사용하면 제약 조건 또는 사용된 라벨과 일치하는 각 노드에서 페일오버 pod가 실행됩니다.

이 Pod는 **Keepalived**를 실행하여 엔드포인트를 모니터링하고 **VRRP(Virtual Router Redundancy Protocol)**를 사용하여 첫 번째 노드가 서비스 또는 엔드포인트에 연결할 수 없는 경우 한 노드에서 다른 노드로의 가상 IP(VIP)를 페일오버할 수 있습니다.

프로덕션 용도의 경우 두 개 이상의 노드를 선택하는 **selector**를 설정하고 선택한 노드 수와 동일한 **replicas**를 설정합니다.

#### 사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 풀 시크릿을 생성했습니다.

#### 프로세스

1. IP 페일오버 서비스 계정을 생성합니다.
 

```
$ oc create sa ipfailover
```
2. **hostNetwork**의 **SCC(보안 컨텍스트 제약 조건)**를 업데이트합니다.
 

```
$ oc adm policy add-scc-to-user privileged -z ipfailover
$ oc adm policy add-scc-to-user hostnetwork -z ipfailover
```
3. IP 페일오버를 구성하기 위해 배포 **YAML** 파일을 만듭니다.

IP 페일오버 구성을 위한 배포 **YAML**의 예

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: ipfailover-keepalived ❶
  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: quay.io/openshift/origin-keepalived-ipfailover
          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: OPENSIFT_HA_CONFIG_NAME
              value: "ipfailover"
            - name: OPENSIFT_HA_VIRTUAL_IPS ❷
              value: "1.1.1.1-2"
            - name: OPENSIFT_HA_VIP_GROUPS ❸
              value: "10"
            - name: OPENSIFT_HA_NETWORK_INTERFACE ❹
              value: "ens3" #The host interface to assign the VIPs
            - name: OPENSIFT_HA_MONITOR_PORT ❺

```

```

    value: "30060"
  - name: OPENSIFT_HA_VRRP_ID_OFFSET 6
    value: "0"
  - name: OPENSIFT_HA_REPLICA_COUNT 7
    value: "2" #Must match the number of replicas in the deployment
  - name: OPENSIFT_HA_USE_UNICAST
    value: "false"
  #- name: OPENSIFT_HA_UNICAST_PEERS
    #value: "10.0.148.40,10.0.160.234,10.0.199.110"
  - name: OPENSIFT_HA_IPTABLES_CHAIN 8
    value: "INPUT"
  #- name: OPENSIFT_HA_NOTIFY_SCRIPT 9
    # value: /etc/keepalive/mynotifyscript.sh
  - name: OPENSIFT_HA_CHECK_SCRIPT 10
    value: "/etc/keepalive/mycheckscript.sh"
  - name: OPENSIFT_HA_PREEMPTION 11
    value: "preempt_delay 300"
  - name: OPENSIFT_HA_CHECK_INTERVAL 12
    value: "2"
  livenessProbe:
    initialDelaySeconds: 10
    exec:
      command:
        - pgrep
        - keepalived
  volumes:
  - name: lib-modules
    hostPath:
      path: /lib/modules
  - name: host-slash
    hostPath:
      path: /
  - name: etc-sysconfig
    hostPath:
      path: /etc/sysconfig
  # config-volume contains the check script
  # created with `oc create configmap keepalived-checkscript --from-
  file=mycheckscript.sh`
  - configMap:
      defaultMode: 0755
      name: keepalived-checkscript
      name: config-volume
  imagePullSecrets:
  - name: openshift-pull-secret 13

```

1

2



3

VRRP에 대해 생성할 그룹 수입니다. 설정하지 않으면 **OPENSIFT\_HA\_VIP\_GROUPS** 변수로 지정된 각 가상 IP 범위에 대해 그룹이 생성됩니다.

4

IP 페일오버가 VRRP 트래픽을 보내는 데 사용하는 인터페이스 이름입니다. 기본적으로 **eth0**이 사용됩니다.

5

IP 페일오버 pod는 각 VIP에서 이 포트에 대한 TCP 연결을 열려고 합니다. 연결이 설정되면 서비스가 실행 중인 것으로 간주됩니다. 이 포트가 0으로 설정되면 테스트가 항상 통과합니다. 기본값은 80입니다.

6

가상 라우터 ID를 설정하는 데 사용되는 오프셋 값입니다. 다른 오프셋 값을 사용하면 동일한 클러스터 내에 여러 IP 페일오버 구성이 존재할 수 있습니다. 기본 오프셋은 0이며 허용되는 범위는 0에서 255 사이입니다.

7

생성할 복제본 수입니다. 이는 IP 페일오버 배포 구성의 **spec.replicas** 값과 일치해야 합니다. 기본값은 2입니다.

8

VRRP 트래픽을 허용하는 iptables 규칙을 자동으로 추가하는 iptables 체인의 이름입니다. 값을 설정하지 않으면 iptables 규칙이 추가되지 않습니다. 체인이 존재하지 않으면 이 체인이 생성되지 않으며 Keepalived는 유니캐스트 모드로 작동합니다. 기본값은 INPUT입니다.

9

상태가 변경될 때마다 실행되는 스크립트의 Pod 파일 시스템의 전체 경로 이름입니다.

10

애플리케이션이 작동하는지 확인하기 위해 정기적으로 실행되는 스크립트의 Pod 파일 시스템에 있는 전체 경로 이름입니다.

11

더 높은 우선 순위의 호스트를 처리하는 전략입니다. 기본값은 **preempt\_delay 300**으로, 우선순위가 낮은 마스터가 VIP를 보유하는 경우 Keepalived 인스턴스가 5분 후에 VIP를

넘겨받습니다.

12

확인 스크립트가 실행되는 기간(초)입니다. 기본값은 2입니다.

13

배포를 만들기 전에 풀 시크릿을 생성합니다. 그렇지 않으면 배포를 생성할 때 오류가 발생합니다.

### 12.3. 가상 IP 주소 정보

**keepalived**는 가상 IP 주소 집합(VIP)을 관리합니다. 관리자는 다음 주소를 모두 확인해야 합니다.

- 클러스터 외부에서 구성된 호스트에서 액세스할 수 있습니다.
- 클러스터 내의 다른 용도로는 사용되지 않습니다.

각 노드의 **keepalive**는 필요한 서비스가 실행 중인지 여부를 결정합니다. 이 경우 **VIP**가 지원되고 **Keepalived**가 협상에 참여하여 **VIP**를 제공하는 노드를 결정합니다. 노드가 참여하려면 **VIP**의 감시 포트에서 서비스를 수신 대기하거나 검사를 비활성화해야 합니다.



참고

세트의 각 **VIP**는 다른 노드에서 제공할 수 있습니다.

### 12.4. 검사 구성 및 스크립트 알림

**keepalived**는 사용자가 제공한 선택적 검사 스크립트를 주기적으로 실행하여 애플리케이션의 상태를 모니터링합니다. 예를 들어 스크립트는 요청을 발행하고 응답을 확인하여 웹 서버를 테스트할 수 있습니다.

검사 스크립트를 제공하지 않으면 **TCP** 연결을 테스트하는 간단한 기본 스크립트가 실행됩니다. 이 기본 테스트는 모니터 포트가 **0**이면 비활성화됩니다.

각 IP 페일오버 pod는 pod가 실행 중인 노드에서 하나 이상의 가상 IP(VIP)를 관리하는 **Keepalived** 데몬을 관리합니다. **Keepalived** 데몬은 해당 노드의 각 VIP 상태를 유지합니다. 특정 노드의 특정 VIP는 **master**, **backup**, **fault** 상태일 수 있습니다.

**master** 상태에 있는 노드에서 해당 VIP에 대한 검사 스크립트가 실패하면 해당 노드의 VIP가 **fault** 상태가 되어 재협상을 트리거합니다. 재협상하는 동안 **fault** 상태에 있지 않은 노드의 모든 VIP가 VIP를 인수하는 노드를 결정하는 데 참여합니다. 결과적으로 VIP는 일부 노드에서 **master** 상태로 전환되고 VIP는 다른 노드에서 **backup** 상태로 유지됩니다.

**backup** 상태의 VIP 노드가 실패하면 해당 노드의 VIP가 **fault** 상태가 됩니다. 검사 스크립트가 **fault** 상태의 노드에서 VIP를 다시 전달하면 해당 노드의 VIP 상태가 **fault** 상태를 종료하고 **master** 상태로 전환하도록 협상합니다. 그런 다음 해당 노드의 VIP는 **master** 또는 **backup** 상태에 들어갈 수 있습니다.

클러스터 관리자는 상태가 변경될 때마다 호출되는 선택적 알림 스크립트를 제공할 수 있습니다. **keepalived**는 다음 세 개의 매개변수를 스크립트에 전달합니다.

- **\$1 - group 또는 instance**
- **\$2 - group 또는 instance 이름**
- **\$3 - 새 상태: master, backup 또는 fault**

검사 및 알림 스크립트가 IP 페일오버 Pod에서 실행되고 호스트 파일 시스템이 아닌 Pod 파일 시스템을 사용합니다. 그러나 IP 페일오버 Pod를 사용하면 **/hosts** 마운트 경로에서 호스트 파일 시스템을 사용할 수 있습니다. 검사 또는 알림 스크립트를 구성할 때 스크립트의 전체 경로를 제공해야 합니다. 스크립트를 제공하는 데 권장되는 접근 방식은 구성 맵을 사용하는 것입니다.

**Keepalived**가 시작될 때마다 로드되는 검사 및 알림 스크립트의 전체 경로 이름이 **Keepalived** 구성 파일인 **\_etc/keepalived/keepalived.conf**에 추가됩니다. 스크립트는 다음과 같이 구성 맵을 사용하여 Pod에 추가할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.

- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

프로세스

1. 원하는 스크립트를 생성하고 해당 스크립트를 유지할 구성 맵을 생성합니다. 스크립트에는 입력 인수가 없으며 **OK**의 경우 **0**을 **fail**의 경우 **1**을 반환해야 합니다.

검사 스크립트, **mycheckscript.sh**:

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

2. **config map**을 생성합니다.

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

3. **pod**에 스크립트를 추가합니다. 마운트된 구성 맵 파일의 **defaultMode**는 **oc** 명령을 사용하거나 배포 구성을 편집하여 실행할 수 있어야 합니다. **0755, 493 10**진수 값이 일반적입니다.

```
$ oc set env deploy/ipfailover-keepalived \
  OPENSIFT_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}}'
```



참고

**oc set env** 명령은 공백 문자를 구분합니다. = 기호 양쪽에 공백이 없어야 합니다.

작은 정보

또는 `ipfailover-keepalived` 배포 구성을 편집할 수 있습니다.

```
$ oc edit deploy ipfailover-keepalived

spec:
  containers:
  - env:
    - name: OPENSIFT_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

`spec.container.env` 필드에서 마운트된 스크립트 파일을 가리키도록 `OPENSIFT_HA_CHECK_SCRIPT` 환경 변수를 추가합니다.

2

`spec.container.volumeMounts` 필드를 추가하여 마운트 지점을 생성합니다.

3

새 `spec.volumes` 필드를 추가하여 구성 맵을 언급합니다.

4

파일에 대한 실행 권한을 설정합니다. 다시 읽으면 10진수 493으로 표시됩니다.

변경 사항을 저장하고 편집기를 종료합니다. 이렇게 하면 `ipfailover-keepalived`가 다시 시작됩니다.

## 12.5. VRRP 선점 구성

노드의 가상 IP(VIP)가 검사 스크립트를 전달하여 `fault` 상태를 벗어나면 노드의 VIP가 현재 `master` 상태에 있는 노드의 VIP보다 우선 순위가 낮은 경우 `backup` 상태가 됩니다. 그러나 `fault` 상태를 벗어나는

노드의 **VIP**가 우선 순위가 더 높은 경우 선점 전략이 클러스터에서 해당 역할을 결정합니다.

**nopreempt** 전략에서는 호스트의 우선 순위가 낮은 **VIP**에서 호스트의 우선 순위가 높은 **VIP**로 **master**를 이동하지 않습니다. **preempt\_delay 300**을 사용하면 기본값인 **Keepalived**가 지정된 **300**초 동안 기다린 후 **fault**를 호스트의 우선 순위 **VIP**로 이동합니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.

프로세스

- 선점 기능을 지정하려면 **oc edit deploy ipfailover-keepalived**를 입력하여 라우터 배포 구성을 편집합니다.

```
$ oc edit deploy ipfailover-keepalived
...
spec:
  containers:
  - env:
    - name: OPENSIFT_HA_PREEMPTION 1
      value: preempt_delay 300
...
1
```

**OPENSIFT\_HA\_PREEMPTION** 값을 설정합니다.

- **preempt\_delay 300**: **Keepalived**는 지정된 **300**초 동안 기다린 후 호스트의 우선 순위가 높은 **VIP**로 **master**를 이동합니다. 이는 기본값입니다.
- **nopreempt**: 더 낮은 우선 순위 호스트에서 더 높은 우선 순위 호스트로 **master**를 이동하지 않습니다.

12.6. VRRP ID 오프셋 정보

**IP** 페일오버 배포 구성에서 관리하는 각 **IP** 페일오버 **pod**는 노드 또는 복제본당 **1**개의 **Pod**를 실행하고 **Keepalived** 데몬을 실행합니다. 더 많은 **IP** 페일오버 배포 구성이 설정되면 더 많은 **Pod**가 생성되고 더

많은 데몬이 일반 VRRP(Virtual Router Redundancy Protocol) 협상에 연결됩니다. 이 협상은 모든 Keepalived 데몬에서 수행되며 어떤 노드가 어떤 VIP(가상 IP)를 서비스할 지 결정합니다.

내부적으로 Keepalived는 각 VIP에 고유한 vrrp-id를 할당합니다. 협상은 이 vrrp-id 세트를 사용하며, 결정이 내려지면 vrrp-id에 해당하는 VIP가 노드에 제공됩니다.

따라서 IP 페일오버 배포 구성에 정의된 모든 VIP에 대해 IP 페일오버 Pod에서 해당 vrrp-id를 할당해야 합니다. 이 작업은 OPENSIFT\_HA\_VRRP\_ID\_OFFSET에서 시작하고 vrrp-ids를 VIP 목록에 순차적으로 할당하여 수행됩니다. vrrp-ids는 1..255 범위의 값이 있을 수 있습니다.

IP 페일오버 배포 구성이 여러 개인 경우 배포 구성의 VIP 수를 늘리고 vrrp-id 범위가 겹치지 않도록 OPENSIFT\_HA\_VRRP\_ID\_OFFSET을 지정해야 합니다.

## 12.7. 254개 이상의 주소에 대한 IP 페일오버 구성

IP 페일오버 관리는 254개의 가상 IP(VIP) 주소 그룹으로 제한됩니다. 기본적으로 OpenShift Container Platform은 각 그룹에 하나의 IP 주소를 할당합니다. OPENSIFT\_HA\_VIP\_GROUPS 변수를 사용하여 이를 변경하여 여러 IP 주소가 각 그룹에 속하도록 하고 IP 페일오버를 구성할 때 각 VRRP(Virtual Router Redundancy Protocol) 인스턴스에 사용 가능한 VIP 그룹 수를 정의할 수 있습니다.

VIP 그룹화는 VRRP 페일오버 이벤트의 경우 VRRP당 VIP의 할당 범위가 넓어지며 클러스터의 모든 호스트가 로컬에서 서비스에 액세스할 수 있는 경우에 유용합니다. 예를 들어 서비스가 ExternalIP를 사용하여 노출되는 경우입니다.

### 참고

페일오버에 대한 규칙으로 라우터와 같은 서비스를 하나의 특정 호스트로 제한하지 마십시오. 대신 IP 페일오버의 경우 새 호스트에서 서비스를 다시 생성할 필요가 없도록 각 호스트에 서비스를 복제해야 합니다.

### 참고

OpenShift Container Platform 상태 확인을 사용하는 경우 IP 페일오버 및 그룹의 특성으로 인해 그룹의 모든 인스턴스가 확인되지 않습니다. 따라서 서비스가 활성화되어 있는지 확인하려면 [Kubernetes 상태 점검](#)을 사용해야 합니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

프로세스

- 각 그룹에 할당된 IP 주소 수를 변경하려면 **OPENSIFT\_HA\_VIP\_GROUPS** 변수의 값을 변경합니다. 예를 들면 다음과 같습니다.

IP 페일오버 구성을 위한 Deployment YAML의 예

```

...
spec:
  env:
    - name: OPENSIFT_HA_VIP_GROUPS 1
      value: "3"
...

```

1

7개의 VIP가 있는 환경에서 **OPENSIFT\_HA\_VIP\_GROUPS**가 3으로 설정된 경우 3개의 그룹을 생성하여 3개의 VIP를 첫 번째 그룹에 할당하고 2개의 VIP를 나머지 2개의 그룹에 할당합니다.



참고

**OPENSIFT\_HA\_VIP\_GROUPS**로 설정된 그룹 수가 페일오버로 설정된 IP 주소 수보다 적으면 그룹에는 두 개 이상의 IP 주소가 포함되어 있으며 모든 주소가 하나의 단위로 이동합니다.

### 12.8. INGRESSIP의 고가용성

클라우드 이외의 클러스터에서 서비스에 대한 IP 페일오버 및 **ingressIP**를 결합할 수 있습니다. 그 결과 **ingressIP**를 사용하여 서비스를 생성하는 사용자를 위한 고가용성 서비스가 생성됩니다.

사용 방법은 **ingressIPNetworkCIDR** 범위를 지정한 다음 **ipfailover** 구성을 생성할 때 동일한 범위를 사용하는 것입니다.



IP 페일오버는 전체 클러스터에 대해 최대 255개의 VIP를 지원할 수 있으므로 `ingressIPNetworkCIDR`은 /24 이하이어야 합니다.

## 12.9. IP 페일오버 제거

IP 페일오버가 처음 구성되면 클러스터의 작업자 노드는 `Keepalived`에 대해 224.0.0.18의 멀티 캐스트 패킷을 명시적으로 허용하는 `iptables` 규칙을 사용하여 수정됩니다. 노드를 변경하여 IP 페일오버를 제거하려면 `iptables` 규칙을 제거하고 `Keepalived`에서 사용하는 가상 IP 주소를 제거하는 작업을 실행해야 합니다.

### 절차

1. 선택 사항: 구성 맵으로 저장된 점검 및 알람 스크립트를 식별하고 삭제합니다.
  - a. IP 페일오버에 대한 Pod가 구성 맵을 볼륨으로 사용하는지 여부를 확인합니다.

```
$ 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}"
```

### 출력 예

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

- b. 이전 단계에서 볼륨으로 사용되는 구성 맵의 이름을 제공한 경우 구성 맵을 삭제합니다.

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

2.

**IP 페일오버를 위한 기존 배포를 식별합니다.**

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

출력 예

```

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

```

3.

**배포를 삭제합니다.**

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

4.

**ipfailover 서비스 계정을 제거합니다.**

```
$ oc delete sa ipfailover
```

5.

**IP 페일오버를 처음 구성할 때 추가된 IP 테이블 규칙을 제거하는 작업을 실행합니다.**

a.

다음 예와 유사한 콘텐츠를 사용하여 **remove-ipfailover-job.yaml**과 같은 파일을 생성합니다.

```

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: quay.io/openshift/origin-keepalived-ipfailover:4.11
          command: ["/var/lib/ipfailover/keepalived/remove-failover.sh"]

```

```
nodeSelector:
  kubernetes.io/hostname: <host_name> <.>
restartPolicy: Never
```

<.> IP 페일오버용으로 구성된 클러스터의 각 노드에 대해 작업을 실행하고 매번 호스트 이름을 바꿉니다.

- b. 작업을 실행합니다.

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

출력 예

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

검증

- 작업이 IP 페일오버의 초기 구성을 제거했는지 확인합니다.

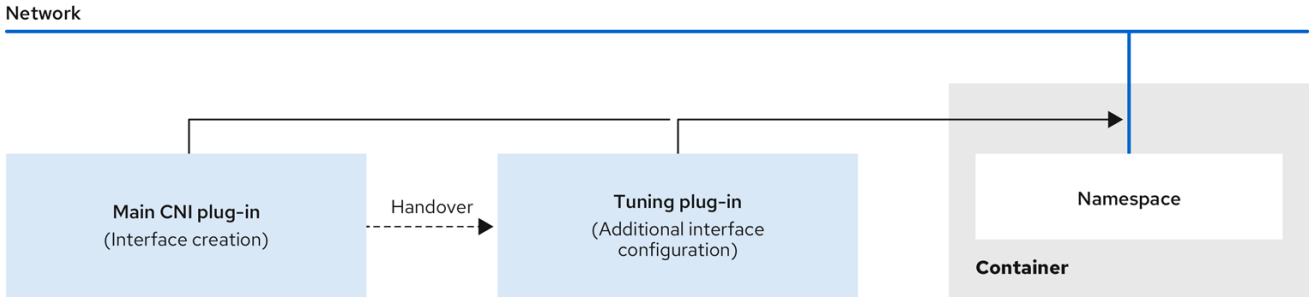
```
$ oc logs job/remove-ipfailover-2h8dm
```

출력 예

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

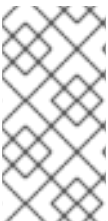
### 13장. 인터페이스 수준 네트워크 SYSCTL 구성

Linux에서 **sysctl**을 사용하면 관리자가 런타임에 커널 매개변수를 수정할 수 있습니다. 튜닝 **CNI(Container Network Interface)** 메타 플러그인을 사용하여 인터페이스 수준 네트워크 **sysctl**을 수정할 수 있습니다. 튜닝 **CNI** 메타 플러그인은 설명된 대로 기본 **CNI** 플러그인이 있는 체인에서 작동합니다.



264\_OpenShift\_0722

기본 **CNI** 플러그인은 인터페이스를 할당하고 런타임 시 튜닝 **CNI** 메타 플러그인에 이를 전달합니다. 튜닝 **CNI** 메타 플러그인을 사용하여 네트워크 네임스페이스에서 일부 **sysctl** 및 여러 인터페이스 속성(프로미스 모드, **all-multicast** 모드, **MTU** 및 **MAC** 주소)을 변경할 수 있습니다. 튜닝 **CNI** 메타 플러그인 구성에서 인터페이스 이름은 **IFNAME** 토큰으로 표시되고 런타임 시 인터페이스 이름으로 교체됩니다.



#### 참고

**OpenShift Container Platform**에서 튜닝 **CNI** 메타 플러그인은 인터페이스 수준 네트워크 **sysctl** 변경만 지원합니다.

#### 13.1. 튜닝 CNI 구성

다음 절차에서는 인터페이스 수준 네트워크 **net.ipv4.conf.IFNAME.accept\_redirects sysctl**을 변경하기 위해 튜닝 **CNI**를 구성합니다. 이 예제에서는 **ICMP** 리더렉션 패킷을 수락하고 전송할 수 있습니다.

#### 프로세스

1. 다음 콘텐츠를 사용하여 **tuning-example.yaml** 과 같은 네트워크 연결 정의를 생성합니다.

```
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

생성할 추가 네트워크 연결의 이름을 지정합니다. 이름은 지정된 네임스페이스 내에서 고유해야 합니다.

②

개체가 연결된 네임스페이스를 지정합니다.

③

CNI 사양 버전을 지정합니다.

④

구성의 이름을 지정합니다. 구성 이름을 네트워크 연결 정의의 **name** 값과 일치시키는 것이 좋습니다.

⑤

구성할 기본 CNI 플러그인의 이름을 지정합니다.

⑥

CNI 메타 플러그인의 이름을 지정합니다.

⑦

설정할 **sysctl**을 지정합니다.

**yaml** 파일의 예는 다음과 같습니다.

```

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

```

2. 다음 명령을 실행하여 **yaml**을 적용합니다.

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

출력 예

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

3. 다음과 유사한 네트워크 연결 정의를 사용하여 **examplepod.yaml** 과 같은 **Pod**를 생성합니다.

```

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

구성된 `NetworkAttachmentDefinition` 의 이름을 지정합니다.

2

`Run AsUser`는 컨테이너가 실행되는 사용자 ID를 제어합니다.

3

`runAsGroup` 은 컨테이너가 실행되는 기본 그룹 ID를 제어합니다.

4

`allowPrivilegeEscalation` 은 Pod에서 권한 에스컬레이션을 허용하도록 요청할 수 있는지 여부를 결정합니다. 지정되지 않은 경우 기본값은 `true`입니다. 이 부울은 컨테이너 프로세스에 `no_new_privs` 플래그가 설정되는지 여부를 직접 제어합니다.

5

기능을 사용하면 전체 루트 액세스 권한을 부여하지 않고 권한 있는 작업을 수행할 수 있습니다. 이 정책은 모든 기능이 Pod에서 삭제되도록 합니다.

6

`runAsNonRoot: true` 를 사용하려면 컨테이너가 0이 아닌 다른 UID와 함께 컨테이너를 실행해야 합니다.

7

`RuntimeDefault` 는 Pod 또는 컨테이너 워크로드에 대한 기본 `seccomp` 프로필을 활성화합니다.

4.

다음 명령을 실행하여 `yaml`을 적용합니다.

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

5. 다음 명령을 실행하여 Pod가 생성되었는지 확인합니다.

```
$ oc get pod
```

출력 예

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

6. 다음 명령을 실행하여 Pod에 로그인합니다.

```
$ oc rsh tunepod
```

7. 구성된 `sysctl` 플래그 값을 확인합니다. 예를 들어 다음 명령을 실행하여 `net.ipv4.conf.net1.accept_redirects` 값을 찾습니다.

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

예상 출력

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

## 13.2. 추가 리소스

- [컨테이너의 sysctl 사용](#)



## 14장. 베어 메탈 클러스터에서 SCTP(STREAM CONTROL TRANSMISSION PROTOCOL) 사용

클러스터 관리자는 클러스터에서 SCTP(Stream Control Transmission Protocol)를 사용할 수 있습니다.

### 14.1. OPENSIFT CONTAINER PLATFORM에서의 SCTP(스트림 제어 전송 프로토콜)

클러스터 관리자는 클러스터의 호스트에서 SCTP를 활성화 할 수 있습니다. RHCOS(Red Hat Enterprise Linux CoreOS)에서 SCTP 모듈은 기본적으로 비활성화되어 있습니다.

SCTP는 IP 네트워크에서 실행되는 안정적인 메시지 기반 프로토콜입니다.

활성화하면 Pod, 서비스, 네트워크 정책에서 SCTP를 프로토콜로 사용할 수 있습니다. type 매개변수를 ClusterIP 또는 NodePort 값으로 설정하여 Service를 정의해야 합니다.

#### 14.1.1. SCTP 프로토콜을 사용하는 구성의 예

protocol 매개변수를 포트 또는 서비스 오브젝트의 SCTP 값으로 설정하여 SCTP를 사용하도록 포트 또는 서비스를 구성할 수 있습니다.

다음 예에서는 pod가 SCTP를 사용하도록 구성되어 있습니다.

```
apiVersion: v1
kind: Pod
metadata:
  namespace: project1
  name: example-pod
spec:
  containers:
    - name: example-pod
    ...
  ports:
    - containerPort: 30100
      name: sctpserver
      protocol: SCTP
```

다음 예에서는 서비스가 SCTP를 사용하도록 구성되어 있습니다.

```
apiVersion: v1
kind: Service
```

```

metadata:
  namespace: project1
  name: sctpserver
spec:
  ...
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30100
      targetPort: 30100
      type: ClusterIP

```

다음 예에서 **NetworkPolicy** 오브젝트는 특정 레이블이 있는 모든 **Pod**의 포트 80에서 **SCTP** 네트워크 트래픽에 적용되도록 구성되어 있습니다.

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-sctp-on-http
spec:
  podSelector:
    matchLabels:
      role: web
  ingress:
    - ports:
      - protocol: SCTP
        port: 80

```

## 14.2. SCTP(스트림 제어 전송 프로토콜) 활성화

클러스터 관리자는 클러스터의 작업자 노드에 블랙리스트 **SCTP** 커널 모듈을 로드하고 활성화할 수 있습니다.

### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

### 프로세스

1. 다음 **YAML** 정의가 포함된 **load-sctp-module.yaml** 파일을 생성합니다.

```

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

```

2.

**MachineConfig** 오브젝트를 생성하려면 다음 명령을 입력합니다.

```
$ oc create -f load-sctp-module.yaml
```

3.

선택 사항: **MachineConfig Operator**가 구성 변경 사항을 적용하는 동안 노드의 상태를 보려면 다음 명령을 입력합니다. 노드 상태가 **Ready**로 전환되면 구성 업데이트가 적용됩니다.

```
$ oc get nodes
```

### 14.3. SCTP(STREAM CONTROL TRANSMISSION PROTOCOL)의 활성화 여부 확인

**SCTP** 트래픽을 수신하는 애플리케이션으로 **pod**를 만들고 서비스와 연결한 다음, 노출된 서비스에 연결하여 **SCTP**가 클러스터에서 작동하는지 확인할 수 있습니다.

#### 사전 요구 사항

- 클러스터에서 인터넷에 액세스하여 **nc** 패키지를 설치합니다.
- **OpenShift CLI(oc)**를 설치합니다.

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

## 프로세스

1. **SCTP** 리스너를 시작하는 포드를 생성합니다.

- a. 다음 **YAML**로 **pod**를 정의하는 **sctp-server.yaml** 파일을 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpserver
  labels:
    app: sctpserver
spec:
  containers:
    - name: sctpserver
      image: registry.access.redhat.com/ubi8/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
      ports:
        - containerPort: 30102
          name: sctpserver
          protocol: SCTP
```

- b. 다음 명령을 입력하여 **pod**를 생성합니다.

```
$ oc create -f sctp-server.yaml
```

2. **SCTP** 리스너 **pod**에 대한 서비스를 생성합니다.

- a. 다음 **YAML**을 사용하여 서비스를 정의하는 **sctp-service.yaml** 파일을 생성합니다.

```
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. 서비스를 생성하려면 다음 명령을 입력합니다.

```
$ oc create -f sctp-service.yaml
```

3. **SCTP** 클라이언트에 대한 **pod**를 생성합니다.

- a. 다음 **YAML**을 사용하여 **sctp-client.yaml** 파일을 만듭니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: sctpclient
  labels:
    app: sctpclient
spec:
  containers:
    - name: sctpclient
      image: registry.access.redhat.com/ubi8/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]

```

- b. **Pod** 오브젝트를 생성하려면 다음 명령을 입력합니다.

```
$ oc apply -f sctp-client.yaml
```

4. 서버에서 **SCTP** 리스너를 실행합니다.

- a. 서버 **Pod**에 연결하려면 다음 명령을 입력합니다.

```
$ oc rsh sctpserver
```

- b. **SCTP** 리스너를 시작하려면 다음 명령을 입력합니다.

■

```
$ nc -l 30102 --sctp
```

5.

서버의 **SCTP** 리스너에 연결합니다.

a.

터미널 프로그램에서 새 터미널 창 또는 탭을 엽니다.

b.

**sctp** 서비스의 IP 주소를 얻습니다. 다음 명령을 실행합니다.

```
$ oc get services sctp -o go-template='{{.spec.clusterIP}}{\n}'
```

c.

클라이언트 Pod에 연결하려면 다음 명령을 입력합니다.

```
$ oc rsh sctpclient
```

d.

**SCTP** 클라이언트를 시작하려면 다음 명령을 입력합니다. <cluster\_IP>를 **sctp** 서비스의 클러스터 IP 주소로 변경합니다.

```
# nc <cluster_IP> 30102 --sctp
```

## 15장. PTP 하드웨어 사용

**OpenShift Container Platform** 클러스터 노드에서 **linuxptp** 서비스를 구성하고 **PTP** 지원 하드웨어를 사용할 수 있습니다.

### 15.1. PTP 하드웨어 정보

**PTP Operator**를 배포하여 **OpenShift Container Platform** 콘솔 또는 **OpenShift CLI (oc)**를 사용하여 **PTP**를 설치할 수 있습니다. **PTP Operator**는 **linuxptp** 서비스를 생성 및 관리하고 다음 기능을 제공합니다.

- 클러스터에서 **PTP** 가능 장치 검색.
- **linuxptp** 서비스의 구성 관리.
- **PTP Operator cloud-event-proxy** 사이드카를 사용하여 애플리케이션의 성능 및 안정성에 부정적인 영향을 주는 **PTP** 클록 이벤트 알림



#### 참고

**PTP Operator**는 베어 메탈 인프라에서만 프로비저닝된 클러스터에서 **PTP** 가능 장치와 함께 작동합니다.

### 15.2. PTP 정보

**PTP(Precision Time Protocol)**는 네트워크에서 클럭을 동기화하는 데 사용됩니다. 하드웨어 지원과 함께 사용할 경우 **PTP**는 마이크로초 미만의 정확성을 수행할 수 있으며 **NTP(Network Time Protocol)**보다 더 정확합니다.

**linuxptp** 패키지에는 클럭 동기화를 위한 **ptp4l** 및 **phc2sys** 프로그램이 포함되어 있습니다. **ptp4l**은 **PTP** 경계 클럭과 일반 클럭을 구현합니다. **ptp4l**은 하드웨어 타임스탬프를 사용하여 **PTP** 하드웨어 클럭을 소스 클럭에 동기화하고 소프트웨어 타임스탬프를 사용하여 시스템 클럭을 소스 클럭에 동기화합니다. **phc2sys**는 하드웨어 타임스탬프에 **NIC(네트워크 인터페이스 컨트롤러)**의 **PTP** 하드웨어 클럭에 동기화하는 데 사용됩니다.

#### 15.2.1. PTP 도메인의 요소

**PTP**는 네트워크에 연결된 여러 노드를 각 노드의 클럭과 동기화하는 데 사용됩니다. **PTP**에 의해 동기화된 클럭은 소스 대상 계층 구조로 구성됩니다. 계층 구조는 모든 클럭에서 실행되는 최상의 마스터 시계 (**BMC**) 알고리즘에 의해 자동으로 생성 및 업데이트됩니다. 대상 클럭은 소스 클럭에 동기화되며 대상 클럭은 다른 다운스트림 시계의 소스가 될 수 있습니다. 다음 유형의 클럭을 구성에 포함할 수 있습니다.

### GRandmaster 클럭

마스터 클럭은 네트워크의 다른 클럭에 표준 시간 정보를 제공하며 정확하고 안정적인 동기화를 보장합니다. 타임스탬프를 작성하고 다른 클럭의 시간 요청에 응답합니다. **Grandmaster** 시계를 **GPS(Global Positioning System)** 시간 소스에 동기화할 수 있습니다.

### 일반 클럭

일반 클럭에는 네트워크의 위치에 따라 소스 또는 대상 클럭의 역할을 수행할 수 있는 단일 포트가 연결되어 있습니다. 일반 클럭은 타임스탬프를 읽고 쓸 수 있습니다.

### 경계 클럭

경계 클럭에는 두 개 이상의 통신 경로에 포트가 있으며, 동시에 소스와 다른 대상 클럭의 대상일 수 있습니다. 경계 클럭은 대상 클럭으로 작동합니다. 대상 클럭이 타이밍 메시지를 수신하고 지연을 조정하여 다음 네트워크를 전달하기 위한 새 소스 시간 신호를 생성합니다. 경계 클럭은 소스 클럭과 정확하게 동기화되는 새로운 타이밍 패킷을 생성하며 소스 클럭에 직접 보고하는 연결된 장치의 수를 줄일 수 있습니다.

## 15.2.2. NTP를 통한 PTP의 이점

**PTP**가 **NTP**를 능가하는 주요 이점 중 하나는 다양한 **NIC**(네트워크 인터페이스 컨트롤러) 및 네트워크 스위치에 있는 하드웨어 지원입니다. 특수 하드웨어를 사용하면 **PTP**가 메시지 전송 지연을 고려하여 시간 동기화의 정확성을 향상시킬 수 있습니다. 최대한의 정확성을 달성하려면 **PTP** 클럭 사이의 모든 네트워킹 구성 요소를 **PTP** 하드웨어를 사용하도록 설정하는 것이 좋습니다.

**NIC**는 전송 및 수신 즉시 **PTP** 패킷을 타임스탬프할 수 있으므로 하드웨어 기반 **PTP**는 최적의 정확성을 제공합니다. 이를 운영 체제에서 **PTP** 패킷을 추가로 처리해야 하는 소프트웨어 기반 **PTP**와 비교합니다.



### 중요

**PTP**를 활성화하기 전에 필수 노드에 대해 **NTP**가 비활성화되어 있는지 확인합니다. **MachineConfig** 사용자 정의 리소스를 사용하여 **chrony** 타임 서비스 (**chronyd**)를 비활성화할 수 있습니다. 자세한 내용은 [chrony 타임 서비스 비활성화](#)를 참조하십시오.

## 15.2.3. 듀얼 NIC 하드웨어에서 PTP 사용



**OpenShift Container Platform**은 클러스터에서 정밀한 **PTP** 타이밍을 위해 단일 및 듀얼 **NIC** 하드웨어를 지원합니다.

대역 중 사양을 제공하는 **5G** 통신망의 경우 각 가상 분산 장치(**vDU**)는 **6**개의 무선 단위(**RU**)에 연결되어 있어야 합니다. 이러한 연결을 수행하기 위해 각 **vDU** 호스트에 경계 시계로 구성된 **2**개의 **NIC**가 필요합니다.

듀얼 **NIC** 하드웨어를 사용하면 각 **NIC**가 다운스트림 클럭을 공급하는 별도의 **ptp4l** 인스턴스와 함께 각 **NIC**를 동일한 업스트림 리더 시계에 연결할 수 있습니다.

### 15.3. CLI를 사용하여 PTP OPERATOR 설치

클러스터 관리자는 **CLI**를 사용하여 **Operator**를 설치할 수 있습니다.

#### 사전 요구 사항

- **PTP**를 지원하는 하드웨어가 있는 노드로 베어 메탈 하드웨어에 설치된 클러스터
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. **PTP Operator**의 네임스페이스를 생성합니다.
  - a. 다음 **YAML**을 **ptp-namespace.yaml** 파일에 저장합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    name: openshift-ptp
    openshift.io/cluster-monitoring: "true"
```

- b. 네임스페이스 **CR**을 생성합니다.

```
$ oc create -f ptp-namespace.yaml
```

2. **PTP Operator**를 위한 **Operator** 그룹을 생성합니다.

- a. 다음 **YAML**을 **ptp-operatorgroup.yaml** 파일에 저장합니다.

```
apiVersion: operators.coreos.com/v1  
kind: OperatorGroup  
metadata:  
  name: ptp-operators  
  namespace: openshift-ptp  
spec:  
  targetNamespaces:  
  - openshift-ptp
```

- b. **OperatorGroup CR**을 생성합니다.

```
$ oc create -f ptp-operatorgroup.yaml
```

3. **PTP Operator**에 등록합니다.

- a. 다음 **YAML**을 **ptp-sub.yaml** 파일에 저장합니다.

```
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. **서브스크립션 CR**을 생성합니다.

```
$ oc create -f ptp-sub.yaml
```

4.

**Operator**가 설치되었는지 확인하려면 다음 명령을 입력합니다.

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

출력 예

Name	Phase
4.12.0-202301261535	Succeeded

#### 15.4. 웹 콘솔을 사용하여 PTP OPERATOR 설치

클러스터 관리자는 웹 콘솔을 사용하여 **PTP Operator**를 설치할 수 있습니다.



참고

이전 섹션에서 언급한 것처럼 네임스페이스 및 **Operator group**을 생성해야 합니다.

프로세스

1.

**OpenShift Container Platform** 웹 콘솔을 사용하여 **PTP Operator**를 설치합니다.

a.

**OpenShift Container Platform** 웹 콘솔에서 **Operator** → **OperatorHub**를 클릭합니다.

b.

사용 가능한 **Operator** 목록에서 **PTP Operator**를 선택한 다음 설치를 클릭합니다.

c.

**Operator** 설치 페이지의 클러스터의 특정 네임스페이스에서 **openshift-ntp**를 선택합니다. 그런 다음, 설치를 클릭합니다.

2.

선택 사항: **PTP Operator**가 설치되었는지 확인합니다.

- a. **Operator** → 설치된 **Operator** 페이지로 전환합니다.
- b. **PTP Operator**가 **openshift-ptp** 프로젝트에 **InstallSucceeded** 상태로 나열되어 있는지 확인합니다.



참고

설치 중에 **Operator**는 실패 상태를 표시할 수 있습니다. 나중에 **InstallSucceeded** 메시지와 함께 설치에 성공하면 이 실패 메시지를 무시할 수 있습니다.

**Operator**가 설치된 것으로 나타나지 않으면 다음과 같이 추가 문제 해결을 수행합니다.

- **Operator** → 설치된 **Operator** 페이지로 이동하고 **Operator** 서브스크립션 및 설치 계획 탭의 상태에 장애나 오류가 있는지 검사합니다.
- **Workloads** → **Pod** 페이지로 이동하여 **openshift-ptp** 프로젝트에서 **Pod** 로그를 확인합니다.

### 15.5. PTP 장치 구성

**PTP Operator**는 **NodePtpDevice.ptp.openshift.io CRD(custom resource definition)**를 **OpenShift Container Platform**에 추가합니다.

**PTP Operator**는 각 노드에서 **PTP** 가능 네트워크 장치를 클러스터에서 검색합니다. 호환 가능한 **PTP** 가능 네트워크 장치를 제공하는 각 노드에 대해 **NodePtpDevice CR(사용자 정의 리소스)** 오브젝트를 생성하고 업데이트합니다.

#### 15.5.1. 클러스터에서 PTP 가능 네트워크 장치 검색

- 클러스터에서 **PTP** 가능 네트워크 장치의 전체 목록을 반환하려면 다음 명령을 실행합니다.

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

출력 예

```

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**

**name** 매개변수의 값은 상위 노드의 이름과 동일합니다.

**2**

**devices** 컬렉션에는 **PTP Operator**가 노드를 검색할 **PTP** 지원 장치 목록이 포함됩니다.

### 15.5.2. linuxptp 서비스를 일반 클러스터로 구성

**PtpConfig CR**(사용자 정의 리소스) 오브젝트를 생성하여 **linuxptp** 서비스(**ptp4l,phc2sys**)를 일반 시계로 구성할 수 있습니다.



참고

다음 예제 **PtpConfig CR**을 기반으로 **linuxptp** 서비스를 특정 하드웨어 및 환경에 대한 일반 클럭으로 구성합니다. 이 예제 **CR**에서는 **ptp4IOpts**, **ptp4IConf** 및 **ptpClockThreshold**에 대한 적절한 값을 설정하여 **PTP** 빠른 이벤트를 구성합니다. **ptpClockThreshold**는 이벤트가 활성화된 경우에만 사용됩니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- **PTP Operator**를 설치합니다.

절차

1. 다음 **PtpConfig CR**을 생성한 다음 **YAML**을 **ordinary-clock-ptp-config.yaml** 파일에 저장합니다.

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ordinary-clock-ptp-config
  namespace: openshift-ptp
spec:
  profile:
    - name: "<profile_name>"
      interface: ""<interface_name>"
      ptp4IOpts: "-2 -s --summary_interval -4"
      phc2sysOpts: "-a -r -n 24"
      ptp4IConf: |
        [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  10
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
#
# 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

```

8

```

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
ptpSchedulingPolicy: SCHED_OTHER
ptpSchedulingPriority: 10
ptpClockThreshold:
holdOverTimeout:    5
maxOffsetThreshold: 100
minOffsetThreshold: -100
recommend:
- profile: "profile1"
priority: 0
match:
- nodeLabel: "node-role.kubernetes.io/worker"
  nodeName: "compute-0.example.com"

```

1

**PtpConfig CR의 이름입니다.**

2

하나 이상의 **profile** 오브젝트의 배열을 지정합니다.

3



4

**ptp4l** 서비스에서 사용할 네트워크 인터페이스를 지정합니다(예: **ens787f1** ).

5

**ptp4l** 서비스에 대한 시스템 구성 옵션을 지정합니다. 예를 들어 **-2** 는 **IEEE 802.3** 네트워크 전송을 선택합니다. 옵션은 네트워크 인터페이스 이름과 서비스 구성 파일이 자동으로 추가되므로 네트워크 인터페이스 이름 **-i <interface>** 및 서비스 구성 파일 **-f /etc/ptp4l.conf**를 포함하지 않아야 합니다. 이 인터페이스에서 **PTP** 빠른 이벤트를 사용하려면 **--summary\_interval -4** 를 추가합니다.

6

**phc2sys** 서비스에 대한 시스템 구성 옵션을 지정합니다. 이 필드가 비어 있으면 **PTP Operator**에서 **phc2sys** 서비스를 시작하지 않습니다. **Intel Coumbiaville 800** 시리즈 **NIC**의 경우 **phc2sysOpts** 옵션을 **-a -r -m -n 24 -N 8 -R 16** 으로 설정합니다. **-m** 은 **stdout** 에 메시지를 출력합니다. **linuxptp-daemon DaemonSet** 은 로그를 구문 분석하고 **Prometheus** 지표를 생성합니다.

7

기본 **/etc/ptp4l.conf** 파일을 대체할 구성이 포함된 문자열을 지정합니다. 기본 구성을 사용하려면 필드를 비워 둡니다.

8

**Intel Coumbiaville 800** 시리즈 **NIC**의 경우 **tx\_timestamp\_timeout** 을 **50** 으로 설정합니다.

9

**Intel Columbiaville 800** 시리즈 **NIC**의 경우 **boundary\_clock\_jbod** 를 **0** 으로 설정합니다.

10

**ptp4l** 및 **phc2sys** 프로세스에 대한 스케줄링 정책입니다. 기본값은 **-4.8\_OTHER** 입니다. **Havana** 스케줄링을 지원하는 시스템에서 **Retain\_VRF**를 사용합니다.

11

**ptpSchedulingPolicy** 가 **ECDHE\_FIFO**로 설정된 경우 **ptp4l** 및 **phc2sys** 프로세스의 **FIFO** 우선 순위를 설정하는 데 사용되는 **1-65**의 정수 값입니다. **ptpSchedulingPriority** 필드는 **ptpSchedulingPolicy** 가 **ECDHE\_OTHER** 로 설정된 경우 사용되지 않습니다.

12

13

프로필을 노드에 적용하는 방법에 대한 규칙을 정의하는 하나 이상의 **recommend** 오브젝트 배열을 지정합니다.

14

**profile** 섹션에 정의된 **profile** 오브젝트 이름을 지정합니다.

15

일반 시계의 경우 **priority** 를 **0** 으로 설정합니다.

16

**nodeLabel** 또는 **nodeName**으로 일치 규칙을 지정합니다.

17

**oc get nodes --show-labels** 명령을 사용하여 노드 오브젝트에서 **node.Labels** 의 키로 **nodeLabel** 을 지정합니다.

18

**oc get nodes** 명령을 사용하여 노드 오브젝트에서 **node.Name** 으로 **nodeName** 을 지정합니다.

2.

다음 명령을 실행하여 **PtpConfig CR**을 생성합니다.

```
$ oc create -f ordinary-clock-ptp-config.yaml
```

검증

1.

**PtpConfig** 프로필이 노드에 적용되었는지 확인합니다.

a.

다음 명령을 실행하여 **openshift-ptp** 네임스페이스에서 **Pod** 목록을 가져옵니다.

```
$ oc get pods -n openshift-ptp -o wide
```

출력 예

```

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.

프로필이 올바른지 확인합니다. **PtpConfig** 프로필에 지정한 노드에 해당하는 **linuxptp** 데몬의 로그를 검사합니다. 다음 명령을 실행합니다.

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ntp -c linuxptp-daemon-container
```

출력 예

```

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] Ptp4lOpts: -2 -s --
summary_interval -4
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] -----

```

추가 리소스

•

**PTP 하드웨어의 FIFO 우선 순위 스케줄링에 대한 자세한 내용은 PTP 하드웨어에 대한 FIFO 우선 순위 스케줄링 구성** 을 참조하십시오.

### 15.5.3. linuxptp 서비스를 경계 시계로 구성

**PtpConfig CR**(사용자 정의 리소스) 오브젝트를 생성하여 **linuxptp** 서비스(**ptp4l,phc2sys**)를 경계 시계로 구성할 수 있습니다.



참고

다음 예제 **PtpConfig CR**을 기반으로 **linuxptp** 서비스를 특정 하드웨어 및 환경에 대한 경계 클럭으로 구성합니다. 이 예제 **CR**은 또한 **ptp4IOpts**, **ptp4IConf**, **ptpClockThreshold** 에 적절한 값을 설정하여 **PTP** 빠른 이벤트를 구성합니다. **ptpClockThreshold** 는 이벤트가 활성화된 경우에만 사용됩니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- **PTP Operator**를 설치합니다.

절차

1. 다음 **PtpConfig CR**을 만든 다음 **YAML**을 **boundary-clock-ptp-config.yaml** 파일에 저장합니다.

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config
  namespace: openshift-ptp
spec:
  profile:
    - name: "<profile_name>"
      ptp4IOpts: "-2 --summary_interval -4"
      ptp4IConf: |
        [ens1f0]
        masterOnly 0
        [ens1f3]
        masterOnly 1
        [global]
        #
        # Default Data Set
        #
        twoStepFlag          1
        #slaveOnly           1
        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
#
# Runtime 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  10
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
#
# 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

```

8

```

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
phc2sysOpts:               "-a -r -n 24"
ptpSchedulingPolicy:        SCHED_OTHER
ptpSchedulingPriority:      10
ptpClockThreshold:         13
holdOverTimeout:           5
maxOffsetThreshold:         100
minOffsetThreshold:        -100
recommend:
- profile: "<profile_name>"
priority: 10
match:
- nodeLabel: "<node_label>"
  nodeName: "<node_name>"

```

1

PtpConfig CR의 이름입니다.

2

하나 이상의 **profile** 오브젝트의 배열을 지정합니다.

3

프로파일 오브젝트를 고유하게 식별하는 프로파일 오브젝트의 이름을 지정합니다.

4

**ptp4l** 서비스에 대한 시스템 구성 옵션을 지정합니다. 옵션은 네트워크 인터페이스 이름과 서비스 구성 파일이 자동으로 추가되므로 네트워크 인터페이스 이름 **-i <interface>** 및 서비스 구성 파일 **-f /etc/ptp4l.conf**를 포함하지 않아야 합니다.

5

**ptp4l**을 경계 클럭으로 시작하는 데 필요한 구성을 지정합니다. 예를 들어 **ens1f0** 은 그랜드 마스터 클럭에서 동기화되고 **ens1f3**은 연결된 장치를 동기화합니다.

6

동기화 클럭을 수신하는 인터페이스입니다.

7

동기화 시계를 전송하는 인터페이스입니다.

8

**Intel Coumbiaville 800** 시리즈 NIC의 경우 **tx\_timestamp\_timeout** 을 50 으로 설정합니다.

9

**Intel Columbiaville 800** 시리즈 NIC의 경우 **boundary\_clock\_jbod** 가 0 으로 설정되어 있는지 확인합니다. **Intel Fortville X710** 시리즈 NIC의 경우 **boundary\_clock\_jbod** 가 1 로 설정되어 있는지 확인합니다.

10

**phc2sys** 서비스에 대한 시스템 구성 옵션을 지정합니다. 이 필드가 비어 있으면 **PTP Operator**에서 **phc2sys** 서비스를 시작하지 않습니다.

11

**ptp4l** 및 **phc2sys** 프로세스에 대한 스케줄링 정책입니다. 기본값은 **-4.8\_OTHER** 입니다. **Havana** 스케줄링을 지원하는 시스템에서 **Retain\_VRF**를 사용합니다.

12

**ptpSchedulingPolicy** 가 **ECDHE\_FIFO**로 설정된 경우 **ptp4l** 및 **phc2sys** 프로세스의 **FIFO** 우선 순위를 설정하는 데 사용되는 1-65의 정수 값입니다. **ptpSchedulingPriority** 필드는 **ptpSchedulingPolicy** 가 **ECDHE\_OTHER** 로 설정된 경우 사용되지 않습니다.

13

선택 사항: **ptpClockThreshold** 스탠자가 없으면 **ptpClockThreshold** 필드에 기본값이 사용됩니다. 스탠자는 기본 **ptpClockThreshold** 값을 표시합니다. **ptpClockThreshold** 값은 **PTP** 이벤트가 트리거되기 전에 **PTP** 마스터 클럭이 연결 해제된 후의 기간을 구성합니다. **holdOverTimeout** 은 **PTP** 마스터 클럭의 연결이 끊어지면 **PTP** 클럭 이벤트 상태가 **FREERUN** 로 변경되기 전 시간(초)입니다. **maxOffsetThreshold** 및 **minOffsetThreshold** 설정은 **CLOCK\_REALTIME** (**phc2sys**) 또는 마스터 오프셋(**ptp4l**)의 값과 비교되는 나노초에 오프셋 값을 구성합니다. **ptp4l** 또는 **phc2sys** 오프셋 값이 이 범위를 벗어나는 경우 **PTP** 클럭 상태가 **FREERUN** 로 설정됩니다. 오프셋 값이 이 범위 내에 있으면 **PTP** 클럭 상태가 **LOCKED** 로 설정됩니다.

14

프로필을 노드에 적용하는 방법에 대한 규칙을 정의하는 하나 이상의 **recommend** 오브젝트 배열을 지정합니다.

15

**profile** 섹션에 정의된 **profile** 오브젝트 이름을 지정합니다.

16

0에서 99 사이의 정수 값으로 **priority**를 지정합니다. 숫자가 클수록 우선순위가 낮으므로 우선순위 99는 우선순위 10보다 낮습니다. **match** 필드에 정의된 규칙에 따라 여러 프로필과 노드를 일치시킬 수 있는 경우 우선 순위가 높은 프로필이 해당 노드에 적용됩니다.

17

**nodeLabel** 또는 **nodeName**으로 일치 규칙을 지정합니다.

18

**oc get nodes --show-labels** 명령을 사용하여 노드 오브젝트에서 **node.Labels** 의 키로 **nodeLabel** 을 지정합니다. 예: **node-role.kubernetes.io/worker**.

19

**oc get nodes** 명령을 사용하여 노드 오브젝트에서 **node.Name** 으로 **nodeName** 을 지정합니다. 예: **node-role.kubernetes.io/worker**. 예: **compute-0.example.com**.



2. 다음 명령을 실행하여 **CR**을 생성합니다.

```
$ oc create -f boundary-clock-ptp-config.yaml
```

## 검증

1. **PtpConfig** 프로필이 노드에 적용되었는지 확인합니다.

- a. 다음 명령을 실행하여 **openshift-ptp** 네임스페이스에서 **Pod** 목록을 가져옵니다.

```
$ oc get pods -n openshift-ptp -o wide
```

### 출력 예

```

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. 프로필이 올바른지 확인합니다. **PtpConfig** 프로필에 지정한 노드에 해당하는 **linuxptp** 데몬의 로그를 검사합니다. 다음 명령을 실행합니다.

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
```

### 출력 예

```

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 --summary_interval -

```

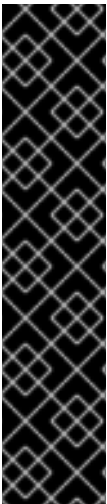
4

```
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] -----
```

추가 리소스

- **PTP 하드웨어의 FIFO 우선 순위 스케줄링에 대한 자세한 내용은 PTP 하드웨어에 대한 FIFO 우선 순위 스케줄링 구성 을 참조하십시오.**

15.5.4. 듀얼 NIC 하드웨어의 경계 시계로 linuxptp 서비스 구성



중요

경계 시계로 구성된 이중 NIC가 있는 PTP(Precision Time Protocol) 하드웨어는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에 서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사 용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

각 NIC에 대한 PtpConfig CR(사용자 정의 리소스) 오브젝트를 생성하여 이중 NIC 하드웨어의 경계 시계로 linuxptp 서비스(ntp4l,phc2sys)를 구성할 수 있습니다.

듀얼 NIC 하드웨어를 사용하면 각 NIC가 다운스트림 클럭을 공급하는 별도의 ntp4l 인스턴스와 함께 각 NIC를 동일한 업스트림 리더 시계에 연결할 수 있습니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- cluster-admin 권한이 있는 사용자로 로그인합니다.

• **PTP Operator**를 설치합니다.

## 절차

1. 각 NIC에 대해 하나씩 두 개의 개별 **PtpConfig CR**을 생성하고 각 CR의 기반으로 "**Linuxptp 서비스 구성**"의 참조 CR을 사용합니다. 예를 들어 다음과 같습니다.

- a. **phc2sysOpts**의 값을 지정하여 **boundary-clock-ptp-config-nic1.yaml**을 생성합니다.

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic1
  namespace: openshift-ptp
spec:
  profile:
    - name: "profile1"
      ptp4IOpts: "-2 --summary_interval -4"
      ptp4IConf: | ①
        [ens5f1]
        masterOnly 1
        [ens5f0]
        masterOnly 0
      ...
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ②
```

①

**ptp4I**을 경계 시계로 시작하는 데 필요한 인터페이스를 지정합니다. 예를 들어 **ens5f0**은 할머신 시계와 **ens5f1**이 연결된 장치를 동기화합니다.

②

필수 **phc2sysOpts** 값. **-m**은 **stdout**에 메시지를 출력합니다. **linuxptp-daemon DaemonSet**은 로그를 구문 분석하고 **Prometheus** 지표를 생성합니다.

- b. **boundary-clock-ptp-config-nic2.yaml**을 생성하고 **phc2sys** 필드를 완전히 제거하여 두 번째 NIC에 대해 **phc2sys** 서비스를 비활성화합니다.

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic2
  namespace: openshift-ptp
```

```
spec:
  profile:
    - name: "profile2"
      ptp4IOpts: "-2 --summary_interval -4"
      ptp4IConf: | 1
        [ens7f1]
        masterOnly 1
        [ens7f0]
        masterOnly 0
  ...
```

**1**

두 번째 NIC에서 **ptp4I** 을 경계 시계로 시작하는 데 필요한 인터페이스를 지정합니다.



참고

두 번째 NIC에서 **phc2sysOpts** 필드를 두 번째 **PtpConfig CR**에서 완전히 제거하여 **phc2sys** 서비스를 비활성화해야 합니다.

2.

다음 명령을 실행하여 듀얼 NIC **PtpConfig CR**을 생성합니다.

a.

첫 번째 NIC에 대해 **PTP**를 구성하는 **CR**을 생성합니다.

```
$ oc create -f boundary-clock-ntp-config-nic1.yaml
```

b.

두 번째 NIC에 대해 **PTP**를 구성하는 **CR**을 생성합니다.

```
$ oc create -f boundary-clock-ntp-config-nic2.yaml
```

검증

•

**PTP Operator**가 두 NIC에 **PtpConfig CR**을 적용했는지 확인합니다. 이중 NIC 하드웨어가 설치된 노드에 해당하는 **linuxptp** 데몬의 로그를 검사합니다. 예를 들어 다음 명령을 실행합니다.

```
$ oc logs linuxptp-daemon-cvgr6 -n openshift-ptp -c linuxptp-daemon-container
```

출력 예

```

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

```

### 15.5.5. Intel coumbiaville E800 시리즈 NIC as PTP 일반 클럭 참조

다음 표에서는 Intel coumbiaville E800 시리즈 NIC를 일반 시계로 사용하기 위해 참조 PTP 설정을 변경해야 하는 변경 사항을 설명합니다. 클러스터에 적용하는 PtpConfig CR(사용자 정의 리소스)을 변경합니다.

표 15.1. Intel coumbiaville NIC에 권장되는 PTP 설정

PTP 구성	권장 설정
phc2sysOpts	-a -r -m -n 24 -N 8 -R 16
tx_timestamp_timeout	50
boundary_clock_jbod	0



#### 참고

phc 2sysOpts의 경우 -m 은 stdout 에 메시지를 출력합니다. linuxptp-daemon DaemonSet 은 로그를 구문 분석하고 Prometheus 지표를 생성합니다.

#### 추가 리소스

- PTP 빠른 이벤트가 있는 일반 시계로 linuxptp 서비스를 구성하는 전체 예제 CR은 일반 시계 로 [linuxptp 서비스](#) 구성을 참조하십시오.

### 15.5.6. PTP 하드웨어에 대한 VRF 우선 순위 스케줄링 구성

대기 시간이 짧은 성능을 요구하는 통신 또는 기타 배포 구성에서 PTP 데몬 스레드는 나머지 인프라 구성 요소와 함께 제한된 CPU 풋프린트에서 실행됩니다. 기본적으로 PTP 스레드는 SCHED\_OTHER 정책으로 실행됩니다. 높은 로드에서 이러한 스레드는 오류가 없는 작업에 필요한 스케줄링 대기 시간을 얻지 못할 수 있습니다.

잠재적인 스케줄링 대기 시간 오류가 발생하지 않도록 **PTP Operator linuxptp** 서비스를 구성하여 스레드를 **schedule\_VRF** 정책으로 실행할 수 있습니다. **ppc\_VRF**가 **PtpConfig CR**에 대해 설정된 경우 **ptp4l** 및 **phc2sys** 는 **PtpConfig CR**의 **ptpSchedulingPriority** 필드에 의해 설정된 우선 순위로 **chrt** 의 상위 컨테이너에서 실행됩니다.



#### 참고

**ptpSchedulingPolicy** 설정은 선택 사항이며 대기 시간 오류가 발생하는 경우에만 필요합니다.

#### 절차

1. **PtpConfig CR** 프로필을 편집합니다.

```
$ oc edit PtpConfig -n openshift-ptp
```

2. **ptpSchedulingPolicy** 및 **ptpSchedulingPriority** 필드를 변경합니다.

```
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

**ptp4l** 및 **phc2sys** 프로세스에 대한 스케줄링 정책입니다. **Havana** 스케줄링을 지원하는 시스템에서 **Retain\_VRF**를 사용합니다.

2

필수 항목입니다. **ptp4l** 및 **phc2sys** 프로세스의 **FIFO** 우선 순위를 구성하는 데 사용되는 정수 값 **1-65**를 설정합니다.

3. 저장 후 종료하여 **PtpConfig CR**에 변경 사항을 적용합니다.

## 검증

1.

**linuxptp-daemon Pod의 이름과 PtpConfig CR이 적용된 해당 노드를 가져옵니다.**

```
$ oc get pods -n openshift-ntp -o wide
```

출력 예

```

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

```

2.

**업데이트된 chrt first priority로 ptp4l 프로세스가 실행 중인지 확인합니다.**

```
$ oc -n openshift-ntp logs linuxptp-daemon-lgm55 -c linuxptp-daemon-container|grep chrt
```

출력 예

```
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
```

## 15.6. 일반적인 PTP OPERATOR 문제 해결

다음 단계를 수행하여 PTP Operator의 일반적인 문제를 해결합니다.

### 사전 요구 사항

- **OpenShift Container Platform CLI (oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- **PTP**를 지원하는 호스트가 있는 베어 메탈 클러스터에 **PTP Operator**를 설치합니다.

절차

1. 구성된 노드를 위해 **Operator** 및 **Operand**가 클러스터에 성공적으로 배포되었는지 확인합니다.

```
$ oc get pods -n openshift-ptp -o wide
```

출력 예

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-6b8dcbf7f4-zndk7	1/1	Running	0	5d7h	10.129.0.61	control-plane-1.example.com



참고

**PTP** 빠른 이벤트 버스가 활성화되면 준비된 **linuxptp-daemon Pod** 수는 **3/3**가 됩니다. **PTP** 빠른 이벤트 버스가 활성화되지 않으면 **2/2**가 표시됩니다.

2. 지원되는 하드웨어가 클러스터에 있는지 확인합니다.

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io
```

출력 예



<b>NAME</b>	<b>AGE</b>
<b>control-plane-0.example.com</b>	<b>10d</b>
<b>control-plane-1.example.com</b>	<b>10d</b>
<b>compute-0.example.com</b>	<b>10d</b>
<b>compute-1.example.com</b>	<b>10d</b>
<b>compute-2.example.com</b>	<b>10d</b>

3.

노드에 사용 가능한 **PTP** 네트워크 인터페이스를 확인합니다.

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io <node_name> -o yaml
```

다음과 같습니다.

**<node\_name>**

쿼리할 노드를 지정합니다 (예: **compute-0.example.com**).

출력 예

```
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.

해당 노드의 `linuxptp-daemon Pod`에 액세스하여 `PTP` 인터페이스가 기본 클록에 성공적으로 동기화되었는지 확인합니다.

a.

다음 명령을 실행하여 `linuxptp-daemon Pod`의 이름과 문제를 해결하려는 해당 노드를 가져옵니다.

```
$ oc get pods -n openshift-ptp -o wide
```

출력 예

```
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-6b8dcbf7f4-zndk7      1/1   Running 0       5d7h  10.129.0.61 control-plane-1.example.com
```

b.

필수 `linuxptp-daemon` 컨테이너로의 원격 셸:

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container <linux_daemon_container>
```

다음과 같습니다.

```
<linux_daemon_container>
```

진단할 컨테이너입니다 (예: `linuxptp-daemon-lmvgn`).

c.

`linuxptp-daemon` 컨테이너에 대한 원격 셸 연결에서 `PTP` 관리 클라이언트(`pmc`) 툴을 사용하여 네트워크 인터페이스를 진단합니다. 다음 `pmc` 명령을 실행하여 `PTP` 장치의 동기화 상태를 확인합니다(예: `ptp4l`).

```
# pmc -u -f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

노드가 기본 클록에 성공적으로 동기화되었을 때의 출력 예

```

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

```

## 15.7. PTP 하드웨어 빠른 이벤트 알림 프레임워크

### 15.7.1. PTP 및 클럭 동기화 오류 이벤트 정보

가상 RAN과 같은 클라우드 네이티브 애플리케이션에서는 전체 네트워크의 작동에 중요한 하드웨어 타이밍 이벤트에 대한 알림에 액세스해야 합니다. 빠른 이벤트 알림은 임박한 실시간 PTP(Precision Time Protocol) 클럭 동기화 이벤트에 대한 조기 경고 신호입니다. PTP 클럭 동기화 오류는 낮은 대기 시간 애플리케이션의 성능과 안정성에 부정적인 영향을 줄 수 있습니다(예: 분산 장치(DU)에서 실행되는 vRAN 애플리케이션).

PTP 동기화 손실은 RAN 네트워크에 심각한 오류입니다. 노드에서 동기화가 손실된 경우 라디오가 종료될 수 있으며 네트워크 Over the Air (OTA) 트래픽이 무선 네트워크의 다른 노드로 이동될 수 있습니다. 클러스터 노드에서 PTP 클럭 동기화 상태를 DU에서 실행 중인 vRAN 애플리케이션에 통신할 수 있도록 함으로써 이벤트 알림이 워크로드 오류와 비교하여 완화됩니다.

동일한 DU 노드에서 실행되는 RAN 애플리케이션에서 이벤트 알림을 사용할 수 있습니다. 게시/서브스크립션 REST API는 이벤트 알림을 메시징 버스에 전달합니다. 게시/서브스크립션 메시징 또는 pub/sub 메시징은 주제에 게시된 모든 메시지가 해당 주제에 대한 모든 가입자에 의해 즉시 수신되는 서비스 통신 아키텍처에 대한 비동기식 서비스입니다.

빠른 이벤트 알림은 OpenShift Container Platform의 PTP Operator에서 모든 PTP 가능 네트워크 인터페이스에 대해 생성됩니다. 이 이벤트는 AMQP(Advanced Message Queuing Protocol) 메시지 버스를 통해 cloud-event-proxy 사이드카 컨테이너를 사용하여 사용할 수 있습니다. AMQP 메시지 버스는 AMQ Interconnect Operator에서 제공합니다.



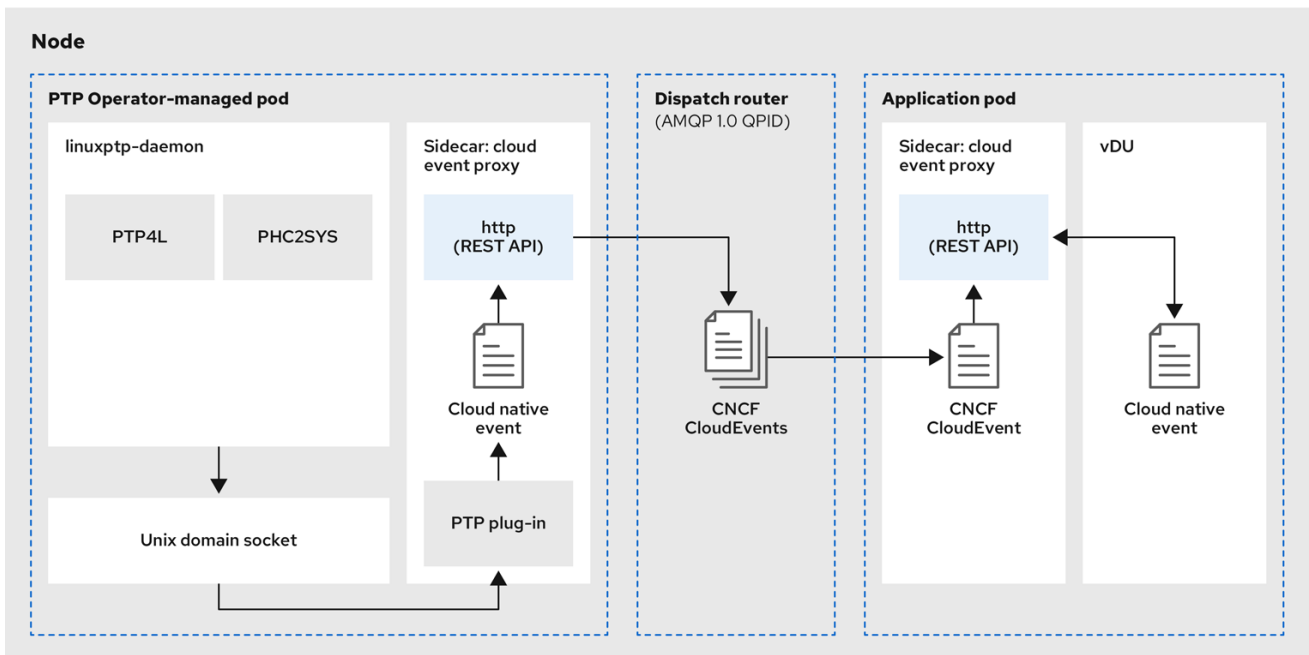
참고

**PTP 빠른 이벤트 알람은 PTP 일반 클럭 또는 PTP 경계 클럭을 사용하도록 구성된 네트워크 인터페이스에 사용할 수 있습니다.**

15.7.2. PTP 빠른 이벤트 알람 프레임워크 정보

DCN(Distributed Unit) 애플리케이션을 PTP(Precision Time Protocol) 빠른 이벤트 알람을 PTP Operator 및 cloud-event-proxy 사이드카 컨테이너를 사용하여 생성한 빠른 이벤트 알람에 등록할 수 있습니다. ptpOperatorConfig CR(사용자 정의 리소스)에서 enableEventPublisher 필드를 true 로 설정하고 AMQPP(Advanced Messageauthorization Protocol) transportHost 주소를 지정하여 cloud-event-proxy 사이드카 컨테이너를 활성화합니다. PTP 빠른 이벤트는 AMQ Interconnect Operator가 제공하는 AMQP 이벤트 알람 버스를 사용합니다. AMQ Interconnect는 AMQP 지원 엔드포인트 간에 유연한 메시지 라우팅을 제공하는 메시징 라우터인 Red Hat AMQ의 구성 요소입니다. PTP 빠른 이벤트 프레임워크의 개요는 다음과 같습니다.

그림 15.1. PTP 빠른 이벤트 개요



218\_OpenShift\_0122

**cloud-event-proxy** 사이드카 컨테이너는 기본 애플리케이션의 리소스를 사용하지 않고 대기 시간 없이 기본 vRAN 애플리케이션과 동일한 리소스에 액세스할 수 있습니다.

빠른 이벤트 알람 프레임워크는 통신에 REST API를 사용하며 O-RAN REST API 사양을 기반으로 합니다. 프레임워크는 게시자 및 구독자 애플리케이션 간의 통신을 처리하는 게시자, 구독자 및 AMQ 메시징 버스로 구성됩니다. cloud-event-proxy 사이드카는 DU 노드의 기본 DU 애플리케이션 컨테이너에 느슨하게 연결된 Pod에서 실행되는 유틸리티 컨테이너입니다. DU 애플리케이션을 게시된 PTP 이벤트에 등록할 수 있는 이벤트 게시 프레임워크를 제공합니다.

**DU 애플리케이션은 사이드카 패턴에서 cloud-event-proxy 컨테이너를 실행하여 PTP 이벤트를 구독합니다. 다음 워크플로는 DU 애플리케이션에서 PTP 빠른 이벤트를 사용하는 방법을 설명합니다.**

1.
 

**DU 애플리케이션에서 서브스크립션 요청: DU는 API 요청을 cloud-event-proxy 사이드카로 전송하여 PTP 이벤트 서브스크립션을 생성합니다. cloud-event-proxy 사이드카는 서브스크립션 리소스를 생성합니다.**
2.
 

**cloud-event-proxy 사이드카는 서브스크립션을 생성: 이벤트 리소스는 cloud-event-proxy 사이드카에 의해 유지됩니다. cloud-event-proxy 사이드카 컨테이너는 ID 및 URL 위치가 있는 승인을 전송하여 저장된 서브스크립션 리소스에 액세스합니다. 사이드카는 서브스크립션에 지정된 리소스에 대한 AMQ 메시징 리스너 프로토콜을 생성합니다.**
3.
 

**DU 애플리케이션에서 PTP 이벤트 알림 수신: cloud-event-proxy 사이드카 컨테이너가 리소스 한정자에 지정된 주소를 수신합니다. DU 이벤트 소비자는 메시지를 처리하고 서브스크립션에 지정된 반환 URL로 전달합니다.**
4.
 

**cloud-event-proxy 사이드카는 PTP 이벤트를 검증하고 DU 애플리케이션에 게시: cloud-event-proxy 사이드카는 이벤트를 수신하고 클라우드 이벤트 오브젝트를 래핑하여 데이터를 검색하고 반환 URL을 가져와 이벤트를 DU 소비자 애플리케이션에 다시 게시합니다.**
5.
 

**DU 애플리케이션은 PTP 이벤트 사용: DU 애플리케이션 이벤트 소비자가 PTP 이벤트를 수신하고 처리합니다.**

### 15.7.3. AMQ 메시징 버스 설치

노드에서 게시자와 구독자 간에 PTP 빠른 이벤트 알림을 전달하려면 노드에서 로컬로 실행되도록 AMQ 메시징 버스를 설치하고 구성해야 합니다. 클러스터에서 사용할 AMQ Interconnect Operator를 설치하여 이 작업을 수행합니다.

#### 사전 요구 사항

- **OpenShift Container Platform CLI (oc)를 설치합니다.**
- **cluster-admin 권한이 있는 사용자로 로그인합니다.**

#### 절차

-

**AMQ Interconnect Operator**를 자체 `amq-interconnect` 네임스페이스에 설치합니다. [Add the Red Hat Integration - AMQ Interconnect Operator](#) 를 참조하십시오.

## 검증

1. **AMQ Interconnect Operator**를 사용할 수 있고 필요한 **Pod**가 실행 중인지 확인합니다.

```
$ oc get pods -n amq-interconnect
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
amq-interconnect-645db76c76-k8ghs	1/1	Running	0	23h
interconnect-operator-5cb5fc7cc-4v7qm	1/1	Running	0	23h

- 2.

필수 `linuxptp-daemon` **PTP** 이벤트 생산자 **Pod**가 `openshift-ptp` 네임스페이스에서 실행되고 있는지 확인합니다.

```
$ oc get pods -n openshift-ptp
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
linuxptp-daemon-2t78p	3/3	Running	0	12h
linuxptp-daemon-k8n88	3/3	Running	0	12h

### 15.7.4. PTP 빠른 이벤트 알림 게시자 구성

클러스터에서 네트워크 인터페이스에 **PTP** 빠른 이벤트 알림을 사용하려면 **PTP Operator** `PtpOperatorConfig` **CR**(사용자 정의 리소스)에서 빠른 이벤트 게시자를 활성화하고 생성한 `PtpConfig` **CR**에서 `ptpClockThreshold` 값을 구성해야 합니다.

사전 요구 사항

- **OpenShift Container Platform CLI (oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- **PTP Operator** 및 **AMQ Interconnect Operator**를 설치합니다.

## 절차

1. **PTP 빠른 이벤트를 활성화하도록 기본 PTP Operator 구성을 수정합니다.**

- a. 다음 YAML을 `ptp-operatorconfig.yaml` 파일에 저장합니다.

```

apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    enableEventPublisher: true ①
    transportHost: amqp://<instance_name>.<namespace>.svc.cluster.local ②

```

①

`enableEventPublisher`를 `true`로 설정하여 **PTP 빠른 이벤트 알림을 활성화**합니다.

②

`< instance_name >` 및 `< namespace >`가 **AMQ Interconnect 라우터 인스턴스 이름과 네임스페이스(예: `amqp://amq-interconnect.amq-interconnect.svc.cluster.local`)에 해당하는 AMQ 라우터로 `transportHost`를 설정**합니다.

- b. **PtpOperatorConfig CR**을 업데이트합니다.

```
$ oc apply -f ptp-operatorconfig.yaml
```

2.

PTP 지원 인터페이스에 대한 **PtpConfig CR**(사용자 정의 리소스)을 생성하고 **ptpClockThreshold** 및 **ptp4IOpts** 에 필요한 값을 설정합니다. 다음 **YAML**은 **PtpConfig CR**에 설정해야 하는 필수 값을 보여줍니다.

```
spec:
  profile:
    - name: "profile1"
      interface: "enp5s0f0"
      ptp4IOpts: "-2 -s --summary_interval -4" 1
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
      ptp4IConf: "" 3
      ptpClockThreshold: 4
      holdOverTimeout: 5
      maxOffsetThreshold: 100
      minOffsetThreshold: -100
```

1

PTP 빠른 이벤트를 사용하려면 `--summary_interval -4` 를 추가합니다.

2

필수 `phc2sysOpts` 값. `-m` 은 `stdout` 에 메시지를 출력합니다. `linuxptp-daemon DaemonSet` 은 로그를 구문 분석하고 `Prometheus` 지표를 생성합니다.

3

기본 `/etc/ptp4l.conf` 파일을 대체할 구성이 포함된 문자열을 지정합니다. 기본 구성을 사용하려면 필드를 비워 둡니다.

4

선택 사항: `ptpClockThreshold` 가 없으면 기본값이 `ptpClockThreshold` 필드에 사용 됩니다. 스탠자는 기본 `ptpClockThreshold` 값을 표시합니다. `ptpClockThreshold` 값은 PTP 이벤트가 트리거되기 전에 PTP 마스터 클럭이 연결 해제된 후의 기간을 구성합니다. `holdOverTimeout` 은 PTP 마스터 클럭의 연결이 끊어지면 PTP 클럭 이벤트 상태가 `FREERUN` 로 변경되기 전 시간(초)입니다. `maxOffsetThreshold` 및 `minOffsetThreshold` 설정은 `CLOCK_REALTIME (phc2sys)` 또는 마스터 오프셋(`ptp4l`)의 값과 비교되는 나노초에 오프셋 값을 구성합니다. `ptp4l` 또는 `phc2sys` 오프셋 값이 이 범위를 벗어나는 경우 PTP 클럭 상태가 `FREERUN` 로 설정됩니다. 오프셋 값이 이 범위 내에 있으면 PTP 클럭 상태가 `LOCKED` 로 설정됩니다.

#### 추가 리소스

•

PTP 빠른 이벤트가 있는 일반 시계로 `linuxptp` 서비스를 구성하는 전체 예제 CR은 일반 시계 로 `linuxptp 서비스` 구성을 참조하십시오.



### 15.7.5. DU 애플리케이션을 PTP 이벤트 REST API 참조에 구독

PTP 이벤트 알림을 REST API를 사용하여 DU(Distributed Unit) 애플리케이션을 상위 노드에 생성된 PTP 이벤트에 서브스크립션합니다.

리소스 주소 `/cluster/node/<node_name>/ptp` 을 사용하여 애플리케이션을 PTP 이벤트에 서브스크립션합니다. 여기서 `<node_name >`은 DU 애플리케이션을 실행하는 클러스터 노드입니다.

별도의 DU 애플리케이션 Pod에 `cloud-event-consumer` DU 애플리케이션 컨테이너 및 `cloud-event-proxy` 사이드카 컨테이너를 배포합니다. `cloud-event-consumer` DU 애플리케이션은 애플리케이션 Pod의 `cloud-event-proxy` 컨테이너에 가입합니다.

다음 API 끝점을 사용하여 DU 애플리케이션 Pod의 `http://localhost:8089/api/ocloudNotifications/v1/` 에서 `cloud-event-consumer` DU 애플리케이션을 게시한 PTP 이벤트에 등록합니다.

- `/api/ocloudNotifications/v1/subscriptions`
  - **POST:** 새 서브스크립션을 생성합니다.
  - **GET:** 서브스크립션 목록 검색합니다.
- `/api/ocloudNotifications/v1/subscriptions/<subscription_id>`
  - **GET:** 지정된 서브스크립션 ID에 대한 세부 정보를 반환합니다.
- `api/ocloudNotifications/v1/subscriptions/status/<subscription_id>`
  - **PUT:** 지정된 서브스크립션 ID에 대한 새 상태 ping 요청 생성
- `/api/ocloudNotifications/v1/health`

- **GET: ocloudNotifications API의 상태를 반환합니다.**
- **api/ocloudNotifications/v1/publishers**
- **GET: 클러스터 노드에 대한 os-clock-sync-state,ptp-clock-class-change, lock-state 메시지 배열을 반환합니다.**
- **/api/ocloudnotifications/v1/<resource\_address>/CurrentState**
- **GET: os-clock-sync-state,ptp-clock-class-change, lock-state 이벤트 등 하나의 현재 상태를 반환합니다.**



참고

9089 는 애플리케이션 Pod에 배포된 cloud-event-consumer 컨테이너의 기본 포트입니다. 필요에 따라 DU 애플리케이션에 대해 다른 포트를 구성할 수 있습니다.

15.7.5.1. api/ocloudNotifications/v1/subscriptions

HTTP 방법

GET api/ocloudNotifications/v1/subscriptions

설명

서브스크립션 목록을 반환합니다. 서브스크립션이 존재하는 경우 200 OK 상태 코드가 서브스크립션 목록과 함께 반환됩니다.

API 응답 예

```
[
  {
    "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 방법****POST `api/ocloudNotifications/v1/subscriptions`****설명**

새 서브스크립션을 생성합니다. 서브스크립션이 성공적으로 생성되었거나 이미 존재하는 경우 **201 Created** 상태 코드가 반환됩니다.

표 15.2. 쿼리 매개변수

매개변수	유형
subscription	data

**페이로드 예**

```
{
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

**15.7.5.2. `api/ocloudNotifications/v1/subscriptions/<subscription_id>`****HTTP 방법****GET `api/ocloudNotifications/v1/subscriptions/<subscription_id>`****설명**

ID `<subscription_id>`가 있는 서브스크립션의 세부 정보를 반환합니다.

표 15.3. 쿼리 매개변수

매개변수	유형
<code>&lt;subscription_id&gt;</code>	string

**API 응답 예**

```
{
  "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"
}
```

**15.7.5.3. api/ocloudNotifications/v1/subscriptions/status/<subscription\_id>****HTTP 방법****PUT api/ocloudNotifications/v1/subscriptions/status/<subscription\_id>****설명**

ID <subscription\_id>를 사용하여 서브스크립션에 대한 새 상태 ping 요청을 생성합니다. 서브스크립션이 있는 경우 상태 요청이 성공하고 **202 Accepted** 상태 코드가 반환됩니다.

표 15.4. 쿼리 매개변수

매개변수	유형
<subscription_id>	string

**API 응답 예**

```
{"status": "ping sent"}
```

**15.7.5.4. api/ocloudNotifications/v1/health/****HTTP 방법****GET api/ocloudNotifications/v1/health/**

**설명**

**ocloudNotifications REST API의 상태를 반환합니다.**

**API 응답 예**

**OK**

**15.7.5.5. api/ocloudNotifications/v1/publishers****HTTP 방법**

**GET api/ocloudNotifications/v1/publishers**

**설명**

클러스터 노드의 **os-clock-sync-state**, **ptp-clock-class-change** 및 **lock-state** 세부 정보를 반환합니다. 시스템은 관련 장비 상태 변경이 있을 때 알람을 생성합니다.

- **OS-clock-sync-state** 알람은 호스트 운영 체제 클럭 동기화 상태를 설명합니다. **LOCKED** 또는 **free RUN** 상태에 있을 수 있습니다.
- **PTP-clock-class-change** 알람은 **PTP** 클럭 클래스의 현재 상태를 설명합니다.
- **lock-state** 알람은 **PTP** 장비 잠금 상태의 현재 상태를 설명합니다. **LOCKED**, **HOLDOVER** 또는 **free RUN** 상태에 있을 수 있습니다.

**API 응답 예**

```
[
  {
    "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/ptp-clock-class-change"
},
{
  "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"
}
]

```

`cloud-event-proxy` 컨테이너의 로그에서 `os-clock-sync-state`, `ptp-clock-class-change` 및 `lock-state` 이벤트를 찾을 수 있습니다. 예를 들어 다음과 같습니다.

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

`os-clock-sync-state` 이벤트의 예

```

{
  "id": "c8a784d1-5f4a-4c16-9a81-a3b4313affe5",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "source": "/cluster/compute-1.example.com/ptp/CLOCK_REALTIME",
  "dataContentType": "application/json",
  "time": "2022-05-06T15:31:23.906277159Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/sync/sync-status/os-clock-sync-state",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/sync/sync-status/os-clock-sync-state",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "-53"
      }
    ]
  }
}

```

```

    ]
  }
}

```

### *ptp-clock-class-change* 이벤트의 예

```

{
  "id": "69eddb52-1650-4e56-b325-86d44688d02b",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType": "application/json",
  "time": "2022-05-06T15:31:23.147100033Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/sync/ptp-status/ptp-clock-class-change",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "135"
      }
    ]
  }
}

```

### *lock-state* 이벤트의 예

```

{
  "id": "305ec18b-1472-47b3-aadd-8f37933249a9",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType": "application/json",
  "time": "2022-05-06T15:31:23.467684081Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/sync/ptp-status/lock-state",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      }
    ],
  }
}

```

```

    {
      "resource":"/sync/ptp-status/lock-state",
      "dataType":"metric",
      "valueType":"decimal64.3",
      "value":"62"
    }
  ]
}
}

```

#### 15.7.5.6. /api/ocloudnotifications/v1/<resource\_address>/CurrentState

##### HTTP 방법

**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/ptp-clock-class-change/CurrentState`

##### 설명

클러스터 노드의 `os-clock-sync-state`, `ptp-clock-class-change`, `lock-state` 이벤트의 현재 상태를 반환하도록 **CurrentState** API 엔드포인트를 구성합니다.

- **OS-clock-sync-state** 알림은 호스트 운영 체제 클럭 동기화 상태를 설명합니다. **LOCKED** 또는 **free RUN** 상태에 있을 수 있습니다.
- **PTP-clock-class-change** 알림은 **PTP** 클럭 클래스의 현재 상태를 설명합니다.
- **lock-state** 알림은 **PTP** 장비 잠금 상태의 현재 상태를 설명합니다. **LOCKED**, **HOLDOVER** 또는 **free RUN** 상태에 있을 수 있습니다.

표 15.5. 쿼리 매개변수



매개 변수	유형
<resource_address>	string

### lock-state API 응답 예

```
{
  "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": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

### os-clock-sync-state API 응답 예

```
{
  "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": "v1",
    "values": [
```

```

{
  "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
  "dataType": "notification",
  "valueType": "enumeration",
  "value": "LOCKED"
},
{
  "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
  "dataType": "metric",
  "valueType": "decimal64.3",
  "value": "27"
}
]
}
}

```

### *ptp-clock-class-change API 응답 예*

```

{
  "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/ptp-clock-class-change",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
}

```

### 15.7.6. CLI를 사용하여 PTP 빠른 이벤트 메트릭 모니터링

oc CLI를 사용하여 cloud-event-proxy 컨테이너에서 직접 빠른 이벤트 버스 메트릭을 모니터링할 수 있습니다.



## 참고

**OpenShift Container Platform** 웹 콘솔에서 **PTP** 빠른 이벤트 알림 메트릭도 사용할 수 있습니다.

## 사전 요구 사항

- **OpenShift Container Platform CLI (oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- **PTP Operator**를 설치하고 구성합니다.

## 절차

1. 활성 **linuxptp-daemon Pod** 목록을 가져옵니다.

```
$ oc get pods -n openshift-ptp
```

출력 예

```
NAME                READY STATUS RESTARTS AGE
linuxptp-daemon-2t78p 3/3   Running 0      8h
linuxptp-daemon-k8n88 3/3   Running 0      8h
```

2. 다음 명령을 실행하여 필요한 **cloud-event-proxy** 컨테이너의 메트릭에 액세스합니다.

```
$ oc exec -it <linuxptp-daemon> -n openshift-ptp -c cloud-event-proxy -- curl 127.0.0.1:9091/metrics
```

다음과 같습니다.

```
<linuxptp-daemon>
```

쿼리할 Pod를 지정합니다(예: `linuxptp-daemon-2t78p`).

출력 예

```
# HELP cne_amqp_events_published Metric to get number of events published by
the transport
# TYPE cne_amqp_events_published gauge
cne_amqp_events_published{address="/cluster/node/compute-
1.example.com/ptp/status",status="success"} 1041
# HELP cne_amqp_events_received Metric to get number of events received by the
transport
# TYPE cne_amqp_events_received gauge
cne_amqp_events_received{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 1019
# HELP cne_amqp_receiver Metric to get number of receiver created
# TYPE cne_amqp_receiver gauge
cne_amqp_receiver{address="/cluster/node/mock",status="active"} 1
cne_amqp_receiver{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 1
cne_amqp_receiver{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"}
...
```

### 15.7.7. 웹 콘솔에서 PTP 빠른 이벤트 메트릭 모니터링

사전 구성 및 자체 업데이트 Prometheus 모니터링 스택을 사용하여 OpenShift Container Platform 웹 콘솔에서 PTP 빠른 이벤트 메트릭을 모니터링할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform CLI `oc`를 설치합니다.
- `cluster-admin` 권한이 있는 사용자로 로그인합니다.

절차

1. 다음 명령을 입력하여 `cloud-event-proxy` 사이드카 컨테이너에서 사용 가능한 PTP 메트릭 목록을 반환합니다.

```
$ oc exec -it <linuxptp_daemon_pod> -n openshift-ntp -c cloud-event-proxy -- curl 127.0.0.1:9091/metrics
```

다음과 같습니다.

<linuxptp\_daemon\_pod>

쿼리할 Pod를 지정합니다(예: linuxptp-daemon-2t78p ).

2. 반환된 메트릭 목록에서 쿼리할 PTP 메트릭의 이름을 복사합니다(예: `cne_amqp_events_received` ).
3. **OpenShift Container Platform** 웹 콘솔에서 모니터링 → 메트릭을 클릭합니다.
4. **PTP 메트릭을 표현식 필드에 붙여넣고 쿼리 실행을 클릭합니다.**

추가 리소스

- [메트릭 관리](#)

## 16장. 외부 DNS OPERATOR

### 16.1. OPENSIFT CONTAINER PLATFORM의 외부 DNS OPERATOR

외부 DNS Operator는 ExternalDNS 를 배포 및 관리하여 외부 DNS 공급자에서 OpenShift Container Platform으로의 서비스 및 경로에 대한 이름 확인을 제공합니다.

#### 16.1.1. 외부 DNS Operator

외부 DNS Operator는 olm.openshift.io API 그룹에서 외부 DNS API를 구현합니다. 외부 DNS Operator는 배포 리소스를 사용하여 ExternalDNS 를 배포합니다. ExternalDNS 배포는 클러스터의 서비스 및 경로와 같은 리소스를 감시하고 외부 DNS 공급자를 업데이트합니다.

#### 절차

OperatorHub에서 필요에 따라 ExternalDNS Operator를 배포할 수 있으므로 Subscription 오브젝트가 생성됩니다.

1. 설치 계획의 이름을 확인합니다.

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

출력 예

```
install-zcvlr
```

2. 설치 계획의 상태를 확인하면 설치 계획의 상태가 완료 여야 합니다.

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

출력 예

```
Complete
```

3.

**oc get** 명령을 사용하여 배포 상태를 확인합니다.

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

출력 예

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

### 16.1.2. 외부 DNS Operator 로그

**oc logs** 명령을 사용하여 외부 DNS Operator 로그를 볼 수 있습니다.

절차

1.

외부 DNS Operator의 로그를 확인합니다.

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

#### 16.1.2.1. 외부 DNS Operator 도메인 이름 제한

외부 DNS Operator는 새 형식을 따르고 TXT 레코드의 접두사를 추가하는 TXT 레지스트리를 사용합니다. 이렇게 하면 TXT 레코드의 도메인 이름의 최대 길이가 줄어듭니다. 해당 TXT 레코드 없이는 DNS 레코드를 존재할 수 없으므로 DNS 레코드의 도메인 이름은 TXT 레코드와 동일한 제한을 따라야 합니다. 예를 들어 DNS 레코드는 < domain-name-from-source >이며 TXT 레코드는 external-dns-< records-type>-<domain-name-from-source >입니다.

외부 DNS Operator에서 생성한 DNS 레코드의 도메인 이름에는 다음과 같은 제한 사항이 있습니다.

레코드 유형	문자 수
CNAME	44
AzureDNS의 와일드카드 CNAME 레코드	42
A	48
AzureDNS의 와일드카드 A 레코드	46

외부 DNS에서 생성한 도메인 이름이 도메인 이름 제한을 초과하면 외부 DNS 인스턴스에서 다음 오류가 발생합니다.

```
$ oc -n external-dns-operator logs external-dns-aws-7ddb9c7f8-2jqjh 1
```

1

`external-dns-aws-7ddb9c7f8-2jqjh` 매개 변수는 외부 DNS Pod의 이름을 지정합니다.

출력 예

```
time="2022-09-02T08:53:57Z" level=info msg="Desired change: CREATE external-dns-cname-hello-openshift-aaaaaaaaa-bbbbbbbbbb-cccccc.test.example.io TXT [Id: /hostedzone/Z06988883Q0H0RL6UMXXX]"
time="2022-09-02T08:53:57Z" level=info msg="Desired change: CREATE external-dns-hello-openshift-aaaaaaaaa-bbbbbbbbbb-cccccc.test.example.io TXT [Id: /hostedzone/Z06988883Q0H0RL6UMXXX]"
time="2022-09-02T08:53:57Z" level=info msg="Desired change: CREATE hello-openshift-aaaaaaaaa-bbbbbbbbbb-cccccc.test.example.io A [Id: /hostedzone/Z06988883Q0H0RL6UMXXX]"
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-openshift-aaaaaaaaa-bbbbbbbbbb-cccccc']\n\tstatus code: 400, request id: e54dfd5a-06c6-47b0-bcb9-a4f7c3a4e0c6"
```

## 16.2. 클라우드 공급자에 외부 DNS OPERATOR 설치



**AWS, Azure 및 GCP와 같은 클라우드 공급자에 외부 DNS Operator를 설치할 수 있습니다.**

### 16.2.1. 외부 DNS Operator 설치

**OpenShift Container Platform OperatorHub를 사용하여 외부 DNS Operator를 설치할 수 있습니다.**

#### 절차

1. **OpenShift Container Platform 웹 콘솔에서 Operators → OperatorHub 를 클릭합니다.**
2. **외부 DNS Operator 를 클릭합니다. 키워드로 텍스트 상자 또는 필터 목록을 사용하여 Operator 목록에서 외부 DNS Operator를 검색할 수 있습니다.**
3. **external-dns-operator 네임스페이스를 선택합니다.**
4. **External DNS Operator 페이지에서 설치를 클릭합니다.**
5. **Operator 설치 페이지에서 다음 옵션을 선택했는지 확인합니다.**
  - a. **채널을 stable-v1.0 으로 업데이트합니다.**
  - b. **클러스터의 특정 이름으로 설치 모드입니다.**
  - c. **external-dns-operator 로 설치된 네임스페이스입니다. external-dns-operator 네임스페이스가 없는 경우 Operator 설치 중에 생성됩니다.**
  - d. **승인 전략을 자동 또는 수동으로 선택합니다. 승인 전략은 기본적으로 자동으로 설정됩니다.**
  - e. **설치를 클릭합니다.**

자동 업데이트를 선택하면 **OLM(Operator Lifecycle Manager)**에서 개입 없이 실행 중인 **Operator** 인스턴스를 자동으로 업그레이드합니다.

수동 업데이트를 선택하면 **OLM**에서 업데이트 요청을 생성합니다. 클러스터 관리자는 **Operator**를 새 버전으로 업데이트하려면 **OLM** 업데이트 요청을 수동으로 승인해야 합니다.

검증

외부 **DNS Operator**가 설치된 **Operator** 대시보드에서 상태가 **Succeeded** 로 표시되는지 확인합니다.

16.3. 외부 DNS OPERATOR 구성 매개변수

외부 **DNS Operator**에는 다음 구성 매개변수가 포함됩니다.

16.3.1. 외부 DNS Operator 구성 매개변수

외부 **DNS Operator**에는 다음 구성 매개변수가 포함됩니다.

매개 변수	설명
spec	<p>클라우드 공급자의 유형을 활성화합니다.</p> <pre>spec:   provider:     type: AWS 1   aws:     credentials:       name: aws-access-key 2</pre> <p>1 AWS, GCP, Azure와 같은 사용 가능한 옵션을 정의합니다.</p> <p>2 클라우드 공급자에 대한 인증 정보가 포함된 <b>시크릿</b> 이름을 정의합니다.</p>

매개변수	설명
영역	<p>해당 도메인에서 DNS 영역을 지정할 수 있습니다. 영역을 지정하지 않으면 <b>ExternalDNS</b> 는 클라우드 공급자 계정에 있는 모든 영역을 검색합니다.</p> <pre>zones: - "myzoneid" 1</pre> <p>1 DNS 영역의 ID를 지정합니다.</p>
domain	<p>해당 도메인에서 AWS 영역을 지정할 수 있습니다. 도메인을 지정하지 않으면 <b>ExternalDNS</b> 에서 클라우드 공급자 계정에 있는 모든 영역을 검색합니다.</p> <pre>domains: - filterType: Include 1   matchType: Exact 2   name: "myzonedomain1.com" 3 - filterType: Include   matchType: Pattern 4   pattern: ".*\otherzonedomain\com" 5</pre> <p>1 지정된 도메인을 포함하도록 <b>ExternalDNS</b> 에 지시합니다.</p> <p>2 <b>ExternalDNS</b> 에서 도메인 일치에 정규 표현식과 일치하지 않는 것과 정확히 일치해야 함을 지시합니다.</p> <p>3 <b>ExternalDNS</b> 필터를 사용하는 정확한 도메인 이름을 정의합니다.</p> <p>4 <b>ExternalDNS</b> 에서 <b>regex-domain-filter</b> 플래그를 설정합니다. Regex 필터를 사용하여 가능한 도메인을 제한할 수 있습니다.</p> <p>5 대상 영역의 도메인을 필터링하기 위해 <b>ExternalDNS</b> 에서 사용할 regex 패턴을 정의합니다.</p>
소스	<p>DNS 레코드, 서비스 또는 경로 의 소스를 지정할 수 있습니다.</p> <pre>source: 1 type: Service 2 service:   serviceType: 3   - LoadBalancer   - ClusterIP labelFilter: 4 matchLabels:   external-dns.mydomain.org/publish: "yes" hostnameAnnotation: "Allow" 5 fqdnTemplate: - "{{.Name}}.myzonedomain.com" 6</pre> <p>1 DNS 레코드 소스에 대한 설정을 정의합니다.</p>

매개변수	2 설명
	<p><b>2</b> ExternalDNS 는 dns 레코드를 생성하는 소스로서 서비스 유형을 사용합니다.</p> <p><b>3</b> ExternalDNS 에서 <b>service-type-filter</b> 플래그를 설정합니다. <b>serviceType</b> 에는 다음 필드가 포함되어 있습니다.</p> <ul style="list-style-type: none"> <li>● default: LoadBalancer</li> <li>● 예상:ClusterIP</li> <li>● NodePort</li> <li>● LoadBalancer</li> <li>● ExternalName</li> </ul> <p><b>4</b> 컨트롤러에서 레이블 필터와 일치하는 리소스만 고려하도록 합니다.</p> <p><b>5</b> <b>hostnameAnnotation</b> 의 기본값은 <b>Ignore</b> that externalDNS 가 <b>fqdnTemplates</b> 필드에 지정된 템플릿을 사용하여 DNS 레코드를 생성하도록 지시하는 Ignore입니다. 값이 <b>Allow</b> the DNS records get generated based on the value specified in the <b>external-dns.alpha.kubernetes.io/hostname</b> 주석에 지정된 값을 기반으로 DNS 레코드가 생성될 수 있습니다.</p> <p><b>6</b> 외부 DNS Operator는 문자열을 사용하여 호스트 이름을 정의하지 않는 소스에서 DNS 이름을 생성하거나 페이크 소스와 함께 사용할 때 호스트 이름 접미사를 추가합니다.</p> <pre style="border-left: 2px solid gray; padding-left: 10px;"> 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> externalDNS는 dns 레코드를 생성하는 데 유형 <b>경로</b> 를 소스로 사용합니다.</p> <p><b>2</b> 소스가 <b>OpenShiftRoute</b> 인 경우 Ingress 컨트롤러 이름을 전달할 수 있습니다. <b>ExternalDNS</b> 는 Ingress 컨트롤러의 정식 이름을 CNAME 레코드의 대상으로 사용합니다.</p>

## 16.4. AWS에서 DNS 레코드 생성

외부 DNS Operator를 사용하여 AWS 및 AWS GovCloud에서 DNS 레코드를 생성할 수 있습니다.

### 16.4.1. Red Hat External DNS Operator를 사용하여 AWS의 퍼블릭 호스팅 영역에 DNS 레코드 생성

Red Hat External DNS Operator를 사용하여 AWS의 퍼블릭 호스팅 영역에서 DNS 레코드를 생성할 수 있습니다. 동일한 지침을 사용하여 AWS GovCloud의 호스팅 영역에 DNS 레코드를 생성할 수 있습니다.

## 절차

1.

사용자를 확인합니다. 사용자는 **kube-system** 네임스페이스에 액세스할 수 있어야 합니다. 인증 정보가 없는 경우 **kube-system** 네임스페이스에서 인증 정보를 가져와 클라우드 공급자 클라이언트를 사용할 수 있습니다.

```
$ oc whoami
```

출력 예

```
system:admin
```

2.

**kube-system** 네임스페이스에 있는 **aws-creds** 시크릿에서 값을 가져옵니다.

```
$ 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)
```

3.

경로를 가져와 도메인을 확인합니다.

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

출력 예

```
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
```

4.

**dns** 영역 목록을 가져와서 이전에 찾은 경로의 도메인에 해당하는 항목을 찾습니다.

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

출력 예

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

5.

경로 소스를 위한 **ExternalDNS** 리소스를 생성합니다.

```
$ 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: testtextdnsoperator.apacshift.support 4
  provider:
    type: AWS 5
  source: 6
    type: OpenShiftRoute 7
  openshiftRouteOptions:
    routerName: default 8
EOF
```

1

외부 **DNS** 리소스의 이름을 정의합니다.

2

기본적으로 모든 호스팅 영역이 잠재적인 대상으로 선택됩니다. 필요한 호스팅 영역을 포함할 수 있습니다.

3

대상 영역의 도메인 일치는 정확해야 합니다(정식 표현식과 대조되는 경우).

4

5

**AWS Route53 DNS** 공급자를 정의합니다.

6

**DNS** 레코드 소스에 대한 옵션을 정의합니다.

7

**OpenShift** 경로 리소스를 이전에 지정한 **DNS** 공급자에서 생성되는 **DNS** 레코드의 소스로 정의합니다.

8

소스가 **OpenShiftRoute** 인 경우 **OpenShift Ingress** 컨트롤러 이름을 전달할 수 있습니다. 외부 **DNS Operator**는 **CNAME** 레코드를 생성하는 동안 해당 라우터의 정식 호스트 이름을 대상으로 선택합니다.

6.

다음 명령을 사용하여 **OCP** 경로에 대해 생성된 레코드를 확인합니다.

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

## 16.5. AZURE에서 DNS 레코드 생성

외부 **DNS Operator**를 사용하여 **Azure**에서 **DNS** 레코드를 생성할 수 있습니다.

### 16.5.1. Red Hat External DNS Operator를 사용하여 Azure의 퍼블릭 DNS 영역에 DNS 레코드 생성

**Red Hat External DNS Operator**를 사용하여 **Azure**의 퍼블릭 **DNS** 영역에서 **DNS** 레코드를 생성할 수 있습니다.

#### 절차

1.

사용자를 확인합니다. 사용자는 **kube-system** 네임스페이스에 액세스할 수 있어야 합니다. 인증 정보가 없는 경우 **kube-system** 네임스페이스에서 인증 정보를 가져와 클라우드 공급자 클라이언트를 사용할 수 있습니다.

```
$ oc whoami
```

출력 예

```
system:admin
```

2.

**kube-system** 네임스페이스에 있는 **azure-credentials** 시크릿에서 값을 가져옵니다.

```
$ 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)
```

3.

**base64** 디코딩된 값을 사용하여 **Azure**에 로그인합니다.

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

4.

경로를 가져와 도메인을 확인합니다.

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

출력 예

```
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
```



5. **dns** 영역 목록을 가져와서 이전에 찾은 경로의 도메인에 해당하는 항목을 찾습니다.

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

6. 경로 소스를 위한 **ExternalDNS** 리소스를 생성합니다.

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
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
EOF
```

1

외부 **DNS CR**의 이름을 지정합니다.

2

영역 **ID**를 정의합니다.

3

**Azure DNS** 공급자를 정의합니다.

4

**DNS** 레코드 소스에 대한 옵션을 정의할 수 있습니다.

5

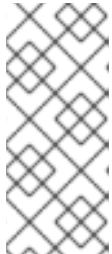
소스가 **OpenShiftRoute** 인 경우 **OpenShift Ingress** 컨트롤러 이름을 전달할 수 있습니다. 외부 **DNS**는 **CNAME** 레코드를 생성하는 동안 해당 라우터의 정식 호스트 이름을 대상으로 선택합니다.

6

7.

다음 명령을 사용하여 OCP 경로에 대해 생성된 레코드를 확인합니다.

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



참고

프라이빗 Azure dns의 프라이빗 호스팅 영역에 레코드를 생성하려면 ExternalDNS 컨테이너 args에서 azure-private-dns 에 공급자 유형을 채우는 프라이빗 영역을 지정해야 합니다.

### 16.6. GCP에서 DNS 레코드 생성

외부 DNS Operator를 사용하여 GCP에서 DNS 레코드를 생성할 수 있습니다.

#### 16.6.1. Red Hat External DNS Operator를 사용하여 GCP의 퍼블릭 관리형 영역에 DNS 레코드 생성

Red Hat External DNS Operator를 사용하여 GCP의 퍼블릭 관리형 영역에서 DNS 레코드를 생성할 수 있습니다.

#### 절차

1.

사용자를 확인합니다. 사용자는 kube-system 네임스페이스에 액세스할 수 있어야 합니다. 인증 정보가 없는 경우 kube-system 네임스페이스에서 인증 정보를 가져와 클라우드 공급자 클라이언트를 사용할 수 있습니다.

```
$ oc whoami
```

출력 예

```
system:admin
```

2.

다음 명령을 실행하여 `encoded-gcloud.json` 파일의 `gcp-credentials` 시크릿의 `service_account.json` 값을 복사합니다.

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

3.

Google 인증 정보를 내보냅니다.

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

4.

다음 명령을 사용하여 계정을 활성화합니다.

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

5.

프로젝트를 설정합니다.

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

6.

경로를 가져와 도메인을 확인합니다.

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

출력 예

```
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
```

7.

관리형 영역 목록을 가져와서 이전에 찾은 경로의 도메인에 해당하는 영역을 찾습니다.

```
$ gcloud dns managed-zones list | grep test.gcp.example.com
qe-cvs4g-private-zone test.gcp.example.com
```

8.

경로 소스를 위한 **ExternalDNS** 리소스를 생성합니다.

```

apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
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
EOF

```

1

외부 **DNS CR**의 이름을 지정합니다.

2

기본적으로 모든 호스팅 영역이 잠재적인 대상으로 선택됩니다. 필요한 호스팅 영역을 포함할 수 있습니다.

3

대상 영역의 도메인 일치는 정확해야 합니다(정식 표현식과 대조되는 경우).

4

업데이트할 영역의 정확한 도메인을 지정합니다. 경로의 호스트 이름은 지정된 도메인의 하위 도메인이어야 합니다.

5

**Google Cloud DNS** 공급자를 정의합니다.

6

**DNS 레코드 소스**에 대한 옵션을 정의할 수 있습니다.

7

8

**OpenShift** 경로 리소스를 이전에 지정한 **DNS** 공급자에서 생성되는 **DNS** 레코드의 소스로 정의합니다.

9.

다음 명령을 사용하여 **OCP** 경로에 대해 생성된 레코드를 확인합니다.

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

## 16.7. INFOBLOX에서 DNS 레코드 생성

**Red Hat External DNS Operator**를 사용하여 **Infoblox**에서 **DNS** 레코드를 생성할 수 있습니다.

### 16.7.1. Infoblox의 퍼블릭 DNS 영역에서 DNS 레코드 생성

**Red Hat External DNS Operator**를 사용하여 **Infoblox**의 퍼블릭 **DNS** 영역에 **DNS** 레코드를 생성할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**에 액세스할 수 있습니다.
- **Infoblox UI**에 액세스할 수 있습니다.

#### 프로세스

1.

다음 명령을 실행하여 **Infoblox** 인증 정보로 보안 오브젝트를 생성합니다.

```
$ 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.

다음 명령을 실행하여 **routes** 오브젝트를 가져와 클러스터 도메인을 확인합니다.

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

출력 예

```

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. 다음과 같이 **ExternalDNS** 리소스 **YAML** 파일을 생성합니다(예: **sample-infoblox.yaml**).

```

apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-infoblox
spec:
  provider:
    type: Infoblox
    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
    openshiftRouteOptions:
      routerName: default
    
```

4. 다음 명령을 실행하여 **Infoblox**에서 **ExternalDNS** 리소스를 생성합니다.

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

5. **Infoblox UI**에서 콘솔 경로에 대해 생성된 **DNS** 레코드를 확인합니다.

- a. 데이터 관리 → **DNS** → 영역을 클릭합니다.

- b.  
영역 이름을 선택합니다.

## 16.8. 외부 DNS OPERATOR에서 클러스터 전체 프록시 구성

외부 DNS Operator에서 클러스터 전체 프록시를 구성할 수 있습니다. 외부 DNS Operator에서 클러스터 전체 프록시를 구성한 후 OLM(Operator Lifecycle Manager)은 HTTP\_PROXY, HTTPS\_PROXY 및 NO\_PROXY 와 같은 환경 변수를 사용하여 Operator의 모든 배포를 자동으로 업데이트합니다.

### 16.8.1. 클러스터 전체 프록시의 인증 기관을 신뢰하도록 외부 DNS Operator 구성

클러스터 전체 프록시의 인증 기관을 신뢰하도록 외부 DNS Operator를 구성할 수 있습니다.

#### 프로세스

1. 다음 명령을 실행하여 외부-dns-operator 네임스페이스에 CA 번들을 포함하도록 구성 맵을 생성합니다.

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

2. 신뢰할 수 있는 CA 번들을 구성 맵에 삽입하려면 다음 명령을 실행하여 config.openshift.io/inject-trusted-cabundle=true 라벨을 구성 맵에 추가합니다.

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

3. 다음 명령을 실행하여 외부 DNS Operator의 서브스크립션을 업데이트합니다.

```
$ 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"}]}]'
```

#### 검증

- 외부 DNS Operator 배포가 완료되면 다음 명령을 실행하여 신뢰할 수 있는 CA 환경 변수가 external-dns-operator 배포에 추가되었는지 확인합니다.

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

출력 예

**|** *trusted-ca*



## 17장. 네트워크 정책

### 17.1. 네트워크 정책 정의

클러스터 관리자는 클러스터의 **pod**로 트래픽을 제한하는 네트워크 정책을 정의할 수 있습니다.

#### 17.1.1. 네트워크 정책 정의

**Kubernetes** 네트워크 정책을 지원하는 **CNI(Kubernetes Container Network Interface)** 플러그인을 사용하는 클러스터에서 네트워크 격리는 **NetworkPolicy** 개체에 의해서만 제어됩니다. **OpenShift Container Platform 4.11**에서 **OpenShift SDN**은 기본 네트워크 격리 모드에서 네트워크 정책의 사용을 지원합니다.



#### 주의

네트워크 정책은 호스트 네트워크 네임스페이스에 적용되지 않습니다. 호스트 네트워킹이 활성화된 **Pod**는 네트워크 정책 규칙의 영향을 받지 않습니다.

기본적으로 네트워크 정책 모드에서는 다른 **Pod** 및 네트워크 끝점에서 프로젝트의 모든 **Pod**에 액세스할 수 있습니다. 프로젝트에서 하나 이상의 **Pod**를 분리하기 위해 해당 프로젝트에서 **NetworkPolicy** 오브젝트를 생성하여 수신되는 연결을 표시할 수 있습니다. 프로젝트 관리자는 자신의 프로젝트 내에서 **NetworkPolicy** 오브젝트를 만들고 삭제할 수 있습니다.

하나 이상의 **NetworkPolicy** 오브젝트에서 선택기와 **Pod**가 일치하면 **Pod**는 해당 **NetworkPolicy** 오브젝트 중 하나 이상에서 허용되는 연결만 허용합니다. **NetworkPolicy** 오브젝트가 선택하지 않은 **Pod**에 완전히 액세스할 수 있습니다.

다음 예제 **NetworkPolicy** 오브젝트는 다양한 시나리오 지원을 보여줍니다.

- 

모든 트래픽 거부:

기본적으로 프로젝트를 거부하려면 모든 **Pod**와 일치하지만 트래픽을 허용하지 않는 **NetworkPolicy** 오브젝트를 추가합니다.

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector: {}
  ingress: []

```

- **OpenShift Container Platform Ingress** 컨트롤러의 연결만 허용합니다.

프로젝트에서 **OpenShift Container Platform Ingress** 컨트롤러의 연결만 허용하도록 하려면 다음 **NetworkPolicy** 개체를 추가합니다.

```

apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
  podSelector: {}
  policyTypes:
    - Ingress

```

- 프로젝트 내 **Pod** 연결만 허용:

**Pod**가 동일한 프로젝트 내 다른 **Pod**의 연결은 수락하지만 다른 프로젝트에 속하는 **Pod**의 기타 모든 연결을 거부하도록 하려면 다음 **NetworkPolicy** 오브젝트를 추가합니다.

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector: {}
  ingress:
    - from:
      - podSelector: {}

```

- **Pod** 레이블을 기반으로 하는 **HTTP** 및 **HTTPS** 트래픽만 허용:

특정 레이블(다음 예에서 **role=frontend**)을 사용하여 **Pod**에 대한 **HTTP** 및 **HTTPS** 액세스만 활성화하려면 다음과 유사한 **NetworkPolicy** 오브젝트를 추가합니다.

```

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

```

네임스페이스와 Pod 선택기를 모두 사용하여 연결 수락:

네임스페이스와 Pod 선택기를 결합하여 네트워크 트래픽을 일치시키려면 다음과 유사한 NetworkPolicy 오브젝트를 사용하면 됩니다.

```

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 오브젝트는 추가 기능이므로 여러 NetworkPolicy 오브젝트를 결합하여 복잡한 네트워크 요구 사항을 충족할 수 있습니다.

예를 들어, 이전 샘플에서 정의된 NetworkPolicy 오브젝트의 경우 동일한 프로젝트 내에서 **allow-same-namespace** 정책과 **allow-http-and-https** 정책을 모두 정의할 수 있습니다. 따라서 레이블이 **role=frontend**로 지정된 Pod는 각 정책에서 허용하는 모든 연결을 허용할 수 있습니다. 즉 동일한 네임스페이스에 있는 Pod의 모든 포트 연결과 모든 네임스페이스에 있는 Pod에서 포트 80 및 443에 대한 연결이 허용됩니다.

### 17.1.2. 네트워크 정책 최적화

네트워크 정책을 사용하여 네임스페이스 내의 라벨에 따라 서로 다른 포드를 분리합니다.



#### 참고

네트워크 정책 규칙을 효율적으로 사용하기 위한 지침은 **OpenShift SDN 클러스터 네트워크 공급자에게만** 적용됩니다.

**NetworkPolicy** 오브젝트를 단일 네임스페이스에서 개별 포드의 많은 수에 적용하는 것은 비효율적입니다. 포드 라벨은 IP 주소 수준에 존재하지 않으므로 네트워크 정책은 **podSelector**로 선택한 모든 포드 간에 가능한 모든 링크에 대한 별도의 **OVS(Open vSwitch)** 흐름 규칙을 생성합니다.

예를 들어 **NetworkPolicy** 오브젝트 내의 **spec podSelector** 및 **ingress podSelector**가 각각 200개의 포드와 일치하는 경우 **40,000(200\*200)**개의 **OVS** 흐름 규칙이 생성됩니다. 이 경우 노드가 느려질 수 있습니다.

네트워크 정책을 설계할 때 다음 지침을 참조하십시오.

- 분리해야 하는 포드 그룹을 포함하도록 네임스페이스를 사용하여 **OVS** 흐름 규칙의 수를 줄입니다.  
  
**namespaceSelector** 또는 빈 **podSelector**를 사용하여 전체 네임스페이스를 선택하는 **NetworkPolicy** 오브젝트는 네임스페이스의 **VXLAN** 가상 네트워크 ID(**VNID**)와 일치하는 단일 **OVS** 흐름 규칙만 생성합니다.
- 원래 네임스페이스에서 분리할 필요가 없는 포드를 유지하고, 분리해야 하는 포드를 하나 이상의 네임스페이스로 이동합니다.
- 분리된 포드에서 허용하려는 특정 트래픽을 허용하도록 추가 대상의 네임스페이스 간 네트워크 정책을 생성합니다.

### 17.1.3. 다음 단계

- [네트워크 정책 생성](#)

- [선택사항: 기본 네트워크 정책 정의](#)

#### 17.1.4. 추가 리소스

- [프로젝트 및 네임스페이스](#)
- [다중 테넌트 네트워크 정책 구성](#)
- [NetworkPolicy API](#)

## 17.2. 네트워크 정책 이벤트 로깅

클러스터 관리자는 클러스터에 대한 네트워크 정책 감사 로깅을 구성하고 하나 이상의 네임스페이스에 대해 로깅을 활성화할 수 있습니다.



### 참고

네트워크 정책의 감사 로깅은 [OVN-Kubernetes 클러스터 네트워크 공급자](#)에서만 사용할 수 있습니다.

### 17.2.1. 네트워크 정책 감사 로깅

[OVN-Kubernetes 클러스터 네트워크 공급자](#)는 [OVN\(Open Virtual Network\) ACL](#)을 사용하여 네트워크 정책을 관리합니다. 감사 로깅은 [ACL](#) 이벤트를 허용 및 거부합니다.

[syslog](#) 서버 또는 [UNIX](#) 도메인 소켓과 같은 네트워크 정책 감사 로그의 대상을 구성할 수 있습니다. 추가 구성에 관계없이 감사 로그는 항상 클러스터의 각 [OVN-Kubernetes Pod](#)의 `/var/log/ovn/acl-audit-log.log`에 저장됩니다.

다음 예와 같이 [k8s.ovn.org/acl-logging](#) 키로 네임스페이스 주석을 달아 네임스페이스별로 네트워크 정책 감사 로깅을 사용할 수 있습니다.

네임스페이스 주석의 예

■

```

kind: Namespace
apiVersion: v1
metadata:
  name: example1
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "info",
        "allow": "info"
      }
    
```

로깅 형식은 RFC5424에 정의된 대로 **syslog**와 호환됩니다. **syslog** 기능은 구성 가능하며 기본값은 **local0**입니다. 예제 로그 항목은 다음과 유사합니다.

**ACL 거부 로그 항목의 예**

```

2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-
all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,n
w_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
    
```

다음 표에서는 네임스페이스 주석 값에 대해 설명합니다.

**표 17.1. 네트워크 정책 감사 로깅 네임스페이스 주석**

주석	값
<b>k8s.ovn.org/acl-logging</b>	네임스페이스에 대해 네트워크 정책 감사 로깅을 활성화하려면 <b>allow,deny</b> 또는 둘 중 하나를 지정해야 합니다.  <b>deny</b> 선택 사항: <b>alert, warning, notice, info, debug</b> 를 지정합니다.  <b>allow</b> 선택 사항: <b>alert, warning, notice, info, debug</b> 를 지정합니다.

**17.2.2. 네트워크 정책 감사 구성**

감사 로깅 구성은 **OVN-Kubernetes** 클러스터 네트워크 공급자 구성의 일부로 지정됩니다. 다음 **YAML**은 네트워크 정책 감사 로깅 기능의 기본값을 보여줍니다.

#### 감사 로깅 구성

```

apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
        syslogFacility: local0
  
```

다음 표에서는 네트워크 정책 감사 로깅을 위한 구성 필드를 설명합니다.

표 17.2. **policyAuditConfig** 오브젝트

필드	유형	설명
<b>rateLimit</b>	integer	노드당 1초마다 생성할 최대 메시지 수입니다. 기본값은 초당 <b>20</b> 개의 메시지입니다.
<b>maxFileSize</b>	integer	감사 로그의 최대 크기(바이트)입니다. 기본값은 <b>50000000</b> 또는 50MB입니다.
대상	string	다음 추가 감사 로그 대상 중 하나입니다. <b>libc</b> 호스트에서 journald 프로세스의 libc <b>syslog()</b> 함수입니다. <b>udp:&lt;host&gt;:&lt;port&gt;</b> syslog 서버입니다. <b>&lt;host&gt;:&lt;port&gt;</b> 를 syslog 서버의 호스트 및 포트로 바꿉니다. <b>unix:&lt;file&gt;</b> <b>&lt;file&gt;</b> 로 지정된 Unix Domain Socket 파일입니다. <b>null</b> 감사 로그를 추가 대상으로 보내지 마십시오.

필드	유형	설명
<b>syslogFacility</b>	string	RFC5424에 정의된 <b>kern</b> 과 같은 syslog 기능입니다. 기본값은 <b>local0</b> 입니다.

### 17.2.3. 클러스터에 대한 네트워크 정책 감사 구성

클러스터 관리자는 클러스터의 네트워크 정책 감사 로깅을 사용자 지정할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

#### 프로세스

- 네트워크 정책 감사 로깅 구성을 사용자 정의하려면 다음 명령을 입력합니다.

```
$ oc edit network.operator.openshift.io/cluster
```

#### 작은 정보

또는 다음 **YAML**을 사용자 지정하고 적용하여 감사 로깅을 구성할 수 있습니다.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
        syslogFacility: local0
```

#### 검증



1.

네트워크 정책을 사용하여 네임스페이스를 생성하려면 다음 단계를 완료합니다.

a.

검증을 위해 네임스페이스를 생성합니다.

```
$ 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
```

출력 예

```
namespace/verify-audit-logging created
```

b.

감사 로깅을 활성화합니다.

```
$ oc annotate namespace verify-audit-logging k8s.ovn.org/acl-logging='{ "deny":
"alert", "allow": "alert" }'
```

```
namespace/verify-audit-logging annotated
```

c.

네임스페이스의 네트워크 정책을 생성합니다.

```
$ 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
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
      - podSelector: {}
  egress:
    - to:
      - namespaceSelector:
        matchLabels:
          namespace: verify-audit-logging
EOF

```

출력 예

```

networkpolicy.networking.k8s.io/deny-all created
networkpolicy.networking.k8s.io/allow-from-same-namespace created

```

2.

**default** 네임스페이스에서 소스 트래픽에 사용할 **Pod**를 생성합니다.

```

$ 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.

**verify-audit-logging** 네임스페이스에 두 개의 **Pod**를 생성합니다.

```

$ 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

```

출력 예

```

pod/client created
pod/server created

```

4.

트래픽을 생성하고 네트워크 정책 감사 로그 항목을 생성하려면 다음 단계를 완료합니다.

a.

**verify-audit-logging** 네임스페이스에서 **server**라는 Pod의 IP 주소를 가져옵니다.

```

$ POD_IP=$(oc get pods server -n verify-audit-logging -o
jsonpath='{.status.podIP}')

```

b.

**default** 네임스페이스에 있는 **client**라는 Pod에서 이전 명령의 IP 주소를 ping하고 모든 패킷이 삭제되었는지 확인합니다.

```

$ oc exec -it client -n default -- /bin/ping -c 2 $POD_IP

```

출력 예

```

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.

**verify-audit-logging** 네임스페이스의 **client**라는 Pod에서 **POD\_IP** 셀 환경 변수에 저장된 IP 주소를 ping하고 모든 패킷이 허용되는지 확인합니다.

```
$ oc exec -it client -n verify-audit-logging -- /bin/ping -c 2 $POD_IP
```

출력 예

```
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.

네트워크 정책 감사 로그의 최신 항목을 표시합니다.

```
$ 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
```

출력 예

```
Defaulting container name to ovn-controller.
Use 'oc describe pod/ovnkube-node-hdb8v -n openshift-ovn-kubernetes' to see all of
the containers in this pod.
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.12
8.2.57,nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:33:12.614Z|00006|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.12
8.2.57,nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:10.037Z|00007|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.1
28.2.59,nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:11.037Z|00008|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
```

```
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.1
28.2.59,nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

#### 17.2.4. 네임스페이스에 대한 네트워크 정책 감사 로깅 활성화

클러스터 관리자는 네임스페이스에 대한 기존 네트워크 정책 감사 로깅을 활성화할 수 있습니다.

##### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

##### 프로세스

- 네임스페이스의 네트워크 정책 감사 로깅을 활성화하려면 다음 명령을 입력합니다.

```
$ oc annotate namespace <namespace> \
k8s.ovn.org/acl-logging='{ "deny": "alert", "allow": "notice" }'
```

다음과 같습니다.

**<namespace>**

네임스페이스의 이름을 지정합니다.

## 작은 정보

다음 **YAML**을 적용하여 감사 로깅을 활성화할 수 있습니다.

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "alert",
        "allow": "notice"
      }
```

## 출력 예

```
namespace/verify-audit-logging annotated
```

## 검증

- 네트워크 정책 감사 로그의 최신 항목을 표시합니다.

```
$ 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
```

## 출력 예

```
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

## 17.2.5. 네임스페이스의 네트워크 정책 감사 로깅 비활성화

클러스터 관리자는 네임스페이스에 대한 네트워크 정책 감사 로깅을 비활성화할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

#### 절차

- 네임스페이스에 대한 네트워크 정책 감사 로깅을 비활성화하려면 다음 명령을 입력합니다.

```
$ oc annotate --overwrite namespace <namespace> k8s.ovn.org/acl-logging-
```

다음과 같습니다.

**<namespace>**

네임스페이스의 이름을 지정합니다.

#### 작은 정보

다음 **YAML**을 적용하여 감사 로깅을 비활성화할 수 있습니다.

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: null
```

#### 출력 예

```
namespace/verify-audit-logging annotated
```

### 17.2.6. 추가 리소스

- [네트워크 정책 정의](#)

### 17.3. 네트워크 정책 생성

**admin** 역할이 있는 사용자는 네임스페이스에 대한 네트워크 정책을 생성할 수 있습니다.

#### 17.3.1. NetworkPolicy 오브젝트 예

다음은 예제 NetworkPolicy 오브젝트에 대한 주석입니다.

```

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

NetworkPolicy 오브젝트의 이름입니다.

②

정책이 적용되는 Pod를 설명하는 선택기입니다. 정책 오브젝트는 NetworkPolicy 오브젝트를 정의하는 프로젝트에서만 Pod를 선택할 수 있습니다.

③

정책 오브젝트가 수신 트래픽을 허용하는 Pod와 일치하는 선택기입니다. 선택기는 NetworkPolicy와 동일한 네임스페이스의 Pod와 일치합니다.

④



트래픽을 허용할 하나 이상의 대상 포트 목록입니다.

### 17.3.2. CLI를 사용하여 네트워크 정책 생성

클러스터의 네임스페이스에서 허용된 수신 또는 송신 네트워크 트래픽을 설명하는 세분화된 규칙을 정의하기 위해 네트워크 정책을 생성할 수 있습니다.



#### 참고

**cluster-admin** 역할로 사용자로 로그인하는 경우 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

#### 사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자(예: **mode: NetworkPolicy** 로 설정된 **OpenShift SDN** 네트워크 공급자)를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

#### 절차

1. 다음과 같이 정책 규칙을 생성합니다.
  - a. **<policy\_name>.yaml** 파일을 생성합니다.

```
$ touch <policy_name>.yaml
```

다음과 같습니다.

**<policy\_name>**

네트워크 정책 파일 이름을 지정합니다.

b.

방금 만든 파일에서 다음 예와 같이 네트워크 정책을 정의합니다.

모든 네임스페이스의 모든 **Pod**에서 수신 거부

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
    ingress: []
```

동일한 네임 스페이스에 있는 모든 **Pod**의 수신 허용

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
    ingress:
      - from:
        - podSelector: {}
```

2.

다음 명령을 실행하여 네트워크 정책 오브젝트를 생성합니다.

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

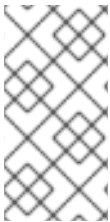
네트워크 정책 파일 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

출력 예

```
networkpolicy.networking.k8s.io/default-deny created
```



참고

**cluster-admin** 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 **YAML** 또는 웹 콘솔의 양식에서 직접 네트워크 정책을 생성할 수 있습니다.

### 17.3.3. 추가 리소스

- [웹 콘솔에 액세스](#)

## 17.4. 네트워크 정책 보기

**admin** 역할이 있는 사용자는 네임스페이스에 대한 네트워크 정책을 볼 수 있습니다.

### 17.4.1. NetworkPolicy 오브젝트 예

다음은 예제 **NetworkPolicy** 오브젝트에 대한 주석입니다.

```
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

NetworkPolicy 오브젝트의 이름입니다.

2

정책이 적용되는 Pod를 설명하는 선택기입니다. 정책 오브젝트는 NetworkPolicy 오브젝트를 정의하는 프로젝트에서만 Pod를 선택할 수 있습니다.

3

정책 오브젝트가 수신 트래픽을 허용하는 Pod와 일치하는 선택기입니다. 선택기는 NetworkPolicy와 동일한 네임스페이스의 Pod와 일치합니다.

4

트래픽을 허용할 하나 이상의 대상 포트 목록입니다.

### 17.4.2. CLI를 사용하여 네트워킹 정책 보기

네임스페이스에서 네트워킹 정책을 검사할 수 있습니다.



#### 참고

**cluster-admin** 역할을 가진 사용자로 로그인하면 클러스터의 모든 네트워킹 정책을 볼 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.

- **admin 권한이 있는 사용자로 클러스터에 로그인합니다.**
- **네트워크 정책이 존재하는 네임스페이스에서 작업하고 있습니다.**

#### 프로세스

- **네임스페이스의 네트워크 정책을 나열합니다.**
  - **네임스페이스에 정의된 네트워크 정책 개체를 보려면 다음 명령을 입력합니다.**

```
$ oc get networkpolicy
```

- **선택 사항: 특정 네트워크 정책을 검사하려면 다음 명령을 입력합니다.**

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

검사할 네트워크 정책의 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

예를 들어 다음과 같습니다.

```
$ oc describe networkpolicy allow-same-namespace
```

**oc describe** 명령의 출력

```
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

```



참고

**cluster-admin** 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 **YAML** 또는 웹 콘솔의 양식에서 직접 네트워크 정책을 볼 수 있습니다.

### 17.5. 네트워크 정책 편집

관리자 역할이 있는 사용자는 네임스페이스에 대한 기존 네트워크 정책을 편집할 수 있습니다.

#### 17.5.1. 네트워크 정책 편집

네임스페이스에서 네트워크 정책을 편집할 수 있습니다.



참고

**cluster-admin** 역할을 가진 사용자로 로그인하면 클러스터의 모든 네임스페이스에서 네트워크 정책을 편집할 수 있습니다.

#### 사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자(예: **mode: NetworkPolicy** 로 설정된 **OpenShift SDN** 네트워크 공급자)를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.

- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 존재하는 네임스페이스에서 작업하고 있습니다.

#### 프로세스

1. 선택 사항: 네임스페이스의 네트워크 정책 개체를 나열하려면 다음 명령을 입력합니다.

```
$ oc get networkpolicy
```

다음과 같습니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

2. 네트워크 정책 오브젝트를 편집합니다.

- 네트워크 정책 정의를 파일에 저장한 경우 파일을 편집하고 필요한 사항을 변경한 후 다음 명령을 입력합니다.

```
$ oc apply -n <namespace> -f <policy_file>.yaml
```

다음과 같습니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

**<policy\_file>**

네트워크 정책이 포함된 파일의 이름을 지정합니다.

- 네트워크 정책 개체를 직접 업데이트해야 하는 경우 다음 명령을 입력합니다.

■

```
$ oc edit networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

네트워크 정책의 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

3.

네트워크 정책 개체가 업데이트되었는지 확인합니다.

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

네트워크 정책의 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.



참고

**cluster-admin** 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 YAML에서 직접 또는 작업 메뉴를 통해 웹 콘솔의 정책에서 네트워크 정책을 편집할 수 있습니다.

### 17.5.2. NetworkPolicy 오브젝트 예

다음은 예제 NetworkPolicy 오브젝트에 대한 주석입니다.

```
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

**NetworkPolicy** 오브젝트의 이름입니다.

②

정책이 적용되는 **Pod**를 설명하는 선택기입니다. 정책 오브젝트는 **NetworkPolicy** 오브젝트를 정의하는 프로젝트에서만 **Pod**를 선택할 수 있습니다.

③

정책 오브젝트가 수신 트래픽을 허용하는 **Pod**와 일치하는 선택기입니다. 선택기는 **NetworkPolicy**와 동일한 네임스페이스의 **Pod**와 일치합니다.

④

트래픽을 허용할 하나 이상의 대상 포트 목록입니다.

### 17.5.3. 추가 리소스

- [네트워크 정책 생성](#)

## 17.6. 네트워크 정책 삭제

**admin** 역할이 있는 사용자는 네임스페이스에서 네트워크 정책을 삭제할 수 있습니다.

### 17.6.1. CLI를 사용하여 네트워크 정책 삭제

네임스페이스에서 네트워크 정책을 삭제할 수 있습니다.



#### 참고

**cluster-admin** 역할을 가진 사용자로 로그인하면 클러스터의 모든 네트워크 정책을 삭제할 수 있습니다.

#### 사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자(예: **mode: NetworkPolicy** 로 설정된 **OpenShift SDN** 네트워크 공급자)를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 존재하는 네임스페이스에서 작업하고 있습니다.

#### 프로세스

- 네트워크 정책 개체를 삭제하려면 다음 명령을 입력합니다.

```
$ oc delete networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

네트워크 정책의 이름을 지정합니다.

**<namespace>**

**선택 사항:** 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

#### 출력 예

[networkpolicy.networking.k8s.io/default-deny-deleted](https://networkpolicy.networking.k8s.io/default-deny-deleted)



참고

**cluster-admin** 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 **YAML**에서 직접 또는 작업 메뉴를 통해 웹 콘솔의 정책에서 네트워크 정책을 삭제할 수 있습니다.

## 17.7. 프로젝트의 기본 네트워크 정책 정의

클러스터 관리자는 새 프로젝트를 만들 때 네트워크 정책을 자동으로 포함하도록 새 프로젝트 템플릿을 수정할 수 있습니다. 새 프로젝트에 대한 사용자 정의 템플릿이 아직 없는 경우에는 우선 생성해야 합니다.

### 17.7.1. 새 프로젝트의 템플릿 수정

클러스터 관리자는 사용자 정의 요구 사항을 사용하여 새 프로젝트를 생성하도록 기본 프로젝트 템플릿을 수정할 수 있습니다.

사용자 정의 프로젝트 템플릿을 만들려면:

프로세스

1. **cluster-admin** 권한이 있는 사용자로 로그인합니다.
2. 기본 프로젝트 템플릿을 생성합니다.

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. 텍스트 편집기를 사용하여 오브젝트를 추가하거나 기존 오브젝트를 수정하여 생성된 **template.yaml** 파일을 수정합니다.
- 4.

프로젝트 템플릿은 **openshift-config** 네임스페이스에서 생성해야 합니다. 수정된 템플릿을 불러옵니다.

```
$ oc create -f template.yaml -n openshift-config
```

5.

웹 콘솔 또는 **CLI**를 사용하여 프로젝트 구성 리소스를 편집합니다.

- 

웹 콘솔에 액세스:

- i.

관리 → 클러스터 설정으로 이동합니다.

- ii.

구성을 클릭하여 모든 구성 리소스를 확인합니다.

- iii.

프로젝트 항목을 찾아 **YAML** 편집을 클릭합니다.

- 

**CLI** 사용:

- i.

다음과 같이 **project.config.openshift.io/cluster** 리소스를 편집합니다.

```
$ oc edit project.config.openshift.io/cluster
```

6.

**projectRequestTemplate** 및 **name** 매개변수를 포함하도록 **spec** 섹션을 업데이트하고 업로드된 프로젝트 템플릿의 이름을 설정합니다. 기본 이름은 **project-request**입니다.

사용자 정의 프로젝트 템플릿이 포함된 프로젝트 구성 리소스

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7.

변경 사항을 저장한 후 새 프로젝트를 생성하여 변경 사항이 성공적으로 적용되었는지 확인합니다.

### 17.7.2. 새 프로젝트 템플릿에 네트워크 정책 추가

클러스터 관리자는 네트워크 정책을 새 프로젝트의 기본 템플릿에 추가할 수 있습니다. **OpenShift Container Platform**은 프로젝트의 템플릿에 지정된 모든 **NetworkPolicy** 개체를 자동으로 생성합니다.

#### 사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 기본 **CNI** 네트워크 공급자(예: **mode: NetworkPolicy**로 설정된 **OpenShift SDN** 네트워크 공급자)를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인해야 합니다.
- 새 프로젝트에 대한 사용자 정의 기본 프로젝트 템플릿을 생성해야 합니다.

#### 프로세스

1.

다음 명령을 실행하여 새 프로젝트의 기본 템플릿을 편집합니다.

```
$ oc edit template <project_template> -n openshift-config
```

**<project\_template>**을 클러스터에 대해 구성한 기본 템플릿의 이름으로 변경합니다. 기본 템플릿 이름은 **project-request**입니다.

2.

템플릿에서 각 **NetworkPolicy** 오브젝트를 **objects** 매개변수의 요소로 추가합니다. **objects** 매개변수는 하나 이상의 오브젝트 컬렉션을 허용합니다.

다음 예제에서 **objects** 매개변수 컬렉션에는 여러 **NetworkPolicy** 오브젝트가 포함됩니다.

```

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:
            network.openshift.io/policy-group: 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.

선택 사항: 다음 명령을 실행하여 새 프로젝트를 생성하고 네트워크 정책 오브젝트가 생성되었는지 확인합니다.

a.

새 프로젝트를 생성합니다.

```
$ oc new-project <project> 1
```

1

<project> 를 생성 중인 프로젝트의 이름으로 변경합니다.

b.

새 프로젝트 템플릿의 네트워크 정책 오브젝트가 새 프로젝트에 있는지 확인합니다.

```
$ oc get networkpolicy
NAME                POD-SELECTOR  AGE
allow-from-openshift-ingress <none>       7s
allow-from-same-namespace <none>       7s
```

## 17.8. 네트워크 정책으로 다중 테넌트 격리 구성

클러스터 관리자는 다중 테넌트 네트워크 격리를 제공하도록 네트워크 정책을 구성할 수 있습니다.



참고

**OpenShift SDN** 클러스터 네트워크 공급자를 사용하는 경우 이 섹션에 설명된 대로 네트워크 정책을 구성하는 경우 다중 테넌트 모드와 유사하지만 네트워크 정책 모드가 설정된 네트워크 격리를 제공합니다.

### 17.8.1. 네트워크 정책을 사용하여 다중 테넌트 격리 구성

다른 프로젝트 네임스페이스의 Pod 및 서비스에서 격리하도록 프로젝트를 구성할 수 있습니다.

사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자(예: **mode: NetworkPolicy** 로 설정된 **OpenShift SDN** 네트워크 공급자)를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

프로세스

1.

다음 **NetworkPolicy** 오브젝트를 생성합니다.

a.

이름이 **allow-from-openshift-ingress**인 정책입니다.

```
$ 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
```



참고

**policy-group.network.openshift.io/ingress: ""** 는 OpenShift SDN의 기본 네임스페이스 선택기 레이블입니다. **network.openshift.io/policy-group: ingress** 네임스페이스 선택기 레이블을 사용할 수 있지만 이는 레거시 레이블입니다.

b.

이름이 **allow-from-openshift-monitoring**인 정책:

```
$ 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.

이름이 `allow-same-namespace`인 정책:

```
$ cat << EOF | oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
    ingress:
      - from:
        - podSelector: {}
EOF
```

d.

이름이 `allow-from-kube-apiserver-operator`인 정책:

```
$ 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
```

자세한 내용은 웹 후크의 상태를 검증하는 새로운 `kube-apiserver-operator` 웹 후크 컨트롤러를 참조하십시오.

2.

선택 사항: 현재 프로젝트에 네트워크 정책이 있는지 확인하려면 다음 명령을 입력합니다.

```
$ oc describe networkpolicy
```

출력 예

```
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:** *network.openshift.io/policy-group: 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*

### 17.8.2. 다음 단계

- [기본 네트워크 정책 정의](#)

### 17.8.3. 추가 리소스

- [OpenShift SDN 네트워크 격리 모드](#)

## 18장. AWS LOAD BALANCER OPERATOR

### 18.1. OPENSIFT CONTAINER PLATFORM의 AWS LOAD BALANCER OPERATOR

ALB(AWS Load Balancer) Operator는 `aws-load-balancer-controller`의 인스턴스를 배포 및 관리합니다. OpenShift Container Platform 웹 콘솔 또는 CLI를 사용하여 OperatorHub에서 ALB Operator를 설치할 수 있습니다.

#### 18.1.1. AWS Load Balancer Operator

`kubernetes.io/role/elb` 태그가 누락된 경우 AWS Load Balancer Operator에서 퍼블릭 서브넷에 태그를 지정할 수 있습니다. 또한 AWS Load Balancer Operator는 기본 AWS 클라우드에서 다음을 탐지합니다.

- Operator를 호스팅하는 클러스터가 배포되는 VPC(가상 프라이빗 클라우드)의 ID입니다.
- 검색된 VPC의 퍼블릭 및 프라이빗 서브넷입니다.

#### 사전 요구 사항

- AWS 인증 정보 시크릿이 있어야 합니다. 인증 정보는 서브넷 태그 지정 및 VPC 검색을 제공하는 데 사용됩니다.

#### 프로세스

1. Subscription 오브젝트를 생성하여 OperatorHub의 요구에 AWS Load Balancer Operator를 배포할 수 있습니다.

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

출력 예

```
install-zlfbt
```

2.

설치 계획의 상태를 확인합니다. 설치 계획의 상태는 **Complete** 여야 합니다.

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

출력 예

```
Complete
```

3.

**oc get** 명령을 사용하여 배포 상태를 확인합니다.

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

출력 예

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

### 18.1.2. AWS Load Balancer Operator 로그

**oc logs** 명령을 사용하여 **AWS Load Balancer Operator** 로그를 확인합니다.

프로세스

- **AWS Load Balancer Operator**의 로그를 확인합니다.

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

### 18.2. AWS LOAD BALANCER OPERATOR 이해

**ALB(AWS Load Balancer) Operator**는 `aws-load-balancer-controller` 리소스의 인스턴스를 배포 및 관리합니다. **OpenShift Container Platform** 웹 콘솔 또는 **CLI**를 사용하여 **OperatorHub**에서 **AWS Load Balancer Operator**를 설치할 수 있습니다.

### 중요

**AWS Load Balancer Operator** 배포는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

**Red Hat** 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

#### 18.2.1. AWS Load Balancer Operator 설치

**OpenShift Container Platform** 웹 콘솔을 사용하여 **OperatorHub**에서 **AWS Load Balancer Operator**를 설치할 수 있습니다.

#### 사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 **OpenShift Container Platform** 웹 콘솔에 로그인했습니다.

#### 프로세스

1. **OpenShift Container Platform** 웹 콘솔에서 **Operators** → **OperatorHub** 로 이동합니다.
2. **AWS Load Balancer Operator** 를 선택합니다. 키워드로 필터링 텍스트 상자를 사용하거나 필터 목록을 사용하여 **Operator** 목록에서 **AWS Load Balancer Operator**를 검색할 수 있습니다.
3. **aws-load-balancer-operator** 네임스페이스를 선택합니다.
4. 지침에 따라 **Operator**에서 설치를 준비합니다.

5.

**AWS Load Balancer Operator** 페이지에서 설치를 클릭합니다.

6.

**Operator** 설치 페이지에서 다음 옵션을 선택합니다.

a.

채널을 **stable-v0.1** 으로 업데이트합니다.

b.

클러스터의 특정 네임스페이스로 설치 모드.

c.

설치된 네임스페이스를 **aws-load-balancer-operator**. **aws-load-balancer-operator** 네임스페이스가 없는 경우 **Operator** 설치 중에 생성됩니다.

d.

승인 업데이트를 자동 또는 수동으로 선택합니다. 기본적으로 업데이트 승인은 자동으로 설정됩니다. 자동 업데이트를 선택하면 **OLM(Operator Lifecycle Manager)**에서 개입 없이 **Operator**의 실행 중인 인스턴스를 자동으로 업그레이드합니다. 수동 업데이트를 선택하면 **OLM**에서 업데이트 요청을 생성합니다. 클러스터 관리자는 **Operator**를 새 버전으로 업데이트하려면 해당 업데이트 요청을 수동으로 승인해야 합니다.

e.

설치를 클릭합니다.

## 검증

•

설치된 **Operator** 대시보드에서 **AWS Load Balancer Operator**에 상태가 **Succeeded**로 표시되는지 확인합니다.

## 18.3. AWS 로드 밸런서 컨트롤러 인스턴스 생성

**Operator**를 설치한 후 **AWS** 로드 밸런서 컨트롤러의 인스턴스를 생성할 수 있습니다.

### 18.3.1. AWS Load Balancer Operator를 사용하여 AWS Load Balancer 컨트롤러 인스턴스 생성

클러스터에 **aws-load-balancer-controller**의 단일 인스턴스만 설치할 수 있습니다. **CLI**를 사용하여 **AWS Load Balancer** 컨트롤러를 생성할 수 있습니다. **AWS Load Balancer(ALB) Operator**는 이름이 **cluster**인 리소스만 조정합니다.

## 사전 요구 사항

- `echoserver` 네임스페이스를 생성했습니다.
- `OpenShift CLI(oc)`에 액세스할 수 있습니다.

## 프로세스

1. 다음과 같이 `aws-load-balancer-controller` 리소스 YAML 파일을 생성합니다(예: `sample-aws-lb.yaml`).

```

apiVersion: networking.olm.openshift.io/v1alpha1
kind: AWSLoadBalancerController 1
metadata:
  name: cluster 2
spec:
  subnetTagging: Auto 3
  additionalResourceTags: 4
    example.org/cost-center: 5113232
    example.org/security-scope: staging
  ingressClass: cloud 5
config:
  replicas: 2 6
  enabledAddons: 7
    - AWSWAFv2 8

```

1

`aws-load-balancer-controller` 리소스를 정의합니다.

2

`AWS Load Balancer Controller` 인스턴스 이름을 정의합니다. 이 인스턴스 이름은 모든 관련 리소스에 접미사로 추가됩니다.

3

유효한 옵션은 `Auto` 및 `Manual`입니다. 값을 `Auto`로 설정하면 `Operator`에서 클러스터에 속하는 서브넷을 확인하고 적절하게 태그를 지정하려고 합니다. 내부 서브넷 태그가 내부 서브넷에 없는 경우 `Operator`에서 역할을 올바르게 결정할 수 없습니다. 사용자 제공 인 프라에 클러스터를 설치하는 경우 적절한 역할 태그로 서브넷에 수동으로 태그를 지정하고 서브넷 태그 지정 정책을 `Manual`로 설정할 수 있습니다.

4

`AWS` 리소스를 프로비저닝할 때 컨트롤러가 사용하는 태그를 정의합니다.

5

이 필드의 기본값은 **alb** 입니다. **Operator**는 이름이 없는 경우 **IngressClass** 리소스를 동일한 이름으로 프로비저닝합니다.

6

컨트롤러의 복제본 수를 지정합니다.

7

주석을 통해 지정된 **AWS** 로드 밸런서의 애드온을 지정합니다.

8

**alb.ingress.kubernetes.io/wafv2-acl-arn** 주석을 활성화합니다.

2.

다음 명령을 실행하여 **aws-load-balancer-controller** 리소스를 생성합니다.

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

3.

**AWS** 로드 밸런서 컨트롤러가 실행된 후 배포 리소스를 생성합니다.

```
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
          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

배포 리소스를 정의합니다.

2

배포 이름을 지정합니다.

3

배포 복제본 수를 지정합니다.

4.

서비스 리소스를 생성합니다.

```

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

서비스 리소스를 정의합니다.

2

서비스 이름을 지정합니다.

5.

ALB 지원 **Ingress** 리소스를 배포합니다.

```

apiVersion: networking.k8s.io/v1
kind: Ingress 1
metadata:
  name: <echoserver> 2
  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> 3
          port:
            number: 80

```

1

*Ingress* 리소스를 정의합니다.

2

*Ingress* 리소스의 이름을 지정합니다.

3

서비스 리소스의 이름을 지정합니다.

#### 검증

- 다음 명령을 실행하여 프로비저닝된 **AWS Load Balancer(ALB)** 호스트를 표시하는 *Ingress* 리소스의 상태를 확인합니다.

```

$ HOST=$(kubectl get ingress -n echoserver echoserver -o json | jq -r
'.status.loadBalancer.ingress[0].hostname')

```

- 다음 명령을 실행하여 프로비저닝된 **AWS Load Balancer(ALB)** 호스트의 상태를 확인합니다.

```

$ curl $HOST

```

#### 18.4. 여러 INGRESS 생성

단일 **AWS Load Balancer(ALB)**를 통해 단일 도메인에 속하는 다른 서비스로 트래픽을 라우팅할 수 있습니다. 각 *Ingress* 리소스는 도메인의 다양한 끝점을 제공합니다.

### 18.4.1. 단일 AWS 로드 밸런서를 통해 여러 인그레스 생성

CLI를 사용하여 단일 AWS Load Balancer(ALB)를 통해 트래픽을 여러 Ingress로 라우팅할 수 있습니다.

사전 요구 사항

- OpenShift CLI(oc)에 액세스할 수 있습니다.

프로세스

1. 다음과 같이 IngressClassParams 리소스 YAML 파일을 생성합니다(예: `sample-single-lb-params.yaml`).

```
apiVersion: elbv2.k8s.aws/v1beta1 1
kind: IngressClassParams
metadata:
  name: <single-lb-params> 2
spec:
  group:
    name: single-lb 3
```

**1**

IngressClassParams 리소스의 API 그룹 및 버전을 정의합니다.

**2**

IngressClassParams 리소스의 이름을 지정합니다.

**3**

IngressGroup의 이름을 지정합니다. 이 클래스의 모든 Ingress는 이 IngressGroup에 속합니다.

- 2.

다음 명령을 실행하여 IngressClassParams 리소스를 생성합니다.

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

- 3.

다음과 같이 IngressClass 리소스 YAML 파일을 생성합니다(예: `sample-single-lb.yaml`).

```

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 6

```

**1**

API 그룹 및 **IngressClass** 리소스의 버전을 정의합니다.

**2**

**IngressClass**의 이름을 지정합니다.

**3**

모든 **IngressClasses**에 공통인 컨트롤러 이름을 정의합니다. **aws-load-balancer-controller**가 컨트롤러를 조정합니다.

**4**

**IngressClassParams** 리소스의 API 그룹을 정의합니다.

**5**

**IngressClassParams** 리소스의 리소스 유형을 정의합니다.

**6**

**IngressClassParams** 리소스의 이름을 정의합니다.

4.

다음 명령을 실행하여 **IngressClass** 리소스를 생성합니다.

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

5.

다음과 같이 **Ingress** 리소스 YAML 파일을 생성합니다(예: **sample-multiple-ingress.yaml**).

```
apiVersion: networking.k8s.io/v1 1
```

```

kind: Ingress
metadata:
  name: <example-1> 2
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing 3
    alb.ingress.kubernetes.io/group.order: "1" 4
spec:
  ingressClass: alb 5
  rules:
  - host: example.com 6
    http:
      paths:
      - path: /blog 7
        backend:
          service:
            name: <example-1> 8
            port:
              number: 80 9

```

```

kind: Ingress
metadata:
  name: <example-2>
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "2"
spec:
  ingressClass: alb
  rules:
  - host: example.com
    http:
      paths:
      - path: /store
        backend:
          service:
            name: <example-2>
            port:
              number: 80

```

```

kind: Ingress
metadata:
  name: <example-3>
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/group.order: "3"
spec:
  ingressClass: alb
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        backend:
          service:
            name: <example-3>
            port:
              number: 80

```

1 2

*Ingress*의 이름을 지정합니다.

3

공용 서브넷에서 프로비저닝할 로드 밸런서를 나타내며 인터넷을 통해 액세스할 수 있도록 합니다.

4

로드 밸런서에서 요청이 수신될 때 *Ingress*의 규칙과 일치하는 순서를 지정합니다.

5

이 인그레스에 속하는 *Ingress* 클래스를 지정합니다.

6

요청 라우팅에 사용되는 도메인의 이름을 정의합니다.

7

서비스에 라우팅해야 하는 경로를 정의합니다.

8

*Ingress*에 구성된 끝점을 제공하는 서비스의 이름을 정의합니다.

9

엔드포인트를 제공하는 서비스의 포트를 정의합니다.

6.

다음 명령을 실행하여 *Ingress* 리소스를 생성합니다.

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

## 18.5. TLS 종료 추가

**AWS** 로드 밸런서에 **TLS** 종료를 추가할 수 있습니다.

### 18.5.1. AWS 로드 밸런서에 TLS 종료 추가

도메인의 트래픽을 서비스 Pod로 라우팅하고 AWS Load Balancer에서 TLS 종료를 추가할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**에 액세스할 수 있습니다.

#### 절차

1. **Operator**를 설치하고 **aws-load-balancer-controller** 리소스의 인스턴스를 생성합니다.

```
apiVersion: networking.k8s.io/v1
kind: AWSLoadBalancerController
group: networking.olm.openshift.io/v1alpha1 1
metadata:
  name: cluster 2
spec:
  subnetTagging: auto
  ingressClass: tls-termination 3
```

1

**aws-load-balancer-controller** 리소스의 **API** 그룹을 정의합니다.

2

**aws-load-balancer-controller** 인스턴스를 정의합니다.

3

**AWS** 로드 밸런서 컨트롤러에서 조정 한 **ingressClass** 리소스의 이름을 정의합니다. 이 **ingressClass** 리소스가 없으면 생성됩니다. **ingressClass** 값을 더 추가할 수 있습니다. **spec.controller** 가 **ingress.k8s.aws/alb** 로 설정된 경우 컨트롤러가 **ingressClass** 값을 조정합니다.

2. **인그레스 리소스**를 생성합니다.

```
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:xxxxx 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

*Ingress*의 이름을 지정합니다.

2

컨트롤러는 인터넷을 통해 로드 밸런서에 연결할 수 있도록 이 *Ingress* 리소스의 로드 밸런서를 공용 서브넷에 프로비저닝합니다.

3

로드 밸런서에 연결하는 인증서의 **Amazon** 리소스 이름입니다.

4

*Ingress* 클래스 이름을 정의합니다.

5

트래픽 라우팅에 대한 도메인을 정의합니다.

6

트래픽 라우팅에 대한 서비스를 정의합니다.



## 19장. 다중 네트워크

### 19.1. 다중 네트워크 이해하기

**Kubernetes**에서 컨테이너 네트워킹은 **CNI**(컨테이너 네트워크 인터페이스)를 구현하는 네트워킹 플러그인에 위임됩니다.

**OpenShift Container Platform**은 **Multus CNI** 플러그인을 사용하여 **CNI** 플러그인 체인을 허용합니다. 클러스터 설치 중에 기본 pod 네트워크를 구성합니다. 기본 네트워크는 클러스터의 모든 일반 네트워크 트래픽을 처리합니다. 사용 가능한 **CNI** 플러그인을 기반으로 추가 네트워크를 정의하고 이러한 네트워크 중 하나 이상을 Pod에 연결할 수 있습니다. 필요에 따라 클러스터에 2개 이상의 추가 네트워크를 정의할 수 있습니다. 따라서 스위칭 또는 라우팅과 같은 네트워크 기능을 제공하는 pod를 구성할 때 유연성이 제공됩니다.

#### 19.1.1. 추가 네트워크 사용 시나리오

데이터 플레인 및 컨트롤 플레인 분리를 포함하여 네트워크 격리가 필요한 상황에서 추가 네트워크를 사용할 수 있습니다. 네트워크 트래픽 격리는 다음과 같은 성능 및 보안상의 이유로 유용합니다.

##### 성능

각 플레인의 트래픽 수량을 관리하기 위해 두 개의 다른 플레인으로 트래픽을 보낼 수 있습니다.

##### 보안

보안 고려 사항을 위해 특별히 관리되는 네트워크 플레인으로 중요한 트래픽을 보낼 수 있으며 테넌트 또는 고객 간에 공유되지 않아야 하는 개인 데이터를 분리할 수 있습니다.

클러스터의 모든 pod는 여전히 클러스터 전체의 기본 네트워크를 사용하여 클러스터 전체의 연결을 유지합니다. 모든 pod에는 클러스터 전체 pod 네트워크에 연결된 eth0 인터페이스가 있습니다. `oc exec -it <pod_name> -- ip a` 명령을 사용하여 pod의 인터페이스를 확인할 수 있습니다. **Multus CNI**를 사용하는 네트워크 인터페이스를 추가하는 경우 이름은 `net1`, `net2`, ... , `netN`입니다.

Pod에 추가 네트워크 인터페이스를 연결하려면 인터페이스 연결 방법을 정의하는 구성을 생성해야 합니다. **NetworkAttachmentDefinition CR**(사용자 정의 리소스)을 사용하여 각 인터페이스를 지정합니다. 각 CR 내부의 CNI 구성은 해당 인터페이스의 생성 방법을 정의합니다.

#### 19.1.2. OpenShift Container Platform의 그룹은 중첩되지 않습니다.

**OpenShift Container Platform**은 클러스터에서 추가 네트워크를 생성하기 위해 다음 **CNI** 플러그인을 제공합니다.

- **bridge:** 동일한 호스트의 Pod가 서로 그리고 호스트와 통신할 수 있도록 브리지 기반 추가 네트워크를 구성합니다.
- **host-device:** pod가 호스트 시스템의 물리적 이더넷 네트워크 장치에 액세스할 수 있도록 호스트 장치 추가 네트워크를 구성합니다.
- **ipvlan:** macvlan 기반 추가 네트워크와 유사하게 호스트의 pod가 해당 호스트의 다른 호스트 및 pod와 통신할 수 있도록 ipvlan 기반 추가 네트워크를 구성합니다. macvlan 기반 추가 네트워크와 달리 각 pod는 상위 물리적 네트워크 인터페이스와 동일한 MAC 주소를 공유합니다.
- **macvlan:** 물리적 네트워크 인터페이스를 사용하여 호스트의 pod가 해당 호스트의 다른 호스트 및 pod와 통신할 수 있도록 macvlan 기반 추가 네트워크를 구성합니다. macvlan 기반 추가 네트워크에 연결된 각 pod에는 고유한 MAC 주소가 제공됩니다.
- **SR-IOV:** 호스트 시스템의 SR-IOV 가능 하드웨어에서 pod를 VF(가상 기능) 인터페이스에 연결할 수 있도록 SR-IOV 기반 추가 네트워크를 구성합니다.

## 19.2. 추가 네트워크 구성

클러스터 관리자는 클러스터에 대한 추가 네트워크를 구성할 수 있습니다. 다음과 같은 네트워크 유형이 지원됩니다.

- **Bridge**
- **호스트 장치**
- **IPVLAN**
- **MACVLAN**

### 19.2.1. 추가 네트워크 관리 방법

두 가지 방법으로 추가 네트워크의 라이프사이클을 관리할 수 있습니다. 각 접근 방식은 상호 배타적이

며 한 번에 추가 네트워크를 관리하는 데 하나의 방법만 사용할 수 있습니다. 두 방법 모두 추가 네트워크는 사용자가 구성하는 **CNI(Container Network Interface)** 플러그인에 의해 관리됩니다.

추가 네트워크의 경우 추가 네트워크의 일부로 구성하는 **IPAM(IP 주소 관리) CNI** 플러그인을 통해 **IP** 주소가 프로비저닝됩니다. **IPAM** 플러그인은 **DHCP** 및 고정 할당을 포함한 다양한 **IP** 주소 할당 방식을 지원합니다.

- CNO(Cluster Network Operator)** 설정을 수정합니다. **CNO**는 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성하고 관리합니다. **CNO**에서 오브젝트 라이프사이클을 관리하는 것 외에도 **DHCP**가 할당된 **IP** 주소를 사용하는 추가 네트워크에 **DHCP**를 사용할 수 있는지 확인합니다.
- YAML 매니페스트 적용: NetworkAttachmentDefinition** 오브젝트를 생성하여 직접 추가 네트워크를 관리할 수 있습니다. 이 방법을 사용하면 **CNI** 플러그인을 연결할 수 있습니다.

### 19.2.2. 추가 네트워크 연결을 위한 구성

추가 네트워크는 **k8s.cni.cncf.io** API 그룹의 **NetworkAttachmentDefinition** API를 통해 구성됩니다.



#### 중요

이 정보가 프로젝트 관리 사용자가 액세스할 수 있으므로 중요한 정보 또는 시크릿을 **NetworkAttachmentDefinition** 오브젝트에 저장하지 마십시오.

**API**의 구성은 다음 표에 설명되어 있습니다.

표 19.1. **NetworkAttachmentDefinition** API 필드

필드	유형	설명
<b>metadata.name</b>	<b>string</b>	추가 네트워크의 이름입니다.
<b>metadata.namespace</b>	<b>string</b>	오브젝트와 연결된 네임스페이스입니다.
<b>spec.config</b>	<b>string</b>	JSON 형식의 <b>CNI</b> 플러그인 구성입니다.

#### 19.2.2.1. Cluster Network Operator를 통한 추가 네트워크 구성

추가 네트워크 연결 구성은 **CNO(Cluster Network Operator)** 구성의 일부로 지정됩니다.

다음 **YAML**은 **CNO**를 사용하여 추가 네트워크를 관리하는 구성 매개변수를 설명합니다.

### CNO(Cluster Network Operator) 구성

```

apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
  additionalNetworks: 1
  - name: <name> 2
    namespace: <namespace> 3
    rawCNICfg: |- 4
      {
        ...
      }
  type: Raw

```

1

하나 이상의 추가 네트워크 구성으로 구성된 배열입니다.

2

생성 중인 추가 네트워크 연결의 이름입니다. 이름은 지정된 **namespace** 내에서 고유해야 합니다.

3

네트워크 연결을 생성할 네임스페이스입니다. 값을 지정하지 않으면 **default** 네임스페이스가 사용됩니다.

4

**JSON** 형식의 **CNI** 플러그인 구성입니다.

#### 19.2.2.2. YAML 매니페스트에서 추가 네트워크 구성

추가 네트워크의 구성은 다음 예와 같이 **YAML** 구성 파일에서 지정됩니다.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: <name> ❶
spec:
  config: |- ❷
  {
  ...
  }
```

❶

생성 중인 추가 네트워크 연결의 이름입니다.

❷

**JSON** 형식의 **CNI** 플러그인 구성입니다.

### 19.2.3. 추가 네트워크 유형의 구성

추가 네트워크의 특정 구성 필드는 다음 섹션에 설명되어 있습니다.

#### 19.2.3.1. 브리지 추가 네트워크에 대한 구성

다음 오브젝트는 브릿지 **CNI** 플러그인의 구성 매개변수를 설명합니다.

표 19.2. 브릿지 **CNI** 플러그인 **JSON** 구성 오브젝트

필드	유형	설명
<b>cniVersion</b>	<b>string</b>	CNI 사양 버전입니다. <b>0.3.1</b> 값이 필요합니다.
<b>name</b>	<b>string</b>	CNO 구성에 대해 이전에 입력한 <b>name</b> 매개변수의 값입니다.
<b>type</b>	<b>string</b>	
<b>bridge</b>	<b>string</b>	사용할 가상 브릿지의 이름을 지정합니다. 브릿지 인터페이스가 호스트에 없으면 생성됩니다. 기본값은 <b>cni0</b> 입니다.
<b>ipam</b>	<b>object</b>	IPAM CNI 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.

필드	유형	설명
ipMasq	boolean	가상 네트워크에서 전송되는 트래픽에 IP 마스크레이딩을 사용하려면 <b>true</b> 로 설정합니다. 모든 트래픽의 소스 IP 주소가 브리지의 IP 주소로 다시 작성됩니다. 브리지에 IP 주소가 없으면 이 설정이 적용되지 않습니다. 기본값은 <b>false</b> 입니다.
isGateway	boolean	브리지에 IP 주소를 할당하려면 <b>true</b> 로 설정합니다. 기본값은 <b>false</b> 입니다.
isDefaultGateway	boolean	브릿지를 가상 네트워크의 기본 게이트웨이로 구성하려면 <b>true</b> 로 설정합니다. 기본값은 <b>false</b> 입니다. <b>isDefaultGateway</b> 가 <b>true</b> 로 설정되면 <b>isGateway</b> 도 자동으로 <b>true</b> 로 설정됩니다.
forceAddress	boolean	이전에 할당된 IP 주소를 가상 브리지에 할당할 수 있도록 하려면 <b>true</b> 로 설정합니다. <b>false</b> 로 설정하면 중첩되는 하위 집합의 IPv4 주소 또는 IPv6 주소가 가상 브릿지에 지정되는 경우 오류가 발생합니다. 기본값은 <b>false</b> 입니다.
hairpinMode	boolean	가상 브릿지가 수신한 가상 포트를 통해 이더넷 프레임을 다시 보낼 수 있도록 하려면 <b>true</b> 로 설정합니다. 이 모드를 반사 릴레이라고도 합니다. 기본값은 <b>false</b> 입니다.
promiscMode	boolean	브릿지에서 무차별 모드를 사용하려면 <b>true</b> 로 설정합니다. 기본값은 <b>false</b> 입니다.
vlan	string	VLAN(가상 LAN) 태그를 정수 값으로 지정합니다. 기본적으로 VLAN 태그는 할당되지 않습니다.
mtu	string	최대 전송 단위(MTU)를 지정된 값으로 설정합니다. 기본값은 커널에 의해 자동으로 설정됩니다.

### 19.2.3.1.1. 브릿지 구성 예

다음 예제는 이름이 *bridge-net*인 추가 네트워크를 구성합니다.

```
{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
  }
}
```

### 19.2.3.2. 호스트 장치 추가 네트워크에 대한 구성



참고

**device**, **hwaddr**, **kernelpath** 또는 **pciBusID** 매개변수 중 하나만 설정하여 네트워크 장치를 지정합니다.

다음 오브젝트는 호스트 장치 CNI 플러그인의 구성 매개변수를 설명합니다.

표 19.3. 호스트 장치 CNI 플러그인 JSON 구성 오브젝트

필드	유형	설명
<b>cniVersion</b>	<b>string</b>	CNI 사양 버전입니다. <b>0.3.1</b> 값이 필요합니다.
<b>name</b>	<b>string</b>	CNO 구성에 대해 이전에 입력한 <b>name</b> 매개변수의 값입니다.
<b>type</b>	<b>string</b>	구성할 CNI 플러그인의 이름: <b>호스트 장치</b> .
<b>device</b>	<b>string</b>	선택사항: 장치 이름(예: <b>eth0</b> )입니다.
<b>hwaddr</b>	<b>string</b>	선택사항: 장치 하드웨어 MAC 주소입니다.
<b>kernelpath</b>	<b>string</b>	선택 사항: <b>/sys/devices/pci0000:00/0000:00:1f.6</b> 과 같은 Linux 커널 장치 경로입니다.
<b>pciBusID</b>	<b>string</b>	선택 사항: 네트워크 장치의 PCI 주소입니다(예: <b>0000:00:1f.6</b> ).

#### 19.2.3.2.1. 호스트 장치 구성 예

다음 예제는 이름이 **hostdev-net**인 추가 네트워크를 구성합니다.

```
{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "host-device",
  "device": "eth1"
}
```

#### 19.2.3.3. IPVLAN 추가 네트워크를 위한 구성

다음 오브젝트는 **IPVLAN CNI 플러그인**의 구성 매개변수를 설명합니다.

표 19.4. IPVLAN CNI 플러그인 JSON 구성 오브젝트

필드	유형	설명
<b>cniVersion</b>	<b>string</b>	CNI 사양 버전입니다. <b>0.3.1</b> 값이 필요합니다.
<b>name</b>	<b>string</b>	CNO 구성에 대해 이전에 입력한 <b>name</b> 매개변수의 값입니다.
<b>type</b>	<b>string</b>	구성할 CNI 플러그인의 이름: <b>ipvlan</b> .
<b>mode</b>	<b>string</b>	가상 네트워크의 운영 모드입니다. 값은 <b>I2</b> , <b>I3</b> 또는 <b>I3s</b> 여야 합니다. 기본값은 <b>I2</b> 입니다.
<b>master</b>	<b>string</b>	네트워크 연결과 연결할 이더넷 인터페이스입니다. <b>마스터</b> 를 지정하지 않으면 기본 네트워크 경로에 대한 인터페이스가 사용됩니다.
<b>mtu</b>	<b>integer</b>	최대 전송 단위(MTU)를 지정된 값으로 설정합니다. 기본값은 커널에 의해 자동으로 설정됩니다.
<b>ipam</b>	<b>object</b>	IPAM CNI 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.  <b>dhcp</b> 를 지정하지 마십시오. IPVLAN 인터페이스가 호스트 인터페이스와 MAC 주소를 공유하므로 DHCP를 사용하여 IPVLAN 구성은 지원되지 않습니다.

#### 19.2.3.3.1. *ipvlan* 구성 예

다음 예제는 이름이 *ipvlan-net*인 추가 네트워크를 구성합니다.

```
{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "I3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

#### 19.2.3.4. MACVLAN 추가 네트워크 구성



다음 오브젝트는 **macvlan CNI 플러그인**의 구성 매개변수를 설명합니다.

표 19.5. MACVLAN CNI 플러그인 JSON 구성 오브젝트

필드	유형	설명
<b>cniVersion</b>	<b>string</b>	CNI 사양 버전입니다. <b>0.3.1</b> 값이 필요합니다.
<b>name</b>	<b>string</b>	CNO 구성에 대해 이전에 입력한 <b>name</b> 매개변수의 값입니다.
<b>type</b>	<b>string</b>	구성할 CNI 플러그인의 이름입니다. <b>macvlan</b> .
<b>mode</b>	<b>string</b>	가상 네트워크에서 트래픽 가시성을 구성합니다. <b>bridge</b> , <b>passthru</b> , <b>private</b> 또는 <b>vepa</b> 중 하나여야 합니다. 값을 입력하지 않으면 기본값은 <b>bridge</b> 입니다.
<b>master</b>	<b>string</b>	새로 생성된 macvlan 인터페이스와 연결할 호스트 네트워크 인터페이스입니다. 값을 지정하지 않으면 기본 경로 인터페이스가 사용됩니다.
<b>mtu</b>	<b>string</b>	지정된 값에 대한 최대 전송 단위(MTU)입니다. 기본값은 커널에 의해 자동으로 설정됩니다.
<b>ipam</b>	<b>object</b>	IPAM CNI 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.



#### 참고

플러그인 구성에 대한 마스터 키를 지정하는 경우, 가능한 충돌을 방지하려면 기본 네트워크 플러그인과 연결된 것과 다른 물리적 네트워크 인터페이스를 사용합니다.

#### 19.2.3.4.1. macvlan 구성 예

다음 예제는 이름이 **macvlan-net**인 추가 네트워크를 구성합니다.

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
  }
}
```

### 19.2.4. 추가 네트워크에 대한 IP 주소 할당 구성

IP 주소 관리(IPAM) CNI(Container Network Interface) 플러그인은 다른 CNI 플러그인에 대한 IP 주소를 제공합니다.

다음 IP 주소 할당 유형을 사용할 수 있습니다.

- 정적 할당
- DHCP 서버를 통한 동적 할당. 지정한 DHCP 서버는 추가 네트워크에서 연결할 수 있어야 합니다.
- Whereabouts IPAM CNI 플러그인을 통한 동적 할당

#### 19.2.4.1. 고정 IP 주소 할당 구성

다음 표에서는 고정 IP 주소 할당 구성에 대해 설명합니다.

표 19.6. IPAM 고정 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. <b>static</b> 값이 필요합니다.
주소	array	가상 인터페이스에 할당할 IP 주소를 지정하는 개체의 배열입니다. IPv4 및 IPv6 IP 주소가 모두 지원됩니다.
routes	array	Pod 내부에서 구성할 경로를 지정하는 오브젝트의 배열입니다.
dns	array	선택 사항: DNS 구성을 지정하는 오브젝트의 배열입니다.

addresses 배열에는 다음 필드가 있는 오브젝트가 필요합니다.

표 19.7. ipam.addresses[] array

필드	유형	설명
----	----	----

필드	유형	설명
주소	string	지정하는 IP 주소 및 네트워크 접두사입니다. 예를 들어 <b>10.10.21.10/24</b> 를 지정하면 추가 네트워크에 IP 주소 <b>10.10.21.10</b> 이 할당되고 넷마스크는 <b>255.255.255.0</b> 입니다.
gateway	string	송신 네트워크 트래픽을 라우팅할 기본 게이트웨이입니다.

표 19.8. *ipam.routes[]* array

필드	유형	설명
dst	string	CIDR 형식의 IP 주소 범위(예: 기본 경로의 경우 <b>192.168.17.0/24</b> 또는 <b>0.0.0.0/0</b> )입니다.
gw	string	네트워크 트래픽이 라우팅되는 게이트웨이입니다.

표 19.9. *ipam.dns* object

필드	유형	설명
nameservers	array	DNS 쿼리를 보낼 하나 이상의 IP 주소로 구성된 배열입니다.
domain	array	호스트 이름에 추가할 기본 도메인입니다. 예를 들어 도메인이 <b>example.com</b> 으로 설정되면 <b>example-host</b> 에 대한 DNS 조회 쿼리가 <b>example-host.example.com</b> 으로 다시 작성됩니다.
search	array	DNS 조회 쿼리 중에 규정되지 않은 호스트 이름(예: <b>example-host</b> )에 추가할 도메인 이름 배열입니다.

## 고정 IP 주소 할당 구성 예

```

{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}

```

### 19.2.4.2. DHCP(Dynamic IP 주소) 할당 구성

다음 JSON은 DHCP를 사용한 동적 IP 주소 할당 구성을 설명합니다.

#### DHCP 리스 갱신

pod는 생성될 때 원래 DHCP 리스를 얻습니다. 리스는 클러스터에서 실행되는 최소 DHCP 서버 배포를 통해 주기적으로 갱신되어야 합니다.

DHCP 서버 배포를 트리거하려면 다음 예와 같이 Cluster Network Operator 구성을 편집하여 shim 네트워크 연결을 만들어야 합니다.

#### shim 네트워크 연결 정의 예

```
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"
        }
      }
  # ...
```

표 19.10. IPAM DHCP 구성 오브젝트

필드	유형	설명
<b>type</b>	<b>string</b>	IPAM 주소 유형입니다. 값 <b>dhcp</b> 가 필요합니다.

### DHCP(Dynamic IP 주소) 할당 구성 예

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

### 19.2.4.3. Whereabouts를 사용한 동적 IP 주소 할당 구성

**Whereabouts CNI** 플러그인을 사용하면 **DHCP** 서버를 사용하지 않고도 **IP** 주소를 추가 네트워크에 동적으로 할당할 수 있습니다.

다음 표에서는 **Whereabouts**를 사용한 동적 **IP** 주소 할당 구성을 설명합니다.

표 19.11. IPAM 위치 구성 오브젝트

필드	유형	설명
<b>type</b>	<b>string</b>	IPAM 주소 유형입니다. 값 <b>whereabouts</b> 가 필요합니다.
<b>범위</b>	<b>string</b>	CIDR 표기법의 IP 주소 및 범위입니다. IP 주소는 이 주소 범위 내에서 할당됩니다.
<b>exclude</b>	<b>array</b>	선택 사항: CIDR 표기법으로 0개 이상의 IP 주소 및 범위 목록입니다. 제외된 주소 범위 내의 IP 주소는 할당되지 않습니다.

### Whereabouts를 사용하는 동적 IP 주소 할당 구성 예

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
  }
}
```

```

"exclude": [
  "192.0.2.192/30",
  "192.0.2.196/32"
]
}
}

```

### 19.2.5. Cluster Network Operator를 사용하여 추가 네트워크 연결 생성

**CNO(Cluster Network Operator)**는 추가 네트워크 정의를 관리합니다. 생성할 추가 네트워크를 지정하면 **CNO**가 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성합니다.



중요

**Cluster Network Operator**가 관리하는 **NetworkAttachmentDefinition** 오브젝트를 편집하지 마십시오. 편집하면 추가 네트워크의 네트워크 트래픽이 중단될 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. 선택 사항: 추가 네트워크의 네임스페이스를 생성합니다.

```
$ oc create namespace <namespace_name>
```

2. **CNO** 구성을 편집하려면 다음 명령을 입력합니다.

```
$ oc edit networks.operator.openshift.io cluster
```

3. 다음 예제 **CR**과 같이 생성 중인 추가 네트워크의 구성을 추가하여 생성 중인 **CR**을 수정합니다.

```

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": "l2",
        "ipam": {
          "type": "static",
          "addresses": [
            {
              "address": "192.168.1.23/24"
            }
          ]
        }
      }

```

4.

변경 사항을 저장하고 텍스트 편집기를 종료하여 변경 사항을 커밋합니다.

## 검증

•

**CNO가 다음 명령을 실행하여 NetworkAttachmentDefinition 오브젝트를 생성했는지 확인**합니다. **CNO가 오브젝트를 생성하기 전에 지연이 발생할 수 있습니다.**

```
$ oc get network-attachment-definitions -n <namespace>
```

다음과 같습니다.

<namespace>

**CNO 구성에 추가한 네트워크 연결의 네임스페이스를 지정합니다.**

출력 예

<b>NAME</b>	<b>AGE</b>
<b>test-network-1</b>	<b>14m</b>

### 19.2.6. YAML 매니페스트를 적용하여 추가 네트워크 연결 생성

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. 다음 예와 같이 추가 네트워크 구성을 사용하여 **YAML** 파일을 생성합니다.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: next-net
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "name": "work-network",
      "type": "host-device",
      "device": "eth1",
      "ipam": {
        "type": "dhcp"
      }
    }
  }
```

2. 추가 네트워크를 생성하려면 다음 명령을 입력합니다.

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

다음과 같습니다.

**<file>**



YAML 매니페스트가 포함된 파일의 이름을 지정합니다.

### 19.3. 가상 라우팅 및 전달 정보

#### 19.3.1. 가상 라우팅 및 전달 정보

IP 규칙과 결합된 가상 라우팅 및 전달(VRF) 장치는 가상 라우팅 및 전달 도메인을 생성하는 기능을 제공합니다. VRF는 CNF에 필요한 권한 수를 줄이고 보조 네트워크의 네트워크 토폴로지의 가시성을 증대시킵니다. VRF는 예를 들어 각 테넌트마다 고유한 라우팅 테이블이 있고 다른 기본 게이트웨이가 필요한 멀티 테넌시 기능을 제공하는 데 사용됩니다.

프로세스는 소켓을 VRF 장치에 바인딩할 수 있습니다. 바인딩된 소켓을 통한 패킷은 VRF 장치와 연결된 라우팅 테이블을 사용합니다. VRF의 중요한 기능은 OSI 모델 레이어 3 트래픽 및 LLDP와 같은 L2 도구에만 영향을 미치지 않는다는 것입니다. 이를 통해 정책 기반 라우팅과 같은 우선순위가 높은 IP 규칙이 특정 트래픽을 지시하는 VRF 장치 규칙보다 우선합니다.

##### 19.3.1.1. 통신 운영자의 포드에 대한 보조 네트워크 이점

통신사용 사례에서 각 CNF는 동일한 주소 공간을 공유하는 여러 다른 네트워크에 잠재적으로 연결할 수 있습니다. 이러한 보조 네트워크는 클러스터의 기본 네트워크 CIDR과 잠재적으로 충돌할 수 있습니다. CNI VRF 플러그인을 사용하여 네트워크 기능은 동일한 IP 주소를 사용하여 다른 고객의 인프라에 연결할 수 있으므로 서로 다른 고객을 분리할 수 있습니다. IP 주소는 OpenShift Container Platform IP 공간과 겹치게 됩니다. CNI VRF 플러그인은 CNF에 필요한 권한 수를 줄이고 보조 네트워크의 네트워크 토폴로지의 가시성을 높입니다.

### 19.4. 다중 네트워크 정책 구성

클러스터 관리자는 추가 네트워크에 대한 네트워크 정책을 구성할 수 있습니다.



#### 참고

macvlan 추가 네트워크에 대해서만 다중 네트워크 정책을 지정할 수 있습니다. ipvlan과 같은 기타 유형의 추가 네트워크는 지원되지 않습니다.

#### 19.4.1. 다중 네트워크 정책과 네트워크 정책의 차이점

MultiNetworkPolicy API는 NetworkPolicy API를 구현하지만 다음과 같은 몇 가지 중요한 차이점이 있습니다.

- **MultiNetworkPolicy API를 사용해야 합니다.**

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
```

- **CLI를 사용하여 다중 네트워크 정책과 상호 작용할 때 multi-networkpolicy 리소스 이름을 사용해야 합니다. 예를 들어 oc get multi-networkpolicy <name> 명령을 사용하여 다중 네트워크 정책 오브젝트를 볼 수 있습니다. 여기서 <name>은 다중 네트워크 정책의 이름입니다.**

- **macvlan 추가 네트워크를 정의하는 네트워크 연결 정의의 이름으로 주석을 지정해야 합니다.**

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
```

다음과 같습니다.

**<network\_name>**

네트워크 연결 정의의 이름을 지정합니다.

### 19.4.2. 클러스터의 다중 네트워크 정책 활성화

클러스터 관리자는 클러스터에서 다중 네트워크 정책 지원을 활성화할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)를 설치합니다.**
- **cluster-admin 권한이 있는 사용자로 클러스터에 로그인합니다.**

#### 프로세스

1. **다음 YAML을 사용하여 multinetwork-enable-patch.yaml 파일을 생성합니다.**

```

apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  useMultiNetworkPolicy: true

```

2.

다중 네트워크 정책을 활성화하도록 클러스터를 구성합니다.

```

$ oc patch network.operator.openshift.io cluster --type=merge --patch-
file=multinetwork-enable-patch.yaml

```

출력 예

```

network.operator.openshift.io/cluster patched

```

### 19.4.3. 다중 네트워크 정책 작업

클러스터 관리자는 다중 네트워크 정책을 생성, 편집, 보기 및 삭제할 수 있습니다.

#### 19.4.3.1. 사전 요구 사항

- 클러스터에 대한 다중 네트워크 정책 지원을 활성화했습니다.

#### 19.4.3.2. CLI를 사용하여 다중 네트워크 정책 생성

클러스터의 네임스페이스에서 허용된 수신 또는 송신 네트워크 트래픽을 설명하는 세분화된 규칙을 정의하기 위해 다중 네트워크 정책을 생성할 수 있습니다.

사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자(예: **mode: NetworkPolicy** 로 설정된 **OpenShift SDN** 네트워크 공급자)를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 다중 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

#### 프로세스

1. 다음과 같이 정책 규칙을 생성합니다.
  - a. **<policy\_name>.yaml** 파일을 생성합니다.

```
$ touch <policy_name>.yaml
```

다음과 같습니다.

**<policy\_name>**

다중 네트워크 정책 파일 이름을 지정합니다.

- b. 방금 만든 파일에서 다음 예와 같이 다중 네트워크 정책을 정의합니다.

모든 네임스페이스의 모든 **Pod**에서 수신 거부

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    ingress: []
```

다음과 같습니다.

**<network\_name>**

네트워크 연결 정의의 이름을 지정합니다.

동일한 네임 스페이스에 있는 모든 Pod의 수신 허용

```

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: {}

```

다음과 같습니다.

**<network\_name>**

네트워크 연결 정의의 이름을 지정합니다.

2.

다음 명령을 실행하여 다중 네트워크 정책 오브젝트를 생성합니다.

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

다중 네트워크 정책 파일 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

출력 예

`multinetworkpolicy.k8s.cni.cncf.io/default-deny created`



참고

`cluster-admin` 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 `YAML` 또는 웹 콘솔의 양식에서 직접 네트워크 정책을 생성할 수 있습니다.

### 19.4.3.3. 다중 네트워크 정책 편집

네임스페이스에서 다중 네트워크 정책을 편집할 수 있습니다.

사전 요구 사항

- 클러스터는 `NetworkPolicy` 오브젝트를 지원하는 클러스터 네트워크 공급자(예: `mode: NetworkPolicy` 로 설정된 `OpenShift SDN` 네트워크 공급자)를 사용합니다. 이 모드는 `OpenShift SDN`의 기본값입니다.
- `OpenShift CLI(oc)`를 설치합니다.
- `cluster-admin` 권한이 있는 사용자로 클러스터에 로그인합니다.
- 다중 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

프로세스

1. 선택 사항: 네임스페이스의 다중 네트워크 정책 오브젝트를 나열하려면 다음 명령을 입력합니다.

```
$ oc get multi-networkpolicy
```

다음과 같습니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

2.

다중 네트워크 정책 오브젝트를 편집합니다.

- 

다중 네트워크 정책 정의를 파일에 저장한 경우 파일을 편집하고 필요한 사항을 변경한 후 다음 명령을 입력합니다.

```
$ oc apply -n <namespace> -f <policy_file>.yaml
```

다음과 같습니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

**<policy\_file>**

네트워크 정책이 포함된 파일의 이름을 지정합니다.

- 

다중 네트워크 정책 오브젝트를 직접 업데이트해야 하는 경우 다음 명령을 입력합니다.

```
$ oc edit multi-networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

네트워크 정책의 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

3.

다중 네트워크 정책 오브젝트가 업데이트되었는지 확인합니다.

```
$ oc describe multi-networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

<policy\_name>

다중 네트워크 정책의 이름을 지정합니다.

<namespace>

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.



참고

**cluster-admin** 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 **YAML**에서 직접 또는 작업 메뉴를 통해 웹 콘솔의 정책에서 네트워크 정책을 편집할 수 있습니다.

### 19.4.3.4. CLI를 사용하여 다중 네트워크 정책 보기

네임스페이스에서 다중 네트워크 정책을 검사할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 다중 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

#### 프로세스



- 네임스페이스의 다중 네트워크 정책을 나열합니다.
- 네임스페이스에 정의된 다중 네트워크 정책 오브젝트를 보려면 다음 명령을 입력합니다.

```
$ oc get multi-networkpolicy
```

- 선택 사항: 특정 다중 네트워크 정책을 검사하려면 다음 명령을 입력합니다.

```
$ oc describe multi-networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

검사할 다중 네트워크 정책의 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.



참고

**cluster-admin** 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 YAML 또는 웹 콘솔의 양식에서 직접 네트워크 정책을 볼 수 있습니다.

#### 19.4.3.5. CLI를 사용하여 다중 네트워크 정책 삭제

네임스페이스에서 다중 네트워크 정책을 삭제할 수 있습니다.

사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자(예: **mode: NetworkPolicy** 로 설정된 **OpenShift SDN** 네트워크 공급자)를 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 다중 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

프로세스

- 다중 네트워크 정책 오브젝트를 삭제하려면 다음 명령을 입력합니다.

```
$ oc delete multi-networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

**<policy\_name>**

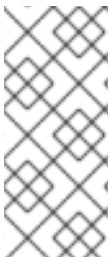
다중 네트워크 정책의 이름을 지정합니다.

**<namespace>**

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 이를 사용하여 네임스페이스를 지정합니다.

출력 예

```
multinetworkpolicy.k8s.cni.cncf.io/default-deny deleted
```



참고

**cluster-admin** 권한을 사용하여 웹 콘솔에 로그인하면 클러스터의 모든 네임스페이스에서 **YAML**에서 직접 또는 작업 메뉴를 통해 웹 콘솔의 정책에서 네트워크 정책을 삭제할 수 있습니다.

#### 19.4.4. 추가 리소스

- [네트워크 정책 정의](#)
- [다중 네트워크 이해하기](#)
- [macvlan 네트워크 구성](#)

#### 19.5. 추가 네트워크에 POD 연결

클러스터 사용자는 pod를 추가 네트워크에 연결할 수 있습니다.

##### 19.5.1. 추가 네트워크에 Pod 추가

추가 네트워크에 Pod를 추가할 수 있습니다. Pod는 기본 네트워크를 통해 정상적인 클러스터 관련 네트워크 트래픽을 계속 전송합니다.

Pod가 생성되면 추가 네트워크가 연결됩니다. 그러나 Pod가 이미 있는 경우에는 추가 네트워크를 연결할 수 없습니다.

Pod는 추가 네트워크와 동일한 네임스페이스에 있어야 합니다.

#### 사전 요구 사항

- [OpenShift CLI\(oc\)](#)를 설치합니다.
- 클러스터에 로그인합니다.

#### 프로세스

1. Pod 오브젝트에 주석을 추가합니다. 다음 주석 형식 중 하나만 사용할 수 있습니다.
  - a. 사용자 정의 없이 추가 네트워크를 연결하려면 다음 형식으로 주석을 추가합니다. `<network>`를 Pod와 연결할 추가 네트워크의 이름으로 변경합니다.

```

metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
    
```

1

둘 이상의 추가 네트워크를 지정하려면 각 네트워크를 쉼표로 구분합니다. 쉼표 사이에 공백을 포함하지 마십시오. 동일한 추가 네트워크를 여러 번 지정하면 Pod에 해당 네트워크에 대한 인터페이스가 여러 개 연결됩니다.

b.

사용자 정의된 추가 네트워크를 연결하려면 다음 형식으로 주석을 추가합니다.

```

metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>", 1
          "namespace": "<namespace>", 2
          "default-route": ["<default-route>"] 3
        }
      ]
    
```

1

**NetworkAttachmentDefinition** 오브젝트에서 정의한 추가 네트워크의 이름을 지정합니다.

2

**NetworkAttachmentDefinition** 오브젝트가 정의된 네임스페이스를 지정합니다.

3

선택 사항: 기본 경로에 대한 재정의를 지정합니다(예: 192.168.17.1).

2.

Pod를 생성하려면 다음 명령을 입력합니다. <name>을 Pod 이름으로 교체합니다.

```
$ oc create -f <name>.yaml
```

3.

선택사항: Pod CR에 주석이 있는지 확인하려면 다음 명령을 입력하고 <name>을 Pod 이름으로 교체합니다.

```
$ oc get pod <name> -o yaml
```

다음 예에서 `example-pod Pod`는 `net1` 추가 네트워크에 연결되어 있습니다.

```
$ 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/networks-status: |- 1
    [{
      "name": "openshift-sdn",
      "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

`k8s.v1.cni.cncf.io/networks-status` 매개 변수는 **JSON** 오브젝트 배열입니다. 각 오브젝트는 `Pod`에 연결된 추가 네트워크의 상태를 설명합니다. 주석 값은 일반 텍스트 값으로 저장됩니다.

### 19.5.1.1. Pod별 주소 지정 및 라우팅 옵션 지정

추가 네트워크에 `Pod`를 연결할 때 특정 `Pod`에서 해당 네트워크에 대한 추가 속성을 지정할 수 있습니다. 이를 통해 라우팅의 일부 측면을 변경하고 고정 **IP** 주소 및 **MAC** 주소를 지정할 수 있습니다. 이를 위해 **JSON** 형식의 주석을 사용할 수 있습니다.

## 사전 요구 사항

- **Pod**는 추가 네트워크와 동일한 네임스페이스에 있어야 합니다.
- **OpenShift CLI(oc)**를 설치합니다.
- 클러스터에 로그인해야 합니다.

## 프로세스

주소 지정 및/또는 라우팅 옵션을 지정하는 동안 추가 네트워크에 **Pod**를 추가하려면 다음 단계를 완료하십시오.

1. **Pod** 리소스 정의를 편집합니다. 기존 **Pod** 리소스를 편집하는 경우 다음 명령을 실행하여 기본 편집기에서 정의를 편집합니다. <name>을 편집할 **Pod** 리소스의 이름으로 교체합니다.

```
$ oc edit pod <name>
```

2. **Pod** 리소스 정의에서 **k8s.v1.cni.cncf.io/networks** 매개변수를 **Pod metadata** 매핑에 추가합니다. **k8s.v1.cni.cncf.io/networks**는 추가 특성을 지정하는 것 외에도 **NetworkAttachmentDefinition Custom Resource(CR)** 이름을 참조하는 오브젝트 목록의 **JSON** 문자열을 허용합니다.

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: ' [<network>,<network>,...]' 1
```

1

다음 예제와 같이 <network>를 **JSON** 오브젝트로 변경합니다. 작은 따옴표를 사용해야 합니다.

3. 다음 예에서 주석은 **default-route** 매개변수를 사용하여 기본 경로로 지정될 네트워크 연결을 지정합니다.

```
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

`name` 키는 Pod와 연결할 추가 네트워크의 이름입니다.

②

`default-route` 키는 라우팅 테이블에 다른 라우팅 항목이 없는 경우 트래픽이 라우팅 될 게이트웨이 값을 지정합니다. `default-route` 키가 두 개 이상 지정되면 Pod가 활성화되지 않습니다.

기본 경로는 다른 경로에 지정되지 않은 모든 트래픽이 게이트웨이로 라우팅되도록 합니다.

### 중요

OpenShift Container Platform의 기본 네트워크 인터페이스 이외의 인터페이스로 기본 경로를 설정하면 Pod 사이에서 트래픽이 라우팅될 것으로 예상되는 트래픽이 다른 인터페이스를 통해 라우팅될 수 있습니다.

Pod의 라우팅 속성을 확인하려면 `oc` 명령을 사용하여 Pod에서 `ip` 명령을 실행하십시오.

```
$ oc exec -it <pod_name> -- ip route
```

### 참고

JSON 형식의 오브젝트 목록에 `default-route` 키가 있으면 Pod의 `k8s.v1.cni.cncf.io/networks-status`를 참조하여 어떤 추가 네트워크에 기본 경로가 할당되었는지를 확인할 수도 있습니다.

**Pod**의 고정 IP 주소 또는 **MAC** 주소를 설정하려면 **JSON** 형식의 주석을 사용하면 됩니다. 이를 위해서는 이러한 기능을 특별하게 허용하는 네트워크를 생성해야 합니다. 이는 다음과 같이 **CNO**의 **rawCNICongig**에서 지정할 수 있습니다.

1.

다음 명령을 실행하여 **CNO CR**을 편집합니다.

```
$ oc edit networks.operator.openshift.io cluster
```

다음 **YAML**은 **CNO**의 구성 매개변수를 설명합니다.

### CNO(Cluster Network Operator) YAML 구성

```
name: <name> 1
namespace: <namespace> 2
rawCNICongig: '{ 3
  ...
}'
type: Raw
```

1

생성 중인 추가 네트워크 연결의 이름을 지정합니다. 이름은 지정된 **namespace** 내에서 고유해야 합니다.

2

네트워크를 연결한 네임스페이스를 지정합니다. 값을 지정하지 않으면 **default** 네임스페이스가 사용됩니다.

3

다음 템플릿을 기반으로 **CNI** 플러그인 구성을 **JSON** 형식으로 지정합니다.

다음 오브젝트는 **macvlan CNI** 플러그인을 사용하여 고정 **MAC** 주소 및 **IP** 주소를 사용하기 위한 구성 매개변수를 설명합니다.

### 고정 IP 및 MAC 주소를 사용하는 macvlan CNI 플러그인 JSON 구성 오브젝트



```

{
  "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

생성할 추가 네트워크 연결의 이름을 지정합니다. 이름은 지정된 **namespace** 내에서 고유해야 합니다.

2

**CNI** 플러그인 구성의 배열을 지정합니다. 첫 번째 오브젝트는 **macvlan** 플러그인 구성을 지정하고 두 번째 오브젝트는 튜닝 플러그인 구성을 지정합니다.

3

**CNI** 플러그인 런타임 구성 기능의 고정 **IP** 주소 기능을 사용하도록 요청하도록 지정합니다.

4

**macvlan** 플러그인이 사용하는 인터페이스를 지정합니다.

5

**CNI** 플러그인의 정적 **MAC** 주소 기능을 사용하도록 요청하도록 지정합니다.

그런 다음 위의 네트워크 연결을 키와 함께 **JSON** 형식 주석에서 참조하여 지정된 **Pod**에 할당할 고정 **IP** 및 **MAC** 주소를 지정할 수 있습니다.

다음을 사용하여 **Pod**를 편집합니다.

```
$ oc edit pod <name>
```

고정 IP 및 MAC 주소를 사용하는 **macvlan CNI 플러그인 JSON 구성 오브젝트**

```
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

위의 **rawCNICConfig**를 구성하는 경우에는 제공되는 **<name>**을 사용해야 합니다.

2

서브넷 마스크를 포함하여 **IP** 주소를 제공합니다.

3

**MAC** 주소를 입력합니다.



참고

고정 IP 주소와 **MAC** 주소를 동시에 사용할 필요는 없으며 개별적으로 또는 함께 사용할 수 있습니다.

추가 네트워크가 있는 **Pod**의 **IP** 주소 및 **MAC** 속성을 확인하려면 **oc** 명령을 사용하여 **Pod**에서 **ip** 명령을 실행합니다.

```
$ oc exec -it <pod_name> -- ip a
```

## 19.6. 추가 네트워크에서 POD 제거

클러스터 사용자는 추가 네트워크에서 Pod를 제거할 수 있습니다.

### 19.6.1. 추가 네트워크에서 Pod 제거

Pod를 삭제해야만 추가 네트워크에서 Pod를 제거할 수 있습니다.

#### 사전 요구 사항

- Pod에 추가 네트워크가 연결되어 있어야 합니다.
- OpenShift CLI(oc)를 설치합니다.
- 클러스터에 로그인합니다.

#### 프로세스

- Pod를 삭제하려면 다음 명령을 입력합니다.

```
$ oc delete pod <name> -n <namespace>
```

- <name>은 Pod의 이름입니다.
- <namespace>는 Pod가 포함된 네임스페이스입니다.

## 19.7. 추가 네트워크 편집

클러스터 관리자는 기존 추가 네트워크의 구성을 수정할 수 있습니다.

### 19.7.1. 추가 네트워크 연결 정의 수정

클러스터 관리자는 기존 추가 네트워크를 변경할 수 있습니다. 추가 네트워크에 연결된 기존 **Pod**는 업데이트되지 않습니다.

#### 사전 요구 사항

- 클러스터에 추가 네트워크가 구성되어야 합니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

클러스터의 추가 네트워크를 편집하려면 다음 단계를 완료하십시오.

1. 기본 텍스트 편집기에서 **CNO(Cluster Network Operator) CR**을 편집하려면 다음 명령을 실행합니다.

```
$ oc edit networks.operator.openshift.io cluster
```

2. **additionalNetworks** 컬렉션에서 변경 내용으로 추가 네트워크를 업데이트합니다.

3. 변경 사항을 저장하고 텍스트 편집기를 종료하여 변경 사항을 커밋합니다.

4. 선택 사항: **CNO**에서 다음 명령을 실행하여 **NetworkAttachmentDefinition** 오브젝트를 업데이트했는지 확인합니다. **<network-name>**을 표시할 추가 네트워크의 이름으로 변경합니다. **CNO**가 변경 사항을 반영하기 위해서 **NetworkAttachmentDefinition** 오브젝트를 업데이트하기 전에 지연이 발생할 수 있습니다.

```
$ oc get network-attachment-definitions <network-name> -o yaml
```

예를 들어, 다음 콘솔 출력은 **net1**이라는 **NetworkAttachmentDefinition** 오브젝트를 표시합니다.

```
$ 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"]} } }
```

## 19.8. 추가 네트워크 제거

클러스터 관리자는 추가 네트워크의 연결을 제거할 수 있습니다.

### 19.8.1. 추가 네트워크 연결 정의 제거

클러스터 관리자는 **OpenShift Container Platform** 클러스터에서 추가 네트워크를 제거할 수 있습니다. 추가 네트워크는 연결된 Pod에서 제거되지 않습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

클러스터에서 추가 네트워크를 제거하려면 다음 단계를 완료하십시오.

1. 다음 명령을 실행하여 기본 텍스트 편집기에서 **CNO(Cluster Network Operator)**를 편집합니다.

```
$ oc edit networks.operator.openshift.io cluster
```

2. 제거할 네트워크 연결 정의에 대한 **additionalNetworks** 컬렉션에서 구성을 제거하여 **CR**을 수정합니다.

```
apiVersion: operator.openshift.io/v1
kind: Network
```

```

metadata:
  name: cluster
spec:
  additionalNetworks: [] 1
    
```

**1**

**additionalNetworks** 컬렉션에서 유일한 추가 네트워크 첨부 파일 정의에 대한 구성 매핑을 제거하는 경우 빈 컬렉션을 지정해야 합니다.

3. 변경 사항을 저장하고 텍스트 편집기를 종료하여 변경 사항을 커밋합니다.
4. 선택 사항: 추가 네트워크 **CR**이 삭제되었는지 확인하려면 다음 명령을 실행합니다.

```

$ oc get network-attachment-definition --all-namespaces
    
```

### 19.9. VRF에 보조 네트워크 할당

#### 19.9.1. VRF에 보조 네트워크 할당

클러스터 관리자는 **CNI VRF** 플러그인을 사용하여 **VRF** 도메인에 대한 추가 네트워크를 구성할 수 있습니다. 이 플러그인으로 생성된 가상 네트워크는 지정된 물리적 인터페이스와 연결됩니다.



#### 참고

**VRF**를 사용하는 애플리케이션은 특정 장치에 바인딩해야 합니다. 일반적인 사용은 소켓에 **SO\_BINDTODEVICE** 옵션을 사용하는 것입니다. **SO\_BINDTODEVICE**는 소켓을 전달된 인터페이스 이름(예: **eth1**)에 지정된 장치에 바인딩합니다. **SO\_BINDTODEVICE**를 사용하려면 애플리케이션에 **CAP\_NET\_RAW** 기능이 있어야 합니다.

**OpenShift Container Platform Pod**에서는 **ip vrf exec** 명령을 통해 **VRF**를 사용할 수 없습니다. **VRF**를 사용하려면 애플리케이션을 **VRF** 인터페이스에 직접 바인딩합니다.

#### 19.9.1.1. CNI VRF 플러그인으로 추가 네트워크 연결 생성

**CNO(Cluster Network Operator)**는 추가 네트워크 정의를 관리합니다. 생성할 추가 네트워크를 지정하면 **CNO**가 **NetworkAttachmentDefinition CR**(사용자 정의 리소스)을 자동으로 생성합니다.



## 참고

**CNO가 관리하는 NetworkAttachmentDefinition CR을 편집하지 마십시오.** 편집하면 추가 네트워크의 네트워크 트래픽이 중단될 수 있습니다.

**CNI VRF 플러그인**으로 추가 네트워크 연결을 생성하려면 다음 절차를 수행하십시오.

## 사전 요구 사항

- **OpenShift Container Platform CLI, oc**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 **OpenShift** 클러스터에 로그인합니다.

## 절차

1.

추가 **Network** 연결에 사용할 네트워크 **CR**(사용자 정의 리소스)을 생성하고 다음 예제 **CR** 과 같이 추가 네트워크의 **rawCNICongig** 구성을 삽입합니다. **YAML**을 **additional-network-attachment.yaml** 파일로 저장합니다.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: test-network-1
    namespace: additional-network-1
    type: Raw
    rawCNICongig: '{
      "cniVersion": "0.3.1",
      "name": "macvlan-vrf",
      "plugins": [ 1
      {
        "type": "macvlan", 2
        "master": "eth1",
        "ipam": {
          "type": "static",
          "addresses": [
            {
              "address": "191.168.1.23/24"
            }
          ]
        }
      }
    ]
  },
  {
```

```
"type": "vrf",
"vrfname": "example-vrf-name", 3
"table": 1001 4
  }
}
```

1

**plugins**는 목록이어야 합니다. 목록의 첫 번째 항목은 **VRF** 네트워크를 기반으로 하는 보조 네트워크여야 합니다. 목록의 두 번째 항목은 **VRF** 플러그인 구성입니다.

2

**type**은 **vrf**로 설정해야 합니다.

3

**vrfname**은 인터페이스가 할당된 **VRF**의 이름입니다. 포트에 없는 경우 생성됩니다.

4

**선택 사항:** **table**은 라우팅 테이블 ID입니다. 기본적으로 **tableid** 매개변수가 사용됩니다. 지정하지 않으면 **CNI**에서 무료 라우팅 테이블 ID를 **VRF**에 할당합니다.



참고

**VRF**는 리소스의 유형이 **netdevice**인 경우에만 올바르게 작동합니다.

2.

**Network** 리소스를 생성합니다.

```
$ oc create -f additional-network-attachment.yaml
```

3.

**CNO**가 다음 명령을 실행하여 **NetworkAttachmentDefinition CR**을 생성했는지 확인합니다. **<namespace>**를 네트워크 연결을 구성할 때 지정한 네임스페이스(예: **additional-network-1**)로 바꿉니다.

```
$ oc get network-attachment-definitions -n <namespace>
```

출력 예



NAME	AGE
additional-network-1	14m



참고

**CNO가 CR을 생성하기 전에 지연이 발생할 수 있습니다.**

추가 VRF 네트워크 연결에 성공했는지 확인

VRF CNI가 올바르게 구성되어 추가 네트워크 연결이 연결되었는지 확인하려면 다음을 수행하십시오.

1. **VRF CNI를 사용하는 네트워크를 생성합니다.**
2. **포드에 네트워크를 할당합니다.**
3. **포드 네트워크 연결이 VRF 추가 네트워크에 연결되어 있는지 확인합니다. Pod로 원격 셸을 설치하고 다음 명령을 실행합니다.**

```
$ ip vrf show
```

출력 예

Name	Table
red	10

4. **VRF 인터페이스가 보조 인터페이스의 마스터인지 확인합니다.**

```
$ ip link
```

## 출력 예

```
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master  
red state UP mode
```

## 20장. 하드웨어 네트워크

### 20.1. SR-IOV(SINGLE ROOT I/O VIRTUALIZATION) 하드웨어 네트워크 정보

**SR-IOV(Single Root I/O Virtualization)** 사양은 단일 장치를 여러 Pod와 공유할 수 있는 PCI 장치 할당 유형의 표준입니다.

**SR-IOV**를 사용하면 호스트 노드에서 물리적 기능(PF)으로 인식되는 호환 네트워크 장치를 여러 VF(가상 기능)로 분할할 수 있습니다. VF는 다른 네트워크 장치와 같이 사용됩니다. 장치의 **SR-IOV** 네트워크 장치 드라이버는 컨테이너에서 VF가 노출되는 방식을 결정합니다.

- **netdevice** 드라이버: 컨테이너의 netns에 있는 일반 커널 네트워크 장치
- **vfio-pci** 드라이버: 컨테이너에 마운트된 문자 장치

높은 대역폭 또는 짧은 대기 시간이 필요한 애플리케이션에 베어 메탈 또는 **RHOSP(Red Hat OpenStack Platform)** 인프라에 설치된 **OpenShift Container Platform** 클러스터에 추가 네트워크와 함께 **SR-IOV** 네트워크 장치를 사용할 수 있습니다.

다음 명령을 사용하여 노드에서 **SR-IOV**를 활성화할 수 있습니다.

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```

#### 20.1.1. SR-IOV 네트워크 장치를 관리하는 구성 요소

**SR-IOV** 네트워크 Operator는 **SR-IOV** 스택의 구성 요소를 생성하고 관리합니다. 다음과 같은 기능을 수행합니다.

- **SR-IOV** 네트워크 장치 검색 및 관리 오케스트레이션
- **SR-IOV** 컨테이너 네트워크 인터페이스(CNI)에 대한 NetworkAttachmentDefinition 사용자 정의 리소스 생성
- **SR-IOV** 네트워크 장치 플러그인의 구성을 생성하고 업데이트

- 노드별 **SriovNetworkNodeState** 사용자 정의 리소스 생성
- 각 **SriovNetworkNodeState** 사용자 정의 리소스에서 **spec.interfaces** 필드 업데이트

**Operator**는 다음 구성 요소를 프로비저닝합니다.

#### SR-IOV 네트워크 구성 데몬

**SR-IOV** 네트워크 **Operator**가 시작될 때 작업자 노드에 배포되는 데몬 세트입니다. 데몬은 클러스터에서 **SR-IOV** 네트워크 장치를 검색하고 초기화합니다.

#### SR-IOV 네트워크 Operator webhook

**Operator** 사용자 정의 리소스의 유효성을 검증하고 설정되지 않은 필드에 적절한 기본값을 설정하는 동적 승인 컨트롤러 **webhook**.

#### SR-IOV 네트워크 리소스 인젝터

**SR-IOV VF**와 같은 사용자 정의 네트워크 리소스에 대한 요청 및 제한으로 **Kubernetes pod** 사양을 패치하는 기능을 제공하는 동적 승인 컨트롤러 **webhook**. **SR-IOV** 네트워크 리소스 인젝터는 **Pod**의 첫 번째 컨테이너에만 리소스 필드를 자동으로 추가합니다.

#### SR-IOV 네트워크 장치 플러그인

**SR-IOV** 네트워크 **VF**(가상 기능) 리소스를 검색, 승격 및 할당하는 장치 플러그인입니다. **Kubernetes**에서는 장치 플러그인을 사용하여 일반적으로 물리적 장치에서 제한된 리소스를 사용할 수 있습니다. 장치 플러그인은 **Kubernetes** 스케줄러에 리소스 가용성을 인식하여 스케줄러가 충분한 리소스가 있는 노드에서 **Pod**를 예약할 수 있도록 합니다.

#### SR-IOV CNI 플러그인

**SR-IOV** 네트워크 장치 플러그인에서 할당된 **VF** 인터페이스를 **pod**에 직접 연결하는 **CNI** 플러그인입니다.

#### SR-IOV InfiniBand CNI 플러그인

**SR-IOV** 네트워크 장치 플러그인에서 할당된 **IB(InfiniBand) VF** 인터페이스를 **pod**에 직접 연결하는 **CNI** 플러그인입니다.



## 참고

**SR-IOV 네트워크 리소스 인젝터 및 SR-IOV 네트워크 Operator webhook**는 기본적으로 활성화되어 있으며 기본 **SriovOperatorConfig CR**을 편집하여 비활성화할 수 있습니다. **SR-IOV Network Operator Admission Controller** 웹 후크를 비활성화할 때 주의하십시오. 문제 해결과 같은 특정 상황에서 웹 후크를 비활성화하거나 지원되지 않는 장치를 사용하려는 경우 사용할 수 있습니다.

## 20.1.1.1. 지원되는 플랫폼

**SR-IOV Network Operator**는 다음 플랫폼에서 지원됩니다.

- **베어 메탈**
- **Red Hat OpenStack Platform (RHOSP)**

## 20.1.1.2. 지원되는 장치

**OpenShift Container Platform**에서는 다음 네트워크 인터페이스 컨트롤러를 지원합니다.

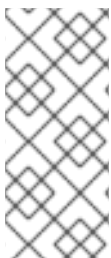
표 20.1. 지원되는 네트워크 인터페이스 컨트롤러

제조업체	모델	벤더 ID	장치 ID
Broadcom	BCM57414	14e4	16d7
Broadcom	BCM57508	14e4	1750
Intel	X710	8086	1572
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

제조업체	모델	벤더 ID	장치 ID
Mellanox	MT27700 제품군 [ConnectX-4]	15b3	1013
Mellanox	MT27710 제품군 [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 제품군 [ConnectX-5]	15b3	1017
Mellanox	MT28880 제품군 [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 제품군 [ConnectX-6]	15b3	101b
Mellanox	MT2892 제품군 [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 제품군 [ConnectX-6 Lx]	15b3	101f
Pensando <sup>[1]</sup>	DSC-25 듀얼 포트 25G 분산 서비스 카드 ionic 드라이버	0x1dd8	0x1002
Pensando <sup>[1]</sup>	ionic 드라이버용 DSC-100 듀얼 포트 100G 분산 서비스 카드	0x1dd8	0x1003

1.

**OpenShift SR-IOV**는 지원되지만 **SR-IOV**를 사용할 때 **SR-IOV**를 사용할 때 **SR-IOV CNI** 구성 파일을 사용하여 정적인 **VF**(가상 기능) 미디어 액세스 제어(**MAC**) 주소를 설정해야 합니다.



참고

지원되는 카드 및 호환 가능한 **OpenShift Container Platform** 버전의 최신 목록은 [OpenShift Single Root I/O Virtualization\(SR-IOV\)](#) 및 [PTP 하드웨어 네트워크 지원 매트릭스](#) 를 참조하십시오.

**20.1.1.3. SR-IOV 네트워크 장치의 자동 검색**

**SR-IOV Network Operator**는 작업자 노드에서 **SR-IOV** 가능 네트워크 장치를 클러스터에서 검색합니다. **Operator**는 호환되는 **SR-IOV** 네트워크 장치를 제공하는 각 작업자 노드에 대해 **SriovNetworkNodeState CR**(사용자 정의 리소스)을 생성하고 업데이트합니다.

**CR**에는 작업자 노드와 동일한 이름이 할당됩니다. **status.interfaces** 목록은 노드의 네트워크 장치에 대한 정보를 제공합니다.



중요

**SriovNetworkNodeState** 오브젝트를 수정하지 마십시오. **Operator**는 이러한 리소스를 자동으로 생성하고 관리합니다.

### 20.1.1.3.1. SriovNetworkNodeState 오브젝트의 예

다음 YAML은 SR-IOV Network Operator가 생성한 **SriovNetworkNodeState** 오브젝트의 예입니다.

#### **SriovNetworkNodeState** 오브젝트

```

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

`name` 필드의 값은 작업자 노드의 이름과 동일합니다.

2

인터페이스 스탠자에는 작업자 노드에서 **Operator**가 감지한 모든 **SR-IOV** 장치 목록이 포함되어 있습니다.

#### 20.1.1.4. Pod에서 가상 함수 사용 예

**SR-IOV VF**가 연결된 **pod**에서 **RDMA**(Remote Direct Memory Access) 또는 **DPDK**(Data Plane Development Kit) 애플리케이션을 실행할 수 있습니다.

이 예는 **RDMA** 모드에서 **VF**(가상 기능)를 사용하는 **pod**를 보여줍니다.

#### **RDMA** 모드를 사용하는 Pod 사양

```

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"]

```

다음 예는 DPDK 모드에서 VF가 있는 pod를 보여줍니다.

#### DPDK 모드를 사용하는 Pod 사양

```

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

```

### 20.1.1.5. 컨테이너 애플리케이션에서 사용하는 DPDK 라이브러리

**선택적 라이브러리**인 **app-netutil**은 해당 포드에서 실행 중인 컨테이너 내에서 포드에 관한 네트워크 정보를 수집하기 위해 여러 **API** 메서드를 제공합니다.

이 라이브러리는 **DPDK(Data Plane Development Kit)** 모드의 **SR-IOV VF**(가상 기능)를 컨테이너에 통합하는 데 도움이 될 수 있습니다. 라이브러리는 **Golang API**와 **C API**를 모두 제공합니다.

현재 세 가지 **API** 메서드가 구현되어 있습니다.

#### GetCPUInfo()

이 함수는 컨테이너에서 사용할 수 있는 **CPU**를 결정하고 목록을 반환합니다.

#### GetHugepages()

이 함수는 각 컨테이너에 대해 **Pod** 사양에서 요청된 대량의 페이지 메모리의 양을 결정하고 값을 반환합니다.

#### GetInterfaces()

이 함수는 컨테이너의 인터페이스 집합을 결정하고 목록을 반환합니다. 반환 값에는 각 인터페이스에 대한 인터페이스 유형 및 유형별 데이터가 포함됩니다.

라이브러리 리포지토리에는 컨테이너 이미지 **dpgk-app-centos**를 빌드하는 샘플 **Dockerfile**이 포함되어 있습니다. 컨테이너 이미지는 **pod** 사양의 환경 변수에 따라 다음 **DPDK** 샘플 애플리케이션 중 하나를 실행할 수 있습니다. **l2fwd**, **l3wd** 또는 **testpmd**. 컨테이너 이미지는 **app-netutil** 라이브러리를 컨테이너 이미지 자체에 통합하는 예를 제공합니다. 라이브러리는 **init** 컨테이너에 통합할 수도 있습니다. **init** 컨테이너는 필요한 데이터를 수집하고 기존 **DPDK** 워크로드에 데이터를 전달할 수 있습니다.

### 20.1.1.6. Downward API 에 대한 대규모 페이지 리소스 주입

**Pod** 사양에 대규모 페이지에 대한 리소스 요청 또는 제한이 포함된 경우 **Network Resources Injector**는 컨테이너에 대규모 페이지 정보를 제공하기 위해 **Pod** 사양에 **Downward API** 필드를 자동으로 추가합니다.

**Network Resources Injector**는 **podnetinfo**라는 볼륨을 추가하고 **Pod**의 각 컨테이너에 대해 **/etc/podnetinfo**에 마운트됩니다. 볼륨은 **Downward API**를 사용하며 대규모 페이지 요청 및 제한에 대한 파일을 포함합니다. 파일 이름 지정 규칙은 다음과 같습니다.

- `/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>`

이전 목록에 지정된 경로는 `app-netutil` 라이브러리와 호환됩니다. 기본적으로 라이브러리는 `/etc/podnetinfo` 디렉터리에서 리소스 정보를 검색하도록 구성됩니다. **Downward API** 경로 항목을 수동으로 지정하도록 선택하는 경우 `app-netutil` 라이브러리는 이전 목록의 경로 외에도 다음 경로를 검색합니다.

- `/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`

`Network Resources Injector`에서 생성할 수 있는 경로와 마찬가지로, 이전 목록의 경로는 선택적으로 `_<container-name>` 접미사로 종료할 수 있습니다.

### 20.1.2. 다음 단계

- [SR-IOV Network Operator 설치](#)

- 선택사항: [SR-IOV Network Operator](#) 구성
- [SR-IOV 네트워크 장치 구성](#)
- OpenShift Virtualization을 사용하는 경우: [가상 머신을 SR-IOV 네트워크에 연결](#)
- [SR-IOV 네트워크 연결 구성](#)
- [SR-IOV 추가 네트워크에 pod 추가](#)

## 20.2. SR-IOV NETWORK OPERATOR 설치

**SR-IOV(Single Root I/O Virtualization) Network Operator**를 클러스터에 설치하여 **SR-IOV 네트워크 장치 및 네트워크 연결**을 관리할 수 있습니다.

### 20.2.1. SR-IOV Network Operator 설치

클러스터 관리자는 **OpenShift Container Platform CLI** 또는 웹 콘솔을 사용하여 **SR-IOV Network Operator**를 설치할 수 있습니다.

#### 20.2.1.1. CLI: SR-IOV Network Operator 설치

클러스터 관리자는 **CLI**를 사용하여 **Operator**를 설치할 수 있습니다.

#### 사전 요구 사항

- **SR-IOV**를 지원하는 하드웨어가 있는 노드로 베어 메탈 하드웨어에 설치된 클러스터.
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 계정.

## 프로세스

1.

**openshift-sriov-network-operator** 네임스페이스를 생성하려면 다음 명령을 입력합니다.

```
$ cat << EOF | oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
EOF
```

2.

**OperatorGroup CR**을 생성하려면 다음 명령을 입력합니다.

```
$ 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.

**SR-IOV Network Operator**를 서브스크립션합니다.

a.

다음 명령을 실행하여 **OpenShift Container Platform** 주 버전 및 부 버전을 가져옵니다. 다음 단계의 **channel** 값에 필요합니다.

```
$ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
grep -o '[0-9]*[.][0-9]*' | head -1)
```

b.

**SR-IOV Network Operator**에 대한 서브스크립션 **CR**을 만들려면 다음 명령을 입력합니다.

```
$ 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: "${OC_VERSION}"
```

```
name: sriov-network-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
EOF
```

4.

**Operator가 설치되었는지 확인하려면 다음 명령을 입력합니다.**

```
$ oc get csv -n openshift-sriov-network-operator \
-o custom-columns=Name:.metadata.name,Phase:.status.phase
```

출력 예

```
Name                               Phase
sriov-network-operator.4.12.0-202310121402 Succeeded
```

### 20.2.1.2. 웹 콘솔 : SR-IOV Network Operator 설치

클러스터 관리자는 웹 콘솔을 사용하여 **Operator**를 설치할 수 있습니다.

#### 사전 요구 사항

- **SR-IOV**를 지원하는 하드웨어가 있는 노드로 베어 메탈 하드웨어에 설치된 클러스터.
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 계정.

#### 프로세스

1.

**SR-IOV Network Operator** 설치:

a.

**OpenShift Container Platform** 웹 콘솔에서 **Operator** → **OperatorHub**를 클릭합니다.

- b. 사용 가능한 **Operator** 목록에서 **SR-IOV Network Operator**를 선택한 다음 설치를 클릭합니다.
- c. **Operator** 설치 페이지의 설치된 네임스페이스에서 **Operator** 권장 네임스페이스를 선택합니다.
- d. 설치를 클릭합니다.

2.

**SR-IOV Network Operator**가 설치되었는지 확인하십시오.

- a. **Operator** → 설치된 **Operator** 페이지로 이동합니다.
- b. **SR-IOV Network Operator**가 **openshift-sriov-network-operator** 프로젝트에 **InstallSucceeded** 상태로 나열되어 있는지 확인하십시오.



## 참고

설치 중에 **Operator**는 실패 상태를 표시할 수 있습니다. 나중에 **InstallSucceeded** 메시지와 함께 설치에 성공하면 이 실패 메시지를 무시할 수 있습니다.

**Operator**가 설치된 것으로 나타나지 않으면 다음과 같이 추가 문제 해결을 수행합니다.

- **Operator** 서브스크립션 및 설치 계획 탭의 상태 아래에서 장애 또는 오류가 있는지 점검합니다.
- **Workloads** → **Pod** 페이지로 이동하여 **openshift-sriov-network-operator** 프로젝트에서 **Pod** 로그를 확인하십시오.
- **YAML** 파일의 네임스페이스를 확인합니다. 주석이 누락된 경우 다음 명령을 사용하여 주석 **workload.openshift.io/allowed=management**를 **Operator** 네임스페이스에 추가할 수 있습니다.

```
$ oc annotate ns/openshift-sriov-network-operator workload.openshift.io/allowed=management
```



참고

단일 노드 OpenShift 클러스터의 경우 주석 `workload.openshift.io/allowed=management` 가 네임스페이스에 필요합니다.

### 20.2.2. 다음 단계

- 선택사항: [SR-IOV Network Operator 구성](#)

## 20.3. SR-IOV NETWORK OPERATOR 구성

SR-IOV(Single Root I/O Virtualization) Network Operator는 클러스터의 SR-IOV 네트워크 장치 및 네트워크 첨부 파일을 관리합니다.

### 20.3.1. SR-IOV Network Operator 구성



중요

SR-IOV Network Operator 구성 수정은 일반적으로 필요하지 않습니다. 대부분의 사용 사례에는 기본 구성이 권장됩니다. Operator의 기본 동작이 사용 사례와 호환되지 않는 경우에만 관련 구성을 수정하는 단계를 완료하십시오.

SR-IOV Network Operator는 `SriovOperatorConfig.sriovnetwork.openshift.io CustomResourceDefinition` 리소스를 추가합니다. Operator는 `openshift-sriov-network-operator` 네임스페이스에 `default` 라는 `SriovOperatorConfig CR`(사용자 정의 리소스)을 자동으로 생성합니다.



참고

`default CR`에는 클러스터에 대한 SR-IOV Network Operator 구성이 포함됩니다. Operator 구성을 변경하려면 이 CR을 수정해야 합니다.

#### 20.3.1.1. SR-IOV Network Operator 구성 사용자 정의 리소스



다음 표에는 `sriovoperatorconfig` 사용자 정의 리소스의 필드가 설명되어 있습니다.

표 20.2. SR-IOV Network Operator 구성 사용자 정의 리소스

필드	유형	설명
<code>metadata.name</code>	string	SR-IOV Network Operator 인스턴스의 이름을 지정합니다. 기본 값은 <b>default</b> 입니다. 다른 값을 설정하지 마십시오.
<code>metadata.name space</code>	string	SR-IOV Network Operator 인스턴스의 네임스페이스를 지정합니다. 기본 값은 <b>openshift-sriov-network-operator</b> 입니다. 다른 값을 설정하지 마십시오.
<code>spec.configDaemonNodeSelector</code>	string	선택한 노드에서 SR-IOV Network Config Daemon을 제어하는 노드 선택을 지정합니다. 기본적으로 이 필드는 설정되지 않으며 Operator는 작업자 노드에 SR-IOV Network Config 데몬 세트를 배포합니다.
<code>spec.disableDrain</code>	boolean	노드에 NIC를 구성하기 위해 새 정책을 적용할 때 노드 트레이닝 프로세스를 비활성화하거나 노드 트레이닝 프로세스를 활성화할지 여부를 지정합니다. 이 필드를 <b>true</b> 로 설정하면 소프트웨어 개발 및 OpenShift Container Platform을 단일 노드에 설치할 수 있습니다. 기본적으로 이 필드는 설정되지 않습니다.  단일 노드 클러스터의 경우 Operator를 설치한 후 이 필드를 <b>true</b> 로 설정합니다. 이 필드는 <b>true</b> 로 설정되어야 합니다.
<code>spec.enableInjector</code>	boolean	Network Resources Injector 데몬 세트를 활성화하거나 비활성화할지 여부를 지정합니다. 기본적으로 이 필드는 <b>true</b> 로 설정됩니다.
<code>spec.enableOperatorWebhook</code>	boolean	Operator Admission Controller 웹 후크 데몬 세트를 활성화하거나 비활성화할지 여부를 지정합니다. 기본적으로 이 필드는 <b>true</b> 로 설정됩니다.
<code>spec.logLevel</code>	integer	Operator의 로그 세부 정보 표시 수준을 지정합니다. 기본 로그만 표시하려면 <b>0</b> 으로 설정합니다. 사용 가능한 모든 로그를 표시하려면 <b>2</b> 로 설정합니다. 기본적으로 이 필드는 <b>2</b> 로 설정됩니다.

### 20.3.1.2. Network Resources Injector 정보

**Network Resources Injector**는 **Kubernetes Dynamic Admission Controller** 애플리케이션입니다. 다음과 같은 기능을 제공합니다.

- SR-IOV 네트워크 연결 정의 주석에 따라 SR-IOV 리소스 이름을 추가하기 위해 Pod 사양의 리소스 요청 및 제한 변경**

- Pod 사양을 Downward API 볼륨으로 변경하여 Pod 주석, 라벨 및 대규모 페이지 요청 및 제한을 노출합니다. pod에서 실행되는 컨테이너는 /etc/podnetinfo 경로에 있는 파일로 노출된 정보에 액세스할 수 있습니다.**

기본적으로 **Network Resources Injector**는 **SR-IOV Network Operator**에 의해 활성화되며 모든 컨트롤 플레인 노드에서 데몬 세트로 실행됩니다. 다음은 3개의 컨트롤 플레인 노드가 있는 클러스터에서 실행 중인 **Network Resources Injector Pod**의 예입니다.

```
$ oc get pods -n openshift-sriov-network-operator
```

출력 예

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

### 20.3.1.3. SR-IOV 네트워크 Operator Admission Controller webhook 정보

**SR-IOV 네트워크 Operator Admission Controller webhook**은 **Kubernetes Dynamic Admission Controller** 애플리케이션입니다. 다음과 같은 기능을 제공합니다.

- SriovNetworkNodePolicy CR이 생성 또는 업데이트될 때 유효성 검사**
- CR을 만들거나 업데이트할 때 priority 및 deviceType 필드의 기본값을 설정하여 SriovNetworkNodePolicy CR 변경**

기본적으로 **SR-IOV 네트워크 Operator Admission Controller** 웹 후크는 **Operator**에서 활성화하며 모든 컨트롤 플레인 노드에서 데몬 세트로 실행됩니다.



## 참고

**SR-IOV Network Operator Admission Controller** 웹 후크를 비활성화할 때 주의하십시오. 문제 해결과 같은 특정 상황에서 웹 후크를 비활성화하거나 지원되지 않는 장치를 사용하려는 경우 사용할 수 있습니다.

다음은 3개의 컨트롤 플레인 노드가 있는 클러스터에서 실행되는 **Operator Admission Controller** 웹 후크 Pod의 예입니다.

```
$ oc get pods -n openshift-sriov-network-operator
```

출력 예

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

#### 20.3.1.4. 사용자 정의 노드 선택기 정보

**SR-IOV Network Config** 데몬은 클러스터 노드에서 **SR-IOV** 네트워크 장치를 검색하고 구성합니다. 기본적으로 클러스터의 모든 **worker** 노드에 배포됩니다. 노드 레이블을 사용하여 **SR-IOV Network Config** 데몬이 실행되는 노드를 지정할 수 있습니다.

#### 20.3.1.5. Network Resources Injector 비활성화 또는 활성화

기본적으로 활성화되어 있는 **Network Resources Injector**를 비활성화하거나 활성화하려면 다음 절차를 완료하십시오.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

- **SR-IOV Network Operator가 설치되어 있어야 합니다.**

프로세스

- **enableInjector 필드를 설정합니다. 기능을 비활성화하려면 <value>를 false로 바꾸고 기능을 활성화하려면 true로 바꿉니다.**

```
$ oc patch sriovoperatorconfig default \
--type=merge -n openshift-sriov-network-operator \
--patch '{ "spec": { "enableInjector": <value> } }'
```

작은 정보

또는 다음 YAML을 적용하여 Operator를 업데이트할 수 있습니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: <value>
```

20.3.1.6. SR-IOV 네트워크 Operator Admission Controller webhook 비활성화 또는 활성화

Admission Controller webhook를 비활성화하거나 활성화하려면(기본적으로 활성화되어 있음) 다음 절차를 완료하십시오.

사전 요구 사항

- **OpenShift CLI(oc)를 설치합니다.**
- **cluster-admin 권한이 있는 사용자로 로그인합니다.**
- **SR-IOV Network Operator가 설치되어 있어야 합니다.**

프로세스

- **enableOperatorWebhook 필드를 설정합니다. 기능을 비활성화하려면 <value>를 false로**

바꾸고 활성화하려면 **true**로 바꿉니다.

```
$ oc patch sriovoperatorconfig default --type=merge \
-n openshift-sriov-network-operator \
--patch '{ "spec": { "enableOperatorWebhook": <value> } }'
```

작은 정보

또는 다음 **YAML**을 적용하여 **Operator**를 업데이트할 수 있습니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableOperatorWebhook: <value>
```

### 20.3.1.7. SR-IOV Network Config 데몬에 대한 사용자 정의 NodeSelector 구성

**SR-IOV Network Config** 데몬은 클러스터 노드에서 **SR-IOV** 네트워크 장치를 검색하고 구성합니다. 기본적으로 클러스터의 모든 **worker** 노드에 배포됩니다. 노드 레이블을 사용하여 **SR-IOV Network Config** 데몬이 실행되는 노드를 지정할 수 있습니다.

**SR-IOV Network Config** 데몬이 배포된 노드를 지정하려면 다음 절차를 완료하십시오.

중요

**configDaemonNodeSelector** 필드를 업데이트하면 선택한 각 노드에서 **SR-IOV Network Config** 데몬이 다시 생성됩니다. 데몬이 다시 생성되는 동안 클러스터 사용자는 새로운 **SR-IOV** 네트워크 노드 정책을 적용하거나 새로운 **SR-IOV Pod**를 만들 수 없습니다.

절차

•

**Operator**의 노드 선택기를 업데이트하려면 다음 명령을 입력합니다.

```
$ oc patch sriovoperatorconfig default --type=json \
-n openshift-sriov-network-operator \
--patch '[{
  "op": "replace",
```

```
"path": "/spec/configDaemonNodeSelector",
"value": {<node_label>}
}]'
```

"node-role.kubernetes.io/worker": ""에서와 같이 적용하려면 <node\_label>을 레이블로 바꿉니다.

작은 정보

또는 다음 **YAML**을 적용하여 **Operator**를 업데이트할 수 있습니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    <node_label>
```

### 20.3.1.8. 단일 노드 설치를 위한 SR-IOV Network Operator 구성

기본적으로 **SR-IOV Network Operator**는 모든 정책이 변경되기 전에 노드에서 워크로드를 드레인합니다. **Operator**는 재구성하기 전에 가상 기능을 사용하는 워크로드가 있는지 확인하기 위해 이 작업을 수행합니다.

단일 노드에 설치하는 경우 워크로드를 수신할 다른 노드가 없습니다. 결과적으로 단일 노드에서 워크로드를 드레이닝하지 않도록 **Operator**를 구성해야 합니다.

중요

드레이닝 워크로드를 비활성화하려면 다음 절차를 수행한 후 **SR-IOV** 네트워크 인터페이스를 사용하는 모든 워크로드를 제거한 후 **SR-IOV** 네트워크 노드 정책을 변경해야 합니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

- **SR-IOV Network Operator가 설치되어 있어야 합니다.**

#### 절차

- **disableDrain 필드를 true 로 설정하려면 다음 명령을 입력합니다.**

```
$ oc patch sriovoperatorconfig default --type=merge \
-n openshift-sriov-network-operator \
--patch '{ "spec": { "disableDrain": true } }'
```

#### 작은 정보

또는 다음 **YAML**을 적용하여 **Operator**를 업데이트할 수 있습니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: true
```

### 20.3.2. 다음 단계

- **SR-IOV 네트워크 장치 구성**

## 20.4. SR-IOV 네트워크 장치 구성

클러스터에서 **SR-IOV(Single Root I/O Virtualization)** 장치를 구성할 수 있습니다.

### 20.4.1. SR-IOV 네트워크 노드 구성 오브젝트

**SR-IOV** 네트워크 노드 정책을 생성하여 노드의 **SR-IOV** 네트워크 장치 구성을 지정합니다. 정책의 **API** 오브젝트는 **sriovnetwork.openshift.io** **API** 그룹의 일부입니다.

다음 **YAML**은 **SR-IOV** 네트워크 노드 정책을 설명합니다.

```
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
  nicSelector: 9
    vendor: "<vendor_code>" 10
    deviceID: "<device_id>" 11
    pfNames: ["<pf_name>", ...] 12
    rootDevices: ["<pci_bus_id>", ...] 13
    netFilter: "<filter_string>" 14
  deviceType: <device_type> 15
  isRdma: false 16
  linkType: <link_type> 17
  eSwitchMode: "switchdev" 18

```

1

사용자 정의 리소스 오브젝트의 이름입니다.

2

SR-IOV Network Operator가 설치된 네임스페이스입니다.

3

SR-IOV 네트워크 장치 플러그인의 리소스 이름입니다. 리소스 이름에 대한 SR-IOV 네트워크 노드 정책을 여러 개 생성할 수 있습니다.

이름을 지정할 때 `resourceName` 에서 허용된 구문 표현식 `^[a-zA-Z0-9_]+$` 를 사용해야 합니다.

4

노드 선택기는 구성할 노드를 지정합니다. 선택한 노드의 SR-IOV 네트워크 장치만 구성됩니다. SR-IOV CNI(Container Network Interface) 플러그인 및 장치 플러그인은 선택한 노드에만 배포됩니다.

5



6

선택사항: 가상 기능의 최대 전송 단위(MTU)입니다. 최대 MTU 값은 네트워크 인터페이스 컨트롤러(NIC) 모델마다 다를 수 있습니다.

7

선택 사항: pod에 /dev/vhost-net 장치를 마운트하려면 needVhostNet을 true로 설정합니다. DPDK(Data Plane Development Kit)와 함께 마운트된 /dev/vhost-net 장치를 사용하여 트래픽을 커널 네트워크 스택으로 전달합니다.

8

SR-IOV 물리적 네트워크 장치에 생성할 VF(가상 기능) 수입니다. Intel NIC(Network Interface Controller)의 경우 VF 수는 장치에서 지원하는 총 VF보다 클 수 없습니다. Mellanox NIC의 경우 VF 수는 128보다 클 수 없습니다.

9

NIC 선택기는 Operator가 구성할 장치를 식별합니다. 모든 매개변수에 값을 지정할 필요는 없습니다. 실수로 장치를 선택하지 않도록 네트워크 장치를 정확하게 파악하는 것이 좋습니다.

rootDevices를 지정하면 vendor, deviceID 또는 pfNames의 값도 지정해야 합니다. pfNames와 rootDevices를 동시에 지정하는 경우 동일한 장치를 참조하는지 확인하십시오. netFilter의 값을 지정하는 경우 네트워크 ID가 고유하므로 다른 매개변수를 지정할 필요가 없습니다.

10

선택 사항: SR-IOV 네트워크 장치의 벤더 16진수 코드입니다. 허용되는 값은 8086 및 15b3입니다.

11

선택사항: SR-IOV 네트워크 장치의 장치 16진수 코드입니다. 예를 들어 101b는 Mellanox ConnectX-6 장치의 장치 ID입니다.

12

선택사항: 장치에 대해 하나 이상의 물리적 기능(PF) 이름으로 구성된 배열입니다.

13

선택 사항: 장치의 PF에 대해 하나 이상의 PCI 버스 주소로 구성된 배열입니다. 주소를 0000:02:00.1 형식으로 입력합니다.

14

**선택 사항:** 플랫폼별 네트워크 필터입니다. 지원되는 유일한 플랫폼은 **RHOSP(Red Hat OpenStack Platform)**입니다. 허용 가능한 값은 다음 형식을 사용합니다. `openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx. xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx`를 `/var/config/openstack/latest/network_data.json` 메타데이터 파일의 값으로 바꿉니다.

15

**선택사항:** 가상 기능의 드라이버 유형입니다. 허용되는 값은 **netdevice** 및 **vfio-pci**입니다. 기본값은 **netdevice**입니다.

베어 메탈 노드의 **DPDK** 모드에서 **Mellanox NIC**가 작동하려면 **netdevice** 드라이버 유형을 사용하고 **isRdma**를 **true**로 설정합니다.

16

**선택 사항:** 원격 직접 메모리 액세스(**RDMA**) 모드를 활성화할지 여부를 구성합니다. 기본값은 **false**입니다.

**isRdma** 매개변수가 **true**로 설정된 경우 **RDMA** 사용 **VF**를 일반 네트워크 장치로 계속 사용할 수 있습니다. 어느 모드에서나 장치를 사용할 수 있습니다.

**isRdma**를 **true**로 설정하고 추가로 **needVhostNet**을 **true**로 설정하여 **Fast Datapath DPDK** 애플리케이션에서 사용할 **Mellanox NIC**를 구성합니다.

17

**선택사항:** **VF**의 링크 유형입니다. 기본값은 이더넷의 경우 **eth**입니다. **InfiniBand**의 경우 이 값을 **'ib'**로 변경합니다.

**linkType**을 **ib**로 설정하면 **isRdma**가 **SR-IOV Network Operator** 웹 후크에 의해 자동으로 **true**로 설정됩니다. **linkType**을 **ib**로 설정하면 **deviceType**을 **vfio-pci**로 설정해서는 안 됩니다.

장치 플러그인에서 보고한 장치 수가 잘못될 수 있으므로 **linkType**을 **SriovNetworkNodePolicy**의 **'eth'**로 설정하지 마십시오.

18

**선택 사항:** 하드웨어 오프로드를 활성화하려면 **'eSwitchMode'** 필드를 **"switchdev"** 로 설정해야 합니다.

### 20.4.1.1. SR-IOV 네트워크 노드 구성 예

다음 예제에서는 **InfiniBand** 장치의 구성을 설명합니다.

#### InfiniBand 장치의 구성 예

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SrioVNetworkNodePolicy
metadata:
  name: policy-ib-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: ibnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
  rootDevices:
    - "0000:19:00.0"
  linkType: ib
  isRdma: true

```

다음 예제에서는 **RHOSP** 가상 머신의 **SR-IOV** 네트워크 장치에 대한 구성을 설명합니다.

#### 가상 머신의 SR-IOV 장치 구성 예

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SrioVNetworkNodePolicy
metadata:
  name: policy-sriov-net-openstack-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 1 1
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
  netFilter: "openstack/NetworkID:ea24bd04-8674-4f69-b0ee-fa0b3bd20509" 2

```

1

가상 머신에 대한 노드 네트워크 정책을 구성할 때 `numVfs` 필드는 항상 1로 설정됩니다.

2

가상 머신이 RHOSP에 배포될 때 `netFilter` 필드는 네트워크 ID를 참조해야 합니다. `netFilter`의 유효한 값은 `SriovNetworkNodeState` 오브젝트에서 사용할 수 있습니다.

#### 20.4.1.2. SR-IOV 장치의 VF(가상 기능) 파티셔닝

경우에 따라 동일한 물리적 기능(PF)의 VF(가상 기능)를 여러 리소스 풀로 분할할 수 있습니다. 예를 들어, 일부 VF를 기본 드라이버로 로드하고 나머지 VF를 `vfio-pci` 드라이버로 로드할 수 있습니다. 이러한 배포에서 `SriovNetworkNodePolicy CR`(사용자 정의 리소스)의 `pfNames` 선택기를 사용하여 `<pfname>#<first_vf>-<last_vf>` 형식을 사용하여 풀의 VF 범위를 지정할 수 있습니다.

예를 들어 다음 YAML은 VF 2에서 7까지의 `netpf0` 인터페이스에 대한 선택기를 보여줍니다.

```
pfNames: ["netpf0#2-7"]
```

- `netpf0`은 PF 인터페이스 이름입니다.
- 2는 범위에 포함된 첫 번째 VF 인덱스(0 기반)입니다.
- 7은 범위에 포함된 마지막 VF 인덱스(0 기반)입니다.

다음 요구 사항이 충족되면 다른 정책 CR을 사용하여 동일한 PF에서 VF를 선택할 수 있습니다.

- 동일한 PF를 선택하는 정책의 경우 `numVfs` 값이 동일해야 합니다.
- VF 색인은 0에서 `<numVfs>-1`까지의 범위 내에 있어야 합니다. 예를 들어, `numVfs`가 8로 설정된 정책이 있는 경우 `<first_vf>` 값은 0보다 작아야 하며 `<last_vf>`는 7보다 크지 않아야 합니다.

- 다른 정책의 VF 범위는 겹치지 않아야 합니다.
- <first\_vf>는 <last\_vf>보다 클 수 없습니다.

다음 예는 SR-IOV 장치의 NIC 파티셔닝을 보여줍니다.

정책 `policy-net-1`은 기본 VF 드라이버와 함께 PF `netpf0`의 VF 0을 포함하는 리소스 풀 `net-1`을 정의합니다. 정책 `policy-net-1-dpdk`는 vfio VF 드라이버와 함께 PF `netpf0`의 VF 8 ~ 15를 포함하는 리소스 풀 `net-1-dpdk`를 정의합니다.

정책 `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
```

정책 `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
```

인터페이스가 성공적으로 분할되었는지 확인

다음 명령을 실행하여 SR-IOV 장치의 VF(가상 기능)로 분할된 인터페이스를 확인합니다.

`$ ip link show <interface>` **1**

**1**

& It;interface >를 SR-IOV 장치의 VF로 파티셔닝할 때 지정한 인터페이스로 교체합니다(예: ens3f1 ).

출력 예

```
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
```

### 20.4.2. SR-IOV 네트워크 장치 구성

SR-IOV Network Operator는 SrioNetworkNodePolicy.sriovnetwork.openshift.io CustomResourceDefinition을 OpenShift Container Platform에 추가합니다. SrioNetworkNodePolicy CR(사용자 정의 리소스)을 만들어 SR-IOV 네트워크 장치를 구성할 수 있습니다.



참고

SrioNetworkNodePolicy 오브젝트에 지정된 구성을 적용하면 SR-IOV Operator가 노드를 비우고 경우에 따라 노드를 재부팅할 수 있습니다.

구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다.

## 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **SR-IOV Network Operator**가 설치되어 있습니다.
- 비운 노드에서 제거된 워크로드를 처리하기 위해 클러스터에 사용 가능한 노드가 충분합니다.
- **SR-IOV** 네트워크 장치 구성에 대한 컨트롤 플레인 노드를 선택하지 않았습니다.

## 절차

1. **SriovNetworkNodePolicy** 오브젝트를 생성한 후 **YAML**을 **<name>-sriov-node-network.yaml** 파일에 저장합니다. **<name>**을 이 구성의 이름으로 바꿉니다.
2. 선택 사항: **SR-IOV** 가능 클러스터 노드에 **SriovNetworkNodePolicy.Spec.NodeSelector** 레이블이 지정되지 않은 경우 레이블을 지정합니다. 노드 레이블링에 대한 자세한 내용은 "노드에서 라벨을 업데이트하는 방법"을 참조하십시오.
3. **SriovNetworkNodePolicy** 오브젝트를 생성합니다.
 

```
$ oc create -f <name>-sriov-node-network.yaml
```

**<name>**은 이 구성의 이름을 지정합니다.

구성 업데이트를 적용하면 **sriov-network-operator** 네임스페이스의 모든 **Pod**가 **Running** 상태로 전환됩니다.
4. **SR-IOV** 네트워크 장치가 구성되어 있는지 확인하려면 다음 명령을 입력합니다. **<node\_name>**을 방금 구성한 **SR-IOV** 네트워크 장치가 있는 노드 이름으로 바꿉니다.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

추가 리소스

- [노드에서 라벨을 업데이트하는 방법 이해](#)

### 20.4.3. SR-IOV 구성 문제 해결

SR-IOV 네트워크 장치를 구성하는 절차를 수행한 후 다음 섹션에서는 일부 오류 조건을 다룹니다.

노드 상태를 표시하려면 다음 명령을 실행합니다.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name>
```

여기서 <node\_name>은 SR-IOV 네트워크 장치가 있는 노드의 이름을 지정합니다.

오류 출력 : 메모리를 할당할 수 없음

```
"lastSyncError": "write /sys/bus/pci/devices/0000:3b:00.1/sriov_numvfs: cannot allocate memory"
```

노드가 메모리를 할당할 수 없음을 나타내는 경우 다음 항목을 확인합니다.

- [글로벌 SR-IOV 설정이 노드의 BIOS에서 활성화되어 있는지 확인합니다.](#)
- [BIOS에서 노드에 대해 VT-d가 활성화되어 있는지 확인합니다.](#)

### 20.4.4. SR-IOV 네트워크를 VRF에 할당

클러스터 관리자는 CNI VRF 플러그인을 사용하여 SR-IOV 네트워크 인터페이스를 VRF 도메인에 할당할 수 있습니다.



이렇게 하려면 **SriovNetwork** 리소스의 선택적 **metaPlugins** 매개변수에 **VRF** 구성을 추가합니다.

#### 참고

**VRF**를 사용하는 애플리케이션은 특정 장치에 바인딩해야 합니다. 일반적인 사용은 소켓에 **SO\_BINDTODEVICE** 옵션을 사용하는 것입니다. **SO\_BINDTODEVICE**는 소켓을 전달된 인터페이스 이름(예: **eth1**)에 지정된 장치에 바인딩합니다. **SO\_BINDTODEVICE**를 사용하려면 애플리케이션에 **CAP\_NET\_RAW** 기능이 있어야 합니다.

**OpenShift Container Platform Pod**에서는 **ip vrf exec** 명령을 통해 **VRF**를 사용할 수 있습니다. **VRF**를 사용하려면 애플리케이션을 **VRF** 인터페이스에 직접 바인딩합니다.

#### 20.4.4.1. CNI VRF 플러그인으로 추가 SR-IOV 네트워크 연결 생성

**SR-IOV Network Operator**는 추가 네트워크 정의를 관리합니다. 생성할 추가 **SR-IOV** 네트워크를 지정하면 **SR-IOV Network Operator**가 **NetworkAttachmentDefinition CR**(사용자 정의 리소스)을 자동으로 생성합니다.

#### 참고

**SR-IOV Network Operator**가 관리하는 **NetworkAttachmentDefinition** 사용자 정의 리소스를 편집하지 마십시오. 편집하면 추가 네트워크의 네트워크 트래픽이 중단될 수 있습니다.

**CNI VRF** 플러그인으로 추가 **SR-IOV** 네트워크 연결을 생성하려면 다음 절차를 수행합니다.

#### 사전 요구 사항

- **OpenShift Container Platform CLI, oc**를 설치합니다.
- **cluster-admin** 역할의 사용자로 **OpenShift Container Platform** 클러스터에 로그인합니다.

#### 절차

1. 추가 **SR-IOV** 네트워크 연결에 대한 **SriovNetwork CR**(사용자 정의 리소스)을 생성하고 다

음 예제 CR과 같이 **metaPlugins** 구성을 삽입합니다. **YAML**을 **sriov-network-attachment.yaml** 파일로 저장합니다.

```
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: intelnic3
  metaPlugins : |
    {
      "type": "vrf", ①
      "vrfname": "example-vrf-name" ②
    }
```

①

`type`은 `vrf`로 설정해야 합니다.

②

`vrfname`은 인터페이스가 할당된 **VRF**의 이름입니다. 포드에 없는 경우 생성됩니다.

2.

**SriovNetwork** 리소스를 생성합니다.

```
$ oc create -f sriov-network-attachment.yaml
```

**NetworkAttachmentDefinition CR**이 성공적으로 생성되었는지 확인

•

**SR-IOV Network Operator**가 다음 명령을 실행하여 **NetworkAttachmentDefinition CR**을 생성했는지 확인합니다.

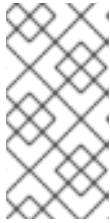
```
$ oc get network-attachment-definitions -n <namespace> ①
```

1

**<namespace>**를 네트워크 연결을 구성할 때 지정한 네임스페이스(예: **additional-sriov-network-1**)로 바꿉니다.

출력 예

NAME	AGE
additional-sriov-network-1	14m



참고

**SR-IOV Network Operator**가 **CR**을 생성하기 전에 지연이 발생할 수 있습니다.

추가 **SR-IOV** 네트워크 연결에 성공했는지 확인

**VRF CNI**가 올바르게 구성되어 추가 **SR-IOV** 네트워크 연결이 연결되었는지 확인하려면 다음을 수행하십시오.

1. **VRF CNI**를 사용하는 **SR-IOV** 네트워크를 생성합니다.
2. 포트에 네트워크를 할당합니다.
3. 포트 네트워크 연결이 **SR-IOV** 추가 네트워크에 연결되어 있는지 확인합니다. Pod로 원격 셸을 설치하고 다음 명령을 실행합니다.

```
$ ip vrf show
```

출력 예

Name	Table
red	10

4. **VRF 인터페이스가 보조 인터페이스의 마스터인지 확인합니다.**

```
$ ip link
```

출력 예

```
...
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master
red state UP mode
...
```

#### 20.4.5. 다음 단계

- [SR-IOV 네트워크 연결 구성](#)

### 20.5. SR-IOV 이더넷 네트워크 연결 구성

클러스터에서 **SR-IOV(Single Root I/O Virtualization)** 장치에 대한 이더넷 네트워크 연결을 구성할 수 있습니다.

#### 20.5.1. 이더넷 장치 구성 오브젝트

**SriovNetwork** 오브젝트를 정의하여 이더넷 네트워크 장치를 구성할 수 있습니다.

다음 **YAML**은 **SriovNetwork** 오브젝트를 설명합니다.

```
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

오브젝트의 이름입니다. **SR-IOV Network Operator**는 동일한 이름으로 **NetworkAttachmentDefinition** 오브젝트를 생성합니다.

2

**SR-IOV Network Operator**가 설치된 네임스페이스입니다.

3

이 추가 네트워크에 대한 **SR-IOV** 하드웨어를 정의하는 **SriovNetworkNodePolicy** 오브젝트의 **spec.resourceName** 매개변수 값입니다.

4

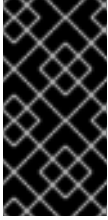
**SriovNetwork** 오브젝트의 대상 네임스페이스입니다. 대상 네임스페이스의 포드만 추가 네트워크에 연결할 수 있습니다.

5

선택사항: 추가 네트워크의 **VLAN(Virtual LAN) ID**입니다. 정수 값은 **0**에서 **4095** 사이여야 합니다. 기본값은 **0**입니다.

6

선택사항: **VF**의 스푸핑 검사 모드입니다. 허용되는 값은 문자열 **"on"** 및 **"off"**입니다.



중요

**SR-IOV Network Operator**가 지정한 값을 따옴표로 묶거나 오브젝트를 거부해야 합니다.

7

**YAML** 블록 스칼라 **IPAM CNI** 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 **IP** 주소 할당을 관리합니다.

8

선택사항: **VF**(가상 기능)의 링크 상태입니다. 허용되는 값은 **enable**, **disable** 및 **auto**입니다.

9

선택사항: **VF**의 경우 최대 전송 속도(**Mbps**)입니다.

10

선택사항: **VF**의 경우 최소 전송 속도(**Mbps**)입니다. 이 값은 최대 전송 속도보다 작거나 같아야 합니다.



참고

인텔 **NIC**는 **minTxRate** 매개변수를 지원하지 않습니다. 자세한 내용은 [BZ#1772847](#)에서 참조하십시오.

11

선택사항: **VF**의 **IEEE 802.1p** 우선순위 수준입니다. 기본값은 **0**입니다.

12

선택사항: **VF**의 신뢰 모드입니다. 허용되는 값은 문자열 **"on"** 및 **"off"**입니다.



중요

지정한 값을 따옴표로 묶어야 합니다. 그렇지 않으면 **SR-IOV Network Operator**에서 오브젝트를 거부합니다.

13

### 20.5.1.1. 추가 네트워크에 대한 IP 주소 할당 구성

IP 주소 관리(IPAM) CNI(Container Network Interface) 플러그인은 다른 CNI 플러그인에 대한 IP 주소를 제공합니다.

다음 IP 주소 할당 유형을 사용할 수 있습니다.

- 정적 할당
- DHCP 서버를 통한 동적 할당. 지정한 DHCP 서버는 추가 네트워크에서 연결할 수 있어야 합니다.
- Whereabouts IPAM CNI 플러그인을 통한 동적 할당

#### 20.5.1.1.1. 고정 IP 주소 할당 구성

다음 표에서는 고정 IP 주소 할당 구성에 대해 설명합니다.

표 20.3. IPAM 고정 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. <b>static</b> 값이 필요합니다.
주소	array	가상 인터페이스에 할당할 IP 주소를 지정하는 개체의 배열입니다. IPv4 및 IPv6 IP 주소가 모두 지원됩니다.
routes	array	Pod 내부에서 구성할 경로를 지정하는 오브젝트의 배열입니다.
dns	array	선택 사항: DNS 구성을 지정하는 오브젝트의 배열입니다.

addresses 배열에는 다음 필드가 있는 오브젝트가 필요합니다.

표 20.4. ipam.addresses[] array

필드	유형	설명
주소	string	지정하는 IP 주소 및 네트워크 접두사입니다. 예를 들어 <b>10.10.21.10/24</b> 를 지정하면 추가 네트워크에 IP 주소 <b>10.10.21.10</b> 이 할당되고 넷마스크는 <b>255.255.255.0</b> 입니다.
gateway	string	송신 네트워크 트래픽을 라우팅할 기본 게이트웨이입니다.

표 20.5. ipam.routes[] array

필드	유형	설명
dst	string	CIDR 형식의 IP 주소 범위(예: 기본 경로의 경우 <b>192.168.17.0/24</b> 또는 <b>0.0.0.0/0</b> )입니다.
gw	string	네트워크 트래픽이 라우팅되는 게이트웨이입니다.

표 20.6. ipam.dns object

필드	유형	설명
nameservers	array	DNS 쿼리를 보낼 하나 이상의 IP 주소로 구성된 배열입니다.
domain	array	호스트 이름에 추가할 기본 도메인입니다. 예를 들어 도메인이 <b>example.com</b> 으로 설정되면 <b>example-host</b> 에 대한 DNS 조회 쿼리가 <b>example-host.example.com</b> 으로 다시 작성됩니다.
search	array	DNS 조회 쿼리 중에 규정되지 않은 호스트 이름(예: <b>example-host</b> )에 추가할 도메인 이름 배열입니다.

고정 IP 주소 할당 구성 예

```

{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}

```



### 20.5.1.1.2. DHCP(Dynamic IP 주소) 할당 구성

다음 JSON은 DHCP를 사용한 동적 IP 주소 할당 구성을 설명합니다.

#### DHCP 리스 갱신

*pod*는 생성될 때 원래 DHCP 리스를 얻습니다. 리스는 클러스터에서 실행되는 최소 DHCP 서버 배포를 통해 주기적으로 갱신되어야 합니다.

SR-IOV Network Operator는 DHCP 서버 배포를 생성하지 않습니다. Cluster Network Operator는 최소 DHCP 서버 배포를 생성합니다.

DHCP 서버 배포를 트리거하려면 다음 예와 같이 Cluster Network Operator 구성을 편집하여 shim 네트워크 연결을 만들어야 합니다.

#### shim 네트워크 연결 정의 예

```
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"
        }
      }
# ...
```

표 20.7. IPAM DHCP 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 <b>dhcp</b> 가 필요합니다.

**DHCP(Dynamic IP 주소) 할당 구성 예**

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

**20.5.1.1.3. Whereabouts를 사용한 동적 IP 주소 할당 구성**

**Whereabouts CNI 플러그인을 사용하면 DHCP 서버를 사용하지 않고도 IP 주소를 추가 네트워크에 동적으로 할당할 수 있습니다.**

다음 표에서는 **Whereabouts**를 사용한 동적 IP 주소 할당 구성을 설명합니다.

표 20.8. IPAM 위치 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 <b>whereabouts</b> 가 필요합니다.
범위	string	CIDR 표기법의 IP 주소 및 범위입니다. IP 주소는 이 주소 범위 내에서 할당됩니다.
exclude	array	선택 사항: CIDR 표기법으로 0개 이상의 IP 주소 및 범위 목록입니다. 제외된 주소 범위 내의 IP 주소는 할당되지 않습니다.

**Whereabouts를 사용하는 동적 IP 주소 할당 구성 예**

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
```

```

"192.0.2.192/30",
"192.0.2.196/32"
]
}
}

```

### 20.5.2. SR-IOV 추가 네트워크 구성

**SriovNetwork** 오브젝트를 생성하여 **SR-IOV** 하드웨어를 사용하는 추가 네트워크를 구성할 수 있습니다. **SriovNetwork** 오브젝트를 생성하면 **SR-IOV Network Operator**가 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성합니다.



참고

**SriovNetwork** 오브젝트가 **running** 상태의 **Pod**에 연결된 경우 해당 오브젝트를 수정하거나 삭제하지 마십시오.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

1. **SriovNetwork** 오브젝트를 생성한 다음 **<name>.yaml** 파일에 **YAML**을 저장합니다. 여기서 **<name>**은 이 추가 네트워크의 이름입니다. 오브젝트 사양은 다음 예와 유사할 수 있습니다.

```

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.

오브젝트를 생성하려면 다음 명령을 입력합니다:

```
$ oc create -f <name>.yaml
```

여기서 <name>은 추가 네트워크의 이름을 지정합니다.

3.

**선택사항:** 이전 단계에서 생성한 **SriovNetwork** 오브젝트에 연결된 **NetworkAttachmentDefinition** 오브젝트가 존재하는지 확인하려면 다음 명령을 입력합니다. <namespace>를 **SriovNetwork** 오브젝트에 지정한 **networkNamespace**로 바꿉니다.

```
$ oc get net-attach-def -n <namespace>
```

### 20.5.3. 다음 단계

•

[SR-IOV 추가 네트워크에 pod 추가](#)

### 20.5.4. 추가 리소스

•

[SR-IOV 네트워크 장치 구성](#)

## 20.6. SR-IOV INFINIBAND 네트워크 연결 구성

클러스터에서 **SR-IOV(Single Root I/O Virtualization)** 장치에 대한 **IB(InfiniBand)** 네트워크 연결을 구성할 수 있습니다.

### 20.6.1. InfiniBand 장치 구성 오브젝트

**SriovIBNetwork** 오브젝트를 정의하여 **IB(InfiniBand)** 네트워크 장치를 구성할 수 있습니다.

다음 **YAML**은 **SriovIBNetwork** 오브젝트를 설명합니다.

```
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

오브젝트의 이름입니다. **SR-IOV Network Operator**는 동일한 이름으로 **NetworkAttachmentDefinition** 오브젝트를 생성합니다.

2

**SR-IOV Operator**가 설치된 네임스페이스입니다.

3

이 추가 네트워크에 대한 **SR-IOV** 하드웨어를 정의하는 **SriovNetworkNodePolicy** 오브젝트의 **spec.resourceName** 매개변수 값입니다.

4

**SriovIBNetwork** 오브젝트의 대상 네임스페이스입니다. 대상 네임스페이스의 포트만 네트워크 장치에 연결할 수 있습니다.

5

선택사항: **YAML** 블록 스칼라인 **IPAM CNI** 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 **IP** 주소 할당을 관리합니다.

6

선택사항: **VF**(가상 기능)의 링크 상태입니다. 허용되는 값은 **enable**, **disable**, **auto**입니다.

7

선택사항: 이 네트워크에 구성할 수 있는 기능입니다. **IP** 주소 지원을 사용하려면 "{ **"ips"**: true }"를 지정하고 **IB GUID(Global Unique Identifier)** 지원을 사용하려면 "{ **"infinibandGUID"**: true }"를 지정하면 됩니다.

#### 20.6.1.1. 추가 네트워크에 대한 IP 주소 할당 구성

**IP 주소 관리(IPAM) CNI(Container Network Interface) 플러그인은 다른 CNI 플러그인에 대한 IP 주소를 제공합니다.**

다음 IP 주소 할당 유형을 사용할 수 있습니다.

- 정적 할당
- **DHCP** 서버를 통한 동적 할당. **지정한 DHCP 서버는 추가 네트워크에서 연결할 수 있어야 합니다.**
- **Whereabouts IPAM CNI 플러그인을 통한 동적 할당**

#### 20.6.1.1.1. 고정 IP 주소 할당 구성

다음 표에서는 고정 IP 주소 할당 구성에 대해 설명합니다.

표 20.9. IPAM 고정 구성 오브젝트

필드	유형	설명
<b>type</b>	<b>string</b>	IPAM 주소 유형입니다. <b>static</b> 값이 필요합니다.
주소	<b>array</b>	가상 인터페이스에 할당할 IP 주소를 지정하는 개체의 배열입니다. IPv4 및 IPv6 IP 주소가 모두 지원됩니다.
<b>routes</b>	<b>array</b>	Pod 내부에서 구성할 경로를 지정하는 오브젝트의 배열입니다.
<b>dns</b>	<b>array</b>	선택 사항: DNS 구성을 지정하는 오브젝트의 배열입니다.

**addresses** 배열에는 다음 필드가 있는 오브젝트가 필요합니다.

표 20.10. ipam.addresses[] array

필드	유형	설명
주소	<b>string</b>	지정하는 IP 주소 및 네트워크 접두사입니다. 예를 들어 <b>10.10.21.10/24</b> 를 지정하면 추가 네트워크에 IP 주소 <b>10.10.21.10</b> 이 할당되고 넷마스크는 <b>255.255.255.0</b> 입니다.

필드	유형	설명
gateway	string	송신 네트워크 트래픽을 라우팅할 기본 게이트웨이입니다.

표 20.11. ipam.routes[] array

필드	유형	설명
dst	string	CIDR 형식의 IP 주소 범위(예: 기본 경로의 경우 <b>192.168.17.0/24</b> 또는 <b>0.0.0.0/0</b> )입니다.
gw	string	네트워크 트래픽이 라우팅되는 게이트웨이입니다.

표 20.12. ipam.dns object

필드	유형	설명
nameservers	array	DNS 쿼리를 보낼 하나 이상의 IP 주소로 구성된 배열입니다.
domain	array	호스트 이름에 추가할 기본 도메인입니다. 예를 들어 도메인이 <b>example.com</b> 으로 설정되면 <b>example-host</b> 에 대한 DNS 조회 쿼리가 <b>example-host.example.com</b> 으로 다시 작성됩니다.
search	array	DNS 조회 쿼리 중에 규정되지 않은 호스트 이름(예: <b>example-host</b> )에 추가할 도메인 이름 배열입니다.

## 고정 IP 주소 할당 구성 예

```
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}
```

## 20.6.1.1.2. DHCP(Dynamic IP 주소) 할당 구성

다음 JSON은 DHCP를 사용한 동적 IP 주소 할당 구성을 설명합니다.

### DHCP 리스 갱신

pod는 생성될 때 원래 DHCP 리스를 얻습니다. 리스는 클러스터에서 실행되는 최소 DHCP 서버 배포를 통해 주기적으로 갱신되어야 합니다.

DHCP 서버 배포를 트리거하려면 다음 예와 같이 Cluster Network Operator 구성을 편집하여 shim 네트워크 연결을 만들어야 합니다.

### shim 네트워크 연결 정의 예

```
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"
        }
      }
# ...
```

표 20.13. IPAM DHCP 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 <b>dhcp</b> 가 필요합니다.

### DHCP(Dynamic IP 주소) 할당 구성 예



```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

### 20.6.1.1.3. Whereabouts를 사용한 동적 IP 주소 할당 구성

**Whereabouts CNI** 플러그인을 사용하면 **DHCP** 서버를 사용하지 않고도 **IP** 주소를 추가 네트워크에 동적으로 할당할 수 있습니다.

다음 표에서는 **Whereabouts**를 사용한 동적 **IP** 주소 할당 구성을 설명합니다.

표 20.14. IPAM 위치 구성 오브젝트

필드	유형	설명
<b>type</b>	<b>string</b>	IPAM 주소 유형입니다. 값 <b>whereabouts</b> 가 필요합니다.
범위	<b>string</b>	CIDR 표기법의 IP 주소 및 범위입니다. IP 주소는 이 주소 범위 내에서 할당됩니다.
<b>exclude</b>	<b>array</b>	선택 사항: CIDR 표기법으로 0개 이상의 IP 주소 및 범위 목록입니다. 제외된 주소 범위 내의 IP 주소는 할당되지 않습니다.

### Whereabouts를 사용하는 동적 IP 주소 할당 구성 예

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

### 20.6.2. SR-IOV 추가 네트워크 구성

**SriovIBNetwork** 오브젝트를 생성하여 **SR-IOV** 하드웨어를 사용하는 추가 네트워크를 구성할 수 있습니다. **SriovIBNetwork** 오브젝트를 생성하면 **SR-IOV Network Operator**가 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성합니다.



#### 참고

**SriovIBNetwork** 오브젝트가 **running** 상태의 **Pod**에 연결된 경우 해당 오브젝트를 수정하거나 삭제하지 마십시오.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

1. **SriovIBNetwork** 오브젝트를 생성한 다음 **<name>.yaml** 파일에 **YAML**을 저장합니다. 여기서 **<name>**은 이 추가 네트워크의 이름입니다. 오브젝트 사양은 다음 예와 유사할 수 있습니다.

```

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.

오브젝트를 생성하려면 다음 명령을 입력합니다:

```
$ oc create -f <name>.yaml
```

여기서 <name>은 추가 네트워크의 이름을 지정합니다.

3.

선택 사항: 이전 단계에서 생성한 **SriovIBNetwork** 오브젝트에 연결된 **NetworkAttachmentDefinition** 오브젝트가 존재하는지 확인하려면 다음 명령을 입력합니다. <namespace>를 **SriovIBNetwork** 오브젝트에 지정한 **networkNamespace**로 바꿉니다.

```
$ oc get net-attach-def -n <namespace>
```

### 20.6.3. 다음 단계

•

[SR-IOV 추가 네트워크에 pod 추가](#)

### 20.6.4. 추가 리소스

•

[SR-IOV 네트워크 장치 구성](#)

## 20.7. SR-IOV 추가 네트워크에 POD 추가

기존 SR-IOV(Single Root I/O Virtualization) 네트워크에 pod를 추가할 수 있습니다.

### 20.7.1. 네트워크 연결을 위한 런타임 구성

추가 네트워크에 pod를 연결할 때 런타임 구성을 지정하여 pod에 대한 특정 사용자 정의를 수행할 수 있습니다. 예를 들어 특정 MAC 하드웨어 주소를 요청할 수 있습니다.

Pod 사양에서 주석을 설정하여 런타임 구성을 지정합니다. 주석 키는 `k8s.v1.cni.cncf.io/networks`이며 런타임 구성을 설명하는 JSON 오브젝트를 허용합니다.

#### 20.7.1.1. 이더넷 기반 SR-IOV 연결을 위한 런타임 구성

다음 JSON은 이더넷 기반 SR-IOV 네트워크 연결에 대한 런타임 구성 옵션을 설명합니다.

```
[
  {
    "name": "<name>", 1
    "mac": "<mac_address>", 2
    "ips": ["<cidr_range>"] 3
  }
]
```

1

SR-IOV 네트워크 연결 정의 CR의 이름입니다.

2

선택사항: SR-IOV 네트워크 연결 정의 CR에 정의된 리소스 유형에서 할당된 SR-IOV 장치의 MAC 주소입니다. 이 기능을 사용하려면 SrioNetwork 오브젝트에 { "mac": true }도 지정해야 합니다.

3

선택사항: SR-IOV 네트워크 연결 정의 CR에 정의된 리소스 유형에서 할당된 SR-IOV 장치의 IP 주소입니다. IPv4 및 IPv6 주소가 모두 지원됩니다. 이 기능을 사용하려면 SrioNetwork 오브젝트에 { "ips": true }도 지정해야 합니다.

### 컨테이너 구성 예

```
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"]
```

### 20.7.1.2. InfiniBand 기반 SR-IOV 연결을 위한 런타임 구성

다음 JSON은 InfiniBand 기반 SR-IOV 네트워크 연결에 대한 런타임 구성 옵션을 설명합니다.

```
[
  {
    "name": "<network_attachment>", 1
    "infiniband-guid": "<guid>", 2
    "ips": ["<cidr_range>"] 3
  }
]
```

1

SR-IOV 네트워크 연결 정의 CR의 이름입니다.

2

SR-IOV 장치의 InfiniBand GUID입니다. 이 기능을 사용하려면 `SriovIBNetwork` 오브젝트에 `{ "infinibandGUID": true }`도 지정해야 합니다.

3

SR-IOV 네트워크 연결 정의 CR에 정의된 리소스 유형에서 할당된 SR-IOV 장치의 IP 주소입니다. IPv4 및 IPv6 주소가 모두 지원됩니다. 이 기능을 사용하려면 `SriovIBNetwork` 오브젝트에 `{ "ips": true }`도 지정해야 합니다.

런타임 구성 예

```
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"]
```

**20.7.2. 추가 네트워크에 Pod 추가**

추가 네트워크에 Pod를 추가할 수 있습니다. Pod는 기본 네트워크를 통해 정상적인 클러스터 관련 네트워크 트래픽을 계속 전송합니다.

Pod가 생성되면 추가 네트워크가 연결됩니다. 그러나 Pod가 이미 있는 경우에는 추가 네트워크를 연결할 수 없습니다.

Pod는 추가 네트워크와 동일한 네임스페이스에 있어야 합니다.

**참고**

**SR-IOV Network Resource Injector**는 리소스 필드를 Pod의 첫 번째 컨테이너에 자동으로 추가합니다.

**DPDK(Data Plane Development Kit)** 모드에서 Intel NIC(네트워크 인터페이스 컨트롤러)를 사용하는 경우 Pod의 첫 번째 컨테이너만 NIC에 액세스하도록 구성되어 있습니다. **SriovNetworkNodePolicy** 오브젝트에서 **deviceType** 이 **vfiopci** 로 설정된 경우 SR-IOV 추가 네트워크가 DPDK 모드에 대해 구성됩니다.

NIC에 액세스해야 하는 컨테이너가 Pod 오브젝트에 정의된 첫 번째 컨테이너인지 아니면 **Network Resource Injector**를 비활성화하여 이 문제를 해결할 수 있습니다. 자세한 내용은 [BZ#1990953](#) 에서 참조하십시오.

**사전 요구 사항**

- **OpenShift CLI(oc)**를 설치합니다.
- 클러스터에 로그인합니다.

- **SR-IOV Operator**를 설치합니다.
- **Pod**를 연결할 **SriovNetwork** 오브젝트 또는 **SriovIBNetwork** 오브젝트를 생성합니다.

### 절차

1. **Pod** 오브젝트에 주석을 추가합니다. 다음 주석 형식 중 하나만 사용할 수 있습니다.
  - a. 사용자 정의 없이 추가 네트워크를 연결하려면 다음 형식으로 주석을 추가합니다. **<network>**를 **Pod**와 연결할 추가 네트워크의 이름으로 변경합니다.

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

1

둘 이상의 추가 네트워크를 지정하려면 각 네트워크를 쉼표로 구분합니다. 쉼표 사이에 공백을 포함하지 마십시오. 동일한 추가 네트워크를 여러 번 지정하면 **Pod**에 해당 네트워크에 대한 인터페이스가 여러 개 연결됩니다.

- b. 사용자 정의된 추가 네트워크를 연결하려면 다음 형식으로 주석을 추가합니다.

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>", 1
          "namespace": "<namespace>", 2
          "default-route": ["<default-route>"] 3
        }
      ]
```

1

**NetworkAttachmentDefinition** 오브젝트에서 정의한 추가 네트워크의 이름을 지정합니다.

2

**3**

선택 사항: 기본 경로에 대한 재정의를 지정합니다(예: 192.168.17.1).

2.

Pod를 생성하려면 다음 명령을 입력합니다. <name>을 Pod 이름으로 교체합니다.

```
$ oc create -f <name>.yaml
```

3.

선택사항: Pod CR에 주석이 있는지 확인하려면 다음 명령을 입력하고 <name>을 Pod 이름으로 교체합니다.

```
$ oc get pod <name> -o yaml
```

다음 예에서 **example-pod** Pod는 **net1** 추가 네트워크에 연결되어 있습니다.

```
$ 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/networks-status: |- 1
    [{"name": "openshift-sdn",
      "interface": "eth0",
      "ips": [
        "10.128.2.14"
      ]},
     {"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

`k8s.v1.cni.cncf.io/networks-status` 매개 변수는 **JSON** 오브젝트 배열입니다. 각 오브젝트는 **Pod**에 연결된 추가 네트워크의 상태를 설명합니다. 주석 값은 일반 텍스트 값으로 저장됩니다.

### 20.7.3. NUMA(Non-Uniform Memory Access) 정렬 SR-IOV Pod 생성

**SR-IOV** 및 제한된 또는 **single-numa-node** 토폴로지 관리자 정책으로 동일한 **NUMA** 노드에서 할당된 **CPU** 리소스를 제한하여 **NUMA** 정렬 **SR-IOV Pod**를 생성할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**가 설치되어 있습니다.
- **CPU** 관리자 정책을 **static**으로 구성했습니다. **CPU** 관리자에 대한 자세한 내용은 "추가 리소스" 섹션을 참조하십시오.
- 토폴로지 관리자 정책을 **single-numa-node**로 구성했습니다.



#### 참고

**single-numa-node**가 요청을 충족할 수 없는 경우 **Topology Manager** 정책을 **restricted**로 구성할 수 있습니다.

#### 절차

1. 다음과 같은 **SR-IOV Pod** 사양을 생성한 다음 **YAML**을 `<name>-sriov-pod.yaml` 파일에 저장합니다. `<name>`을 이 **Pod**의 이름으로 바꿉니다.

다음 예는 **SR-IOV Pod** 사양을 보여줍니다.

```
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

<name>을 SR-IOV 네트워크 첨부 파일 정의 CR의 이름으로 바꿉니다.

2

<image>를 sample-pod 이미지의 이름으로 바꿉니다.

3

보장된 QoS로 SR-IOV Pod를 생성하려면 메모리 제한을 메모리 요청과 동일하게 설정합니다.

4

보장된 QoS로 SR-IOV Pod를 생성하려면 cpu 제한을 CPU 요청과 동일하게 설정합니다.

2.

다음 명령을 실행하여 샘플 SR-IOV Pod를 만듭니다.

```
$ oc create -f <filename> 1
```

1

<filename>을 이전 단계에서 생성한 파일 이름으로 바꿉니다.

3.

sample-pod가 보장된 QoS로 구성되어 있는지 확인하십시오.

```
$ oc describe pod sample-pod
```

4. **sample-pod에 전용 CPU가 할당되어 있는지 확인하십시오.**

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. **sample-pod에 할당된 SR-IOV 장치 및 CPU가 동일한 NUMA 노드에 있는지 확인하십시오.**

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

#### 20.7.4. OpenStack에서 SR-IOV를 사용하는 클러스터의 테스트 pod 템플릿

다음 testpmd pod는 대규모 페이지, 예약된 CPU 및 SR-IOV 포트로 컨테이너 생성을 보여줍니다.

##### 예제 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**

이 예에서는 성능 프로파일의 이름이 `cnf-performance` 프로파일 이라고 가정합니다.

### 20.7.5. 추가 리소스

- [SR-IOV 이더넷 네트워크 연결 구성](#)
- [SR-IOV InfiniBand 네트워크 연결 구성](#)
- [CPU 관리자 사용](#)

## 20.8. SR-IOV 네트워크의 인터페이스 수준 네트워크 **SYSCTL** 설정 구성

클러스터 관리자는 **SR-IOV** 네트워크 장치에 연결된 Pod의 **CNI(Container Network Interface)** 메타 플러그인을 사용하여 인터페이스 수준 네트워크 **sysctl**을 수정할 수 있습니다.

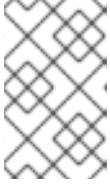
### 20.8.1. SR-IOV 활성화된 NIC로 노드 레이블 지정

**SR-IOV** 가능 노드에서만 **SR-IOV**를 활성화하려면 다음과 같은 몇 가지 방법이 있습니다.

1. **NFD(Node Feature Discovery) Operator**를 설치합니다. **NFD**는 **SR-IOV** 활성화된 **NIC**의 존재를 감지하고 `node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = true` 로 노드에 레이블을 지정합니다.
2. 각 노드의 **SriovNetworkNodeState CR**을 검사합니다. 인터페이스 스탠자에는 작업자 노드에서 **SR-IOV Network Operator**가 검색한 모든 **SR-IOV** 장치 목록이 포함되어 있습니다. 다음 명

령을 사용하여 `feature.node.kubernetes.io/network-sriov.capable: "true"` 로 각 노드에 레이블을 지정합니다.

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```



참고

원하는 이름으로 노드에 라벨을 지정할 수 있습니다.

### 20.8.2. 하나의 sysctl 플래그 설정

**SR-IOV** 네트워크 장치에 연결된 Pod의 인터페이스 수준 네트워크 **sysctl** 설정을 설정할 수 있습니다.

이 예에서는 생성된 가상 인터페이스에서 `net.ipv4.conf.IFNAME.accept_redirects` 가 1로 설정됩니다.

`sysctl-tuning-test` 는 이 예제에서 사용되는 네임스페이스입니다.

- 

다음 명령을 사용하여 `sysctl-tuning-test` 네임스페이스를 생성합니다.

```
$ oc create namespace sysctl-tuning-test
```

#### 20.8.2.1. SR-IOV 네트워크 장치를 사용하여 노드에서 하나의 sysctl 플래그 설정

**SR-IOV Network Operator**는 `SriovNetworkNodePolicy.sriovnetwork.openshift.io` CRD(사용자 정의 리소스 정의)를 **OpenShift Container Platform**에 추가합니다. `SriovNetworkNodePolicy` CR(사용자 정의 리소스)을 생성하여 **SR-IOV** 네트워크 장치를 구성할 수 있습니다.



참고

`SriovNetworkNodePolicy` 오브젝트에 지정된 구성을 적용하면 **SR-IOV Operator**가 노드를 비우고 재부팅할 수 있습니다.

구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다.

**SriovNetworkNodePolicy CR(사용자 정의 리소스)을 생성하려면 다음 절차를 따르십시오.**

## 프로세스

1.

**SriovNetworkNodePolicy CR(사용자 정의 리소스)을 생성합니다. 예를 들어 다음 YAML을 policyoneflag-sriov-node-network.yaml 파일로 저장합니다.**

```
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

사용자 정의 리소스 오브젝트의 이름입니다.

2

SR-IOV Network Operator가 설치된 네임스페이스입니다.

3

SR-IOV 네트워크 장치 플러그인의 리소스 이름입니다. 리소스 이름에 대한 SR-IOV 네트워크 노드 정책을 여러 개 생성할 수 있습니다.

4

노드 선택기는 구성할 노드를 지정합니다. 선택한 노드의 SR-IOV 네트워크 장치만 구성됩니다. SR-IOV CNI(Container Network Interface) 플러그인 및 장치 플러그인은 선택한 노드에만 배포됩니다.

5

선택 사항: 우선순위는 0에서 99 사이의 정수 값입니다. 작은 값은 우선순위가 높습니다. 예를 들어 우선순위 10은 우선순위 99보다 높습니다. 기본값은 99입니다.

6

**SR-IOV 물리적 네트워크 장치에 생성할 VF(가상 기능) 수입니다. Intel NIC(Network Interface Controller)의 경우 VF 수는 장치에서 지원하는 총 VF보다 클 수 없습니다. Mellanox NIC의 경우 VF 수는 128보다 클 수 없습니다.**

7

**NIC 선택기는 Operator가 구성할 장치를 식별합니다. 모든 매개변수에 값을 지정할 필요는 없습니다. 실수로 장치를 선택하지 않도록 네트워크 장치를 정확하게 파악하는 것이 좋습니다. rootDevices를 지정하면 vendor, deviceID 또는 pfNames의 값도 지정해야 합니다. pfNames와 rootDevices를 동시에 지정하는 경우 동일한 장치를 참조하는지 확인하십시오. netFilter의 값을 지정하는 경우 네트워크 ID가 고유하므로 다른 매개변수를 지정할 필요가 없습니다.**

8

**선택사항: 장치에 대해 하나 이상의 물리적 기능(PF) 이름으로 구성된 배열입니다.**

9

**선택사항: 가상 기능의 드라이버 유형입니다. 허용되는 유일한 값은 netdevice입니다. 베어 메탈 노드에서 Mellanox NIC가 DPDK 모드에서 작동하려면 isRdma를 true로 설정합니다.**

10

**선택 사항: 원격 직접 메모리 액세스(RDMA) 모드를 활성화할지 여부를 구성합니다. 기본값은 false입니다. isRdma 매개변수가 true로 설정된 경우 RDMA 사용 VF를 일반 네트워크 장치로 계속 사용할 수 있습니다. 어느 모드에서나 장치를 사용할 수 있습니다. isRdma를 true로 설정하고 추가로 needVhostNet을 true로 설정하여 Fast Datapath DPDK 애플리케이션에서 사용할 Mellanox NIC를 구성합니다.**



참고

**vfio-pci 드라이버 유형은 지원되지 않습니다.**

2.

**SriovNetworkNodePolicy 오브젝트를 생성합니다.**

```
$ oc create -f policyoneflag-sriov-node-network.yaml
```

구성 업데이트를 적용하면 sriov-network-operator 네임스페이스의 모든 Pod가 실행 중 상태로 변경됩니다.

3.

**SR-IOV 네트워크 장치**가 구성되어 있는지 확인하려면 다음 명령을 입력합니다.  
**<node\_name>**을 방금 구성한 **SR-IOV 네트워크 장치**가 있는 노드 이름으로 바꿉니다.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

출력 예

Succeeded

### 20.8.2.2. SR-IOV 네트워크에서 sysctl 구성

**SriovNetwork** 리소스의 선택적 **metaPlugins** 매개변수에 튜닝 구성을 추가하여 **SR-IOV**에서 생성한 가상 인터페이스에서 인터페이스별 **sysctl** 설정을 설정할 수 있습니다.

**SR-IOV Network Operator**는 추가 네트워크 정의를 관리합니다. 생성할 추가 **SR-IOV** 네트워크를 지정하면 **SR-IOV Network Operator**가 **NetworkAttachmentDefinition CR**(사용자 정의 리소스)을 자동으로 생성합니다.



참고

**SR-IOV Network Operator**가 관리하는 **NetworkAttachmentDefinition** 사용자 정의 리소스를 편집하지 마십시오. 편집하면 추가 네트워크의 네트워크 트래픽이 중단될 수 있습니다.

인터페이스 수준 네트워크 **net.ipv4.conf.IFNAME.accept\_redirects sysctl** 설정을 변경하려면 **CNI(Container Network Interface)** 튜닝 플러그인을 사용하여 추가 **SR-IOV** 네트워크를 생성합니다.

사전 요구 사항

- **OpenShift Container Platform CLI, oc**를 설치합니다.
- **cluster-admin** 역할의 사용자로 **OpenShift Container Platform** 클러스터에 로그인합니다.



## 프로세스

1.

추가 SR-IOV 네트워크 연결에 대한 **SriovNetwork CR**(사용자 정의 리소스)을 생성하고 다음 예제 CR과 같이 **metaPlugins** 구성을 삽입합니다. **YAML**을 **sriov-network-interface-sysctl.yaml** 파일로 저장합니다.

```
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

오브젝트의 이름입니다. **SR-IOV Network Operator**는 동일한 이름으로 **NetworkAttachmentDefinition** 오브젝트를 생성합니다.

2

**SR-IOV Network Operator**가 설치된 네임스페이스입니다.

3

이 추가 네트워크에 대한 **SR-IOV** 하드웨어를 정의하는 **SriovNetworkNodePolicy** 오브젝트의 **spec.resourceName** 매개변수 값입니다.

4

**SriovNetwork** 오브젝트의 대상 네임스페이스입니다. 대상 네임스페이스의 포트만 추가 네트워크에 연결할 수 있습니다.

5

**YAML 블록 스칼라 IPAM CNI 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.**

6

**선택 사항:** 추가 네트워크의 기능을 설정합니다. `{"ips": true}` 를 지정하여 IP 주소 지원을 활성화하거나 `{"mac": true}` 를 지정하여 MAC 주소 지원을 활성화할 수 있습니다.

7

**선택 사항:** `metaPlugins` 매개 변수는 장치에 추가 기능을 추가하는 데 사용됩니다. 이 사용 사례에서는 유형 필드를 튜닝 으로 설정합니다. `sysctl` 필드에 설정할 인터페이스 수준 네트워크 `sysctl` 을 지정합니다.

2.

**SriovNetwork 리소스를 생성합니다.**

```
$ oc create -f sriov-network-interface-sysctl.yaml
```

**NetworkAttachmentDefinition CR이 성공적으로 생성되었는지 확인**

- 

**SR-IOV Network Operator가 다음 명령을 실행 하여 NetworkAttachmentDefinition CR 을 생성했는지 확인합니다.**

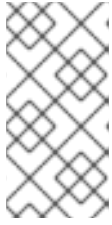
```
$ oc get network-attachment-definitions -n <namespace> 1
```

1

**& lt;namespace >를 SriovNetwork 오브젝트에 지정한 networkNamespace 의 값으 로 바꿉니다. 예를 들면 `sysctl-tuning-test` 입니다.**

출력 예

NAME	AGE
onevalidflag	14m



참고

**SR-IOV Network Operator가 CR을 생성하기 전에 지연이 발생할 수 있습니다.**

추가 **SR-IOV** 네트워크 연결에 성공했는지 확인

튜닝 **CNI**가 올바르게 구성되어 추가 **SR-IOV** 네트워크 연결이 연결되었는지 확인하려면 다음을 수행하십시오.

1.

**Pod CR**을 생성합니다. 다음 **YAML**을 **examplepod.yaml** 파일로 저장합니다.

```

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

**SR-IOV** 네트워크 연결 정의 **CR**의 이름입니다.

②

3

선택사항: **SR-IOV** 네트워크 연결 정의 **CR**에 정의된 리소스 유형에서 할당된 **SR-IOV** 장치의 **IP** 주소입니다. **IPv4** 및 **IPv6** 주소가 모두 지원됩니다. 이 기능을 사용하려면 **SriovNetwork** 오브젝트에 { **"ips": true** }도 지정해야 합니다.

2.

**Pod CR**을 생성합니다.

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

3.

다음 명령을 실행하여 **Pod**가 생성되었는지 확인합니다.

```
$ oc get pod -n sysctl-tuning-test
```

출력 예

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

4.

다음 명령을 실행하여 **Pod**에 로그인합니다.

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5.

구성된 **sysctl** 플래그 값을 확인합니다. 다음 명령을 실행하여 **net.ipv4.conf.IFNAME.accept\_redirects** 값을 찾습니다.

```
$ sysctl net.ipv4.conf.net1.accept_redirects
```

출력 예

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

### 20.8.3. 결합된 SR-IOV 인터페이스 플러그와 연결된 Pod의 sysctl 설정 구성

결합된 SR-IOV 네트워크 장치에 연결된 Pod의 인터페이스 수준 네트워크 sysctl 설정을 설정할 수 있습니다.

이 예에서 구성할 수 있는 특정 네트워크 인터페이스 수준의 sysctl 설정은 본딩된 인터페이스에 설정됩니다.

**sysctl-tuning-test** 는 이 예제에서 사용되는 네임스페이스입니다.

- 

다음 명령을 사용하여 **sysctl-tuning-test** 네임스페이스를 생성합니다.

```
$ oc create namespace sysctl-tuning-test
```

#### 20.8.3.1. 결합된 SR-IOV 네트워크 장치를 사용하여 노드에서 모든 sysctl 플러그 설정

**SR-IOV Network Operator**는 **SriovNetworkNodePolicy.sriovnetwork.openshift.io CRD**(사용자 정의 리소스 정의)를 **OpenShift Container Platform**에 추가합니다. **SriovNetworkNodePolicy CR**(사용자 정의 리소스)을 생성하여 **SR-IOV** 네트워크 장치를 구성할 수 있습니다.



참고

**SriovNetworkNodePolicy** 오브젝트에 지정된 구성을 적용하면 **SR-IOV Operator**가 노드를 비우고 경우에 따라 노드를 재부팅할 수 있습니다.

구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다.

**SriovNetworkNodePolicy CR**(사용자 정의 리소스)을 생성하려면 다음 절차를 따르십시오.

프로세스

- 1.

**SriovNetworkNodePolicy CR**(사용자 정의 리소스)을 생성합니다. 다음 **YAML**을 **policyallflags-sriov-node-network.yaml** 파일로 저장합니다. **policyallflags** 를 구성 이름으로

바꿉니다.

```

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

사용자 정의 리소스 오브젝트의 이름입니다.

2

SR-IOV Network Operator가 설치된 네임스페이스입니다.

3

SR-IOV 네트워크 장치 플러그인의 리소스 이름입니다. 리소스 이름에 대한 SR-IOV 네트워크 노드 정책을 여러 개 생성할 수 있습니다.

4

노드 선택기는 구성할 노드를 지정합니다. 선택한 노드의 SR-IOV 네트워크 장치만 구성됩니다. SR-IOV CNI(Container Network Interface) 플러그인 및 장치 플러그인은 선택한 노드에만 배포됩니다.

5

선택 사항: 우선순위는 0에서 99 사이의 정수 값입니다. 작은 값은 우선순위가 높습니다. 예를 들어 우선순위 10은 우선순위 99보다 높습니다. 기본값은 99입니다.

6

SR-IOV 물리적 네트워크 장치에 생성할 VF(가상 기능) 수입니다. Intel NIC(Network Interface Controller)의 경우 VF 수는 장치에서 지원하는 총 VF보다 클 수 없습니다. Mellanox NIC의 경우 VF 수는 128보다 클 수 없습니다.

7

**NIC** 선택기는 **Operator**가 구성할 장치를 식별합니다. 모든 매개변수에 값을 지정할 필요는 없습니다. 실수로 장치를 선택하지 않도록 네트워크 장치를 정확하게 파악하는 것이 좋습니다. **rootDevices**를 지정하면 **vendor**, **deviceID** 또는 **pfNames**의 값도 지정해야 합니다. **pfNames**와 **rootDevices**를 동시에 지정하는 경우 동일한 장치를 참조하는지 확인하십시오. **netFilter**의 값을 지정하는 경우 네트워크 ID가 고유하므로 다른 매개변수를 지정할 필요가 없습니다.

8

선택사항: 장치에 대해 하나 이상의 물리적 기능(PF) 이름으로 구성된 배열입니다.

9

선택사항: 가상 기능의 드라이버 유형입니다. 허용되는 유일한 값은 **netdevice**입니다. 베어 메탈 노드에서 Mellanox NIC가 DPDK 모드에서 작동하려면 **isRdma**를 **true**로 설정합니다.

10

선택 사항: 원격 직접 메모리 액세스(RDMA) 모드를 활성화할지 여부를 구성합니다. 기본값은 **false**입니다. **isRdma** 매개변수가 **true**로 설정된 경우 RDMA 사용 VF를 일반 네트워크 장치로 계속 사용할 수 있습니다. 어느 모드에서나 장치를 사용할 수 있습니다. **isRdma**를 **true**로 설정하고 추가로 **needVhostNet**을 **true**로 설정하여 Fast Datapath DPDK 애플리케이션에서 사용할 Mellanox NIC를 구성합니다.



참고

**vfiopci** 드라이버 유형은 지원되지 않습니다.

2.

**SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f policyallflags-sriov-node-network.yaml
```

구성 업데이트를 적용하면 **sriov-network-operator** 네임스페이스의 모든 Pod가 실행 중 상태로 변경됩니다.

3.

**SR-IOV** 네트워크 장치가 구성되어 있는지 확인하려면 다음 명령을 입력합니다. **<node\_name>**을 방금 구성한 **SR-IOV** 네트워크 장치가 있는 노드 이름으로 바꿉니다.

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

출력 예

Succeeded

### 20.8.3.2. 결합된 SR-IOV 네트워크에서 sysctl 구성

두 SR-IOV 인터페이스에서 생성된 결합된 인터페이스에서 인터페이스별 **sysctl** 설정을 설정할 수 있습니다. 본딩 네트워크 연결 정의의 선택적 **Plugins** 매개변수에 튜닝 구성을 추가하여 이 작업을 수행합니다.



참고

**SR-IOV Network Operator**가 관리하는 **NetworkAttachmentDefinition** 사용자 정의 리소스를 편집하지 마십시오. 편집하면 추가 네트워크의 네트워크 트래픽이 중단될 수 있습니다.

특정 인터페이스 수준 네트워크 **sysctl** 설정을 변경하려면 다음 절차를 사용하여 **CNI(Container Network Interface)** 튜닝 플러그인을 사용하여 **SriovNetwork CR**(사용자 정의 리소스)을 생성합니다.

사전 요구 사항

- **OpenShift Container Platform CLI, oc**를 설치합니다.
- **cluster-admin** 역할의 사용자로 **OpenShift Container Platform** 클러스터에 로그인합니다.

프로세스

1. 다음 예제 **CR**과 같이 결합된 인터페이스에 대한 **SriovNetwork CR**(사용자 정의 리소스)을 생성합니다. **YAML**을 **sriov-network-attachment.yaml** 파일로 저장합니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
```



```

metadata:
  name: allvalidflags ❶
  namespace: openshift-sriov-network-operator ❷
spec:
  resourceName: policyallflags ❸
  networkNamespace: sysctl-tuning-test ❹
  capabilities: '{ "mac": true, "ips": true }' ❺

```

❶

오브젝트의 이름입니다. **SR-IOV Network Operator**는 동일한 이름으로 **NetworkAttachmentDefinition** 오브젝트를 생성합니다.

❷

**SR-IOV Network Operator**가 설치된 네임스페이스입니다.

❸

이 추가 네트워크에 대한 **SR-IOV** 하드웨어를 정의하는 **SriovNetworkNodePolicy** 오브젝트의 **spec.resourceName** 매개변수 값입니다.

❹

**SriovNetwork** 오브젝트의 대상 네임스페이스입니다. 대상 네임스페이스의 포드만 추가 네트워크에 연결할 수 있습니다.

❺

선택사항: 이 추가 네트워크에 구성할 수 있는 기능입니다. **"{"ips": true}"** 를 지정하여 **IP** 주소 지원을 활성화하거나 **"{"mac": true}"**를 지정하여 **MAC** 주소 지원을 활성화할 수 있습니다.

2.

**SriovNetwork** 리소스를 생성합니다.

```
$ oc create -f sriov-network-attachment.yaml
```

3.

다음 예제 **CR**에서와 같이 본딩 네트워크 연결 정의를 생성합니다. **YAML**을 **sriov-bond-network-interface.yaml** 파일로 저장합니다.

```

apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-sysctl-network
  namespace: sysctl-tuning-test

```

```

spec:
  config: '{
    "cniVersion":"0.4.0",
    "name":"bond-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

**type**은 **bond** 입니다.

②

**mode** 속성은 본딩 모드를 지정합니다. 지원되는 본딩 모드는 다음과 같습니다.

•

**balance-rr - 0**

- **active-backup - 1**

- **balance-xor - 2**

**balance-rr** 또는 **balance-xor** 모드의 경우 **SR-IOV** 가상 기능에 대해 신뢰 모드를 **on** 으로 설정해야 합니다.

3

장애 조치(**failover**) 속성은 **active-backup** 모드에 필요합니다.

4

**linksInContainer=true** 플래그는 **Bond CNI**에 컨테이너 내부에 필요한 인터페이스가 있음을 알립니다. 기본적으로 **Bond CNI**는 **SRIOV** 및 **Multus**와의 통합에 작동하지 않는 호스트에서 이러한 인터페이스를 찾습니다.

5

**links** 섹션에서는 본딩을 만드는 데 사용할 인터페이스를 정의합니다. 기본적으로 **Multus**는 연결된 인터페이스의 이름을 "**net**"과 함께 1부터 시작하여 연속 숫자로 지정합니다.

6

**YAML** 블록 스칼라 **IPAM CNI** 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 **IP** 주소 할당을 관리합니다. 이 **Pod** 예제 **IP** 주소는 수동으로 구성되므로 이 경우 **ipam** 이 **static**으로 설정됩니다.

7

장치에 추가 기능을 추가합니다. 예를 들어 **type** 필드를 튜닝 으로 설정합니다. **sysctl** 필드에 설정할 인터페이스 수준 네트워크 **sysctl** 을 지정합니다. 이 예에서는 설정할 수 있는 모든 인터페이스 수준 네트워크 **sysctl** 설정을 설정합니다.

4.

본딩 네트워크 연결 리소스를 생성합니다.

```
$ oc create -f sriov-bond-network-interface.yaml
```

**NetworkAttachmentDefinition CR**이 성공적으로 생성되었는지 확인

•

**SR-IOV Network Operator**가 다음 명령을 실행 하여 **NetworkAttachmentDefinition CR**

을 생성했는지 확인합니다.

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1

< namespace >를 네트워크 연결을 구성할 때 지정한 `networkNamespace`로 변경합니다(예: `sysctl-tuning-test` ).

출력 예

NAME	AGE
<code>bond-sysctl-network</code>	22m
<code>allvalidflags</code>	47m



참고

**SR-IOV Network Operator**가 CR을 생성하기 전에 지연이 발생할 수 있습니다.

추가 **SR-IOV** 네트워크 리소스가 성공했는지 확인

튜닝 **CNI**가 올바르게 구성되어 추가 **SR-IOV** 네트워크 연결이 연결되었는지 확인하려면 다음을 수행하십시오.

1.

**Pod CR**을 생성합니다. 예를 들어 다음 **YAML**을 `examplepod.yaml` 파일로 저장합니다.

```
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

SR-IOV 네트워크 연결 정의 CR의 이름입니다.

2

선택사항: SR-IOV 네트워크 연결 정의 CR에 정의된 리소스 유형에서 할당된 SR-IOV 장치의 MAC 주소입니다. 이 기능을 사용하려면 SrioNetwork 오브젝트에 { "mac": true } 도 지정해야 합니다.

3

선택사항: SR-IOV 네트워크 연결 정의 CR에 정의된 리소스 유형에서 할당된 SR-IOV 장치의 IP 주소입니다. IPv4 및 IPv6 주소가 모두 지원됩니다. 이 기능을 사용하려면 SrioNetwork 오브젝트에 { "ips": true }도 지정해야 합니다.

2.

YAML을 적용합니다.

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

3.

다음 명령을 실행하여 Pod가 생성되었는지 확인합니다.

```
$ oc get pod -n sysctl-tuning-test
```

출력 예

```

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

```

4.

다음 명령을 실행하여 Pod에 로그인합니다.

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5.

구성된 `sysctl` 플래그 값을 확인합니다. 다음 명령을 실행하여 `net.ipv6.neigh.IFNAME.base_reachable_time_ms` 값을 찾습니다.

```
$ sysctl net.ipv6.neigh.bond0.base_reachable_time_ms
```

출력 예

```
net.ipv6.neigh.bond0.base_reachable_time_ms = 20000
```

## 20.9. 고성능 멀티 캐스트 사용

**SR-IOV(Single Root I/O Virtualization)** 하드웨어 네트워크에서 멀티 캐스트를 사용할 수 있습니다.

### 20.9.1. 고성능 멀티 캐스트

**OpenShift SDN 기본 CNI(Container Network Interfac)** 네트워크 공급자는 기본 네트워크에서 Pod 간 멀티 캐스트를 지원합니다. 이는 고 대역폭 애플리케이션이 아닌 저 대역폭 조정 또는 서비스 검색에 가장 적합합니다. **IPTV(Internet Protocol Television)** 및 멀티 포인트 화상 회의와 같은 스트리밍 미디어와 같은 애플리케이션의 경우 **SR-IOV(Single Root I/O Virtualization)** 하드웨어를 사용하여 거의 네이티브와 같은 성능을 제공할 수 있습니다.

멀티 캐스트에 추가 **SR-IOV** 인터페이스를 사용하는 경우:

- 멀티 캐스트 패키지는 추가 **SR-IOV** 인터페이스를 통해 **pod**에서 보내거나 받아야 합니다.
- **SR-IOV** 인터페이스를 연결하는 물리적 네트워크는 멀티 캐스트 라우팅 및 토폴로지를 결정하며 **OpenShift Container Platform**에서 제어하지 않습니다.

### 20.9.2. 멀티 캐스트를 위한 **SR-IOV** 인터페이스 구성

다음 프로시저는 멀티 캐스트용 **SR-IOV** 인터페이스 예제를 만듭니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

#### 프로세스

1. **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```

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. **SriovNetwork** 오브젝트를 생성합니다.

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator

```

```
spec:
  networkNamespace: default
  ipam: | ❶
    {
      "type": "host-local", ❷
      "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
```

❶ ❷

DHCP를 IPAM으로 구성하도록 선택한 경우 DHCP 서버를 통해 224.0.0.0/5 및 232.0.0.0/5 기본 경로를 프로비저닝해야 합니다. 이는 기본 네트워크 공급자가 설정한 정적 멀티 캐스트 경로를 재정의하는 것입니다.

3.

멀티 캐스트 애플리케이션으로 pod를 생성합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:
  - name: example
    image: rhel7:latest
    securityContext:
      capabilities:
        add: ["NET_ADMIN"] ❶
    command: [ "sleep", "infinity"]
```

❶

NET\_ADMIN 기능은 애플리케이션이 멀티 캐스트 IP 주소를 SR-IOV 인터페이스에 할당해야 하는 경우에만 필요합니다. 그 밖의 경우에는 생략할 수 있습니다.

## 20.10. DPDK 및 RDMA 사용

컨테이너화된 DPDK(Data Plane Development Kit) 애플리케이션은 OpenShift Container Platform



에서 지원됩니다. DPDK(Data Plane Development Kit) 및 RDMA(Remote Direct Memory Access)와 함께 SR-IOV(Single Root I/O Virtualization) 네트워크 하드웨어를 사용할 수 있습니다.

지원되는 장치에 대한 자세한 내용은 [지원되는 장치를](#) 참조하십시오.

### 20.10.1. Intel NIC와 함께 DPDK 모드에서 가상 기능 사용

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **SR-IOV Network Operator** 설치.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

1. 다음 **SriovNetworkNodePolicy** 오브젝트를 생성한 다음 **YAML**을 **intel-dpdk-node-policy.yaml** 파일에 저장합니다.

```
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

가상 기능의 드라이버 유형을 **vfio-pci**로 지정합니다.



## 참고

**SriovNetworkNodePolicy**의 각 옵션에 대한 자세한 설명은 **Configuring SR-IOV network devices** 섹션을 참조하십시오.

**SriovNetworkNodePolicy** 오브젝트에 지정된 구성을 적용하면 **SR-IOV Operator**가 노드를 비우고 경우에 따라 노드를 재부팅할 수 있습니다. 구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다. 제거된 워크로드를 사전에 처리하는 데 클러스터에 사용 가능한 노드가 충분한지 확인하십시오.

구성 업데이트가 적용되면 **openshift-sriov-network-operator** 네임스페이스의 모든 **Pod** 상태가 **Running**으로 변경됩니다.

2.

다음 명령을 실행하여 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f intel-dpdk-node-policy.yaml
```

3.

다음 **SriovNetwork** 오브젝트를 생성한 다음 **YAML**을 **intel-dpdk-network.yaml** 파일에 저장합니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |-
# ... 1
  vlan: <vlan>
  resourceName: intelnic
```

1

**ipam CNI** 플러그인의 구성 오브젝트를 **YAML** 블록 스칼라로 지정합니다. 플러그인은 연결 정의에 대한 **IP** 주소 할당을 관리합니다.



## 참고

**SriovNetwork**의 각 옵션에 대한 자세한 설명은 "**SR-IOV 추가 네트워크 구성**" 섹션을 참조하십시오.

선택적 라이브러리인 **app-netutil**은 컨테이너의 상위 **pod**에 대한 네트워크 정보를 수집하기 위한 여러 **API** 메시지를 제공합니다.

4.

다음 명령을 실행하여 **SriovNetwork** 오브젝트를 생성합니다.

```
$ oc create -f intel-dpdk-network.yaml
```

5.

다음 **Pod** 사양을 생성한 다음 **YAML**을 **intel-dpdk-pod.yaml** 파일에 저장합니다.

```
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: /dev/hugepages 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

**SriovNetwork** 오브젝트 **intel-dpdk-network**가 생성되는 동일한 **target\_namespace**를 지정합니다. 다른 네임스페이스에서 포드를 생성하려면 **Pod** 사양과

**SriovNetowrk** 오브젝트 모두에서 **target\_namespace**를 변경합니다.

2

애플리케이션 및 애플리케이션이 사용하는 **DPDK** 라이브러리를 포함하는 **DPDK** 이미지를 지정합니다.

3

**hugepage** 할당, 시스템 리소스 할당 및 네트워크 인터페이스 액세스를 위해 컨테이너 내부의 애플리케이션에 필요한 추가 기능을 지정합니다.

4

**/dev/hugepages** 아래 **DPDK pod**에 **hugepage** 볼륨을 마운트합니다. **hugepage** 볼륨은 매체가 **Hugepages**인 **emptyDir** 볼륨 유형으로 지원됩니다.

5

**선택사항:** **DPDK Pod**에 할당된 **DPDK** 장치 수를 지정합니다. 명시적으로 지정되지 않은 경우 이 리소스 요청 및 제한은 **SR-IOV** 네트워크 리소스 인젝터에 의해 자동으로 추가됩니다. **SR-IOV** 네트워크 리소스 인젝터는 **SR-IOV Operator**에서 관리하는 승인 컨트롤러 구성 요소입니다. 기본적으로 활성화되어 있으며 기본 **SriovOperatorConfig CR**에서 **enableInjector** 옵션을 **false**로 설정하여 비활성화할 수 있습니다.

6

**CPU** 수를 지정합니다. **DPDK pod**는 일반적으로 **kubelet**에서 배타적 **CPU**를 할당해야 합니다. 이를 위해 **CPU** 관리자 정책을 **static**으로 설정하고 **QoS**가 **Guaranteed Pod**를 생성합니다.

7

**hugepage** 크기 **hugepages-1Gi** 또는 **hugepages-2Mi**를 지정하고 **DPDK Pod**에 할당할 **hugepage** 수량을 지정합니다. **2Mi** 및 **1Gi hugepage**를 별도로 구성합니다. **1Gi hugepage**를 구성하려면 커널 인수를 노드에 추가해야 합니다. 예를 들어, 커널 인수 **default\_hugepagesz = 1GB**, **hugepagesz = 1G** 및 **hugepages = 16**을 추가하면 시스템 부팅 시 **16 \* 1Gi hugepage**가 할당됩니다.

6.

다음 명령을 실행하여 **DPDK Pod**를 생성합니다.

```
$ oc create -f intel-dpdk-pod.yaml
```

### 20.10.2. Mellanox NIC와 함께 DPDK 모드에서 가상 기능 사용

Mellanox NIC와 함께 DPDK 모드의 가상 기능을 사용하여 네트워크 노드 정책을 생성하고 DPDK(Data Plane Development Kit) Pod를 생성할 수 있습니다.

#### 사전 요구 사항

- OpenShift CLI(oc)가 설치되어 있습니다.
- SR-IOV(Single Root I/O Virtualization) Network Operator를 설치했습니다.
- cluster-admin 권한이 있는 사용자로 로그인했습니다.

#### 프로세스

1. 다음 SrioNetworkNodePolicy YAML 구성을 `mlx-dpdk-node-policy.yaml` 파일에 저장합니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SrioNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnic
  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

SR-IOV 네트워크 장치의 장치 16진수 코드를 지정합니다.

2

netdevice에 가상 기능의 드라이버 유형을 지정합니다. Mellanox SR-IOV VF(가상 기능)는 vfio-pci 장치 유형을 사용하지 않고도 DPDK 모드에서 작동할 수 있습니다. VF 장치는 컨테이너 내부에 커널 네트워크 인터페이스로 나타납니다.

3

**RDMA(Remote Direct Memory Access) 모드를 활성화합니다. Mellanox 카드가 DPDK 모드에서 작동하려면 이 카드가 필요합니다.**



참고

**SriovNetworkNodePolicy** 오브젝트의 각 옵션에 대한 자세한 설명은 **SR-IOV** 네트워크 장치 구성을 참조하십시오.

**SriovNetworkNodePolicy** 오브젝트에 지정된 구성을 적용하면 **SR-IOV Operator**가 노드를 비우고 경우에 따라 노드를 재부팅할 수 있습니다. 구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다. 제거된 워크로드를 사전에 처리하는 데 클러스터에 사용 가능한 노드가 충분한지 확인하십시오.

구성 업데이트가 적용되면 **openshift-sriov-network-operator** 네임스페이스의 모든 **Pod** 상태가 **Running**으로 변경됩니다.

2.

다음 명령을 실행하여 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f mlx-dpdk-node-policy.yaml
```

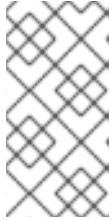
3.

다음 **SriovNetwork** **YAML** 구성을 **mlx-dpdk-network.yaml** 파일에 저장합니다.

```
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
```

1

**IP 주소 관리(IPAM) CNI(Container Network Interface) 플러그인**의 구성 오브젝트를 **YAML** 블록 스칼라로 지정합니다. 플러그인은 연결 정의에 대한 **IP** 주소 할당을 관리합니다.



## 참고

**SriovNetwork** 오브젝트의 각 옵션에 대한 자세한 설명은 **SR-IOV** 네트워크 장치 구성을 참조하십시오.

**app-netutil** 옵션 라이브러리는 컨테이너의 상위 포드에 대한 네트워크 정보를 수집하기 위한 여러 API 메서드를 제공합니다.

4.

다음 명령을 실행하여 **SriovNetwork** 오브젝트를 생성합니다.

```
$ oc create -f mlx-dpdk-network.yaml
```

5.

다음 Pod YAML 구성을 **mlx-dpdk-pod.yaml** 파일에 저장합니다.

```
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: testpmd
    image: <DPDK_image> ②
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"] ③
    volumeMounts:
      - mountPath: /dev/hugepages ④
        name: hugepage
    resources:
      limits:
        openshift.io/mlxnics: "1" ⑤
        memory: "1Gi"
        cpu: "4" ⑥
        hugepages-1Gi: "4Gi" ⑦
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
      command: ["sleep", "infinity"]
  volumes:
```

```
- name: hugepage
  emptyDir:
    medium: HugePages
```

1

**SriovNetwork** 오브젝트 **mlx-dpdk-network**가 생성되는 동일한 **target\_namespace**를 지정합니다. 다른 네임스페이스에서 **Pod**를 생성하려면 **Pod** 사양 및 **SriovNetwork** 오브젝트 모두에서 **target\_namespace**를 변경합니다.

2

애플리케이션 및 애플리케이션에서 사용하는 **DPDK** 라이브러리를 포함하는 **DPDK** 이미지를 지정합니다.

3

**hugepage** 할당, 시스템 리소스 할당 및 네트워크 인터페이스 액세스를 위해 컨테이너 내부의 애플리케이션에 필요한 추가 기능을 지정합니다.

4

**hugepage** 볼륨을 **/dev/hugepages** 아래의 **DPDK Pod**에 마운트합니다. **hugepage** 볼륨은 매체가 **Hugepages**인 **emptyDir** 볼륨 유형으로 지원됩니다.

5

선택 사항: **DPDK Pod**에 할당된 **DPDK** 장치 수를 지정합니다. 명시적으로 지정하지 않으면 **SR-IOV** 네트워크 리소스 인젝터에 의해 이 리소스 요청 및 제한이 자동으로 추가됩니다. **SR-IOV** 네트워크 리소스 인젝터는 **SR-IOV Operator**에서 관리하는 승인 컨트롤러 구성 요소입니다. 기본적으로 활성화되어 있으며 기본 **SriovOperatorConfig CR**에서 **enableInjector** 옵션을 **false**로 설정하여 비활성화할 수 있습니다.

6

**CPU** 수를 지정합니다. **DPDK Pod**는 일반적으로 **kubelet**에서 전용 **CPU**를 할당해야 합니다. 이렇게 하려면 **CPU** 관리자 정책을 **static**으로 설정하고 **QoS(Quality of Service)**가 보장된 **Pod**를 생성합니다.

7

**hugepage** 크기 **hugepages-1Gi** 또는 **hugepages-2Mi**를 지정하고 **DPDK Pod**에 할당할 **hugepage** 수량을 지정합니다. **2Mi** 및 **1Gi hugepage**를 별도로 구성합니다. **1Gi hugepages**를 구성하려면 커널 인수를 노드에 추가해야 합니다.

6.

다음 명령을 실행하여 **DPDK Pod**를 생성합니다.

■



```
$ oc create -f mlx-dpdk-pod.yaml
```

### 20.10.3. 특정 DPDK 라인 비율 달성 개요

특정 DPDK(Data Plane Development Kit) 라인 속도를 달성하려면 **Node Tuning Operator**를 배포하고 **SR-IOV(Single Root I/O Virtualization)**를 구성합니다. 다음 리소스의 DPDK 설정도 조정해야 합니다.

- 분리된 CPU
- hugepages
- 토폴로지 스케줄러

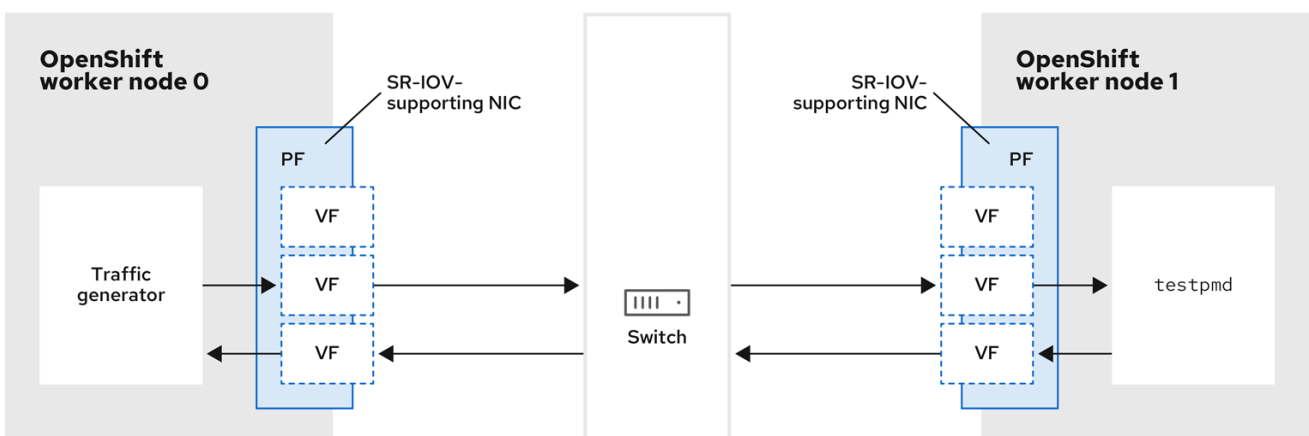


#### 참고

이전 버전의 **OpenShift Container Platform**에서는 **OpenShift Container Platform** 애플리케이션에 대해 짧은 대기 시간 성능을 실현하기 위해 자동 튜닝을 구현하는 데 **Performance Addon Operator**를 사용했습니다. **OpenShift Container Platform 4.11** 이상에서는 이 기능이 **Node Tuning Operator**의 일부입니다.

### DPDK 테스트 환경

다음 다이어그램은 트래픽 테스트 환경의 구성 요소를 보여줍니다.



261\_OpenShift\_0722

- **트래픽 생성기:** 대량의 패킷 트래픽을 생성할 수 있는 애플리케이션입니다.
- **SR-IOV 지원 NIC:** SR-IOV와 호환되는 네트워크 인터페이스 카드입니다. 이 카드는 물리적 인터페이스에서 여러 가상 기능을 실행합니다.
- **PF(물리적 기능):** SR-IOV 인터페이스를 지원하는 네트워크 어댑터의 **PCI Express(PCI Express)** 기능입니다.
- **VF(가상 기능):** SR-IOV를 지원하는 네트워크 어댑터의 경량 **DASD** 기능입니다. VF는 네트워크 어댑터의 **controlPlane PF**와 연결되어 있습니다. VF는 네트워크 어댑터의 가상화된 인스턴스를 나타냅니다.
- **Switch:** 네트워크 스위치입니다. 또한 노드는 **back-to-back**으로 연결할 수 있습니다.
- **testpmd:** DPDK에 포함된 예제 애플리케이션입니다. testpmd 애플리케이션을 사용하여 **packet-forwarding** 모드에서 DPDK를 테스트할 수 있습니다. testpmd 애플리케이션은 DPDK 소프트웨어 개발 키트(SDK)를 사용하여 완전한 애플리케이션을 빌드하는 방법의 예입니다.
- **worker 0 및 worker 1:** OpenShift Container Platform 노드

#### 20.10.4. SR-IOV 및 Node Tuning Operator를 사용하여 DPDK 라인 속도 달성

Node Tuning Operator를 사용하여 분리된 CPU, hugepages 및 토폴로지 스케줄러를 구성할 수 있습니다. 그런 다음 Node Tuning Operator를 SR-IOV(Single Root I/O Virtualization)와 함께 사용하여 특정 DPDK(Data Plane Development Kit) 라인 속도를 달성할 수 있습니다.

##### 사전 요구 사항

- OpenShift CLI(oc)가 설치되어 있습니다.
- SR-IOV Network Operator가 설치되어 있습니다.
- cluster-admin 권한이 있는 사용자로 로그인했습니다.

독립 실행형 **Node Tuning Operator**를 배포했습니다.



#### 참고

이전 버전의 **OpenShift Container Platform**에서는 **OpenShift** 애플리케이션에 대해 짧은 대기 시간 성능을 실현하기 위해 자동 튜닝을 구현하는 데 **Performance Addon Operator**를 사용했습니다. **OpenShift Container Platform 4.11** 이상에서는 이 기능이 **Node Tuning Operator**의 일부입니다.

#### 프로세스

1.

다음 예제를 기반으로 **PerformanceProfile** 오브젝트를 생성합니다.

```
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

시스템에서 하이퍼 스레딩이 활성화되면 관련 심볼릭 링크를 분리 및 예약된 CPU 그룹에 할당합니다. 시스템에 NUMA(Non-uniform 메모리 액세스 노드)가 여러 개 포함된 경우 두 NUMA에서 두 그룹으로 CPU를 할당합니다. 이 작업에 **Performance Profile Creator**를 사용할 수도 있습니다. 자세한 내용은 성능 프로파일 생성을 참조하십시오.

2

대기열이 예약된 CPU 수로 설정될 장치 목록을 지정할 수도 있습니다. 자세한 내용은 **Node Tuning Operator**를 사용하여 NIC 대기열 단축을 참조하십시오.

3

2. **yaml** 파일을 **mlx-dpdk-perfprofile-policy.yaml** 로 저장합니다.
3. 다음 명령을 사용하여 성능 프로필을 적용합니다.

```
$ oc create -f mlx-dpdk-perfprofile-policy.yaml
```

#### 20.10.4.1. 가상 기능을 위한 SR-IOV Network Operator의 예

**SR-IOV(Single Root I/O Virtualization) Network Operator**를 사용하여 노드의 **SR-IOV** 지원 물리적 기능 NIC에서 **VF**(가상 기능)를 할당하고 구성할 수 있습니다.

**Operator** 배포에 대한 자세한 내용은 **SR-IOV Network Operator** 설치를 참조하십시오. **SR-IOV** 네트워크 장치 구성에 대한 자세한 내용은 **SR-IOV** 네트워크 장치 구성을 참조하십시오.

**Intel VF**와 **Mellanox VF**에서 **DPDK(Data Plane Development Kit)** 워크로드 실행 사이에 몇 가지 차이점이 있습니다. 이 섹션에서는 두 **VF** 유형의 오브젝트 구성 예제를 제공합니다. 다음은 **Intel NIC**에서 **DPDK** 애플리케이션을 실행하는 데 사용되는 **sriovNetworkNodePolicy** 오브젝트의 예입니다.

```
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

Intel NIC의 경우 `deviceType` 은 `vfio-pci` 여야 합니다.

2

DPDK 워크로드와의 커널 통신이 필요한 경우 `needVhostNet: true` 를 추가합니다. 그러면 애 플리케이션이 탭 장치를 생성하고 탭 장치를 DPDK 워크로드에 연결할 수 있도록 `/dev/net/tun` 및 `/dev/vhost-net` 장치가 컨테이너에 마운트됩니다.

다음은 Mellanox NIC의 `sriovNetworkNodePolicy` 오브젝트의 예입니다.

```

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

Mellanox 장치의 경우 `deviceType` 은 `netdevice` 여야 합니다.

2

Mellanox 장치의 경우 `isRdma` 가 `true` 여야 합니다. Mellanox 카드는 `Flow Bifurcation`을 사용하여 DPDK 애플리케이션에 연결됩니다. 이 메커니즘은 Linux 사용자 공간과 커널 공간 간에 트래픽을 분할하고 라인 속도 처리 기능을 향상시킬 수 있습니다.

#### 20.10.4.2. SR-IOV 네트워크 Operator의 예

다음은 `sriovNetwork` 오브젝트 정의의 예입니다. 이 경우 Intel 및 Mellanox 구성은 동일합니다.

```
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

`Whereabouts`와 같은 다른 IP 주소 관리 (IPAM) 구현을 사용할 수 있습니다. 자세한 내용은 `Whereabouts`를 사용한 동적 IP 주소 할당 구성을 참조하십시오.

2

네트워크 연결 정의가 생성될 `networkNamespace` 를 요청해야 합니다. `openshift-sriov-network-operator` 네임스페이스에서 `sriovNetwork CR`을 생성해야 합니다.

3

`resourceName` 값은 `sriovNetworkNodePolicy` 에서 생성된 `resourceName` 의 값과 일치해야 합니다.

### 20.10.4.3. DPDK 기본 워크로드의 예

다음은 DPDK(Data Plane Development Kit) 컨테이너의 예입니다.

```

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

필요한 **SR-IOV** 네트워크를 요청합니다. 장치의 리소스가 자동으로 삽입됩니다.

2

**CPU** 및 **IRQ** 로드 밸런싱 기반을 비활성화합니다. 자세한 내용은 개별 **Pod**의 인터럽트 처리 비활성화를 참조하십시오.

3

**runtimeClass** 를 **performance-performance** 로 설정합니다. **runtimeClass** 를 **HostNetwork** 또는 **privileged** 로 설정하지 마십시오.

4

**QoS (Quality of Service)**를 사용하여 **Pod**를 시작하기 위해 요청 및 제한에 대해 동일한 수의 리소스를 요청합니다.





## 참고

**SLEEP** 로 Pod를 시작한 다음 Pod로 실행하여 **testpmd** 또는 **DPDK** 워크로드를 시작합니다. 그러면 **exec** 프로세스가 CPU에 고정되지 않으므로 추가 인터럽트가 추가될 수 있습니다.

## 20.10.4.4. testpmd 스크립트의 예

다음은 **testpmd** 를 실행하는 스크립트의 예입니다.

```
#!/bin/bash
set -ex
export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
echo ${CPU}

dpdk-testpmd -l ${CPU} -a ${PCIDEVICE_OPENSIFT_IO_DPDK_NIC_1} -a
${PCIDEVICE_OPENSIFT_IO_DPDK_NIC_2} -n 4 -- -i --nb-cores=15 --rxd=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
```

이 예에서는 두 개의 다른 **sriovNetwork CR**을 사용합니다. 환경 변수에는 Pod에 할당된 **VF**(가상 기능) **PCI** 주소가 포함됩니다. Pod 정의에서 동일한 네트워크를 사용하는 경우 **pciAddress** 를 분할해야 합니다. 트래픽 생성기의 올바른 **MAC** 주소를 구성하는 것이 중요합니다. 이 예에서는 사용자 정의 **MAC** 주소를 사용합니다.

## 20.10.5. Mellanox NIC와 함께 RDMA 모드에서 가상 기능 사용



## 중요

**RoCE(RDMA over Converged Ethernet)**는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(**SLA**)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

**Red Hat** 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

**OpenShift Container Platform**에서 **RDMA**를 사용할 때 **RoCE(RDMA over Converged Ethernet)**가 지원되는 유일한 모드입니다.

## 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **SR-IOV Network Operator** 설치.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

## 프로세스

1.

다음 **SriovNetworkNodePolicy** 오브젝트를 생성한 다음 **YAML**을 **mlx-rdma-node-policy.yaml** 파일에 저장합니다.

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnic
  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

**SR-IOV** 네트워크 장치의 장치 **16**진수 코드를 지정합니다.

2

**netdevice**에 가상 기능의 드라이버 유형을 지정합니다.

3

**RDMA** 모드를 활성화합니다.



## 참고

**SriovNetworkNodePolicy**의 각 옵션에 대한 자세한 설명은 **Configuring SR-IOV network devices** 섹션을 참조하십시오.

**SriovNetworkNodePolicy** 오브젝트에 지정된 구성을 적용하면 **SR-IOV Operator**가 노드를 비우고 경우에 따라 노드를 재부팅할 수 있습니다. 구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다. 제거된 워크로드를 사전에 처리하는 데 클러스터에 사용 가능한 노드가 충분한지 확인하십시오.

구성 업데이트가 적용되면 **openshift-sriov-network-operator** 네임스페이스의 모든 **Pod** 상태가 **Running**으로 변경됩니다.

2.

다음 명령을 실행하여 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f mlx-rdma-node-policy.yaml
```

3.

다음 **SriovNetwork** 오브젝트를 생성한 다음 **YAML**을 **mlx-rdma-network.yaml** 파일에 저장합니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-rdma-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- ❶
# ...
  vlan: <vlan>
  resourceName: mlxnic
```

❶

**ipam CNI** 플러그인의 구성 오브젝트를 **YAML** 블록 스칼라로 지정합니다. 플러그인은 연결 정의에 대한 **IP** 주소 할당을 관리합니다.



## 참고

**SriovNetwork**의 각 옵션에 대한 자세한 설명은 "**SR-IOV 추가 네트워크 구성**" 섹션을 참조하십시오.

선택적 라이브러리인 **app-netutil**은 컨테이너의 상위 **pod**에 대한 네트워크 정보를 수집하기 위한 여러 **API** 메서드를 제공합니다.

4.

다음 명령을 실행하여 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f mlx-rdma-network.yaml
```

5.

다음 **Pod** 사양을 생성한 다음 **YAML**을 **mlx-rdma-pod.yaml** 파일에 저장합니다.

```
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: /dev/hugepages 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

**SriovNetwork** 오브젝트 **mlx-rdma-network**가 생성되는 동일한 **target\_namespace**를 지정합니다. 다른 네임스페이스에서 포드를 생성하려면 **Pod** 사양과 **SriovNetwork** 오브젝트 모두에서 **target\_namespace**를 변경합니다.

2

애플리케이션 및 애플리케이션에서 사용하는 **RDMA 라이브러리**를 포함하는 **RDMA** 이미지를 지정합니다.

3

**hugepage** 할당, 시스템 리소스 할당 및 네트워크 인터페이스 액세스를 위해 컨테이너 내부의 애플리케이션에 필요한 추가 기능을 지정합니다.

4

**hugepage** 볼륨을 **/dev/hugepages** 아래의 **RDMA Pod**에 마운트합니다. **hugepage** 볼륨은 매체가 **Hugepages**인 **emptyDir** 볼륨 유형으로 지원됩니다.

5

**CPU** 수를 지정합니다. **RDMA Pod**는 일반적으로 **kubelet**에서 전용 **CPU**를 할당해야 합니다. 이를 위해 **CPU** 관리자 정책을 **static**으로 설정하고 **QoS**가 **Guaranteed Pod**를 생성합니다.

6

**hugepage** 크기 **hugepages-1Gi** 또는 **hugepages-2Mi**를 지정하고 **RDMA Pod**에 할당할 **hugepage** 수량을 지정합니다. **2Mi** 및 **1Gi hugepage**를 별도로 구성합니다. **1Gi hugepage**를 구성하려면 커널 인수를 노드에 추가해야 합니다.

6.

다음 명령을 실행하여 **RDMA Pod**를 생성합니다.

```
$ oc create -f mlx-rdma-pod.yaml
```

### 20.10.6. OpenStack에서 OVS-DPDK를 사용하는 클러스터에 대한 테스트 Pod 템플릿

다음 **testpmd pod**는 대규모 페이지, 예약된 **CPU** 및 **SR-IOV** 포트로 컨테이너 생성을 보여줍니다.

예제 **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", "999999"]
    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: /dev/hugepages
      name: hugepage
      readOnly: False
    runtimeClassName: performance-cnf-performanceprofile 2
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages

```

**1**

이 예제의 이름 `dpdk1` 은 사용자가 생성한 `SriovNetworkNodePolicy` 리소스입니다. 이 이름을 생성한 리소스의 해당 이름으로 대체할 수 있습니다.

**2**

성능 프로파일의 이름이 `cnf-performance` 프로파일 인 경우 해당 문자열을 올바른 성능 프로파일 이름으로 바꿉니다.

### 20.10.7. OpenStack에서 OVS 하드웨어 오프로드를 사용하는 클러스터의 테스트 pod 템플릿

다음 `testpmd pod`는 RHOSP(Red Hat OpenStack Platform)에서 OVS(Open vSwitch) 하드웨어 오

프로드를 보여줍니다.

예제 `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: /dev/hugepages
      name: hugepage
      readOnly: False
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages

```

**1**

성능 프로필의 이름이 `cnf-performance` 프로필 인 경우 해당 문자열을 올바른 성능 프로필 이름으로 바꿉니다.

### 20.10.8. 추가 리소스

- [성능 프로파일 작성](#)
- [Node Tuning Operator를 사용하여 NIC 큐 감소](#)
- [실시간 기능이 있는 작업자 프로비저닝](#)
- [SR-IOV Network Operator 설치](#)
- [SR-IOV 네트워크 장치 구성](#)
- [Whereabouts를 사용한 동적 IP 주소 할당 구성](#)
- [개별 pod에 대한 인터럽트 처리 비활성화](#)
- [SR-IOV 이더넷 네트워크 연결 구성](#)
- [app-netutil 라이브러리 는 컨테이너의 상위 포트에 대한 네트워크 정보를 수집하기 위한 여러 API 메서드를 제공합니다.](#)

### 20.11. POD 수준 본딩 사용

Pod 수준의 본딩은 고가용성과 처리량이 필요한 Pod 내부의 워크로드를 활성화하는 데 중요합니다. Pod 수준 본딩을 사용하면 커널 모드 인터페이스에서 여러 개의 단일 루트 I/O 가상화(SR-IOV) 가상 기능 인터페이스에서 본딩 인터페이스를 생성할 수 있습니다. SR-IOV 가상 기능은 Pod에 전달되고 커널 드라이버에 연결됩니다.

Pod 수준 본딩이 필요한 한 가지 시나리오는 다양한 물리적 기능에 여러 SR-IOV 가상 함수에서 본딩 인터페이스를 생성하는 것입니다. 호스트의 서로 다른 물리 함수 두 개에서 본딩 인터페이스를 생성하여 Pod 수준에서 고가용성 및 처리량을 달성할 수 있습니다.

SR-IOV 네트워크, 네트워크 정책, 네트워크 연결 정의 및 Pod 생성과 같은 작업에 대한 지침은 [SR-IOV](#)



네트워크 장치 구성을 참조하십시오.

### 20.11.1. SR-IOV 인터페이스 두 개에서 본딩 인터페이스 구성

본딩을 사용하면 여러 네트워크 인터페이스를 단일 논리 "보딩" 인터페이스로 통합할 수 있습니다. 본딩 컨테이너 네트워크 인터페이스(**Bond-CNI**)는 본딩 기능을 컨테이너에 제공합니다.

**bond-CNI**는 **SR-IOV(Single Root I/O Virtualization)** 가상 기능을 사용하여 생성할 수 있으며 컨테이너 네트워크 네임스페이스에 배치할 수 있습니다.

**OpenShift Container Platform**은 **SR-IOV** 가상 기능을 사용하는 **Bond-CNI**만 지원합니다. **SR-IOV Network Operator**는 가상 기능을 관리하는 데 필요한 **SR-IOV CNI** 플러그인을 제공합니다. 기타 **CNI** 또는 인터페이스 유형은 지원되지 않습니다.

#### 사전 요구 사항

- 컨테이너에서 가상 기능을 가져오도록 **SR-IOV Network Operator**를 설치하고 구성해야 합니다.
- **SR-IOV** 인터페이스를 구성하려면 각 인터페이스에 대해 **SR-IOV** 네트워크 및 정책을 생성해야 합니다.
- **SR-IOV Network Operator**는 정의된 **SR-IOV** 네트워크 및 정책을 기반으로 각 **SR-IOV** 인터페이스에 대한 네트워크 연결 정의를 생성합니다.
- **linkState** 는 **SR-IOV** 가상 기능의 기본값 **auto** 로 설정됩니다.

#### 20.11.1.1. 본딩 네트워크 연결 정의 생성

이제 **SR-IOV** 가상 기능을 사용할 수 있으므로 본딩 네트워크 연결 정의를 생성할 수 있습니다.

```
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"
}
}'

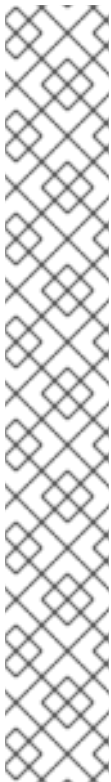
```

1

*cni-type*은 항상 **bond** 로 설정됩니다.

2

*mode* 속성은 본딩 모드를 지정합니다.



참고

지원되는 본딩 모드는 다음과 같습니다.

- **balance-rr - 0**
- **active-backup - 1**
- **balance-xor - 2**

**balance-rr** 또는 **balance-xor** 모드의 경우 **SR-IOV** 가상 기능에 대해 신뢰 모드를 **on** 으로 설정해야 합니다.

3

장애 조치(**failover**) 속성은 **active-backup** 모드의 경우 필수이며 1로 설정해야 합니다.

4

**linksInContainer=true** 플래그는 **Bond CNI**에 컨테이너 내부에 필요한 인터페이스가 있음을 알립니다. 기본적으로 **Bond CNI**는 **SRIOV** 및 **Multus**와의 통합에 작동하지 않는 호스트에서 이러한 인터페이스를 찾습니다.

5

**links** 섹션에서는 본딩을 만드는 데 사용할 인터페이스를 정의합니다. 기본적으로 **Multus**는 연결된 인터페이스의 이름을 "**net**"과 함께 1부터 시작하여 연속 숫자로 지정합니다.

### 20.11.1.2. 본딩 인터페이스를 사용하여 Pod 생성

1.

다음과 유사한 콘텐츠를 사용하여 이름이 **podbonding.yaml** 과 같은 **YAML** 파일로 **Pod**를 생성하여 설정을 테스트합니다.

```
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

네트워크 주석: **SR-IOV** 네트워크 첨부 2개와 본딩 네트워크 연결 1개가 포함되어 있습니다. 본딩 연결에서는 두 개의 **SR-IOV** 인터페이스를 결합된 포트 인터페이스로 사용합니다.

2.

다음 명령을 실행하여 **yaml**을 적용합니다.

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

3.

다음 명령을 사용하여 **Pod** 인터페이스를 검사합니다.

```
$ 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_UP800> 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_UP800> 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

본당 인터페이스의 이름은 자동으로 **net3** 입니다. 특정 인터페이스 이름을 설정하려면 Pod의 **k8s.v1.cni.cncf.io/networks** 주석에 **@name** 접미사를 추가합니다.

②

**net1** 인터페이스는 **SR-IOV** 가상 기능을 기반으로 합니다.

③

**net2** 인터페이스는 **SR-IOV** 가상 기능을 기반으로 합니다.



참고

Pod 주석에 인터페이스 이름이 구성되지 않은 경우 인터페이스 이름은 **net<n>** 으로 자동으로 할당되고 **<n>** 은 1 부터 시작됩니다.

4.

선택 사항: **bond0** 과 같은 특정 인터페이스 이름을 설정하려면

**k8s.v1.cni.cncf.io/networks** 주석을 편집하고 **bond0** 을 다음과 같이 설정합니다.

**annotations:**

**k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1@bond0**

## 20.12. 하드웨어 오프로드 구성

클러스터 관리자는 호환 노드에서 하드웨어 오프로드를 구성하여 데이터 처리 성능을 개선하고 호스트 CPU의 부하를 줄일 수 있습니다.

### 20.12.1. 하드웨어 오프로드 정보

**Open vSwitch** 하드웨어 오프로드는 CPU에서 벗어나 네트워크 인터페이스 컨트롤러의 전용 프로세서로 오프로드하여 네트워크 작업을 처리하는 방법입니다. 결과적으로 클러스터는 데이터 전송 속도가 빨라지고 CPU 워크로드를 줄이며 컴퓨팅 비용을 절감할 수 있습니다.

이 기능의 주요 요소는 **SmartNICs**라는 네트워크 인터페이스 컨트롤러의 최신 클래스입니다. **SmartNIC**는 컴퓨팅 방식으로 네트워크 처리 작업을 처리할 수 있는 네트워크 인터페이스 컨트롤러입니다. 전용 그래픽 카드가 그래픽 성능을 향상시킬 수 있는 것과 마찬가지로 **SmartNIC**는 네트워크 성능을 향상시킬 수 있습니다. 각각의 경우 전용 프로세서는 특정 유형의 처리 작업에 대한 성능을 향상시킵니다.

**OpenShift Container Platform**에서는 호환되는 **SmartNIC**가 있는 베어 메탈 노드의 하드웨어 오프로드를 구성할 수 있습니다. 하드웨어 오프로드는 **SR-IOV Network Operator**에 의해 구성 및 활성화됩니다.

하드웨어 오프로드는 모든 워크로드 또는 애플리케이션 유형과 호환되지 않습니다. 다음 두 가지 통신 유형만 지원됩니다.

- **pod-to-pod**
- 서비스에서 일반 Pod에서 지원하는 **ClusterIP** 서비스인 **pod-to-service**

모든 경우에서 하드웨어 오프로드는 해당 Pod 및 서비스가 호환 가능한 **SmartNIC**인 노드에 할당될 때만 수행됩니다. 예를 들어 하드웨어 오프로드가 있는 노드의 Pod가 일반 노드의 서비스와 통신하려고 한다고 가정합니다. 일반 노드에서 모든 처리가 커널에서 수행되므로 **pod-to-service** 통신의 전체 성능은 해당 일반 노드의 최대 성능으로 제한됩니다. 하드웨어 오프로드는 **DPDK** 애플리케이션과 호환되지 않습니다.

### 20.12.2. 지원되는 장치

하드웨어 오프로드는 다음 네트워크 인터페이스 컨트롤러에서 지원됩니다.

표 20.15. 지원되는 네트워크 인터페이스 컨트롤러

제조업체	모델	벤더 ID	장치 ID
Mellanox	MT27800 제품군 [ConnectX-5]	15b3	1017
Mellanox	MT28880 제품군 [ConnectX-5 Ex]	15b3	1019

### 20.12.3. 사전 요구 사항

- 클러스터에는 하드웨어 오프로드에 대해 지원되는 네트워크 인터페이스 컨트롤러가 있는 하나 이상의 베어 메탈 머신이 있습니다.
- SR-IOV Network Operator**가 설치되어 있어야 합니다.
- 클러스터는 **OVN-Kubernetes CNI** 를 사용합니다.
- OVN-Kubernetes CNI** 구성에서는 `gatewayConfig.routingViaHost` 필드가 `false` 로 설정됩니다.

### 20.12.4. 하드웨어 오프로드를 위한 머신 구성 풀 구성

하드웨어 오프로드를 활성화하려면 먼저 전용 머신 구성 풀을 생성하고 **SR-IOV Network Operator**에서 작동하도록 구성해야 합니다.

#### 사전 요구 사항

- OpenShift CLI(oc)**를 설치합니다.
- `cluster-admin` 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 절차

1.

에서 하드웨어 오프로드를 사용할 머신의 머신 구성 풀을 생성합니다.

a.

다음 예제와 같은 콘텐츠를 사용하여 `mcp-offloading.yaml` 과 같은 파일을 생성합니다.

```
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

하드웨어 오프로드를 위한 머신 구성 풀의 이름입니다.

3

이 노드 역할 레이블은 머신을 구성 풀에 노드를 추가하는 데 사용됩니다.

b.

머신 구성 풀에 대한 구성을 적용합니다.

```
$ oc create -f mcp-offloading.yaml
```

2.

머신 구성 풀에 노드를 추가합니다. 각 노드에 풀의 노드 역할 레이블로 레이블을 지정합니다.

```
$ oc label node worker-2 node-role.kubernetes.io/mcp-offloading=""
```

3.

선택 사항: 새 풀이 생성되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get nodes
```

출력 예

NAME	STATUS	ROLES	AGE	VERSION
master-0	Ready	master	2d	v1.24.0
master-1	Ready	master	2d	v1.24.0
master-2	Ready	master	2d	v1.24.0
worker-0	Ready	worker	2d	v1.24.0
worker-1	Ready	worker	2d	v1.24.0
worker-2	Ready	mcp-offloading,worker	47h	v1.24.0
worker-3	Ready	mcp-offloading,worker	47h	v1.24.0

4.

**SriovNetworkPoolConfig** 사용자 정의 리소스에 이 머신 구성 풀을 추가합니다.

a.

다음 예와 같은 콘텐츠를 사용하여 **sriov-pool-config.yaml** 과 같은 파일을 생성합니다.

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading ①
    
```

①

하드웨어 오프로드를 위한 머신 구성 풀의 이름입니다.

b.

설정을 적용합니다.

```

$ oc create -f <SriovNetworkPoolConfig_name>.yaml
    
```





## 참고

**SriovNetworkPoolConfig** 오브젝트에 지정된 구성을 적용하면 **SR-IOV Operator**가 머신 구성 풀의 노드를 비우고 재시작합니다.

구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다.

### 20.12.5. SR-IOV 네트워크 노드 정책 구성

**SR-IOV** 네트워크 노드 정책을 생성하여 노드의 **SR-IOV** 네트워크 장치 구성을 생성할 수 있습니다. 하드웨어 오프로드를 활성화하려면 값 **"switchdev"** 로 **.spec.eSwitchMode** 필드를 정의해야 합니다.

다음 절차에서는 하드웨어 오프로드를 사용하여 네트워크 인터페이스 컨트롤러에 대한 **SR-IOV** 인터페이스를 생성합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 절차

1. 다음 예와 같은 콘텐츠를 사용하여 **sriov-node-policy.yaml** 과 같은 파일을 생성합니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy <.>
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice <.>
  eSwitchMode: "switchdev" <.>
  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
```

<.> 사용자 정의 리소스 오브젝트의 이름입니다. <.> Required. vfio-pci. <.> 필요에서는 하드웨어 오프로드가 지원되지 않습니다.

2. 정책에 대한 구성을 적용합니다.

```
$ oc create -f sriov-node-policy.yaml
```



참고

**SriovNetworkPoolConfig** 오브젝트에 지정된 구성을 적용하면 **SR-IOV Operator**가 머신 구성 풀의 노드를 비우고 재시작합니다.

구성 변경 사항을 적용하는 데 몇 분이 걸릴 수 있습니다.

### 20.12.5.1. OpenStack의 SR-IOV 네트워크 노드 정책에

다음 예제에서는 **RHOSP(Red Hat OpenStack Platform)**에서 하드웨어 오프로드가 있는 **NIC(네트워크 인터페이스 컨트롤러)의 SR-IOV 인터페이스**를 설명합니다.

**RHOSP**에서 하드웨어 오프로드가 있는 **NIC의 SR-IOV 인터페이스**

```
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}
```

### 20.12.6. 네트워크 연결 정의 생성

머신 구성 풀 및 **SR-IOV** 네트워크 노드 정책을 정의한 후 사용자가 지정한 네트워크 인터페이스 카드에 대한 네트워크 연결 정의를 생성할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 절차

1. 다음 예제와 같은 콘텐츠를 사용하여 **net-attach-def.yaml** 과 같은 파일을 생성합니다.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net-attach-def <.>
  namespace: net-attach-def <.>
  annotations:
    v1.multus-cni.io/default-network: openshift.io/mlxnic <.>
spec:
  config: '{"cniVersion":"0.3.1","name":"ovn-kubernetes","type":"ovn-k8s-cni-overlay","ipam":{"},"dns":{"}}'
```

<.> 네트워크 연결 정의의 이름입니다. <.> 네트워크 연결 정의의 네임스페이스입니다. <.> **SriovNetworkNodePolicy** 오브젝트에 지정한 **spec.resourceName** 필드의 값입니다.

2. 네트워크 연결 정의에 대한 구성을 적용합니다.

```
$ oc create -f net-attach-def.yaml
```

#### 검증

- 다음 명령을 실행하여 새 정의가 있는지 확인합니다.

```
$ oc get net-attach-def -A
```

출력 예

```

NAMESPACE   NAME           AGE
net-attach-def net-attach-def 43h

```

### 20.12.7. Pod에 네트워크 연결 정의 추가

머신 구성 풀, **SriovNetworkPoolConfig** 및 **SriovNetworkNodePolicy** 사용자 정의 리소스를 생성한 후 Pod 사양에 네트워크 연결 정의를 추가하여 해당 구성을 Pod에 적용할 수 있습니다.

절차

- Pod 사양에서 **.metadata.annotations.k8s.v1.cni.cncf.io/networks** 필드를 추가하고 하드웨어 오프로드에 대해 생성한 네트워크 연결 정의를 지정합니다.

```

....
metadata:
  annotations:
    k8s.v1.cni.cncf.io/default-network: net-attach-def/net-attach-def <.>

```

<.> 값은 하드웨어 오프로드를 위해 생성한 네트워크 연결 정의의 이름과 네임스페이스여야 합니다.

### 20.13. SR-IOV NETWORK OPERATOR 설치 제거

**SR-IOV Network Operator**를 설치 제거하려면 실행 중인 **SR-IOV** 워크로드를 삭제하고 **Operator**를 제거한 다음 **Operator**에서 사용하는 **Webhook**를 삭제해야 합니다.

#### 20.13.1. SR-IOV Network Operator 설치 제거

클러스터 관리자는 **SR-IOV Network Operator**를 제거할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.
- **SR-IOV Network Operator**가 설치되어 있어야 합니다.

### 절차

1.

모든 **SR-IOV** 사용자 정의 리소스(**CR**)를 삭제합니다.

```
$ 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
```

2.

"클러스터에서 **Operator** 삭제" 섹션의 지침에 따라 클러스터에서 **SR-IOV Network Operator**를 제거합니다.

3.

**SR-IOV Network Operator**가 제거된 후 클러스터에 남아 있는 **SR-IOV** 사용자 정의 리소스 정의를 삭제합니다.

```
$ 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.

**SR-IOV** 웹 후크를 삭제합니다.

```
$ oc delete mutatingwebhookconfigurations network-resources-injector-config
```

```
$ oc delete MutatingWebhookConfiguration sriov-operator-webhook-config
```

```
$ oc delete ValidatingWebhookConfiguration sriov-operator-webhook-config
```

5.

**SR-IOV Network Operator** 네임스페이스를 삭제합니다.

```
$ oc delete namespace openshift-sriov-network-operator
```

추가 리소스

- [클러스터에서 Operator 삭제](#)

## 21장. OPENSIFT SDN 기본 CNI 네트워크 공급자

### 21.1. OPENSIFT SDN 기본 CNI 네트워크 공급자 정보

OpenShift Container Platform에서는 소프트웨어 정의 네트워킹(SDN) 접근법을 사용하여 OpenShift Container Platform 클러스터 전체의 pod 간 통신이 가능한 통합 클러스터 네트워크를 제공합니다. 이 pod 네트워크는 OVS(Open vSwitch)를 사용하여 오버레이 네트워크를 구성하는 OpenShift SDN에 의해 설정 및 유지 관리됩니다.

#### 21.1.1. OpenShift SDN 네트워크 격리 모드

OpenShift SDN은 pod 네트워크 구성을 위한 세 가지 SDN 모드를 제공합니다.

- 네트워크 정책 모드를 통해 프로젝트 관리자는 NetworkPolicy 개체를 사용하여 자체 격리 정책을 구성할 수 있습니다. 네트워크 정책은 OpenShift Container Platform 4.11의 기본 모드입니다.
- 다중 테넌트 모드를 사용하면 Pod 및 서비스에 대한 프로젝트 수준 격리를 제공할 수 있습니다. 다른 프로젝트의 Pod는 다른 프로젝트의 Pod 및 Service에서 패킷을 보내거나 받을 수 없습니다. 프로젝트에 대한 격리를 비활성화하여 전체 클러스터의 모든 pod 및 service에 네트워크 트래픽을 보내고 해당 pod 및 service로부터 네트워크 트래픽을 수신할 수 있습니다.
- 서브넷 모드는 모든 pod가 다른 모든 pod 및 service와 통신할 수 있는 플랫폼 pod 네트워크를 제공합니다. 네트워크 정책 모드는 서브넷 모드와 동일한 기능을 제공합니다.

#### 21.1.2. 지원되는 기본 CNI 네트워크 공급자 기능 매트릭스

OpenShift Container Platform은 기본 CNI(Container Network Interface) 네트워크 공급자를 위해 OpenShift SDN 및 OVN-Kubernetes의 두 가지 지원 옵션을 제공합니다. 다음 표는 두 네트워크 공급자 모두에 대한 현재 기능 지원을 요약합니다.

표 21.1. 기본 CNI 네트워크 공급자 기능 비교

기능	OpenShift SDN	OVN-Kubernetes
송신 IP	지원됨	지원됨
송신 방화벽 [1]	지원됨	지원됨
송신 라우터	지원됨	지원됨 [2]

기능	OpenShift SDN	OVN-Kubernetes
하이브리드 네트워킹	지원되지 않음	지원됨
IPsec 암호화	지원되지 않음	지원됨
IPv6	지원되지 않음	지원됨 [3]
Kubernetes 네트워크 정책	지원됨	지원됨
Kubernetes 네트워크 정책 로그	지원되지 않음	지원됨
멀티 캐스트	지원됨	지원됨
하드웨어 오프로드	지원되지 않음	지원됨

1. 송신 방화벽은 **OpenShift SDN**에서 송신 네트워크 정책이라고도 합니다. 이것은 네트워크 정책 송신과 동일하지 않습니다.
2. **OVN-Kubernetes**용 송신 라우터는 리디렉션 모드만 지원합니다.
3. **IPv6**는 베어 메탈 클러스터에서만 지원됩니다.

## 21.2. 프로젝트의 송신 IP 구성

클러스터 관리자는 **OpenShift SDN CNI(Container Network Interface)** 클러스터 네트워크 공급자를 구성하여 하나 이상의 송신 IP 주소를 프로젝트에 할당할 수 있습니다.

### 21.2.1. 송신 IP 주소 아키텍처 설계 및 구현

**OpenShift Container Platform** 송신 IP 주소 기능을 사용하면 하나 이상의 네임스페이스에 있는 하나 이상의 Pod에서 발생하는 트래픽의 소스 IP 주소가 클러스터 네트워크 외부 서비스에 일관되게 표시되도록 할 수 있습니다.

예를 들어 클러스터 외부 서버에서 호스팅되는 데이터베이스를 주기적으로 쿼리하는 Pod가 있을 수 있습니다. 서버에 대한 액세스 요구 사항을 적용하기 위해 패킷 필터링 장치는 특정 IP 주소의 트래픽만 허용하도록 구성됩니다. 특정 Pod에서만 서버에 안정적으로 액세스할 수 있도록 허용하려면 서버에 요청하는 Pod에 대해 특정 송신 IP 주소를 구성하면 됩니다.



네임스페이스에 할당된 송신 IP 주소는 특정 대상으로 트래픽을 보내는 데 사용되는 송신 라우터와 다릅니다.

일부 클러스터 구성에서 애플리케이션 Pod 및 인그레스 라우터 Pod가 동일한 노드에서 실행됩니다. 이 시나리오에서 애플리케이션 프로젝트에 대한 송신 IP 주소를 구성하는 경우 애플리케이션 프로젝트에서 경로로 요청을 보낼 때 IP 주소가 사용되지 않습니다.

송신 IP 주소는 노드의 기본 네트워크 인터페이스에서 추가 IP 주소로 구현되며 노드의 기본 IP 주소와 동일한 서브넷에 있어야 합니다. 추가 IP 주소를 클러스터의 다른 노드에 할당해서는 안 됩니다.



중요

송신 IP 주소는 `ifcfg-eth0`과 같은 Linux 네트워크 구성 파일에서 구성하지 않아야 합니다.

#### 21.2.1.1. 플랫폼 지원

다음 표에는 다양한 플랫폼의 송신 IP 주소 기능에 대한 지원이 요약되어 있습니다.

플랫폼	지원됨
베어 메탈	있음
VMware vSphere	있음
Red Hat OpenStack Platform (RHOSP)	없음
AWS(Amazon Web Services)	있음
GCP(Google Cloud Platform)	있음
Microsoft Azure	있음



중요

**EgressIP** 기능을 사용하여 컨트롤 플레인 노드에 송신 IP 주소를 할당하는 것은 **AWS(Amazon Web Services)**에서 프로비저닝된 클러스터에서는 지원되지 않습니다. ([BZ#2039656](#))

### 21.2.1.2. 퍼블릭 클라우드 플랫폼 고려 사항

퍼블릭 클라우드 인프라에 프로비저닝된 클러스터의 경우 노드당 절대 할당 가능한 IP 주소 수에 대한 제약 조건이 있습니다. 노드당 최대 할당 가능한 IP 주소 수 또는 IP 용량은 다음 공식에서 설명할 수 있습니다.

$$\text{IP capacity} = \text{public cloud default capacity} - \text{sum}(\text{current IP assignments})$$

**Egress IP** 기능은 노드당 IP 주소 용량을 관리하는 반면, 배포에서 이 제약 조건을 계획하는 것이 중요합니다. 예를 들어 8개의 노드가 있는 베어 메탈 인프라에 설치된 클러스터의 경우 150개의 송신 IP 주소를 구성할 수 있습니다. 그러나 공용 클라우드 공급자가 IP 주소 용량을 노드당 10개의 IP 주소로 제한하는 경우 총 할당 가능한 IP 주소 수는 80입니다. 이 예제에서 동일한 IP 주소 용량을 얻으려면 7개의 추가 노드를 할당해야 합니다.

퍼블릭 클라우드 환경에서 노드의 IP 용량 및 서브넷을 확인하려면 `oc get node <node_name> -o yaml` 명령을 입력합니다. `cloud.network.openshift.io/egress-ipconfig` 주석에는 노드에 대한 용량 및 서브넷 정보가 포함됩니다.

주석 값은 기본 네트워크 인터페이스에 다음 정보를 제공하는 필드가 포함된 단일 오브젝트가 포함된 배열입니다.

- **Interface:** AWS 및 Azure의 인터페이스 ID와 GCP의 인터페이스 이름을 지정합니다.
- **ifaddr:** 하나 또는 두 IP 주소 제품군에 대한 서브넷 마스크를 지정합니다.
- **capacity:** 노드의 IP 주소 용량을 지정합니다. AWS에서 IP 주소 용량은 IP 주소 제품군별로 제공됩니다. Azure 및 GCP에서 IP 주소 용량에는 IPv4 및 IPv6 주소가 모두 포함됩니다.

다음 예제에서는 여러 공용 클라우드 공급자의 노드의 주석을 보여줍니다. 주석은 가독성을 위해 들여 쓰기되어 있습니다.

AWS의 `cloud.network.openshift.io/egress-ipconfig` 주석 예

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface": "eni-078d267045138e436",
    "ifaddr": {"ipv4": "10.0.128.0/18"},
  }
]
```

```

    "capacity":{"ipv4":14,"ipv6":15}
  }
]

```

GCP의 `cloud.network.openshift.io/egress-ipconfig` 주석의 예

```

cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]

```

다음 섹션에서는 용량 계산에 사용할 지원되는 퍼블릭 클라우드 환경에 대한 IP 주소 용량에 대해 설명합니다.

#### 21.2.1.2.1. AWS(Amazon Web Services) IP 주소 용량 제한

AWS에서 IP 주소 할당에 대한 제한은 구성된 인스턴스 유형에 따라 다릅니다. 자세한 내용은 [인스턴스 유형당 네트워크 인터페이스당 IP 주소](#)를 참조하십시오.

#### 21.2.1.2.2. GCP(Google Cloud Platform) IP 주소 용량 제한

GCP에서 네트워킹 모델은 IP 주소 할당이 아닌 IP 주소 별칭을 통해 추가 노드 IP 주소를 구현합니다. 그러나 IP 주소 용량은 IP 별칭 용량에 직접 매핑됩니다.

IP 별칭 할당에는 다음 용량 제한이 있습니다.

- 노드당 최대 IP 별칭(IPv4 및 IPv6 모두)은 10입니다.
- VPC당 최대 IP 별칭 수는 지정되지 않지만 OpenShift Container Platform 확장성 테스트에서는 최대 15,000개가 됩니다.

자세한 내용은 [인스턴스 할당량별 및 IP 범위 개요](#) 를 참조하십시오.

### 21.2.1.2.3. Microsoft Azure IP 주소 용량 제한

Azure에서 IP 주소 할당에 대해 다음과 같은 용량 제한이 있습니다. **On Azure, the following capacity limits exist for IP address assignment:**

- **NIC당 IPv4 및 IPv6 모두에 대해 할당 가능한 최대 IP 주소 수는 256입니다.**
- **가상 네트워크당 할당된 IP 주소의 최대 수는 65,536을 초과할 수 없습니다.**

자세한 내용은 [네트워킹 제한](#) 을 참조하십시오.

### 21.2.1.3. 제한 사항

다음 제한 사항은 **OpenShift SDN** 클러스터 네트워크 공급자와 함께 송신 IP 주소를 사용하는 경우 적용됩니다.

- 동일한 노드에서 수동 할당 및 자동 할당 송신 IP 주소를 사용할 수 없습니다.
- IP 주소 범위에서 송신 IP 주소를 수동으로 할당하는 경우 해당 범위를 자동 IP 할당에 사용할 수 있도록 설정해서는 안 됩니다.
- **OpenShift SDN** 송신 IP 주소 구현을 사용하여 여러 네임스페이스에서 송신 IP 주소를 공유할 수 없습니다.

네임스페이스 간에 IP 주소를 공유해야 하는 경우 **OVN-Kubernetes** 클러스터 네트워크 공급자 송신 IP 주소 구현을 통해 여러 네임스페이스에서 IP 주소를 확장할 수 있습니다.



## 참고

다중 테넌트 모드에서 OpenShift SDN을 사용하는 경우 연결된 프로젝트에 의해 다른 네임스페이스에 조인된 네임스페이스와 함께 송신 IP 주소를 사용할 수 없습니다. 예를 들어 `oc adm pod-network join-projects --to=project1 project2` 명령을 실행하여 `project1` 및 `project2`를 조인한 경우 두 프로젝트 모두 송신 IP 주소를 사용할 수 없습니다. 자세한 내용은 [BZ#1645577](#)를 참조하십시오.

### 21.2.1.4. IP 주소 할당 접근 방식

**NetNamespace** 오브젝트의 **egressIPs** 매개변수를 설정하여 네임스페이스에 송신 IP 주소를 지정할 수 있습니다. 송신 IP 주소가 프로젝트와 연결된 후 OpenShift SDN을 사용하면 다음 두 가지 방법으로 송신 IP 주소를 호스트에 할당할 수 있습니다.

- 자동 할당 방식에서는 송신 IP 주소 범위가 노드에 할당됩니다.
- 수동 할당 방식에서는 하나 이상의 송신 IP 주소 목록이 노드에 할당됩니다.

송신 IP 주소를 요청하는 네임스페이스는 해당 송신 IP 주소를 호스트할 수 있는 노드와 일치되며 송신 IP 주소가 해당 노드에 할당됩니다. **egressIPs** 매개변수가 **NetNamespace** 오브젝트에 설정되었지만 IP 주소를 송신하는 노드 호스트가 없는 경우 네임스페이스에서 송신하는 트래픽이 삭제됩니다.

노드의 고가용성은 자동입니다. 송신 IP 주소를 호스팅하는 노드에 도달할 수 없고 해당 송신 IP 주소를 호스트할 수 있는 노드가 있으면 송신 IP 주소가 새 노드로 이동합니다. 연결할 수 없는 노드가 다시 온라인 상태가 되면 송신 IP 주소가 자동으로 이동하여 노드 간에 송신 IP 주소의 균형을 조정합니다.

#### 21.2.1.4.1. 자동 할당된 송신 IP 주소 사용 시 고려사항

송신 IP 주소에 자동 할당 방식을 사용할 때는 다음 사항을 고려해야 합니다.

- 각 노드의 **HostSubnet** 리소스의 **egressCIDRs** 매개변수를 설정하여 노드가 호스트할 수 있는 송신 IP 주소 범위를 나타냅니다. OpenShift Container Platform은 지정한 IP 주소 범위를 기반으로 **HostSubnet** 리소스의 **egressIPs** 매개변수를 설정합니다.

네임스페이스의 송신 IP 주소를 호스팅하는 노드에 도달할 수 없는 경우 OpenShift Container Platform은 호환되는 송신 IP 주소 범위를 가진 다른 노드에 송신 IP 주소를 다시 할당합니다. 자동 할당 방식은 추가 IP 주소를 노드와 연결할 수 있는 유연성이 있는 환경에 설치된 클러스터에 가장 적합합니다.

### 21.2.1.4.2. 수동으로 할당된 송신 IP 주소 사용 시 고려사항

이 방법을 사용하면 송신 IP 주소를 호스팅할 수 있는 노드를 제어할 수 있습니다.



**참고**

클러스터가 퍼블릭 클라우드 인프라에 설치된 경우 송신 IP 주소를 할당하는 각 노드에 IP 주소를 호스팅하는 데 충분한 예비 용량이 있는지 확인해야 합니다. 자세한 내용은 이전 섹션의 "플랫폼 고려 사항"을 참조하십시오.

송신 IP 주소에 수동 할당 방식을 사용할 때는 다음 사항을 고려해야 합니다.

- 각 노드의 **HostSubnet** 리소스의 **egressIPs** 매개변수를 설정하여 노드가 호스트할 수 있는 IP 주소를 표시합니다.
- 네임스페이스당 여러 개의 송신 IP 주소가 지원됩니다.

네임스페이스에 여러 송신 IP 주소가 있고 해당 주소가 여러 노드에서 호스팅되는 경우 다음과 같은 추가 고려 사항이 적용됩니다.

- Pod가 송신 IP 주소를 호스팅하는 노드에 있는 경우 해당 pod는 항상 노드에서 송신 IP 주소를 사용합니다.
- Pod가 송신 IP 주소를 호스팅하는 노드에 없는 경우 해당 Pod는 송신 IP 주소를 임의로 사용합니다.

### 21.2.2. 네임스페이스에 자동으로 할당된 송신 IP 주소 구성

OpenShift Container Platform에서는 하나 이상의 노드에서 특정 네임스페이스에 대한 송신 IP 주소를 자동으로 할당할 수 있습니다.

**사전 요구 사항**

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

• OpenShift CLI(oc)가 설치되어 있습니다.

## 프로세스

1.

다음 JSON을 사용하여 송신 IP 주소로 NetNamespace 오브젝트를 업데이트합니다.

```
$ oc patch netnamespace <project_name> --type=merge -p \
  '{
    "egressIPs": [
      "<ip_address>"
    ]
  }'
```

다음과 같습니다.

**<project\_name>**

프로젝트 이름을 지정합니다.

**<ip\_address>**

egressIPs 배열에 대해 하나 이상의 송신 IP 주소를 지정합니다.

예를 들어 project1을 IP 주소 192.168.1.100에 할당하고 project2를 IP 주소 192.168.1.101에 할당하려면 다음을 수행합니다.

```
$ oc patch netnamespace project1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100"]}'
$ oc patch netnamespace project2 --type=merge -p \
  '{"egressIPs": ["192.168.1.101"]}'
```



### 참고

OpenShift SDN은 NetNamespace 오브젝트를 관리하므로 기존 NetNamespace 오브젝트를 수정하기만 하면 됩니다. 새 NetNamespace 오브젝트를 생성하지 마십시오.

2.

다음 JSON을 사용하여 각 호스트에 대해 egressCIDRs 매개변수를 설정하여 송신 IP 주소를 호스팅할 수 있는 노드를 표시합니다.

```
$ oc patch hostsubnet <node_name> --type=merge -p \
  '{
    "egressCIDRs": [
      "<ip_address_range>", "<ip_address_range>"
    ]
  }'
```

다음과 같습니다.

**<node\_name>**

노드 이름을 지정합니다.

**<ip\_address\_range>**

CIDR 형식의 IP 주소 범위를 지정합니다. **egressCIDRs** 배열에 대해 두 개 이상의 주소 범위를 지정할 수 있습니다.

예를 들어, **node1** 및 **node2**를 192.168.1.0에서 192.168.1.255 범위의 송신 IP 주소를 호스팅 하도록 설정하려면 다음을 수행합니다.

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
$ oc patch hostsubnet node2 --type=merge -p \
  '{"egressCIDRs": ["192.168.1.0/24"]}'
```

OpenShift Container Platform은 특정 송신 IP 주소를 균형 잡힌 방식으로 사용 가능한 노드에 자동으로 할당합니다. 이 경우 송신 IP 주소 192.168.1.100을 **node1**에 할당하고 송신 IP 주소 192.168.1.101을 **node2**에 할당하거나 그 반대의 경우도 마찬가지입니다.

### 21.2.3. 네임스페이스에 수동으로 할당된 송신 IP 주소 구성

OpenShift Container Platform에서 하나 이상의 송신 IP 주소를 네임스페이스와 연결할 수 있습니다.

사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- OpenShift CLI(**oc**)가 설치되어 있습니다.



## 프로세스

1.

원하는 IP 주소로 다음 JSON 오브젝트를 지정하여 **NetNamespace** 오브젝트를 업데이트합니다.

```
$ oc patch netnamespace <project_name> --type=merge -p \
  '{
    "egressIPs": [
      "<ip_address>"
    ]
  }'
```

다음과 같습니다.

**<project\_name>**

프로젝트 이름을 지정합니다.

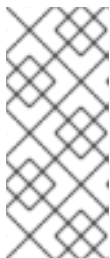
**<ip\_address>**

**egressIPs** 배열에 대해 하나 이상의 송신 IP 주소를 지정합니다.

예를 들어, **project1** 프로젝트를 IP 주소 **192.168.1.100** 및 **192.168.1.101**에 할당하려면 다음을 수행합니다.

```
$ oc patch netnamespace project1 --type=merge \
  -p '{"egressIPs": ["192.168.1.100","192.168.1.101"]}'
```

고가용성을 제공하기 위해 **egressIPs** 값을 서로 다른 노드에서 둘 이상의 IP 주소로 설정합니다. 여러 송신 IP 주소가 설정되면 **Pod**는 모든 송신 IP 주소를 거의 동일하게 사용합니다.



## 참고

**OpenShift SDN**은 **NetNamespace** 오브젝트를 관리하므로 기존 **NetNamespace** 오브젝트를 수정하기만 하면 됩니다. 새 **NetNamespace** 오브젝트를 생성하지 마십시오.

2.

송신 IP 주소를 노드 호스트에 수동으로 할당합니다.

클러스터가 퍼블릭 클라우드 인프라에 설치된 경우 노드에 사용 가능한 IP 주소 용량이 있는지 확인해야 합니다.

노드 호스트의 **HostSubnet** 오브젝트에서 **egressIPs** 매개변수를 설정합니다. 다음 **JSON**을 사용하여 해당 노드 호스트에 할당하려는 만큼의 **IP** 주소를 포함합니다.

```
$ oc patch hostssubnet <node_name> --type=merge -p \
  '{
    "egressIPs": [
      "<ip_address>",
      "<ip_address>"
    ]
  }'
```

다음과 같습니다.

**<node\_name>**

노드 이름을 지정합니다.

**<ip\_address>**

**IP** 주소를 지정합니다. **egressIPs** 배열에 대해 두 개 이상의 **IP** 주소를 지정할 수 있습니다.

예를 들어 **node1**에 송신 **IP** **192.168.1.100**, **192.168.1.101** 및 **192.168.1.102**가 있도록 지정하려면 다음을 수행합니다.

```
$ oc patch hostssubnet node1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100", "192.168.1.101", "192.168.1.102"]}'
```

이전 예에서 **project1**의 모든 송신 트래픽은 지정된 송신 **IP**를 호스팅하는 노드로 라우팅된 다음 **NAT(Network Address Translation)**를 사용하여 해당 **IP** 주소에 연결됩니다.

#### 21.2.4. 추가 리소스

- 수동 송신 **IP** 주소 할당을 구성하는 경우 **IP** 용량 계획에 대한 자세한 내용은 [플랫폼 고려 사항을 참조하십시오](#).

### 21.3. 프로젝트에 대한 송신 방화벽 구성

클러스터 관리자는 **OpenShift Container Platform** 클러스터에서 나가는 송신 트래픽을 제한하는 프

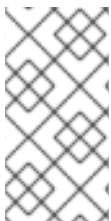
로젝트에 대한 송신 방화벽을 생성할 수 있습니다.

### 21.3.1. 프로젝트에서 송신 방화벽이 작동하는 방식

클러스터 관리자는 송신 방화벽을 사용하여 일부 또는 모든 Pod가 클러스터 내에서 액세스할 수 있는 외부 호스트를 제한할 수 있습니다. 송신 방화벽은 다음 시나리오를 지원합니다.

- Pod는 내부 호스트에만 연결할 수 있으며 공용 인터넷 연결을 시작할 수 없습니다.
- Pod는 공용 인터넷에만 연결할 수 있으며 OpenShift Container Platform 클러스터 외부에 있는 내부 호스트에 대한 연결을 시작할 수 없습니다.
- Pod는 지정된 내부 서브넷이나 OpenShift Container Platform 클러스터 외부의 호스트에 연결할 수 없습니다.
- Pod는 특정 외부 호스트에만 연결할 수 있습니다.

예를 들어, 한 프로젝트가 지정된 IP 범위에 액세스하도록 허용하지만 다른 프로젝트에 대한 동일한 액세스는 거부할 수 있습니다. 또는 애플리케이션 개발자가 Python pip 미러에서 업데이트하지 못하도록 하고 승인된 소스에서만 업데이트를 수행하도록 할 수 있습니다.



#### 참고

송신 방화벽은 호스트 네트워크 네임스페이스에 적용되지 않습니다. 호스트 네트워킹이 활성화된 Pod는 송신 방화벽 규칙의 영향을 받지 않습니다.

**EgressNetworkPolicy CR**(사용자 정의 리소스) 오브젝트를 만들어 송신 방화벽 정책을 구성합니다. 송신 방화벽은 다음 기준 중 하나를 충족하는 네트워크 트래픽과 일치합니다.

- CIDR 형식의 IP 주소 범위
- IP 주소로 확인되는 DNS 이름

## 중요

송신 방화벽에 0.0.0.0/0에 대한 거부 규칙이 포함된 경우 OpenShift Container Platform API 서버에 대한 액세스 권한이 차단됩니다. Pod가 OpenShift Container Platform API 서버에 액세스할 수 있도록 하려면 EgressFirewall과 함께 노드 포트를 사용할 때 액세스할 수 있도록 OVN(Open Virtual Network)의 기본 연결 네트워크 100.64.0.0/16 을 포함해야 합니다. 다음 예와 같이 송신 방화벽 규칙에서 API 서버가 청취하는 IP 주소 범위도 포함해야 합니다.

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
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

송신 방화벽의 네임스페이스입니다.

2

OpenShift Container Platform API 서버를 포함하는 IP 주소 범위입니다.

3

글로벌 거부 규칙은 OpenShift Container Platform API 서버에 액세스할 수 없습니다.

API 서버의 IP 주소를 찾으려면 `oc get ep kubernetes -n default` 를 실행합니다.

자세한 내용은 [BZ#1988324](#)에서 참조하십시오.

## 중요

송신 방화벽을 구성하려면 네트워크 정책 또는 다중 테넌트 모드를 사용하도록 OpenShift SDN을 구성해야 합니다.

네트워크 정책 모드를 사용하는 경우 송신 방화벽은 네임스페이스당 하나의 정책과만 호환되며 글로벌 프로젝트와 같이 네트워크를 공유하는 프로젝트에서는 작동하지 않습니다.



## 주의

송신 방화벽 규칙은 라우터를 통과하는 트래픽에는 적용되지 않습니다. Route CR 오브젝트를 생성할 권한이 있는 모든 사용자는 허용되지 않은 대상을 가리키는 경로를 생성하여 송신 방화벽 정책 규칙을 바이패스할 수 있습니다.

## 21.3.1.1. 송신 방화벽의 제한

송신 방화벽에는 다음과 같은 제한이 있습니다.

- EgressNetworkPolicy 오브젝트를 두 개 이상 보유할 수 있는 프로젝트는 없습니다.
- 프로젝트당 최대 1000개의 규칙이 있는 최대 하나의 EgressNetworkPolicy 오브젝트를 정의할 수 있습니다.
- 기본 프로젝트는 송신 방화벽을 사용할 수 없습니다.
- 다중 테넌트 모드에서 OpenShift SDN 기본 컨테이너 네트워크 인터페이스(CNI) 네트워크 공급자를 사용하는 경우 다음 제한 사항이 적용됩니다.
  - 글로벌 프로젝트는 송신 방화벽을 사용할 수 없습니다. `oc adm pod-network make-projects-global` 명령을 사용하여 프로젝트를 글로벌로 만들 수 있습니다.
  - `oc adm pod-network join-projects` 명령을 사용하여 병합된 프로젝트는 결합된 프로

젝트에서 송신 방화벽을 사용할 수 없습니다.

이러한 제한 사항을 위반하면 프로젝트의 송신 방화벽이 손상되고 모든 외부 네트워크 트래픽이 삭제될 수 있습니다.

**Egress** 방화벽 리소스는 `kube-node-lease`, `kube-public`, `kube-system`, `openshift` 및 `openshift-` 프로젝트에서 생성할 수 있습니다.

### 21.3.1.2. 송신 방화벽 정책 규칙에 대한 일치 순서

송신 방화벽 정책 규칙은 정의된 순서대로 처음부터 마지막까지 평가됩니다. **Pod**의 송신 연결과 일치하는 첫 번째 규칙이 적용됩니다. 해당 연결에 대한 모든 후속 규칙은 무시됩니다.

### 21.3.1.3. DNS(Domain Name Server) 확인 작동 방식

송신 방화벽 정책 규칙에서 **DNS** 이름을 사용하는 경우 도메인 이름의 적절한 확인에는 다음 제한 사항이 적용됩니다.

- 도메인 이름 업데이트는 **TTL(Time To Live)** 기간에 따라 폴링됩니다. 기본적으로 기간은 **30초**입니다. 송신 방화벽 컨트롤러가 도메인 이름을 위해 로컬 이름 서버를 쿼리할 때 응답에 **30초 미만 TTL**이 포함된 경우 컨트롤러는 반환된 값으로 기간을 설정합니다. 응답의 **TTL**이 **30분**보다 크면 컨트롤러에서 기간을 **30분**으로 설정합니다. **TTL**이 **30초**에서 **30분** 사이인 경우 컨트롤러는 값을 무시하고 기간을 **30초**로 설정합니다.
- Pod**는 필요한 경우 동일한 로컬 이름 서버에서 도메인을 확인해야 합니다. 확인하지 않으면 송신 방화벽 컨트롤러와 **Pod**에 의해 알려진 도메인의 **IP** 주소가 다를 수 있습니다. 호스트 이름의 **IP** 주소가 다르면 송신 방화벽이 일관되게 적용되지 않을 수 있습니다.
- 송신 방화벽 컨트롤러와 **Pod**는 동일한 로컬 이름 서버를 비동기적으로 폴링하기 때문에 **Pod**가 송신 컨트롤러보다 먼저 업데이트된 **IP** 주소를 얻을 수 있으며 이로 인해 경쟁 조건이 발생합니다. 현재 이런 제한으로 인해 **EgressNetworkPolicy** 오브젝트의 도메인 이름 사용은 **IP** 주소가 자주 변경되지 않는 도메인에만 권장됩니다.



## 참고

송신 방화벽은 Pod가 DNS 확인을 위해 Pod가 있는 노드의 외부 인터페이스에 항상 액세스할 수 있도록 합니다.

송신 방화벽 정책에서 도메인 이름을 사용하고 로컬 노드의 DNS 서버에서 DNS 확인을 처리하지 않으면 Pod에서 도메인 이름을 사용하는 경우, DNS 서버의 IP 주소에 대한 액세스를 허용하는 송신 방화벽 규칙을 추가해야 합니다.

### 21.3.2. EgressNetworkPolicy CR(사용자 정의 리소스) 오브젝트

송신 방화벽에 대해 하나 이상의 규칙을 정의할 수 있습니다. 규칙이 적용되는 트래픽에 대한 사양을 담은 허용 규칙 또는 거부 규칙입니다.

다음 YAML은 EgressNetworkPolicy CR 오브젝트를 설명합니다.

#### EgressNetworkPolicy 오브젝트

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> 1
spec:
  egress: 2
  ...
```

1

송신 방화벽 정책의 이름입니다.

2

다음 섹션에서 설명하는 하나 이상의 송신 네트워크 정책 규칙 컬렉션입니다.

#### 21.3.2.1. EgressNetworkPolicy 규칙

다음 **YAML**은 송신 방화벽 규칙 오브젝트를 설명합니다. 송신 스텐자는 하나 이상의 오브젝트 배열을 예상합니다.

송신 정책 규칙 스텐자

```

egress:
- type: <type> 1
  to: 2
  cidrSelector: <cidr> 3
  dnsName: <dns_name> 4

```

1

규칙 유형입니다. 값은 허용 또는 거부여야 합니다.

2

송신 트래픽 일치 규칙을 설명하는 스텐자입니다. 규칙에 대한 **cidrSelector** 필드 또는 **dnsName** 필드의 값입니다. 동일한 규칙에서 두 필드를 모두 사용할 수 없습니다.

3

**CIDR** 형식의 IP 주소 범위입니다,

4

도메인 이름입니다.

### 21.3.2.2. EgressNetworkPolicy CR 오브젝트의 예

다음 예는 여러 가지 송신 방화벽 정책 규칙을 정의합니다.

```

apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
spec:
  egress: 1
  - type: Allow
    to:

```



```

cidrSelector: 1.2.3.0/24
- type: Allow
  to:
    dnsName: www.example.com
- type: Deny
  to:
    cidrSelector: 0.0.0.0/0

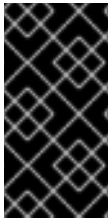
```

1

송신 방화벽 정책 규칙 오브젝트의 컬렉션입니다.

### 21.3.3. 송신 방화벽 정책 오브젝트 생성

클러스터 관리자는 프로젝트에 대한 송신 방화벽 정책 오브젝트를 만들 수 있습니다.



#### 중요

프로젝트에 이미 **EgressNetworkPolicy** 오브젝트가 정의되어 있으면 기존 정책을 편집하여 송신 방화벽 규칙을 변경해야 합니다.

#### 사전 요구 사항

- **OpenShift SDN 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용**하는 클러스터입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- 클러스터 관리자로 클러스터에 로그인해야 합니다.

#### 프로세스

1. 다음과 같이 정책 규칙을 생성합니다.
  - a. **<policy\_name>**이 송신 정책 규칙을 설명하는 **<policy\_name>.yaml** 파일을 만듭니다.
  - b. 생성한 파일에서 송신 정책 오브젝트를 정의합니다.

2.

다음 명령을 입력하여 정책 오브젝트를 생성합니다. `<policy_name>`을 정책 이름으로 바꾸고 `<project>`를 규칙이 적용되는 프로젝트로 바꿉니다.

```
$ oc create -f <policy_name>.yaml -n <project>
```

다음 예제에서는 `project1`이라는 프로젝트에 새 `EgressNetworkPolicy` 오브젝트를 생성합니다.

```
$ oc create -f default.yaml -n project1
```

출력 예

```
egressnetworkpolicy.network.openshift.io/v1 created
```

3.

선택사항: 나중에 변경할 수 있도록 `<policy_name>.yaml` 파일을 저장합니다.

## 21.4. 프로젝트의 송신 방화벽 편집

클러스터 관리자는 기존 송신 방화벽에 대한 네트워크 트래픽 규칙을 수정할 수 있습니다.

### 21.4.1. EgressNetworkPolicy 오브젝트 보기

클러스터의 `EgressNetworkPolicy` 오브젝트를 확인할 수 있습니다.

#### 사전 요구 사항

- **OpenShift SDN 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용하는 클러스터입니다.**
- **oc로 알려진 OpenShift 명령 인터페이스 (CLI)를 설치합니다.**

- 클러스터에 로그인해야 합니다.

#### 프로세스

1. 선택사항: 클러스터에 정의된 **EgressNetworkPolicy** 오브젝트의 이름을 보려면 다음 명령을 입력합니다.

```
$ oc get egressnetworkpolicy --all-namespaces
```

2. 정책을 검사하려면 다음 명령을 입력하십시오. <policy\_name>을 검사할 정책 이름으로 교체합니다.

```
$ oc describe egressnetworkpolicy <policy_name>
```

#### 출력 예

```
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
```

### 21.5. 프로젝트의 송신 방화벽 편집

클러스터 관리자는 기존 송신 방화벽에 대한 네트워크 트래픽 규칙을 수정할 수 있습니다.

#### 21.5.1. EgressNetworkPolicy 오브젝트 편집

클러스터 관리자는 프로젝트의 송신 방화벽을 업데이트할 수 있습니다.

#### 사전 요구 사항

- OpenShift SDN 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용

하는 클러스터입니다.

- **OpenShift CLI(oc)**를 설치합니다.
- 클러스터 관리자로 클러스터에 로그인해야 합니다.

#### 프로세스

1. 프로젝트의 **EgressNetworkPolicy** 오브젝트 찾습니다. **<project>**를 프로젝트 이름으로 바꿉니다.

```
$ oc get -n <project> egressnetworkpolicy
```

2. 선택 사항: 송신 네트워크 방화벽을 생성할 때 **EgressNetworkPolicy** 오브젝트 사본을 저장하지 않은 경우 다음 명령을 입력하여 사본을 생성합니다.

```
$ oc get -n <project> egressnetworkpolicy <name> -o yaml > <filename>.yaml
```

**<project>**를 프로젝트 이름으로 바꿉니다. **<name>**을 오브젝트 이름으로 변경합니다. **YAML**을 저장할 파일의 이름으로 **<filename>**을 바꿉니다.

3. 정책 규칙을 변경한 후 다음 명령을 입력하여 **EgressNetworkPolicy** 오브젝트를 바꿉니다. 업데이트된 **EgressNetworkPolicy** 오브젝트가 포함된 파일 이름으로 **<filename>**을 바꿉니다.

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

## 21.6. 프로젝트에서 송신 방화벽 제거

클러스터 관리자는 프로젝트에서 송신 방화벽을 제거하여 **OpenShift Container Platform** 클러스터를 나가는 프로젝트에서 네트워크 트래픽에 대한 모든 제한을 제거할 수 있습니다.

### 21.6.1. EgressNetworkPolicy 오브젝트 제거

클러스터 관리자는 프로젝트에서 송신 방화벽을 제거할 수 있습니다.

#### 사전 요구 사항

- **OpenShift SDN 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용하는 클러스터입니다.**
- **OpenShift CLI(oc)를 설치합니다.**
- **클러스터 관리자로 클러스터에 로그인해야 합니다.**

#### 프로세스

1. **프로젝트의 EgressNetworkPolicy 오브젝트 찾습니다. <project>를 프로젝트 이름으로 바꿉니다.**

```
$ oc get -n <project> egressnetworkpolicy
```

2. **EgressNetworkPolicy 오브젝트를 삭제하려면 다음 명령을 입력합니다. <project>를 프로젝트 이름으로 바꾸고 <name>을 오브젝트 이름으로 바꿉니다.**

```
$ oc delete -n <project> egressnetworkpolicy <name>
```

### 21.7. 송신 라우터 POD 사용에 대한 고려 사항

#### 21.7.1. 송신 라우터 Pod 정보

**OpenShift Container Platform** 송신 라우터 포드는 다른 용도로 사용되지 않는 프라이빗 소스 IP 주소에서 지정된 원격 서버로 트래픽을 리디렉션합니다. 송신 라우터 포드를 통해 특정 IP 주소에서만 액세스할 수 있도록 설정된 서버로 네트워크 트래픽을 보낼 수 있습니다.



#### 참고

송신 라우터 Pod는 모든 발신 연결을 위한 것은 아닙니다. 다수의 송신 라우터 Pod를 생성하는 경우 네트워크 하드웨어 제한을 초과할 수 있습니다. 예를 들어 모든 프로젝트 또는 애플리케이션에 대해 송신 라우터 Pod를 생성하면 소프트웨어에서 MAC 주소 필터링으로 돌아가기 전에 네트워크 인터페이스에서 처리할 수 있는 로컬 MAC 주소 수를 초과할 수 있습니다.



## 중요

송신 라우터 이미지는 **Amazon AWS, Azure Cloud** 또는 **macvlan** 트래픽과의 비호환성으로 인해 계층 2 조작을 지원하지 않는 기타 클라우드 플랫폼과 호환되지 않습니다.

### 21.7.1.1. 송신 라우터 모드

리디렉션 모드에서는 송신 라우터 포드가 자체 IP 주소에서 하나 이상의 대상 IP 주소로 트래픽을 리디렉션하도록 **iptables** 규칙을 구성합니다. 예약된 소스 IP 주소를 사용해야 하는 클라이언트 Pod는 대상 IP에 직접 연결하는 대신 송신 라우터에 연결하도록 수정해야 합니다.

**HTTP** 프록시 모드에서는 송신 라우터 Pod가 포트 **8080**에서 **HTTP** 프록시로 실행됩니다. 이 모드는 **HTTP** 기반 또는 **HTTPS** 기반 서비스에 연결하는 클라이언트에 대해서만 작동하지만 일반적으로 클라이언트 Pod를 덜 변경해야 작동합니다. 대부분의 프로그램은 환경 변수를 설정하여 **HTTP** 프록시를 사용하도록 지시할 수 있습니다.

**DNS** 프록시 모드에서는 송신 라우터 Pod가 자체 IP 주소에서 하나 이상의 대상 IP 주소로 **TCP** 기반 서비스의 **DNS** 프록시로 실행됩니다. 예약된 소스 IP 주소를 사용하려면 대상 IP 주소에 직접 연결하는 대신 송신 라우터 Pod에 연결하도록 클라이언트 Pod를 수정해야 합니다. 이렇게 수정하면 외부 대상에서 트래픽을 알려진 소스에서 발생하는 것처럼 처리합니다.

리디렉션 모드는 **HTTP** 및 **HTTPS**를 제외한 모든 서비스에서 작동합니다. **HTTP** 및 **HTTPS** 서비스의 경우 **HTTP** 프록시 모드를 사용하십시오. IP 주소 또는 도메인 이름이 있는 **TCP** 기반 서비스는 **DNS** 프록시 모드를 사용하십시오.

### 21.7.1.2. 송신 라우터 Pod 구현

송신 라우터 Pod 설정은 초기화 컨테이너에서 수행합니다. 해당 컨테이너는 **macvlan** 인터페이스를 구성하고 **iptables** 규칙을 설정할 수 있도록 권한 있는 컨텍스트에서 실행됩니다. 초기화 컨테이너는 **iptables** 규칙 설정을 완료한 후 종료됩니다. 그런 다음 송신 라우터 포드는 컨테이너를 실행하여 송신 라우터 트래픽을 처리합니다. 사용되는 이미지는 송신 라우터 모드에 따라 다릅니다.

환경 변수는 송신 라우터 이미지에서 사용하는 주소를 결정합니다. 이미지는 IP 주소로 **EGRESS\_SOURCE**를, 게이트웨이 IP 주소로 **EGRESS\_GATEWAY**를 사용하도록 **macvlan** 인터페이스를 구성합니다.

**NAT(Network Address Translation)** 규칙은 **TCP** 또는 **UDP** 포트에 있는 Pod의 클러스터 IP 주소에 대한 연결이 **EGRESS\_DESTINATION** 변수에서 지정하는 IP 주소의 동일한 포트로 리디렉션되도록 설정됩니다.

클러스터의 일부 노드만 지정된 소스 IP 주소를 요청하고 지정된 게이트웨이를 사용할 수 있는 경우 허용 가능한 노드를 나타내는 `nodeName` 또는 `nodeSelector`를 지정할 수 있습니다.

### 21.7.1.3. 배포 고려 사항

송신 라우터 Pod는 노드의 기본 네트워크 인터페이스에 추가 IP 주소와 MAC 주소를 추가합니다. 따라서 추가 주소를 허용하도록 하이퍼바이저 또는 클라우드 공급자를 구성해야 할 수 있습니다.

#### Red Hat OpenStack Platform (RHOSP)

RHOSP에서 OpenShift Container Platform을 배포하는 경우 OpenStack 환경에서 송신 라우터 포드의 IP 및 MAC 주소의 트래픽을 허용해야 합니다. 트래픽을 허용하지 않으면 통신이 실패합니다.

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

#### RHV(Red Hat Virtualization)

RHV를 사용하는 경우 가상 네트워크 인터페이스 컨트롤러(vNIC)에 대해 No Network Filter (네트워크 필터 없음)를 선택해야 합니다.

#### VMware vSphere

VMware vSphere를 사용하는 경우 vSphere 표준 스위치 보안을 위한 VMware 설명서를 참조하십시오. vSphere Web Client에서 호스트 가상 스위치를 선택하여 VMware vSphere 기본 설정을 보고 변경합니다.

특히 다음이 활성화되어 있는지 확인하십시오.

- **MAC 주소 변경**
- **위조된 전송**
- **무차별 모드 작동**

### 21.7.1.4. 장애 조치 구성

다운타임을 방지하기 위해 다음 예와 같이 Deployment 리소스를 사용하여 송신 라우터 Pod를 배포

할 수 있습니다. 예제 배포를 위해 새 **Service** 오브젝트를 생성하려면 `oc expose deployment/egress-demo-controller` 명령을 사용하십시오.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 ①
  selector:
    matchLabels:
      name: egress-router
  template:
    metadata:
      name: egress-router
    labels:
      name: egress-router
    annotations:
      pod.network.openshift.io/assign-macvlan: "true"
  spec: ②
    initContainers:
      ...
    containers:
      ...

```

①

항상 하나의 **Pod**만 지정된 송신 소스 IP 주소를 사용할 수 있으므로 복제본이 1로 설정되어 있는지 확인합니다. 이는 하나의 라우터 사본만 노드에서 실행됨을 의미합니다.

②

송신 라우터 **Pod**에 대한 **Pod** 오브젝트 템플릿을 지정합니다.

### 21.7.2. 추가 리소스

- [리디렉션 모드에서 송신 라우터 배포](#)
- [HTTP 프록시 모드에서 송신 라우터 배포](#)
- [DNS 프록시 모드에서 송신 라우터 배포](#)

## 21.8. 리디렉션 모드에서 송신 라우터 **POD** 배포



클러스터 관리자는 트래픽을 지정된 대상 IP 주소로 리디렉션하도록 구성된 송신 라우터 Pod를 배포할 수 있습니다.

### 21.8.1. 리디렉션 모드에 대한 송신 라우터 Pod 사양

Pod 오브젝트에서 송신 라우터 Pod에 대한 구성을 정의합니다. 다음 YAML은 리디렉션 모드에서 송신 라우터 Pod를 구성하는 데 필요한 필드를 나타냅니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
  securityContext:
    privileged: true
  env:
  - name: EGRESS_SOURCE 2
    value: <egress_router>
  - name: EGRESS_GATEWAY 3
    value: <egress_gateway>
  - name: EGRESS_DESTINATION 4
    value: <egress_destination>
  - name: EGRESS_ROUTER_MODE
    value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift4/ose-pod
```

1

주석은 OpenShift Container Platform이 기본 NIC(네트워크 인터페이스 컨트롤러)에서 macvlan 네트워크 인터페이스를 생성하고 해당 macvlan 인터페이스를 Pod의 네트워크 네임스페이스로 이동하도록 지시합니다. "true" 값을 따옴표로 묶어야 합니다. OpenShift Container Platform에서 다른 NIC 인터페이스에서 macvlan 인터페이스를 생성하려면 주석 값을 해당 인터페이스 이름으로 설정합니다. 예를 들면 eth1입니다.

2

송신 라우터 Pod에서 사용하도록 예약된 노드가 있는 물리적 네트워크의 IP 주소입니다. 선택 사항: 서브넷 길이를 나타내는 /24 접미사를 포함하여 로컬 서브넷 경로를 적절하게 설정할 수 있습니다. 서브넷 길이를 지정하지 않으면 송신 라우터에서 EGRESS\_GATEWAY 변수로 지정된 호스트에 만 액세스하고 서브넷의 다른 호스트에는 액세스할 수 없습니다.

3

노드에서 사용하는 기본 게이트웨이와 동일한 값입니다.

4

트래픽을 전달할 외부 서버입니다. 이 예제를 사용하면 Pod에 대한 연결이 소스 IP 주소가 192.168.12.99인 203.0.113.25로 리디렉션됩니다.

송신 라우터 pod 사양의 예

```

apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
      - name: EGRESS_SOURCE
        value: 192.168.12.99/24
      - name: EGRESS_GATEWAY
        value: 192.168.12.1
      - name: EGRESS_DESTINATION
        value: |
          80 tcp 203.0.113.25
          8080 tcp 203.0.113.26 80
          8443 tcp 203.0.113.26 443
          203.0.113.27
      - name: EGRESS_ROUTER_MODE
        value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift4/ose-pod

```

### 21.8.2. 송신 대상 구성 형식

송신 라우터 Pod가 리디렉션 모드로 배포되면 다음 형식 중 하나 이상을 사용하여 리디렉션 규칙을 지

정할 수 있습니다.

- **<port> <protocol> <ip\_address>** - 지정된 **<port>**로 들어오는 연결을 지정된 **<ip\_address>**의 동일한 포트에 리디렉션해야 합니다. **<protocol>**은 **tcp** 또는 **udp**입니다.
- **<port> <protocol> <ip\_address> <remote\_port>** - 연결이 **<ip\_address>**의 다른 **<remote\_port>**로 리디렉션된다는 점을 제외하고는 위와 같습니다.
- **<ip\_address>** - 마지막 줄이 단일 IP 주소인 경우 기타 포트의 모든 연결이 이 IP 주소의 해당 포트에 리디렉션됩니다. 대체 IP 주소가 없으면 기타 포트의 연결이 거부됩니다.

이어지는 예제에서는 몇 가지 규칙이 정의됩니다.

- 첫 번째 줄에서는 트래픽을 로컬 포트 80에서 203.0.113.25의 포트 80으로 리디렉션합니다.
- 두 번째 및 세 번째 줄에서는 로컬 포트 8080 및 8443을 203.0.113.26의 원격 포트 80 및 443으로 리디렉션합니다.
- 마지막 줄은 이전 규칙에 지정되지 않은 모든 포트의 트래픽과 일치합니다.

설정 예

```
80 tcp 203.0.113.25
8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443
203.0.113.27
```

### 21.8.3. 리디렉션 모드에서 송신 라우터 Pod 배포

리디렉션 모드에서는 송신 라우터 Pod가 자체 IP 주소에서 하나 이상의 대상 IP 주소로 트래픽을 리디렉션하도록 iptables 규칙을 설정합니다. 예약된 소스 IP 주소를 사용해야 하는 클라이언트 Pod는 대상 IP에 직접 연결하는 대신 송신 라우터에 연결하도록 수정해야 합니다.

### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

### 프로세스

1. 송신 라우터 **Pod**를 생성합니다.
2. 다른 **Pod**에서 송신 라우터 **Pod**의 **IP** 주소를 찾을 수 있도록 하려면 다음 예제와 같이 송신 라우터 **Pod**를 가리키는 서비스를 만듭니다.

```

apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http
      port: 80
    - name: https
      port: 443
  type: ClusterIP
  selector:
    name: egress-1

```

이제 **Pod**에서 이 서비스에 연결할 수 있습니다. 이러한 연결은 예약된 송신 **IP** 주소를 사용하여 외부 서버의 해당 포트에 리디렉션됩니다.

#### 21.8.4. 추가 리소스

- [ConfigMap](#)을 사용하여 송신 라우터 대상 매핑 구성

### 21.9. HTTP 프록시 모드에서 송신 라우터 **POD** 배포

클러스터 관리자는 지정된 **HTTP** 및 **HTTPS** 기반 서비스로 트래픽을 프록시하도록 구성된 송신 라우터 **Pod**를 배포할 수 있습니다.

#### 21.9.1. HTTP 모드에 대한 송신 라우터 **Pod** 사양

**Pod** 오브젝트에서 송신 라우터 **Pod**에 대한 구성을 정의합니다. 다음 **YAML**은 **HTTP** 모드에서 송신 라우터 **Pod**를 구성하는 데 필요한 필드를 나타냅니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
  - name: EGRESS_SOURCE 2
    value: <egress-router>
  - name: EGRESS_GATEWAY 3
    value: <egress-gateway>
  - name: EGRESS_ROUTER_MODE
    value: http-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-http-proxy
    env:
  - name: EGRESS_HTTP_PROXY_DESTINATION 4
    value: |-
      ...
      ...

```

1

주석은 **OpenShift Container Platform**이 기본 **NIC**(네트워크 인터페이스 컨트롤러)에서 **macvlan** 네트워크 인터페이스를 생성하고 해당 **macvlan** 인터페이스를 **Pod**의 네트워크 네임스페이스로 이동하도록 지시합니다. **"true"** 값을 따옴표로 묶어야 합니다. **OpenShift Container Platform**에서 다른 **NIC** 인터페이스에서 **macvlan** 인터페이스를 생성하려면 주석 값을 해당 인터페이스 이름으로 설정합니다. 예를 들면 **eth1**입니다.

2

송신 라우터 **Pod**에서 사용하도록 예약된 노드가 있는 물리적 네트워크의 **IP** 주소입니다. 선택 사항: 서브넷 길이를 나타내는 **/24** 접미사를 포함하여 로컬 서브넷 경로를 적절하게 설정할 수 있습니다. 서브넷 길이를 지정하지 않으면 송신 라우터에서 **EGRESS\_GATEWAY** 변수로 지정된 호스트에만 액세스하고 서브넷의 다른 호스트에는 액세스할 수 없습니다.

3

노드에서 사용하는 기본 게이트웨이와 동일한 값입니다.

4

프록시 구성 방법을 지정하는 문자열 또는 여러 줄로 된 **YAML** 문자열입니다. 이 문자열은 **init** 컨테이너의 다른 환경 변수가 아닌 **HTTP** 프록시 컨테이너의 환경 변수로 지정됩니다.

### 21.9.2. 송신 대상 구성 형식

송신 라우터 **Pod**가 **HTTP** 프록시 모드로 배포되면 다음 형식 중 하나 이상을 사용하여 리디렉션 규칙을 지정할 수 있습니다. 구성의 각 줄은 허용 또는 거부할 하나의 연결 그룹을 지정합니다.

- **IP** 주소는 **192.168.1.1**과 같은 해당 **IP** 주소에 대한 연결을 허용합니다.
- **CIDR** 범위는 **192.168.1.0/24**와 같은 해당 **CIDR** 범위에 대한 연결을 허용합니다.
- 호스트 이름을 사용하면 **www.example.com**과 같은 해당 호스트에 대한 프록시를 허용합니다.
- **\***으로 시작하는 도메인 이름은 해당 도메인 및 **\*.example.com**과 같은 모든 하위 도메인에 대한 프록시 사용을 허용합니다.
- 위의 일치 식 뒤에 **!**가 있으면 연결이 거부됩니다.
- 마지막 줄이 **\***이면 명시적으로 거부되지 않은 모든 것이 허용됩니다. 또는 허용되지 않은 모든 것이 거부됩니다.

**\***를 사용하여 모든 원격 대상에 대한 연결을 허용할 수도 있습니다.

설정 예

```
!*example.com
!192.168.1.0/24
192.168.2.1
```

| \*

### 21.9.3. HTTP 프록시 모드에서 송신 라우터 Pod 배포

HTTP 프록시 모드에서는 송신 라우터 Pod가 포트 8080에서 HTTP 프록시로 실행됩니다. 이 모드는 HTTP 기반 또는 HTTPS 기반 서비스에 연결하는 클라이언트에 대해서만 작동하지만 일반적으로 클라이언트 Pod를 덜 변경해야 작동합니다. 대부분의 프로그램은 환경 변수를 설정하여 HTTP 프록시를 사용하도록 지시할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

1. 송신 라우터 Pod를 생성합니다.
2. 다른 Pod에서 송신 라우터 Pod의 IP 주소를 찾을 수 있도록 하려면 다음 예제와 같이 송신 라우터 Pod를 가리키는 서비스를 만듭니다.

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http-proxy
      port: 8080 ①
  type: ClusterIP
selector:
  name: egress-1
```

①

http 포트가 8080으로 설정되어 있는지 확인하십시오.

- 3.

HTTP 프록시를 사용하도록 클라이언트 Pod(송신 프록시 Pod가 아님)를 구성하려면 `http_proxy` 또는 `https_proxy` 변수를 설정합니다.

```

apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  containers:
    env:
      - name: http_proxy
        value: http://egress-1:8080/ 1
      - name: https_proxy
        value: http://egress-1:8080/
    ...

```

1

이전 단계에서 생성한 서비스입니다.



참고

모든 설정에 `http_proxy` 및 `https_proxy` 환경 변수를 사용할 필요는 없습니다. 위 방법으로 유효한 설정이 생성되지 않으면 Pod에서 실행 중인 틀이나 소프트웨어에 대한 설명서를 참조하십시오.

#### 21.9.4. 추가 리소스

- 

[ConfigMap을 사용하여 송신 라우터 대상 매핑 구성](#)

#### 21.10. DNS 프록시 모드에서 송신 라우터 POD 배포

클러스터 관리자는 지정된 DNS 이름 및 IP 주소로 트래픽을 프록시하도록 구성된 송신 라우터 Pod를 배포할 수 있습니다.

##### 21.10.1. DNS 모드에 대한 송신 라우터 Pod 사양

Pod 오브젝트에서 송신 라우터 Pod에 대한 구성을 정의합니다. 다음 YAML은 DNS 모드에서 송신 라우터 Pod를 구성하는 데 필요한 필드를 나타냅니다.



```

apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
  env:
  - name: EGRESS_SOURCE ❷
    value: <egress-router>
  - name: EGRESS_GATEWAY ❸
    value: <egress-gateway>
  - name: EGRESS_ROUTER_MODE
    value: dns-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-dns-proxy
    securityContext:
      privileged: true
    env:
  - name: EGRESS_DNS_PROXY_DESTINATION ❹
    value: |-
      ...
  - name: EGRESS_DNS_PROXY_DEBUG ❺
    value: "1"
    ...
  ...

```

❶

주석은 **OpenShift Container Platform**이 기본 **NIC**(네트워크 인터페이스 컨트롤러)에서 **macvlan** 네트워크 인터페이스를 생성하고 해당 **macvlan** 인터페이스를 **Pod**의 네트워크 네임스페이스로 이동하도록 지시합니다. **"true"** 값을 따옴표로 묶어야 합니다. **OpenShift Container Platform**에서 다른 **NIC** 인터페이스에서 **macvlan** 인터페이스를 생성하려면 주석 값을 해당 인터페이스 이름으로 설정합니다. 예를 들면 **eth1**입니다.

❷

송신 라우터 **Pod**에서 사용하도록 예약된 노드가 있는 물리적 네트워크의 **IP** 주소입니다. 선택사항: 서브넷 길이를 나타내는 **/24** 접미사를 포함하여 로컬 서브넷 경로를 적절하게 설정할 수 있습니다. 서브넷 길이를 지정하지 않으면 송신 라우터에서 **EGRESS\_GATEWAY** 변수로 지정된 호스트에만 액세스하고 서브넷의 다른 호스트에는 액세스할 수 없습니다.

❸

노드에서 사용하는 기본 게이트웨이와 동일한 값입니다.

4

하나 이상의 프록시 대상 목록을 지정합니다.

5

선택사항: **DNS** 프록시 로그 출력을 **stdout**로 출력하도록 지정합니다.

### 21.10.2. 송신 대상 구성 형식

라우터가 **DNS** 프록시 모드에서 배포되면 포트 및 대상 매핑 목록을 지정합니다. 대상은 **IP** 주소 또는 **DNS** 이름일 수 있습니다.

송신 라우터 **Pod**는 포트 및 대상 매핑을 지정하기 위해 다음 형식을 지원합니다.

#### 포트 및 원격 주소

두 가지 필드 형식인 `<port> <remote_address>`를 사용하여 소스 포트와 대상 호스트를 지정할 수 있습니다.

호스트는 **IP** 주소 또는 **DNS** 이름일 수 있습니다. **DNS** 이름을 제공하면 런타임에 **DNS**를 확인합니다. 지정된 호스트의 경우 프록시는 대상 호스트 **IP** 주소에 연결할 때 대상 호스트의 지정된 소스 포트에 연결합니다.

#### 포트 및 원격 주소 쌍의 예

```
80 172.16.12.11
100 example.com
```

#### 포트, 원격 주소, 원격 포트

세 가지 필드 형식인 `<port> <remote_address> <remote_port>`를 사용하여 소스 포트, 대상 호스트, 대상 포트를 지정할 수 있습니다.

세 가지 필드 형식은 대상 포트가 소스 포트와 다를 수 있다는 점을 제외하고 두 가지 필드 버전과 동일하게 작동합니다.

포트, 원격 주소, 원격 포트의 예

```
8080 192.168.60.252 80
8443 web.example.com 443
```

### 21.10.3. DNS 프록시 모드에서 송신 라우터 Pod 배포

DNS 프록시 모드에서는 송신 라우터 Pod가 자체 IP 주소에서 하나 이상의 대상 IP 주소로 TCP 기반 서비스의 DNS 프록시 역할을 합니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

프로세스

1. 송신 라우터 Pod를 생성합니다.
2. 송신 라우터 Pod에 대한 서비스를 생성합니다.
  - a. 다음 YAML 정의가 포함된 **egress-router-service.yaml** 파일을 생성합니다. **spec.ports**를 **EGRESS\_DNS\_PROXY\_DESTINATION** 환경 변수에 대해 이전에 정의한 포트 목록으로 설정합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
    ...
```

```

type: ClusterIP
selector:
  name: egress-dns-proxy

```

예를 들면 다음과 같습니다.

```

apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
    - name: con1
      protocol: TCP
      port: 80
      targetPort: 80
    - name: con2
      protocol: TCP
      port: 100
      targetPort: 100
  type: ClusterIP
  selector:
    name: egress-dns-proxy

```

b.

서비스를 생성하려면 다음 명령을 입력합니다.

```
$ oc create -f egress-router-service.yaml
```

이제 **Pod**에서 이 서비스에 연결할 수 있습니다. 이러한 연결은 예약된 송신 **IP** 주소를 사용하여 외부 서버의 해당 포트에 프록시로 연결됩니다.

#### 21.10.4. 추가 리소스

- 

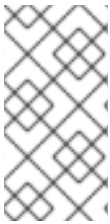
[ConfigMap을 사용하여 송신 라우터 대상 매핑 구성](#)

#### 21.11. 구성 맵에서 송신 라우터 **POD** 대상 목록 구성

클러스터 관리자는 송신 라우터 **Pod**에 대한 대상 매핑을 지정하는 **ConfigMap** 오브젝트를 정의할 수 있습니다. 구체적인 구성 형식은 송신 라우터 **Pod** 유형에 따라 다릅니다. 형식에 대한 자세한 내용은 해당 송신 라우터 **Pod**에 대한 설명서를 참조하십시오.

##### 21.11.1. 구성 맵을 사용하여 송신 라우터 대상 매핑 구성

대규모 또는 자주 변경되는 대상 매핑 집합의 경우 구성 맵을 사용하여 목록을 외부에서 관리할 수 있습니다. 이 접근 방식의 장점은 구성 맵을 편집할 수 있는 권한을 **cluster-admin** 권한이 없는 사용자에게 위임할 수 있다는 점입니다. 송신 라우터 **Pod**에는 권한 있는 컨테이너가 필요하기 때문에 **cluster-admin** 권한이 없는 사용자는 **Pod** 정의를 직접 편집할 수 없습니다.



#### 참고

송신 라우터 **Pod**는 구성 맵이 변경될 때 자동으로 업데이트되지 않습니다. 업데이트하려면 송신 라우터 **Pod**를 재시작해야 합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 프로세스

1. 다음 예와 같이 송신 라우터 **Pod**에 대한 매핑 데이터가 포함된 파일을 만듭니다.

```
# Egress routes for Project "Test", version 3

80 tcp 203.0.113.25

8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

이 파일에 빈 줄과 주석을 넣을 수 있습니다.

2. 파일에서 **ConfigMap** 오브젝트를 만듭니다.

```
$ oc delete configmap egress-routes --ignore-not-found
```

```
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

이전 명령에서 **egress-routes** 값은 생성할 **ConfigMap** 오브젝트의 이름이고, **my-egress-destination.txt**는 데이터를 읽을 파일의 이름입니다.

작은 정보

다음 **YAML**을 적용하여 구성 맵을 만들 수 있습니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: egress-routes
data:
  destination: |
    # Egress routes for Project "Test", version 3

    80 tcp 203.0.113.25

    8080 tcp 203.0.113.26 80
    8443 tcp 203.0.113.26 443

    # Fallback
    203.0.113.27
```

3.

송신 라우터 **Pod** 정의를 생성하고 환경 스탠자의 **EGRESS\_DESTINATION** 필드에 **configMapKeyRef** 스탠자를 지정합니다.

```
...
env:
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
...
```

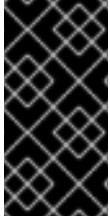
### 21.11.2. 추가 리소스

- [리디렉션 모드](#)
- [HTTP 프록시 모드](#)
- [DNS 프록시 모드](#)

## 21.12. 프로젝트에 멀티 캐스트 사용

### 21.12.1. 멀티 캐스트 정보

IP 멀티 캐스트를 사용하면 데이터가 여러 IP 주소로 동시에 브로드캐스트됩니다.



#### 중요

현재 멀티 캐스트는 고 대역폭 솔루션이 아닌 저 대역폭 조정 또는 서비스 검색에 가장 적합합니다.

**OpenShift Container Platform Pod 간 멀티 캐스트 트래픽은 기본적으로 비활성화되어 있습니다. OpenShift SDN 기본 CNI(Container Network Interface) 네트워크 공급자를 사용하는 경우 프로젝트별로 멀티 캐스트를 활성화할 수 있습니다.**

네트워크 정책 격리 모드에서 **OpenShift SDN** 네트워크 플러그인을 사용하는 경우:

- Pod에서 전송한 멀티 캐스트 패킷은 **NetworkPolicy** 오브젝트에 관계없이 프로젝트의 다른 모든 Pod로 전달됩니다. Pod는 유니 캐스트를 통해 통신할 수 없는 경우에도 멀티 캐스트를 통해 통신할 수 있습니다.
- 한 프로젝트에서 Pod가 전송한 멀티 캐스트 패킷은 프로젝트 간에 통신을 허용하는 **NetworkPolicy** 오브젝트가 있더라도 다른 프로젝트의 Pod로 전달되지 않습니다.

다중 테넌트 격리 모드에서 **OpenShift SDN** 네트워크 플러그인을 사용하는 경우:

- Pod에서 전송한 멀티 캐스트 패킷은 프로젝트의 다른 모든 Pod로 전달됩니다.
- 한 프로젝트에서 Pod가 전송한 멀티 캐스트 패킷은 각 프로젝트가 함께 결합되고 각 참여 프로젝트에서 멀티 캐스트가 활성화된 경우에만 다른 프로젝트의 Pod로 전달됩니다.

### 21.12.2. Pod 간 멀티 캐스트 활성화

프로젝트의 Pod 간 멀티 캐스트를 활성화할 수 있습니다.

## 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

## 프로세스

- 다음 명령을 실행하여 프로젝트에 대한 멀티 캐스트를 활성화합니다. 멀티 캐스트를 활성화하려는 프로젝트의 네임스페이스로 **<namespace>**를 바꿉니다.

```
$ oc annotate netnamespace <namespace> |
netnamespace.network.openshift.io/multicast-enabled=true
```

## 검증

프로젝트에 멀티 캐스트가 활성화되어 있는지 확인하려면 다음 절차를 완료합니다.

1. 멀티 캐스트를 활성화한 프로젝트로 현재 프로젝트를 변경합니다. **<project>**를 프로젝트 이름으로 바꿉니다.

```
$ oc project <project>
```

2. 멀티 캐스트 수신자 역할을 할 **pod**를 만듭니다.

```
$ 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/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat hostname && sleep inf"]
    ports:
      - containerPort: 30102
        name: mlistener
        protocol: UDP
EOF
```



3.

멀티 캐스트 발신자 역할을 할 pod를 만듭니다.

```
$ 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/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat && sleep inf"]
EOF
```

4.

새 터미널 창 또는 탭에서 멀티 캐스트 리스너를 시작합니다.

a.

Pod의 IP 주소를 가져옵니다.

```
$ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
```

b.

다음 명령을 입력하여 멀티 캐스트 리스너를 시작합니다.

```
$ oc exec mlistener -i -t -- \
  socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
EXEC:hostname
```

5.

멀티 캐스트 송신기를 시작합니다.

a.

Pod 네트워크 IP 주소 범위를 가져옵니다.

```
$ CIDR=$(oc get Network.config.openshift.io cluster \
  -o jsonpath='{.status.clusterNetwork[0].cidr}')
```

b.

멀티 캐스트 메시지를 보내려면 다음 명령을 입력합니다.

```
$ oc exec msender -i -t -- \
  /bin/bash -c "echo | socat STDIO UDP4-
  DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
```

멀티 캐스트가 작동하는 경우 이전 명령은 다음 출력을 반환합니다.

```
mlistener
```

## 21.13. 프로젝트에 대한 멀티 캐스트 비활성화

### 21.13.1. Pod 간 멀티 캐스트 비활성화

프로젝트의 **Pod** 간 멀티 캐스트를 비활성화할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

#### 프로세스

- 다음 명령을 실행하여 멀티 캐스트를 비활성화합니다.

```
$ oc annotate netnamespace <namespace> | 1
  netnamespace.network.openshift.io/multicast-enabled-
```

1

멀티 캐스트를 비활성화하려는 프로젝트의 **namespace**입니다.

## 21.14. OPENSHIFT SDN을 사용하여 네트워크 격리 구성

**OpenShift SDN CNI** 플러그인에 다중 테넌트 격리 모드를 사용하도록 클러스터를 구성하면 기본적으로 각 프로젝트가 격리됩니다. 다중 테넌트 격리 모드에서 다른 프로젝트의 **pod** 또는 **Service** 간에 네트워크 트래픽이 허용되지 않습니다.

두 가지 방법으로 프로젝트의 다중 테넌트 격리 동작을 변경할 수 있습니다.

- 하나 이상의 프로젝트에 참여하여 다른 프로젝트의 **pod**와 **service** 간에 네트워크 트래픽을 허용할 수 있습니다.
- 프로젝트의 네트워크 격리를 비활성화할 수 있습니다. 다른 모든 프로젝트에서 **pod** 및 **service**의 네트워크 트래픽을 수락하여 전역에서 액세스할 수 있습니다. 전역에서 액세스 가능한 프로젝트는 다른 모든 프로젝트의 **pod** 및 **service**에 액세스할 수 있습니다.

#### 21.14.1. 사전 요구 사항

- 다중 테넌트 격리 모드에서 **OpenShift SDN CNI(Container Network Interface)** 플러그인을 사용하도록 구성된 클러스터가 있어야 합니다.

#### 21.14.2. 프로젝트 참여

두 개 이상의 프로젝트에 참여하여 다른 프로젝트의 **Pod**와 **Service** 간 네트워크 트래픽을 허용할 수 있습니다.

##### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

##### 프로세스

1. 다음 명령을 사용하여 기존 프로젝트 네트워크에 프로젝트를 결합합니다.

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

또는 특정 프로젝트 이름을 지정하는 대신 **--selector=<project\_selector>** 옵션을 사용하여 관련 레이블을 기반으로 프로젝트를 지정할 수 있습니다.

2. 선택 사항: 다음 명령을 실행하여 결합한 **Pod** 네트워크를 봅니다.

## \$ oc get netnamespaces

동일한 Pod 네트워크에 있는 프로젝트는 NETID 열에서 동일한 네트워크 ID를 보유합니다.

### 21.14.3. 프로젝트 격리

다른 프로젝트의 Pod 및 Service가 해당 Pod 및 Service에 액세스할 수 없도록 프로젝트를 격리할 수 있습니다.

#### 사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- cluster-admin 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

#### 프로세스

- 클러스터에서 프로젝트를 격리하려면 다음 명령을 실행합니다.

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

또는 특정 프로젝트 이름을 지정하는 대신 --selector=<project\_selector> 옵션을 사용하여 관련 레이블을 기반으로 프로젝트를 지정할 수 있습니다.

### 21.14.4. 프로젝트의 네트워크 격리 비활성화

프로젝트의 네트워크 격리를 비활성화할 수 있습니다.

#### 사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- cluster-admin 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

#### 프로세스

- 프로젝트에 대해 다음 명령을 실행합니다.

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

또는 특정 프로젝트 이름을 지정하는 대신 `--selector=<project_selector>` 옵션을 사용하여 관련 레이블을 기반으로 프로젝트를 지정할 수 있습니다.

## 21.15. KUBE-PROXY 설정

**Kubernetes** 네트워크 프록시(`kube-proxy`)는 각 노드에서 실행되며 **CNO(Cluster Network Operator)**에 의해 관리됩니다. `kube-proxy`는 서비스와 관련된 끝점에 대한 연결을 전달하기 위한 네트워크 규칙을 유지 관리합니다.

### 21.15.1. iptables 규칙 동기화 정보

동기화 기간은 **Kubernetes** 네트워크 프록시(`kube-proxy`)가 노드에서 `iptables` 규칙을 동기화하는 빈도를 결정합니다.

다음 이벤트 중 하나가 발생하면 동기화가 시작됩니다.

- 서비스 또는 끝점과 같은 이벤트가 클러스터에 추가되거나 클러스터에서 제거됩니다.
- 마지막 동기화 이후 시간이 `kube-proxy`에 대해 정의된 동기화 기간을 초과합니다.

### 21.15.2. kube-proxy 구성 매개변수

다음 `kubeProxyConfig` 매개변수를 수정할 수 있습니다.



참고

**OpenShift Container Platform 4.3** 이상에서는 성능이 개선되어 더 이상 `iptablesSyncPeriod` 매개변수를 조정할 필요가 없습니다.

표 21.2. 매개변수

매개변수	설명	값	기본
<b>iptablesSyncPeriod</b>	<b>iptables</b> 규칙의 새로 고침 간격으로,	<b>30s</b> 또는 <b>2m</b> 과 같은 시간 간격입니다. 유효 접미사로 <b>s, m, h</b> 가 있으며, 자세한 설명은 <a href="#">Go time 패키지</a> 문서를 참조하십시오.	<b>30s</b>
<b>proxyArguments.iptables-min-sync-period</b>	<b>iptables</b> 규칙을 새로 고치기 전 최소 기간입니다. 이 매개변수를 이용하면 새로 고침 간격이 너무 짧지 않도록 조정할 수 있습니다. 기본적으로 새로 고침은 <b>iptables</b> 규칙에 영향을 주는 변경이 발생하는 즉시 시작됩니다.	<b>30s</b> 또는 <b>2m</b> 과 같은 시간 간격입니다. 유효 접미사로 <b>s, m</b> 및 <b>h</b> 가 있으며, 자세한 설명은 <a href="#">Go time 패키지</a> 를 참조하십시오.	<b>0s</b>

### 21.15.3. kube-proxy 구성 수정

클러스터의 **Kubernetes** 네트워크 프록시 구성을 수정할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할을 사용하여 실행 중인 클러스터에 로그인합니다.

#### 프로세스

1. 다음 명령을 실행하여 **Network.operator.openshift.io CR(사용자 정의 리소스)**을 편집합니다.

```
$ oc edit network.operator.openshift.io cluster
```

2. 다음 예제 **CR**과 같이 **kube-proxy** 구성을 변경하여 **CR**의 **kubeProxyConfig** 매개변수를 수정합니다.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  kubeProxyConfig:
```

```

iptablesSyncPeriod: 30s
proxyArguments:
  iptables-min-sync-period: ["30s"]

```

3.

파일을 저장하고 텍스트 편집기를 종료합니다.

파일을 저장하고 편집기를 종료하면 **oc** 명령에 의해 구문의 유효성이 검사됩니다. 수정 사항에 구문 오류가 포함되어 있으면 편집기가 파일을 열고 오류 메시지를 표시합니다.

4.

다음 명령을 입력하여 구성 업데이트를 확인하십시오.

```
$ oc get networks.operator.openshift.io -o yaml
```

출력 예

```

apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    clusterNetwork:
      - cidr: 10.128.0.0/14
        hostPrefix: 23
    defaultNetwork:
      type: OpenShiftSDN
    kubeProxyConfig:
      iptablesSyncPeriod: 30s
      proxyArguments:
        iptables-min-sync-period:
          - 30s
    serviceNetwork:
      - 172.30.0.0/16
  status: {}
kind: List

```

5.

선택 사항: **Cluster Network Operator**가 구성 변경을 승인했는지 확인하려면 다음 명령을 입력합니다.

```
$ oc get clusteroperator network
```

출력 예

<b>NAME</b>	<b>VERSION</b>	<b>AVAILABLE</b>	<b>PROGRESSING</b>	<b>DEGRADED</b>	<b>SINCE</b>
<i>network</i>	<i>4.1.0-0.9</i>	<i>True</i>	<i>False</i>	<i>False</i>	<i>1m</i>

구성 업데이트가 성공적으로 적용되면 **AVAILABLE** 필드는 **True**입니다.



## 22장. OVN-KUBERNETES 기본 CNI 네트워크 공급자

### 22.1. OVN-KUBERNETES 기본 CNI(CONTAINER NETWORK INTERFACE) 네트워크 공급자 정보

OpenShift Container Platform 클러스터는 pod 및 service 네트워크에 가상화된 네트워크를 사용합니다. OVN-Kubernetes CNI(Container Network Interface) 플러그인은 기본 클러스터 네트워크의 네트워크 공급자입니다. OVN-Kubernetes는 OVN(Open Virtual Network)을 기반으로 하며 오버레이 기반 네트워킹 구현을 제공합니다. OVN-Kubernetes 네트워크 공급자를 사용하는 클러스터도 각 노드에서 OVS(Open vSwitch)를 실행합니다. OVN은 각 노드에서 선언된 네트워크 구성을 구현하도록 OVS를 구성합니다.

OVN-Kubernetes는 단일 노드 OpenShift 배포 전용 기본 네트워킹 솔루션입니다.

#### 22.1.1. OVN-Kubernetes 기능

OVN-Kubernetes CNI(Container Network Interface) 클러스터 네트워크 공급자는 다음 기능을 구현합니다.

- OVN(Open Virtual Network)을 사용하여 네트워크 트래픽 흐름을 관리합니다. OVN은 커뮤니티에서 개발한 벤더와 무관한 네트워크 가상화 솔루션입니다.
- 수신 및 송신 규칙을 포함한 Kubernetes 네트워크 정책 지원을 구현합니다.
- VXLAN 대신 Geneve(Generic Network Virtualization Encapsulation) 프로토콜을 사용하여 노드 간에 오버레이 네트워크를 만듭니다.

#### 22.1.2. 지원되는 기본 CNI 네트워크 공급자 기능 매트릭스

OpenShift Container Platform은 기본 CNI(Container Network Interface) 네트워크 공급자를 위해 OpenShift SDN 및 OVN-Kubernetes의 두 가지 지원 옵션을 제공합니다. 다음 표는 두 네트워크 공급자 모두에 대한 현재 기능 지원을 요약합니다.

표 22.1. 기본 CNI 네트워크 공급자 기능 비교

기능	OVN-Kubernetes	OpenShift SDN
송신 IP	지원됨	지원됨

기능	OVN-Kubernetes	OpenShift SDN
송신 방화벽 [1]	지원됨	지원됨
송신 라우터	지원됨 [2]	지원됨
하이브리드 네트워킹	지원됨	지원되지 않음
클러스터 내 통신에 대한 IPsec 암호화	지원됨	지원되지 않음
IPv6	지원됨 [3]	지원되지 않음
Kubernetes 네트워크 정책	지원됨	지원됨
Kubernetes 네트워크 정책 로그	지원됨	지원되지 않음
하드웨어 오프로드	지원됨	지원되지 않음
멀티 캐스트	지원됨	지원됨

1. 송신 방화벽은 **OpenShift SDN**에서 송신 네트워크 정책이라고도 합니다. 이것은 네트워크 정책 송신과 동일하지 않습니다.
2. **OVN-Kubernetes**용 송신 라우터는 리디렉션 모드만 지원합니다.
3. **IPv6**는 베어 메탈 클러스터에서만 지원됩니다.

### 22.1.3. OVN-Kubernetes 제한 사항

**OVN-Kubernetes CNI(Container Network Interface)** 클러스터 네트워크 공급자에는 다음과 같은 제한 사항이 있습니다.

- **sessionAffinityConfig.clientIP.timeoutSeconds** 서비스는 **OpenShift OVN** 환경에는 적용되지 않지만 **OpenShift SDN** 환경에서는 적용되지 않습니다. 이로 인해 사용자가 **OpenShift SDN**에서 **OVN**으로 마이그레이션하기가 어려울 수 있습니다.
- 듀얼 스택 네트워킹용으로 구성된 클러스터의 경우 **IPv4** 및 **IPv6** 트래픽 모두 기본 게이트웨이와 동일한 네트워크 인터페이스를 사용해야 합니다. 이 요구 사항이 충족되지 않으면 **ovnkube-node** 데몬 세트의 호스트의 Pod가 **CrashLoopBackOff** 상태가 됩니다. **oc get pod -n**

`openshift-ovn-kubernetes -l app=ovnkube-node -o yaml` 과 같은 명령으로 Pod를 표시하는 경우 `status` 필드에는 다음 출력에 표시된 대로 기본 게이트웨이에 대한 두 개 이상의 메시지가 포함됩니다.

```
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
```

유일한 해결 방법은 두 IP 제품군이 모두 기본 게이트웨이에 동일한 네트워크 인터페이스를 사용하도록 호스트 네트워킹을 재구성하는 것입니다.

- 듀얼 스택 네트워킹용으로 구성된 클러스터의 경우 IPv4 및 IPv6 라우팅 테이블에는 기본 게이트웨이가 포함되어야 합니다. 이 요구 사항이 충족되지 않으면 `ovnkube-node` 데몬 세트의 호스트의 Pod가 `CrashLoopBackOff` 상태가 됩니다. `oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml` 과 같은 명령으로 Pod를 표시하는 경우 `status` 필드에는 다음 출력에 표시된 대로 기본 게이트웨이에 대한 두 개 이상의 메시지가 포함됩니다.

```
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
```

유일한 해결 방법은 두 IP 제품군에 기본 게이트웨이를 포함하도록 호스트 네트워킹을 재구성하는 것입니다.

## 추가 리소스

- [프로젝트에 대한 송신 방화벽 구성](#)
- [네트워크 정책 정의](#)
- [네트워크 정책 이벤트 로깅](#)
- [프로젝트에 멀티 캐스트 사용](#)
- [IPsec 암호화 구성](#)

- [Network \[operator.openshift.io/v1\]](https://operator.openshift.io/v1)

## 22.2. OPENSIFT SDN 클러스터 네트워크 공급자에서 마이그레이션

클러스터 관리자는 **OpenShift SDN CNI 클러스터 네트워크 공급자에서 OVN-Kubernetes CNI (Container Network Interface) 클러스터 네트워크 공급자로 마이그레이션**할 수 있습니다.

OVN-Kubernetes에 대한 자세한 내용은 [OVN-Kubernetes 네트워크 공급자 정보](#)를 읽어보십시오.

### 22.2.1. OVN-Kubernetes 네트워크 공급자로 마이그레이션

OVN-Kubernetes CNI(Container Network Interface) 클러스터 네트워크 공급자로 마이그레이션하는 것은 클러스터에 연결할 수 없는 몇 가지 중단 시간을 포함하는 수동 프로세스입니다. 롤백 절차가 제공되지만 마이그레이션은 단방향 프로세스로 설정됩니다.

다음 플랫폼에서 OVN-Kubernetes 클러스터 네트워크 공급자로의 마이그레이션이 지원됩니다.

- 베어 메탈 하드웨어
- AWS(Amazon Web Services)
- GCP(Google Cloud Platform)
- Microsoft Azure
- Red Hat OpenStack Platform (RHOSP)
- RHV(Red Hat Virtualization)
- VMware vSphere

### 22.2.1.1. OVN-Kubernetes 네트워크 공급자로 마이그레이션에 대한 고려 사항

**OpenShift Container Platform** 클러스터에 150개 이상의 노드가 있는 경우 **OVN-Kubernetes** 네트워크 플러그인으로 마이그레이션하기 위한 지원 케이스를 엽니다.

노드에 할당된 서브넷과 개별 포드에 할당된 IP 주소는 마이그레이션 중에 유지되지 않습니다.

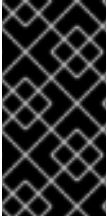
**OVN-Kubernetes** 네트워크 공급자는 **OpenShift SDN** 네트워크 공급자에 있는 많은 기능을 구현하지 만 구성은 동일하지 않습니다.

- 클러스터에서 다음 **OpenShift SDN** 기능을 사용하는 경우 **OVN-Kubernetes**에서 동일한 기능을 수동으로 구성해야 합니다.
  - 네임스페이스 격리
  - 송신 IP 주소
  - 송신 네트워크 정책
  - 송신 라우터 포드
  - 멀티 캐스트
- 클러스터에서 100.64.0.0/16 IP 주소 범위의 모든 부분을 사용하는 경우 이 IP 주소 범위를 내적으로 사용하므로 **OVN-Kubernetes**로 마이그레이션할 수 없습니다.

다음 섹션에서는 **OVN-Kubernetes**와 **OpenShift SDN**의 앞서 언급한 기능 간 구성의 차이점을 설명합니다.

#### 네임스페이스 격리

**OVN-Kubernetes**는 네트워크 정책 격리 모드만 지원합니다.



**중요**

클러스터가 다중 테넌트 또는 서브넷 격리 모드에서 구성된 **OpenShift SDN**을 사용하는 경우 **OVN-Kubernetes** 네트워크 공급자로 마이그레이션할 수 없습니다.

**송신 IP 주소**

**OVN-Kubernetes**와 **OpenShift SDN** 간의 송신 IP 주소를 구성하는 데 있어서 차이점은 다음 표에 설명되어 있습니다.

**표 22.2. 송신 IP 주소 구성의 차이점**

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> <li>● <b>EgressIPs</b> 오브젝트 생성</li> <li>● <b>Node</b> 오브젝트에 주석 추가</li> </ul>	<ul style="list-style-type: none"> <li>● <b>NetNamespace</b> 오브젝트 패치</li> <li>● <b>HostSubnet</b> 오브젝트 패치</li> </ul>

**OVN-Kubernetes**에서 송신 IP 주소를 사용하는 방법에 대한 자세한 내용은 "**송신 IP 주소 구성**"을 참조하십시오.

**송신 네트워크 정책**

**OVN-Kubernetes**와 **OpenShift SDN** 간의 송신 방화벽이라고도 하는 송신 네트워크 정책 구성의 차이점은 다음 표에 설명되어 있습니다.

**표 22.3. 송신 네트워크 정책 구성의 차이점**

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> <li>● 네임스페이스에 <b>EgressFirewall</b> 오브젝트 생성</li> </ul>	<ul style="list-style-type: none"> <li>● 네임스페이스에서 <b>EgressNetworkPolicy</b> 오브젝트 생성</li> </ul>

**OVN-Kubernetes**에서 송신 방화벽을 사용하는 방법에 대한 자세한 내용은 "**프로젝트에 대한 송신 방화벽 구성**"을 참조하십시오.

**송신 라우터 Pod**

**OVN-Kubernetes**는 리디렉션 모드에서 송신 라우터 pod를 지원합니다. **OVN-Kubernetes**는 HTTP 프록시 모드 또는 DNS 프록시 모드에서 송신 라우터 Pod를 지원하지 않습니다.

**Cluster Network Operator**를 사용하여 송신 라우터를 배포할 때 송신 라우터 **Pod**를 호스팅하는 데 사용되는 노드를 제어하기 위해 노드 선택기를 지정할 수 없습니다.

## 멀티 캐스트

**OVN-Kubernetes** 및 **OpenShift SDN**에서 멀티 캐스트 트래픽 활성화의 차이점은 다음 표에 설명되어 있습니다.

표 22.4. 멀티 캐스트 구성의 차이점

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> <li>● <b>Namespace</b> 오브젝트에 주석 추가</li> </ul>	<ul style="list-style-type: none"> <li>● <b>NetNamespace</b> 오브젝트에 주석 추가</li> </ul>

**OVN-Kubernetes**에서 멀티 캐스트를 사용하는 방법에 대한 자세한 내용은 "**프로젝션에 멀티 캐스트 사용**"을 참조하십시오.

## 네트워크 정책

**OVN-Kubernetes**는 **networking.k8s.io/v1 API** 그룹에서 **Kubernetes NetworkPolicy API**를 완전히 지원합니다. **OpenShift SDN**에서 마이그레이션할 때 네트워크 정책에 변경 사항이 필요하지 않습니다.

### 22.2.1.2. 마이그레이션 프로세스의 작동 방식

다음 표는 프로세스의 사용자 시작 단계와 마이그레이션이 수행하는 작업 간에 분할하여 마이그레이션 프로세스를 요약합니다.

표 22.5. OpenShift SDN에서 OVN-Kubernetes로 마이그레이션

사용자 시작 단계	마이그레이션 활동
<p><b>cluster</b>라는 <b>Network.operator.openshift.io</b> CR(사용자 정의 리소스)의 <b>migration</b> 필드를 <b>OVN-Kubernetes</b>로 설정합니다. 값으로 설정하기 전에 <b>migration</b> 필드가 <b>null</b>인지 확인합니다.</p>	<p><b>CNO(Cluster Network Operator)</b>  <b>cluster</b>라는 <b>Network.config.openshift.io</b> CR의 상태를 적절하게 업데이트합니다.</p> <p><b>Machine Config Operator (MCO)</b>            OVN-Kubernetes에 필요한 <b>systemd</b> 구성에 대한 업데이트를 몰아냅니다. MCO는 기본적으로 풀당 단일 머신을 업데이트하여 기본적으로 클러스터 크기로 마이그레이션을 늘리는 데 걸리는 총 시간을 생성합니다.</p>

사용자 시작 단계	마이그레이션 활동
<p><b>Network.config.openshift.io</b> CR의 <b>networkType</b> 필드를 업데이트합니다.</p>	<p><b>CNO</b></p> <p>다음과 같은 작업을 수행합니다.</p> <ul style="list-style-type: none"> <li>● OpenShift SDN 컨트롤 플레인 pod를 삭제합니다.</li> <li>● OVN-Kubernetes 컨트롤 플레인 pod를 배포합니다.</li> <li>● 새 클러스터 네트워크 공급자를 반영하도록 Multus 오브젝트를 업데이트합니다.</li> </ul>
<p>클러스터의 각 노드를 재부팅합니다.</p>	<p><b>Cluster</b></p> <p>노드가 재부팅되면 클러스터에서 OVN-Kubernetes 클러스터 네트워크의 Pod에 IP 주소를 할당합니다.</p>

**OpenShift SDN**으로의 롤백이 필요한 경우 다음 표에서 프로세스를 설명합니다.

표 22.6. **OpenShift SDN**으로 롤백 수행

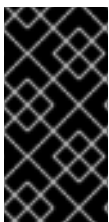
사용자 시작 단계	마이그레이션 활동
<p>MCO가 마이그레이션을 중단하지 않도록 일시 중지합니다.</p>	<p>MCO가 중지됩니다.</p>
<p><b>cluster</b> 라는 <b>Network.operator.openshift.io</b> CR(사용자 정의 리소스)의 <b>migration</b> 필드를 <b>OpenShiftSDN</b> 으로 설정합니다. 값으로 설정하기 전에 <b>migration</b> 필드가 <b>null</b>인지 확인합니다.</p>	<p><b>CNO</b></p> <p><b>cluster</b>라는 <b>Network.config.openshift.io</b> CR의 상태를 적절하게 업데이트합니다.</p>
<p><b>networkType</b> 필드를 업데이트합니다.</p>	<p><b>CNO</b></p> <p>다음과 같은 작업을 수행합니다.</p> <ul style="list-style-type: none"> <li>● OVN-Kubernetes 컨트롤 플레인 pod를 삭제합니다.</li> <li>● OpenShift SDN 컨트롤 플레인 포드를 배포합니다.</li> <li>● 새 클러스터 네트워크 공급자를 반영하도록 Multus 오브젝트를 업데이트합니다.</li> </ul>



사용자 시작 단계	마이그레이션 활동
클러스터의 각 노드를 재부팅합니다.	<b>Cluster</b> 노드가 재부팅되면 클러스터에서 OpenShift-SDN 네트워크의 pod에 IP 주소를 할당합니다.
클러스터 재부팅의 모든 노드 후에 MCO를 활성화합니다.	<b>MCO</b> OpenShift SDN에 필요한 systemd 구성에 대한 업데이트를 롤아웃합니다. MCO는 기본적으로 풀당한 번에 단일 시스템을 업데이트하므로 마이그레이션에 걸리는 총 시간은 클러스터 크기에 따라 늘어납니다.

### 22.2.2. OVN-Kubernetes 기본 CNI 네트워크 공급자로 마이그레이션

클러스터 관리자는 클러스터의 기본 CNI(Container Network Interface) 네트워크 공급자를 **OVN-Kubernetes**로 변경할 수 있습니다. 마이그레이션하는 동안 클러스터의 모든 노드를 재부팅해야 합니다.



#### 중요

마이그레이션을 수행하는 동안 클러스터를 사용할 수 없으며 워크로드가 중단될 수 있습니다. 서비스 중단이 허용되는 경우에만 마이그레이션을 수행합니다.

#### 사전 요구 사항

- 네트워크 정책 격리 모드에서 **OpenShift SDN CNI** 네트워크 공급자로 구성된 클러스터입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **etcd** 데이터베이스의 최근 백업을 사용할 수 있습니다.
- 각 노드에 대해 재부팅을 수동으로 트리거할 수 있습니다.

- 클러스터가 오류 없이 알려진 정상 상태입니다.

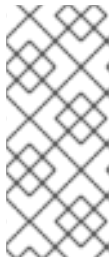
프로세스

1. 클러스터 네트워크의 구성을 백업하려면 다음 명령을 입력합니다.

```
$ oc get Network.config.openshift.io cluster -o yaml > cluster-openshift-sdn.yaml
```

2. 마이그레이션을 위해 모든 노드를 준비하려면 다음 명령을 입력하여 **Cluster Network Operator** 구성 개체에서 **migration** 필드를 설정합니다.

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
--patch '{ "spec": { "migration": { "networkType": "OVNKubernetes" } } }'
```



참고

이 단계는 **OVN-Kubernetes**를 즉시 배포하지 않습니다. 대신 **migration** 필드를 지정하면 **OVN-Kubernetes** 배포를 준비하기 위해 **MCO(Machine Config Operator)**가 클러스터의 모든 노드에 새 머신 구성을 적용합니다.

3. 선택 사항: **OVN-Kubernetes**에 대해 다음 설정을 사용자 정의하여 네트워크 인프라 요구 사항을 충족할 수 있습니다.

- 최대 전송 단위(MTU)
- Geneve(Generic Network Virtualization Encapsulation) 오버레이 네트워크 포트

이전에 명시된 설정 중 하나를 사용자 정의하려면 다음 명령을 입력하고 사용자 정의합니다. 기본값을 변경할 필요가 없는 경우 패치에서 키를 생략합니다.

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
--patch '{
  "spec":{
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "mtu":<mtu>,
        "genevePort":<port>
      }
    }
  }
}'
```

**mtu**

**Geneve** 오버레이 네트워크용 **MTU**입니다. **MTU** 값은 일반적으로 자동으로 지정되지만 클러스터의 모든 노드가 동일한 **MTU**를 사용하지 않을 때는 최소 노드 **MTU** 값에서 **100**을 뺀 값으로 명시적으로 설정해야 합니다.

**port**

**Geneve** 오버레이 네트워크용 **UDP** 포트입니다. 값을 지정하지 않으면 기본값은 **6081**입니다. 이 포트는 **OpenShift SDN**에서 사용하는 **VXLAN** 포트와 같을 수 없습니다. **VXLAN** 포트의 기본값은 **4789**입니다.

**mtu** 필드를 업데이트하는 패치 명령 예

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
    "spec":{
      "defaultNetwork":{
        "ovnKubernetesConfig":{
          "mtu":1200
        }
      }
    }
  }'
```

4.

**MCO**는 각 머신 구성 풀의 머신을 업데이트할 때 각 노드를 하나씩 재부팅합니다. 모든 노드가 업데이트될 때까지 기다려야 합니다. 다음 명령을 입력하여 머신 구성 풀 상태를 확인합니다.

```
$ oc get mcp
```

업데이트된 노드의 상태가 **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**입니다.

**참고**

기본적으로 **MCO**는 풀당 한 번에 하나의 시스템을 업데이트하므로 클러스터 크기에 따라 마이그레이션에 걸리는 총 시간이 증가합니다.

5.

호스트의 새 머신 구성 상태를 확인합니다.

a.

머신 구성 상태 및 적용된 머신 구성 이름을 나열하려면 다음 명령을 입력합니다.

```
$ oc describe node | egrep "hostname|machineconfig"
```

출력 예

```
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
```

다음 구문이 올바른지 확인합니다.

- `machineconfiguration.openshift.io/state` 필드의 값은 **Done**입니다.
  - `machineconfiguration.openshift.io/currentConfig` 필드의 값은 `machineconfiguration.openshift.io/desiredConfig` 필드의 값과 동일합니다.
- b.
- 머신 구성이 올바른지 확인하려면 다음 명령을 입력합니다.

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

여기서 `<config_name>`은 `machineconfiguration.openshift.io/currentConfig` 필드에서 머신 구성의 이름입니다.

머신 구성은 다음 업데이트를 `systemd` 구성에 포함해야 합니다.

```
ExecStart=/usr/local/bin/configure-ovs.sh OVNKubernetes
```

- c.
- 노드가 **NotReady** 상태에 있는 경우 머신 구성 데몬 포드 로그를 조사하고 오류를 해결합니다.

i.

포드를 나열하려면 다음 명령을 입력합니다.

```
$ oc get pod -n openshift-machine-config-operator
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
machine-config-controller-75f756f89d-sjp8b	1/1	Running	0	37m
machine-config-daemon-5cf4b	2/2	Running	0	43h
machine-config-daemon-7wzcd	2/2	Running	0	43h
machine-config-daemon-fc946	2/2	Running	0	43h
machine-config-daemon-g2v28	2/2	Running	0	43h
machine-config-daemon-gcl4f	2/2	Running	0	43h
machine-config-daemon-l5tnv	2/2	Running	0	43h
machine-config-operator-79d9c55d5-hth92	1/1	Running	0	37m
machine-config-server-bsc8h	1/1	Running	0	43h
machine-config-server-hklrm	1/1	Running	0	43h
machine-config-server-k9rtx	1/1	Running	0	43h

구성 데몬 포드의 이름은 다음 형식입니다. `machine-config-daemon-<seq>`. `<seq>` 값은 임의 5자 영숫자 순서입니다.

ii.

다음 명령을 입력하여 이전 출력에 표시된 첫 번째 머신 구성 데몬 포드에 대한 포드 로그를 표시합니다.

```
$ oc logs <pod> -n openshift-machine-config-operator
```

여기서 `pod`는 머신 구성 데몬 포드의 이름입니다.

iii.

이전 명령의 출력에 표시된 로그의 오류를 해결합니다.

6.

마이그레이션을 시작하려면 다음 명령 중 하나를 사용하여 OVN-Kubernetes 클러스터 네트워크 공급자를 구성합니다.

•

클러스터 네트워크 IP 주소 블록을 변경하지 않고 네트워크 공급자를 지정하려면 다음 명령을 입력합니다.

```
$ oc patch Network.config.openshift.io cluster \
--type='merge' --patch '{ "spec": { "networkType": "OVNKubernetes" } }'
```

- 

다른 클러스터 네트워크 IP 주소 블록을 지정하려면 다음 명령을 입력합니다.

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

여기서 *cidr*은 CIDR 블록이며 *prefix*는 클러스터의 각 노드에 승인된 CIDR 블록 조각입니다. **OVN-Kubernetes** 네트워크 공급자가 이 블록을 내부에서 사용하므로 **100.64.0.0/16** CIDR 블록과 겹치는 CIDR 블록을 사용할 수 없습니다.



**중요**

마이그레이션 중에 서비스 네트워크 주소 블록을 변경할 수 없습니다.

- 7.

후속 단계를 계속 진행하기 전에 **Multus** 데몬 세트 롤아웃이 완료되었는지 확인합니다.

```
$ oc -n openshift-multus rollout status daemonset/multus
```

**Multus pod**의 이름은 **multus-<xxxxx>** 형식이며 여기서 **<xxxxx>**는 임의 문자 순서입니다. 포드를 다시 시작하는 데 시간이 다소 걸릴 수 있습니다.

출력 예

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
```

...

**Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...  
daemon set "multus" successfully rolled out**

8.

마이그레이션을 완료하려면 클러스터의 각 노드를 재부팅합니다. 예를 들어 다음 예와 유사한 **bash** 스크립트를 사용할 수 있습니다. 이 스크립트는 **ssh**를 사용하여 각 호스트에 연결할 수 있고 암호를 묻지 않도록 **sudo**를 구성했다고 가정합니다.

```
#!/bin/bash

for ip in $(oc get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}')
do
  echo "reboot node $ip"
  ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

**ssh** 액세스를 사용할 수 없는 경우 인프라 공급자의 관리 포털을 통해 각 노드를 재부팅할 수 있습니다.

9.

마이그레이션이 성공했는지 확인합니다.

a.

**CNI** 클러스터 네트워크 공급자가 **OVN-Kubernetes**인지 확인하려면 다음 명령을 입력합니다. **status.networkType**의 값은 **OVNKubernetes**이어야 합니다.

```
$ oc get network.config/cluster -o jsonpath='{.status.networkType}'
```

b.

클러스터 노드가 준비 상태에 있는지 확인하려면 다음 명령을 입력합니다.

```
$ oc get nodes
```

c.

**Pod**가 오류 상태가 아닌지 확인하려면 다음 명령을 입력합니다.

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

노드의 **Pod**가 오류 상태인 경우 해당 노드를 재부팅합니다.

d.

모든 클러스터 **Operator**가 비정상적인 상태가 아닌지 확인하려면 다음 명령을 입력합니다.

```
$ oc get co
```

모든 클러스터 **Operator**의 상태는 **AVAILABLE="True"**, **PROGRESSING="False"**, **DEGRADED="False"**여야 합니다. 클러스터 **Operator**를 사용할 수 없거나 성능이 저하된 경우 자세한 내용은 클러스터 **Operator**의 로그를 확인합니다.

10.

마이그레이션이 성공하고 클러스터가 양호한 상태인 경우에만 다음 단계를 완료합니다.

a.

**CNO** 구성 오브젝트에서 마이그레이션 구성을 제거하려면 다음 명령을 입력합니다.

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
--patch '{ "spec": { "migration": null } }'
```

b.

**OpenShift SDN** 네트워크 제공자에 대한 사용자 정의 구성을 제거하려면 다음 명령을 입력합니다.

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
--patch '{ "spec": { "defaultNetwork": { "openshiftSDNConfig": null } } }'
```

c.

**OpenShift SDN** 네트워크 공급자 네임스페이스를 제거하려면 다음 명령을 입력합니다.

```
$ oc delete namespace openshift-sdn
```

### 22.2.3. 추가 리소스

- [OVN-Kubernetes 기본 CNI 네트워크 공급자에 대한 구성 매개변수](#)
- [etcd 백업](#)
- [네트워크 정책 정의](#)
- [OVN-Kubernetes 기능](#)



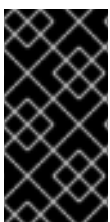
- 송신 IP 주소 구성
- 프로젝트에 대한 송신 방화벽 구성
- 프로젝트에 멀티 캐스트 사용
- **OpenShift SDN 기능**
  - 프로젝트의 송신 IP 구성
  - 프로젝트에 대한 송신 방화벽 구성
  - 프로젝트에 멀티 캐스트 사용
- **Network [operator.openshift.io/v1]**

### 22.3. OPENSIFT SDN 네트워크 공급자로 롤백

클러스터 관리자는 OVN-Kubernetes로의 마이그레이션에 실패한 경우 OVN-Kubernetes CNI 클러스터 네트워크 공급자에서 OpenShift SDN CNI(Container Network Interface) 클러스터 네트워크 공급자로 롤백할 수 있습니다.

#### 22.3.1. 기본 CNI 네트워크 공급자를 OpenShift SDN으로 롤백

클러스터 관리자는 클러스터를 OpenShift SDN CNI(Container Network Interface) 클러스터 네트워크 공급자로 롤백할 수 있습니다. 롤백 중에 클러스터의 모든 노드를 재부팅해야 합니다.



중요

OVN-Kubernetes로의 마이그레이션이 실패한 경우에만 OpenShift SDN으로 롤백하십시오.

사전 요구 사항

- **OpenShift CLI(oc)를 설치합니다.**
- **cluster-admin 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.**
- **OVN-Kubernetes CNI 클러스터 네트워크 공급자로 구성된 인프라에 설치된 클러스터입니다.**

프로세스

1. **MCO(Machine Config Operator)에서 관리하는 모든 머신 구성 풀을 중지합니다.**

- **마스터 구성 풀을 중지합니다.**

```
$ oc patch MachineConfigPool master --type='merge' --patch \
  '{"spec": {"paused": true } }'
```

- **작업자 머신 구성 풀을 중지합니다.**

```
$ oc patch MachineConfigPool worker --type='merge' --patch \
  '{"spec": {"paused": true } }'
```

2. **마이그레이션을 시작하려면 다음 명령을 입력하여 클러스터 네트워크 공급자를 다시 OpenShift SDN으로 설정합니다.**

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{"spec": {"migration": {"networkType": "OpenShiftSDN"} } }'
```

```
$ oc patch Network.config.openshift.io cluster --type='merge' \
  --patch '{"spec": {"networkType": "OpenShiftSDN"} }'
```

3. **선택 사항: OpenShift SDN에 대해 네트워크 인프라 요구 사항을 충족하도록 다음 설정을 사용자 정의할 수 있습니다.**

- **최대 전송 단위(MTU)**

## VXLAN 포트

이전에 명시된 설정 중 하나 또는 둘 다 사용자 정의하려면 사용자 정의하고 다음 명령을 입력합니다. 기본값을 변경할 필요가 없는 경우 패치에서 키를 생략합니다.

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
--patch '{
  "spec":{
    "defaultNetwork":{
      "openshiftSDNConfig":{
        "mtu":<mtu>,
        "vxlanPort":<port>
      }
    }
  }
}'
```

### mtu

**VXLAN** 오버레이 네트워크의 **MTU**입니다. **MTU** 값은 일반적으로 자동으로 지정되지만 클러스터의 모든 노드가 동일한 **MTU**를 사용하지 않을 때는 최소 노드 **MTU** 값에서 **50**을 뺀 값으로 명시적으로 설정해야 합니다.

### port

**VXLAN** 오버레이 네트워크용 **UDP** 포트입니다. 값을 지정하지 않으면 기본값은 **4789**입니다. 이 포트는 **OVN-Kubernetes**에서 사용하는 **Geneve** 포트와 같을 수 없습니다. **Geneve** 포트의 기본값은 **6081**입니다.

### 패치 명령 예

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
--patch '{
  "spec":{
    "defaultNetwork":{
      "openshiftSDNConfig":{
        "mtu":1200
      }
    }
  }
}'
```

4.

**Multus** 데몬 세트 롤아웃이 완료될 때까지 기다립니다.

```
$ oc -n openshift-multus rollout status daemonset/multus
```

**Multus** 포드의 이름은 **multus-<xxxxxx>** 형식이며 여기서 **<xxxxxx>**는 임의 문자 순서입니다. 포드를 다시 시작하는 데 시간이 다소 걸릴 수 있습니다.

출력 예

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

5.

롤백을 완료하려면 클러스터의 각 노드를 재부팅합니다. 예를 들어 다음과 유사한 **bash** 스크립트를 사용할 수 있습니다. 이 스크립트는 **ssh**를 사용하여 각 호스트에 연결할 수 있고 암호를 묻지 않도록 **sudo**를 구성했다고 가정합니다.

```
#!/bin/bash

for ip in $(oc get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}');
do
  echo "reboot node $ip"
  ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

**ssh** 액세스를 사용할 수 없는 경우 인프라 공급자의 관리 포털을 통해 각 노드를 재부팅할 수 있습니다.

6.

클러스터의 노드가 재부팅된 후 모든 머신 구성 풀을 시작합니다.

- 마스터 구성 풀을 시작합니다.

```
$ oc patch MachineConfigPool master --type='merge' --patch \
  '{"spec": {"paused": false } }'
```

- 작업자 구성 풀을 시작합니다.

```
$ oc patch MachineConfigPool worker --type='merge' --patch \
  '{"spec": {"paused": false } }'
```

■ **MCO**는 각 구성 풀에서 머신을 업데이트하므로 각 노드를 재부팅합니다.

기본적으로 **MCO**는 한 번에 풀당 단일 머신을 업데이트하므로 마이그레이션이 완료하는 데 필요한 시간은 클러스터 크기와 함께 증가합니다.

7.

호스트의 새 머신 구성 상태를 확인합니다.

a.

머신 구성 상태 및 적용된 머신 구성 이름을 나열하려면 다음 명령을 입력합니다.

```
$ oc describe node | egrep "hostname|machineconfig"
```

출력 예

```
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
```

다음 구문이 올바른지 확인합니다.

•

`machineconfiguration.openshift.io/state` 필드의 값은 **Done**입니다.

•

`machineconfiguration.openshift.io/currentConfig` 필드의 값은 `machineconfiguration.openshift.io/desiredConfig` 필드의 값과 동일합니다.

b.

머신 구성이 올바른지 확인하려면 다음 명령을 입력합니다.

```
$ oc get machineconfig <config_name> -o yaml
```

여기서 <config\_name>은 `machineconfiguration.openshift.io/currentConfig` 필드에서 머신 구성의 이름입니다.

8.

마이그레이션이 성공했는지 확인합니다.

a.

기본 CNI 네트워크 공급자가 **OVN-Kubernetes**인지 확인하려면 다음 명령을 입력합니다. `status.networkType` 값은 **OpenShiftSDN**이어야 합니다.

```
$ oc get network.config/cluster -o jsonpath='{.status.networkType}{"\n"}'
```

b.

클러스터 노드가 준비 상태에 있는지 확인하려면 다음 명령을 입력합니다.

```
$ oc get nodes
```

c.

노드가 **NotReady** 상태에 있는 경우 머신 구성 데몬 포드 로그를 조사하고 오류를 해결합니다.

i.

포드를 나열하려면 다음 명령을 입력합니다.

```
$ oc get pod -n openshift-machine-config-operator
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
<code>machine-config-controller-75f756f89d-sjp8b</code>	1/1	Running	0	37m
<code>machine-config-daemon-5cf4b</code>	2/2	Running	0	43h
<code>machine-config-daemon-7wzcd</code>	2/2	Running	0	43h
<code>machine-config-daemon-fc946</code>	2/2	Running	0	43h
<code>machine-config-daemon-g2v28</code>	2/2	Running	0	43h
<code>machine-config-daemon-gcl4f</code>	2/2	Running	0	43h
<code>machine-config-daemon-l5tnv</code>	2/2	Running	0	43h
<code>machine-config-operator-79d9c55d5-hth92</code>	1/1	Running	0	37m
<code>machine-config-server-bsc8h</code>	1/1	Running	0	43h
<code>machine-config-server-hklrm</code>	1/1	Running	0	43h
<code>machine-config-server-k9rtx</code>	1/1	Running	0	43h

구성 데몬 포드의 이름은 다음 형식입니다. `machine-config-daemon-<seq>`.  
`<seq>` 값은 임의 5자 영숫자 순서입니다.

ii.

이전 출력에 표시된 각 머신 구성 데몬 포드에 대한 포드 로그를 표시하려면 다음 명령을 입력합니다.

```
$ oc logs <pod> -n openshift-machine-config-operator
```

여기서 `pod`는 머신 구성 데몬 포드의 이름입니다.

iii.

이전 명령의 출력에 표시된 로그의 오류를 해결합니다.

d.

`Pod`가 오류 상태가 아닌지 확인하려면 다음 명령을 입력합니다.

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

노드의 `Pod`가 오류 상태인 경우 해당 노드를 재부팅합니다.

9.

마이그레이션이 성공하고 클러스터가 양호한 상태인 경우에만 다음 단계를 완료합니다.

a.

`Cluster Network Operator` 구성 오브젝트에서 마이그레이션 구성을 제거하려면 다음 명령을 입력합니다.

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
--patch '{ "spec": { "migration": null } }'
```

b.

`OVN-Kubernetes` 구성을 제거하려면 다음 명령을 입력합니다.

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
--patch '{ "spec": { "defaultNetwork": { "ovnKubernetesConfig": null } } }'
```

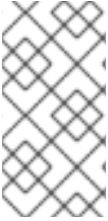
c.

`OVN-Kubernetes` 네트워크 공급자 네임스페이스를 제거하려면 다음 명령을 입력합니다.

```
$ oc delete namespace openshift-ovn-kubernetes
```

## 22.4. IPV4/IPV6 듀얼 스택 네트워킹으로 변환

클러스터 관리자는 **IPv4** 단일 스택 클러스터를 **IPv4** 및 **IPv6** 주소 제품군을 지원하는 듀얼 네트워크 클러스터 네트워크로 변환할 수 있습니다. 듀얼 스택으로 변환한 후 새로 생성된 모든 **pod**는 듀얼 스택이 활성화됩니다.



### 참고

이중 스택 네트워크는 베어 메탈, **IBM Power** 인프라 및 단일 노드 **OpenShift** 클러스터에서 프로비저닝된 클러스터에서 지원됩니다.

### 22.4.1. 듀얼 스택 클러스터 네트워크로 변환

클러스터 관리자는 단일 스택 클러스터 네트워크를 듀얼 스택 클러스터 네트워크로 변환할 수 있습니다.



### 참고

듀얼 스택 네트워킹으로 변환한 후에는 새로 생성된 **pod**만 **IPv6** 주소에 할당됩니다. **IPv6** 주소를 받으려면 변환하기 전에 생성된 모든 **Pod**를 다시 생성해야 합니다.

### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 클러스터는 **OVN-Kubernetes** 클러스터 네트워크 공급자를 사용합니다.
- 클러스터 노드에는 **IPv6** 주소가 있습니다.
- 인프라를 기반으로 **IPv6** 지원 라우터를 구성했습니다.

### 절차

- 1.



클러스터 및 서비스 네트워크에 대한 IPv6 주소 블록을 지정하려면 다음 YAML이 포함된 파일을 생성합니다.

```
- op: add
  path: /spec/clusterNetwork/-
  value: ①
    cidr: fd01::/48
    hostPrefix: 64
- op: add
  path: /spec/serviceNetwork/-
  value: fd02::/112 ②
```

①

`cidr` 및 `hostPrefix` 필드를 사용하여 오브젝트를 지정합니다. 호스트 접두사는 64 이상이어야 합니다. IPv6 CIDR 접두사는 지정된 호스트 접두사를 수용할 수 있을 만큼 커야 합니다.

②

접두사가 112인 IPv6 CIDR을 지정합니다. Kubernetes는 가장 낮은 16비트만 사용합니다. 접두사 112의 경우 IP 주소는 112비트에서 128비트로 할당됩니다.

2.

클러스터 네트워크 구성을 패치하려면 다음 명령을 입력합니다.

```
$ oc patch network.config.openshift.io cluster \
  --type='json' --patch-file <file>.yaml
```

다음과 같습니다.

**file**

이전 단계에서 만든 파일의 이름을 지정합니다.

출력 예

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

다음 단계를 완료하여 클러스터 네트워크가 이전 프로세스에서 지정한 IPv6 주소 블록을 인식하는지 확인합니다.

1. 네트워크 구성을 표시합니다.

```
$ oc describe network
```

출력 예

```
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
```

#### 22.4.2. 단일 스택 클러스터 네트워크로 변환

클러스터 관리자는 듀얼 스택 클러스터 네트워크를 단일 스택 클러스터 네트워크로 변환할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 클러스터는 **OVN-Kubernetes** 클러스터 네트워크 공급자를 사용합니다.

- 클러스터 노드에는 IPv6 주소가 있습니다.
- 듀얼 스택 네트워킹을 활성화했습니다.

#### 프로세스

1. 다음 명령을 실행하여 `networks.config.openshift.io` CR(사용자 정의 리소스)을 편집합니다.

```
$ oc edit networks.config.openshift.io
```

2. 이전 프로세스의 `cidr` 및 `hostPrefix` 필드에 추가한 IPv6 특정 구성을 제거합니다.

### 22.5. IPSEC 암호화 구성

IPsec을 사용하면 OVN-Kubernetes 클러스터 네트워크의 노드 간 모든 pod-to-pod 네트워크 트래픽은 IPsec 전송 모드로 암호화됩니다.

IPsec은 기본적으로 비활성화되어 있습니다. 클러스터를 설치하는 동안 또는 설치 후 활성화할 수 있습니다. 클러스터 설치에 대한 자세한 내용은 [OpenShift Container Platform 설치 개요](#)를 참조하십시오. 클러스터 설치 후 IPsec을 활성화해야 하는 경우 먼저 IPsec ESP IP 헤더의 오버헤드를 고려하여 클러스터 MTU의 크기를 조정해야 합니다.

다음 문서에서는 클러스터 설치 후 IPsec을 활성화 및 비활성화하는 방법을 설명합니다.

#### 22.5.1. 사전 요구 사항

- IPsec ESP 헤더의 추가 오버헤드를 허용하도록 클러스터 MTU 크기를 46 바이트로 줄였습니다. 클러스터가 사용하는 MTU 크기 조정에 대한 자세한 내용은 [클러스터 네트워크의 MTU 변경](#)을 참조하십시오.

#### 22.5.2. IPsec에서 암호화하는 네트워크 트래픽 흐름 유형

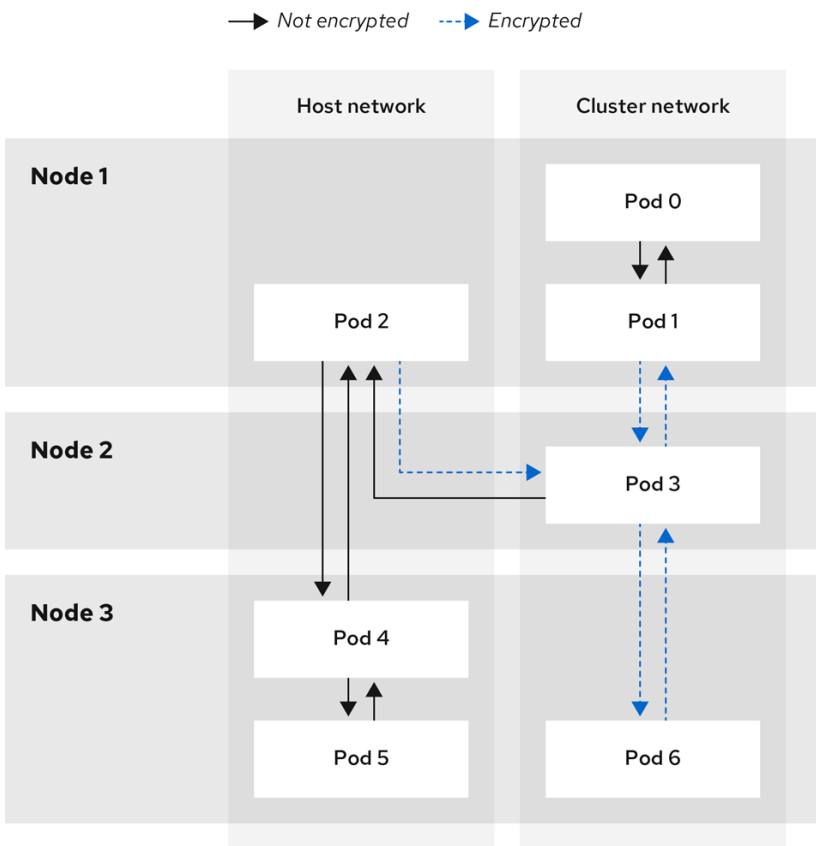
IPsec을 활성화하면 포드 간 다음 네트워크 트래픽 흐름만 암호화됩니다.

- 클러스터 네트워크의 서로 다른 노드에 있는 pod 간 트래픽
- 호스트 네트워크의 포드에서 클러스터 네트워크의 포드로의 트래픽

다음 트래픽 흐름은 암호화되지 않습니다.

- 클러스터 네트워크의 동일한 노드에 있는 pod 간 트래픽
- 호스트 네트워크의 포드 간 트래픽
- 클러스터 네트워크의 포드에서 호스트 네트워크 포드로의 트래픽

암호화되거나 암호화되지 않은 흐름은 다음 다이어그램에 설명되어 있습니다.



138\_OpenShift\_0421

22.5.2.1. IPsec이 활성화된 경우 네트워크 연결 요구 사항

**OpenShift Container Platform** 클러스터 구성 요소가 통신할 수 있도록 시스템 간 네트워크 연결을 구성해야 합니다. 각 시스템에서 클러스터에 있는 다른 모든 시스템의 호스트 이름을 확인할 수 있어야 합니다.

표 22.7. 모든 시스템 간 통신에 사용되는 포트

프로토콜	포트	설명
UDP	500	IPsec IKE 패킷
	4500	IPsec NAT-T 패킷
ESP	해당 없음	IPsec Encapsulating Security Payload (ESP)

### 22.5.3. 암호화 프로토콜 및 IPsec 모드

사용된 암호화 암호는 **AES-GCM-16-256**입니다. 무결성 검사 값(ICV)은 16바이트입니다. 키 길이는 256비트입니다.

사용된 IPsec 모드는 전송 모드입니다. **ESP(Encapsulated Security Payload)** 헤더를 원래 패킷의 IP 헤더에 추가하고 패킷 데이터를 암호화하여 엔드 투 엔드 통신을 암호화하는 모드입니다. **OpenShift Container Platform**은 현재 pod-to-pod 통신을 위해 IPsec CloudEvent 모드를 사용하거나 지원하지 않습니다.

### 22.5.4. 보안 인증서 생성 및 교체

**CNO(Cluster Network Operator)**는 암호화에 IPsec에서 사용하는 자체 서명된 X.509 인증 기관(CA)을 생성합니다. 각 노드의 CSR(인증서 서명 요청)은 CNO에서 자동으로 충족됩니다.

CA는 10년 동안 유효합니다. 개별 노드 인증서는 5년간 유효하며 4년 6개월 경과 후 자동으로 교체됩니다.

### 22.5.5. IPsec 암호화 활성화

클러스터 관리자는 클러스터 설치 후 IPsec 암호화를 활성화할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.

- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- **IPsec ESP** 헤더의 오버헤드를 허용하도록 클러스터 **MTU** 크기를 **46** 바이트로 줄였습니다.

프로세스

- **IPsec** 암호화를 활성화하려면 다음 명령을 입력합니다.

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"ipsecConfig":{}}}}'
```

22.5.6. IPsec이 활성화되었는지 확인

클러스터 관리자는 **IPsec**이 활성화되어 있는지 확인할 수 있습니다.

검증

1. **OVN-Kubernetes** 컨트롤 플레인 Pod의 이름을 찾으려면 다음 명령을 입력합니다.

```
$ oc get pods -n openshift-ovn-kubernetes | grep ovnkube-master
```

출력 예

```
ovnkube-master-4496s    1/1    Running 0    6h39m
ovnkube-master-d6cht   1/1    Running 0    6h42m
ovnkube-master-skblc   1/1    Running 0    6h51m
ovnkube-master-vf8rf   1/1    Running 0    6h51m
ovnkube-master-w7hjr   1/1    Running 0    6h51m
ovnkube-master-zsk7x   1/1    Running 0    6h42m
```

2. 클러스터에서 **IPsec**이 활성화되어 있는지 확인합니다.

```
$ oc -n openshift-ovn-kubernetes -c nbdb rsh ovnkube-master-<XXXXX> \
  ovn-nbctl --no-leader-only get nb_global . ipsec
```

다음과 같습니다.

<XXXXXX>

이전 단계의 **Pod**에 대한 임의의 문자 시퀀스를 지정합니다.

출력 예

**true**

### 22.5.7. IPsec 암호화 비활성화

클러스터 관리자는 클러스터 설치 후 **IPsec**을 활성화한 경우에만 **IPsec** 암호화를 비활성화할 수 있습니다.



참고

클러스터를 설치할 때 **IPsec**을 활성화한 경우 이 절차를 사용하여 **IPsec**을 비활성화할 수 없습니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

프로세스

1. **IPsec** 암호화를 비활성화하려면 다음 명령을 입력합니다.

```
$ oc patch networks.operator.openshift.io/cluster --type=json \
-p='[{"op": "remove",
"path": "/spec/defaultNetwork/ovnKubernetesConfig/ipsecConfig"}]'
```

2.

선택 사항: 클러스터 MTU 크기를 46 바이트로 늘릴 수 있습니다. IP 패킷의 IPsec ESP 헤더에 대한 오버헤드가 더 이상 없기 때문입니다.

### 22.5.8. 추가 리소스

- [OVN-Kubernetes CNI\(Container Network Interface\) 네트워크 공급자 정보](#)
- [클러스터 네트워크의 MTU 변경](#)
- [Network \[operator.openshift.io/v1\] API](#)

## 22.6. 프로젝트에 대한 송신 방화벽 구성

클러스터 관리자는 **OpenShift Container Platform** 클러스터에서 나가는 송신 트래픽을 제한하는 프로젝트에 대한 송신 방화벽을 생성할 수 있습니다.

### 22.6.1. 프로젝트에서 송신 방화벽이 작동하는 방식

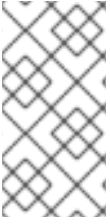
클러스터 관리자는 송신 방화벽을 사용하여 일부 또는 모든 Pod가 클러스터 내에서 액세스할 수 있는 외부 호스트를 제한할 수 있습니다. 송신 방화벽은 다음 시나리오를 지원합니다.

- Pod는 내부 호스트에만 연결할 수 있으며 공용 인터넷 연결을 시작할 수 없습니다.
- Pod는 공용 인터넷에만 연결할 수 있으며 **OpenShift Container Platform** 클러스터 외부에 있는 내부 호스트에 대한 연결을 시작할 수 없습니다.
- Pod는 지정된 내부 서브넷이나 **OpenShift Container Platform** 클러스터 외부의 호스트에 연결할 수 없습니다.
- Pod는 특정 외부 호스트에만 연결할 수 있습니다.

예를 들어, 한 프로젝트가 지정된 IP 범위에 액세스하도록 허용하지만 다른 프로젝트에 대한 동일한 액세스는 거부할 수 있습니다. 또는 애플리케이션 개발자가 **Python pip** 미러에서 업데이트하지 못하도록



하고 승인된 소스에서만 업데이트를 수행하도록 할 수 있습니다.



#### 참고

송신 방화벽은 호스트 네트워크 네임스페이스에 적용되지 않습니다. 호스트 네트워킹이 활성화된 Pod는 송신 방화벽 규칙의 영향을 받지 않습니다.

**EgressFirewall CR**(사용자 정의 리소스) 오브젝트를 만들어 송신 방화벽 정책을 구성합니다. 송신 방화벽은 다음 기준 중 하나를 충족하는 네트워크 트래픽과 일치합니다.

- **CIDR 형식의 IP 주소 범위**
- **IP 주소로 확인되는 DNS 이름**
- **포트 번호**
- **다음 프로토콜 중 하나인 프로토콜 : TCP, UDP 및 SCTP**

## 중요

송신 방화벽에 0.0.0.0/0에 대한 거부 규칙이 포함된 경우 OpenShift Container Platform API 서버에 대한 액세스 권한이 차단됩니다. Pod가 OpenShift Container Platform API 서버에 액세스할 수 있도록 하려면 EgressFirewall과 함께 노드 포트를 사용할 때 액세스할 수 있도록 OVN(Open Virtual Network)의 기본 연결 네트워크 100.64.0.0/16 을 포함해야 합니다. 다음 예와 같이 송신 방화벽 규칙에서 API 서버가 청취하는 IP 주소 범위도 포함해야 합니다.

```
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

송신 방화벽의 네임스페이스입니다.

②

OpenShift Container Platform API 서버를 포함하는 IP 주소 범위입니다.

③

글로벌 거부 규칙은 OpenShift Container Platform API 서버에 액세스할 수 없습니다.

API 서버의 IP 주소를 찾으려면 `oc get ep kubernetes -n default` 를 실행합니다.

자세한 내용은 [BZ#1988324](#)에서 참조하십시오.



### 주의

송신 방화벽 규칙은 라우터를 통과하는 트래픽에는 적용되지 않습니다. **Route CR** 오브젝트를 생성할 권한이 있는 모든 사용자는 허용되지 않은 대상을 가리키는 경로를 생성하여 송신 방화벽 정책 규칙을 바이패스할 수 있습니다.

#### 22.6.1.1. 송신 방화벽의 제한

송신 방화벽에는 다음과 같은 제한이 있습니다.

- **EgressFirewall** 오브젝트를 두 개 이상 보유할 수 있는 프로젝트는 없습니다.
- 프로젝트당 최대 50개의 규칙이 있는 최대 하나의 **EgressFirewall** 오브젝트를 정의할 수 있습니다.
- **Red Hat OpenShift Networking**에서 공유 게이트웨이 모드와 함께 **OVN-Kubernetes** 네트워크 플러그인을 사용하는 경우 수신 응답을 송신 방화벽 규칙의 영향을 받습니다. 송신 방화벽 규칙이 수신 응답 대상 IP를 삭제하면 트래픽이 삭제됩니다.

이러한 제한 사항을 위반하면 프로젝트의 송신 방화벽이 손상되고 모든 외부 네트워크 트래픽이 삭제될 수 있습니다.

**Egress** 방화벽 리소스는 **kube-node-lease,kube-public,kube-system,openshift** 및 **openshift-** 프로젝트에서 생성할 수 있습니다.

#### 22.6.1.2. 송신 방화벽 정책 규칙에 대한 일치 순서

송신 방화벽 정책 규칙은 정의된 순서대로 처음부터 마지막까지 평가됩니다. **Pod**의 송신 연결과 일치하는 첫 번째 규칙이 적용됩니다. 해당 연결에 대한 모든 후속 규칙은 무시됩니다.

#### 22.6.1.3. DNS(Domain Name Server) 확인 작동 방식

송신 방화벽 정책 규칙에서 **DNS** 이름을 사용하는 경우 도메인 이름의 적절한 확인에는 다음 제한 사항이 적용됩니다.

- 도메인 이름 업데이트는 **TTL(Time To- Live)** 기간에 따라 폴링됩니다. 기본적으로 기간은 **30분**입니다. 송신 방화벽 컨트롤러가 로컬 이름 서버에 도메인 이름을 쿼리할 때 응답에 **TTL**이 포함되고 **TTL**이 **30분 미만**이면 컨트롤러는 해당 **DNS** 이름의 기간을 반환된 값으로 설정합니다. 각 **DNS** 이름은 **DNS** 레코드의 **TTL**이 만료된 후에 쿼리됩니다.
- Pod**는 필요한 경우 동일한 로컬 이름 서버에서 도메인을 확인해야 합니다. 확인하지 않으면 송신 방화벽 컨트롤러와 **Pod**에 의해 알려진 도메인의 **IP** 주소가 다를 수 있습니다. 호스트 이름의 **IP** 주소가 다르면 송신 방화벽이 일관되게 적용되지 않을 수 있습니다.
- 송신 방화벽 컨트롤러와 **Pod**는 동일한 로컬 이름 서버를 비동기적으로 폴링하기 때문에 **Pod**가 송신 컨트롤러보다 먼저 업데이트된 **IP** 주소를 얻을 수 있으며 이로 인해 경쟁 조건이 발생합니다. 현재 이런 제한으로 인해 **EgressFirewall** 오브젝트의 도메인 이름 사용은 **IP** 주소가 자주 변경되지 않는 도메인에만 권장됩니다.



참고

송신 방화벽은 **Pod**가 **DNS** 확인을 위해 **Pod**가 있는 노드의 외부 인터페이스에 항상 액세스할 수 있도록 합니다.

송신 방화벽 정책에서 도메인 이름을 사용하고 로컬 노드의 **DNS** 서버에서 **DNS** 확인을 처리하지 않으면 **Pod**에서 도메인 이름을 사용하는 경우, **DNS** 서버의 **IP** 주소에 대한 액세스를 허용하는 송신 방화벽 규칙을 추가해야 합니다.

**22.6.2. EgressFirewall CR(사용자 정의 리소스) 오브젝트**

송신 방화벽에 대해 하나 이상의 규칙을 정의할 수 있습니다. 규칙이 적용되는 트래픽에 대한 사양을 담은 **Allow** 규칙 또는 **Deny** 규칙입니다.

다음 **YAML**은 **EgressFirewall CR** 오브젝트를 설명합니다.

**EgressFirewall** 오브젝트

```

apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: <name> 1
    
```

```
spec:
  egress: 2
  ...
```

1

오브젝트의 이름은 **default**이어야 합니다.

2

다음 섹션에서 설명하는 하나 이상의 송신 네트워크 정책 규칙 컬렉션입니다.

### 22.6.2.1. EgressFirewall 규칙

다음 **YAML**은 송신 방화벽 규칙 오브젝트를 설명합니다. 송신 스텐자는 하나 이상의 오브젝트 배열을 예상합니다.

송신 정책 규칙 스텐자

```
egress:
- type: <type> 1
  to: 2
  cidrSelector: <cidr> 3
  dnsName: <dns_name> 4
  ports: 5
  ...
```

1

규칙 유형입니다. 값은 **Allow** 또는 **Deny**여야 합니다.

2

**cidrSelector** 필드 또는 **dnsName** 필드를 지정하는 송신 트래픽 일치 규칙을 설명하는 스텐자입니다. 동일한 규칙에서 두 필드를 모두 사용할 수 없습니다.

3

4

DNS 도메인 이름입니다.

5

선택 사항: 규칙에 대한 네트워크 포트 및 프로토콜 컬렉션을 설명하는 스탠자입니다.

포트 스탠자

```
ports:
- port: <port> 1
  protocol: <protocol> 2
```

1

80 또는 443과 같은 네트워크 포트. 이 필드의 값을 지정하는 경우 protocol의 값도 지정해야 합니다.

2

네트워크 프로토콜. 값은 TCP, UDP 또는 SCTP여야 합니다.

### 22.6.2.2. EgressFirewall CR 오브젝트의 예

다음 예는 여러 가지 송신 방화벽 정책 규칙을 정의합니다.

```
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

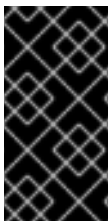
송신 방화벽 정책 규칙 오브젝트의 컬렉션입니다.

다음 예에서는 트래픽이 TCP 프로토콜 및 대상 포트 80 또는 임의의 프로토콜 및 대상 포트 443을 사용하는 경우 172.16.1.1 IP 주소에서 호스트에 대한 트래픽을 거부하는 정책 규칙을 정의합니다.

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress:
    - type: Deny
      to:
        cidrSelector: 172.16.1.1
      ports:
        - port: 80
          protocol: TCP
        - port: 443
```

### 22.6.3. 송신 방화벽 정책 오브젝트 생성

클러스터 관리자는 프로젝트에 대한 송신 방화벽 정책 오브젝트를 만들 수 있습니다.



#### 중요

프로젝트에 이미 **EgressFirewall** 오브젝트가 정의되어 있는 경우 기존 정책을 편집하여 송신 방화벽 규칙을 변경해야 합니다.

#### 사전 요구 사항

- **OVN-Kubernetes** 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용하는 클러스터입니다.
- **OpenShift CLI(oc)**를 설치합니다.

- 클러스터 관리자로 클러스터에 로그인해야 합니다.

## 프로세스

1. 다음과 같이 정책 규칙을 생성합니다.
  - a. `<policy_name>`이 송신 정책 규칙을 설명하는 `<policy_name>.yaml` 파일을 만듭니다.
  - b. 생성한 파일에서 송신 정책 오브젝트를 정의합니다.
2. 다음 명령을 입력하여 정책 오브젝트를 생성합니다. `<policy_name>`을 정책 이름으로 바꾸고 `<project>`를 규칙이 적용되는 프로젝트로 바꿉니다.

```
$ oc create -f <policy_name>.yaml -n <project>
```

다음 예제에서는 `project1`이라는 프로젝트에 새 `EgressFirewall` 오브젝트가 생성됩니다.

```
$ oc create -f default.yaml -n project1
```

출력 예

```
egressfirewall.k8s.ovn.org/v1 created
```

3. 선택사항: 나중에 변경할 수 있도록 `<policy_name>.yaml` 파일을 저장합니다.

## 22.7. 프로젝트의 송신 방화벽 보기

클러스터 관리자는 기존 송신 방화벽의 이름을 나열하고 특정 송신 방화벽에 대한 트래픽 규칙을 볼 수 있습니다.

### 22.7.1. EgressFirewall 오브젝트 보기



클러스터의 **EgressFirewall** 오브젝트를 볼 수 있습니다.

#### 사전 요구 사항

- **OVN-Kubernetes 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용하는 클러스터입니다.**
- **oc로 알려진 OpenShift 명령 인터페이스 (CLI)를 설치합니다.**
- **클러스터에 로그인해야 합니다.**

#### 프로세스

1. **선택사항: 클러스터에 정의된 EgressFirewall 오브젝트의 이름을 보려면 다음 명령을 입력합니다.**

```
$ oc get egressfirewall --all-namespaces
```

2. **정책을 검사하려면 다음 명령을 입력하십시오. <policy\_name>을 검사할 정책 이름으로 교체합니다.**

```
$ oc describe egressfirewall <policy_name>
```

#### 출력 예

```
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
```

## 22.8. 프로젝트의 송신 방화벽 편집

클러스터 관리자는 기존 송신 방화벽에 대한 네트워크 트래픽 규칙을 수정할 수 있습니다.

### 22.8.1. EgressFirewall 오브젝트 편집

클러스터 관리자는 프로젝트의 송신 방화벽을 업데이트할 수 있습니다.

#### 사전 요구 사항

- **OVN-Kubernetes** 기본 **CNI(Container Network Interface)** 네트워크 공급자 플러그인을 사용하는 클러스터입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- 클러스터 관리자로 클러스터에 로그인해야 합니다.

#### 프로세스

1. 프로젝트의 **EgressFirewall** 오브젝트 이름을 찾습니다. **<project>**를 프로젝트 이름으로 바꿉니다.

```
$ oc get -n <project> egressfirewall
```

2. 선택 사항: 송신 네트워크 방화벽을 만들 때 **EgressFirewall** 오브젝트의 사본을 저장하지 않은 경우 다음 명령을 입력하여 사본을 생성합니다.

```
$ oc get -n <project> egressfirewall <name> -o yaml > <filename>.yaml
```

**<project>**를 프로젝트 이름으로 바꿉니다. **<name>**을 오브젝트 이름으로 변경합니다. **YAML**을 저장할 파일의 이름으로 **<filename>**을 바꿉니다.

3. 정책 규칙을 변경한 후 다음 명령을 입력하여 **EgressFirewall** 오브젝트를 바꿉니다. 업데이트된 **EgressFirewall** 오브젝트가 포함된 파일 이름으로 **<filename>**을 바꿉니다.

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

## 22.9. 프로젝트에서 송신 방화벽 제거

클러스터 관리자는 프로젝트에서 송신 방화벽을 제거하여 **OpenShift Container Platform** 클러스터를 나가는 프로젝트에서 네트워크 트래픽에 대한 모든 제한을 제거할 수 있습니다.

### 22.9.1. EgressFirewall 오브젝트 제거

클러스터 관리자는 프로젝트에서 송신 방화벽을 제거할 수 있습니다.

#### 사전 요구 사항

- **OVN-Kubernetes** 기본 **CNI(Container Network Interface)** 네트워크 공급자 플러그인을 사용하는 클러스터입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- 클러스터 관리자로 클러스터에 로그인해야 합니다.

#### 프로세스

1. 프로젝트의 **EgressFirewall** 오브젝트 이름을 찾습니다. **<project>**를 프로젝트 이름으로 바꿉니다.

```
$ oc get -n <project> egressfirewall
```

2. 다음 명령을 입력하여 **EgressFirewall** 오브젝트를 삭제합니다. **<project>**를 프로젝트 이름으로 바꾸고 **<name>**을 오브젝트 이름으로 바꿉니다.

```
$ oc delete -n <project> egressfirewall <name>
```

## 22.10. 송신 IP 주소 구성

클러스터 관리자는 **OVN-Kubernetes CNI(Container Network Interface)** 클러스터 네트워크 공급자를 구성하여 하나 이상의 송신 IP 주소를 네임스페이스 또는 네임스페이스의 특정 **Pod**에 할당할 수 있습니다.

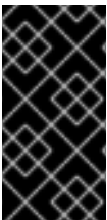
### 22.10.1. 송신 IP 주소 아키텍처 설계 및 구현

**OpenShift Container Platform** 송신 IP 주소 기능을 사용하면 하나 이상의 네임스페이스에 있는 하나 이상의 Pod에서 발생하는 트래픽의 소스 IP 주소가 클러스터 네트워크 외부 서비스에 일관되게 표시되도록 할 수 있습니다.

예를 들어 클러스터 외부 서버에서 호스팅되는 데이터베이스를 주기적으로 쿼리하는 Pod가 있을 수 있습니다. 서버에 대한 액세스 요구 사항을 적용하기 위해 패킷 필터링 장치는 특정 IP 주소의 트래픽만 허용하도록 구성됩니다. 특정 Pod에서만 서버에 안정적으로 액세스할 수 있도록 허용하려면 서버에 요청하는 Pod에 대해 특정 송신 IP 주소를 구성하면 됩니다.

네임스페이스에 할당된 송신 IP 주소는 특정 대상으로 트래픽을 보내는 데 사용되는 송신 라우터와 다릅니다.

일부 클러스터 구성에서 애플리케이션 Pod 및 인그레스 라우터 Pod가 동일한 노드에서 실행됩니다. 이 시나리오에서 애플리케이션 프로젝트에 대한 송신 IP 주소를 구성하는 경우 애플리케이션 프로젝트에서 경로로 요청을 보낼 때 IP 주소가 사용되지 않습니다.



**중요**

송신 IP 주소는 `ifcfg-eth0`과 같은 Linux 네트워크 구성 파일에서 구성하지 않아야 합니다.

**22.10.1.1. 플랫폼 지원**

다음 표에는 다양한 플랫폼의 송신 IP 주소 기능에 대한 지원이 요약되어 있습니다.

플랫폼	지원됨
베어 메탈	있음
VMware vSphere	있음
Red Hat OpenStack Platform (RHOSP)	없음
AWS(Amazon Web Services)	있음
GCP(Google Cloud Platform)	있음
Microsoft Azure	있음



## 중요

**EgressIP 기능을 사용하여 컨트롤 플레인 노드에 송신 IP 주소를 할당하는 것은 AWS(Amazon Web Services)에서 프로비저닝된 클러스터에서는 지원되지 않습니다. (BZ#2039656)**

### 22.10.1.2. 퍼블릭 클라우드 플랫폼 고려 사항

퍼블릭 클라우드 인프라에 프로비저닝된 클러스터의 경우 노드당 절대 할당 가능한 IP 주소 수에 대한 제약 조건이 있습니다. 노드당 최대 할당 가능한 IP 주소 수 또는 IP 용량은 다음 공식에서 설명할 수 있습니다.

$$IP\ capacity = public\ cloud\ default\ capacity - sum(current\ IP\ assignments)$$

**Egress IP** 기능은 노드당 IP 주소 용량을 관리하는 반면, 배포에서 이 제약 조건을 계획하는 것이 중요합니다. 예를 들어 8개의 노드가 있는 베어 메탈 인프라에 설치된 클러스터의 경우 150개의 송신 IP 주소를 구성할 수 있습니다. 그러나 공용 클라우드 공급자가 IP 주소 용량을 노드당 10개의 IP 주소로 제한하는 경우 총 할당 가능한 IP 주소 수는 80입니다. 이 예제에서 동일한 IP 주소 용량을 얻으려면 7개의 추가 노드를 할당해야 합니다.

퍼블릭 클라우드 환경에서 노드의 IP 용량 및 서브넷을 확인하려면 `oc get node <node_name> -o yaml` 명령을 입력합니다. `cloud.network.openshift.io/egress-ipconfig` 주석에는 노드에 대한 용량 및 서브넷 정보가 포함됩니다.

주석 값은 기본 네트워크 인터페이스에 다음 정보를 제공하는 필드가 포함된 단일 오브젝트가 포함된 배열입니다.

- **Interface:** AWS 및 Azure의 인터페이스 ID와 GCP의 인터페이스 이름을 지정합니다.
- **ifaddr:** 하나 또는 두 IP 주소 제품군에 대한 서브넷 마스크를 지정합니다.
- **capacity:** 노드의 IP 주소 용량을 지정합니다. AWS에서 IP 주소 용량은 IP 주소 제품군별로 제공됩니다. Azure 및 GCP에서 IP 주소 용량에는 IPv4 및 IPv6 주소가 모두 포함됩니다.

다음 예제에서는 여러 공용 클라우드 공급자의 노드의 주석을 보여줍니다. 주석은 가독성을 위해 들여 쓰기되어 있습니다.

AWS의 `cloud.network.openshift.io/egress-ipconfig` 주석 예

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

GCP의 cloud.network.openshift.io/egress-ipconfig 주석의 예

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]
```

다음 섹션에서는 용량 계산에 사용할 지원되는 퍼블릭 클라우드 환경에 대한 IP 주소 용량에 대해 설명합니다.

#### 22.10.1.2.1. AWS(Amazon Web Services) IP 주소 용량 제한

AWS에서 IP 주소 할당에 대한 제한은 구성된 인스턴스 유형에 따라 다릅니다. 자세한 내용은 [인스턴스 유형당 네트워크 인터페이스당 IP 주소](#)를 참조하십시오.

#### 22.10.1.2.2. GCP(Google Cloud Platform) IP 주소 용량 제한

GCP에서 네트워킹 모델은 IP 주소 할당이 아닌 IP 주소 별칭을 통해 추가 노드 IP 주소를 구현합니다. 그러나 IP 주소 용량은 IP 별칭 용량에 직접 매핑됩니다.

IP 별칭 할당에는 다음 용량 제한이 있습니다.

- 노드당 최대 IP 별칭(IPv4 및 IPv6 모두)은 10입니다.
- VPC당 최대 IP 별칭 수는 지정되지 않지만 OpenShift Container Platform 확장성 테스트에서는 최대 15,000개가 됩니다.

자세한 내용은 [인스턴스 할당량별 및 IP 범위 개요](#) 를 참조하십시오.

### 22.10.1.2.3. Microsoft Azure IP 주소 용량 제한

Azure에서 IP 주소 할당에 대해 다음과 같은 용량 제한이 있습니다. On Azure, the following capacity limits exist for IP address assignment:

- NIC당 IPv4 및 IPv6 모두에 대해 할당 가능한 최대 IP 주소 수는 256입니다.
- 가상 네트워크당 할당된 IP 주소의 최대 수는 65,536을 초과할 수 없습니다.

자세한 내용은 [네트워킹 제한](#) 을 참조하십시오.

### 22.10.1.3. Pod에 송신 IP 할당

하나 이상의 송신 IP를 네임스페이스 또는 네임스페이스의 특정 Pod에 할당하려면 다음 조건을 충족해야 합니다.

- 클러스터에서 하나 이상의 노드에 `k8s.ovn.org/egress-assignable: ""` 레이블이 있어야 합니다.
- 네임스페이스의 Pod에서 클러스터를 떠나는 트래픽의 소스 IP 주소로 사용할 하나 이상의 송신 IP 주소를 정의하는 `EgressIP` 오브젝트가 있습니다.



## 중요

송신 IP 할당을 위해 클러스터의 노드에 레이블을 지정하기 전에 **EgressIP** 오브젝트를 생성하면 **OpenShift Container Platform**에서 모든 송신 IP 주소를 `k8s.ovn.org/egress-assignable: ""` 레이블이 있는 첫 번째 노드에 할당할 수 있습니다.

송신 IP 주소가 클러스터의 여러 노드에 널리 분산되도록 하려면 **EgressIP** 오브젝트를 만들기 전에 송신 IP 주소를 호스팅할 노드에 항상 레이블을 적용하십시오.

### 22.10.1.4. 노드에 송신 IP 할당

**EgressIP** 오브젝트를 생성할 때 `k8s.ovn.org/egress-assignable: ""` 레이블이 지정된 노드에 다음 조건이 적용됩니다.

- 송신 IP 주소는 한 번에 두 개 이상의 노드에 할당되지 않습니다.
- 송신 IP 주소는 송신 IP 주소를 호스팅할 수 있는 사용 가능한 노드 간에 균형을 이룹니다.
- **EgressIP** 오브젝트의 `spec.EgressIPs` 배열에서 둘 이상의 IP 주소를 지정하는 경우 다음 조건이 적용됩니다.
  - 지정된 IP 주소 중 두 개 이상을 호스팅할 노드는 없습니다.
  - 지정된 네임스페이스에 대해 지정된 IP 주소 간에 트래픽이 거의 동일하게 분산됩니다.
- 노드를 사용할 수 없게 되면 할당된 모든 송신 IP 주소가 이전에 설명한 조건에 따라 자동으로 재할당됩니다.

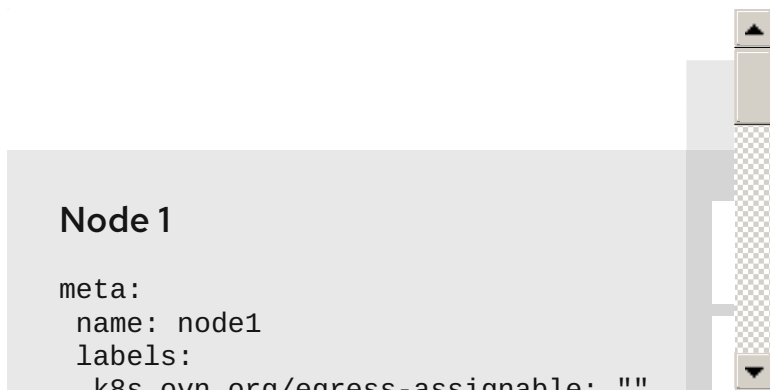
**Pod**가 여러 **EgressIP** 오브젝트의 선택기와 일치하는 경우 **EgressIP** 오브젝트에 지정된 송신 IP 주소 중 어느 것이 **Pod**의 송신 IP 주소로 할당되는지 보장할 수 없습니다.

또한 **EgressIP** 오브젝트에서 여러 송신 IP 주소를 지정하는 경우 송신 IP 주소를 사용할 수 있다는 보장이 없습니다. 예를 들어 **Pod**가 두 개의 송신 IP 주소인 `10.10.20.1` 및 `10.10.20.2` 를 사용하여 **EgressIP** 오브젝트의 선택기와 일치하는 경우 각 **TCP** 연결 또는 **UDP** 대화에 사용할 수 있습니다.



### 22.10.1.5. 송신 IP 주소 구성에 대한 아키텍처 다이어그램

다음 다이어그램에서는 송신 IP 주소 구성을 보여줍니다. 다이어그램은 클러스터의 세 개 노드에서 실행 중인 두 개의 다른 네임스페이스에 있는 포트 4개를 설명합니다. 노드는 호스트 네트워크의 192.168.126.0/18 CIDR 블록에서 할당된 IP 주소입니다.



노드 1과 노드 3은 모두 `k8s.ovn.org/egress-assignable: ""`로 레이블이 지정되어 있으므로 송신 IP 주소 할당에 사용할 수 있습니다.

다이어그램에 있는 점선은 노드 1 및 노드 3에서 클러스터를 나가기 위해 포트 네트워크를 통해 이동하는 pod1, pod2, pod3의 트래픽 흐름을 나타냅니다. 외부 서비스에서 예제의 EgressIP 오브젝트에서 선택한 Pod 중 하나에서 트래픽을 수신하는 경우 소스 IP 주소는 192.168.126.10 또는 192.168.126.102입니다. 트래픽은 이 두 노드 간에 대략적으로 균등하게 분산됩니다.

다이어그램의 다음 리소스는 자세히 설명되어 있습니다.

#### Namespace 오브젝트

네임스페이스는 다음 매니페스트에 정의됩니다.

#### 네임스페이스 오브젝트

```

apiVersion: v1
kind: Namespace
metadata:
  name: namespace1
  labels:
    env: prod
---
apiVersion: v1
kind: Namespace
metadata:
  
```

```

name: namespace2
labels:
  env: prod

```

### EgressIP 오브젝트

다음 **EgressIP** 오브젝트는 **env** 라벨이 **prod**로 설정된 모든 포드를 선택하는 구성을 설명합니다. 선택한 포드의 송신 IP 주소는 **192.168.126.10** 및 **192.168.126.102**입니다.

### EgressIP 오브젝트

```

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

```

이전 예제의 구성에 대해 **OpenShift Container Platform**은 두 송신 IP 주소를 사용 가능한 노드에 할당합니다. **status** 필드는 송신 IP 주소가 할당되었는지 여부와 위치를 반영합니다.

### 22.10.2. EgressIP 오브젝트

다음 **YAML**에서는 **EgressIP** 오브젝트의 **API**를 설명합니다. 오브젝트의 범위는 클러스터 전체이며 네임스페이스에 생성되지 않습니다.

```

apiVersion: k8s.ovn.org/v1

```

```

kind: EgressIP
metadata:
  name: <name> 1
spec:
  egressIPs: 2
  - <ip_address>
  namespaceSelector: 3
  ...
  podSelector: 4
  ...

```

1

**EgressIPs** 오브젝트의 이름입니다.

2

하나 이상의 **IP** 주소로 이루어진 배열입니다.

3

송신 **IP** 주소를 연결할 네임스페이스에 대해 하나 이상의 선택기입니다.

4

**선택사항:** 지정된 네임스페이스에서 송신 **IP** 주소를 연결할 **Pod**에 대해 하나 이상의 선택기입니다. 이러한 선택기를 적용하면 네임스페이스 내에서 **Pod** 하위 집합을 선택할 수 있습니다.

다음 **YAML**은 네임스페이스 선택기에 대한 스탠자를 설명합니다.

네임스페이스 선택기 스탠자

```

namespaceSelector: 1
matchLabels:
  <label_name>: <label_value>

```

1

네임스페이스에 대해 일치하는 하나 이상의 규칙입니다. 둘 이상의 일치 규칙이 제공되면 일치하는 모든 네임스페이스가 선택됩니다.

다음 YAML은 **Pod** 선택기에 대한 선택적 스태자를 설명합니다.

### Pod 선택기 스태자

```
podSelector: 1
  matchLabels:
    <label_name>: <label_value>
```

1

**선택사항:** 지정된 **namespaceSelector** 규칙과 일치하는 네임스페이스의 **Pod**에 대해 일치하는 하나 이상의 규칙입니다. 지정된 경우 일치하는 **Pod**만 선택됩니다. 네임스페이스의 다른 **Pod**는 선택되지 않습니다.

다음 예에서 **EgressIP** 오브젝트는 **192.168.126.11** 및 **192.168.126.102** 송신 IP 주소를 **app** 라벨을 **web**으로 설정하고 **env** 라벨을 **prod**로 설정한 네임스페이스에 있는 포드와 연결합니다.

### EgressIP 오브젝트의 예

```
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
```

다음 예에서 **EgressIP** 오브젝트는 **192.168.127.30** 및 **192.168.127.40** 송신 IP 주소를 **environment**

레이블이 **development**로 설정되지 않은 모든 **Pod**와 연결합니다.

### EgressIP 오브젝트의 예

```

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
  
```

### 22.10.3. 송신 IP 주소 호스팅을 위해 노드에 레이블 지정

OpenShift Container Platform에서 노드에 하나 이상의 송신 IP 주소를 할당할 수 있도록 `k8s.ovn.org/egress-assignable=""` 레이블을 클러스터의 노드에 적용할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- 클러스터 관리자로 클러스터에 로그인합니다.

#### 프로세스

- 하나 이상의 송신 IP 주소를 호스팅할 수 있도록 노드에 레이블을 지정하려면 다음 명령을 입력합니다.

```
$ oc label nodes <node_name> k8s.ovn.org/egress-assignable="" 1
```

1

레이블을 지정할 노드 이름입니다.

작은 정보

다음 **YAML**을 적용하여 노드에 레이블을 추가할 수 있습니다.

```
apiVersion: v1
kind: Node
metadata:
  labels:
    k8s.ovn.org/egress-assignable: ""
    name: <node_name>
```

#### 22.10.4. 다음 단계

- [송신 IP 할당](#)

#### 22.10.5. 추가 리소스

- [LabelSelector meta/v1](#)
- [LabelSelectorRequirement meta/v1](#)

### 22.11. 송신 IP 주소 할당

클러스터 관리자는 네임스페이스 또는 네임스페이스의 특정 **Pod**에서 클러스터를 떠나는 트래픽에 송신 **IP** 주소를 할당할 수 있습니다.

#### 22.11.1. 네임스페이스에 송신 IP 주소 할당

하나 이상의 송신 **IP** 주소를 네임스페이스 또는 네임스페이스의 특정 **Pod**에 할당할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.

- 클러스터 관리자로 클러스터에 로그인합니다.
- 송신 IP 주소를 호스팅할 하나 이상의 노드를 구성합니다.

### 프로세스

1.

**EgressIP** 오브젝트를 만듭니다.

a.

`<egressips_name>.yaml` 파일을 만듭니다. 여기서 `<egressips_name>`은 오브젝트 이름입니다.

b.

생성한 파일에서 다음 예와 같이 **EgressIP** 오브젝트를 정의합니다.

```
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.

오브젝트를 생성하려면 다음 명령을 입력합니다.

```
$ oc apply -f <egressips_name>.yaml 1
```

**1**

`<egressips_name>`을 오브젝트 이름으로 변경합니다.

### 출력 예

```
egressips.k8s.ovn.org/<egressips_name> created
```

3.

선택사항: 나중에 변경할 수 있도록 `<egressips_name>.yaml` 파일을 저장합니다.

4.

송신 IP 주소가 필요한 네임스페이스에 라벨을 추가합니다. 1단계에서 정의된 **EgressIP** 오브젝트의 네임스페이스에 라벨을 추가하려면 다음 명령을 실행합니다.

```
$ oc label ns <namespace> env=qa 1
```

1

`<namespace>`를 송신 IP 주소가 필요한 네임스페이스로 바꿉니다.

### 22.11.2. 추가 리소스

- 

송신 IP 주소 구성

## 22.12. 송신 라우터 POD 사용에 대한 고려 사항

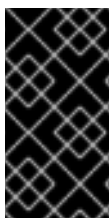
### 22.12.1. 송신 라우터 Pod 정보

**OpenShift Container Platform** 송신 라우터 포드는 다른 용도로 사용되지 않는 프라이빗 소스 IP 주소에서 지정된 원격 서버로 트래픽을 리디렉션합니다. 송신 라우터 포드를 통해 특정 IP 주소에서만 액세스할 수 있도록 설정된 서버로 네트워크 트래픽을 보낼 수 있습니다.



#### 참고

송신 라우터 Pod는 모든 발신 연결을 위한 것은 아닙니다. 다수의 송신 라우터 Pod를 생성하는 경우 네트워크 하드웨어 제한을 초과할 수 있습니다. 예를 들어 모든 프로젝트 또는 애플리케이션에 대해 송신 라우터 Pod를 생성하면 소프트웨어에서 MAC 주소 필터링으로 돌아가기 전에 네트워크 인터페이스에서 처리할 수 있는 로컬 MAC 주소 수를 초과할 수 있습니다.



#### 중요

송신 라우터 이미지는 Amazon AWS, Azure Cloud 또는 macvlan 트래픽과의 비호환성으로 인해 계층 2 조작을 지원하지 않는 기타 클라우드 플랫폼과 호환되지 않습니다.



### 22.12.1.1. 송신 라우터 모드

리디렉션 모드에서는 송신 라우터 포트가 자체 IP 주소에서 하나 이상의 대상 IP 주소로 트래픽을 리디렉션하도록 iptables 규칙을 구성합니다. 예약된 소스 IP 주소를 사용해야 하는 클라이언트 Pod는 대상 IP에 직접 연결하는 대신 송신 라우터에 연결하도록 수정해야 합니다.



#### 참고

송신 라우터 CNI 플러그인은 리디렉션 모드만 지원합니다. 이는 OpenShift SDN과 함께 배포할 수 있는 송신 라우터 구현의 차이점입니다. OpenShift SDN의 송신 라우터와 달리 송신 라우터 CNI 플러그인은 HTTP 프록시 모드 또는 DNS 프록시 모드를 지원하지 않습니다.

### 22.12.1.2. 송신 라우터 Pod 구현

송신 라우터 구현에서는 송신 라우터 CNI(Container Network Interface) 플러그인을 사용합니다. 플러그인은 보조 네트워크 인터페이스를 포트에 추가합니다.

송신 라우터는 두 개의 네트워크 인터페이스가 있는 포트입니다. 예를 들어 포트에는 eth0 및 net1 네트워크 인터페이스가 있을 수 있습니다. eth0 인터페이스는 클러스터 네트워크에 있으며 포트는 일반 클러스터 관련 네트워크 트래픽에 대한 인터페이스를 계속 사용합니다. net1 인터페이스는 보조 네트워크에 있으며 해당 네트워크에 대한 IP 주소 및 게이트웨이가 있습니다. OpenShift Container Platform 클러스터의 다른 포트는 송신 라우터 서비스에 액세스할 수 있으며, 서비스를 통해 포트가 외부 서비스에 액세스할 수 있습니다. 송신 라우터는 포트와 외부 시스템 간의 브리지 역할을 합니다.

송신 라우터를 종료하는 트래픽은 노드를 통해 종료되지만 패킷에는 송신 라우터 포트에서 net1 인터페이스의 MAC 주소가 있습니다.

송신 라우터 사용자 정의 리소스를 추가하면 Cluster Network Operator에서 다음 오브젝트를 생성합니다.

- pod의 net1 보조 네트워크 인터페이스에 대한 네트워크 연결 정의입니다.
- 출력 라우터에 대한 배포입니다.

송신 라우터 사용자 정의 리소스를 삭제하는 경우 Operator는 송신 라우터와 연결된 이전 목록에서 두 개의 오브젝트를 삭제합니다.

### 22.12.1.3. 배포 고려 사항

송신 라우터 Pod는 노드의 기본 네트워크 인터페이스에 추가 IP 주소와 MAC 주소를 추가합니다. 따라서 추가 주소를 허용하도록 하이퍼바이저 또는 클라우드 공급자를 구성해야 할 수 있습니다.

#### Red Hat OpenStack Platform (RHOSP)

RHOSP에서 OpenShift Container Platform을 배포하는 경우 OpenStack 환경에서 송신 라우터 포드의 IP 및 MAC 주소의 트래픽을 허용해야 합니다. 트래픽을 허용하지 않으면 통신이 실패합니다.

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

#### RHV(Red Hat Virtualization)

RHV를 사용하는 경우 가상 네트워크 인터페이스 컨트롤러(vNIC)에 대해 No Network Filter (네트워크 필터 없음)를 선택해야 합니다.

#### VMware vSphere

VMware vSphere를 사용하는 경우 vSphere 표준 스위치 보안을 위한 VMware 설명서를 참조하십시오. vSphere Web Client에서 호스트 가상 스위치를 선택하여 VMware vSphere 기본 설정을 보고 변경합니다.

특히 다음이 활성화되어 있는지 확인하십시오.

- **MAC 주소 변경**
- **위조된 전송**
- **무차별 모드 작동**

### 22.12.1.4. 장애 조치 구성

다운타임을 방지하기 위해 Cluster Network Operator는 송신 라우터 Pod를 배포 리소스로 배포합니다. 배포 이름은 egress-router-cni-deployment입니다. 배포에 해당하는 포드의 레이블은 app=egress-router-cni입니다.

배포에 사용할 새 서비스를 생성하려면 `oc expose deployment/egress-router-cni-deployment --port <port_number>` 명령을 사용하거나 다음 예와 같이 파일을 생성합니다.

```
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
```

### 22.12.2. 추가 리소스

- [리디렉션 모드에서 송신 라우터 배포](#)

### 22.13. 리디렉션 모드에서 송신 라우터 POD 배포

클러스터 관리자는 예약된 소스 IP 주소에서 지정된 대상 IP 주소로 트래픽을 리디렉션하도록 송신 라우터 포드를 배포할 수 있습니다.

송신 라우터 구현에서는 송신 라우터 **CNI(Container Network Interface)** 플러그인을 사용합니다.

#### 22.13.1. 송신 라우터 사용자 정의 리소스

송신 라우터 사용자 정의 리소스에서 송신 라우터 **Pod**에 대한 구성을 정의합니다. 다음 **YAML**은 리디렉션 모드에서 송신 라우터 구성을 위한 필드를 설명합니다.

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: <egress_router_name>
  namespace: <namespace> <.>
spec:
  addresses: [ <.>
```

```

{
  ip: "<egress_router>", <.>
  gateway: "<egress_gateway>" <.>
}
]
mode: Redirect
redirect: {
  redirectRules: [ <.>
    {
      destinationIP: "<egress_destination>",
      port: <egress_router_port>,
      targetPort: <target_port>, <.>
      protocol: <network_protocol> <.>
    },
    ...
  ],
  fallbackIP: "<egress_destination>" <.>
}

```

<.> 선택 사항: **namespace** 필드는 출력 라우터를 생성할 네임스페이스를 지정합니다. 파일 또는 명령 줄에 값을 지정하지 않으면 **default** 네임스페이스가 사용됩니다.

<.> **address** 필드는 보조 네트워크 인터페이스에서 구성할 IP 주소를 지정합니다.

<.> **ip** 필드는 노드가 송신 라우터 Pod에서 사용할 실제 네트워크의 예약된 소스 IP 주소와 넷마스크를 지정합니다. CIDR 표기법을 사용하여 IP 주소와 넷마스크를 지정합니다.

<.> **gateway** 필드는 네트워크 게이트웨이의 IP 주소를 지정합니다.

<.> 선택 사항: **redirectRules** 필드는 송신 대상 IP 주소, 송신 라우터 포트 및 프로토콜의 조합을 지정합니다. 지정된 포트 및 프로토콜의 출력 라우터에 대한 수신 연결은 대상 IP 주소로 라우팅됩니다.

<.> 선택 사항: **targetPort** 필드는 대상 IP 주소에 네트워크 포트를 지정합니다. 이 필드를 지정하지 않으면 트래픽이 도달한 동일한 네트워크 포트에 라우팅됩니다.

<.> **protocol** 필드는 TCP, UDP 또는 SCTP를 지원합니다.

<.> 선택 사항: **fallbackIP** 필드는 대상 IP 주소를 지정합니다. 리디렉션 규칙을 지정하지 않으면 송신 라우터에서 모든 트래픽을 이 폴백 IP 주소로 보냅니다. 리디렉션 규칙을 지정하면 규칙에 정의되지 않은 네트워크 포트에 대한 모든 연결이 송신 라우터에서 이 대체 IP 주소로 전송됩니다. 이 필드를 지정하지 않으면 송신 라우터는 규칙에 정의되지 않은 네트워크 포트에 대한 연결을 거부합니다.

## 송신 라우터 사양의 예

```

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
      }
    ]
  }
}

```

## 22.13.2. 리디렉션 모드에서 송신 라우터 배포

송신 라우터 pod를 배포하여 자체 예약된 소스 IP 주소에서 하나 이상의 대상 IP 주소로 트래픽을 리디렉션할 수 있습니다.

송신 라우터 pod를 추가한 후 예약된 소스 IP 주소를 사용해야 하는 클라이언트 pod는 대상 IP에 직접 연결하는 대신 송신 라우터에 연결하도록 수정해야 합니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

프로세스

1. 송신 라우터 정의를 생성합니다.
2. 다른 포드에서 송신 라우터 pod의 IP 주소를 찾을 수 있도록 하려면 다음 예제와 같이 송신 라우터를 사용하는 서비스를 만듭니다.

```

apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: web-app
      protocol: TCP
      port: 8080
  type: ClusterIP
selector:
  app: egress-router-cni <.>

```

<.> 송신 라우터의 레이블을 지정합니다. 표시된 값은 **Cluster Network Operator**에서 추가 하며 구성 불가능합니다.

서비스를 생성한 후 포드가 서비스에 연결할 수 있습니다. 송신 라우터 pod는 트래픽을 대상 IP 주소의 해당 포트로 리디렉션합니다. 이 연결은 예약된 소스 IP 주소에서 시작됩니다.

검증

**Cluster Network Operator**가 송신 라우터를 시작했는지 확인하려면 다음 절차를 완료합니다.

1.

송신 라우터에 대해 **Operator**가 생성한 네트워크 연결 정의를 확인합니다.

```
$ oc get network-attachment-definition egress-router-cni-nad
```

네트워크 연결 정의의 이름은 구성할 수 없습니다.

출력 예

```
NAME          AGE
egress-router-cni-nad 18m
```

2.

송신 라우터 **pod**에 대한 배포를 확인합니다.

```
$ oc get deployment egress-router-cni-deployment
```

배포 이름은 구성할 수 없습니다.

출력 예

```
NAME                      READY  UP-TO-DATE  AVAILABLE  AGE
egress-router-cni-deployment 1/1    1            1          18m
```

3.

송신 라우터 **pod**의 상태를 확인합니다.

```
$ oc get pods -l app=egress-router-cni
```

출력 예

NAME	READY	STATUS	RESTARTS	AGE
egress-router-cni-deployment-575465c75c-qkq6m	1/1	Running	0	18m

4. 송신 라우터 pod의 로그 및 라우팅 테이블을 확인합니다.

a. 송신 라우터 pod에 대한 노드 이름을 가져옵니다.

```
$ POD_NODENAME=$(oc get pod -l app=egress-router-cni -o jsonpath="{.items[0].spec.nodeName}")
```

b. 대상 노드에서 디버그 세션으로 들어갑니다. 이 단계는 <node\_name>-debug 라는 디버그 Pod를 인스턴스화합니다.

```
$ oc debug node/$POD_NODENAME
```

c. 디버그 셸 내에서 /host를 root 디렉터리로 설정합니다. 디버그 Pod는 Pod 내의 /host에 호스트의 루트 파일 시스템을 마운트합니다. 루트 디렉터리를 /host로 변경하면 호스트의 실행 경로에서 바이너리를 실행할 수 있습니다.

```
# chroot /host
```

d. chroot 환경 콘솔에서 송신 라우터 로그를 표시합니다.

```
# cat /tmp/egress-router-log
```

출력 예

```
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 {lfindex: 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
```

로깅 파일 위치 및 로깅 수준은 이 프로세스에 설명된 대로 **EgressRouter** 오브젝트를 생성하여 송신 라우터를 시작할 때 구성되지 않습니다.

e.

**chroot** 환경 콘솔에서 컨테이너 ID를 가져옵니다.

```
# crictl ps --name egress-router-cni-pod | awk '{print $1}'
```

출력 예

```
CONTAINER
bac9fae69ddb6
```

f.

컨테이너의 프로세스 ID를 확인합니다. 이 예에서 컨테이너 ID는 **bac9fae69ddb6**입니다.

```
# crictl inspect -o yaml bac9fae69ddb6 | grep 'pid:' | awk '{print $2}'
```

출력 예

```
68857
```

- g. 컨테이너의 네트워크 네임스페이스를 입력합니다.

```
# nsenter -n -t 68857
```

- h. 라우팅 테이블을 표시합니다.

```
# ip route
```

다음 예제 출력에서 **net1** 네트워크 인터페이스가 기본 경로입니다. 클러스터 네트워크의 트래픽은 **eth0** 네트워크 인터페이스를 사용합니다. **192.168.12.0/24** 네트워크의 트래픽은 **net1** 네트워크 인터페이스를 사용하며 예약된 소스 IP 주소 **192.168.12.99**에서 시작됩니다. 포드는 다른 모든 트래픽을 IP 주소 **192.168.12.1**의 게이트웨이로 라우팅합니다. 서비스 네트워크의 라우팅이 표시되지 않습니다.

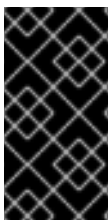
출력 예

```
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
```

## 22.14. 프로젝트에 멀티 캐스트 사용

### 22.14.1. 멀티 캐스트 정보

IP 멀티 캐스트를 사용하면 데이터가 여러 IP 주소로 동시에 브로드캐스트됩니다.



중요

현재 멀티 캐스트는 고 대역폭 솔루션이 아닌 저 대역폭 조정 또는 서비스 검색에 가장 적합합니다.

OpenShift Container Platform Pod 간 멀티 캐스트 트래픽은 기본적으로 비활성화되어 있습니다. OVN-Kubernetes 기본 CNI(Container Network Interface) 네트워크 공급자를 사용하는 경우 프로젝트

별로 멀티 캐스트를 활성화할 수 있습니다.

### 22.14.2. Pod 간 멀티 캐스트 활성화

프로젝트의 Pod 간 멀티 캐스트를 활성화할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

프로세스

- 다음 명령을 실행하여 프로젝트에 대한 멀티 캐스트를 활성화합니다. 멀티 캐스트를 활성화하려는 프로젝트의 네임스페이스로 **<namespace>**를 바꿉니다.

```
$ oc annotate namespace <namespace> \
k8s.ovn.org/multicast-enabled=true
```

작은 정보

다음 YAML을 적용하여 주석을 추가할 수도 있습니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: "true"
```

검증

프로젝트에 멀티 캐스트가 활성화되어 있는지 확인하려면 다음 절차를 완료합니다.

1. 멀티 캐스트를 활성화한 프로젝트로 현재 프로젝트를 변경합니다. **<project>**를 프로젝트 이름으로 바꿉니다.

```
$ oc project <project>
```

2.

멀티 캐스트 수신자 역할을 할 pod를 만듭니다.

```
$ 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/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat hostname && sleep inf"]
    ports:
    - containerPort: 30102
      name: mlistener
      protocol: UDP
EOF
```

3.

멀티 캐스트 발신자 역할을 할 pod를 만듭니다.

```
$ 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/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat && sleep inf"]
EOF
```

4.

새 터미널 창 또는 탭에서 멀티 캐스트 리스너를 시작합니다.

a.

Pod의 IP 주소를 가져옵니다.

```
$ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
```

b.

다음 명령을 입력하여 멀티 캐스트 리스너를 시작합니다.

```
$ oc exec mlistener -i -t -- \
  socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
  EXEC:hostname
```

5.

멀티 캐스트 송신기를 시작합니다.

a.

Pod 네트워크 IP 주소 범위를 가져옵니다.

```
$ CIDR=$(oc get Network.config.openshift.io cluster \
  -o jsonpath='{.status.clusterNetwork[0].cidr}')
```

b.

멀티 캐스트 메시지를 보내려면 다음 명령을 입력합니다.

```
$ oc exec msender -i -t -- \
  /bin/bash -c "echo | socat STDIO UDP4-
  DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
```

멀티 캐스트가 작동하는 경우 이전 명령은 다음 출력을 반환합니다.

```
mlistener
```

## 22.15. 프로젝트에 대한 멀티 캐스트 비활성화

### 22.15.1. Pod 간 멀티 캐스트 비활성화

프로젝트의 Pod 간 멀티 캐스트를 비활성화할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할을 가진 사용자로 클러스터에 로그인해야 합니다.

#### 프로세스

- 다음 명령을 실행하여 멀티 캐스트를 비활성화합니다.

```
$ oc annotate namespace <namespace> \ 1
k8s.ovn.org/multicast-enabled-
```

1

멀티 캐스트를 비활성화하려는 프로젝트의 **namespace**입니다.

#### 작은 정보

다음 **YAML**을 적용하여 주석을 삭제할 수도 있습니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: null
```

## 22.16. 네트워크 흐름 추적

클러스터 관리자는 다음 영역을 지원하기 위해 클러스터에서 **Pod** 네트워크 흐름에 대한 정보를 수집할 수 있습니다.

- **pod** 네트워크에서 수신 및 송신 트래픽을 모니터링합니다.
- 성능 문제를 해결합니다.
- 용량 계획 및 보안 감사를 위한 데이터를 수집합니다.

네트워크 흐름 컬렉션을 활성화하면 트래픽에 대한 메타데이터만 수집됩니다. 예를 들어 패킷 데이터는 수집되지 않지만 프로토콜, 소스 주소, 대상 주소, 포트 번호, 바이트 수 및 기타 패킷 수준 정보가 수집됩니다.

데이터는 다음 레코드 형식 중 하나로 수집됩니다.

- **NetFlow**
- **sFlow**
- **IPFIX**

하나 이상의 컬렉터 IP 주소와 포트 번호를 사용하여 **CNO(Cluster Network Operator)**를 구성하는 경우 **Operator**는 각 노드에 **OVS(Open vSwitch)**를 구성하여 네트워크 흐름 레코드를 각 컬렉터에 전송합니다.

여러 유형의 네트워크 흐름 수집기로 레코드를 보내도록 **Operator**를 구성할 수 있습니다. 예를 들어 **NetFlow** 컬렉터에 레코드를 보내고 레코드를 **sFlow** 수집기에 전송할 수도 있습니다.

**OVS**가 수집기에 데이터를 보내면 각 유형의 컬렉터는 동일한 레코드를 수신합니다. 예를 들어 노드의 **OVS**가 두 개의 **NetFlow** 컬렉터를 구성하는 경우 두 컬렉터에 동일한 레코드를 보냅니다. 또한 두 개의 **sFlow** 컬렉터를 구성하는 경우 두 개의 **sFlow** 수집기는 동일한 레코드를 받습니다. 그러나 각 컬렉터 유형에는 고유한 레코드 형식이 있습니다.

네트워크 흐름 데이터를 수집하고 컬렉터로 레코드를 전송하면 성능에 영향을 미칩니다. 노드는 더 느린 속도로 패킷을 처리합니다. 성능 영향이 너무 크면 컬렉터의 대상을 삭제하여 네트워크 흐름 데이터 수집 및 복원 성능을 비활성화할 수 있습니다.



#### 참고

네트워크 흐름 수집기를 활성화하면 클러스터 네트워크의 전반적인 성능에 영향을 미칠 수 있습니다.

#### 22.16.1. 네트워크 흐름 추적을 위한 네트워크 오브젝트 구성

**CNO(Cluster Network Operator)**에서 네트워크 흐름 수집기를 구성하는 필드는 다음 표에 표시되어 있습니다.

표 22.8. 네트워크 흐름 구성

필드	유형	설명
<code>metadata.name</code>	<code>string</code>	CNO 개체 이름입니다. 이 이름은 항상 <b>cluster</b> 입니다.

필드	유형	설명
----	----	----

<b>spec.exportNetworkFlows</b>	<b>object</b>	<b>netFlow,sFlow</b> 또는 <b>ipfix</b> 중 하나 이상입니다.
<b>spec.exportNetworkFlows.netFlow.collectors</b>	<b>array</b>	최대 10개의 컬렉터에 대한 IP 주소 및 네트워크 포트 쌍 목록입니다.
<b>spec.exportNetworkFlows.sFlow.collectors</b>	<b>array</b>	최대 10개의 컬렉터에 대한 IP 주소 및 네트워크 포트 쌍 목록입니다.
<b>spec.exportNetworkFlows.ipfix.collectors</b>	<b>array</b>	최대 10개의 컬렉터에 대한 IP 주소 및 네트워크 포트 쌍 목록입니다.

CNO에 다음 매니페스트를 적용한 후 Operator는 클러스터의 각 노드에서 OVS(Open vSwitch)를 구성하여 네트워크 흐름 레코드를 192.168.1.99:2056에서 수신 대기 중인 NetFlow 수집기에 전송합니다.

네트워크 흐름 추적을 위한 구성 예

```

apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
    
```

### 22.16.2. 네트워크 흐름 수집기 추가

클러스터 관리자는 네트워크 흐름 수집기로 네트워크 흐름 메타데이터를 전송하도록 CNO(Cluster Network Operator)를 구성할 수 있습니다.

사전 요구 사항



- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 흐름 수집기가 있고 수신 대기하는 **IP** 주소와 포트를 알고 있습니다.

### 프로세스

1. 네트워크 흐름 수집기 유형과 컬렉터의 **IP** 주소 및 포트 정보를 지정하는 패치 파일을 생성합니다.

```
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

2. 네트워크 흐름 수집기를 사용하여 **CNO**를 구성합니다.

```
$ oc patch network.operator cluster --type merge -p "$(cat <file_name>.yaml)"
```

출력 예

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

### 검증

일반적으로 검증은 필요하지 않습니다. 다음 명령을 실행하여 각 노드의 **OVS(Open vSwitch)**가 하나 이상의 컬렉터에 네트워크 흐름 레코드를 전송하도록 구성되어 있는지 확인할 수 있습니다.

1. **Operator** 구성을 보고 **exportNetworkFlows** 필드가 구성되었는지 확인합니다.

```
$ oc get network.operator cluster -o jsonpath="{.spec.exportNetworkFlows}"
```

출력 예

```
{"netFlow":{"collectors":["192.168.1.99:2056"]}}
```

2.

각 노드에서 OVS의 네트워크 흐름 구성을 확인합니다.

```
$ 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-node $pod \
-- bash -c 'for type in ipfix sflow netflow ; do ovs-vsctl find $type ; done';
done
```

출력 예

```
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"]-

...
```

### 22.16.3. 네트워크 흐름 수집기의 모든 대상 삭제

클러스터 관리자는 네트워크 흐름 수집기로 네트워크 흐름 메타데이터를 중지하도록 **CNO(Cluster Network Operator)**를 구성할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

#### 프로세스

1. 모든 네트워크 흐름 수집기를 제거합니다.

```
$ oc patch network.operator cluster --type='json' \
  -p='[{"op": "remove", "path": "/spec/exportNetworkFlows"}]'
```

출력 예

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

#### 22.16.4. 추가 리소스

- [Network \[operator.openshift.io/v1\]](#)

#### 22.17. 하이브리드 네트워킹 구성

클러스터 관리자는 **Linux** 및 **Windows** 노드에서 각각 **Linux** 및 **Windows** 워크로드를 호스팅할 수 있도록 **OVN-Kubernetes CNI(Container Network Interface)** 클러스터 네트워크 공급자를 구성할 수 있습니다.

##### 22.17.1. OVN-Kubernetes로 하이브리드 네트워킹 구성

**OVN-Kubernetes**에서 하이브리드 네트워킹을 사용하도록 클러스터를 구성할 수 있습니다. 이를 통해 다양한 노드 네트워킹 구성을 지원하는 하이브리드 클러스터를 사용할 수 있습니다. 예를 들어 클러스터

에서 **Linux** 및 **Windows** 노드를 모두 실행하려면 이 작업이 필요합니다.



### 중요

클러스터를 설치하는 동안 **OVN-Kubernetes**를 사용하여 하이브리드 네트워킹을 구성해야 합니다. 설치 프로세스 후에는 하이브리드 네트워킹으로 전환할 수 없습니다.

### 사전 요구 사항

- install-config.yaml** 파일에 **networking.networkType** 매개변수의 **OVNKubernetes**가 정의되어 있어야 합니다. 자세한 내용은 선택한 클라우드 공급자에서 **OpenShift Container Platform** 네트워크 사용자 정의 설정에 필요한 설치 문서를 참조하십시오.

### 프로세스

- 설치 프로그램이 포함된 디렉터리로 변경하고 매니페스트를 생성합니다.

```
$ ./openshift-install create manifests --dir <installation_directory>
```

다음과 같습니다.

**<installation\_directory>**

클러스터의 **install-config.yaml** 파일이 포함된 디렉토리의 이름을 지정합니다.

- <installation\_directory>/manifests/** 디렉토리에 **cluster-network-03-config.yml**이라는 stub 매니페스트 파일을 만듭니다.

```
$ cat <<EOF > <installation_directory>/manifests/cluster-network-03-config.yml
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
EOF
```

다음과 같습니다.

**<installation\_directory>**

클러스터의 manifests/ 디렉터리가 포함된 디렉터리 이름을 지정합니다.

3.

편집기에서 **cluster-network-03-config.yml** 파일을 열고 다음 예와 같이 하이브리드 네트워킹을 사용하여 OVN-Kubernetes를 구성합니다.

하이브리드 네트워킹 구성 지정

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      hybridOverlayConfig:
        hybridClusterNetwork: ❶
        - cidr: 10.132.0.0/14
          hostPrefix: 23
        hybridOverlayVXLANPort: 9898 ❷
```

❶

추가 오버레이 네트워크의 노드에 사용되는 CIDR 구성을 지정합니다. hybridClusterNetwork CIDR은 clusterNetwork CIDR과 중복될 수 없습니다.

❷

추가 오버레이 네트워크에 대한 사용자 정의 VXLAN 포트를 지정합니다. 이는 vSphere에 설치된 클러스터에서 Windows 노드를 실행해야 하며 다른 클라우드 공급자에 대해 구성해서는 안 됩니다. 사용자 정의 포트는 기본 4789 포트를 제외한 모든 오픈 포트일 수 있습니다. 이 요구 사항에 대한 자세한 내용은 [호스트 간의 포트 투 포트 연결 중단에 대한 Microsoft 문서](#)를 참조하십시오.



참고

**Windows Server LTSC(Long-Term Servicing Channel):** Windows Server 2019는 사용자 지정 hybridOverlayVXLANPort 값이 있는 클러스터에서 지원되지 않습니다. 이 Windows 서버 버전은 사용자 지정 VXLAN 포트를 선택하는 것을 지원하지 않기 때문입니다.

4.

**cluster-network-03-config.yml** 파일을 저장하고 텍스트 편집기를 종료합니다.오.

5.

선택사항: **manifests/cluster-network-03-config.yml** 파일을 백업합니다. 설치 프로그램은 클러스터를 생성할 때 **manifests/** 디렉토리를 삭제합니다.

추가 설치 구성을 완료한 후 클러스터를 생성합니다. 설치 프로세스가 완료되면 하이브리드 네트워킹이 활성화됩니다.

### 22.17.2. 추가 리소스

- [Windows 컨테이너 워크로드 이해](#)
- [Windows 컨테이너 워크로드 활성화](#)
- [네트워크 사용자 지정으로 AWS에 클러스터 설치](#)
- [네트워크 사용자 지정 설정을 사용하여 Azure에 클러스터 설치](#)

## 23장. 경로 구성

### 23.1. 경로 구성

#### 23.1.1. HTTP 기반 경로 생성

경로를 사용하면 공용 URL에서 애플리케이션을 호스팅할 수 있습니다. 애플리케이션의 네트워크 보안 구성에 따라 보안 또는 보안되지 않을 수 있습니다. HTTP 기반 경로는 기본 HTTP 라우팅 프로토콜을 사용하고 비보안 애플리케이션 포트에 서비스를 노출하는 비보안 경로입니다.

다음 절차에서는 **hello-openshift** 애플리케이션을 예제로 사용하여 웹 애플리케이션에 대한 간단한 HTTP 기반 경로를 생성하는 방법을 설명합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- 관리자로 로그인되어 있습니다.
- 포트에서 트래픽을 수신 대기하는 포트 및 TCP 엔드포인트를 노출하는 웹 애플리케이션이 있습니다.

#### 프로세스

1. 다음 명령을 실행하여 **hello-openshift** 라는 프로젝트를 생성합니다.

```
$ oc new-project hello-openshift
```

2. 다음 명령을 실행하여 프로젝트에 **Pod**를 생성합니다.

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. 다음 명령을 실행하여 **hello-openshift** 라는 서비스를 생성합니다.

```
$ oc expose pod/hello-openshift
```

4.

다음 명령을 실행하여 **hello-openshift** 애플리케이션에 대한 비보안 경로를 생성합니다.

```
$ oc expose svc hello-openshift
```

검증

•

생성한 경로 리소스가 있는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get routes -o yaml <name of resource> 1
```

1

이 예에서 경로는 **hello-openshift** 로 이름이 지정됩니다.

생성된 비보안 경로에 대한 샘플 YAML 정의:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: hello-openshift-hello-openshift.<Ingress_Domain> 1
  port:
    targetPort: 8080 2
  to:
    kind: Service
    name: hello-openshift
```

1

**<Ingress\_Domain >**은 기본 수신 도메인 이름입니다. **ingresses.config/cluster** 오브젝트는 설치 중에 생성되며 변경할 수 없습니다. 다른 도메인을 지정하려면 **appsDomain** 옵션을 사용하여 대체 클러스터 도메인을 지정할 수 있습니다.

2

**targetPort** 는 이 경로가 가리키는 서비스에서 선택한 **Pod**의 대상 포트입니다.





## 참고

기본 **Ingress** 도메인을 표시하려면 다음 명령을 실행합니다.

```
$ oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

### 23.1.2. Ingress 컨트롤러 분할을 위한 경로 생성

경로를 사용하면 **URL**에서 애플리케이션을 호스팅할 수 있습니다. 이 경우 호스트 이름이 설정되지 않고 경로는 대신 하위 도메인을 사용합니다. 하위 도메인을 지정하면 경로를 노출하는 **Ingress** 컨트롤러의 도메인을 자동으로 사용합니다. 여러 **Ingress** 컨트롤러에서 경로가 노출되는 경우 경로는 여러 **URL**에서 호스팅됩니다.

다음 절차에서는 **hello-openshift** 애플리케이션을 예제로 사용하여 **Ingress** 컨트롤러 분할에 대한 경로를 생성하는 방법을 설명합니다.

**Ingress** 컨트롤러 분할은 들어오는 트래픽 부하를 일련의 **Ingress** 컨트롤러에 균형 있게 분배하고 트래픽을 특정 **Ingress** 컨트롤러에 격리할 때 유용합니다. 예를 들어, 회사 **A**는 하나의 **Ingress** 컨트롤러로, 회사 **B**는 다른 **Ingress** 컨트롤러로 이동합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- 프로젝트 관리자로 로그인되어 있습니다.
- 포트에서 트래픽을 수신 대기하는 포트 및 **HTTP** 또는 **TLS** 엔드포인트를 노출하는 웹 애플리케이션이 있습니다.
- 분할을 위해 **Ingress** 컨트롤러가 구성되어 있습니다.

#### 프로세스

1. 다음 명령을 실행하여 **hello-openshift** 라는 프로젝트를 생성합니다.

```
$ oc new-project hello-openshift
```

2.

다음 명령을 실행하여 프로젝트에 **Pod**를 생성합니다.

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3.

다음 명령을 실행하여 **hello-openshift** 라는 서비스를 생성합니다.

```
$ oc expose pod/hello-openshift
```

4.

**hello-openshift-route.yaml** 이라는 경로 정의를 생성합니다.

분할을 위해 생성된 경로의 **YAML** 정의:

```
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

레이블 키와 해당 레이블 값은 **Ingress** 컨트롤러에 지정된 라벨과 일치해야 합니다. 이 예제에서 **Ingress** 컨트롤러에는 레이블 키와 값 **type: sharded** 가 있습니다.

2

경로는 **subdomain** 필드의 값을 사용하여 노출됩니다. 하위 도메인 필드를 지정할 때 호스트 이름을 설정되지 않은 상태로 두어야 합니다. **host** 및 **subdomain** 필드를 모두 지정

하면 경로는 호스트 필드의 값을 사용하고 **subdomain** 필드를 무시합니다.

5.

다음 명령을 실행하여 **hello-openshift-route.yaml** 을 사용하여 **hello-openshift** 애플리케이션에 대한 경로를 생성합니다.

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

검증

•

다음 명령을 사용하여 경로의 상태를 가져옵니다.

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

생성된 **Route** 리소스는 다음과 유사해야 합니다.

출력 예

```
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

**Ingress** 컨트롤러 또는 라우터에서 경로를 노출하는 데 사용하는 호스트 이름입니다. **host** 필드의 값은 **Ingress** 컨트롤러에 의해 자동으로 결정되며 도메인을 사용합니다. 이 예제에서 **Ingress** 컨트롤러의 도메인은 `<apps-sharded.basedomain.example.net>`입니다.

2

**Ingress** 컨트롤러의 호스트 이름입니다.

3

**Ingress** 컨트롤러의 이름입니다. 이 예제에서 **Ingress** 컨트롤러에는 이름 **sharded** 가 있습니다.

### 23.1.3. 경로 시간 초과 구성

**SLA(Service Level Availability)** 목적에 필요한 낮은 시간 초과 또는 백엔드가 느린 경우 높은 시간 초과가 필요한 서비스가 있는 경우 기존 경로에 대한 기본 시간 초과를 구성할 수 있습니다.

#### 사전 요구 사항

- 실행 중인 클러스터에 배포된 **Ingress** 컨트롤러가 필요합니다.

#### 프로세스

1. **oc annotate** 명령을 사용하여 경로에 시간 초과를 추가합니다.

```
$ oc annotate route <route_name> \
  --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> 1
```

1

지원되는 시간 단위는 마이크로초(us), 밀리초(ms), 초(s), 분(m), 시간(h) 또는 일(d)입니다.

다음 예제는 이름이 **myroute**인 경로에서 2초의 시간 초과를 설정합니다.

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

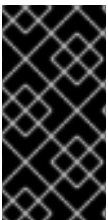
### 23.1.4. HSTS(HTTP Strict Transport Security)

**HSTS(HTTP Strict Transport Security)** 정책은 라우트 호스트에서 **HTTPS** 트래픽만 허용됨을 브라우저 클라이언트에 알리는 보안 강화 정책입니다. 또한 **HSTS**는 **HTTP** 리디렉션을 사용하지 않고 **HTTPS** 전송 신호를 통해 웹 트래픽을 최적화합니다. **HSTS**는 웹사이트와의 상호 작용을 가속화하는 데 유용합니다.

**HSTS** 정책이 적용되면 **HSTS**는 사이트의 **HTTP** 및 **HTTPS** 응답에 **Strict Transport Security** 헤더를 추가합니다. 경로에서 **insecureEdgeTerminationPolicy** 값을 사용하여 **HTTP**를 **HTTPS**로 리디렉션할 수 있습니다. **HSTS**를 적용하면 클라이언트는 요청을 전송하기 전에 **HTTP URL**의 모든 요청을 **HTTPS**로 변경하여 리디렉션이 필요하지 않습니다.

클러스터 관리자는 다음을 수행하도록 **HSTS**를 구성할 수 있습니다.

- 경로당 **HSTS** 활성화
- 라우팅당 **HSTS** 비활성화
- 도메인당 **HSTS** 시행, 도메인 집합 또는 도메인과 함께 네임스페이스 라벨 사용



#### 중요

**HSTS**는 보안 경로(엣지 종료 또는 재암호화)에서만 작동합니다. **HTTP** 또는 패스스루(**passthrough**) 경로에서는 구성이 유효하지 않습니다.

#### 23.1.4.1. 라우팅당 **HSTS(HTTP Strict Transport Security)** 활성화

**HSTS(HTTP Strict Transport Security)**는 **HAProxy** 템플릿에 구현되고 [haproxy.router.openshift.io/hsts\\_header](https://haproxy.router.openshift.io/hsts_header) 주석이 있는 에지 및 재암호화 경로에 적용됩니다.

#### 사전 요구 사항

- 프로젝트에 대한 관리자 권한이 있는 사용자로 클러스터에 로그인되어 있습니다.
- **oc CLI**를 설치했습니다.

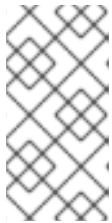
#### 프로세스

- 경로에서 **HSTS**를 활성화하려면 `haproxy.router.openshift.io/hsts_header` 값을 **edge-terminated** 또는 **re-encrypt** 경로에 추가합니다. `oc annotate` 툴을 사용하여 다음 명령을 실행하여 이 작업을 수행할 수 있습니다.

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000;includeSubDomains;preload" 1
```

1

이 예에서 최대 사용 기간은 **31536000 ms**로 설정되며 이는 약 **8시간** 및 반 시간입니다.



참고

이 예에서 등호(=)는 따옴표 안에 있습니다. 이는 `annotate` 명령을 올바르게 실행하는 데 필요합니다.

주석으로 구성된 경로 예

```
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

필수 항목입니다. **Max-age**는 **HSTS** 정책이 적용되는 시간(초)을 측정합니다. **0**으로 설정하면 정책이 무효화됩니다.

2

3

선택 사항: `max-age`가 0보다 크면 `haproxy.router.openshift.io/hsts_header`에 `preload`를 추가하여 외부 서비스에서 이 사이트를 HSTS 사전 로드 목록에 포함할 수 있습니다. 예를 들어 Google과 같은 사이트는 `preload`가 설정된 사이트 목록을 구성할 수 있습니다. 그런 다음 브라우저는 이 목록을 사용하여 사이트와 상호 작용하기 전에 HTTPS를 통해 통신할 수 있는 사이트를 결정할 수 있습니다. `preload`를 설정하지 않으면 브라우저가 HTTPS를 통해 사이트와 상호 작용하여 헤더를 가져와야 합니다.

#### 23.1.4.2. 라우팅당 HSTS(HTTP Strict Transport Security) 비활성화

경로당 HSTS(HTTP Strict Transport Security)를 비활성화하려면 경로 주석에서 `max-age` 값을 0으로 설정할 수 있습니다.

##### 사전 요구 사항

- 프로젝트에 대한 관리자 권한이 있는 사용자로 클러스터에 로그인되어 있습니다.
- `oc CLI`를 설치했습니다.

##### 프로세스

- HSTS를 비활성화하려면 다음 명령을 입력하여 경로 주석의 `max-age` 값을 0으로 설정합니다.

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

## 작은 정보

다음 **YAML**을 적용하여 구성 맵을 만들 수 있습니다.

### 경로당 HSTS 비활성화 예

```

metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=0

```

- 네임스페이스의 모든 경로에 대해 **HSTS**를 비활성화하려면 다음 명령을 입력합니다.

```

$ oc annotate route --all -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"

```

## 검증

1. 모든 경로에 대한 주석을 쿼리하려면 다음 명령을 입력합니다.

```

$ 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 }}'

```

### 출력 예

```

Name: routename HSTS: max-age=0

```

### 23.1.4.3. 도메인별 HSTS(HTTP Strict Transport Security) 적용

도메인당 **HSTS(HTTP Strict Transport Security)**를 적용하여 보안 경로에 **requiredHSTSPolicies** 레코드를 **Ingress** 사양에 추가하여 **HSTS** 정책 구성을 캡처합니다.



**HSTS**를 적용하도록 **requiredHSTSPolicy**를 구성하는 경우 규정 준수 **HSTS** 정책 주석을 사용하여 새로 생성된 경로를 구성해야 합니다.



#### 참고

준수하지 않는 **HSTS** 경로를 사용하여 업그레이드된 클러스터를 처리하려면 소스에서 매니페스트를 업데이트하고 업데이트를 적용할 수 있습니다.



#### 참고

**oc expose route** 또는 **oc create route** 명령을 사용하여 이러한 명령의 **API**에서 주석을 수락하지 않기 때문에 **HSTS**를 적용하는 도메인에 경로를 추가할 수 없습니다.



#### 중요

**HSTS**는 전역의 모든 경로에 **HSTS**가 요청되는 경우에도 비보안 또는 비**TLS** 경로에는 적용되지 않습니다.

#### 사전 요구 사항

- 프로젝트에 대한 관리자 권한이 있는 사용자로 클러스터에 로그인되어 있습니다.
- **oc CLI**를 설치했습니다.

#### 프로세스

1. **Ingress** 구성 파일을 편집합니다.

```
$ oc edit ingresses.config.openshift.io/cluster
```

#### **HSTS** 정책 예

```
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: 7
  - 'abc.example.com'
  - '*xyz.example.com'
namespaceSelector:
  matchLabels: {}
maxAge: {}
preloadPolicy: NoOpinion
includeSubDomainsPolicy: RequireNoIncludeSubDomains

```

1

필수 항목입니다. **requiredHSTSPolicies**는 순서대로 검증되고 일치하는 첫 번째 **domainPatterns**가 적용됩니다.

2 7

필수 항목입니다. 하나 이상의 **domainPatterns** 호스트 이름을 지정해야 합니다. 도메인 수를 나열할 수 있습니다. 다른 **domainPatterns**에 대한 적용 옵션의 여러 섹션을 포함할 수 있습니다.

3

선택 사항: **namespaceSelector**를 포함하는 경우 경로가 있는 프로젝트의 레이블과 일치하여 경로에 설정된 **HSTS** 정책을 적용해야 합니다. **namespaceSelector**만 일치하고 **domainPatterns**와 일치하지 않는 경로는 검증되지 않습니다.

4

필수 항목입니다. **Max-age**는 **HSTS** 정책이 적용되는 시간(초)을 측정합니다. 이 정책 설정을 사용하면 최소 및 최대 **max-age**를 적용할 수 있습니다.

•

**largestMaxAge** 값은 0에서 2147483647 사이여야 합니다. 지정되지 않은 상태로 둘 수 있습니다. 즉, 상한이 적용되지 않습니다.

- **smallestMaxAge** 값은 0에서 2147483647 사이여야 합니다. 문제 해결을 위해 **HSTS**를 비활성화하려면 0을 입력합니다. **HSTS**를 비활성화하지 않으려면 1을 입력합니다. 지정되지 않은 상태로 둘 수 있습니다. 즉, 더 낮은 제한이 적용되지 않습니다.

## 5

선택 사항: `haproxy.router.openshift.io/hsts_header`에 **preload**를 포함하면 외부 서비스에서 이 사이트를 **HSTS** 사전 로드 목록에 포함할 수 있습니다. 그런 다음 브라우저는 이 목록을 사용하여 사이트와 상호 작용하기 전에 **HTTPS**를 통해 통신할 수 있는 사이트를 결정할 수 있습니다. **preload**를 설정하지 않으면 브라우저가 헤더를 얻기 위해 사이트와 한번 이상 상호 작용해야 합니다. 다음 중 하나로 **preload**를 설정할 수 있습니다.

- **RequirePreload: RequiredHSTSPolicy**에 **preload**가 필요합니다.
- **RequireNoPreload: RequiredHSTSPolicy**에서 **preload**를 금지합니다.
- **NoOpinion:preload**는 **RequiredHSTSPolicy**에 중요하지 않습니다.

## 6

선택 사항: **includeSubDomainsPolicy**는 다음 중 하나를 사용하여 설정할 수 있습니다.

- **RequireIncludeSubDomains:includeSubDomains**는 **RequiredHSTSPolicy**에 필요합니다.
- **RequireNoIncludeSubDomains:includeSubDomains**는 **RequiredHSTSPolicy**에서 금지합니다.
- **NoOpinion:includeSubDomains**는 **RequiredHSTSPolicy**와 관련이 없습니다.

## 2.

**oc annotate command**를 입력하여 클러스터 또는 특정 네임스페이스에 **HSTS**를 적용할 수 있습니다.

- 클러스터의 모든 경로에 **HSTS**를 적용하려면 **oc annotate command**를 입력합니다.

예를 들면 다음과 같습니다.

```
$ oc annotate route --all --all-namespaces --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

- 특정 네임스페이스의 모든 경로에 HSTS를 적용하려면 `oc annotate command`를 입력합니다. 예를 들면 다음과 같습니다.

```
$ oc annotate route --all -n my-namespace --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

## 검증

구성한 HSTS 정책을 검토할 수 있습니다. 예를 들면 다음과 같습니다.

- 필요한 HSTS 정책에 대한 `maxAge` 세트를 검토하려면 다음 명령을 입력합니다.

```
$ oc get clusteroperator/ingress -n openshift-ingress-operator -o jsonpath='{range
.spec.requiredHSTSPolicies[*]}
{.spec.requiredHSTSPolicies.maxAgePolicy.largestMaxAge}{"\n"}{end}'
```

- 모든 경로에서 HSTS 주석을 검토하려면 다음 명령을 입력합니다.

```
$ 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}'
```

출력 예

```
Name: <_routename_> HSTS: max-age=31536000;preload;includeSubDomains
```

### 23.1.5. 처리량 문제 해결 방법

OpenShift Container Platform을 사용하여 애플리케이션을 배포하면 특정 서비스 간에 대기 시간이 비정상적으로 길어지는 등 네트워크 처리량 문제가 발생할 수 있습니다.

Pod 로그에 문제의 원인이 표시되지 않는 경우 다음 방법을 사용하여 성능 문제를 분석하십시오.

- ping 또는 tcpdump 와 같은 패킷 Analyzer를 사용하여 Pod와 해당 노드 간 트래픽을 분석합니다.

예를 들어 각 pod에서 tcpdump 도구를 실행하여 문제의 원인이 되는 동작을 재현합니다. Pod에서 나가거나 Pod로 들어오는 트래픽의 대기 시간을 분석하기 위해 전송 및 수신 타임 스탬프를 비교하려면 전송 캡처와 수신 캡처를 둘 다 검토하십시오. 다른 Pod, 스토리지 장치 또는 데이터 플레인의 트래픽으로 노드 인터페이스가 과부하된 경우 OpenShift Container Platform에서 대기 시간이 발생할 수 있습니다.

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> 1
```

1

podip은 Pod의 IP 주소입니다. oc get pod <pod\_name> -o wide 명령을 실행하여 Pod의 IP 주소를 가져옵니다.

tcpdump 명령은 /tmp/dump.pcap 에 이 두 포트 간의 모든 트래픽을 포함하는 파일을 생성합니다. 문제가 재현되기 직전에 Analyzer를 실행하고 문제가 재현된 직후 Analyzer를 중지하여 파일 크기를 최소화할 수 있습니다. 다음을 사용하여 노드 간에 패킷 Analyzer를 실행할 수도 있습니다(방정식에서 SDN 구현).

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- 스트리밍 처리량 및 UDP 처리량을 측정하려면 iperf 와 같은 대역폭 측정 도구를 사용합니다. 먼저 Pod에서 툴을 실행한 다음 노드에서 실행하면 병목 현상을 찾습니다.

○

iperf 설치 및 사용에 대한 자세한 내용은 이 [Red Hat 솔루션](#)을 참조하십시오.

- 경우에 따라 클러스터는 대기 시간 문제로 인해 라우터 Pod가 비정상인 노드를 비정상적으로 표시할 수 있습니다. 작업을 수행하기 전에 클러스터에서 노드의 상태 업데이트를 기다리는 빈도를 조정하려면 작업자 대기 시간 프로필을 사용합니다.

- 클러스터가 더 짧은 대기 시간 및 대기 시간이 짧은 노드를 지정하는 경우 라우터 Pod의 배치를 제어하도록 Ingress 컨트롤러에서 spec.nodePlacement 필드를 구성합니다.

- 대기 시간 급증 또는 원격 작업자로의 처리량 감소
- **Ingress** 컨트롤러 구성 매개변수

### 23.1.6. 쿠키를 사용하여 경로 상태 유지

**OpenShift Container Platform**은 모든 트래픽이 동일한 끝점에 도달하도록 하여 스테이트풀 (stateful) 애플리케이션 트래픽을 사용할 수 있는 고정 세션을 제공합니다. 그러나 재시작, 스케일링 또는 구성 변경 등으로 인해 끝점 pod가 종료되면 이러한 상태 저장 특성이 사라질 수 있습니다.

**OpenShift Container Platform**에서는 쿠키를 사용하여 세션 지속성을 구성할 수 있습니다. **Ingress** 컨트롤러에서는 사용자 요청을 처리할 끝점을 선택하고 세션에 대한 쿠키를 생성합니다. 쿠키는 요청에 대한 응답으로 다시 전달되고 사용자는 세션의 다음 요청과 함께 쿠키를 다시 보냅니다. 쿠키는 세션을 처리하는 끝점을 **Ingress** 컨트롤러에 알려 클라이언트 요청이 쿠키를 사용하여 동일한 Pod로 라우팅되도록 합니다.



#### 참고

**HTTP** 트래픽을 볼 수 없기 때문에 통과 경로에서 쿠키를 설정할 수 없습니다. 대신, 백엔드를 결정하는 소스 IP 주소를 기반으로 번호가 계산됩니다.

백엔드가 변경되면 트래픽을 잘못된 서버로 전달하여 고정을 줄일 수 있습니다. 소스 IP를 숨기는 로드 밸런서를 사용하는 경우 모든 연결에 대해 동일한 번호가 설정되고 트래픽이 동일한 pod로 전송됩니다.

#### 23.1.6.1. 쿠키를 사용하여 경로에 주석 달기

쿠키 이름을 설정하여 경로에 자동 생성되는 기본 쿠키 이름을 덮어쓸 수 있습니다. 그러면 경로 트래픽을 수신하는 애플리케이션에서 쿠키 이름을 확인할 수 있게 됩니다. 쿠키를 삭제하여 다음 요청에서 끝점을 다시 선택하도록 할 수 있습니다. 그러므로 서버에 과부하가 걸리면 클라이언트의 요청을 제거하고 재분배합니다.

#### 프로세스

1. 지정된 쿠키 이름으로 경로에 주석을 달니다.

```
$ oc annotate route <route_name> router.openshift.io/cookie_name="<cookie_name>"
```

다음과 같습니다.

**<route\_name>**

경로 이름을 지정합니다.

**<cookie\_name>**

쿠키 이름을 지정합니다.

예를 들어 쿠키 이름 `my_cookie`로 `my_route` 경로에 주석을 달 수 있습니다.

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2.

경로 호스트 이름을 변수에 캡처합니다.

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

다음과 같습니다.

**<route\_name>**

경로 이름을 지정합니다.

3.

쿠키를 저장한 다음 경로에 액세스합니다.

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

경로에 연결할 때 이전 명령으로 저장된 쿠키를 사용합니다.

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

### 23.1.7. 경로 기반 라우터

경로 기반 라우터는 **URL**과 비교할 수 있는 경로 구성 요소를 지정하며 이를 위해 라우트의 트래픽이 **HTTP** 기반이어야 합니다. 따라서 동일한 호스트 이름을 사용하여 여러 경로를 제공할 수 있으며 각각 다

른 경로가 있습니다. 라우터는 가장 구체적인 경로를 기반으로 하는 라우터와 일치해야 합니다. 그러나 이는 라우터 구현에 따라 다릅니다.

다음 표에서는 경로 및 액세스 가능성을 보여줍니다.

표 23.1. 경로 가용성

경로	비교 대상	액세스 가능
www.example.com/test	www.example.com/test	있음
	www.example.com	없음
www.example.com/test 및 www.example.com	www.example.com/test	있음
	www.example.com	있음
www.example.com	www.example.com/text	예 (경로가 아닌 호스트에 의해 결정됨)
	www.example.com	있음

경로가 있는 보안되지 않은 라우터

```

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
    
```

**1**

경로는 경로 기반 라우터에 대해 추가된 유일한 속성입니다.





## 참고

라우터가 해당 경우 TLS를 종료하지 않고 요청 콘텐츠를 읽을 수 없기 때문에 패스스루 TLS를 사용할 때 경로 기반 라우팅을 사용할 수 없습니다.

## 23.1.8. 경로별 주석

**Ingress** 컨트롤러는 노출하는 모든 경로에 기본 옵션을 설정할 수 있습니다. 개별 경로는 주석에 특정 구성을 제공하는 방식으로 이러한 기본값 중 일부를 덮어쓸 수 있습니다. **Red Hat**은 **operator** 관리 경로에 경로 주석 추가를 지원하지 않습니다.



## 중요

여러 소스 IP 또는 서브넷이 있는 화이트리스트를 생성하려면 공백으로 구분된 목록을 사용합니다. 다른 구분 기호 유형으로 인해 경고 또는 오류 메시지 없이 목록이 무시됩니다.

표 23.2. 경로 주석

변수	설명	기본값으로 사용되는 환경 변수
<code>haproxy.router.openshift.io/balance</code>	로드 밸런싱 알고리즘을 설정합니다. 사용 가능한 옵션은 <b>random</b> , <b>source</b> , <b>roundrobin</b> , <b>leastconn</b> 입니다. 기본값은 <b>random</b> 입니다.	경유 경로의 경우 <b>ROUTER_TCP_BALANCE_SCHEME</b> 입니다. 그 외에는 <b>ROUTER_LOAD_BALANCE_ALGORITHM</b> 을 사용하십시오.
<code>haproxy.router.openshift.io/disable_cookies</code>	쿠키를 사용하여 관련 연결을 추적하지 않도록 설정합니다. <b>'true'</b> 또는 <b>'TRUE'</b> 로 설정된 경우 balance 알고리즘은 들어오는 각 HTTP 요청에 대해 연결을 제공하는 백엔드를 선택하는 데 사용됩니다.	
<code>router.openshift.io/cookie_name</code>	이 경로에 사용할 선택적 쿠키를 지정합니다. 이름은 대문자와 소문자, 숫자, <b>'_'</b> , <b>'-'</b> 의 조합으로 구성해야 합니다. 기본값은 경로의 해시된 내부 키 이름입니다.	

변수	설명	기본값으로 사용되는 환경 변수
<p><b>haproxy.router.openshift.io/pod-concurrent-connections</b></p>	<p>라우터에서 백업 pod로 허용되는 최대 연결 수를 설정합니다. 참고: Pod가 여러 개인 경우 각각 이 수만큼의 연결이 있을 수 있습니다. 라우터가 여러 개 있고 조정이 이루어지지 않는 경우에는 각각 이 횟수만큼 연결할 수 있습니다. 설정하지 않거나 0으로 설정하면 제한이 없습니다.</p>	
<p><b>haproxy.router.openshift.io/rate-limit-connections</b></p>	<p>'true' 또는 'TRUE' 를 설정하면 경로당 특정 백엔드의 stick-tables를 통해 구현되는 속도 제한 기능을 사용할 수 있습니다. 참고: 이 주석을 사용하면 DDoS(Distributed-of-service) 공격에 대한 기본 보호 기능이 제공됩니다.</p>	
<p><b>haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp</b></p>	<p>동일한 소스 IP 주소를 통해 만든 동시 TCP 연결 수를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 DDoS(Distributed-of-service) 공격에 대한 기본 보호 기능이 제공됩니다.</p>	
<p><b>haproxy.router.openshift.io/rate-limit-connections.rate-http</b></p>	<p>동일한 소스 IP 주소가 있는 클라이언트에서 HTTP 요청을 수행할 수 있는 속도를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 DDoS(Distributed-of-service) 공격에 대한 기본 보호 기능이 제공됩니다.</p>	
<p><b>haproxy.router.openshift.io/rate-limit-connections.rate-tcp</b></p>	<p>동일한 소스 IP 주소가 있는 클라이언트에서 TCP 연결을 수행할 수 있는 속도를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 DDoS(Distributed-of-service) 공격에 대한 기본 보호 기능이 제공됩니다.</p>	
<p><b>haproxy.router.openshift.io/timeout</b></p>	<p>경로에 대한 서버 쪽 타임아웃을 설정합니다. (TimeUnits)</p>	<p><b>ROUTER_DEFAULT_SERVER_TIMEOUT</b></p>

변수	설명	기본값으로 사용되는 환경 변수
<code>haproxy.router.openshift.io/timeout-tunnel</code>	이 제한 시간은 터널 연결(예: 일반 텍스트, 에지, 재암호화 경로 또는 패스스루)에 적용됩니다. 일반 텍스트, 에지 또는 재암호화 경로 유형이 있는 이 주석은 기존 시간 초과 값과 함께 시간 제한 터널로 적용됩니다. passthrough 경로 유형의 경우 주석은 기존 시간 초과 값 세트보다 우선합니다.	<code>ROUTER_DEFAULT_TUNNEL_TIMEOUT</code>
<code>ingresses.config/cluster ingress.operator.openshift.io/hard-stop-after</code>	IngressController 또는 Ingress 구성을 설정할 수 있습니다. 이 주석은 라우터를 재배포하고 clean soft-stop를 수행할 수 있는 최대 시간을 정의하는 haproxy <b>hard-stop-after</b> global 옵션을 내보내도록 HA 프록시를 구성합니다.	<code>ROUTER_HARD_STOP_AFTER</code>
<code>router.openshift.io/haproxy.health.check.interval</code>	백엔드 상태 점검 간격을 설정합니다. (TimeUnits)	<code>ROUTER_BACKEND_CHECK_INTERVAL</code>
<code>haproxy.router.openshift.io/ip_whitelist</code>	경로에 대한 화이트리스트를 설정합니다. 화이트리스트는 승인된 소스 주소에 대한 IP 주소 및 CIDR 범위가 공백으로 구분된 목록입니다. 화이트리스트에 없는 IP 주소의 요청은 삭제됩니다.  화이트리스트에 허용되는 최대 IP 주소 및 CIDR 범위 수는 61개입니다.	
<code>haproxy.router.openshift.io/https_header</code>	엣지 종단 경로 또는 재암호화 경로에 Strict-Transport-Security 헤더를 설정합니다.	
<code>haproxy.router.openshift.io/log-send-hostname</code>	Syslog 헤더에 <b>hostname</b> 필드를 설정합니다. 시스템의 호스트 이름을 사용합니다. 라우터에 대해 사이드카 또는 Syslog 기능과 같은 Ingress API 로깅 방법이 활성화된 경우 기본적으로 <b>log-send-hostname</b> 이 사용됩니다.	
<code>haproxy.router.openshift.io/rewrite-target</code>	백엔드의 요청 재작성 경로를 설정합니다.	

변수	설명	기본값으로 사용되는 환경 변수
<b>router.openshift.io/cookie-same-site</b>	<p>쿠키를 제한하는 값을 설정합니다. 값은 다음과 같습니다.</p> <p><b>Lax:</b> 방문한 사이트와 타사 사이트 간에 쿠키가 전송됩니다.</p> <p><b>Strict:</b> 쿠키가 방문한 사이트로 제한됩니다.</p> <p><b>None:</b> 쿠키가 방문한 사이트로 제한됩니다.</p> <p>이 값은 재암호화 및 엣지 경로에만 적용됩니다. 자세한 내용은 <a href="#">SameSite 쿠키 설명서</a>를 참조하십시오.</p>	
<b>haproxy.router.openshift.io/set-forwarded-headers</b>	<p>라우터당 <b>Forwarded</b> 및 <b>X-Forwarded-For</b> HTTP 헤더를 처리하기 위한 정책을 설정합니다. 값은 다음과 같습니다.</p> <p><b>append:</b> 기존 헤더를 유지하면서 헤더를 추가합니다. 이는 기본값입니다.</p> <p><b>replace:</b> 헤더를 설정하고 기존 헤더를 제거합니다.</p> <p><b>never:</b> 헤더를 설정하지 않고 기존 헤더를 유지합니다.</p> <p><b>if-none:</b> 아직 설정되지 않은 경우 헤더를 설정합니다.</p>	<b>ROUTER_SET_FORWARDED_HEADERS</b>



**참고**

환경 변수는 편집할 수 없습니다.

**라우터 시간 제한 변수**

**TimeUnits**는 다음과 같이 표시됩니다. **us** \*(마이크로초), **ms** (밀리초, 기본값), **s** (초), **m** (분), **h** \*(시간), **d** (일).

정규 표현식은 **[1-9][0-9]\*(us|ms|s|m|h|d)**입니다.

변수	기본	설명
<b>ROUTER_BACKEND_CHECK_INTERVAL</b>	<b>5000ms</b>	백엔드에서 후속 활성 검사 사이의 시간입니다.
<b>ROUTER_CLIENT_FIN_TIMEOUT</b>	<b>1s</b>	경로에 연결된 클라이언트의 TCP FIN 시간 제한 기간을 제어합니다. FIN이 연결을 닫도록 전송한 경우 지정된 시간 내에 응답하지 않으면 HAProxy가 연결을 종료합니다. 낮은 값으로 설정하면 문제가 없으며 라우터에서 더 적은 리소스를 사용합니다.
<b>ROUTER_DEFAULT_CLIENT_TIMEOUT</b>	<b>30s</b>	클라이언트가 데이터를 승인하거나 보내야 하는 시간입니다.
<b>ROUTER_DEFAULT_CONNECT_TIMEOUT</b>	<b>5s</b>	최대 연결 시간입니다.
<b>ROUTER_DEFAULT_SERVER_FIN_TIMEOUT</b>	<b>1s</b>	라우터에서 경로를 지원하는 pod로의 TCP FIN 시간 초과를 제어합니다.
<b>ROUTER_DEFAULT_SERVER_TIMEOUT</b>	<b>30s</b>	서버에서 데이터를 승인하거나 보내야 하는 시간입니다.
<b>ROUTER_DEFAULT_TUNNEL_TIMEOUT</b>	<b>1h</b>	TCP 또는 WebSocket 연결이 열린 상태로 유지되는 동안의 시간입니다. 이 시간 제한 기간은 HAProxy를 다시 로드할 때마다 재설정됩니다.
<b>ROUTER_SLOWLORIS_HTTP_KEEPPALIVE</b>	<b>300s</b>	<p>새 HTTP 요청이 표시될 때까지 대기할 최대 시간을 설정합니다. 이 값을 너무 낮게 설정하면 작은 <b>keepalive</b> 값을 예상하지 못하는 브라우저 및 애플리케이션에 문제가 발생할 수 있습니다.</p> <p>일부 유효한 시간 제한 값은 예상되는 특정 시간 초과가 아니라 특정 변수의 합계일 수 있습니다. 예를 들어 <b>ROUTER_SLOWLORIS_HTTP_KEEPPALIVE</b>는 <b>timeout http-keep-alive</b>를 조정합니다. 기본적으로 <b>300s</b>로 설정되지만 HAProxy는 <b>5s</b>로 설정된 <b>tcp-request inspect-delay</b>도 대기합니다. 이 경우 전체 시간 초과는 <b>300s + 5s</b>입니다.</p>
<b>ROUTER_SLOWLORIS_TIMEOUT</b>	<b>10s</b>	HTTP 요청 전송에 걸리는 시간입니다.
<b>RELOAD_INTERVAL</b>	<b>5s</b>	라우터의 최소 빈도가 새 변경 사항을 다시 로드하고 승인하도록 허용합니다.

변수	기본	설명
<b>ROUTER_METRICS_HAPROXY_TIMEOUT</b>	<b>5s</b>	HAProxy 메트릭 수집에 대한 시간 제한입니다.

경로 설정 사용자 정의 타임아웃

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms 1
...
    
```

**1**

HAProxy 지원 단위(us, ms, s, m, h, d)를 사용하여 새 타임아웃을 지정합니다. 단위가 제공되지 않는 경우 ms가 기본값입니다.



참고

패스스루(**passthrough**) 경로에 대한 서버 쪽 타임아웃 값을 너무 낮게 설정하면 해당 경로에서 **WebSocket** 연결이 자주 시간 초과될 수 있습니다.

하나의 특정 IP 주소만 허용하는 경로

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
    
```

여러 IP 주소를 허용하는 경로

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12

```

IP 주소 CIDR 네트워크를 허용하는 경로

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24

```

IP 주소 및 IP 주소 CIDR 네트워크를 둘 다 허용하는 경로

```

metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8

```

재작성 대상을 지정하는 경로

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / 1
  ...

```

**1**

/를 백엔드의 요청 재작성 경로로 설정합니다.

경로에 `haproxy.router.openshift.io/rewrite-target` 주석을 설정하면 **Ingress Controller**에서 요청을 백엔드 애플리케이션으로 전달하기 전에 이 경로를 사용하여 **HTTP** 요청의 경로를 재작성해야 합니다. `spec.path`에 지정된 경로와 일치하는 요청 경로 부분은 주석에 지정된 재작성 대상으로 교체됩니다.

다음 표에 `spec.path`, 요청 경로, 재작성 대상의 다양한 조합에 따른 경로 재작성 동작의 예가 있습니다.

표 23.3. 재작성 대상의 예:

Route.spec.path	요청 경로	재작성 대상	전달된 요청 경로
/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(요청 경로가 라우팅 경로와 일치하지 않음)
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

### 23.1.9. 경로 허용 정책 구성

관리자 및 애플리케이션 개발자는 도메인 이름이 동일한 여러 네임스페이스에서 애플리케이션을 실행할 수 있습니다. 이는 여러 팀이 동일한 호스트 이름에 노출되는 마이크로 서비스를 개발하는 조직을 위한 것입니다.





### 주의

네임스페이스 간 클레임은 네임스페이스 간 신뢰가 있는 클러스터에 대해서만 허용해야 합니다. 그렇지 않으면 악의적인 사용자가 호스트 이름을 인수할 수 있습니다. 따라서 기본 승인 정책에서는 네임스페이스 간에 호스트 이름 클레임을 허용하지 않습니다.

### 사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.

### 절차

- 다음 명령을 사용하여 `ingresscontroller` 리소스 변수의 `.spec.routeAdmission` 필드를 편집합니다.

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default --patch '{"spec": {"routeAdmission":{"namespaceOwnership":"InterNamespaceAllowed"}}}' --type=merge
```

### 샘플 Ingress 컨트롤러 구성

```
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
  ...
```

## 작은 정보

다음 **YAML**을 적용하여 경로 승인 정책을 구성할 수 있습니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  routeAdmission:
    namespaceOwnership: InterNamespaceAllowed
```

## 23.1.10. Ingress 오브젝트를 통해 경로 생성

일부 에코시스템 구성 요소는 **Ingress** 리소스와 통합되지만 경로 리소스와는 통합되지 않습니다. 이러한 경우를 처리하기 위해 **OpenShift Container Platform**에서는 **Ingress** 오브젝트가 생성될 때 관리형 경로 오브젝트를 자동으로 생성합니다. 이러한 경로 오브젝트는 해당 **Ingress** 오브젝트가 삭제될 때 삭제됩니다.

## 절차

1.

**OpenShift Container Platform** 콘솔에서 또는 **oc create** 명령을 입력하여 **Ingress** 오브젝트를 정의합니다.

**Ingress**의 **YAML** 정의

```
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

Ingress에는 Route에 대한 필드가 없으므로 `route.openshift.io/termination` 주석을 사용하여 `spec.tls.termination` 필드를 구성할 수 있습니다. 허용되는 값은 `edge`, `passthrough`, `reencrypt`입니다. 다른 모든 값은 자동으로 무시됩니다. 주석 값이 설정되지 않으면 `edge` 는 기본 경로입니다. 기본 엣지 경로를 구현하려면 템플릿 파일에 TLS 인증서 세부 정보를 정의해야 합니다.

3

Ingress 오브젝트로 작업할 때는 경로를 사용할 때와 달리 명시적 호스트 이름을 지정해야 합니다. `<host_name>.<cluster_ingress_domain >` 구문(예: `apps.openshift demos.com`) 구문을 사용하여 `*.<cluster_ingress_domain >` 와일드카드 DNS 레코드와 클러스터의 제공 인증서를 활용할 수 있습니다. 그렇지 않으면 선택한 호스트 이름에 대한 DNS 레코드가 있는지 확인해야 합니다.

a.

`route.openshift.io/termination` 주석에 `passthrough` 값을 지정하는 경우 `path`를 `"`로 설정하고 `spec`에서 `pathType`을 `ImplementationSpecific`으로 설정합니다.

```

spec:
rules:
- host: www.example.com
http:
paths:
- path: "
pathType: ImplementationSpecific
backend:
service:
name: frontend
port:
number: 443

```

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

2

Ingress 오브젝트에서 `route.openshift.io/destination-ca-certificate-secret` 을 사용하여 사용자 정의 대상 인증서(CA)로 경로를 정의할 수 있습니다. 주석은 생성된 경로에 삽입할 `kubernetes` 시크릿 `secret-ca-cert` 를 참조합니다.

- a. **Ingress** 오브젝트에서 대상 **CA**를 사용하여 경로 오브젝트를 지정하려면 시크릿의 **data.tls.crt** 구성 요소에 **PEM** 인코딩 형식의 인증서가 있는 **kubernetes.io/tls** 또는 **Opaque** 유형 시크릿을 생성해야 합니다.

2.

노드를 나열합니다.

```
$ oc get routes
```

결과에는 이름이 **frontend**-로 시작하는 자동 생성 경로가 포함됩니다.

NAME	HOST/PORT	PATH	SERVICES	PORT	TERMINATION
WILDCARD					
frontend-gnztq	www.example.com		frontend	443	reencrypt/Redirect None

이 경로를 살펴보면 다음과 같습니다.

자동 생성 경로의 **YAML** 정의

```
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

```

### 23.1.11. Ingress 오브젝트를 통해 기본 인증서를 사용하여 경로 생성

TLS 구성을 지정하지 않고 Ingress 오브젝트를 생성하면 OpenShift Container Platform에서 비보안 경로를 생성합니다. 기본 수신 인증서를 사용하여 보안 엣지 종료 경로를 생성하는 Ingress 오브젝트를 생성하려면 다음과 같이 빈 TLS 구성을 지정할 수 있습니다.

#### 사전 요구 사항

- 노출하려는 서비스가 있습니다.
- OpenShift CLI(oc)에 액세스할 수 있습니다.

#### 프로세스

1. Ingress 오브젝트에 대한 YAML 파일을 생성합니다. 이 예제에서는 파일을 `example-ingress.yaml` 이라고 합니다.

#### Ingress 오브젝트의 YAML 정의

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  ...
spec:
  rules:
    ...
  tls:
  - {} 1

```

**1**

이 정확한 구문을 사용하여 사용자 정의 인증서를 지정하지 않고 **TLS**를 지정합니다.

2.

다음 명령을 실행하여 **Ingress** 오브젝트를 생성합니다.

```
$ oc create -f example-ingress.yaml
```

검증

•

다음 명령을 실행하여 **OpenShift Container Platform**에서 **Ingress** 오브젝트에 대한 예상 경로를 생성했는지 확인합니다.

```
$ oc get routes -o yaml
```

출력 예

```
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend-j9sdd 1
    ...
  spec:
    ...
    tls: 2
      insecureEdgeTerminationPolicy: Redirect
      termination: edge 3
    ...
```

**1**

경로 이름에는 **Ingress** 오브젝트의 이름 뒤에 임의의 접미사가 포함됩니다.

**2**

기본 인증서를 사용하려면 경로에서 **spec.certificate** 를 지정하지 않아야 합니다.

3

경로는 옛지 종료 정책을 지정해야 합니다.

### 23.1.12. Ingress 주석의 대상 CA 인증서를 사용하여 경로 생성

**Ingress** 오브젝트에서 **route.openshift.io/destination-ca-certificate-secret** 주석을 사용하여 사용자 정의 대상 CA 인증서로 경로를 정의할 수 있습니다.

#### 사전 요구 사항

- PEM 인코딩 파일에 인증서/키 쌍이 있을 수 있으며 이 파일은 인증서가 경로 호스트에 유효합니다.
- 인증서 체인을 완성하는 PEM 인코딩 파일에 별도의 CA 인증서가 있을 수 있습니다.
- PEM 인코딩 파일에 별도의 대상 CA 인증서가 있어야 합니다.
- 노출하려는 서비스가 있어야 합니다.

#### 프로세스

1.

**Ingress** 주석에 **route.openshift.io/destination-ca-certificate-secret** 을 추가합니다.

```
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

주석은 **kubernetes** 보안을 참조합니다.

2. 이 주석에서 참조하는 보안이 생성된 경로에 삽입됩니다.

출력 예

```

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-----
  ...
  
```

### 23.1.13. 듀얼 스택 네트워킹을 위한 OpenShift Container Platform Ingress 컨트롤러 구성

**OpenShift Container Platform** 클러스터가 IPv4 및 IPv6 이중 스택 네트워킹에 맞게 구성된 경우 **OpenShift Container Platform** 경로에서 외부에서 클러스터에 연결할 수 있습니다.

**Ingress** 컨트롤러는 IPv4 및 IPv6 엔드 포인트가 모두 있는 서비스를 자동으로 제공하지만 단일 스택 또는 듀얼 스택 서비스에 대해 **Ingress** 컨트롤러를 구성할 수 있습니다.

사전 요구 사항

- 베어메탈에 **OpenShift Container Platform** 클러스터를 배포했습니다.
- **OpenShift CLI(oc)**를 설치합니다.

프로세스



1.

**Ingress** 컨트롤러가 워크로드로 IPv4/IPv6을 통해 트래픽을 제공하도록 하려면 **ipFamilies** 및 **ipFamilyPolicy** 필드를 설정하여 서비스 **YAML** 파일을 생성하거나 기존 서비스 **YAML** 파일을 수정할 수 있습니다. 예를 들면 다음과 같습니다.

샘플 서비스 **YAML** 파일

```

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

듀얼 스택 인스턴스에는 두 개의 서로 다른 **clusterIPs**가 제공됩니다.

②

3

단일 스택 인스턴스의 경우 **SingleStack**을 입력합니다. 듀얼 스택 인스턴스의 경우 **RequireDualStack**을 입력합니다.

이러한 리소스는 해당 **endpoints**를 생성합니다. **Ingress** 컨트롤러는 이제 **endpointslices**를 감시합니다.

- 2. **endpoints**를 확인하려면 다음 명령을 입력합니다:

```
$ oc get endpoints
```

- 3. **endpointslices**를 확인하려면 다음 명령을 입력합니다.

```
$ oc get endpointslices
```

추가 리소스

- **appsDomain** 옵션을 사용하여 대체 클러스터 도메인 지정

### 23.2. 보안 경로

보안 경로는 여러 유형의 **TLS** 종료를 사용하여 클라이언트에 인증서를 제공하는 기능을 제공합니다. 다음 섹션에서는 사용자 정의 인증서를 사용하여 재암호화 예지 및 패스스루 경로를 생성하는 방법을 설명합니다.



#### 중요

공용 끝점을 통해 **Microsoft Azure**에서 경로를 생성하는 경우 리소스 이름에 제한이 적용됩니다. 특정 용어를 사용하는 리소스를 생성할 수 없습니다. **Azure**에서 제한하는 용어 목록은 **Azure** 설명서의 **예약된 리소스 이름 오류 해결**을 참조하십시오.

#### 23.2.1. 사용자 정의 인증서를 사용하여 재암호화 경로 생성

**oc create route** 명령을 사용하면 재암호화 **TLS** 종료와 사용자 정의 인증서로 보안 경로를 구성할 수 있습니다.

## 사전 요구 사항

- **PEM 인코딩 파일에 인증서/키 쌍이 있고 해당 인증서가 경로 호스트에 유효해야 합니다.**
- 인증서 체인을 완성하는 **PEM 인코딩 파일에 별도의 CA 인증서가 있을 수 있습니다.**
- **PEM 인코딩 파일에 별도의 대상 CA 인증서가 있어야 합니다.**
- **노출하려는 서비스가 있어야 합니다.**

## 참고

암호로 보호되는 키 파일은 지원되지 않습니다. 키 파일에서 암호를 제거하려면 다음 명령을 사용하십시오.

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

## 프로세스

이 절차에서는 사용자 정의 인증서를 사용하여 **Route** 리소스를 생성하고 **TLS** 종료를 재암호화합니다. 다음 예에서는 인증서/키 쌍이 현재 작업 디렉터리의 **tls.crt** 및 **tls.key** 파일에 있다고 가정합니다. **Ingress** 컨트롤러에서 서비스의 인증서를 신뢰하도록 하려면 대상 **CA** 인증서도 지정해야 합니다. 인증서 체인을 완료하는 데 필요한 경우 **CA** 인증서를 지정할 수도 있습니다. **tls.crt**, **tls.key**, **cacert.crt**, **ca.crt**(옵션)에 실제 경로 이름을 사용하십시오. **frontend**에는 노출하려는 서비스 리소스 이름을 사용합니다. **www.example.com**을 적절한 호스트 이름으로 바꿉니다.

- **재암호화 TLS 종료 및 사용자 정의 인증서를 사용하여 보안 Route 리소스를 생성합니다.**

```
$ 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
```

생성된 **Route** 리소스는 다음과 유사합니다.

## 보안 경로의 YAML 정의

```
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-----

```

자세한 옵션은 `oc create route reencrypt --help`를 참조하십시오.

### 23.2.2. 사용자 정의 인증서를 사용하여 엣지 경로 생성

`oc create route` 명령을 사용하면 엣지 TLS 종료와 사용자 정의 인증서로 보안 경로를 구성할 수 있습니다. 엣지 경로를 사용하면 Ingress 컨트롤러에서 트래픽을 대상 Pod로 전달하기 전에 TLS 암호화를 종료합니다. 이 경로는 Ingress 컨트롤러가 경로에 사용하는 TLS 인증서 및 키를 지정합니다.

#### 사전 요구 사항

- PEM 인코딩 파일에 인증서/키 쌍이 있고 해당 인증서가 경로 호스트에 유효해야 합니다.
- 인증서 체인을 완성하는 PEM 인코딩 파일에 별도의 CA 인증서가 있을 수 있습니다.
- 노출하려는 서비스가 있어야 합니다.



## 참고

암호로 보호되는 키 파일은 지원되지 않습니다. 키 파일에서 암호를 제거하려면 다음 명령을 사용하십시오.

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

## 프로세스

이 절차에서는 사용자 정의 인증서 및 엣지 TLS 종료를 사용하여 Route 리소스를 생성합니다. 다음 예에서는 인증서/키 쌍이 현재 작업 디렉터리의 `tls.crt` 및 `tls.key` 파일에 있다고 가정합니다. 인증서 체인을 완료하는 데 필요한 경우 CA 인증서를 지정할 수도 있습니다. `tls.crt`, `tls.key`, `ca.crt`(옵션)에 실제 경로 이름을 사용하십시오. `frontend`에는 노출하려는 서비스 이름을 사용합니다. `www.example.com`을 적절한 호스트 이름으로 바꿉니다.

- 엣지 TLS 종료 및 사용자 정의 인증서를 사용하여 보안 Route 리소스를 생성합니다.

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=www.example.com
```

생성된 Route 리소스는 다음과 유사합니다.

## 보안 경로의 YAML 정의

```
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-----
```

추가 옵션은 `oc create route edge --help`를 참조하십시오.

### 23.2.3. 패스스루 라우팅 생성

`oc create route` 명령을 사용하면 패스스루 종료와 사용자 정의 인증서로 보안 경로를 구성할 수 있습니다. 패스스루 종료를 사용하면 암호화된 트래픽이 라우터에서 TLS 종료를 제공하지 않고 바로 대상으로 전송됩니다. 따라서 라우터에 키 또는 인증서가 필요하지 않습니다.

#### 사전 요구 사항

- 노출하려는 서비스가 있어야 합니다.

#### 프로세스

- **Route** 리소스를 생성합니다.

```
$ oc create route passthrough route-passthrough-secured --service=frontend --port=8080
```

생성된 **Route** 리소스는 다음과 유사합니다.

#### 패스스루 종료를 사용하는 보안 경로

```
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

63자로 제한되는 개체의 이름입니다.

2

*termination* 필드는 *passthrough*로 설정됩니다. 이 필드는 유일한 필수 *tls* 필드입니다.

3

*insecureEdgeTerminationPolicy*는 선택 사항입니다. 비활성화경우 유효한 값은 *None*, *Redirect* 또는 빈 값입니다.

대상 *Pod*는 끝점의 트래픽에 대한 인증서를 제공해야 합니다. 현재 이 방법은 양방향 인증이라고도 하는 클라이언트 인증서도 지원할 수 있는 유일한 방법입니다.

## 24장. 수신 클러스터 트래픽 구성

### 24.1. 수신 클러스터 트래픽 구성 개요

OpenShift Container Platform에서는 다음 방법을 통해 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다.

순서 또는 기본 설정에 따라 권장되는 방법입니다.

- HTTP/HTTPS가 있는 경우 **Ingress** 컨트롤러를 사용합니다.
- HTTPS 이외의 TLS 암호화 프로토콜이 있는 경우(예: SNI 헤더가 있는 TLS), **Ingress** 컨트롤러를 사용합니다.
- 그 외에는 로드 밸런서, 외부 IP 또는 **NodePort**를 사용합니다.

방법	목적
<a href="#">Ingress 컨트롤러 사용</a>	HTTPS 이외의 HTTP/HTTPS 트래픽 및 TLS 암호화 프로토콜(예: SNI 헤더가 있는 TLS)에 액세스할 수 있습니다.
<a href="#">로드 밸런서 서비스를 사용하여 외부 IP 자동 할당</a>	풀에서 할당된 IP 주소를 통해 비표준 포트로의 트래픽을 허용합니다. 대부분의 클라우드 플랫폼은 로드 밸런서 IP 주소로 서비스를 시작하는 방법을 제공합니다.
<a href="#">MetalLB 및 MetalLB Operator 정보</a>	시스템 네트워크의 풀에서 특정 IP 주소 또는 주소로의 트래픽을 허용합니다. 베어 메탈과 같은 베어 메탈 설치 또는 플랫폼의 경우 MetalLB는 로드 밸런서 IP 주소로 서비스를 시작하는 방법을 제공합니다.
<a href="#">서비스에 외부 IP를 수동으로 할당</a>	특정 IP 주소를 통해 비표준 포트로의 트래픽을 허용합니다.
<a href="#">NodePort 구성</a>	클러스터의 모든 노드에 서비스를 공개합니다.

#### 24.1.1. 비교: 외부 IP 주소에 대한 내결함성 액세스

외부 IP 주소에 대한 액세스를 제공하는 통신 방법의 경우 IP 주소에 대한 내결함성 액세스를 고려해야 합니다. 다음 기능은 외부 IP 주소에 대한 내결함성 액세스를 제공합니다.



## IP 페일오버

IP 페일오버는 노드 집합의 가상 IP 주소 풀을 관리합니다. **Keepalived** 및 **VRRP(Virtual Router Redundancy Protocol)**로 구현됩니다. IP 페일오버는 계층 2 메커니즘일 뿐이며 멀티캐스트를 사용하지 않습니다. 멀티캐스트에는 일부 네트워크에 대한 단점이 있을 수 있습니다.

## MetalLB

**MetalLB**에는 계층 2 모드가 있지만 멀티캐스트를 사용하지 않습니다. 계층 2 모드에는 하나의 노드를 통해 외부 IP 주소에 대한 모든 트래픽을 전송하는 단점이 있습니다.

### 수동으로 외부 IP 주소 할당

외부 IP 주소를 서비스에 할당하는 데 사용되는 IP 주소 블록을 사용하여 클러스터를 구성할 수 있습니다. 이 기능은 기본적으로 비활성화되어 있습니다. 이 기능은 유연하지만 클러스터 또는 네트워크 관리자에게 가장 큰 부담이 됩니다. 클러스터는 외부 IP로 향하는 트래픽을 수신할 준비가 되지만 각 고객은 트래픽을 노드로 라우팅하는 방법을 결정해야 합니다.

## 24.2. 서비스의 EXTERNALIP 구성

클러스터 관리자는 클러스터의 서비스로 트래픽을 보낼 수 있는 클러스터 외부의 IP 주소 블록을 지정할 수 있습니다.

이 기능은 일반적으로 베어 메탈 하드웨어에 설치된 클러스터에 가장 유용합니다.

### 24.2.1. 사전 요구 사항

- 네트워크 인프라는 외부 IP 주소에 대한 트래픽을 클러스터로 라우팅해야 합니다.

### 24.2.2. ExternalIP 정보

클라우드 환경이 아닌 경우 **OpenShift Container Platform**에서는 **ExternalIP** 기능을 통해 **Service** 오브젝트 `spec.externalIPs[]` 필드에 외부 IP 주소 할당을 지원합니다. 이 필드를 설정하면 **OpenShift Container Platform**에서 추가 가상 IP 주소를 서비스에 할당합니다. IP 주소는 클러스터에 정의된 서비스 네트워크 외부에 있을 수 있습니다. **ExternalIP** 함수로 구성된 서비스는 `type=NodePort`인 서비스와 유사하게 작동하므로 부하 분산을 위해 트래픽을 로컬 노드로 보낼 수 있습니다.

정의한 외부 IP 주소 블록이 클러스터로 라우팅되도록 네트워킹 인프라를 구성해야 합니다.

OpenShift Container Platform은 다음 기능을 추가하여 Kubernetes의 ExternalIP 기능을 확장합니다.

- 구성 가능한 정책을 통해 사용자가 외부 IP 주소 사용 제한
- 요청 시 서비스에 자동으로 외부 IP 주소 할당



주의

ExternalIP 기능은 기본적으로 비활성화되어 있으며, 사용 시 외부 IP 주소에 대한 클러스터 내 트래픽이 해당 서비스로 전달되기 때문에 보안 위협이 발생할 수 있습니다. 이 경우 클러스터 사용자가 외부 리소스로 향하는 민감한 트래픽을 가로챌 수 있습니다.



중요

이 기능은 클라우드 배포가 아닌 경우에만 지원됩니다. 클라우드 배포의 경우 클라우드 로드 밸런서 자동 배포를 위한 로드 밸런서 서비스를 사용하여 서비스 끝점을 대상으로 합니다.

다음과 같은 방법으로 외부 IP 주소를 할당할 수 있습니다.

외부 IP 자동 할당

OpenShift Container Platform에서는 spec.type=LoadBalancer가 설정된Service 오브젝트를 생성할 때 autoAssignCIDRs CIDR 블록의 IP 주소를 spec.externalIPs[] 배열에 자동으로 할당합니다. 이 경우 OpenShift Container Platform은 로드 밸런서 서비스 유형의 비클라우드 버전을 구현하고 서비스에 IP 주소를 할당합니다. 자동 할당은 기본적으로 비활성화되어 있으며 다음 섹션에 설명된 대로 클러스터 관리자가 구성해야 합니다.

외부 IP 수동 할당

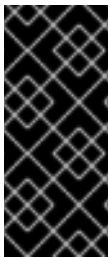
OpenShift Container Platform에서는 Service 오브젝트를 생성할 때 spec.externalIPs[] 배열에 할당된 IP 주소를 사용합니다. 다른 서비스에서 이미 사용 중인 IP 주소는 지정할 수 없습니다.

24.2.2.1. ExternalIP 구성

OpenShift Container Platform에 대한 외부 IP 주소 사용은 `cluster`라는 `Network.config.openshift.io` CR에 있는 다음 필드로 관리합니다.

- `spec.externalIP.autoAssignCIDRs`는 서비스에 대한 외부 IP 주소를 선택할 때 로드 밸런서에서 사용하는 IP 주소 블록을 정의합니다. OpenShift Container Platform에서는 자동 할당에 대해 하나의 IP 주소 블록만 지원합니다. 이렇게 하면 서비스에 ExternalIP를 수동으로 할당 때 제한된 수의 공유 IP 주소로 구성된 포트 공간을 관리하는 것보다 더 간단할 수 있습니다. 자동 할당을 사용하는 경우 `spec.type=LoadBalancer`인 Service에 외부 IP 주소가 할당됩니다.
- `spec.externalIP.policy`는 IP 주소를 수동으로 지정할 때 허용되는 IP 주소 블록을 정의합니다. OpenShift Container Platform은 `spec.externalIP.autoAssignCIDRs`로 정의된 IP 주소 블록에 정책 규칙을 적용하지 않습니다.

올바르게 라우팅되면 구성된 외부 IP 주소 블록의 외부 트래픽이 서비스에서 노출하는 TCP 또는 UDP 포트를 통해 서비스 끝점에 도달할 수 있습니다.



#### 중요

클러스터 관리자는 OpenShiftSDN 및 OVN-Kubernetes 네트워크 유형 모두에서 `externalIPs`로 라우팅을 구성해야 합니다. 또한 할당하는 IP 주소 블록이 클러스터의 하나 이상의 노드에서 종료되어야 합니다. 자세한 내용은 [Kubernetes 외부 IP](#)를 참조하십시오.

OpenShift Container Platform에서는 IP 주소의 자동 및 수동 할당을 모두 지원하며 각 주소는 최대 하나의 서비스에 할당됩니다. 따라서 각 서비스는 다른 서비스에서 노출하는 포트와 관계없이 선택한 포트를 노출할 수 있습니다.



#### 참고

OpenShift Container Platform에서 `autoAssignCIDR`로 정의된 IP 주소 블록을 사용하려면 호스트 네트워크에 필요한 IP 주소 할당 및 라우팅을 구성해야 합니다.

다음 YAML에서는 외부 IP 주소가 구성된 서비스를 설명합니다.

`spec.externalIPs[]`가 설정된 Service 오브젝트의 예

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

```

#### 24.2.2.2. 외부 IP 주소 할당 제한 사항

클러스터 관리자는 허용 및 거부할 IP 주소 블록을 지정할 수 있습니다.

제한 사항은 **cluster-admin** 권한이 없는 사용자에게만 적용됩니다. 클러스터 관리자는 서비스 `spec.externalIPs[]` 필드를 IP 주소로 항상 설정할 수 있습니다.

`spec.ExternalIP.policy` 필드를 지정하여 정의된 **policy** 오브젝트를 사용하여 IP 주소 정책을 구성합니다. 정책 오브젝트의 형태는 다음과 같습니다.

```

{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}

```

정책 제한을 구성할 때는 다음 규칙이 적용됩니다.

-

`policy={}`가 설정된 경우 `spec.ExternalIPs[]`가 설정된 `Service` 오브젝트를 생성할 수 없습니다. 이는 `OpenShift Container Platform`의 기본값입니다. `policy=null`을 설정할 때의 동작은 동일합니다.

- `policy`가 설정되고 `policy.allowedCIDRs[]` 또는 `policy.rejectedCIDRs[]`가 설정된 경우 다음 규칙이 적용됩니다.
  - `allowedCIDRs[]` 및 `rejectedCIDRs[]`가 둘 다 설정된 경우 `rejectedCIDRs[]`가 `allowedCIDRs[]`보다 우선합니다.
  - `allowedCIDRs[]`가 설정된 경우 지정된 IP 주소가 허용되는 경우에만 `spec.ExternalIPs[]`를 사용하여 `Service`를 생성할 수 있습니다.
  - `rejectedCIDRs[]`가 설정된 경우 지정된 IP 주소가 거부되지 않는 경우에만 `spec.ExternalIPs[]`를 사용하여 `Service`를 생성할 수 있습니다.

#### 24.2.2.3. 정책 오브젝트의 예

다음 예제에서는 다양한 정책 구성을 보여줍니다.

- 다음 예에서 정책은 `OpenShift Container Platform`에서 외부 IP 주소가 지정된 서비스를 생성하지 못하도록 합니다.

`Service` 오브젝트 `spec.externalIPs[]`에 지정된 값을 거부하는 정책의 예

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

- 다음 예에서는 **allowedCIDRs** 및 **rejectedCIDRs** 필드가 모두 설정되어 있습니다.

허용되거나 거부된 **CIDR** 블록을 모두 포함하는 정책의 예

```

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
  ...
    
```

- 다음 예에서는 **policy**가 **null**로 설정됩니다. **null**로 설정하면 **oc get networks.config.openshift.io -o yaml**을 입력하여 구성 오브젝트를 검사할 때 **policy** 필드가 출력에 표시되지 않습니다.

**Service** 오브젝트 **spec.externalIPs[]**에 지정된 값을 허용하는 정책의 예

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: null
  ...
    
```

### 24.2.3. ExternalIP 주소 블록 구성

**ExternalIP** 주소 블록에 대한 구성은 **cluster**라는 네트워크 **CR**(사용자 정의 리소스)에 의해 정의됩니다. 네트워크 **CR**은 **config.openshift.io** API 그룹의 일부입니다.



중요

**CVO(Cluster Version Operator)**는 클러스터를 설치하는 동안 **cluster**라는 네트워크 **CR**을 자동으로 생성합니다. 이 유형의 다른 **CR** 오브젝트는 생성할 수 없습니다.

다음 **YAML**에서는 **ExternalIP** 구성을 설명합니다.

**cluster**라는 **Network.config.openshift.io CR**

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    autoAssignCIDRs: [] 1
    policy: 2
    ...
```

1

서비스에 대한 외부 **IP** 주소 자동 할당에 사용할 수 있는 **CIDR** 형식으로 **IP** 주소 블록을 정의합니다. 단일 **IP** 주소 범위만 허용됩니다.

2

서비스에 대한 **IP** 주소 수동 할당에 대한 제한을 정의합니다. 제한이 정의되지 않은 경우 **Service**에서 **spec.externalIP** 필드를 지정할 수 없습니다. 기본적으로는 제한이 정의되어 있지 않습니다.

다음 **YAML**에서는 **policy** 스탠자의 필드를 설명합니다.

**Network.config.openshift.io policy** 스탠자

```
policy:
  allowedCIDRs: [] 1
  rejectedCIDRs: [] 2
```

■

1

**CIDR 형식의 허용된 IP 주소 범위 목록입니다.**

2

**CIDR 형식의 거부된 IP 주소 범위 목록입니다.**

*외부 IP 구성의 예*

외부 IP 주소 풀에 사용 가능한 몇 가지 구성이 다음 예에 표시되어 있습니다.

- 다음 **YAML**에서는 자동으로 할당된 외부 IP 주소를 사용하는 구성을 설명합니다.

**spec.externalIP.autoAssignCIDRs**가 설정된 구성의 예

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    autoAssignCIDRs:
      - 192.168.132.254/29

```

- 다음 **YAML**에서는 허용되거나 거부된 **CIDR** 범위에 대한 정책 규칙을 구성합니다.

**spec.externalIP.policy**가 설정된 구성의 예

```

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
```

#### 24.2.4. 클러스터에 대한 외부 IP 주소 블록 구성

클러스터 관리자는 다음 **ExternalIP** 설정을 구성할 수 있습니다.

- **Service** 오브젝트의 **spec.clusterIP** 필드를 자동으로 채우도록 **OpenShift Container Platform**에서 사용하는 **ExternalIP** 주소 블록입니다.
- **Service** 오브젝트의 **spec.clusterIP** 배열에 수동으로 할당할 수 있는 **IP** 주소를 제한하는 정책 오브젝트입니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.

#### 프로세스

1. 선택 사항: 현재 외부 IP 구성을 표시하려면 다음 명령을 입력합니다.

```
$ oc describe networks.config cluster
```

2. 구성을 편집하려면 다음 명령을 입력합니다.

```
$ oc edit networks.config cluster
```

3. 다음 예와 같이 **ExternalIP** 구성을 수정합니다.

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: 1
  ...
    
```

1

**externalIP** 스텐자에 대한 구성을 지정합니다.

4. 업데이트된 **ExternalIP** 구성을 확인하려면 다음 명령을 입력합니다.

```

$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{\n}'
    
```

### 24.2.5. 다음 단계

- [서비스 외부 IP에 대한 수신 클러스터 트래픽 구성](#)

## 24.3. INGRESS 컨트롤러를 사용한 수신 클러스터 트래픽 구성

**OpenShift Container Platform**에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법에서는 **Ingress** 컨트롤러를 사용합니다.

### 24.3.1. Ingress 컨트롤러 및 경로 사용

**Ingress Operator**에서는 **Ingress** 컨트롤러 및 와일드카드 **DNS**를 관리합니다.

**OpenShift Container Platform** 클러스터에 대한 외부 액세스를 허용하는 가장 일반적인 방법은 **Ingress** 컨트롤러를 사용하는 것입니다.

**Ingress** 컨트롤러는 외부 요청을 수락하고 구성된 경로를 기반으로 이러한 요청을 프록시하도록 구성되어 있습니다. 이는 **HTTP**, **SNI**를 사용하는 **HTTPS**, **SNI**를 사용하는 **TLS**로 제한되며, **SNI**를 사용하는 **TLS**를 통해 작동하는 웹 애플리케이션 및 서비스에 충분합니다.

관리자와 협력하여 구성된 경로를 기반으로 외부 요청을 수락하고 프록시하도록 **Ingress** 컨트롤러를 구성하십시오.

관리자는 와일드카드 **DNS** 항목을 생성한 다음 **Ingress** 컨트롤러를 설정할 수 있습니다. 그러면 관리자에게 문의하지 않고도 옛지 **Ingress** 컨트롤러로 작업할 수 있습니다.

기본적으로 클러스터의 모든 **Ingress** 컨트롤러는 클러스터의 모든 프로젝트에서 생성된 모든 경로를 허용할 수 있습니다.

### **Ingress** 컨트롤러의 경우

- 기본적으로 두 개의 복제본이 있으므로 두 개의 작업자 노드에서 실행되어야 합니다.
- 더 많은 노드에 더 많은 복제본을 갖도록 확장할 수 있습니다.



#### 참고

이 섹션의 절차에는 클러스터 관리자가 수행해야 하는 사전 요구 사항이 필요합니다.

### 24.3.2. 사전 요구 사항

다음 절차를 시작하기 전에 관리자는 다음을 수행해야 합니다.

- 요청이 클러스터에 도달할 수 있도록 외부 포트를 클러스터 네트워킹 환경으로 설정합니다.
- 클러스터 관리자 역할의 사용자가 한 명 이상 있는지 확인합니다. 이 역할을 사용자에게 추가하려면 다음 명령을 실행합니다.

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- 클러스터에 대한 네트워크 액세스 권한이 있는 마스터와 노드가 클러스터 외부에 각각 1개 이상씩 있는 **OpenShift Container Platform** 클러스터가 있어야 합니다. 이 절차에서는 외부 시

스택이 클러스터와 동일한 서브넷에 있다고 가정합니다. 다른 서브넷에 있는 외부 시스템에 필요한 추가 네트워킹은 이 주제에서 다루지 않습니다.

### 24.3.3. 프로젝트 및 서비스 생성

노출하려는 프로젝트 및 서비스가 존재하지 않는 경우 먼저 프로젝트를 생성한 다음 서비스를 생성합니다.

프로젝트와 서비스가 이미 존재하는 경우에는 서비스 노출 절차로 건너뛰어 경로를 생성합니다.

#### 사전 요구 사항

- **oc CLI**를 설치하고 클러스터 관리자로 로그인합니다.

#### 프로세스

1. **oc new-project** 명령을 실행하여 서비스에 대한 새 프로젝트를 생성합니다.

```
$ oc new-project myproject
```

2. **oc new-app** 명령을 사용하여 서비스를 생성합니다.

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

3. 서비스가 생성되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get svc -n myproject
```

#### 출력 예

```
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
nodejs-ex ClusterIP  172.30.197.157 <none>       8080/TCP   70s
```

기본적으로 새 서비스에는 외부 IP 주소가 없습니다.

#### 24.3.4. 경로를 생성하여 서비스 노출

**oc expose** 명령을 사용하여 서비스를 경로로 노출할 수 있습니다.

##### 프로세스

서비스를 노출하려면 다음을 수행하십시오.

1. **OpenShift Container Platform 4**에 로그인합니다.
  2. 노출하려는 서비스가 있는 프로젝트에 로그인합니다.
- ```
$ oc project myproject
```
3. **oc expose service** 명령을 실행하여 경로를 노출합니다.

```
$ oc expose service nodejs-ex
```

출력 예

```
route.route.openshift.io/nodejs-ex exposed
```

4. 서비스가 노출되었는지 확인하려면 **cURL**과 같은 툴을 사용하여 클러스터 외부에서 서비스에 액세스할 수 있는지 확인할 수 있습니다.
  - a. **oc get route** 명령을 사용하여 경로의 호스트 이름을 찾습니다.

```
$ oc get route
```

출력 예

| NAME      | HOST/PORT                       | PATH | SERVICES  | PORT     | TERMINATION |
|-----------|---------------------------------|------|-----------|----------|-------------|
| WILDCARD  |                                 |      |           |          |             |
| nodejs-ex | nodejs-ex-myproject.example.com |      | nodejs-ex | 8080-tcp |             |
| None      |                                 |      |           |          |             |

b.

*cURL*을 사용하여 호스트가 **GET** 요청에 응답하는지 확인합니다.

```
$ curl --head nodejs-ex-myproject.example.com
```

출력 예

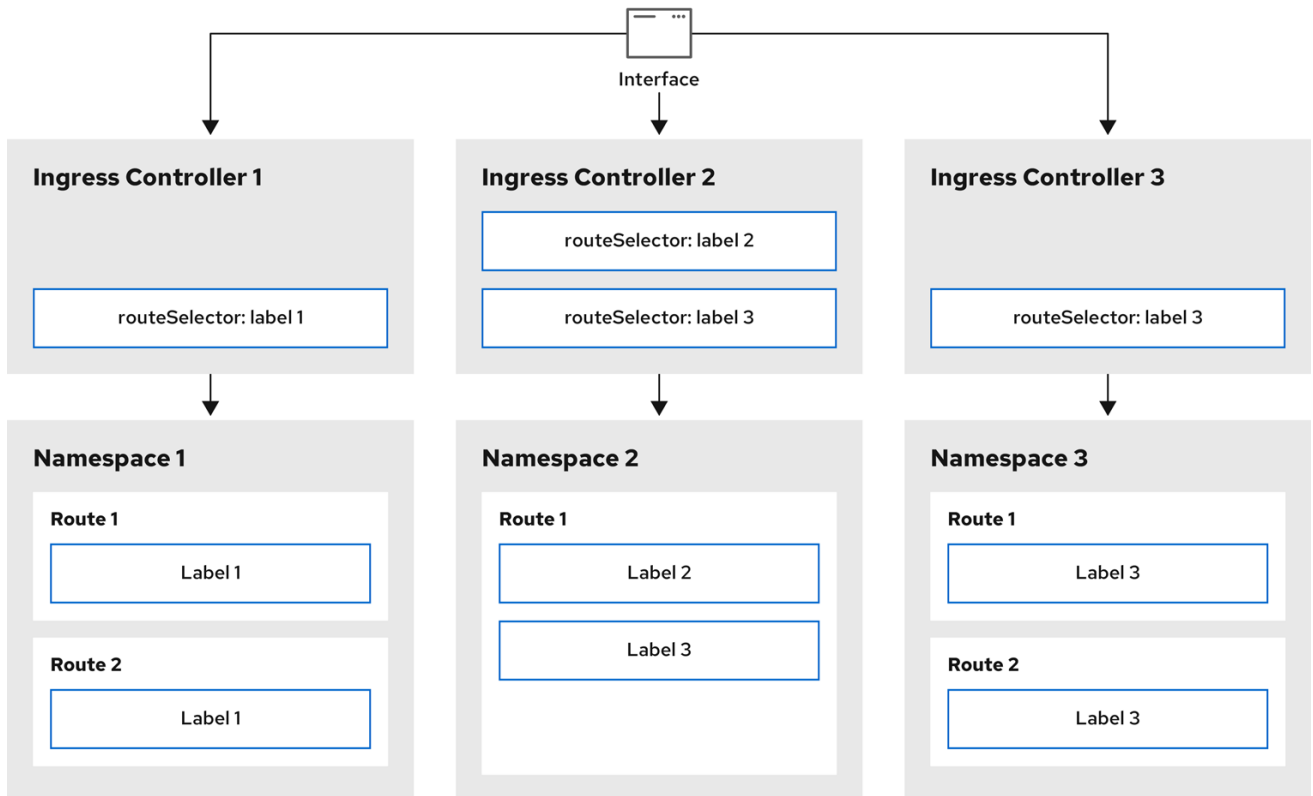
```
HTTP/1.1 200 OK
```

```
...
```

### 24.3.5. 경로 라벨을 사용하여 Ingress 컨트롤러 분할 구성

경로 라벨을 사용한 **Ingress** 컨트롤러 분할이란 **Ingress** 컨트롤러가 경로 선택기에서 선택한 모든 네임스페이스의 모든 경로를 제공한다는 뜻입니다.

그림 24.1. 경로 라벨을 사용한 Ingress 분할



301\_OpenShift\_0123

**Ingress** 컨트롤러 분할은 들어오는 트래픽 부하를 일련의 **Ingress** 컨트롤러에 균형 있게 분배하고 트래픽을 특정 **Ingress** 컨트롤러에 격리할 때 유용합니다. 예를 들어, 회사 A는 하나의 **Ingress** 컨트롤러로, 회사 B는 다른 **Ingress** 컨트롤러로 이동합니다.

### 절차

1.

`router-internal.yaml` 파일을 다음과 같이 편집합니다.

```
# cat router-internal.yaml
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    routeSelector:
      matchLabels:
```

```

type: sharded
status: {}
kind: List
metadata:
resourceVersion: ""
selfLink: ""
    
```

2.

**Ingress** 컨트롤러 `router-internal.yaml` 파일을 적용합니다.

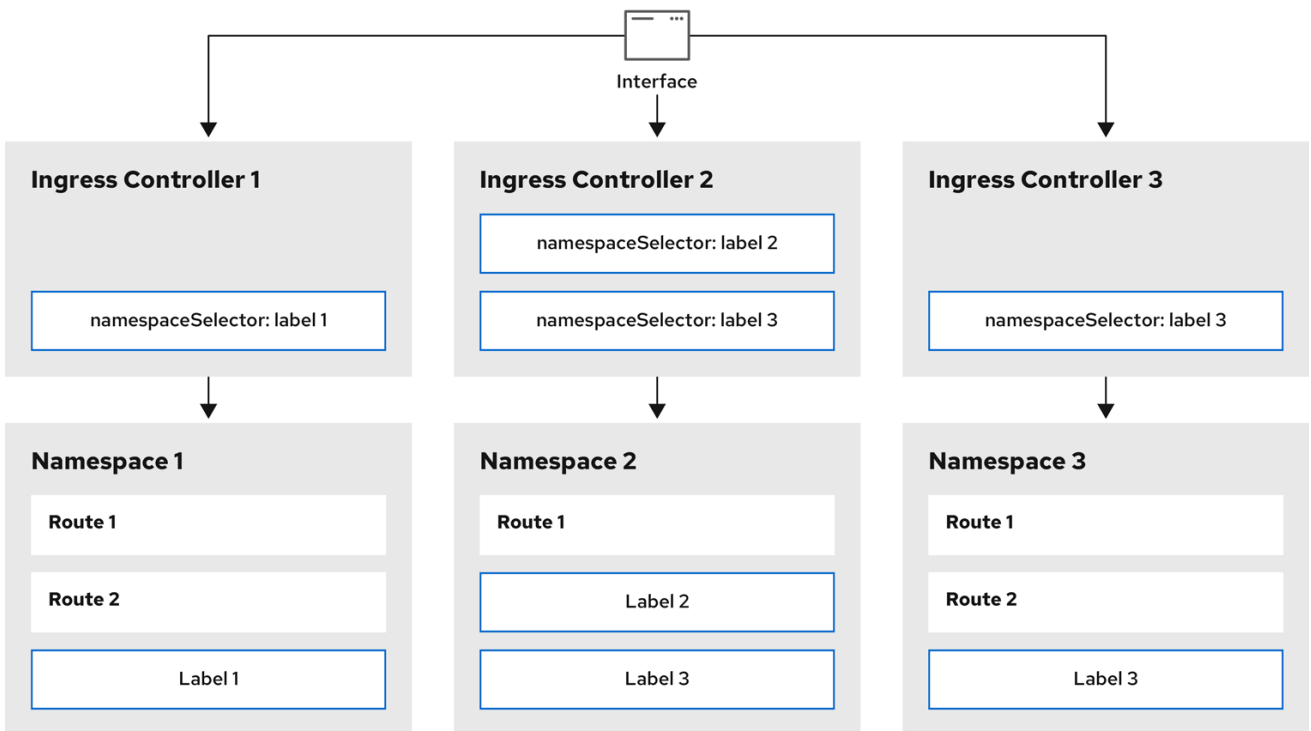
```
# oc apply -f router-internal.yaml
```

**Ingress** 컨트롤러는 `type: sharded` 라벨이 있는 네임스페이스에서 경로를 선택합니다.

### 24.3.6. 네임스페이스 라벨을 사용하여 Ingress 컨트롤러 분할 구성

네임스페이스 라벨을 사용한 **Ingress** 컨트롤러 분할이란 **Ingress** 컨트롤러가 네임스페이스 선택기에 서 선택한 모든 네임스페이스의 모든 경로를 제공한다는 뜻입니다.

그림 24.2. 네임스페이스 라벨을 사용한 Ingress 분할



301\_OpenShift\_0123

**Ingress** 컨트롤러 분할은 들어오는 트래픽 부하를 일련의 **Ingress** 컨트롤러에 균형 있게 분배하고 트래픽을 특정 **Ingress** 컨트롤러에 격리할 때 유용합니다. 예를 들어, 회사 A는 하나의 **Ingress** 컨트롤러로, 회사 B는 다른 **Ingress** 컨트롤러로 이동합니다.



## 프로세스

1. **router-internal.yaml** 파일을 다음과 같이 편집합니다.

```
# cat router-internal.yaml
```

출력 예

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: IngressController
  metadata:
    name: sharded
    namespace: openshift-ingress-operator
  spec:
    domain: <apps-sharded.basedomain.example.net>
    nodePlacement:
      nodeSelector:
        matchLabels:
          node-role.kubernetes.io/worker: ""
    namespaceSelector:
      matchLabels:
        type: sharded
  status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

2. **Ingress** 컨트롤러 **router-internal.yaml** 파일을 적용합니다.

```
# oc apply -f router-internal.yaml
```

**Ingress** 컨트롤러는 네임스페이스 선택기에서 선택한 **type: sharded** 라벨이 있는 네임스페이스에서 경로를 선택합니다.

### 24.3.7. Ingress 컨트롤러 분할을 위한 경로 생성

경로를 사용하면 **URL**에서 애플리케이션을 호스팅할 수 있습니다. 이 경우 호스트 이름이 설정되지 않고 경로는 대신 하위 도메인을 사용합니다. 하위 도메인을 지정하면 경로를 노출하는 **Ingress** 컨트롤러의

도메인을 자동으로 사용합니다. 여러 **Ingress** 컨트롤러에서 경로가 노출되는 경우 경로는 여러 **URL**에서 호스팅됩니다.

다음 절차에서는 **hello-openshift** 애플리케이션을 예제로 사용하여 **Ingress** 컨트롤러 분할에 대한 경로를 생성하는 방법을 설명합니다.

**Ingress** 컨트롤러 분할은 들어오는 트래픽 부하를 일련의 **Ingress** 컨트롤러에 균형 있게 분배하고 트래픽을 특정 **Ingress** 컨트롤러에 격리할 때 유용합니다. 예를 들어, 회사 **A**는 하나의 **Ingress** 컨트롤러로, 회사 **B**는 다른 **Ingress** 컨트롤러로 이동합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- 프로젝트 관리자로 로그인되어 있습니다.
- 포트에서 트래픽을 수신 대기하는 포트 및 **HTTP** 또는 **TLS** 엔드포인트를 노출하는 웹 애플리케이션이 있습니다.
- 분할을 위해 **Ingress** 컨트롤러가 구성되어 있습니다.

#### 프로세스

1. 다음 명령을 실행하여 **hello-openshift** 라는 프로젝트를 생성합니다.

```
$ oc new-project hello-openshift
```

2. 다음 명령을 실행하여 프로젝트에 **Pod**를 생성합니다.

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/hello-openshift/hello-pod.json
```

3. 다음 명령을 실행하여 **hello-openshift** 라는 서비스를 생성합니다.

```
$ oc expose pod/hello-openshift
```

4.

**hello-openshift-route.yaml** 이라는 경로 정의를 생성합니다.

분할을 위해 생성된 경로의 **YAML** 정의:

```
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**

레이블 키와 해당 레이블 값은 **Ingress** 컨트롤러에 지정된 라벨과 일치해야 합니다. 이 예제에서 **Ingress** 컨트롤러에는 레이블 키와 값 **type: sharded** 가 있습니다.

**2**

경로는 **subdomain** 필드의 값을 사용하여 노출됩니다. 하위 도메인 필드를 지정할 때 호스트 이름을 설정되지 않은 상태로 두어야 합니다. **host** 및 **subdomain** 필드를 모두 지정하면 경로는 호스트 필드의 값을 사용하고 **subdomain** 필드를 무시합니다.

5.

다음 명령을 실행하여 **hello-openshift-route.yaml** 을 사용하여 **hello-openshift** 애플리케이션에 대한 경로를 생성합니다.

```
$ oc -n hello-openshift create -f hello-openshift-route.yaml
```

검증

•

다음 명령을 사용하여 경로의 상태를 가져옵니다.

```
$ oc -n hello-openshift get routes/hello-openshift-edge -o yaml
```

생성된 **Route** 리소스는 다음과 유사해야 합니다.

출력 예

```
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

**Ingress** 컨트롤러 또는 라우터에서 경로를 노출하는 데 사용하는 호스트 이름입니다. **host** 필드의 값은 **Ingress** 컨트롤러에 의해 자동으로 결정되며 도메인을 사용합니다. 이 예제에서 **Ingress** 컨트롤러의 도메인은 `< apps-sharded.basedomain.example.net>`입니다.

2

**Ingress** 컨트롤러의 호스트 이름입니다.

3

**Ingress** 컨트롤러의 이름입니다. 이 예제에서 **Ingress** 컨트롤러에는 이름 `sharded` 가 있습니다.

### 24.3.8. 추가 리소스

- **Ingress Operator**는 와일드카드 **DNS**를 관리합니다. 자세한 내용은 [OpenShift Container Platform의 Ingress Operator](#), [베어 메탈에 클러스터 설치](#), [vSphere에 클러스터 설치를 참조하십시오](#).

## 24.4. 로드 밸런서를 사용하여 수신 클러스터 트래픽 구성

**OpenShift Container Platform**에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법에서는 로드 밸런서를 사용합니다.

### 24.4.1. 로드 밸런서를 사용하여 클러스터로 트래픽 가져오기

특정 외부 **IP** 주소가 필요하지 않은 경우 **OpenShift Container Platform** 클러스터에 대한 외부 액세스를 허용하도록 로드 밸런서 서비스를 구성할 수 있습니다.

로드 밸런서 서비스에서는 고유 **IP**를 할당합니다. 로드 밸런서에는 **VIP(가상 IP)**일 수 있는 단일 엣지 라우터 **IP**가 있지만 이는 초기 로드 밸런싱을 위한 단일 머신에 불과합니다.



참고

풀이 구성된 경우 클러스터 관리자가 아닌 인프라 수준에서 수행됩니다.



참고

이 섹션의 절차에는 클러스터 관리자가 수행해야 하는 사전 요구 사항이 필요합니다.

### 24.4.2. 사전 요구 사항

다음 절차를 시작하기 전에 관리자는 다음을 수행해야 합니다.

- 요청이 클러스터에 도달할 수 있도록 외부 포트를 클러스터 네트워크 환경으로 설정합니다.
- 클러스터 관리자 역할의 사용자가 한 명 이상 있는지 확인합니다. 이 역할을 사용자에게 추가하려면 다음 명령을 실행합니다.

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- 클러스터에 대한 네트워크 액세스 권한이 있는 마스터와 노드가 클러스터 외부에 각각 1개 이상씩 있는 **OpenShift Container Platform** 클러스터가 있어야 합니다. 이 절차에서는 외부 시스템이 클러스터와 동일한 서브넷에 있다고 가정합니다. 다른 서브넷에 있는 외부 시스템에 필요한 추가 네트워킹은 이 주제에서 다루지 않습니다.

### 24.4.3. 프로젝트 및 서비스 생성

노출하려는 프로젝트 및 서비스가 존재하지 않는 경우 먼저 프로젝트를 생성한 다음 서비스를 생성합니다.

프로젝트와 서비스가 이미 존재하는 경우에는 서비스 노출 절차로 건너뛰어 경로를 생성합니다.

#### 사전 요구 사항

- oc CLI**를 설치하고 클러스터 관리자로 로그인합니다.

#### 프로세스

- oc new-project** 명령을 실행하여 서비스에 대한 새 프로젝트를 생성합니다.

```
$ oc new-project myproject
```

- oc new-app** 명령을 사용하여 서비스를 생성합니다.

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

- 서비스가 생성되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get svc -n myproject
```

#### 출력 예

```
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
nodejs-ex ClusterIP  172.30.197.157 <none>       8080/TCP   70s
```

기본적으로 새 서비스에는 외부 IP 주소가 없습니다.

#### 24.4.4. 경로를 생성하여 서비스 노출

**oc expose** 명령을 사용하여 서비스를 경로로 노출할 수 있습니다.

##### 프로세스

서비스를 노출하려면 다음을 수행하십시오.

1. **OpenShift Container Platform 4**에 로그인합니다.
2. 노출하려는 서비스가 있는 프로젝트에 로그인합니다.

```
$ oc project myproject
```

3. **oc expose service** 명령을 실행하여 경로를 노출합니다.

```
$ oc expose service nodejs-ex
```

출력 예

```
route.route.openshift.io/nodejs-ex exposed
```

4. 서비스가 노출되었는지 확인하려면 **cURL**과 같은 툴을 사용하여 클러스터 외부에서 서비스에 액세스할 수 있는지 확인할 수 있습니다.
  - a. **oc get route** 명령을 사용하여 경로의 호스트 이름을 찾습니다.

■

```
$ oc get route
```

출력 예

```
NAME          HOST/PORT          PATH SERVICES  PORT  TERMINATION
WILDCARD
nodejs-ex     nodejs-ex-myproject.example.com  nodejs-ex  8080-tcp
None
```

b.

*cURL*을 사용하여 호스트가 **GET** 요청에 응답하는지 확인합니다.

```
$ curl --head nodejs-ex-myproject.example.com
```

출력 예

```
HTTP/1.1 200 OK
```

```
...
```

#### 24.4.5. 로드 밸런서 서비스 생성

다음 절차에 따라 로드 밸런서 서비스를 생성합니다.

##### 사전 요구 사항

- 노출하려는 프로젝트와 서비스가 존재하는지 확인합니다.

##### 프로세스

로드 밸런서 서비스를 생성하려면 다음을 수행합니다.

1. **OpenShift Container Platform 4**에 로그인합니다.



2. *노출하려는 서비스가 있는 프로젝트를 로드합니다.*

```
$ oc project project1
```

3. *필요에 따라 컨트롤 플레인 노드에서 텍스트 파일을 열고 다음 텍스트를 붙여넣고 파일을 편집합니다.*

로드 밸런서 구성 파일 샘플

```
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**

*로드 밸런서 서비스를 설명하는 이름을 입력합니다.*

**2**

*노출하려는 서비스가 수신 대기 중인 포트와 동일한 포트를 입력합니다.*

**3**

*로드 밸런서를 통한 트래픽을 제한하려면 특정 IP 주소 목록을 입력합니다. cloud-provider가 이 기능을 지원하지 않는 경우 이 필드는 무시됩니다.*

**4**

유형으로 **Loadbalancer**를 입력합니다.

5

서비스 이름을 입력합니다.



참고

로드 밸런서를 통한 트래픽을 특정 IP 주소로 제한하려면 **loadBalancerSourceRanges** 필드를 설정하지 않고 **service.beta.kubernetes.io/load-balancer-source-ranges** 주석을 사용하는 것이 좋습니다. 주석을 사용하면 향후 릴리스에서 구현될 **OpenShift API**로 더 쉽게 마이그레이션할 수 있습니다.

4. 파일을 저장하고 종료합니다.

5. 다음 명령을 실행하여 서비스를 생성합니다.

```
$ oc create -f <file-name>
```

예를 들면 다음과 같습니다.

```
$ oc create -f mysql-lb.yaml
```

6. 새 서비스를 보려면 다음 명령을 실행합니다.

```
$ oc get svc
```

출력 예

| NAME     | TYPE         | CLUSTER-IP    | EXTERNAL-IP                           | PORT(S)        |
|----------|--------------|---------------|---------------------------------------|----------------|
| AGE      |              |               |                                       |                |
| egress-2 | LoadBalancer | 172.30.22.226 | ad42f5d8b303045-487804948.example.com | 3306:30357/TCP |
|          |              | 15m           |                                       |                |

활성화된 클라우드 공급자가 있는 경우 서비스에 외부 IP 주소가 자동으로 할당됩니다.

7.

마스터에서 **cURL**과 같은 도구를 사용하여 공개 IP 주소로 서비스에 도달할 수 있는지 확인합니다.

```
$ curl <public-ip>:<port>
```

예를 들면 다음과 같습니다.

```
$ curl 172.29.121.74:3306
```

이 섹션의 예제에서는 클라이언트 애플리케이션이 필요한 **MySQL** 서비스를 사용합니다. 패킷이 잘못됨이라는 메시지가 포함된 문자열이 표시되면 서비스에 연결된 것입니다.

**MySQL** 클라이언트가 있는 경우 표준 **CLI** 명령으로 로그인하십시오.

```
$ mysql -h 172.30.131.89 -u admin -p
```

출력 예

```
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
```

```
MySQL [(none)]>
```

#### 24.5. AWS에서 수신 클러스터 트래픽 구성

**OpenShift Container Platform**에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법은 **AWS**의 로드 밸런서, 특히 **NLB(Network Load Balancer)** 또는 **Classic Load Balancer(CLB)**를 사용합니다. 두 가지 유형의 로드 밸런서 모두 클라이언트의 IP 주소를 노드로 전달할

수 있지만 **CLB**에는 **OpenShift Container Platform**이 자동으로 활성화하는 프록시 프로토콜 지원이 필요합니다.

이러한 로드 밸런서를 새 또는 기존 **AWS** 클러스터에서 구성할 수 있습니다.

### 24.5.1. AWS에서 클래식 로드 밸런서 시간 제한 구성

**OpenShift Container Platform**에서는 특정 경로 또는 **Ingress** 컨트롤러에 대한 사용자 정의 시간 제한 기간을 설정하는 방법을 제공합니다. 또한 **AWS Classic Load Balancer(CLB)**의 기본 시간 제한은 60 초입니다.

**CLB**의 시간제한 기간이 경로 시간 초과 또는 **Ingress** 컨트롤러 타임아웃보다 짧은 경우 로드 밸런서는 연결을 조기에 종료할 수 있습니다. 경로와 **CLB**의 시간 제한 시간을 모두 늘려 이 문제를 방지할 수 있습니다.

#### 24.5.1.1. 경로 시간 초과 구성

**SLA(Service Level Availability)** 목적에 필요한 낮은 시간 초과 또는 백엔드가 느린 경우 높은 시간 초과가 필요한 서비스가 있는 경우 기존 경로에 대한 기본 시간 초과를 구성할 수 있습니다.

#### 사전 요구 사항

- 실행 중인 클러스터에 배포된 **Ingress** 컨트롤러가 필요합니다.

#### 절차

1. **oc annotate** 명령을 사용하여 경로에 시간 초과를 추가합니다.

```
$ oc annotate route <route_name> \
  --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> 1
```

1

지원되는 시간 단위는 마이크로초(us), 밀리초(ms), 초(s), 분(m), 시간(h) 또는 일(d)입니다.

다음 예제는 이름이 **myroute**인 경로에서 2초의 시간 초과를 설정합니다.

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

### 24.5.1.2. 클래식 로드 밸런서 시간 제한 설정

**Classic Load Balancer(CLB)**의 기본 시간 초과를 구성하여 유휴 연결을 확장할 수 있습니다.

#### 사전 요구 사항

- 실행 중인 클러스터에 배포된 **Ingress** 컨트롤러가 있어야 합니다.

#### 절차

1. 다음 명령을 실행하여 기본 **ingresscontroller**에 대해 **AWS** 연결 유휴 시간 제한을 **5분**으로 설정합니다.

```
$ 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. 선택 사항: 다음 명령을 실행하여 시간 초과의 기본값을 복원하십시오.

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
--type=merge --patch='{ "spec":{ "endpointPublishingStrategy": \
{ "loadBalancer": { "providerParameters": { "aws": { "classicLoadBalancer": \
{ "connectionIdleTimeout": null } } } } } } }'
```

#### 참고

현재 범위가 이미 설정되어 있지 않으면 연결 제한 시간 값을 변경할 때 범위 필드를 지정해야 합니다. 범위 필드를 설정하면 기본 제한 시간 값을 복원할 때 다시 수행할 필요가 없습니다.

### 24.5.2. 네트워크 로드 밸런서를 사용하여 AWS에서 수신 클러스터 트래픽 구성

**OpenShift Container Platform**에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있는 방법을 제공합니다. 이러한 방법 중 하나는 **NLB(Network Load Balancer)**를 사용합니다. 신규 또는 기존 **AWS** 클러스터에서 **NLB**를 구성할 수 있습니다.

### 24.5.2.1. Ingress Controller classic 로드 밸런서를 네트워크 로드 밸런서로 교체

클래식 로드 밸런서(CLB)를 사용하는 Ingress 컨트롤러를 AWS에서 NLB(Network Load Balancer)를 사용하는 Ingress 컨트롤러로 교체할 수 있습니다.



#### 주의

이 절차에서는 새로운 DNS 레코드 전파, 새 로드 밸런서 프로비저닝 및 기타 요인으로 인해 예상되는 중단으로 인해 몇 분 동안 중단될 수 있습니다. 이 절차를 적용한 후 Ingress 컨트롤러 로드 밸런서의 IP 주소 및 표준 이름이 변경될 수 있습니다.

#### 절차

1.

새 기본 Ingress 컨트롤러로 파일을 생성합니다. 다음 예제에서는 기본 Ingress 컨트롤러에 외부 범위가 있고 다른 사용자 정의가 없는 것으로 가정합니다.

#### ingresscontroller.yml 파일의 예

```
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
```

기본 Ingress 컨트롤러에 다른 사용자 정의가 있는 경우 그에 따라 파일을 수정해야 합니다.

2.

**Ingress** 컨트롤러 YAML 파일을 강제 대체합니다.

```
$ oc replace --force --wait -f ingresscontroller.yml
```

**Ingress** 컨트롤러가 교체될 때까지 기다립니다. 서버 가동 중단 시간(*several of minutes*) 이 발생할 것으로 예상합니다.

#### 24.5.2.2. 기존 AWS 클러스터에서 Ingress 컨트롤러 네트워크 로드 밸런서 생성

기존 클러스터에서 **AWS NLB(Network Load Balancer)**가 지원하는 **Ingress** 컨트롤러를 생성할 수 있습니다.

사전 요구 사항

- **AWS** 클러스터가 설치되어 있어야 합니다.
- 인프라 리소스의 **PlatformStatus**는 **AWS**여야 합니다.
  - **PlatformStatus**가 **AWS**인지 확인하려면 다음을 실행하십시오.

```
$ oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.type}'
AWS
```

절차

기존 클러스터에서 **AWS NLB**가 지원하는 **Ingress** 컨트롤러를 생성합니다.

1.

**Ingress** 컨트롤러 매니페스트를 생성합니다.

```
$ cat ingresscontroller-aws-nlb.yaml
```

출력 예

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
```

```

name: $my_ingress_controller 1
namespace: openshift-ingress-operator
spec:
  domain: $my_unique_ingress_domain 2
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: External 3
  providerParameters:
    type: AWS
    aws:
      type: NLB

```

1

\$my\_ingress\_controller를 Ingress 컨트롤러에 대해 고유한 이름으로 교체합니다.

2

\$my\_unique\_ingress\_domain을 클러스터의 모든 Ingress 컨트롤러 간에 고유한 도메인 이름으로 교체합니다. 이 변수는 DNS 이름 <clustername>.<domain>의 하위 도메인 이어야 합니다.

3

내부 NLB를 사용하려면 External을 Internal로 교체할 수 있습니다.

2.

클러스터에서 리소스를 생성합니다.

```
$ oc create -f ingresscontroller-aws-nlb.yaml
```



중요

새 AWS 클러스터에서 Ingress 컨트롤러 NLB를 구성하려면 먼저 설치 구성 파일 생성 절차를 완료해야 합니다.

### 24.5.2.3. 새 AWS 클러스터에서 Ingress 컨트롤러 네트워크 로드 밸런서 생성

새 클러스터에서 AWS NLB(Network Load Balancer)가 지원하는 Ingress 컨트롤러를 생성할 수 있습니다.



## 사전 요구 사항

- **install-config.yaml** 파일을 생성하고 수정합니다.

## 절차

새 클러스터에서 **AWS NLB**가 지원하는 **Ingress** 컨트롤러를 생성합니다.

1. 설치 프로그램이 포함된 디렉터리로 변경하고 매니페스트를 생성합니다.

```
$ ./openshift-install create manifests --dir <installation_directory> 1
```

1

**<installation\_directory>**는 클러스터의 **install-config.yaml** 파일이 포함된 디렉터리의 이름을 지정합니다.

2. **<installation\_directory>/manifests/** 디렉터리에 **cluster-ingress-default-ingresscontroller.yaml**이라는 이름으로 파일을 만듭니다.

```
$ touch <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml 1
```

1

**<installation\_directory>**는 클러스터의 **manifests /** 디렉터리가 포함된 디렉터리 이름을 지정합니다.

파일이 생성되면 다음과 같이 여러 네트워크 구성 파일이 **manifests/** 디렉토리에 나타납니다.

```
$ ls <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml
```

출력 예

```
cluster-ingress-default-ingresscontroller.yaml
```

3.

편집기에서 `cluster-ingress-default-ingresscontroller.yaml` 파일을 열고 원하는 운영자 구성을 설명하는 CR(사용자 정의 리소스)을 입력합니다.

```
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.

`cluster-ingress-default-ingresscontroller.yaml` 파일을 저장하고 텍스트 편집기를 종료합니다.

5.

선택 사항: manifests / `cluster-ingress-default-ingresscontroller.yaml` 파일을 백업합니다. 설치 프로그램은 클러스터를 생성할 때 manifests/ 디렉터리를 삭제합니다.

### 24.5.3. 추가 리소스

•

네트워크 사용자 지정으로 [AWS](#)에 클러스터 설치

•

NLBs 지원에 대한 자세한 내용은 [AWS](#)에서 [Network Load Balancer](#) 지원을 참조하십시오.

•

CLBs의 프록시 프로토콜 지원에 대한 자세한 내용은 [클래식 로드 밸런서에 대한 프록시 프](#)로토콜 지원 구성을 참조하십시오.

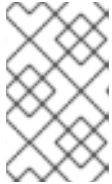
## 24.6. 서비스 외부 IP에 대한 수신 클러스터 트래픽 구성

클러스터 외부의 트래픽에 사용할 수 있도록 외부 IP 주소를 서비스에 연결할 수 있습니다. 이는 일반적으로 베어 메탈 하드웨어에 설치된 클러스터에만 유용합니다. 트래픽을 서비스로 라우팅하려면 외부 네트

워크 인프라를 올바르게 구성해야 합니다.

### 24.6.1. 사전 요구 사항

- 클러스터는 **ExternallIP**가 활성화된 상태로 구성됩니다. 자세한 내용은 [서비스의 ExternallIP 구성을 참조하십시오.](#)



참고

송신 IP에 대해 동일한 **ExternallIP**를 사용하지 마십시오.

### 24.6.2. 서비스에 ExternallIP 연결

서비스에 **ExternallIP**를 연결할 수 있습니다. 클러스터가 **ExternallIP**를 자동으로 할당하도록 구성된 경우, **ExternallIP**를 서비스에 수동으로 연결할 필요가 없습니다.

프로세스

- 선택 사항: **ExternallIP**와 함께 사용하도록 구성된 IP 주소 범위를 확인하려면 다음 명령을 입력합니다.

```
$ oc get networks.config cluster -o jsonpath='{.spec.externallIP}{"\n"}'
```

**autoAssignCIDRs**가 설정된 경우 **spec.externallIPs** 필드가 지정되지 않으면 **OpenShift Container Platform**에서 새 **Service** 오브젝트에 **ExternallIP**를 자동으로 할당합니다.

- 서비스에 **ExternallIP**를 연결합니다.
  - 새 서비스를 생성하는 경우 **spec.externallIPs** 필드를 지정하고 하나 이상의 유효한 IP 주소 배열을 제공합니다. 예를 들면 다음과 같습니다.

```
apiVersion: v1
kind: Service
metadata:
  name: svc-with-externalip
spec:
  ...
  externallIPs:
  - 192.174.120.10
```

b.

**ExternalIP**를 기존 서비스에 연결하는 경우 다음 명령을 입력합니다. **<name>**을 서비스 이름으로 교체합니다. **<ip\_address>**를 유효한 **ExternalIP** 주소로 교체합니다. 쉘표로 구분된 여러 IP 주소를 제공할 수 있습니다.

```
$ oc patch svc <name> -p \
{
  "spec": {
    "externalIPs": [ "<ip_address>" ]
  }
}
```

예를 들면 다음과 같습니다.

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}]'
```

출력 예

```
"mysql-55-rhel7" patched
```

3.

**ExternalIP** 주소가 서비스에 연결되었는지 확인하려면 다음 명령을 입력합니다. 새 서비스에 **ExternalIP**를 지정한 경우 먼저 서비스를 생성해야 합니다.

```
$ oc get svc
```

출력 예

| NAME           | CLUSTER-IP    | EXTERNAL-IP    | PORT(S)  | AGE |
|----------------|---------------|----------------|----------|-----|
| mysql-55-rhel7 | 172.30.131.89 | 192.174.120.10 | 3306/TCP | 13m |

### 24.6.3. 추가 리소스

- 

서비스의 [ExternalIP](#) 구성

## 24.7. NODEPORT를 사용하여 수신 클러스터 트래픽 구성

OpenShift Container Platform에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법에서는 **NodePort**를 사용합니다.

### 24.7.1. NodePort를 사용하여 클러스터로 트래픽 가져오기

클러스터의 모든 노드에서 특정 포트에 서비스를 노출하려면 **NodePort** 유형의 서비스 리소스를 사용하십시오. 포트는 **Service** 리소스의 `.spec.ports[*].nodePort` 필드에 지정됩니다.



#### 중요

노드 포트를 사용하려면 추가 포트 리소스가 필요합니다.

**NodePort**는 서비스를 노드 IP 주소의 정적 포트에 노출합니다. **NodePort**는 기본적으로 30000~32767 범위에 있으며, 서비스에서 의도한 포트와 **NodePort**가 일치하지 않을 수 있습니다. 예를 들어, 포트 8080은 노드에서 포트 31020으로 노출될 수 있습니다.

관리자는 외부 IP 주소가 노드로 라우팅되는지 확인해야 합니다.

**NodePort** 및 외부 IP는 독립적이며 둘 다 동시에 사용할 수 있습니다.



#### 참고

이 섹션의 절차에는 클러스터 관리자가 수행해야 하는 사전 요구 사항이 필요합니다.

### 24.7.2. 사전 요구 사항

다음 절차를 시작하기 전에 관리자는 다음을 수행해야 합니다.

- 요청이 클러스터에 도달할 수 있도록 외부 포트를 클러스터 네트워킹 환경으로 설정합니다.
- 클러스터 관리자 역할의 사용자가 한 명 이상 있는지 확인합니다. 이 역할을 사용자에게 추가하려면 다음 명령을 실행합니다.

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
```

- 클러스터에 대한 네트워크 액세스 권한이 있는 마스터와 노드가 클러스터 외부에 각각 1개 이상씩 있는 **OpenShift Container Platform** 클러스터가 있어야 합니다. 이 절차에서는 외부 시스템이 클러스터와 동일한 서브넷에 있다고 가정합니다. 다른 서브넷에 있는 외부 시스템에 필요한 추가 네트워킹은 이 주제에서 다루지 않습니다.

### 24.7.3. 프로젝트 및 서비스 생성

노출하려는 프로젝트 및 서비스가 존재하지 않는 경우 먼저 프로젝트를 생성한 다음 서비스를 생성합니다.

프로젝트와 서비스가 이미 존재하는 경우에는 서비스 노출 절차로 건너뛰어 경로를 생성합니다.

#### 사전 요구 사항

- **oc CLI**를 설치하고 클러스터 관리자로 로그인합니다.

#### 프로세스

1. **oc new-project** 명령을 실행하여 서비스에 대한 새 프로젝트를 생성합니다.

```
$ oc new-project myproject
```

2. **oc new-app** 명령을 사용하여 서비스를 생성합니다.

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

3. 서비스가 생성되었는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get svc -n myproject
```

#### 출력 예

```
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
nodejs-ex ClusterIP  172.30.197.157 <none>       8080/TCP   70s
```

기본적으로 새 서비스에는 외부 IP 주소가 없습니다.

#### 24.7.4. 경로를 생성하여 서비스 노출

**oc expose** 명령을 사용하여 서비스를 경로로 노출할 수 있습니다.

##### 프로세스

서비스를 노출하려면 다음을 수행하십시오.

1. **OpenShift Container Platform 4**에 로그인합니다.
2. 노출하려는 서비스가 있는 프로젝트에 로그인합니다.

```
$ oc project myproject
```

3. 애플리케이션의 노드 포트를 표시하려면 다음 명령을 입력합니다. **OpenShift Container Platform**은 **30000-32767** 범위에서 사용 가능한 포트를 자동으로 선택합니다.

```
$ oc expose service nodejs-ex --type=NodePort --name=nodejs-ex-nodeport --generator="service/v2"
```

출력 예

```
service/nodejs-ex-nodeport exposed
```

4. 선택 사항: 노드 포트가 노출된 상태로 서비스를 사용할 수 있는지 확인하려면 다음 명령을 입력합니다.

```
$ oc get svc -n myproject
```

출력 예

| 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   |

5.

선택 사항: `oc new-app` 명령에서 자동 생성한 서비스를 제거하려면 다음 명령을 입력합니다.

```
$ oc delete svc nodejs-ex
```

#### 24.7.5. 추가 리소스

- 

[노드 포트 서비스 범위 구성](#)



## 25장. KUBERNETES NMSTATE

### 25.1. KUBERNETES NMSTATE OPERATOR 정보

**Kubernetes NMState Operator**는 **OpenShift Container Platform** 클러스터 노드에서 **NMState**를 사용하여 상태 중심 네트워크 구성을 수행하는 데 필요한 **Kubernetes API**를 제공합니다. **Kubernetes NMState Operator**는 사용자에게 클러스터 노드에서 다양한 네트워크 인터페이스 유형, **DNS** 및 라우팅을 구성하는 기능을 제공합니다. 또한 클러스터 노드의 데몬은 각 노드의 네트워크 인터페이스 상태를 **API 서버**에 정기적으로 보고합니다.



#### 중요

**Red Hat**은 **베어메탈**, **IBM Power**, **IBM Z** 및 **LinuxONE** 설치의 프로덕션 환경에서 **Kubernetes NMState Operator**를 지원합니다.

**OpenShift Container Platform**과 함께 **NMState**를 사용하기 전에 **Kubernetes NMState Operator**를 설치해야 합니다.

#### 25.1.1. Kubernetes NMState Operator 설치

웹 콘솔 또는 **CLI**를 사용하여 **Kubernetes NMState Operator**를 설치할 수 있습니다.

##### 25.1.1.1. 웹 콘솔을 사용하여 Kubernetes NMState Operator 설치

웹 콘솔을 사용하여 **Kubernetes NMState Operator**를 설치할 수 있습니다. **Operator**가 설치되면 **NMState State Controller**를 모든 클러스터 노드에 데몬 세트에 배포할 수 있습니다.

#### 사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 로그인했습니다.

#### 프로세스

1. **Operators** → **OperatorHub**를 선택합니다.
2. 모든 항목 아래의 검색 필드에 **nmstate**를 입력하고 **Enter**를 클릭하여 **Kubernetes NMState Operator**를 검색합니다.

3. **Kubernetes NMState Operator** 검색 결과를 클릭합니다.
4. 설치를 클릭하여 **Operator** 설치 창을 엽니다.
5. 설치를 클릭하여 **Operator**를 설치합니다.
6. **Operator** 설치가 완료되면 **Operator** 보기를 클릭합니다.
7. 제공된 **API** 아래에서 인스턴스 생성을 클릭하여 **kubernetes-nmstate**의 인스턴스 생성을 위해 대화 상자를 엽니다.
8. 대화 상자의 이름 필드에서 인스턴스 이름이 **nmstate**인지 확인합니다.



참고

이름 제한은 알려진 문제입니다. 인스턴스는 전체 클러스터에 대한 단일 생성입니다.

9. 기본 설정을 수락하고 만들기를 클릭하여 인스턴스를 만듭니다.

요약

완료되면 **Operator**가 **NMState State Controller**를 모든 클러스터 노드에 데몬 세트로 배포했습니다.

25.1.1.2. CLI를 사용하여 Kubernetes NMState Operator 설치

**oc**(OpenShift CLI) 를 사용하여 **Kubernetes NMState Operator**를 설치할 수 있습니다. **Operator**가 설치되면 **NMState State Controller**를 모든 클러스터 노드에 데몬 세트로 배포할 수 있습니다.

사전 요구 사항

- **OpenShift CLI(oc)**가 설치되어 있습니다.

- **cluster-admin** 권한이 있는 사용자로 로그인했습니다.

## 프로세스

1. **nmstate Operator** 네임스페이스를 생성합니다.

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

2. **OperatorGroup** 을 생성합니다.

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: NMState.v1.nmstate.io
  generateName: openshift-nmstate-
  name: openshift-nmstate-tn6k8
  namespace: openshift-nmstate
spec:
  targetNamespaces:
    - openshift-nmstate
EOF
```

3. **nmstate Operator**를 서브스크립션합니다.

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  labels:
    operators.coreos.com/kubernetes-nmstate-operator.openshift-nmstate: ""
  name: kubernetes-nmstate-operator
  namespace: openshift-nmstate
spec:
  channel: stable
```

```
installPlanApproval: Automatic
name: kubernetes-nmstate-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
EOF
```

4. *nmstate Operator*의 인스턴스를 생성합니다.

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

**검증**

- *nmstate Operator*의 배포가 실행 중인지 확인합니다.

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

출력 예

```
Name                               Phase
kubernetes-nmstate-operator.4.11.0-202208120157 Succeeded
```

**25.2. 노드 네트워크 상태 관찰**

노드 네트워크 상태는 클러스터의 모든 노드에 대한 네트워크 구성입니다.

**25.2.1. nmstate 정보**

OpenShift Container Platform에서는 **nmstate**를 사용하여 노드 네트워크의 상태를 보고하고 구성합니다. 이를 통해 단일 구성 매니페스트를 클러스터에 적용하여(예: 모든 노드에서 Linux 브리지 생성) 네트워크 정책 구성을 수정할 수 있습니다.

노드 네트워킹은 다음 오브젝트에서 모니터링하고 업데이트합니다.

### NodeNetworkState

해당 노드의 네트워크 상태를 보고합니다.

### NodeNetworkConfigurationPolicy

노드에서 요청된 네트워크 구성을 설명합니다. **NodeNetworkConfigurationPolicy** 매니페스트를 클러스터에 적용하는 방식으로 인터페이스 추가 및 제거를 포함하여 노드 네트워크 구성을 업데이트합니다.

### NodeNetworkConfigurationEnactment

각 노드에 적용된 네트워크 정책을 보고합니다.

OpenShift Container Platform에서는 다음 **nmstate** 인터페이스 유형을 사용할 수 있습니다.

- Linux 브리지
- VLAN
- 본딩
- 이더넷

참고

OpenShift Container Platform 클러스터에서 OVN-Kubernetes를 기본 CNI(Container Network Interface) 공급자로 사용하는 경우, OVN-Kubernetes의 호스트 네트워크 토폴로지 변경으로 인해 호스트의 기본 인터페이스에 Linux 브리지 또는 본딩을 연결할 수 없습니다. 해결 방법으로 호스트에 연결된 보조 네트워크 인터페이스를 사용하거나 OpenShift SDN 기본 CNI 공급자로 전환할 수 있습니다.

#### 25.2.2. 노드의 네트워크 상태 보기

**NodeNetworkState** 오브젝트는 클러스터의 모든 노드에 존재합니다. 이 오브젝트는 주기적으로 업데이트되며 해당 노드의 네트워크 상태를 캡처합니다.

절차

1. 클러스터의 모든 **NodeNetworkState** 오브젝트를 나열합니다.

```
$ oc get nns
```

2. **NodeNetworkState** 오브젝트를 검사하여 해당 노드의 네트워크를 확인합니다. 이 예제의 출력은 명확성을 위해 수정되었습니다.

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

출력 예

```
apiVersion: nmstate.io/v1
kind: NodeNetworkState
metadata:
  name: node01 1
status:
  currentState: 2
  dns-resolver:
  ...
  interfaces:
  ...
  route-rules:
  ...
  routes:
  ...
  lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" 3
```

1

**NodeNetworkState** 오브젝트의 이름은 노드에서 가져옵니다.

2

**currentState**에는 **DNS**, 인터페이스, 경로를 포함하여 노드에 대한 전체 네트워크 구성이 포함됩니다.

3

마지막으로 성공한 업데이트의 타임 스탬프 노드에 연결할 수 있는 동안 주기적으로 업데이트되고 보고서의 최신 상태를 평가하는 데 사용됩니다.

### 25.3. 노드 네트워크 구성 업데이트

**NodeNetworkConfigurationPolicy** 매니페스트를 클러스터에 적용하여 노드 네트워크 구성을 업데이트(예: 노드에서 인터페이스 추가 또는 제거)할 수 있습니다.

#### 25.3.1. nmstate 정보

**OpenShift Container Platform**에서는 **nmstate**를 사용하여 노드 네트워크의 상태를 보고하고 구성합니다. 이를 통해 단일 구성 매니페스트를 클러스터에 적용하여(예: 모든 노드에서 **Linux** 브리지 생성) 네트워크 정책 구성을 수정할 수 있습니다.

노드 네트워킹은 다음 오브젝트에서 모니터링하고 업데이트합니다.

#### **NodeNetworkState**

해당 노드의 네트워크 상태를 보고합니다.

#### **NodeNetworkConfigurationPolicy**

노드에서 요청된 네트워크 구성을 설명합니다. **NodeNetworkConfigurationPolicy** 매니페스트를 클러스터에 적용하는 방식으로 인터페이스 추가 및 제거를 포함하여 노드 네트워크 구성을 업데이트합니다.

#### **NodeNetworkConfigurationEnactment**

각 노드에 적용된 네트워크 정책을 보고합니다.

**OpenShift Container Platform**에서는 다음 **nmstate** 인터페이스 유형을 사용할 수 있습니다.

- **Linux 브리지**
- **VLAN**

- 본딩
- 이더넷



참고

**OpenShift Container Platform 클러스터에서 OVN-Kubernetes를 기본 CNI(Container Network Interface) 공급자로 사용하는 경우, OVN-Kubernetes의 호스트 네트워크 토폴로지 변경으로 인해 호스트의 기본 인터페이스에 Linux 브리지 또는 본딩을 연결할 수 없습니다. 해결 방법으로 호스트에 연결된 보조 네트워크 인터페이스를 사용하거나 OpenShift SDN 기본 CNI 공급자로 전환할 수 있습니다.**

25.3.2. 노드에서 인터페이스 만들기

**NodeNetworkConfigurationPolicy** 매니페스트를 클러스터에 적용하여 클러스터의 노드에서 인터페이스를 만듭니다. 매니페스트는 요청된 인터페이스 구성을 자세히 설명합니다.

기본적으로 매니페스트는 클러스터의 모든 노드에 적용됩니다. 특정 노드에 인터페이스를 추가하려면 **spec: nodeSelector** 매개변수와 노드 선택기에 적합한 **<key>:<value>**를 추가합니다.

여러 **nmstate-enabled** 노드를 동시에 구성할 수 있습니다. 구성은 노드의 **50%**에 병렬로 적용됩니다. 이 전략을 사용하면 네트워크 연결에 실패하면 전체 클러스터를 사용할 수 없습니다. 클러스터의 특정 부분에 병렬로 정책 구성을 적용하려면 **maxUnavailable** 필드를 사용합니다.

절차

1. **NodeNetworkConfigurationPolicy** 매니페스트를 생성합니다. 다음 예제는 모든 작업자 노드에서 **Linux** 브릿지를 구성하고 **DNS** 확인자를 구성합니다.

```

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

정책 이름입니다.

2

선택 사항: **nodeSelector** 매개변수를 포함하지 않으면 정책이 클러스터의 모든 노드에 적용됩니다.

3

이 예제에서는 **node-role.kubernetes.io/worker: ""** 노드 선택기를 사용하여 클러스터의 모든 작업자 노드를 선택합니다.

4

선택 사항: 정책 구성을 동시에 적용할 수 있는 최대 **nmstate-enabled** 노드 수를 지정합니다. 이 매개변수는 백분율 값(문자열)으로 설정할 수 있습니다(예: "10% ") 또는 절대 값(number)(예: 3)으로 설정할 수 있습니다.

5

선택 사항: 사람이 읽을 수 있는 인터페이스 설명입니다.

6

선택 사항: DNS 서버의 검색 및 서버 설정을 지정합니다.

2.

노드 네트워크 정책을 생성합니다.

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

1

노드 네트워크 구성 정책 매니페스트의 파일 이름입니다.

#### 추가 리소스

- [동일한 정책에서 여러 인터페이스를 만드는 예제](#)
- [정책의 다양한 IP 관리 방법 예제](#)

### 25.3.3. 노드에 노드 네트워크 정책 업데이트 확인

**NodeNetworkConfigurationPolicy** 매니페스트는 클러스터의 노드에 대해 요청된 네트워크 구성을 설명합니다. 노드 네트워크 정책에는 요청된 네트워크 구성과 클러스터 전체에 대한 정책 실행 상태가 포함됩니다.

노드 네트워크 정책을 적용하면 클러스터의 모든 노드에 대해 **NodeNetworkConfigurationEnactment** 오브젝트가 생성됩니다. 노드 네트워크 구성 시행은 해당 노드에서 정책의 실행 상태를 나타내는 읽기 전용 오브젝트입니다. 정책이 노드에 적용되지 않으면 문제 해결을 위해 해당 노드에 대한 시행에 역추적이 포함됩니다.

#### 절차

1.

정책이 클러스터에 적용되었는지 확인하려면 정책과 해당 상태를 나열합니다.

```
$ oc get nncp
```

2.

선택 사항: 정책을 구성하는 데 예상보다 오래 걸리는 경우 특정 정책의 요청된 상태 및 상태 조건을 검사할 수 있습니다.

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

3.

선택 사항: 모든 노드에서 정책을 구성하는 데 예상보다 오래 걸리는 경우 클러스터의 시행 상태를 나열할 수 있습니다.

```
$ oc get nnce
```

4.

선택 사항: 구성 실패에 대한 오류 보고를 포함하여 특정 시행의 구성을 확인하려면 다음 명령을 실행하십시오.

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

### 25.3.4. 노드에서 인터페이스 제거

**NodeNetworkConfigurationPolicy** 오브젝트를 편집하고 인터페이스의 **state**를 없음으로 설정하여 클러스터의 1개 이상의 노드에서 인터페이스를 제거할 수 있습니다.

노드에서 인터페이스를 제거해도 노드 네트워크 구성이 이전 상태로 자동 복원되지 않습니다. 이전 상태를 복원하려면 정책에서 노드 네트워크 구성을 정의해야 합니다.

브리지 또는 본딩 인터페이스를 제거하면 이전에 해당 브릿지 또는 본딩 인터페이스에 연결되었거나 종속되었던 클러스터의 모든 노드 **NIC**가 **down** 상태가 되어 연결할 수 없습니다. 연결 손실을 방지하기 위해, 노드 **NIC**를 동일한 정책으로 구성하여 **DHCP** 또는 고정 **IP** 주소의 상태를 **up**으로 구성합니다.

#### 참고

인터페이스를 추가한 노드 네트워크 정책을 삭제해도 노드의 정책 구성은 변경되지 않습니다. **NodeNetworkConfigurationPolicy**는 클러스터의 오브젝트이지만 요청된 구성만 나타냅니다.

마찬가지로 인터페이스를 제거해도 정책은 삭제되지 않습니다.

#### 절차

1.

인터페이스를 생성하는 데 사용되는 **NodeNetworkConfigurationPolicy** 매니페스트를 업데이트합니다. 다음 예에서는 **Linux** 브릿지를 제거한 후 연결이 손실되지 않도록 **DHCP**로 **eth1** **NIC**를 구성합니다.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> ❶
spec:
  nodeSelector: ❷
    node-role.kubernetes.io/worker: "" ❸
  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

정책 이름입니다.

2

선택 사항: **nodeSelector** 매개변수를 포함하지 않으면 정책이 클러스터의 모든 노드에 적용됩니다.

3

이 예제에서는 **node-role.kubernetes.io/worker: ""** 노드 선택기를 사용하여 클러스터의 모든 작업자 노드를 선택합니다.

4

**absent** 상태로 변경하면 인터페이스가 제거됩니다.

5

브리지 인터페이스에서 연결을 해제할 인터페이스의 이름입니다.

6

인터페이스 유형입니다. 이 예제에서는 이더넷 네트워킹 인터페이스를 생성합니다.

7

인터페이스에 요청되는 상태입니다.

8

선택 사항: **dhcp**를 사용하지 않는 경우 고정 IP를 설정하거나 IP 주소 없이 인터페이스를 종료할 수 있습니다.

9

이 예제에서 `ipv4`를 활성화합니다.

2.

노드에서 정책을 업데이트하고 인터페이스를 제거합니다.

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

①

정책 매니페스트의 파일 이름입니다.

### 25.3.5. 다양한 인터페이스에 대한 예제 정책 구성

#### 25.3.5.1. 예: Linux 브리지 인터페이스 노드 네트워크 구성 정책

`NodeNetworkConfigurationPolicy` 매니페스트를 클러스터에 적용하여 클러스터의 노드에서 **Linux** 브리지 인터페이스를 만듭니다.

다음 **YAML** 파일은 **Linux** 브리지 인터페이스의 매니페스트 예제입니다. 여기에는 해당 정보로 교체해야 하는 샘플 값이 포함되어 있습니다.

```
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

정책 이름입니다.

2

선택 사항: **nodeSelector** 매개변수를 포함하지 않으면 정책이 클러스터의 모든 노드에 적용됩니다.

3

이 예제에서는 **hostname** 노드 선택기를 사용합니다.

4

인터페이스 이름입니다.

5

선택 사항: 사람이 읽을 수 있는 인터페이스 설명입니다.

6

인터페이스 유형입니다. 이 예제에서는 브리지를 만듭니다.

7

생성 후 인터페이스에 요청되는 상태입니다.

8

선택 사항: **dhcp**를 사용하지 않는 경우 고정 **IP**를 설정하거나 **IP** 주소 없이 인터페이스를 종료할 수 있습니다.

9

이 예제에서 **ipv4**를 활성화합니다.

10

이 예제에서 **stp**를 비활성화합니다.

11

브릿지가 연결되는 노드 **NIC**입니다.

### 25.3.5.2. 예제: VLAN 인터페이스 노드 네트워크 구성 정책

**NodeNetworkConfigurationPolicy** 매니페스트를 클러스터에 적용하여 클러스터의 노드에서 **VLAN** 인터페이스를 만듭니다.

다음 **YAML** 파일은 **VLAN** 인터페이스의 매니페스트 예제입니다. 여기에는 해당 정보로 교체해야 하는 샘플 값이 포함되어 있습니다.

```
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

정책 이름입니다.

②

선택 사항: **nodeSelector** 매개변수를 포함하지 않으면 정책이 클러스터의 모든 노드에 적용됩니다.

③

이 예제에서는 **hostname** 노드 선택기를 사용합니다.

④

인터페이스 이름입니다.

⑤

선택 사항: 사람이 읽을 수 있는 인터페이스 설명입니다.

6

인터페이스 유형입니다. 이 예제에서는 **VLAN**을 만듭니다.

7

생성 후 인터페이스에 요청되는 상태입니다.

8

**VLAN**이 연결되는 노드 **NIC**입니다.

9

**VLAN** 태그입니다.

### 25.3.5.3. 예제: 본딩 인터페이스 노드 네트워크 구성 정책

**NodeNetworkConfigurationPolicy** 매니페스트를 클러스터에 적용하여 클러스터의 노드에서 본딩 인터페이스를 만듭니다.



## 참고

OpenShift Container Platform에서는 다음과 같은 본딩 모드만 지원합니다.

- `mode=1 active-backup`
- `mode=2 balance-xor`
- `mode=4 802.3ad`
- `mode=5 balance-tlb`
- `mode=6 balance-alb`

다음 YAML 파일은 본딩 인터페이스의 매니페스트 예제입니다. 여기에는 해당 정보로 교체해야 하는 샘플 값이 포함되어 있습니다.

```

apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy ❶
spec:
  nodeSelector: ❷
    kubernetes.io/hostname: <node01> ❸
  desiredState:
    interfaces:
      - name: bond0 ❹
        description: Bond with ports eth1 and eth2 ❺
        type: bond ❻
        state: up ❼
        ipv4:
          dhcp: true ❽
          enabled: true ❾
        link-aggregation:
          mode: active-backup ❿
        options:

```

```
miimon: '140' 11  
port: 12  
- eth1  
- eth2  
mtu: 1450 13
```

1

정책 이름입니다.

2

선택 사항: **nodeSelector** 매개변수를 포함하지 않으면 정책이 클러스터의 모든 노드에 적용됩니다.

3

이 예제에서는 **hostname** 노드 선택기를 사용합니다.

4

인터페이스 이름입니다.

5

선택 사항: 사람이 읽을 수 있는 인터페이스 설명입니다.

6

인터페이스 유형입니다. 이 예제에서는 본딩을 생성합니다.

7

생성 후 인터페이스에 요청되는 상태입니다.

8

선택 사항: **dhcp**를 사용하지 않는 경우 고정 IP를 설정하거나 IP 주소 없이 인터페이스를 종료할 수 있습니다.

9

이 예제에서 **ipv4**를 활성화합니다.

10

11

선택 사항: 이 예제에서는 `miimon`을 사용하여 140ms마다 본딩 링크를 검사합니다.

12

본딩의 하위 노드 `NIC`입니다.

13

선택 사항: 본딩의 `MTU`(최대 전송 단위)입니다. 지정하지 않는 경우 이 값은 기본적으로 1500으로 설정됩니다.

#### 25.3.5.4. 예제: 이더넷 인터페이스 노드 네트워크 구성 정책

`NodeNetworkConfigurationPolicy` 매니페스트를 클러스터에 적용하여 클러스터의 노드에서 이더넷 인터페이스를 구성합니다.

다음 `YAML` 파일은 이더넷 인터페이스의 매니페스트 예제입니다. 여기에는 해당 정보로 교체해야 하는 샘플 값이 포함되어 있습니다.

```

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

정책 이름입니다.

2

3

이 예제에서는 **hostname** 노드 선택기를 사용합니다.

4

인터페이스 이름입니다.

5

선택 사항: 사람이 읽을 수 있는 인터페이스 설명입니다.

6

인터페이스 유형입니다. 이 예제에서는 이더넷 네트워킹 인터페이스를 생성합니다.

7

생성 후 인터페이스에 요청되는 상태입니다.

8

선택 사항: **dhcp**를 사용하지 않는 경우 고정 **IP**를 설정하거나 **IP** 주소 없이 인터페이스를 종료할 수 있습니다.

9

이 예제에서 **ipv4**를 활성화합니다.

#### 25.3.5.5. 예제: 노드 네트워크 구성 정책이 동일한 여러 인터페이스

동일한 노드 네트워크 구성 정책으로 여러 개의 인터페이스를 생성할 수 있습니다. 이러한 인터페이스는 서로를 참조할 수 있으므로 단일 정책 매니페스트를 사용하여 네트워크 구성을 빌드하고 배포할 수 있습니다.

다음 예제 스니펫에서는 두 **NIC**에 걸친 **bond10**이라는 본딩과 이 본딩에 연결되는 **br1**이라는 **Linux** 브리지를 생성합니다.

```
#...
interfaces:
- name: bond10
  description: Bonding eth2 and eth3 for Linux bridge
```

```

type: bond
state: up
link-aggregation:
  port:
  - eth2
  - eth3
- name: br1
description: Linux bridge on bond
type: linux-bridge
state: up
bridge:
  port:
  - name: bond10
#...

```

### 25.3.6. 브리지에 연결된 NIC의 고정 IP 캡처

#### 중요

NIC의 고정 IP 캡처는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 [기술 프리뷰 기능 지원 범위](#)를 참조하십시오.

#### 25.3.6.1. 예: 브리지에 연결된 NIC에서 고정 IP 주소를 상속하는 Linux 브리지 인터페이스 노드 네트워크 구성 정책

클러스터의 노드에서 Linux 브리지 인터페이스를 만들고 단일 `NodeNetworkConfigurationPolicy` 매니페스트를 클러스터에 적용하여 NIC의 고정 IP 구성을 브리지로 전송합니다.

다음 YAML 파일은 Linux 브리지 인터페이스의 매니페스트 예제입니다. 여기에는 해당 정보로 교체해야 하는 샘플 값이 포함되어 있습니다.

```

apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-copy-ipv4-policy ❶
spec:
  nodeSelector: ❷
    node-role.kubernetes.io/worker: ""
  capture:
    eth1-nic: interfaces.name=="eth1" ❸

```

```

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

정책의 이름입니다.

2

선택 사항: `nodeSelector` 매개변수를 포함하지 않으면 정책이 클러스터의 모든 노드에 적용됩니다. 이 예제에서는 `node-role.kubernetes.io/worker: ""` 노드 선택기를 사용하여 클러스터의 모든 작업자 노드를 선택합니다.

3

브릿지가 연결되는 노드 NIC에 대한 참조입니다.

4

인터페이스 유형입니다. 이 예제에서는 브리지를 만듭니다.

5

브리지 인터페이스의 IP 주소입니다. 이 값은 `spec.capture.eth1-nic` 항목에서 참조하는 NIC의 IP 주소와 일치합니다.

6

브릿지가 연결되는 노드 NIC입니다.

추가 리소스

## NMPolicy 프로젝트 - 정책 구문

### 25.3.7. 예제: IP 관리

다음 예제 구성 스니펫에서는 다양한 IP 관리 방법을 보여줍니다.

이 예제에서는 **ethernet** 인터페이스 유형을 사용하여 예제를 단순화하면서 정책 구성에 관련 컨텍스트를 표시합니다. 이러한 IP 관리 예제는 다른 인터페이스 유형과 함께 사용할 수 있습니다.

#### 25.3.7.1. 고정

다음 스니펫은 이더넷 인터페이스에서 IP 주소를 정적으로 구성합니다.

```
...
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**

이 값을 인터페이스의 고정 IP 주소로 교체합니다.

#### 25.3.7.2. IP 주소 없음

다음 스니펫에서는 인터페이스에 IP 주소가 없습니다.

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

```

```

ipv4:
  enabled: false

```

```
...
```

### 25.3.7.3. 동적 호스트 구성

다음 스니펫에서는 동적 **IP** 주소, 게이트웨이 주소, **DNS**를 사용하는 이더넷 인터페이스를 구성합니다.

```

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

```

```
...
```

다음 스니펫에서는 동적 **IP** 주소를 사용하지만 동적 게이트웨이 주소 또는 **DNS**를 사용하지 않는 이더넷 인터페이스를 구성합니다.

```

...
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

```

```
...
```

### 25.3.7.4. DNS

**DNS** 구성을 설정하는 것은 `/etc/resolv.conf` 파일을 수정하는 것을 의미합니다. 다음 스니펫에서는 호스트에 **DNS** 구성을 설정합니다.

```

...
interfaces: 1
  ...
  ipv4:
    ...
    auto-dns: false
    ...

```



```

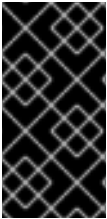
dns-resolver:
  config:
    search:
      - example.com
      - example.org
    server:
      - 8.8.8.8

```

...

1

**auto-dns: false** 로 인터페이스를 구성해야 합니다. 또는 **Kubernetes NMState**가 사용자 지정 DNS 설정을 저장하려면 인터페이스에서 고정 IP 구성을 사용해야 합니다.



중요

DNS 확인자를 구성할 때 인터페이스로 **OVNKubernetes** 관리 **Open vSwitch** 브리지인 **br-ex** 를 사용할 수 없습니다.

#### 25.3.7.5. 고정 라우팅

다음 스니펫에서는 **eth1** 인터페이스에 고정 경로와 고정 IP를 구성합니다.

```

...
interfaces:
- name: eth1
  description: Static routing on eth1
  type: ethernet
  state: up
  ipv4:
    dhcp: false
    address:
      - ip: 192.0.2.251 1
        prefix-length: 24
        enabled: true
  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

이더넷 인터페이스의 고정 IP 주소입니다.

## 2

노드 트래픽의 다음 홉 주소입니다. 이더넷 인터페이스에 설정된 IP 주소와 동일한 서브넷에 있어야 합니다.

## 25.4. 노드 네트워크 구성 문제 해결

노드 네트워크 구성에 문제가 발생하면 정책이 자동으로 롤백되고 시행이 실패로 보고됩니다. 여기에는 다음과 같은 문제가 포함됩니다.

- 호스트에 구성을 적용하지 못했습니다.
- 호스트와 기본 게이트웨이의 연결이 끊어졌습니다.
- 호스트와 API 서버의 연결이 끊어졌습니다.

### 25.4.1. 잘못된 노드 네트워크 구성 정책의 구성 문제 해결

노드 네트워크 구성 정책을 적용하여 전체 클러스터에 노드 네트워크 구성 변경 사항을 적용할 수 있습니다. 잘못된 구성을 적용하는 경우 다음 예제를 사용하여 실패한 노드 네트워크 정책의 문제를 해결하고 수정할 수 있습니다.

이 예에서는 컨트롤 플레인 노드와 세 개의 컴퓨팅 노드가 있는 예제 클러스터에 **Linux** 브리지 정책을 적용합니다. 이 정책은 잘못된 인터페이스를 참조하므로 적용되지 않습니다. 오류를 찾기 위해 가능한 **NMState** 리소스를 조사합니다. 그런 다음 올바른 구성으로 정책을 업데이트할 수 있습니다.

#### 절차

1. 정책을 생성하여 클러스터에 적용합니다. 다음 예제에서는 **ens01** 인터페이스에서 간단한 브리지를 생성합니다.

```
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

```

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

출력 예

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

2.

다음 명령을 실행하여 정책의 상태를 확인합니다.

```
$ oc get nncp
```

출력에 정책이 실패했다는 내용이 표시됩니다.

출력 예

| NAME                  | STATUS            |
|-----------------------|-------------------|
| ens01-bridge-testfail | FailedToConfigure |

그러나 정책 상태만으로는 모든 노드에서 실패했는지 노드 서브 세트에서 실패했는지 알 수 없습니다.

3.

노드 네트워크 구성 시행을 나열하여 정책이 모든 노드에서 성공적인지 확인합니다. 정책이

노드 서브 세트에서만 실패한 경우 특정 노드 구성에 문제가 있음을 나타냅니다. 정책이 모든 노드에서 실패하면 정책에 문제가 있음을 나타냅니다.

```
$ oc get nnce
```

출력에 정책이 모든 노드에서 실패했다는 내용이 표시됩니다.

출력 예

| 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 |

4.

실패한 시행 중 하나에서 역추적을 살펴봅니다. 다음 명령은 출력 톨 `jsonpath`를 사용하여 출력을 필터링합니다.

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

이 명령은 간결하게 편집된 대규모 역추적 정보를 반환합니다.

출력 예

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to
execute nmstatectl set --no-commit --timeout 480: 'exit status 1' "
...
libnmstate.error.NmstateVerificationError:
desired
=====
---
name: br1
type: linux-bridge
state: up
bridge:
options:
```

```

group-forward-mask: 0
mac-ageing-time: 300
multicast-snooping: true
stp:
  enabled: false
  forward-delay: 15
  hello-time: 2
  max-age: 20
  priority: 32768
port:
- name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

current
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
      priority: 32768
  port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

```

```

difference

```

```

=====
--- desired
+++ current
@@ -13,8 +13,7 @@
     hello-time: 2
     max-age: 20
     priority: 32768
- port:
- - name: ens01
+ port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
line 651, in _assert_interfaces_equal\n
current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:

```

**NmstateVerificationError**는 **desired** 정책 구성, 노드에 있는 정책의 **current** 구성, 일치하지 않는 매개변수를 강조하는 **difference**를 나열합니다. 이 예에서 **port**는 **difference**에 포함되어 있으며, 이는 정책의 포트 구성이 문제임을 나타냅니다.

5.

정책이 제대로 구성되었는지 확인하기 위해 **NodeNetworkState** 오브젝트를 요청하여 하나 또는 모든 노드의 네트워크 구성을 확인합니다. 다음 명령에서는 **control-plane-1** 노드의 네트워크 구성을 반환합니다.

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

출력에 노드의 인터페이스 이름이 **ens1**인데 실패한 정책에서 **ens01**로 잘못 사용하고 있다는 내용이 표시됩니다.

출력 예

```

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

```

6. 기존 정책을 편집하여 오류를 수정합니다.

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

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

정책을 저장하여 수정 사항을 적용합니다.

7. 정책 상태를 확인하여 업데이트가 완료되었는지 확인합니다.

```
$ oc get nncp
```

출력 예

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

업데이트된 정책이 클러스터의 모든 노드에 성공적으로 구성되었습니다.

## 26장. 클러스터 전체 프록시 구성

프로덕션 환경에서는 인터넷에 대한 직접 액세스를 거부하고 대신 HTTP 또는 HTTPS 프록시를 사용할 수 있습니다. 기존 클러스터의 프록시 오브젝트를 수정하거나 새 클러스터의 `install-config.yaml` 파일에서 프록시 설정을 구성하여 OpenShift Container Platform을 프록시를 사용하도록 구성할 수 있습니다.

### 26.1. 사전 요구 사항

- 클러스터가 액세스해야 하는 사이트를 검토하고 프록시를 바이패스해야 하는지 확인합니다. 클러스터를 호스팅하는 클라우드의 클라우드 공급자 API에 대한 호출을 포함하여 기본적으로 모든 클러스터 시스템 송신 트래픽이 프록시됩니다. 시스템 전반의 프록시는 사용자 워크로드가 아닌 시스템 구성 요소에만 영향을 미칩니다. 필요한 경우 프록시를 바이패스하려면 프록시 오브젝트의 `spec.noProxy` 필드에 사이트를 추가합니다.



#### 참고

프록시 오브젝트의 `status.noProxy` 필드는 설치 구성에 있는 `networking.machineNetwork[].cidr`, `networking.clusterNetwork[].cidr`, `networking.serviceNetwork[]` 필드의 값으로 채워집니다.

Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure 및 Red Hat OpenStack Platform (RHOSP)에 설치하는 경우 Proxy 오브젝트 `status.noProxy` 필드도 인스턴스 메타데이터 끝점(169.254.169.254)로 채워집니다.

### 26.2. 클러스터 전체 프록시 사용

프록시 오브젝트는 클러스터 전체 송신 프록시를 관리하는 데 사용됩니다. 프록시를 구성하지 않고 클러스터를 설치하거나 업그레이드해도 프록시 오브젝트는 계속 생성되지만 `spec`은 `nil`이 됩니다. 예를 들면 다음과 같습니다.

```

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

```

클러스터 관리자는 이 `cluster` 프록시 오브젝트를 수정하여 OpenShift Container Platform의 프록시를 구성할 수 있습니다.





## 참고

**cluster**라는 **Proxy** 오브젝트만 지원되며 추가 프록시는 생성할 수 없습니다.

## 사전 요구 사항

- 클러스터 관리자 권한
- OpenShift Container Platform oc CLI 도구 설치

## 프로세스

1. **HTTPS** 연결을 프록시하는 데 필요한 추가 **CA** 인증서가 포함된 구성 맵을 생성합니다.



## 참고

프록시의 **ID** 인증서를 **RHCOS** 트러스트 번들에 있는 기관에서 서명한 경우 이 단계를 건너뛸 수 있습니다.

- a. 다음 내용으로 **user-ca-bundle.yaml**이라는 파일을 생성하고 **PEM** 인코딩 인증서 값을 제공합니다.

```
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

이 데이터 키의 이름은 **ca-bundle.crt**여야 합니다.

②

프록시의 **ID** 인증서 서명에 사용되는 하나 이상의 **PEM** 인코딩 **X.509** 인증서입니다.

**3**

프록시 오브젝트에서 참조할 구성 맵 이름입니다.

**4**

구성 맵은 `openshift-config` 네임스페이스에 있어야 합니다.

b.

이 파일에서 구성 맵을 생성합니다.

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

2.

`oc edit` 명령을 사용하여 `Proxy` 오브젝트를 수정합니다.

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

3.

프록시에 필요한 필드를 구성합니다.

```
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**

클러스터 외부에서 **HTTP** 연결을 구축하는 데 사용할 프록시 **URL**입니다. **URL** 스키마는 **http**여야 합니다.

**2**

클러스터 외부에서 **HTTPS** 연결을 구축하는 데 사용할 프록시 **URL**입니다. **URL** 스키마는 **http** 또는 **https** 여야 합니다. **URL** 스키마를 지원하는 프록시의 **URL**을 지정합니다. 예를 들어 대부분의 프록시는 **https** 를 사용하도록 구성된 경우 오류를 보고하지만 **http** 만 지원됩니다. 이 실패 메시지는 로그에 전파되지 않을 수 있으며 대신 네트워크 연결 실패로 표

시될 수 있습니다. 클러스터에서 **https** 연결을 수신하는 프록시를 사용하는 경우 프록시에서 사용하는 **CA** 및 인증서를 허용하도록 클러스터를 구성해야 할 수 있습니다.

3

대상 도메인 이름, 도메인, **IP** 주소 또는 프록시를 제외할 기타 네트워크 **CIDR**로 이루어진 씬표로 구분된 목록입니다.

하위 도메인과 일치하려면 도메인 앞에 **.을** 입력합니다. 예를 들어, **.y.com**은 **x.y.com**과 일치하지만 **y.com**은 일치하지 않습니다. **\***를 사용하여 모든 대상에 대해 프록시를 바이패스합니다. **networking.machineNetwork[].cidr** 필드에 의해 정의된 네트워크에 포함되어 있지 않은 작업자를 설치 구성에서 확장하려면 연결 문제를 방지하기 위해 이 목록에 해당 작업자를 추가해야 합니다.

**httpProxy**와 **httpsProxy** 필드가 모두 설정되지 않은 경우 이 필드는 무시됩니다.

4

**httpProxy** 및 **httpsProxy** 값을 상태에 쓰기 전에 준비 검사를 수행하는 데 사용할 하나 이상의 클러스터 외부 **URL**입니다.

5

**HTTPS** 연결을 프록시하는 데 필요한 추가 **CA** 인증서가 포함된 **openshift-config** 네임스페이스의 구성 맵에 대한 참조입니다. 여기서 구성 맵을 참조하기 전에 구성 맵이 이미 있어야 합니다. 프록시의 **ID** 인증서를 **RHCOS** 트러스트 번들에 있는 기관에서 서명하지 않은 경우 이 필드가 있어야 합니다.

4.

파일을 저장하여 변경 사항을 적용합니다.

### 26.3. 클러스터 전체 프록시 제거

**cluster** 프록시 오브젝트는 삭제할 수 없습니다. 클러스터에서 이 프록시를 제거하려면 프록시 오브젝트에서 모든 **spec** 필드를 제거해야 합니다.

#### 사전 요구 사항

•

클러스터 관리자 권한

- **OpenShift Container Platform oc CLI 도구 설치**

#### 프로세스

1. 프록시를 수정하려면 **oc edit** 명령을 사용합니다.

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

2. 프록시 오브젝트에서 모든 **spec** 필드를 제거합니다. 예를 들면 다음과 같습니다.

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

3. 파일을 저장하여 변경 사항을 적용합니다.

#### 추가 리소스

- [CA Bundle 인증서 교체](#)
- [프록시 인증서 사용자 지정](#)

## 27장. 사용자 정의 PKI 구성

웹 콘솔과 같은 일부 플랫폼 구성 요소에서는 통신에 경로를 사용하고, 다른 구성 요소와의 상호 작용을 위해 해당 구성 요소의 인증서를 신뢰해야 합니다. 사용자 정의 PKI(공개 키 인프라)를 사용하는 경우 개인 서명 CA 인증서가 클러스터에서 인식되도록 PKI를 구성해야 합니다.

프록시 API를 활용하면 클러스터 전체에서 신뢰하는 CA 인증서를 추가할 수 있습니다. 이 작업은 설치 중 또는 런타임에 수행해야 합니다.

- 설치 중 클러스터 전체 프록시를 구성합니다. `install-config.yaml` 파일의 `additionalTrustBundle` 설정에 개인 서명 CA 인증서를 정의해야 합니다.

설치 프로그램에서 사용자가 정의한 추가 CA 인증서가 포함된 `user-ca-bundle`이라는 `ConfigMap`을 생성합니다. 그러면 CNO(Cluster Network Operator)에서 이러한 CA 인증서를 RHCOS(Red Hat Enterprise Linux CoreOS) 신뢰 번들과 병합하는 `trusted-ca-bundle` `ConfigMap`을 생성합니다. 이 `ConfigMap`은 프록시 오브젝트의 `trustedCA` 필드에서 참조됩니다.

- 런타임에 개인 서명 CA 인증서를 포함하도록 기본 프록시 오브젝트를 수정합니다(클러스터의 프록시 사용 워크플로우의 일부). 이를 위해서는 클러스터에서 신뢰해야 하는 개인 서명 CA 인증서가 포함된 `ConfigMap`을 생성한 다음 개인 서명 인증서의 `ConfigMap`을 참조하는 `trustedCA`를 사용하여 프록시 리소스를 수정해야 합니다.

### 참고

설치 관리자 구성의 `additionalTrustBundle` 필드와 프록시 리소스의 `trustedCA` 필드는 클러스터 전체의 트러스트 번들을 관리하는 데 사용됩니다. `additionalTrustBundle`은 설치 시 사용되며, 프록시의 `trustedCA`는 런타임에 사용됩니다.

`trustedCA` 필드는 클러스터 구성 요소에서 사용하는 사용자 정의 인증서와 키 쌍이 포함된 `ConfigMap`에 대한 참조입니다.

### 27.1. 설치 중 클러스터 단위 프록시 구성

프로덕션 환경에서는 인터넷에 대한 직접 액세스를 거부하고 대신 HTTP 또는 HTTPS 프록시를 사용할 수 있습니다. `install-config.yaml` 파일에서 프록시 설정을 구성하여 프록시가 사용되도록 새 OpenShift Container Platform 클러스터를 구성할 수 있습니다.

#### 사전 요구 사항

- 기존 `install-config.yaml` 파일이 있습니다.
- 클러스터에서 액세스해야 하는 사이트를 검토하고 프록시를 바이패스해야 하는지 확인했습니다. 기본적으로 호스팅 클라우드 공급자 API에 대한 호출을 포함하여 모든 클러스터 발신 (Egress) 트래픽이 프록시됩니다. 필요한 경우 프록시를 바이패스하기 위해 Proxy 오브젝트의 `spec.noProxy` 필드에 사이트를 추가했습니다.



참고

Proxy 오브젝트의 `status.noProxy` 필드는 설치 구성에 있는 `networking.machineNetwork[].cidr`, `networking.clusterNetwork[].cidr`, `networking.serviceNetwork[]` 필드의 값으로 채워집니다.

Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure 및 Red Hat OpenStack Platform (RHOSP)에 설치하는 경우 Proxy 오브젝트 `status.noProxy` 필드도 인스턴스 메타데이터 끝점(169.254.169.254)로 채워집니다.

프로세스

1. `install-config.yaml` 파일을 편집하고 프록시 설정을 추가합니다. 예를 들면 다음과 같습니다.

```

apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: ec2.<region>.amazonaws.com,elasticloadbalancing.
<region>.amazonaws.com,s3.<region>.amazonaws.com 3
  additionalTrustBundle: | 4
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
    
```

1

클러스터 외부에서 HTTP 연결을 구축하는 데 사용할 프록시 URL입니다. URL 스키마는 `http`여야 합니다.

2

클러스터 외부에서 HTTPS 연결을 구축하는 데 사용할 프록시 URL입니다.

3

대상 도메인 이름, IP 주소 또는 프록시에서 제외할 기타 네트워크 CIDR로 이루어진 섹트로 구분된 목록입니다. 하위 도메인과 일치하려면 도메인 앞에 .을 입력합니다. 예를 들어, .y.com은 x.y.com과 일치하지만 y.com은 일치하지 않습니다. \*를 사용하여 모든 대상에 대해 프록시를 바이패스합니다. Amazon EC2, Elastic Load Balancing 및 S3 VPC 끝점을 VPC에 추가한 경우, 이러한 끝점을 noProxy 필드에 추가해야 합니다.

4

이 값을 제공하면 설치 프로그램에서 HTTPS 연결을 프록시하는 데 필요한 추가 CA 인증서가 하나 이상 포함된 openshift-config 네임스페이스에 user-ca-bundle이라는 이름으로 구성 맵을 생성합니다. 그러면 CNO(Cluster Network Operator)에서 이러한 콘텐츠를 RHCOS(Red Hat Enterprise Linux CoreOS) 신뢰 번들과 병합하는 trusted-ca-bundle 구성 맵을 생성합니다. 이 구성 맵은 Proxy 오브젝트의 trustedCA 필드에서 참조됩니다. 프록시의 ID 인증서를 RHCOS 트러스트 번들에 있는 기관에서 서명하지 않은 경우 additionalTrustBundle 필드가 있어야 합니다.



참고

설치 프로그램에서 프록시 addressEndpoints 필드를 지원하지 않습니다.

2.

파일을 저장해 놓고 OpenShift Container Platform을 설치할 때 참조하십시오.

제공되는 install-config.yaml 파일의 프록시 설정을 사용하는 cluster라는 이름의 클러스터 전체 프록시가 설치 프로그램에 의해 생성됩니다. 프록시 설정을 제공하지 않아도 cluster Proxy 오브젝트는 계속 생성되지만 spec은 nil이 됩니다.



참고

cluster라는 Proxy 오브젝트만 지원되며 추가 프록시는 생성할 수 없습니다.

## 27.2. 클러스터 전체 프록시 사용

프록시 오브젝트는 클러스터 전체 송신 프록시를 관리하는 데 사용됩니다. 프록시를 구성하지 않고 클러스터를 설치하거나 업그레이드해도 프록시 오브젝트는 계속 생성되지만 spec은 nil이 됩니다. 예를 들면 다음과 같습니다.

```
apiVersion: config.openshift.io/v1
kind: Proxy
metadata:
```

```

name: cluster
spec:
  trustedCA:
    name: ""
status:
    
```

클러스터 관리자는 이 **cluster** 프록시 오브젝트를 수정하여 **OpenShift Container Platform**의 프록시를 구성할 수 있습니다.



참고

**cluster**라는 **Proxy** 오브젝트만 지원되며 추가 프록시는 생성할 수 없습니다.

사전 요구 사항

- 클러스터 관리자 권한
- **OpenShift Container Platform oc CLI** 도구 설치

프로세스

1. **HTTPS** 연결을 프록시하는 데 필요한 추가 **CA** 인증서가 포함된 구성 맵을 생성합니다.



참고

프록시의 **ID** 인증서를 **RHCOS** 트러스트 번들에 있는 기관에서 서명한 경우 이 단계를 건너뛸 수 있습니다.

- a. 다음 내용으로 **user-ca-bundle.yaml**이라는 파일을 생성하고 **PEM** 인코딩 인증서 값을 제공합니다.

```

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

이 데이터 키의 이름은 **ca-bundle.crt**여야 합니다.

2

프록시의 ID 인증서 서명에 사용되는 하나 이상의 PEM 인코딩 X.509 인증서입니다.

3

프록시 오브젝트에서 참조할 구성 맵 이름입니다.

4

구성 맵은 **openshift-config** 네임스페이스에 있어야 합니다.

b.

이 파일에서 구성 맵을 생성합니다.

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

2.

**oc edit** 명령을 사용하여 **Proxy** 오브젝트를 수정합니다.

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

3.

프록시에 필요한 필드를 구성합니다.

```
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

클러스터 외부에서 **HTTP** 연결을 구축하는 데 사용할 프록시 **URL**입니다. **URL** 스키마는 **http**여야 합니다.

2

클러스터 외부에서 **HTTPS** 연결을 구축하는 데 사용할 프록시 **URL**입니다. **URL** 스키마는 **http** 또는 **https** 여야 합니다. **URL** 스키마를 지원하는 프록시의 **URL**을 지정합니다. 예를 들어 대부분의 프록시는 **https** 를 사용하도록 구성된 경우 오류를 보고하지만 **http** 만 지원합니다. 이 실패 메시지는 로그에 전파되지 않을 수 있으며 대신 네트워크 연결 실패로 표시될 수 있습니다. 클러스터에서 **https** 연결을 수신하는 프록시를 사용하는 경우 프록시에서 사용하는 **CA** 및 인증서를 허용하도록 클러스터를 구성해야 할 수 있습니다.

3

대상 도메인 이름, 도메인, **IP** 주소 또는 프록시를 제외할 기타 네트워크 **CIDR**로 이루어진 씬표로 구분된 목록입니다.

하위 도메인과 일치하려면 도메인 앞에 **.**을 입력합니다. 예를 들어, **.y.com**은 **x.y.com**과 일치하지만 **y.com**은 일치하지 않습니다. **\***를 사용하여 모든 대상에 대해 프록시를 바이패스합니다. **networking.machineNetwork[].cidr** 필드에 의해 정의된 네트워크에 포함되어 있지 않은 작업자를 설치 구성에서 확장하려면 연결 문제를 방지하기 위해 이 목록에 해당 작업자를 추가해야 합니다.

**httpProxy**와 **httpsProxy** 필드가 모두 설정되지 않은 경우 이 필드는 무시됩니다.

4

**httpProxy** 및 **httpsProxy** 값을 상태에 쓰기 전에 준비 검사를 수행하는 데 사용할 하나 이상의 클러스터 외부 **URL**입니다.

5

**HTTPS** 연결을 프록시하는 데 필요한 추가 **CA** 인증서가 포함된 **openshift-config** 네임스페이스의 구성 맵에 대한 참조입니다. 여기서 구성 맵을 참조하기 전에 구성 맵이 이미 있어야 합니다. 프록시의 **ID** 인증서를 **RHCOS** 트러스트 번들에 있는 기관에서 서명하지 않은 경우 이 필드가 있어야 합니다.

4.

파일을 저장하여 변경 사항을 적용합니다.

### 27.3. OPERATOR를 사용한 인증서 주입

**ConfigMap**을 통해 사용자 정의 **CA** 인증서가 클러스터에 추가되면 **CNO(Cluster Network Operator)**

에서 사용자 제공 및 시스템 CA 인증서를 단일 번들로 병합한 후 병합한 번들을 신뢰 번들 주입을 요청하는 Operator에 주입합니다.

Operator는 다음 라벨이 있는 빈 ConfigMap을 생성하여 이러한 주입을 요청합니다.

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

빈 ConfigMap의 예입니다.

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

**1**

빈 ConfigMap 이름을 지정합니다.

Operator는 이 ConfigMap을 컨테이너의 로컬 신뢰 저장소에 마운트합니다.



참고

신뢰할 수 있는 CA 인증서를 추가하는 작업은 인증서가 RHCOS(Red Hat Enterprise Linux CoreOS) 신뢰 번들에 포함되지 않은 경우에만 필요합니다.

Operator는 제한 없이 인증서를 주입할 수 있습니다. config.openshift.io/inject-trusted-cabundle=true 라벨을 사용하여 비어있는 ConfigMap이 생성되면 CNO(Cluster Network Operator)에서 모든 네임스페이스에 인증서를 주입합니다.

ConfigMap은 모든 네임스페이스에 상주할 수 있지만 사용자 정의 CA가 필요한 Pod 내의 각 컨테이너에 볼륨으로 마운트되어야 합니다. 예를 들어 다음과 같습니다.

```
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: trusted-ca
              items:
                - key: ca-bundle.crt ①
                  path: tls-ca-bundle.pem ②
```

①

*ca-bundle.crt*는 *ConfigMap* 키로 필요합니다.

②

*tls-ca-bundle.pem*은 *ConfigMap* 경로로 필요합니다.

## 28장. RHOSP의 로드 밸런싱

### 28.1. KURYR SDN으로 OCTAVIA OVN 로드 밸런서 공급자 드라이버 사용

OpenShift Container Platform 클러스터에서 Kuryr를 사용하고 나중에 RHOSP 16으로 업그레이드된 RHOSP(Red Hat OpenStack Platform) 13 클라우드에 설치된 경우, Octavia OVN 공급자 드라이버를 사용하도록 구성할 수 있습니다.



#### 중요

공급자 드라이버를 변경하면 Kuryr가 기존 로드 밸런서를 대신합니다. 이 프로세스로 인해 약간의 다운 타임이 발생합니다.

#### 사전 요구 사항

- **RHOSP CLI, openstack**을 설치합니다.
- **OpenShift Container Platform CLI, oc**를 설치합니다.
- **RHOSP의 Octavia OVN 드라이버가 활성화되었는지 확인**합니다.

#### 작은 정보

사용 가능한 Octavia 드라이버 목록을 보려면 명령줄에서 **openstack loadbalancer provider list**를 입력하십시오.

명령 출력에 **ovn** 드라이버가 표시됩니다.

#### 프로세스

**Octavia Amphora** 공급자 드라이버에서 **Octavia OVN**으로 변경하려면 다음을 수행하십시오.

1. **kuryr-config ConfigMap**을 엽니다. 명령줄에 다음을 입력합니다.

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

2.

**ConfigMap**에서 **kuryr-octavia-provider:default**가 포함된 행을 삭제합니다. 예를 들면 다음과 같습니다.

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: default 1
...
```

**1**

이 행을 삭제합니다. 클러스터에서 **ovn**을 값으로 사용하여 이 행을 다시 생성합니다.

**CNO(Cluster Network Operator)**에서 수정 사항을 감지하고 **kuryr-controller** 및 **kuryr-cni Pod**를 재배포할 때까지 기다리십시오. 이 과정에 몇 분이 걸릴 수 있습니다.

3.

**kuryr-config ConfigMap** 주석이 값 **ovn**과 함께 표시되는지 확인합니다. 명령줄에 다음을 입력합니다.

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

**ovn** 공급자 값이 출력에 표시됩니다.

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: ovn
...
```

4.

**RHOSP**에서 로드 밸런서를 다시 생성했는지 확인합니다.

a.

명령줄에 다음을 입력합니다.

```
$ openstack loadbalancer list | grep amphora
```

하나의 **Amphora** 로드 밸런서가 표시됩니다. 예를 들면 다음과 같습니다.

```
a4db683b-2b7b-4988-a582-c39daaad7981 | ostest-7mbj6-kuryr-api-loadbalancer |
84c99c906edd475ba19478a9a6690efd | 172.30.0.1 | ACTIVE | amphora
```

b.

다음을 입력하여 **ovn** 로드 밸런서를 검색합니다.

```
$ openstack loadbalancer list | grep ovn
```

**ovn** 유형의 나머지 로드 밸런서가 표시됩니다. 예를 들면 다음과 같습니다.

```
2dffe783-98ae-4048-98d0-32aa684664cc | openshift-apiserver-operator/metrics |
84c99c906edd475ba19478a9a6690efd | 172.30.167.119 | ACTIVE | ovn
0b1b2193-251f-4243-af39-2f99b29d18c5 | openshift-etcd/etcd |
84c99c906edd475ba19478a9a6690efd | 172.30.143.226 | ACTIVE | ovn
f05b07fc-01b7-4673-bd4d-adaa4391458e | openshift-dns-operator/metrics |
84c99c906edd475ba19478a9a6690efd | 172.30.152.27 | ACTIVE | ovn
```

## 28.2. OCTAVIA를 사용하여 애플리케이션 트래픽의 클러스터 확장

**RHOSP(Red Hat OpenStack Platform)**에서 실행되는 **OpenShift Container Platform** 클러스터는 **Octavia** 로드 밸런싱 서비스를 사용하여 여러 **VM(가상 머신)** 또는 유동 **IP** 주소에 트래픽을 배포할 수 있습니다. 이 기능을 사용하면 단일 머신 또는 주소가 생성하는 병목 현상이 완화됩니다.

클러스터가 **Kuryr**를 사용하는 경우 **CNO(Cluster Network Operator)**가 배포 시 내부 **Octavia** 로드 밸런서를 생성했습니다. 이 로드 밸런서를 애플리케이션 네트워크 스케일링에 사용할 수 있습니다.

클러스터가 **Kuryr**를 사용하지 않는 경우 애플리케이션 네트워크 확장에 사용할 자체 **Octavia** 로드 밸런서를 생성해야 합니다.

### 28.2.1. Octavia를 사용하여 클러스터 스케일링

여러 **API** 로드 밸런서를 사용하거나 클러스터가 **Kuryr**를 사용하지 않는 경우 **Octavia** 로드 밸런서를 생성하고 이를 사용할 클러스터를 구성합니다.

#### 사전 요구 사항

- 

**Octavia**는 **RHOSP(Red Hat OpenStack Platform)** 배포에서 사용할 수 있습니다.

#### 프로세스

1.

명령줄에서 **Amphora** 드라이버를 사용하는 **Octavia** 로드 밸런서를 생성합니다.

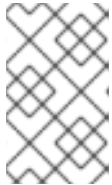
```
$ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
<id_of_worker_vms_subnet>
```

**API\_OCP\_CLUSTER** 대신 선택한 이름을 사용할 수 있습니다.

2.

로드 밸런서가 활성화된 후 리스너를 생성합니다.

```
$ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
HTTPS--protocol-port 6443 API_OCP_CLUSTER
```



참고

로드 밸런서의 상태를 보려면 **openstack loadbalancer list**를 입력합니다.

3.

라운드 로빈 알고리즘을 사용하고 세션 지속성이 활성화된 풀을 생성합니다.

```
$ 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
```

4.

컨트롤 플레인 머신을 사용할 수 있도록 하려면 상태 모니터를 생성합니다.

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --
type TCP API_OCP_CLUSTER_pool_6443
```

5.

컨트롤 플레인 머신을 로드 밸런서 풀의 멤버로 추가합니다.

```
$ 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
```

6.

선택 사항: 클러스터 API 유동 IP 주소를 재사용하려면 설정을 해제합니다.



```
$ openstack floating ip unset $API_FIP
```

7.

생성된 로드 밸런서 VIP에 설정되지 않은 API\_FIP 또는 새 주소를 추가합니다.

```
$ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value API_OCP_CLUSTER) $API_FIP
```

이제 클러스터에서 로드 밸런싱에 Octavia를 사용합니다.

참고

**Kuryr가 Octavia Amphora 드라이버를 사용하는 경우 모든 트래픽은 단일 Amphora VM(가상 머신)을 통해 라우팅됩니다.**

병목 현상을 완화할 수 있는 추가 로드 밸런서를 생성하기 위해 이 절차를 반복할 수 있습니다.

### 28.2.2. Octavia를 사용하여 Kuryr를 사용하는 클러스터 스케일링

클러스터가 Kuryr를 사용하는 경우 클러스터의 API 유동 IP 주소를 기존 Octavia 로드 밸런서와 연결합니다.

사전 요구 사항

- **OpenShift Container Platform** 클러스터는 Kuryr을 사용합니다.
- **Octavia**는 RHOSP(Red Hat OpenStack Platform) 배포에서 사용할 수 있습니다.

프로세스

1.

선택 사항: 명령줄에서 클러스터 API 유동 IP 주소를 재사용하려면 설정을 해제합니다.

```
$ openstack floating ip unset $API_FIP
```

2.

생성된 로드 밸런서 VIP에 설정되지 않은 API\_FIP 또는 새 주소를 추가합니다.

```
$ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value ${OCP_CLUSTER}-kuryr-api-loadbalancer) $API_FIP
```

이제 클러스터에서 로드 밸런싱에 **Octavia**를 사용합니다.



참고

**Kuryr**가 **Octavia Amphora** 드라이버를 사용하는 경우 모든 트래픽은 단일 **Amphora VM**(가상 머신)을 통해 라우팅됩니다.

병목 현상을 완화할 수 있는 추가 로드 밸런서를 생성하기 위해 이 절차를 반복할 수 있습니다.

### 28.3. RHOSP OCTAVIA를 사용하여 수신 트래픽 스케일링

**Octavia** 로드 밸런서를 사용하여 **Kuryr**를 사용하는 클러스터에서 **Ingress** 컨트롤러를 스케일링할 수 있습니다.

사전 요구 사항

- **OpenShift Container Platform** 클러스터는 **Kuryr**을 사용합니다.
- **RHOSP** 배포에서 **Octavia**를 사용할 수 있습니다.

프로세스

1. 현재 내부 라우터 서비스를 복사하려면 명령줄에서 다음을 입력합니다.

```
$ oc -n openshift-ingress get svc router-internal-default -o yaml > external_router.yaml
```

2. 파일 **external\_router.yaml**에서 **metadata.name** 및 **spec.type**의 값을 **LoadBalancer**로 변경합니다.

라우터 파일 예

```

apiVersion: v1
kind: Service
metadata:
  labels:
    ingresscontroller.operator.openshift.io/owning-ingresscontroller: default
  name: router-external-default 1
  namespace: openshift-ingress
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    - name: https
      port: 443
      protocol: TCP
      targetPort: https
    - name: metrics
      port: 1936
      protocol: TCP
      targetPort: 1936
  selector:
    ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
  sessionAffinity: None
  type: LoadBalancer 2

```

**1**

이 값이 **router-external-default**와 같이 설명적인지 확인합니다.

**2**

이 값이 **LoadBalancer**인지 확인합니다.



#### 참고

로드 밸런싱에 적합하지 않은 타임 스탬프 및 기타 정보를 삭제할 수 있습니다.

1.

명령줄에서 **external\_router.yaml** 파일의 서비스를 생성합니다.

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

2.

서비스의 외부 IP 주소가 로드 밸런서와 연결된 항목과 동일한지 확인합니다.

a.

명령줄에서 서비스의 외부 IP 주소를 검색합니다.

```
$ oc -n openshift-ingress get svc
```

출력 예

```
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)
AGE
router-external-default LoadBalancer 172.30.235.33 10.46.22.161
80:30112/TCP,443:32359/TCP,1936:30317/TCP 3m38s
router-internal-default ClusterIP     172.30.115.123 <none>
80/TCP,443/TCP,1936/TCP                22h
```

b.

로드 밸런서의 IP 주소를 검색합니다.

```
$ openstack loadbalancer list | grep router-external
```

출력 예

```
| 21bf6afe-b498-4a16-a958-3229e83c002c | openshift-ingress/router-external-
default | 66f3816acf1b431691b8d132cc9d793c | 172.30.235.33 | ACTIVE | octavia |
```

c.

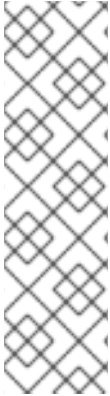
이전 단계에서 검색한 주소가 유동 IP 목록에서 서로 연결되어 있는지 확인합니다.

```
$ openstack floating ip list | grep 172.30.235.33
```

출력 예

| e2f80e97-8266-4b69-8636-e58bacf1879e | 10.46.22.161 | 172.30.235.33 | 655e7122-806a-4e0a-a104-220c6e17bda6 | a565e55a-99e7-4d15-b4df-f9d7ee8c9deb | 66f3816acf1b431691b8d132cc9d793c |

이제 **EXTERNAL-IP** 값을 새 **Ingress** 주소로 사용할 수 있습니다.



참고

**Kuryr**가 **Octavia Amphora** 드라이버를 사용하는 경우 모든 트래픽은 단일 **Amphora VM**(가상 머신)을 통해 라우팅됩니다.

병목 현상을 완화할 수 있는 추가 로드 밸런서를 생성하기 위해 이 절차를 반복할 수 있습니다.

#### 28.4. 외부 로드 밸런서 구성

기본 로드 밸런서 대신 외부 로드 밸런서를 사용하도록 **RHOSP**(Red Hat OpenStack Platform)에서 **OpenShift Container Platform** 클러스터를 구성할 수 있습니다.

사전 요구 사항

- 로드 밸런서에서 포트 **6443**, **443** 및 **80**을 통한 **TCP**는 시스템의 모든 사용자가 사용할 수 있어야 합니다.
- 각 컨트롤 플레인 노드 간에 **API** 포트 **6443**을 로드 밸런싱합니다.
- 모든 컴퓨팅 노드 간에 애플리케이션 포트 **443** 및 **80**을 로드 밸런싱합니다.
- 로드 밸런서에서 노드에 **ignition** 시작 구성을 제공하는 데 사용되는 포트 **22623**은 클러스터 외부에 노출되지 않습니다.
- 로드 밸런서는 클러스터의 모든 머신에 액세스할 수 있어야 합니다. 이러한 액세스를 허용하는 방법은 다음과 같습니다.

- 클러스터의 머신 서브넷에 로드 밸런서를 연결합니다.
- 로드 밸런서를 사용하는 머신에 유동 IP 주소를 연결합니다.



중요

외부 로드 밸런싱 서비스와 컨트롤 플레인 노드는 동일한 L2 네트워크에서 실행해야 하며 VLAN을 사용하여 로드 밸런싱 서비스와 컨트롤 플레인 노드 간에 트래픽을 라우팅할 때 동일한 VLAN에서 실행해야 합니다.

프로세스

1. 포트 6443, 443, 80의 로드 밸런서에서 클러스터에 대한 액세스를 활성화합니다.

예를 들어 이 HAProxy 구성에 유의하십시오.

샘플 HAProxy 구성 섹션

```

...
listen my-cluster-api-6443
  bind 0.0.0.0:6443
  mode tcp
  balance roundrobin
  server my-cluster-master-2 192.0.2.2:6443 check
  server my-cluster-master-0 192.0.2.3:6443 check
  server my-cluster-master-1 192.0.2.1:6443 check
listen my-cluster-apps-443
  bind 0.0.0.0:443
  mode tcp
  balance roundrobin
  server my-cluster-worker-0 192.0.2.6:443 check
  server my-cluster-worker-1 192.0.2.5:443 check
  server my-cluster-worker-2 192.0.2.4:443 check
listen my-cluster-apps-80
  bind 0.0.0.0:80
  mode tcp
  balance roundrobin
  server my-cluster-worker-0 192.0.2.7:80 check
  server my-cluster-worker-1 192.0.2.9:80 check
  server my-cluster-worker-2 192.0.2.8:80 check

```

2.

클러스터 API의 DNS 서버에 레코드를 추가하고 로드 밸런서를 통해 앱을 추가합니다. 예를 들면 다음과 같습니다.

```
<load_balancer_ip_address> api.<cluster_name>.<base_domain>
<load_balancer_ip_address> apps.<cluster_name>.<base_domain>
```

3.

명령줄에서 curl을 사용하여 외부 로드 밸런서 및 DNS 구성이 작동하는지 확인합니다.

a.

클러스터 API에 액세스할 수 있는지 확인합니다.

```
$ curl https://<loadbalancer_ip_address>:6443/version --insecure
```

구성이 올바르면 응답으로 JSON 오브젝트가 표시됩니다.

```
{
  "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.

클러스터 애플리케이션에 액세스할 수 있는지 확인합니다.



참고

웹 브라우저에서 OpenShift Container Platform 콘솔을 여는 방식으로 애플리케이션 액세스 가능성을 확인할 수도 있습니다.

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L
--insecure
```

구성이 올바르면 **HTTP** 응답이 표시됩니다.

```
HTTP/1.1 302 Found  
content-length: 0  
location: https://console-openshift-console.apps.<cluster-name>.<base domain>/  
cache-control: no-cacheHTTP/1.1 200 OK  
referrer-policy: strict-origin-when-cross-origin  
set-cookie: csrf-  
token=39HoZgztDnzjJkq/JuLJMeoKNXIfiVv2YgZc09c3TBOBU4NI6kDXaJH1LdicNh  
N1UsQWzon4Dor9GWGfopaTEQ==; 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
```



## 29장. METALLB로 로드 밸런싱

### 29.1. METALLB 및 METALLB OPERATOR 정보

클러스터 관리자는 **LoadBalancer** 유형의 서비스가 클러스터에 추가되면 **MetalLB**가 서비스에 대한 외부 IP 주소를 추가하도록 클러스터에 **MetalLB Operator**를 추가할 수 있습니다. 외부 IP 주소가 클러스터의 호스트 네트워크에 추가됩니다.

#### 29.1.1. MetalLB 사용 시기

**MetalLB**를 사용하는 것은 베어 메탈 클러스터 또는 베어 메탈과 같은 인프라가 있는 경우 중요하며, 외부 IP 주소를 통해 애플리케이션에 내결함성 액세스를 원할 때 중요합니다.

외부 IP 주소의 네트워크 트래픽이 클라이언트에서 클러스터의 호스트 네트워크로 라우팅되도록 네트워크 인프라를 구성해야 합니다.

**MetalLB Operator**를 사용하여 **MetalLB**를 배포한 후 **LoadBalancer** 유형의 서비스를 추가하면 **MetalLB**에서 플랫폼 네이티브 로드 밸런서를 제공합니다.

**layer2** 모드에서 **operations in layer2 mode**는 IP 페일오버와 유사한 메커니즘을 사용하여 장애 조치 (**failover**)를 지원합니다. 그러나 **VRRP**(가상 라우터 중복 프로토콜) 및 **keepalived**를 사용하는 대신 **gossip** 기반 프로토콜을 활용하여 노드 실패 인스턴스를 식별합니다. 페일오버가 감지되면 다른 노드에서 리더 노드의 역할을 가정하고, 비정상적인 **ARP** 메시지가 전송되어 이 변경 사항을 브로드캐스트합니다.

**layer3** 또는 **BGP**(Border Gateway Protocol) 모드에서 작동하는 **CloudEvent** 모드는 실패 감지를 네트워크에 위임합니다. **OpenShift Container Platform** 노드가 연결된 **BGP** 라우터 또는 라우터는 노드 오류를 식별하고 해당 노드에 대한 경로를 종료합니다.

**Pod** 및 서비스의고가용성을 보장하는 데 **IP** 페일오버 대신 **CloudEvent**를 사용하는 것이 좋습니다.

#### 29.1.2. MetalLB Operator 사용자 정의 리소스

**MetalLB Operator**는 자체 네임스페이스에서 다음 사용자 정의 리소스에 대해 모니터링합니다.

##### **MetalLB**

클러스터에 **MetalLB** 사용자 정의 리소스를 추가하면 **MetalLB Operator**에서 클러스터에

**MetalLB**를 배포합니다. **Operator**는 사용자 정의 리소스의 단일 인스턴스만 지원합니다. 인스턴스가 삭제되면 **Operator**는 클러스터에서 **MetalLB**를 제거합니다.

## IPAddressPool

**MetalLB**에는 **LoadBalancer** 유형의 서비스를 추가할 때 서비스에 할당할 수 있는 하나 이상의 **IP** 주소 풀이 필요합니다. **IPAddressPool**에는 **IP** 주소 목록이 포함되어 있습니다. 목록은 **1.1.1.1-1.1.1.1**과 같은 범위를 사용하여 설정된 단일 **IP** 주소, **CIDR** 표기법으로 지정된 범위, 하이픈으로 구분된 시작 및 종료 주소로 지정된 범위 또는 세 가지 조합의 조합일 수 있습니다. **IPAddressPool**에는 이름이 필요합니다. 이 문서에서는 **doc-example**, **doc-example-reserved**, **doc-example-ipv6** 등의 이름을 사용합니다. **IPAddressPool**는 풀에서 **IP** 주소를 할당합니다. **L2Advertisement** 및 **BGPAdvertisement** 사용자 정의 리소스를 사용하면 지정된 풀에서 지정된 **IP**의 알림을 받을 수 있습니다.



### 참고

단일 **IPAddressPool**은 **L2** 광고 및 **BGP** 광고에 의해 참조될 수 있습니다.

## BGPPeer

**BGP** 피어 사용자 정의 리소스는 통신할 **MetalLB**의 **BGP** 라우터, 라우터의 **AS** 번호, **MetalLB**의 **AS** 번호, 경로 알림에 대한 사용자 정의를 식별합니다. **MetalLB**에서는 서비스 로드 밸런서 **IP** 주소의 경로를 하나 이상의 **BGP** 피어에 알립니다.

## BFDProfile

**BFD** 프로파일 사용자 정의 리소스는 **BGP** 피어에 대해 **BFD**(Bidirectional Forwarding Detection)를 구성합니다. **BFD**는 **BGP**만으로 제공되는 것보다 더 빠른 경로 실패 감지 기능을 제공합니다.

## L2Advertisement

**L2Advertisement** 사용자 정의 리소스는 **L2** 프로토콜을 사용하여 **IPAddressPool**에서 들어오는 **IP**를 알립니다.

## BGPAdvertisement

**BGPAdvertisement** 사용자 정의 리소스는 **BGP** 프로토콜을 사용하여 **IPAddressPool**에서 들어오는 **IP**를 알립니다.

**MetalLB** 사용자 정의 리소스를 클러스터에 추가하고 **Operator**가 **MetalLB**를 배포하면 컨트롤러 및 발표자 **MetalLB** 소프트웨어 구성 요소가 실행을 시작합니다.

**MetalLB**는 모든 관련 사용자 정의 리소스의 유효성을 검사합니다.

### 29.1.3. MetalLB 소프트웨어 구성 요소

**MetalLB Operator**를 설치하면 **metallb-operator-controller-manager** 배포가 **Pod**를 시작합니다. **Pod**는 **Operator**의 구현입니다. **Pod**는 모든 관련 리소스에 대한 변경 사항을 모니터링합니다.

**Operator**에서 **MetalLB** 인스턴스를 시작하면 **controller** 배포 및 **speaker** 데몬 세트를 시작합니다.

#### 컨트롤러

**Operator**는 배포 및 단일 **Pod**를 시작합니다. **LoadBalancer** 유형의 서비스를 추가하면 **Kubernetes**는 **controller**를 사용하여 주소 풀에서 **IP** 주소를 할당합니다. 서비스가 실패하는 경우 컨트롤러 **Pod** 로그에 다음 항목이 있는지 확인합니다.

#### 출력 예

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

#### 발표자

**Operator**는 스피커 **Pod**에 대해 데몬 세트를 시작합니다. 기본적으로 **Pod**는 클러스터의 각 노드에서 시작됩니다. **MetalLB**를 시작할 때 **MetalLB** 사용자 정의 리소스에 노드 선택기를 지정하여 **Pod**를 특정 노드로 제한할 수 있습니다. 컨트롤러가 서비스에 **IP** 주소를 할당하고 서비스를 계속 사용할 수 없는 경우 **speaker Pod** 로그를 읽습니다. **speaker** 포드를 사용할 수 없는 경우 **oc describe pod -n** 명령을 실행합니다.

계층 2 모드의 경우 컨트롤러가 서비스에 **IP** 주소를 할당한 후 스피커 **Pod**는 알고리즘을 사용하여 로드 밸런서 **IP** 주소를 발표할 스피커 **Pod**를 결정합니다. 알고리즘에는 노드 이름과 로드 밸런서 **IP** 주소를 해시하는 작업이 포함됩니다. 자세한 내용은 "**MetalLB** 및 외부 트래픽 정책"을 참조하십시오. **speaker**는 **ARP(Address Resolution Protocol)**를 사용하여 **IPv4** 주소를 알리고 **NDP(neighbor Discovery Protocol)**를 사용하여 **IPv6** 주소를 알립니다.

**BGP(Border Gateway Protocol)** 모드의 경우 컨트롤러에서 서비스에 **IP** 주소를 할당한 후 각 발표자 **Pod**는 **BGP** 피어와 함께 로드 밸런서 **IP** 주소를 알립니다. **BGP** 피어를 사용하여 **BGP** 세션을 시작하는 노드를 구성할 수 있습니다.

로드 밸런서 **IP** 주소에 대한 요청은 **IP** 주소를 알려주는 **speaker**가 있는 노드로 라우팅됩니다. 노드가 패킷을 수신하면 서비스 프록시가 패킷을 서비스의 엔드포인트로 라우팅합니다. 최적의 경우 엔드포인트

가 동일한 노드에 있거나 다른 노드에 있을 수 있습니다. 서비스 프록시는 연결이 설정될 때마다 엔드포인트를 선택합니다.

#### 29.1.4. MetalLB 및 외부 트래픽 정책

계층 2 모드에서는 클러스터의 한 노드에서 서비스 IP 주소에 대한 모든 트래픽을 수신합니다. BGP 모드를 사용하면 호스트 네트워크의 라우터가 새 클라이언트 연결을 위해 클러스터의 노드 중 하나에 대한 연결을 엽니다. 노드가 입력된 후 클러스터에서 트래픽을 처리하는 방법은 외부 트래픽 정책의 영향을 받습니다.

##### cluster

`spec.externalTrafficPolicy`의 기본값입니다.

**cluster** 트래픽 정책을 사용하면 노드가 트래픽을 수신한 후 서비스 프록시에서 서비스의 모든 pod에 트래픽을 배포합니다. 이 정책은 pod에서 균일한 트래픽 배포를 제공하지만 클라이언트 IP 주소가 지워지고 클라이언트 대신 노드에서 트래픽이 시작된 pod의 애플리케이션에 표시될 수 있습니다.

##### 로컬

**local** 트래픽 정책에서는 노드가 트래픽을 수신한 후 서비스 프록시에서 동일한 노드의 pod에만 트래픽을 보냅니다. 예를 들어 A 노드의 **speaker Pod**에서 외부 서비스 IP를 알릴 경우 모든 트래픽이 노드 A로 전송됩니다. 트래픽이 노드 A에 진입하면 서비스 프록시는 A 노드에도 있는 서비스의 Pod에만 트래픽을 전송합니다. 추가 노드에 있는 서비스의 Pod는 A 노드에서 트래픽을 받지 않습니다. 추가 노드의 서비스에 대한 Pod는 장애 조치가 필요한 경우 복제본 역할을 합니다.

이 정책은 클라이언트 IP 주소에 영향을 미치지 않습니다. 애플리케이션 pod는 들어오는 연결에서 클라이언트 IP 주소를 확인할 수 있습니다.

## 참고

**BGP 모드에서 외부 트래픽 정책을 구성할 때 다음 정보가 중요합니다.**

**MetalLB**는 모든 적격 노드에서 로드 밸런서 **IP** 주소를 광고하지만, 노드 수를 **loadbalancing** 라우터의 용량으로 제한하여 동일한 비용 다중 경로(**ECMP**) 경로를 설정할 수 있습니다. **IP**를 알리는 노드 수가 라우터의 **ECMP** 그룹 제한보다 크면 라우터에서 **IP**를 알리는 노드보다 적은 노드를 사용합니다.

예를 들어 외부 트래픽 정책이 **local** 로 설정되고 라우터에 **ECMP** 그룹 제한이 **16**으로 설정되고 **LoadBalancer** 서비스를 구현하는 **Pod**가 **30** 노드에 배포된 경우 **14** 노드에 배포된 **Pod**가 트래픽을 수신하지 않습니다. 이 경우 서비스의 외부 트래픽 정책을 **cluster** 로 설정하는 것이 좋습니다.

## 29.1.5. 계층 2 모드의 MetalLB 개념

계층 2 모드에서 하나의 노드의 **speaker pod**는 서비스의 외부 **IP** 주소를 호스트 네트워크에 알립니다. 네트워크 화면에서 볼 때 노드에는 네트워크 인터페이스에 할당된 여러 **IP** 주소가 있는 것으로 보입니다.

## 참고

계층 2 모드는 **ARP** 및 **NDP**에 의존하기 때문에 클라이언트는 **MetalLB**가 작동하기 위해 서비스를 중단하는 노드와 동일한 서브넷에 있어야 합니다. 또한 서비스에 할당된 **IP** 주소는 클라이언트가 서비스에 도달하기 위해 사용하는 네트워크의 동일한 서브넷에 있어야 합니다.

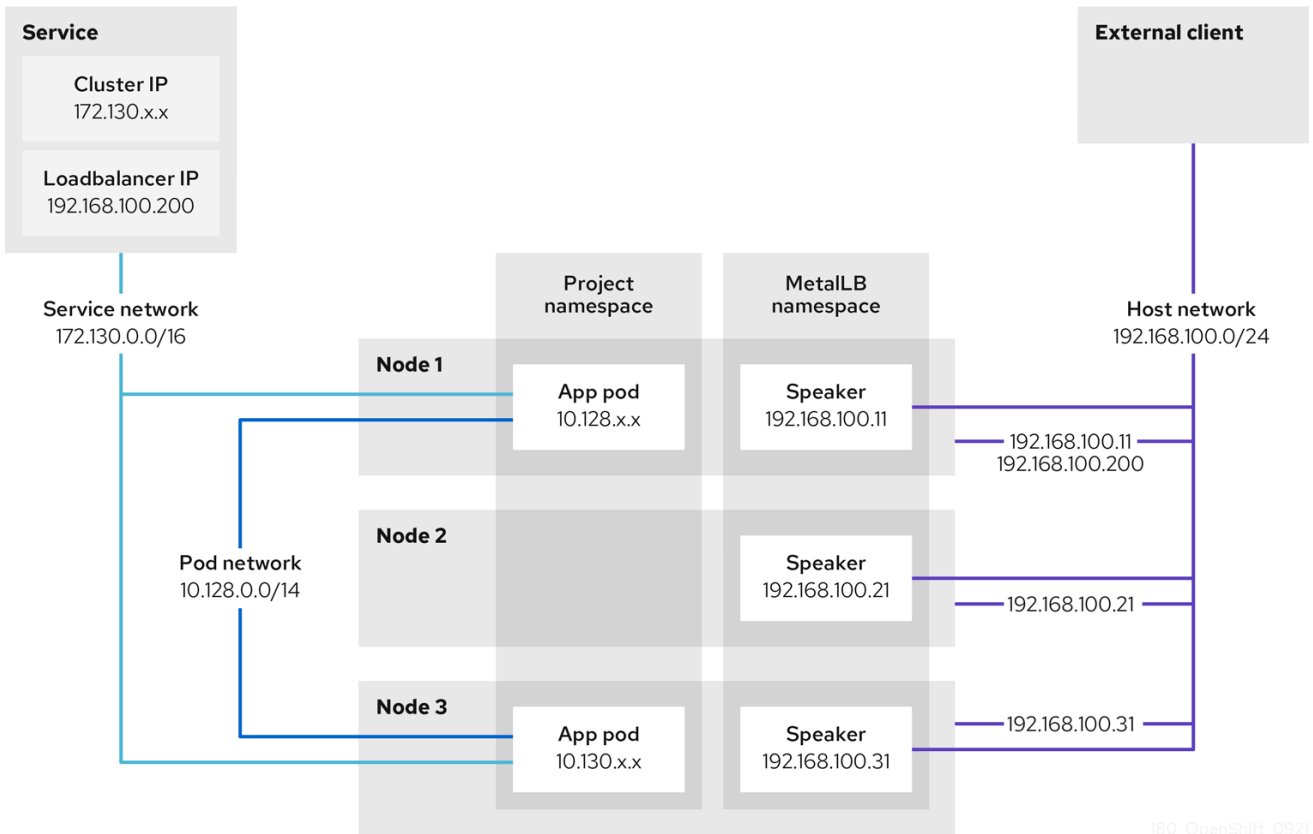
**speaker pod**는 **IPv6**에 대한 **IPv4** 서비스 및 **NDP** 요청에 대한 **ARP** 요청에 응답합니다.

계층 2 모드에서는 서비스 **IP** 주소의 모든 트래픽이 하나의 노드를 통해 라우팅됩니다. 트래픽이 노드에 진입하면 **CNI** 네트워크 공급자의 서비스 프록시에서 서비스의 모든 **Pod**에 트래픽을 배포합니다.

서비스의 모든 트래픽이 계층 2 모드에서 단일 노드를 통해 시작되기 때문에 **MetalLB**는 계층 2에 대한 로드 밸런서를 구현하지 않습니다. 대신 **MetalLB**는 **speaker pod**를 사용할 수 없게 되는 계층 2에 대한 페일오버 메커니즘을 구현하여 다른 노드의 **speaker Pod**에서 서비스 **IP** 주소를 알릴 수 있습니다.

노드를 사용할 수 없게 되면 장애 조치가 자동으로 수행됩니다. 다른 노드의 **speaker Pod**는 노드를 사용할 수 없음을 감지하고 새 **speaker Pod** 및 노드가 실패한 노드에서 서비스 **IP** 주소의 소유권을 가져옵니다.

니다.



180\_OpenShift\_0921

이전 그림에서는 **MetalLB**와 관련된 다음 개념을 보여줍니다.

- 애플리케이션은 **172.130.0.0/16** 서브넷에 클러스터 IP가 있는 서비스를 통해 사용할 수 있습니다. 이 IP 주소는 클러스터 내부에서 액세스할 수 있습니다. 서비스에는 **MetalLB**가 서비스에 할당된 외부 IP 주소 **192.168.100.200**도 있습니다.
- 노드 1 및 3에는 애플리케이션용 pod가 있습니다.
- **speaker** 데몬 세트는 각 노드에서 Pod를 실행합니다. **MetalLB Operator**는 이러한 Pod를 시작합니다.
- 각 **speaker pod**는 호스트 네트워크 포드입니다. pod의 IP 주소는 호스트 네트워크에 있는 노드의 IP 주소와 동일합니다.
- 노드 1의 **speaker pod**는 ARP를 사용하여 서비스의 외부 IP 주소 **192.168.100.200**를 알립니다. 외부 IP 주소를 발표하는 **speaker pod**는 서비스의 엔드포인트와 동일한 노드에 있어야 하며

엔드포인트는 **Ready** 상태에 있어야 합니다.

- 클라이언트 트래픽은 호스트 네트워크로 라우팅되고 **192.168.100.200 IP** 주소에 연결됩니다. 트래픽이 노드로 전환되면 서비스 프록시는 서비스에 설정한 외부 트래픽 정책에 따라 동일한 노드 또는 다른 노드의 애플리케이션 **pod**로 트래픽을 전송합니다.
  - 서비스의 외부 트래픽 정책이 **cluster** 로 설정된 경우, **speaker** 포드가 실행 중인 노드에서 **192.168.100.200** 로드 밸런서 IP 주소를 알리는 노드가 선택됩니다. 해당 노드만 서비스에 대한 트래픽을 수신할 수 있습니다.
  - 서비스의 외부 트래픽 정책이 로컬 로 설정된 경우, **speaker** 포드가 실행 중이고 서비스 끝점이 있는 노드에서 **192.168.100.200** 로드 밸런서 IP 주소를 알리는 노드가 선택됩니다. 해당 노드만 서비스에 대한 트래픽을 수신할 수 있습니다. 앞의 그래픽에서 노드 1 또는 3은 **192.168.100.200** 을 알립니다.
- 노드 1을 사용할 수 없게 되면 외부 IP 주소가 다른 노드로 장애 조치됩니다. 애플리케이션 **pod** 및 서비스 엔드포인트의 인스턴스가 있는 다른 노드에서 **speaker pod**는 외부 IP 주소 **192.168.100.200**을 알리기 시작하고 새 노드는 클라이언트 트래픽을 수신합니다. 다이어그램에서 유일한 후보는 노드 3입니다.

#### 29.1.6. BGP 모드에 대한 MetallB 개념

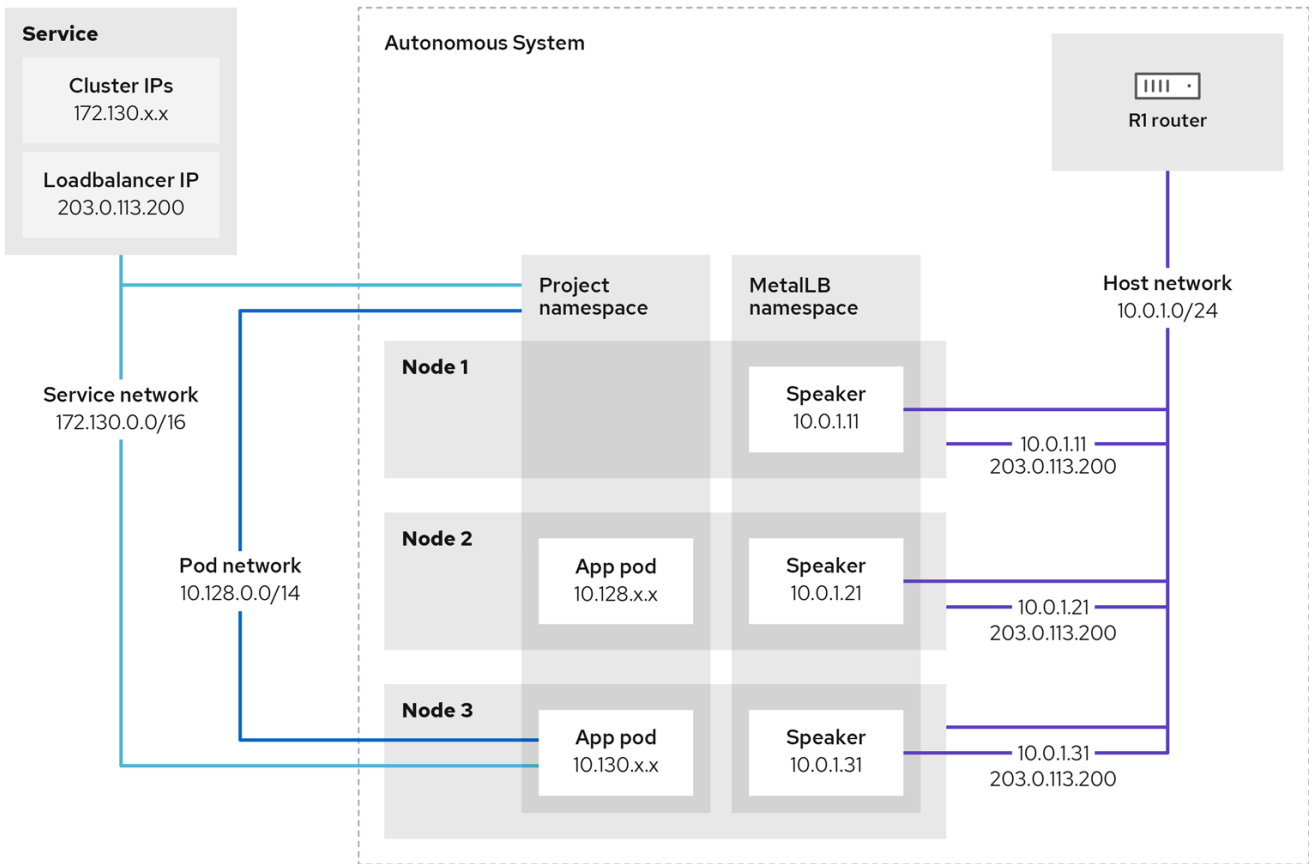
**BGP** 모드에서는 기본적으로 각 **speaker Pod**에서 서비스의 로드 밸런서 IP 주소를 각 **BGP** 피어에 알립니다. **BGP** 피어 목록을 추가하여 지정된 풀에서 들어오는 IP를 특정 피어 집합에 알릴 수도 있습니다. **BGP** 피어는 일반적으로 **BGP** 프로토콜을 사용하도록 구성된 네트워크 라우터입니다. 라우터에서 로드 밸런서 IP 주소의 트래픽을 수신하면 라우터에서 IP 주소를 알리는 스피커 **Pod**가 있는 노드 중 하나를 선택합니다. 라우터는 트래픽을 해당 노드로 보냅니다. 트래픽이 노드에 진입하면 **CNI** 네트워크 공급자의 서비스 프록시에서 서비스의 모든 **Pod**에 트래픽을 배포합니다.

클러스터 노드와 동일한 계층 2 네트워크 세그먼트에서 직접 연결된 라우터를 **BGP** 피어로 구성할 수 있습니다. 직접 연결된 라우터가 **BGP** 피어로 구성되지 않은 경우 로드 밸런서 IP 주소의 패킷이 **BGP** 피어와 스피커 **pod**를 실행하는 클러스터 노드 간에 라우팅되도록 네트워크를 구성해야 합니다.

라우터가 로드 밸런서 IP 주소의 새 트래픽을 수신할 때마다 노드에 대한 새 연결을 생성합니다. 각 라우터 제조업체에는 연결을 시작할 노드를 선택하기 위한 구별 알고리즘이 있습니다. 그러나 알고리즘은 일반적으로 네트워크 부하를 분산하기 위해 사용 가능한 노드에 트래픽을 분산하도록 설계되었습니다.

노드를 사용할 수 없게 되면 라우터에서 로드 밸런서 IP 주소를 알리는 스피커 **Pod**가 있는 다른 노드와 새 연결을 시작합니다.

그림 29.1. BGP 모드에 대한 MetalLB 토폴로지 다이어그램



209\_OpenShift\_0122

이전 그림에서는 MetalLB와 관련된 다음 개념을 보여줍니다.

- 애플리케이션은 172.130.0.0/16 서브넷에 IPv4 클러스터 IP가 있는 서비스를 통해 사용할 수 있습니다. 이 IP 주소는 클러스터 내부에서 액세스할 수 있습니다. 또한 이 서비스에는 MetalLB가 서비스에 할당된 외부 IP 주소인 203.0.113.200 이 있습니다.
- 노드 2 및 3에는 애플리케이션에 대한 포드가 있습니다.
- speaker 데몬 세트는 각 노드에서 Pod를 실행합니다. MetalLB Operator는 이러한 Pod를 시작합니다. 추출기 Pod를 실행하는 노드를 지정하도록 MetalLB를 구성할 수 있습니다.
- 각 speaker pod는 호스트 네트워크 포드입니다. pod의 IP 주소는 호스트 네트워크에 있는 노드의 IP 주소와 동일합니다.
- 각 스피커 Pod는 모든 BGP 피어와 함께 BGP 세션을 시작하고 로드 밸런서 IP 주소 또는 BGP 피어에 통합된 경로를 알립니다. 알림 Pod 는 자동 시스템 65010의 일부임을 알립니다. 이



다이어그램은 동일한 **Autonomous System** 내에서 **BGP** 피어로 라우터 **R1**을 보여줍니다. 그러나 다른 자동 시스템에 속하는 피어로 **BGP** 세션을 시작하도록 **MetalLB**를 구성할 수 있습니다.

- - 로드 밸런서 IP 주소를 알리는 스피커 포드가 있는 모든 노드는 서비스의 트래픽을 수신할 수 있습니다.
  - 서비스의 외부 트래픽 정책이 **cluster**로 설정된 경우, 발표자 포드가 실행 중인 모든 노드는 **203.0.113.200** 로드 밸런서 IP 주소와 발표 포드가 있는 모든 노드는 서비스에 대한 트래픽을 수신할 수 있습니다. 호스트 접두사는 외부 트래픽 정책이 **cluster**로 설정된 경우에만 라우터 피어에 광고됩니다.
  - 서비스의 외부 트래픽 정책이 로컬로 설정된 경우, **speaker** 포드가 실행 중이고 서비스 끝점이 실행 중인 모든 노드는 **203.0.113.200** 로드 밸런서 IP 주소를 알릴 수 있습니다. 해당 노드만 서비스에 대한 트래픽을 수신할 수 있습니다. 앞의 그래픽에서 노드 2와 3은 **203.0.113.200**을 알립니다.
- **BGP** 피어 사용자 정의 리소스를 추가할 때 노드 선택기를 지정하여 특정 **BGP** 피어로 **BGP** 세션을 시작하는 스피커 **Pod**를 제어하도록 **MetalLB**를 구성할 수 있습니다.
- **BGP**를 사용하도록 구성된 **R1**과 같은 모든 라우터를 **BGP** 피어로 설정할 수 있습니다.
- 클라이언트 트래픽은 호스트 네트워크의 노드 중 하나로 라우팅됩니다. 트래픽이 노드로 전환되면 서비스 프록시는 서비스에 설정한 외부 트래픽 정책에 따라 동일한 노드 또는 다른 노드의 애플리케이션 **pod**로 트래픽을 전송합니다.
- 노드를 사용할 수 없게 되면 라우터에서 오류를 감지하고 다른 노드와의 새 연결을 시작합니다. **BGP** 피어에 대해 **BFD(Bidirectional Forwarding Detection)** 프로토콜을 사용하도록 **MetalLB**를 구성할 수 있습니다. **BFD**는 더 빠른 링크 실패 감지 기능을 제공하므로 라우터가 **BFD**가 없는 이전의 새 연결을 시작할 수 있습니다.

### 29.1.7. 제한 사항

#### 29.1.7.1. MetalLB의 인프라 고려 사항

**MetalLB**는 기본적으로 베어 메탈 설치에 유용합니다. 이러한 설치에는 기본 로드 밸런서 기능이 포함되어 있지 않기 때문입니다. 베어 메탈 설치 외에도 일부 인프라에 **OpenShift Container Platform**을 설치할 때 기본 로드 밸런서 기능이 포함되지 않을 수 있습니다. 예를 들어 다음 인프라는 **MetalLB Operator**를 추가하는 데 도움이 될 수 있습니다.

- 베어 메탈
- VMware vSphere

**MetalLB Operator** 및 **MetalLB**는 **OpenShift SDN** 및 **OVN-Kubernetes** 네트워크 공급자에서 지원됩니다.

### 29.1.7.2. 계층 2 모드에 대한 제한 사항

#### 29.1.7.2.1. 단일 노드 성능 장애

**MetalLB**는 단일 노드를 통해 서비스에 대한 모든 트래픽을 라우팅합니다. 이 노드는 병목 현상을 일으키고 성능을 제한할 수 있습니다.

계층 2 모드는 서비스의 수신 대역폭을 단일 노드의 대역폭으로 제한합니다. 이는 **ARP** 및 **NDP**를 사용하여 트래픽을 전달하는 기본 제한 사항입니다.

#### 29.1.7.2.2. 페일오버 성능 저하

노드 간 페일오버는 클라이언트와의 협업에 따라 달라집니다. 페일오버가 발생하면 **MetalLB**에서 적절한 **ARP** 패킷을 전송하여 서비스에 연결된 **MAC** 주소가 변경되었음을 알립니다.

대부분의 클라이언트 운영 체제는 적절한 **ARP** 패킷을 올바르게 처리하고 인접 캐시를 즉시 업데이트합니다. 클라이언트에서 캐시를 빠르게 업데이트하면 몇 초 내에 페일오버가 완료됩니다. 일반적으로 클라이언트는 **10초** 내에 새 노드로 페일오버합니다. 그러나 일부 클라이언트 운영 체제는 적절한 **ARP** 패킷을 전혀 처리하지 않거나 캐시 업데이트를 지연하는 오래된 구현을 보유하고 있습니다.

**Windows**, **macOS** 및 **Linux**와 같은 일반적인 운영 체제의 최신 버전은 계층 2 페일오버를 올바르게 구현합니다. 오래되고 덜 일반적인 클라이언트 운영 체제를 제외하고는 느린 페일오버 문제가 발생하지 않습니다.

계획된 페일오버가 오래된 클라이언트에 미치는 영향을 최소화하려면 리더십 전환 후 몇 분 동안 이전 노드를 계속 실행하십시오. 이전 노드는 캐시가 새로 고쳐질 때까지 오래된 클라이언트의 트래픽을 계속 전달할 수 있습니다.

계획되지 않은 페일오버가 발생하면 오래된 클라이언트가 캐시 항목을 새로 고칠 때까지 서비스 IP에 연결할 수 없습니다.

### 29.1.7.3. BGP 모드에 대한 제한 사항

#### 29.1.7.3.1. 노드 오류가 모든 활성 연결을 중단할 수 있습니다.

**MetalLB**는 **BGP** 기반 로드 밸런싱에 일반적인 제한을 공유합니다. 노드가 실패하는 경우와 같이 **BGP** 세션이 종료되면 또는 스피커 **Pod**가 다시 시작되면 세션 종료 시 모든 활성 연결이 재설정될 수 있습니다. 최종 사용자는 피어 메시지를 통해 연결을 재설정 할 수 있습니다.

종료된 **BGP** 세션의 결과는 각 라우터 제조업체에 따라 다릅니다. 그러나 스피커 **Pod** 수의 변경이 **BGP** 세션 수에 영향을 미치고 **BGP** 피어와의 활성 연결이 중단될 것으로 예상할 수 있습니다.

서비스 중단 가능성을 방지하거나 줄이기 위해 **BGP** 피어를 추가할 때 노드 선택기를 지정할 수 있습니다. **BGP** 세션을 시작하는 노드 수를 제한하면 **BGP** 세션이 없는 노드의 오류가 서비스 연결에 영향을 미치지 않습니다.

#### 29.1.7.3.2. 하나의 ASN 및 단일 라우터 ID만 지원

**BGP** 피어 사용자 정의 리소스를 추가할 때 **spec.myASN** 필드를 지정하여 **MetalLB**가 속한 자동 시스템 번호(**ASN**)를 식별합니다. **OpenShift Container Platform**은 **MetalLB**가 단일 **ASN**에 속해야 하는 **MetalLB**와 함께 **BGP** 구현을 사용합니다. **BGP** 피어를 추가하고 **spec.myASN**에 기존 **BGP** 피어 사용자 정의 리소스와 다른 값을 지정하면 오류가 발생합니다.

마찬가지로 **BGP** 피어 사용자 정의 리소스를 추가하면 **spec.routerID** 필드는 선택 사항입니다. 이 필드의 값을 지정하는 경우 추가하는 다른 모든 **BGP** 피어 사용자 정의 리소스에 대해 동일한 값을 지정해야 합니다.

단일 **ASN** 및 단일 라우터 **ID**를 지원하기 위한 제한 사항은 **MetalLB**의 커뮤니티에서 지원하는 구현과는 다릅니다.

### 29.1.8. 추가 리소스

- [비교: 외부 IP 주소에 대한 내결함성 액세스](#)
- [IP 페일오버 제거](#)

## 29.2. METALLB OPERATOR 설치

클러스터 관리자는 **Operator**가 클러스터에서 **MetalLB** 인스턴스의 라이프사이클을 관리할 수 있도록 **MetalLB Operator**를 추가할 수 있습니다.

**MetalLB** 및 **IP** 페일오버가 호환되지 않습니다. 클러스터에 대한 **IP** 페일오버를 구성한 경우 **Operator**를 설치하기 전에 **IP** 페일오버를 제거하려면 단계를 수행합니다.

### 29.2.1. 웹 콘솔을 사용하여 OperatorHub에서 MetalLB Operator 설치

클러스터 관리자는 **OpenShift Container Platform** 웹 콘솔을 사용하여 **MetalLB Operator**를 설치할 수 있습니다.

#### 사전 요구 사항

- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. **OpenShift Container Platform** 웹 콘솔에서 **Operator** → **OperatorHub**로 이동합니다.
2. **MetalLB Operator**를 검색한 다음 설치를 클릭합니다.
3. **Operator** 설치 페이지에서 기본값을 승인하고 설치를 클릭합니다.

#### 검증

1. 설치에 성공했는지 확인하려면 다음을 수행하십시오.
  - a. **Operator** → 설치된 **Operator** 페이지로 이동합니다.
  - b. **Operator**가 **openshift-operators** 네임스페이스에 설치되어 있고 해당 상태가 **Succeeded** 인지 확인합니다.
2. **Operator**가 성공적으로 설치되지 않은 경우 **Operator**의 상태를 확인하고 로그를 확인합니다.

- a. **Operator** → 설치된 **Operator** 페이지로 이동하여 **Status** 열에 오류 또는 실패가 있는지 점검합니다.
- b. 워크로드 → **Pod** 페이지로 이동하여 **openshift-operators** 프로젝트에서 문제를 보고하는 **Pod**의 로그를 확인합니다.

### 29.2.2. CLI를 사용하여 OperatorHub에서 설치

**OpenShift Container Platform** 웹 콘솔을 사용하는 대신 **CLI**를 사용하여 **OperatorHub**에서 **Operator**를 설치할 수 있습니다. **oc**)를 사용하여 **MetalLB Operator**를 설치할 수 있습니다.

**CLI**를 사용하는 경우 **metallb-system** 네임스페이스에 **Operator**를 설치하는 것이 좋습니다.

#### 사전 요구 사항

- 클러스터가 베어 메탈 하드웨어에 설치되어 있어야 합니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. 다음 명령을 입력하여 **MetalLB Operator**의 네임스페이스를 생성합니다.

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: metallb-system
EOF
```

2. 네임스페이스에서 **Operator group CR**(사용자 정의 리소스)을 생성합니다.

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
```

```

metadata:
  name: metallb-operator
  namespace: metallb-system
EOF

```

3.

**Operator group**이 네임스페이스에 설치되어 있는지 확인합니다.

```

$ oc get operatorgroup -n metallb-system

```

출력 예

```

NAME          AGE
metallb-operator 14m

```

4.

서브스크립션 **CR**을 생성합니다.

a.

**Subscription CR**을 정의하고 **YAML** 파일을 저장합니다(예: **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**

**redhat-operators** 값을 지정해야 합니다.

b.

**Subscription CR**을 생성하려면 다음 명령을 실행합니다.

```

$ oc create -f metallb-sub.yaml

```

5.

선택 사항: Prometheus에 BGP 및 BFD 메트릭이 표시되도록 하려면 다음 명령과 같이 네임스페이스에 레이블을 지정할 수 있습니다.

```
$ oc label ns metallb-system "openshift.io/cluster-monitoring=true"
```

## 검증

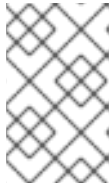
검증 단계에서는 MetalLB Operator가 metallb-system 네임스페이스에 설치되어 있다고 가정합니다.

1. 설치 계획이 네임스페이스에 있는지 확인합니다.

```
$ oc get installplan -n metallb-system
```

출력 예

| NAME          | CSV                                  | APPROVAL  | APPROVED |
|---------------|--------------------------------------|-----------|----------|
| install-wzg94 | metallb-operator.4.11.0-nnnnnnnnnnnn | Automatic | true     |



참고

Operator를 설치하는 데 몇 초가 걸릴 수 있습니다.

2. Operator가 설치되었는지 확인하려면 다음 명령을 입력합니다.

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

출력 예

| Name                                 | Phase     |
|--------------------------------------|-----------|
| metallb-operator.4.11.0-nnnnnnnnnnnn | Succeeded |

### 29.2.3. 클러스터에서 MetalLB 시작

**Operator**를 설치한 후 **MetalLB** 사용자 정의 리소스의 단일 인스턴스를 구성해야 합니다. 사용자 정의 리소스를 구성한 후 **Operator**는 클러스터에서 **MetalLB**를 시작합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- **MetalLB Operator**를 설치합니다.

#### 절차

이 절차에서는 **MetalLB Operator**가 **metallb-system** 네임스페이스에 설치되어 있다고 가정합니다. 웹 콘솔을 사용하여 설치한 경우 네임스페이스의 **openshift-operators** 를 대체합니다.

1. **MetalLB** 사용자 지정 리소스의 단일 인스턴스를 생성합니다.

```
$ cat << EOF | oc apply -f -
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
EOF
```

#### 검증

**MetalLB** 컨트롤러 및 **MetalLB** 발표자의 데몬 세트가 실행 중인지 확인합니다.

1. 컨트롤러의 배포가 실행 중인지 확인합니다.

```
$ oc get deployment -n metallb-system controller
```



출력 예

```
NAME      READY UP-TO-DATE AVAILABLE AGE
controller 1/1    1          1         11m
```

2.

**speaker**의 데몬 세트가 실행 중인지 확인합니다.

```
$ oc get daemonset -n metallb-system speaker
```

출력 예

```
NAME      DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE
SELECTOR          AGE
speaker 6      6      6      6      6      kubernetes.io/os=linux 18m
```

예제 출력은 6개의 발표자 pod를 나타냅니다. 클러스터의 발표자 Pod 수는 예제 출력과 다를 수 있습니다. 출력에 클러스터의 각 노드에 하나의 pod가 표시되는지 확인합니다.

### 29.2.3.1. 스피커 Pod를 특정 노드로 제한

기본적으로 **MetalLB Operator**를 사용하여 **MetalLB**를 시작하면 **Operator**는 클러스터의 각 노드에서 사용자 Pod 인스턴스를 시작합니다. 스피커 Pod가 있는 노드만 로드 밸런서 IP 주소를 알릴 수 있습니다. 노드 선택기로 **MetalLB** 사용자 정의 리소스를 구성하여 사용 중인 Pod를 실행하는 노드를 지정할 수 있습니다.

스피커 Pod를 특정 노드로 제한하는 가장 일반적인 이유는 특정 네트워크에서 네트워크 인터페이스가 있는 노드만 로드 밸런서 IP 주소를 알리도록 하는 것입니다. 실행 중인 스피커 Pod가 있는 노드만 로드 밸런서 IP 주소의 대상으로 광고됩니다.

스피커 Pod를 특정 노드로 제한하고 서비스의 외부 트래픽 정책에 대해 **local** 을 지정하는 경우 서비스의 애플리케이션 Pod가 동일한 노드에 배포되었는지 확인해야 합니다.

스피커 **Pod**를 작업자 노드로 제한하는 구성의 예

```

apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  nodeSelector: <.>
    node-role.kubernetes.io/worker: ""
  speakerTolerations: <.>
    - key: "Example"
      operator: "Exists"
      effect: "NoExecute"

```

<.> 예제 구성은 발표자 **Pod**를 작업자 노드에 할당하도록 지정하지만, 노드 또는 유효한 노드 선택기에 할당한 라벨을 지정할 수 있습니다. 이 예제 구성에서 **Operator**를 사용하여 키 값과 **effect** 값과 일치하는 테인트를 허용하도록 이 허용 오차가 연결된 **Pod**를 허용할 수 있습니다.

**spec.nodeSelector** 필드를 사용하여 매니페스트를 적용한 후 **oc get daemonset -n metallb-system speaker** 명령을 사용하여 **Operator**가 배포한 **Pod** 수를 확인할 수 있습니다. 마찬가지로 **oc get nodes -l node-role.kubernetes.io/worker=** 과 같은 명령을 사용하여 레이블과 일치하는 노드를 표시할 수 있습니다.

선택 옵션으로 노드에서 선호도 규칙을 사용하여 예약해야 하는 발표자 **Pod**를 제어하도록 허용할 수 있습니다. 허용 오차 목록을 적용하여 이러한 **Pod**를 제한할 수도 있습니다. 유사성 규칙, 테인트 및 허용 오차에 대한 자세한 내용은 추가 리소스를 참조하십시오.

#### 29.2.4. 추가 리소스

- [노드 선택기를 사용하여 특정 노드에 Pod 배치](#)
- [테인트 \(Taints\) 및 톨러레이션\(Tolerations\)의 이해](#)

#### 29.2.5. 다음 단계

- **MetalLB 주소 풀 구성**

### 29.3. METALLB OPERATOR 업그레이드

자동 업그레이드 절차는 **OpenShift Container Platform 4.10** 및 이전 버전에서 예상대로 작동하지 않습니다. 업그레이드 절차의 요약은 다음과 같습니다.

1. 이전에 설치한 **Operator** 버전 (예: **4.10**)을 삭제합니다. 네임스페이스 및 **metallb** 사용자 정의 리소스가 제거되지 않았는지 확인합니다.
2. **CLI**를 사용하여 **Operator**의 **4.11** 버전을 설치합니다. 이전에 설치한 **Operator** 버전이 설치된 것과 동일한 네임스페이스에 **Operator**의 **4.11** 버전을 설치합니다.



#### 참고

이 절차는 표준 간단한 방법을 따르는 **MetalLB Operator**의 자동 **z-stream** 업데이트에 적용되지 않습니다.

**Operator**를 업그레이드하는 자세한 단계는 다음 지침을 참조하십시오.

#### 29.3.1. 웹 콘솔을 사용하여 클러스터에서 MetalLB Operator 삭제

클러스터 관리자는 웹 콘솔을 사용하여 선택한 네임스페이스에서 설치된 **Operator**를 삭제할 수 있습니다.

#### 사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터 웹 콘솔에 액세스할 수 있습니다.

#### 절차

1. **Operator** → 설치된 **Operator** 페이지로 이동합니다.

2.

**MetalLB Operator**를 검색합니다. 그런 다음 해당 **Operator**를 클릭합니다.

3.

**Operator** 상세 정보 페이지 오른쪽에 있는 작업 드롭다운 메뉴에서 **Operator** 제거를 선택합니다.

**Operator**를 설치 제거하시겠습니까? 대화 상자가 표시됩니다.

4.

설치 제거를 선택하여 **Operator**, **Operator** 배포 및 **Pod**를 제거합니다. 이 작업 후에 **Operator**는 실행을 중지하고 더 이상 업데이트가 수신되지 않습니다.



참고

이 작업은 **CRD**(사용자 정의 리소스 정의) 및 **CR**(사용자 정의 리소스)을 포함하여 **Operator**에서 관리하는 리소스를 제거하지 않습니다. 웹 콘솔에서 활성화된 대시보드 및 탐색 항목과 계속 실행되는 클러스터 외부 리소스는 수동 정리가 필요할 수 있습니다. **Operator**를 설치 제거한 후 해당 항목을 제거하려면 **Operator CRD**를 수동으로 삭제해야 할 수 있습니다.

29.3.2. CLI를 사용하여 클러스터에서 **MetalLB Operator** 삭제

클러스터 관리자는 **CLI**를 사용하여 선택한 네임스페이스에서 설치된 **Operator**를 삭제할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.
- **oc** 명령이 워크스테이션에 설치되어 있습니다.

절차

1.

**currentCSV** 필드에서 구독된 **MetalLB Operator**의 현재 버전을 확인합니다.

```
$ oc get subscription metallb-operator -n metallb-system -o yaml | grep currentCSV
```

출력 예

```
currentCSV: metallb-operator.4.10.0-202207051316
```

2.

서브스크립션을 삭제합니다.

```
$ oc delete subscription metallb-operator -n metallb-system
```

출력 예

```
subscription.operators.coreos.com "metallb-operator" deleted
```

3.

이전 단계의 **currentCSV** 값을 사용하여 대상 네임스페이스에서 **Operator**의 **CSV**를 삭제합니다.

```
$ oc delete clusterserviceversion metallb-operator.4.10.0-202207051316 -n metallb-system
```

출력 예

```
clusterserviceversion.operators.coreos.com "metallb-operator.4.10.0-202207051316" deleted
```

### 29.3.3. MetalLB Operator 업그레이드

사전 요구 사항

- **cluster-admin** 역할을 가진 사용자로 클러스터에 액세스합니다.

절차

1. **metallb-system** 네임스페이스가 여전히 존재하는지 확인합니다.

```
$ oc get namespaces | grep metallb-system
```

출력 예

```
metallb-system          Active 31m
```

2. **metallb** 사용자 정의 리소스가 여전히 있는지 확인합니다.

```
$ oc get metallb -n metallb-system
```

출력 예

```
NAME    AGE
metallb 33m
```

3. "CLI를 사용하여 OperatorHub에서 설치"의 지침에 따라 **MetalLB Operator**의 최신 4.11 버전을 설치합니다.



참고

**MetalLB Operator**의 최신 4.11 버전을 설치할 때 이전에 설치된 동일한 네임스페이스에 **Operator**를 설치해야 합니다.

4. 업그레이드된 **Operator** 버전이 이제 4.11 버전인지 확인합니다.

```
$ oc get csv -n metallb-system
```

출력 예

| NAME                                 | DISPLAY          | VERSION             | REPLACES | PHASE     |
|--------------------------------------|------------------|---------------------|----------|-----------|
| metallb-operator.4.11.0-202207051316 | MetallB Operator | 4.11.0-202207051316 |          | Succeeded |

#### 29.3.4. 추가 리소스

- [클러스터에서 Operator 삭제](#)
- [MetalLB Operator 설치](#)

### 29.4. METALLB 주소 풀 구성

클러스터 관리자는 주소 풀을 추가, 수정, 삭제할 수 있습니다. MetalLB Operator는 주소 풀 사용자 정의 리소스를 사용하여 MetalLB에서 서비스에 할당할 수 있는 IP 주소를 설정합니다. 예제에서 사용되는 네임스페이스는 네임스페이스가 metallb-system 이라고 가정합니다.

#### 29.4.1. IPAddressPool 사용자 정의 리소스 정보



참고

OpenShift Container Platform 4.10의 "MetalLB"에 설명된 주소 풀 CRD(사용자 정의 리소스 정의) 및 API는 4.11에서 계속 사용할 수 있습니다. 그러나 주소 풀 CRD를 사용하는 경우 계층 2 또는 BGP 프로토콜을 사용하여 IPAddressPools 와 관련된 향상된 기능은 지원되지 않습니다.

IPAddressPool 사용자 정의 리소스의 필드는 다음 표에 설명되어 있습니다.

표 29.1. MetalLB IPAddressPool 풀 사용자 정의 리소스

| 필드 | 유형 | 설명 |
|----|----|----|
|----|----|----|

| 필드                               | 유형      | 설명                                                                                                                                                                         |
|----------------------------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>metadata.name</code>       | string  | 주소 풀의 이름을 지정합니다. 서비스를 추가할 때 <b>metallb.universe.tf/address-pool</b> 주석에 이 풀 이름을 지정하여 특정 풀에서 IP 주소를 선택할 수 있습니다. 문서 전체에서 <b>doc-example, silver, gold</b> 라는 이름이 사용됩니다.      |
| <code>metadata.name space</code> | string  | 주소 풀의 네임스페이스를 지정합니다. MetalLB Operator에서 사용하는 동일한 네임스페이스를 지정합니다.                                                                                                            |
| <code>metadata.label</code>      | string  | 선택 사항: <b>IPAddressPool</b> 에 할당된 키 값 쌍을 지정합니다. 이는 <b>BGPAdvertisement</b> 및 <b>L2Advertisement</b> CRD의 <b>ipAddressPools</b> 에서 참조하여 <b>IPAddressPool</b> 을 연관시킬 수 있습니다. |
| <code>spec.addresses</code>      | string  | 서비스에 할당할 MetalLB Operator의 IP 주소 목록을 지정합니다. 단일 풀에 여러 범위를 지정할 수 있습니다. 모두 동일한 설정을 공유합니다. CIDR 표기법에서 각 범위를 지정하거나 하이픈으로 구분된 시작 및 끝 IP 주소로 지정합니다.                               |
| <code>spec.autoAssign</code>     | boolean | 선택 사항: MetalLB에서 이 풀에서 IP 주소를 자동으로 할당하는지 여부를 지정합니다. <b>metallb.universe.tf/address-pool</b> 주석을 사용하여 이 풀에서 IP 주소를 명시적으로 요청하려면 <b>false</b> 를 지정합니다. 기본값은 <b>true</b> 입니다.  |

### 29.4.2. 주소 풀 구성

클러스터 관리자는 주소 풀을 클러스터에 추가하여 **MetalLB**에서 로드 밸런서 서비스에 할당할 수 있는 IP 주소를 제어할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. 다음 예와 같은 콘텐츠를 사용하여 **ipaddresspool.yaml** 파일과 같은 파일을 생성합니다.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
```



```

name: doc-example
labels: ❶
  zone: east
spec:
  addresses:
  - 203.0.113.1-203.0.113.10
  - 203.0.113.65-203.0.113.75

```

❶

*IPAddressPool*에 할당된 이 레이블은 *BGPAdvertisement CRD*의 *ipAddressPoolSelectors*에서 참조하여 *IPAddressPool*을 광고와 연결할 수 있습니다.

2.

*IP* 주소 풀에 대한 구성을 적용합니다.

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

검증

•

주소 풀을 확인합니다.

```
$ oc describe -n metallb-system IPAddressPool doc-example
```

출력 예

```

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>

```

주소 풀 이름(예: **doc-example**) 및 IP 주소 범위가 출력에 표시되는지 확인합니다.

### 29.4.3. 주소 풀 구성의 예

#### 29.4.3.1. 예: IPv4 및 CIDR 범위

CIDR 표기법에서 IP 주소 범위를 지정할 수 있습니다. 하이픈을 사용하는 표기법과 CIDR 표기법을 결합하여 하한 및 상한을 분리할 수 있습니다.

```
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
```

#### 29.4.3.2. 예: IP 주소

MetallLB가 풀에서 IP 주소를 자동으로 할당하지 못하도록 **autoAssign** 필드를 **false** 로 설정할 수 있습니다. 서비스를 추가할 때 풀에서 특정 IP 주소를 요청하거나 주석에 풀 이름을 지정하여 풀에서 IP 주소를 요청할 수 있습니다.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-reserved
  namespace: metallb-system
spec:
  addresses:
    - 10.0.100.0/28
  autoAssign: false
```

#### 29.4.3.3. 예: IPv4 및 IPv6 주소

IPv4 및 IPv6을 사용하는 주소 풀을 추가할 수 있습니다. 여러 IPv4 예와 마찬가지로 **addresses** 목록에서 여러 범위를 지정할 수 있습니다.

서비스에 단일 IPv4 주소, 단일 IPv6 주소 또는 두 서비스 추가 방법에 따라 결정됩니다. **spec.ipFamilies** 및 **spec.ipFamilyPolicy** 필드는 서비스에 IP 주소를 할당하는 방법을 제어합니다.

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-combined
  namespace: metallb-system
spec:
  addresses:
    - 10.0.100.0/28
    - 2002:2:2::1-2002:2:2::100

```

#### 29.4.4. 추가 리소스

- **L2 알립 및 라벨을 사용하여 MetallB 구성.**

#### 29.4.5. 다음 단계

- **BGP 모드의 경우 MetallB BGP 피어 구성을 참조하십시오.**
- **MetallB를 사용하도록 서비스 구성.**

### 29.5. IP 주소 풀 광고 정보

계층 2 프로토콜, BGP 프로토콜 또는 둘 다로 IP 주소가 광고되도록 MetallB를 구성할 수 있습니다. 계층 2의 경우 MetallB는 내결함성 외부 IP 주소를 제공합니다. BGP를 사용하면 MetallB가 외부 IP 주소 및 로드 밸런싱을 위한 내결함성을 제공합니다.

MetallB는 동일한 IP 주소 세트에 대해 L2 및 BGP를 사용한 광고를 지원합니다.

MetallB는 주소 풀을 특정 BGP 피어에 효과적으로 네트워크 노드의 하위 집합에 할당할 수 있는 유연성을 제공합니다. 이렇게 하면 더 복잡한 구성을 사용할 수 있습니다(예: 노드 격리 또는 네트워크 분할).

#### 29.5.1. BGPAdvertisement 사용자 정의 리소스 정보

BGPAdvertisements 오브젝트의 필드는 다음 표에 정의되어 있습니다.

#### 표 29.2. BGPAdvertisements 구성

| 필드                                        | 유형      | 설명                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>metadata.name</code>                | string  | BGP 광고의 이름을 지정합니다.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>metadata.name space</code>          | string  | BGP 광고의 네임스페이스를 지정합니다. MetalLB Operator에서 사용하는 동일한 네임스페이스를 지정합니다.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>spec.agggregationLength</code>      | integer | 선택 사항: 32비트 CIDR 마스크에 포함할 비트 수를 지정합니다. 스피커가 BGP 피어에 알리는 경로를 집계하기 위해 마스크는 여러 서비스 IP 주소의 경로에 적용되며 스피커는 집계된 경로를 알립니다. 예를 들어 집계 길이가 <b>24</b> 인 경우 사용자는 여러 <b>10.0.1.x/32</b> 서비스 IP 주소를 집계하고 하나의 <b>10.0.1.0/24</b> 경로를 알릴 수 있습니다.                                                                                                                                                                                                                                                                                                                |
| <code>spec.agggregationLengthV6</code>    | integer | 선택 사항: 128비트 CIDR 마스크에 포함할 비트 수를 지정합니다. 예를 들어 집계 길이가 <b>124</b> 인 경우 사용자는 여러 <b>fc00:f853:0ccd:0ccd:e799::x/128</b> 서비스 IP 주소를 집계하고 단일 <b>fc00:f853:0ccd:0ccd:e799::0124</b> 경로를 알릴 수 있습니다.                                                                                                                                                                                                                                                                                                                                                    |
| <code>spec.communities</code>             | string  | <p>선택 사항: 하나 이상의 BGP 커뮤니티를 지정합니다. 각 커뮤니티는 콜론 문자로 구분된 두 개의 16비트 값으로 지정됩니다. 잘 알려진 커뮤니티는 16비트 값으로 지정해야 합니다.</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;">  <div> <p><b>참고</b></p> <p>문자열과 함께 생성된 커뮤니티 오브젝트도 사용할 수 있습니다.</p> </div> </div> |
| <code>spec.localPref</code>               | integer | 선택 사항: 이 광고의 로컬 기본 설정을 지정합니다. 이 BGP 속성은 자동 시스템 내의 BGP 세션에 적용됩니다.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <code>spec.ipAddress Pools</code>         | string  | 선택 사항: 이 광고와 광고할 <b>IPAddressPools</b> 목록입니다.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>spec.ipAddress PoolSelectors</code> | string  | 선택 사항: 이 광고와 함께 광고되는 <b>IPAddressPool</b> 의 선택기입니다. 이는 이름 자체 대신 <b>IPAddressPool</b> 에 할당된 레이블을 기반으로 <b>IPAddressPool</b> 에 <b>IPAddressPool</b> 을 연결하는 것입니다. 이 또는 목록에 의해 <b>IPAddressPool</b> 을 선택하지 않은 경우, 광고는 모든 <b>IPAddressPools</b> 에 적용됩니다.                                                                                                                                                                                                                                                                                             |

| 필드                        | 유형            | 설명                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>spec.nodeSelectors</b> | <b>string</b> | <p>선택 사항: <b>NodeSelectors</b> 를 사용하면 노드가 로드 밸런서 IP의 다음 홉으로 발표되도록 제한할 수 있습니다. 비어있는 경우 모든 노드가 다음 홉으로 발표됩니다.</p>  <p><b>참고</b></p> <p>이 기능이 지원하는 기능은 기술 프리뷰 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다.</p> |
| <b>spec.peers</b>         | <b>string</b> | <p>선택 사항: 피어는 BGP 피어를 제한하여 선택한 풀의 IP를 로 알립니다. 비어있는 경우 로드 밸런서 IP는 구성된 모든 BGP 피어에 공개됩니다.</p>                                                                                                                                                                                                                                                                     |

### 29.5.2. BGP 광고를 사용하여 MetalLB 구성 및 기본 사용 사례

피어 BGP 라우터가 하나의 203.0.113.200/32 경로를 수신하고 MetalLB가 서비스에 할당하는 각 로드 밸런서 IP 주소에 대한 fc00:f853:ccd:e799::1/128 경로를 다음과 같이 구성하려면 다음과 같이 MetalLB를 구성합니다. localPref 및 커뮤니티 필드가 지정되지 않으므로 localPref가 0으로 설정되고 BGP 커뮤니티없이 경로가 광고됩니다.

#### 29.5.2.1. 예: BGP를 사용하여 기본 주소 풀 구성 알람

**IPAddressPool** 이 BGP 프로토콜을 통해 알리도록 다음과 같이 MetalLB를 구성합니다.

##### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

##### 절차

1.

**IP 주소 풀을 만듭니다.**

a.

다음 예와 같은 콘텐츠를 사용하여 `ipaddresspool.yaml` 파일과 같은 파일을 생성합니다.

```
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.

IP 주소 풀에 대한 구성을 적용합니다.

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

2.

**BGP 광고를 생성합니다.**

a.

다음 예와 같은 콘텐츠를 사용하여 `bgpadvertisement.yaml` 과 같은 파일을 만듭니다.

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-basic
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-basic
```

b.

설정을 적용합니다.

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

### 29.5.3. BGP 광고를 사용하여 MetalLB 구성 및 고급 사용 사례

MetalLB가 203.0.113.200 및 203.0.113.203 사이의 범위에서 로드 밸런서 서비스에 IP 주소를 할당하고 `fc00:f853:ccd:e799::0` 및 `fc00:f853:ccd:e00:f853:ccd:e799::f.f.f` 를 구성하도록 MetalLB를 구성합니다.

두 개의 BGP 알립을 설명하려면 MetalLB가 203.0.113.200의 IP 주소를 서비스에 할당할 때 인스턴스를 고려하십시오. 그 IP 주소를 예로 들면, 스피커는 BGP 피어에 두 개의 경로를 알립니다.

- **203.0.113.200/32, localPref** 를 100으로 설정하고 커뮤니티가 **NO\_ADVERTISE** 커뮤니티의 숫자 값으로 설정합니다. 이 사양은 이 경로를 사용할 수 있는 피어 라우터에 표시되지만 이 경로에 대한 정보를 BGP 피어에 전파해서는 안 됩니다.
- **203.0.113.200/30. MetalLB**에서 할당한 로드 밸런서 IP 주소를 단일 경로로 집계합니다. MetalLB에서는 커뮤니티 속성이 8000:800으로 설정된 BGP 피어에 집계된 경로를 알립니다. BGP 피어는 203.0.113.200/30 경로를 다른 BGP 피어에 전파합니다. 트래픽이 스피커가 있는 노드로 라우팅되면 203.0.113.200/32 경로는 트래픽을 클러스터로 전달하고 서비스와 관련된 Pod로 이동합니다.

더 많은 서비스를 추가하면 MetalLB가 풀에서 더 많은 로드 밸런서 IP 주소를 할당하므로 피어 라우터는 각 서비스에 대해 하나의 로컬 경로, 203.0.113.20x/32 및 203.0.113.200/30 집계 경로를 수신합니다. 추가하는 각 서비스는 /30 경로를 생성하지만 MetalLB는 피어 라우터와 통신하기 전에 경로를 하나의 BGP 광고로 분할합니다.

### 29.5.3.1. 예: BGP를 사용하여 고급 주소 풀 구성 알립

IPAddressPool이 BGP 프로토콜을 통해 알리도록 다음과 같이 MetalLB를 구성합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. **IP 주소 풀을 만듭니다.**
  - a. 다음 예와 같은 콘텐츠를 사용하여 `ipaddresspool.yaml` 파일과 같은 파일을 생성합니다.

```
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. *IP 주소 풀에 대한 구성을 적용합니다.*

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

2. *BGP 광고를 생성합니다.*

- a. *다음 예제와 같은 콘텐츠를 사용하여 `bgpadvertisement1.yaml` 과 같은 파일을 만듭니다.*

```

apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 65535:65282
  aggregationLength: 32
  localPref: 100

```

- b. *설정을 적용합니다.*

```
$ oc apply -f bgpadvertisement1.yaml
```

- c. *다음 예제와 같은 콘텐츠를 사용하여 `bgpadvertisement2.yaml` 과 같은 파일을 만듭니다.*

```

apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-2
  namespace: metallb-system
spec:

```



```

ipAddressPools:
  - doc-example-bgp-adv
communities:
  - 8000:800
aggregationLength: 30
aggregationLengthV6: 124

```

d.

설정을 적용합니다.

```
$ oc apply -f bgpadvertisement2.yaml
```

#### 29.5.4. L2Advertisement 사용자 정의 리소스 정보

**L2Advertisements** 오브젝트의 필드는 다음 표에 정의되어 있습니다.

표 29.3. L2 알람 구성

| 필드                                  | 유형            | 설명                                                                                                                                                                                                                                                 |
|-------------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>metadata.name</b>                | <b>string</b> | L2 광고의 이름을 지정합니다.                                                                                                                                                                                                                                  |
| <b>metadata.name space</b>          | <b>string</b> | L2 광고의 네임스페이스를 지정합니다. MetalLB Operator에서 사용하는 동일한 네임스페이스를 지정합니다.                                                                                                                                                                                   |
| <b>spec.ipAddress Pools</b>         | <b>string</b> | 선택 사항: 이 광고와 광고할 <b>IPAddressPools</b> 목록입니다.                                                                                                                                                                                                      |
| <b>spec.ipAddress PoolSelectors</b> | <b>string</b> | 선택 사항: 이 광고와 함께 광고되는 <b>IPAddressPool</b> 의 선택기입니다. 이는 이름 자체 대신 <b>IPAddressPool</b> 에 할당된 레이블을 기반으로 <b>IPAddressPool</b> 에 <b>IPAddressPool</b> 을 연결하는 것입니다. 이 또는 목록에 의해 <b>IPAddressPool</b> 을 선택하지 않은 경우, 광고는 모든 <b>IPAddressPools</b> 에 적용됩니다. |

| 필드                        | 유형            | 설명                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|---------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>spec.nodeSelectors</b> | <b>string</b> | <p>선택 사항: <b>NodeSelectors</b> 는 로드 밸런서 IP의 다음 홉으로 발표하도록 노드를 제한합니다. 비어있는 경우 모든 노드가 다음 홉으로 발표됩니다.</p> <div style="display: flex; align-items: flex-start;"> <div style="width: 150px; height: 100px; background: repeating-linear-gradient(45deg, transparent, transparent 2px, black 2px, black 4px); border: 1px solid black; margin-right: 10px;"></div> <div> <p><b>중요</b></p> <p>다음 홉으로 노드 제한은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.</p> <p>Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 <a href="#">기술 프리뷰 기능 지원 범위</a>를 참조하십시오.</p> </div> </div> |

**29.5.5. L2 알림을 사용하여 MetalLB 구성**

**IPAddressPool 이 L2 프로토콜을 통해 알리도록 다음과 같이 MetalLB를 구성합니다.**

**사전 요구 사항**

- **OpenShift CLI(oc)를 설치합니다.**
- **cluster-admin 권한이 있는 사용자로 로그인합니다.**

**절차**

1. **IP 주소 풀을 만듭니다.**
  - a. **다음 예와 같은 콘텐츠를 사용하여 `ipaddresspool.yaml` 파일과 같은 파일을 생성합니다.**

```

apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
  addresses:
    - 4.4.4.0/24
  autoAssign: false

```

- b. IP 주소 풀에 대한 구성을 적용합니다.

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

2. L2 광고 생성.

- a. 다음 예와 같은 콘텐츠를 사용하여 `l2advertisement.yaml` 과 같은 파일을 생성합니다.

```

apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-l2

```

- b. 설정을 적용합니다.

```
$ oc apply -f l2advertisement.yaml
```

### 29.5.6. L2 알리프 및 라벨을 사용하여 MetalLB 구성

**BGPAdvertisement** 및 **L2Advertisement** 사용자 정의 리소스 정의의 **ipAddressPools** 필드는 이름 자체 대신 **IPAddressPool** 에 할당된 레이블을 기반으로 **IPAddressPool** 을 광고에 연결하는 데 사용됩니다.

이 예에서는 **ipAddressPoolSelectors** 필드를 구성하여 **L2** 프로토콜을 사용하여 **IPAddressPool** 가 알리도록 **MetalLB** 를 구성하는 방법을 보여줍니다.

사전 요구 사항

- **OpenShift CLI(oc)를 설치합니다.**
- **cluster-admin 권한이 있는 사용자로 로그인합니다.**

절차

1. **IP 주소 풀을 만듭니다.**
  - a. 다음 예와 같은 콘텐츠를 사용하여 **ipaddresspool.yaml** 파일과 같은 파일을 생성합니다.

```
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. **IP 주소 풀에 대한 구성을 적용합니다.**

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

2. **ipAddressPoolSelectors** 를 사용하여 **L2** 광고의 **IP**를 생성합니다.

- a. 다음 예와 같은 콘텐츠를 사용하여 **l2advertisement.yaml** 과 같은 파일을 생성합니다.

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement-label
  namespace: metallb-system
spec:
  ipAddressPoolSelectors:
    - matchExpressions:
      - key: zone
```

```
operator: In
values:
- east
```

b.

설정을 적용합니다.

```
$ oc apply -f l2advertisement.yaml
```

### 29.5.7. 추가 리소스

- 커뮤니티 별칭 구성.

## 29.6. METALLB BGP 피어 구성

클러스터 관리자는 **BGP(Border Gateway Protocol)** 피어를 추가, 수정, 삭제할 수 있습니다. **MetalLB Operator**는 **BGP 피어 사용자 정의 리소스**를 사용하여 **MetalLB 스피커 Pod**가 **BGP 세션**을 시작하기 위해 연결하는 피어를 식별합니다. 피어는 **MetalLB**가 서비스에 할당하는 로드 밸런서 **IP 주소**에 대한 경로 알림을 수신합니다.

### 29.6.1. BGP 피어 사용자 정의 리소스 정보

**BGP 피어 사용자 정의 리소스**의 필드는 다음 표에 설명되어 있습니다.

표 29.4. MetalLB BGP 피어 사용자 정의 리소스

| 필드                         | 유형             | 설명                                                                                                               |
|----------------------------|----------------|------------------------------------------------------------------------------------------------------------------|
| <b>metadata.name</b>       | <b>string</b>  | BGP 피어 사용자 정의 리소스의 이름을 지정합니다.                                                                                    |
| <b>metadata.name space</b> | <b>string</b>  | BGP 피어 사용자 정의 리소스의 네임스페이스를 지정합니다.                                                                                |
| <b>spec.myASN</b>          | <b>integer</b> | BGP 세션의 로컬 끝에 대한 자동 시스템 번호를 지정합니다. 추가하는 모든 BGP 피어 사용자 정의 리소스에서 동일한 값을 지정합니다. 범위는 <b>0</b> 에서 <b>65535</b> 사이입니다. |
| <b>spec.peerASN</b>        | <b>integer</b> | BGP 세션의 원격 끝에 대한 Autonomous System 번호를 지정합니다. 범위는 <b>0</b> 에서 <b>65535</b> 사이입니다.                                |
| <b>spec.peerAddress</b>    | <b>string</b>  | BGP 세션을 설정하기 위해 연결할 피어의 IP 주소를 지정합니다.                                                                            |

| 필드                         | 유형              | 설명                                                                                                                                                                                                                           |
|----------------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>spec.sourceAddress</b>  | <b>string</b>   | 선택 사항: BGP 세션을 설정할 때 사용할 IP 주소를 지정합니다. 값은 IPv4 주소여야 합니다.                                                                                                                                                                     |
| <b>spec.peerPort</b>       | <b>integer</b>  | 선택 사항: BGP 세션을 설정하기 위해 연결할 피어의 네트워크 포트를 지정합니다. 범위는 <b>0</b> 에서 <b>16384</b> 입니다.                                                                                                                                             |
| <b>spec.holdTime</b>       | <b>string</b>   | 선택 사항: BGP 피어를 제안할 보류 시간 기간을 지정합니다. 최소 값은 3초(10초)입니다. 일반 단위는 <b>3s,1m</b> 및 <b>5m30s</b> 와 같은 초 및 분입니다. 경로 실패를 더 빠르게 감지하려면 BFD도 구성합니다.                                                                                       |
| <b>spec.keepaliveTime</b>  | <b>string</b>   | 선택 사항: keep-alive 메시지를 BGP 피어로 보내는 간격의 최대 간격을 지정합니다. 이 필드를 지정하는 경우 <b>holdTime</b> 필드의 값도 지정해야 합니다. 지정된 값은 <b>holdTime</b> 필드의 값보다 작아야 합니다.                                                                                  |
| <b>spec.routerID</b>       | <b>string</b>   | 선택 사항: BGP 피어에 알릴 라우터 ID를 지정합니다. 이 필드를 지정하는 경우 모든 BGP 피어 사용자 정의 리소스에 동일한 값을 지정해야 합니다.                                                                                                                                        |
| <b>spec.password</b>       | <b>string</b>   | 선택 사항: TCP MD5 인증된 BGP 세션을 적용하는 라우터의 피어에 보낼 MD5 암호를 지정합니다.                                                                                                                                                                   |
| <b>spec.passwordSecret</b> | <b>string</b>   | 선택 사항: BGP Peer에 대한 인증 보안의 이름을 지정합니다. 시크릿은 <b>metallb</b> 네임스페이스에 있어야 하며 type basic-auth여야 합니다.                                                                                                                              |
| <b>spec.bfdProfile</b>     | <b>string</b>   | 선택 사항: BFD 프로파일의 이름을 지정합니다.                                                                                                                                                                                                  |
| <b>spec.nodeSelectors</b>  | <b>object[]</b> | 선택 사항: 일치 표현식과 일치 라벨을 사용하여 BGP 피어에 연결할 수 있는 노드를 제어하기 위해 선택기를 지정합니다.                                                                                                                                                          |
| <b>spec.ebgpMultiHop</b>   | <b>boolean</b>  | 선택 사항: BGP 피어가 여러 네트워크 홉을 제거하도록 지정합니다. BGP 피어가 동일한 네트워크에 직접 연결되지 않은 경우 이 필드가 <b>true</b> 로 설정되어 있지 않으면 스피커가 BGP 세션을 설정할 수 없습니다. 이 필드는 <i>외부 BGP</i> 에 적용됩니다. 외부 BGP는 BGP 피어가 다른 Autonomous System에 속하는 경우를 설명하는데 사용되는 용어입니다. |



참고

**passwordSecret** 필드는 암호 필드와 함께 사용할 수 없으며 사용할 암호가 포함된 보안에 대한 참조를 포함합니다. 두 필드를 모두 설정하면 구문 분석 오류가 발생합니다.

29.6.2. BGP 피어 구성

클러스터 관리자는 **BGP 피어 사용자 정의 리소스**를 추가하여 네트워크 라우터와 라우팅 정보를 교환하고 서비스에 대한 **IP 주소**를 알릴 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- **BGP 광고**를 사용하여 **MetalLB**를 구성합니다.

#### 절차

1. 다음 예와 같은 콘텐츠를 사용하여 **bgppeer.yaml** 과 같은 파일을 생성합니다.

```
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. **BGP 피어**에 대한 구성을 적용합니다.

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

### 29.6.3. 지정된 주소 풀에 대해 특정 BGP 피어 세트 구성

다음 절차에서는 다음 방법을 설명합니다.

- 주소 풀 세트(**pool1** 및 **pool2**)를 구성합니다.
- **BGP 피어 세트(peer1** 및 **peer2)**를 구성합니다.

- **pool1** 을 **peer1** 에 할당하고 **pool2** 를 **peer2** 에 할당하도록 **BGP** 알림을 구성합니다.

사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

절차

1.

주소 **pool1** 을 만듭니다.

a.

다음 예와 같은 콘텐츠를 사용하여 **ipaddresspool1.yaml** 과 같은 파일을 생성합니다.

```
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.

**IP** 주소 풀 **pool1** 에 대한 구성을 적용합니다.

```
$ oc apply -f ipaddresspool1.yaml
```

2.

주소 **pool2** 를 생성합니다.

a.

다음 예와 같은 콘텐츠를 사용하여 **ipaddresspool2.yaml** 과 같은 파일을 생성합니다.

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool2
spec:
```



**addresses:**

- 5.5.5.100-5.5.5.200
- 2001:100:5::200-2001:100:5::400

b.

*IP 주소 풀 pool2 에 대한 구성을 적용합니다.*

```
$ oc apply -f ipaddresspool2.yaml
```

3.

**BGP peer1 만들기.**

a.

*다음 예와 같은 콘텐츠를 사용하여 bgppeer1.yaml 과 같은 파일을 생성합니다.*

```
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.

*BGP 피어에 대한 구성을 적용합니다.*

```
$ oc apply -f bgppeer1.yaml
```

4.

**BGP 피어2 만들기.**

a.

*다음 예와 같은 콘텐츠를 사용하여 bgppeer2.yaml 과 같은 파일을 생성합니다.*

```
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.

**BGP peer2의 구성을 적용합니다.**

```
$ oc apply -f bgppeer2.yaml
```

5.

**BGP 광고 1을 생성합니다.**

a.

다음 예제와 같은 콘텐츠를 사용하여 **bgpadvertisement1.yaml** 과 같은 파일을 만듭니다.

```
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.

설정을 적용합니다.

```
$ oc apply -f bgpadvertisement1.yaml
```

6.

**BGP 광고 2를 생성합니다.**

a.

다음 예제와 같은 콘텐츠를 사용하여 **bgpadvertisement2.yaml** 과 같은 파일을 만듭니다.

```
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.

설정을 적용합니다.

```
$ oc apply -f bgpadvertisement2.yaml
```

#### 29.6.4. BGP 피어 구성의 예

##### 29.6.4.1. 예: BGP 피어에 연결하는 노드 제한

노드 선택기 필드를 지정하여 BGP 피어에 연결할 수 있는 노드를 제어할 수 있습니다.

```

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]

```

##### 29.6.4.2. 예: BGP 피어의 BFD 프로파일 지정

BGP 피어와 연결할 BFD 프로파일을 지정할 수 있습니다. BFD는 BGP보다 피어 간 통신 오류를 더 빠르게 감지하여 BGP를 보완합니다.

```

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
```



참고

**BFD(Bidirectional forwarding detection) 프로필을 삭제하고 BGP(Border Gateway Protocol) 피어 리소스에 추가된 bfdProfile 을 제거하면 BFD가 비활성화되지 않습니다. 대신 BGP 피어는 기본 BFD 프로필 사용을 시작합니다. BGP 피어 리소스에서 BFD를 비활성화하려면 BGP 피어 구성을 삭제하고 BFD 프로필없이 다시 생성합니다. 자세한 내용은 [BZ#2050824](#) 에서 참조하십시오.**

### 29.6.4.3. 예: 듀얼 스택 네트워킹용 BGP 피어 지정

듀얼 스택 네트워킹을 지원하려면 IPv4에 대해 하나의 BGP 피어 사용자 정의 리소스와 IPv6에 대해 하나의 BGP 피어 사용자 지정 리소스를 추가합니다.

```
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
```

### 29.6.5. 다음 단계

- [MetalLB를 사용하도록 서비스 구성](#)

## 29.7. 커뮤니티 별칭 구성

클러스터 관리자는 커뮤니티 별칭을 구성하고 다양한 알림에서 사용할 수 있습니다.

### 29.7.1. 커뮤니티 사용자 정의 리소스 정보

커뮤니티 사용자 지정 리소스는 커뮤니티의 별칭 컬렉션입니다. 사용자는 **BGPAdvertisement** 를 사용하여 **ipAddressPools** 를 알릴 때 사용할 이름 지정된 별칭을 정의할 수 있습니다. 커뮤니티 사용자 지정 리소스의 필드는 다음 표에 설명되어 있습니다.



참고

커뮤니티 **CRD**는 **BGPAdvertisement**에만 적용됩니다.

표 29.5. MetalLB 커뮤니티 사용자 정의 리소스

| 필드                         | 유형            | 설명                                                                                                                                |
|----------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>metadata.name</b>       | <b>string</b> | 커뮤니티의 이름을 지정합니다.                                                                                                                  |
| <b>metadata.name space</b> | <b>string</b> | 커뮤니티의 네임스페이스를 지정합니다. MetalLB Operator에서 사용하는 동일한 네임스페이스를 지정합니다.                                                                   |
| <b>spec.communities</b>    | <b>string</b> | 서비스에 할당할 MetalLB의 IP 주소 목록을 지정합니다. 단일 풀에 여러 범위를 지정할 수 있으며 모두 동일한 설정을 공유합니다. CIDR 표기법에서 각 범위를 지정하거나 하이픈으로 구분된 시작 및 끝 IP 주소로 지정합니다. |

표 29.6. CommunityAlias

| 필드           | 유형            | 설명                          |
|--------------|---------------|-----------------------------|
| <b>name</b>  | <b>string</b> | 커뮤니티의 별칭의 이름입니다.            |
| <b>value</b> | <b>string</b> | 지정된 이름에 해당하는 BGP 커뮤니티 값입니다. |

### 29.7.2. BGP 광고 및 커뮤니티 별칭을 사용하여 MetalLB 구성

**IPAddressPool** 이 **BGP** 프로토콜과 함께 광고되고 커뮤니티 별칭이 **NO\_ADVERTISE** 커뮤니티의 숫자 값으로 설정되도록 **MetalLB**를 다음과 같이 구성합니다.

다음 예에서 피어 **BGP** 라우터 **doc-example-peer-community** 는 **MetalLB**가 서비스에 할당하는 각 로드 밸런서 IP 주소에 대해 하나의 **203.0.113.200/32** 경로 및 하나의 **fc00:f853:ccd:e799::1/128** 경로를 수신합니다. 커뮤니티 별칭은 **NO\_ADVERTISE** 커뮤니티로 구성됩니다.

사전 요구 사항

- **OpenShift CLI(oc)를 설치합니다.**
- **cluster-admin 권한이 있는 사용자로 로그인합니다.**

## 절차

1.

**IP 주소 풀을 만듭니다.**

a.

다음 예와 같은 콘텐츠를 사용하여 **ipaddresspool.yaml** 파일과 같은 파일을 생성합니다.

```
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::/124
```

b.

**IP 주소 풀에 대한 구성을 적용합니다.**

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

2.

**community1** 이라는 커뮤니티 별칭을 만듭니다.

```
apiVersion: metallb.io/v1beta1
kind: Community
metadata:
  name: community1
  namespace: metallb-system
spec:
  communities:
    - name: NO_ADVERTISE
      - value: '65535:65282'
```

3.

**doc-example-bgp-peer** 라는 BGP 피어를 생성합니다.

a.

다음 예와 같은 콘텐츠를 사용하여 **bgppeer.yaml** 과 같은 파일을 생성합니다.

```
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
```

b.

**BGP** 피어에 대한 구성을 적용합니다.

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

4.

커뮤니티 별칭으로 **BGP** 광고를 만듭니다.

a.

다음 예와 같은 콘텐츠를 사용하여 **bgpadvertisement.yaml** 과 같은 파일을 만듭니다.

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgp-community-sample
  namespace: metallb-system
spec:
  aggregationLength: 32
  aggregationLengthV6: 128
  communities:
    - community1
  ipAddressPools:
    - doc-example-bgp-community
  peers:
    - doc-example-peer
```

b.

설정을 적용합니다.

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

## 29.8. METALLB BFD 프로파일 구성

클러스터 관리자는 **BFD(Bidirectional Forwarding Detection)** 프로필을 추가, 수정, 삭제할 수 있습니다

다. **MetalLB Operator**는 **BFD 프로파일 사용자 정의 리소스**를 사용하여 **BFD 세션**을 사용하여 **BFD**가 제공하는 것보다 더 빠른 경로 실패 탐지를 제공하기 위해 **BFD 세션**을 식별합니다.

### 29.8.1. BFD 프로파일 사용자 정의 리소스 정보

**BFD 프로파일 사용자 정의 리소스의 필드**는 다음 표에 설명되어 있습니다.

표 29.7. BFD 프로파일 사용자 정의 리소스

| 필드                           | 유형             | 설명                                                                                                                                                                                                                                                                      |
|------------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>metadata.name</b>         | <b>string</b>  | BFD 프로파일 사용자 정의 리소스의 이름을 지정합니다.                                                                                                                                                                                                                                         |
| <b>metadata.name space</b>   | <b>string</b>  | BFD 프로파일 사용자 정의 리소스의 네임스페이스를 지정합니다.                                                                                                                                                                                                                                     |
| <b>spec.detectMultiplier</b> | <b>integer</b> | 패킷 손실을 결정하기 위해 탐지 다중 값을 지정합니다. 이 값을 사용하여 원격 전송 간격을 곱하여 연결 손실 탐지 타이머를 결정합니다.<br><br>예를 들어, 로컬 시스템이 <b>3</b> 으로 설정되고 원격 시스템에 전송 간격이 <b>300</b> 으로 설정되어 있는 경우 로컬 시스템은 패킷을 수신하지 않고 <b>900 ms</b> 후에만 오류를 탐지합니다.<br><br>범위는 <b>2</b> 에서 <b>255</b> 사이입니다. 기본값은 <b>3</b> 입니다. |
| <b>spec.echoMode</b>         | <b>boolean</b> | 에코 전송 모드를 지정합니다. 분산 BFD를 사용하지 않는 경우 피어가 FRR인 경우에만 에코 전송 모드가 작동합니다. 기본값은 <b>false</b> 이고 에코 전송 모드가 비활성화되어 있습니다.<br><br>에코 전송 모드가 활성화되면 대역폭 사용량을 줄이기 위해 제어 패킷의 전송 간격을 늘리는 것이 좋습니다. 예를 들어, 전송 간격을 <b>2000 ms</b> 로 늘리는 것이 좋습니다.                                            |
| <b>spec.echoInterval</b>     | <b>integer</b> | 이 시스템이 에코 패킷을 보내고 받는 데 사용하는 최소 전송 간격 (더 적은 지터)을 지정합니다. 범위는 <b>10 ~ 60000</b> 입니다. 기본값은 <b>50 ms</b> 입니다.                                                                                                                                                                |
| <b>spec.minimumTtl</b>       | <b>integer</b> | 들어오는 제어 패킷에 대해 예상되는 최소 TTL을 지정합니다. 이 필드는 멀티 홉 세션에만 적용됩니다.<br><br>최소 TTL을 설정하는 목적은 패킷 유효성 검사 요구 사항을 보다 엄격하게 만들고 다른 세션에서 제어 패킷을 수신하지 않도록 하는 것입니다.<br><br>기본값은 <b>254</b> 이며 시스템은 이 시스템과 피어 간에 하나의 홉만 예상함을 나타냅니다.                                                          |



| 필드                           | 유형             | 설명                                                                                                                                                                                                                                                                 |
|------------------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>spec.passiveMode</b>      | <b>boolean</b> | <p>세션이 active 또는 passive 것으로 표시되는지 여부를 지정합니다. 수동적 세션에서는 연결을 시작하려고 하지 않습니다. 대신 수동 세션은 응답하기 전에 피어의 패킷을 제어하도록 대기합니다.</p> <p>세션을 패시브로 표시하는 것은 별 네트워크의 중앙 노드로 작동하는 라우터가 있고 보낼 시스템이 필요하지 않은 제어 패킷을 보내지 않도록 하려는 경우 유용합니다.</p> <p>기본값은 <b>false</b> 이며 세션을 활성화로 표시합니다.</p> |
| <b>spec.receiveInterval</b>  | <b>integer</b> | 이 시스템에서 제어 패킷을 수신할 수 있는 최소 간격을 지정합니다. 범위는 <b>10 ~ 60000</b> 입니다. 기본값은 <b>300</b> ms입니다.                                                                                                                                                                            |
| <b>spec.transmitInterval</b> | <b>integer</b> | 이 시스템이 제어 패킷을 보내는 데 사용하는 최소 전송 간격(더 적은 지터)을 지정합니다. 범위는 <b>10 ~ 60000</b> 입니다. 기본값은 <b>300</b> ms입니다.                                                                                                                                                               |

### 29.8.2. BFD 프로파일 구성

클러스터 관리자는 **BFD** 프로필을 추가하고 프로필을 사용하도록 **BGP** 피어를 구성할 수 있습니다. **BFD**는 **BGP**만으로 더 빠른 경로 실패 감지 기능을 제공합니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

#### 절차

1. 다음 예와 같은 콘텐츠를 사용하여 **bfdprofile.yaml** 과 같은 파일을 생성합니다.

```

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.

*BFD 프로파일에 대한 구성을 적용합니다.*

```

$ oc apply -f bfdprofile.yaml

```

### 29.8.3. 다음 단계

•

*BFD 프로필을 사용하도록 **BGP 피어**를 구성합니다.*

## 29.9. METALLB를 사용하도록 서비스 구성

클러스터 관리자는 **LoadBalancer** 유형의 서비스를 추가할 때 **MetalLB**에서 IP 주소를 할당하는 방법을 제어할 수 있습니다.

### 29.9.1. 특정 IP 주소 요청

다른 로드 밸런서 구현과 마찬가지로 **MetalLB**에는 서비스 사양에서 **spec.loadBalancerIP** 필드가 허용됩니다.

요청된 IP 주소가 주소 풀의 범위 내에 있는 경우 **MetalLB**는 요청된 IP 주소를 할당합니다. 요청된 IP 주소가 범위 내에 없는 경우 **MetalLB**에서 경고를 보고합니다.

특정 IP 주소에 대한 서비스 **YAML**의 예

```

apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080

```

```

protocol: TCP
type: LoadBalancer
loadBalancerIP: <ip_address>

```

**MetalLB**에서 요청된 IP 주소를 할당할 수 없는 경우 서비스의 **EXTERNAL-IP**는 **<pending>**을 보고하고 **oc describe service <service\_name>**을 실행하면 다음 예와 같은 이벤트가 포함됩니다.

**MetalLB**에서 요청된 IP 주소를 할당할 수 없는 이벤트의 예

```

...
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

```

### 29.9.2. 특정 풀에서 IP 주소 요청

특정 범위의 IP 주소를 할당하지만 특정 IP 주소와 관련이 없는 경우 **metallb.universe.tf/address-pool** 주석을 사용하여 지정된 주소 풀의 IP 주소를 요청할 수 있습니다.

특정 풀의 IP 주소에 대한 서비스 **YAML**의 예

```

apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
    type: LoadBalancer

```

`<address_pool_name>`에 대해 지정한 주소 풀이 없는 경우 **MetalLB**는 자동 할당을 허용하는 모든 풀에서 **IP** 주소를 할당하려고 시도합니다.

### 29.9.3. IP 주소 수락

기본적으로 주소 풀은 자동 할당을 허용하도록 구성됩니다. **MetalLB**는 이러한 주소 풀에서 **IP** 주소를 할당합니다.

자동 할당을 위해 구성된 풀의 **IP** 주소를 수락하려면 특별한 주석이나 구성이 필요하지 않습니다.

#### IP 주소를 수락하는 서비스 **YAML**의 예

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
```

### 29.9.4. 특정 IP 주소 공유

기본적으로 서비스는 **IP** 주소를 공유하지 않습니다. 그러나 단일 **IP** 주소에 서비스를 공동 배치해야 하는 경우 `metallb.universe.tf/allow-shared-ip` 주석을 서비스에 추가하여 선택적 **IP** 공유를 활성화할 수 있습니다.

```
apiVersion: v1
kind: Service
metadata:
  name: service-http
  annotations:
```

```

metallb.universe.tf/address-pool: doc-example
metallb.universe.tf/allow-shared-ip: "web-server-svc" ❶
spec:
  ports:
    - name: http
      port: 80 ❷
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value> ❸
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7 ❹
---
apiVersion: v1
kind: Service
metadata:
  name: service-https
  annotations:
    metallb.universe.tf/address-pool: doc-example
    metallb.universe.tf/allow-shared-ip: "web-server-svc" ❺
spec:
  ports:
    - name: https
      port: 443 ❻
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value> ❼
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7 ❽

```

❶ ❺

`metallb.universe.tf/allow-shared-ip` 주석에 동일한 값을 지정합니다. 이 값을 공유 키라고 합니다.

❷ ❻

서비스에 대해 서로 다른 포트 번호를 지정합니다.

❸ ❼

서비스가 동일한 Pod 세트에 트래픽을 보내도록 `externalTrafficPolicy: local`을 지정해야 하는 경우 동일한 Pod 선택기를 지정합니다. cluster 외부 트래픽 정책을 사용하는 경우 Pod 선택기를 동일할 필요가 없습니다.

❹ ❽

선택 사항: 이전 항목 3개를 지정하는 경우 MetalLB에서 서비스를 동일한 IP 주소에 배치할 수 있습니다. 서비스가 IP 주소를 공유하도록 하려면 공유할 IP 주소를 지정합니다.

기본적으로 **Kubernetes**는 다중 프로토콜 로드 밸런서 서비스를 허용하지 않습니다. 이 제한으로 인해 일반적으로 **TCP** 및 **UDP**에서 수신 대기해야 하는 **DNS**와 같은 서비스를 실행할 수 없습니다. **MetalLB**를 사용하여 이 **Kubernetes** 제한 사항을 해결하려면 다음 두 서비스를 생성합니다.

- 한 서비스에 대해 **TCP**를 지정하고 두 번째 서비스에 대해 **UDP**를 지정합니다.
- 두 서비스 모두에서 동일한 **pod** 선택기를 지정합니다.
- 동일한 공유 키와 **spec.loadBalancerIP** 값을 지정하여 **TCP** 및 **UDP** 서비스를 동일한 **IP** 주소에 공동 배치합니다.

### 29.9.5. MetalLB를 사용하여 서비스 구성

주소 풀에서 외부 **IP** 주소를 사용하도록 로드 밸런싱 서비스를 구성할 수 있습니다.

#### 사전 요구 사항

- **OpenShift CLI(oc)**를 설치합니다.
- **MetalLB Operator**를 설치하고 **MetalLB**를 시작합니다.
- 하나 이상의 주소 풀을 구성합니다.
- 클라이언트의 트래픽을 클러스터의 호스트 네트워크로 라우팅하도록 네트워크를 구성합니다.

#### 절차

1. **<service\_name>.yaml** 파일을 생성합니다. 파일에서 **spec.type** 필드가 **LoadBalancer**로 설정되어 있는지 확인합니다.

**MetalLB**에서 서비스에 할당하는 외부 **IP** 주소를 요청하는 방법에 대한 자세한 내용은 예제를 참조하십시오.

2.

*서비스를 생성합니다.***\$ oc apply -f <service\_name>.yaml***출력 예***service/<service\_name> created****검증**

•

*서비스를 설명합니다.***\$ oc describe service <service\_name>***출력 예*

```

Name:          <service_name>
Namespace:     default
Labels:        <none>
Annotations:   metallb.universe.tf/address-pool: doc-example <.>
Selector:      app=service_name
Type:          LoadBalancer <.>
IP Family Policy: SingleStack
IP Families:   IPv4
IP:            10.105.237.254
IPs:           10.105.237.254
LoadBalancer Ingress: 192.168.100.5 <.>
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: <.>
  Type Reason      Age          From          Message
  ---- -
  Normal nodeAssigned 32m (x2 over 32m) metallb-speaker announcing from node "
  <node_name>"

```

<.> 특정 풀에서 IP 주소를 요청하는 경우 주석이 있습니다. <.> 서비스 유형은 **LoadBalancer** 를 지정해야 합니다. <.> 로드 밸런서 수신 필드는 서비스가 올바르게 할당되면 외부 IP 주소를 나타냅니다. <.> 이벤트 필드는 외부 IP 주소를 표시하여 외부 IP 주소를 어셈블하기 위해 할당되었음을 나타냅니다. 오류가 발생하면 이벤트 필드에 오류 이유가 표시됩니다.

### 29.10. METALLB 로깅, 문제 해결 및 지원

**MetalLB** 구성 문제를 해결해야 하는 경우 일반적으로 사용되는 명령의 다음 섹션을 참조하십시오.

#### 29.10.1. MetalLB 로깅 수준 설정

**MetalLB**는 기본 정보 설정이 포함된 컨테이너에서 **FRRouting(FRR)**을 사용하면 많은 로깅이 생성됩니다. 이 예에 설명된 대로 **logLevel** 을 설정하여 생성된 로그의 상세 정보를 제어할 수 있습니다.

다음과 같이 **logLevel** 을 **debug** 로 설정하여 **MetalLB**에 대한 보다 깊은 통찰력을 얻을 수 있습니다.

#### 사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **OpenShift CLI(oc)**가 설치되어 있습니다.

#### 절차

1. 다음 예와 같은 콘텐츠를 사용하여 **setdebugloglevel.yaml** 과 같은 파일을 생성합니다.

```

apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  nodeSelector:
    node-role.kubernetes.io/worker: ""

```



2.

설정을 적용합니다.

```
$ oc replace -f setdebugloglevel.yaml
```



참고

`oc replace` 를 사용하면 `metallb CR`이 이미 생성되고 여기에서 로그 수준을 변경하고 있습니다.

3.

`speaker Pod`의 이름을 표시합니다.

```
$ oc get -n metallb-system pods -l component=speaker
```

출력 예

| 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 |



참고

업데이트된 로그 수준을 적용하기 위해 사용자 및 컨트롤러 포드가 다시 생성됩니다. 로그 수준은 **MetalLB**의 모든 구성 요소에 대해 수정됩니다.

4.

발표자 로그를 확인합니다.

```
$ oc logs -n metallb-system speaker-7m4qw -c speaker
```

출력 예

```
{"branch":"main","caller":"main.go:92","commit":"3d052535","goversion":"gc / go1.17.1 / amd64","level":"info","msg":"MetalLB 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-xtsrx", "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:endpoints.kubernetes.io/last-change-trigger-time": {}}, "f:generateName": {}, "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", "namespace": "openshift-kube-controller-manager-operator", "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.

**FRR 로그 보기:**

**\$ oc logs -n metallb-system speaker-7m4qw -c frr**

## 출력 예

```

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

```

### 29.10.1.1. FRRouting (FRR) 로그 수준

다음 표에서는 **FRR** 로깅 수준에 대해 설명합니다.

#### 표 29.8. 로그 수준

| 로그 수준        | 설명                                                                                              |
|--------------|-------------------------------------------------------------------------------------------------|
| <b>all</b>   | 모든 로깅 수준에 대한 모든 로깅 정보를 제공합니다.                                                                   |
| <b>debug</b> | 이 정보는 사람에게 도움이 되는 정보입니다. <b>디버그</b> 로 설정하여 자세한 문제 해결 정보를 제공합니다.                                 |
| <b>info</b>  | 항상 기록되어야 하지만 정상적인 상황에서는 사용자 개입이 필요하지 않은 정보를 제공합니다. 이는 기본 로깅 수준입니다.                              |
| <b>warn</b>  | 잠재적으로 일치하지 않는 <b>MetallB</b> 동작을 유발할 수 있는 모든 것. 일반적으로 <b>MetallB</b> 는 이러한 유형의 오류에서 자동으로 복구됩니다. |
| <b>error</b> | <b>MetallB</b> 기능에 치명적인 모든 오류입니다. 이러한 오류는 일반적으로 관리자가 수정해야 합니다.                                  |
| <b>none</b>  | 모든 로깅을 중지합니다.                                                                                   |

### 29.10.2. BGP 문제 해결

**Red Hat**이 지원하는 **BGP** 구현은 스피커 **Pod**의 컨테이너에서 **FRRouting(FRR)**을 사용합니다. 클러스터 관리자는 **BGP** 구성 문제를 해결해야 하는 경우 **FRR** 컨테이너에서 명령을 실행해야 합니다.

#### 사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **OpenShift CLI(oc)**가 설치되어 있습니다.

#### 절차

1. **speaker Pod**의 이름을 표시합니다.

```
$ oc get -n metallb-system pods -l component=speaker
```

출력 예

```
NAME          READY STATUS RESTARTS AGE
speaker-66bth 4/4   Running 0        56m
speaker-gvfnf 4/4   Running 0        56m
```

...

2.

**FRR에 대한 실행 중인 구성을 표시합니다.**

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show running-config"
```

출력 예

```
Building configuration...
```

```
Current configuration:
```

```
!
```

```

fr version 7.5.1_git
fr defaults traditional
hostname some-hostname
log file /etc/frr/frr.log informational
log timestamp precision 3
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 4
  neighbor 10.0.2.4 bfd profile doc-example-bfd-profile-full 5
```

```

  neighbor 10.0.2.4 timers 5 15
  !
```

```

address-family ipv4 unicast
  network 203.0.113.200/30 6
```

```

  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 7
```

```

  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 8
transmit-interval 35
receive-interval 35
passive-mode
echo-mode
echo-interval 35
minimum-ttl 10
!
!
end

```

<.> 라우터 **bgp** 섹션은 **MetalLB**의 **ASN**을 나타냅니다. <.> **BFD**를 추가한 각 **BGP** 피어 사용자 정의 리소스에 대해 < ip-address > remote-as <peer-ASN > 행이 존재하는지 확인합니다. <.> **BFD** 프로파일을 구성하면 **BFD** 프로파일을 확인하십시오. 는 올바른 **BGP** 피어와 연결되고 **BFD** 프로파일 명령 출력에 표시됩니다. <.> 네트워크 <ip-address-range > 행이 추가한 주소 풀 사용자 정의 리소스에 지정한 **IP** 주소 범위와 일치하는지 확인합니다.

3.

**BGP** 요약을 표시합니다.

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp summary"
```

출력 예

```

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  3
10.0.2.4    4    64500     0       0     0    0    0 never    Active    0  4

Total number of neighbors 2
    
```

1 1 3

출력에 사용자가 추가한 각 **BGP** 피어 사용자 정의 리소스에 대한 행이 포함되어 있는지 확인합니다.

2 4 2 4

수신한 **0** 개의 메시지와 전송된 메시지를 표시하는 출력은 **BGP** 세션이 없는 **BGP** 피어를 나타냅니다. 네트워크 연결 및 **BGP** 피어 구성을 확인합니다.

- 주소 풀을 수신한 **BGP** 피어를 표시합니다.

```

$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp ipv4 unicast
203.0.113.200/30"
    
```

**ipv4** 를 **ipv6** 로 교체하여 **IPv6** 주소 풀을 수신한 **BGP** 피어를 표시합니다. **203.0.113.200/30** 을 주소 풀에서 **IPv4** 또는 **IPv6 IP** 주소 범위로 바꿉니다.

출력 예

```

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 <.>
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
```

<.> 출력에 **BGP 피어의 IP 주소가 포함되어 있는지 확인합니다.**

### 29.10.3. BFD 문제 해결

Red Hat이 지원하는 BFD(Bidirectional Forwarding Detection) 구현으로 스피커 Pod의 컨테이너에서 FRRouting(FRR)을 사용합니다. BFD 구현은 BFD 피어도 설정된 BGP 세션을 가진 BGP 피어로 구성되어 있습니다. 클러스터 관리자는 BFD 구성 문제를 해결해야 하는 경우 FRR 컨테이너에서 명령을 실행해야 합니다.

#### 사전 요구 사항

- **cluster-admin** 역할의 사용자로 클러스터에 액세스할 수 있어야 합니다.
- **OpenShift CLI(oc)**가 설치되어 있습니다.

#### 절차

1. **speaker Pod의 이름을 표시합니다.**

```
$ oc get -n metallb-system pods -l component=speaker
```

출력 예

```
NAME          READY STATUS RESTARTS AGE
speaker-66bth 4/4   Running 0        26m
speaker-gvfnf 4/4   Running 0        26m
...
```

2.

**BFD 피어를 표시합니다.**

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bfd peers brief"
```

출력 예

```
Session count: 2
SessionId LocalAddress      PeerAddress      Status
=====
3909139637 10.0.1.2         10.0.2.3         up <.>
```

<.> PeerAddress 열에 각 BFD 피어가 포함되어 있는지 확인합니다. 출력에 출력에 포함할 것으로 예상되는 BFD 피어 IP 주소가 나열되지 않으면 피어와의 BGP 연결 문제 해결. status 필드가 down 으로 표시되면 노드와 피어 사이의 링크와 장치에서 연결을 확인합니다. `oc get pods -n metallb-system speaker-66bth -o jsonpath='{.spec.nodeName}'` 과 같은 명령을 사용하여 스피커 Pod의 노드 이름을 확인할 수 있습니다.

#### 29.10.4. BGP 및 BFD에 대한 MetalLB 지표

OpenShift Container Platform은 MetalLB 및 BGP 피어 및 BFD 프로파일과 관련된 다음 메트릭을 캡처합니다.

- **metallb\_bfd\_control\_packet\_input** 은 각 BFD 피어에서 수신한 BFD 제어 패킷 수를 계산합니다.
- **metallb\_bfd\_control\_packet\_output** 은 각 BFD 피어에 전송된 BFD 제어 패킷 수를 계산합니다.
- **metallb\_bfd\_echo\_packet\_input** 은 각 BFD 피어에서 수신한 BFD 에코 패킷 수를 계산합니다.
- **metallb\_bfd\_echo\_packet\_output** 은 각 BFD 피어에 전송된 BFD 에코 패킷 수를 계산합니다.
-

**metallb\_bfd\_session\_down\_events** 는 BFD 세션과 피어가 down 상태가 되는 횟수를 계산합니다.

- **metallb\_bfd\_session\_up** 은 BFD 피어와의 연결 상태를 나타냅니다. 1 세션이 가동 중임을 나타내며 0 은 세션이 다운 되었음을 나타냅니다.
- **metallb\_bfd\_session\_up\_events** 는 BFD 세션에서 up 상태를 입력한 피어와 횟수를 계산합니다.
- **metallb\_bfd\_zebra\_Forwardeds** 는 각 BFD 피어에 대한 BFD Zebra 알림 수를 계산합니다.
- **metallb\_bgp\_announced\_prefixes\_total** 은 BGP 피어에 표시되는 로드 밸런서 IP 주소 접두사 수를 계산합니다. 접두사 및 집계된 경로 라는 용어는 동일한 의미를 갖습니다.
- **metallb\_bgp\_session\_up** 은 BGP 피어와의 연결 상태를 나타냅니다. 1 세션이 가동 중임을 나타내며 0 은 세션이 다운 되었음을 나타냅니다.
- **metallb\_bgp\_updates\_total** 은 BGP 피어로 전송된 BGP 업데이트 메시지 수를 계산합니다.

#### 추가 리소스

- [모니터링 대시보드 사용에 대한 정보는 메트릭 쿼리](#) 를 참조하십시오.

#### 29.10.5. MetalLB 데이터 수집 정보

**oc adm must-gather CLI** 명령을 사용하여 클러스터, MetalLB 구성 및 MetalLB Operator에 대한 정보를 수집할 수 있습니다. 다음 기능 및 오브젝트는 MetalLB 및 MetalLB Operator와 연결되어 있습니다.

- **MetalLB Operator가 배포된 네임스페이스 및 하위 오브젝트**
- **모든 MetalLB Operator CRD(사용자 정의 리소스 정의)**

**oc adm must-gather CLI 명령은 Red Hat이 BGP 및 BFD를 구현하는 데 사용하는 FRRouting(FRR)에서 다음 정보를 수집합니다.**

- **`/etc/frr/frr.conf`**
- **`/etc/frr/frr.log`**
- **`/etc/frr/daemons` 설정 파일**
- **`/etc/frr/vtysh.conf`**

이전 목록의 로그 및 구성 파일은 각 스피커 Pod의 **frr** 컨테이너에서 수집됩니다.

로그 및 구성 파일 외에도 **oc adm must-gather CLI 명령은 다음 vtysh 명령에서 출력을 수집합니다.**

- **`show running-config`**
- **`show bgp ipv4`**
- **`show bgp ipv6`**
- **`bgp neighbors` 표시**
- **`bfd peer` 표시**

**oc adm must-gather CLI 명령을 실행할 때 추가 구성이 필요하지 않습니다.**

추가 리소스

- [클러스터에 대한 데이터 수집](#)

## 30장. 보조 인터페이스 지표와 네트워크 연결 연관 짓기

### 30.1. 모니터링을 위한 보조 네트워크 메트릭 확장

보조 장치 또는 인터페이스는 다양한 용도로 사용됩니다. 동일한 분류 기준으로 보조 장치에 대한 지표를 집계하려면 보조 장치를 분류할 방법이 있어야 합니다.

노출된 지표는 인터페이스를 포함하지만 인터페이스가 시작되는 위치는 지정하지 않습니다. 추가 인터페이스가 없는 경우 사용할 수 있습니다. 그러나 보조 인터페이스를 추가하는 경우 인터페이스 이름만 사용하여 인터페이스를 식별하기 어려울 수 있습니다.

보조 인터페이스를 추가할 때는 이름이 추가하는 순서에 따라 달라집니다. 서로 다른 보조 인터페이스는 다른 네트워크에 속할 수 있으며 다른 용도로 사용할 수 있습니다.

`pod_network_name_info`를 사용하면 인터페이스 유형을 식별하는 추가 정보로 현재 지표를 확장할 수 있습니다. 이러한 방식으로 지표를 집계하고 특정 인터페이스 유형에 특정 경보를 추가할 수 있습니다.

네트워크 유형은 관련 `NetworkAttachmentDefinition` 이름을 사용하여 생성되며 보조 네트워크의 다양한 클래스를 구별하는 데 사용됩니다. 예를 들어 서로 다른 네트워크에 속하거나 서로 다른 CNI를 사용하는 서로 다른 인터페이스는 서로 다른 네트워크 연결 정의 이름을 사용합니다.

#### 30.1.1. 네트워크 지표 데몬

네트워크 지표 데몬은 네트워크 관련 지표를 수집하고 게시하는 데몬 구성 요소입니다.

`kubelet`은 이미 관찰 가능한 네트워크 관련 지표를 게시하고 있습니다. 이러한 지표는 다음과 같습니다.

- `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`

이러한 지표의 레이블에는 다음이 포함됩니다.

- 포드 이름
- 포드 네임스페이스
- 인터페이스 이름(예: `eth0`)

이러한 지표는 예를 들면 `Multus`를 통해 `Pod`에 새 인터페이스를 추가할 때까지는 인터페이스 이름이 무엇을 나타내는지 명확하지 않기 때문에 잘 작동합니다.

인터페이스 레이블은 인터페이스 이름을 나타내지만 해당 인터페이스가 무엇을 의미하는지는 명확하지 않습니다. 인터페이스가 다양한 경우 모니터링 중인 지표에서 어떤 네트워크를 참조하는지 파악하기란 불가능합니다.

이 문제는 다음 섹션에 설명된 새로운 `pod_network_name_info`를 도입하여 해결됩니다.

### 30.1.2. 네트워크 이름이 있는 지표

이 `daemonset`는 고정 값이 0인 `pod_network_name_info` 게이지 지표를 게시합니다.

■

```
pod_network_name_info{interface="net0",namespace="namespacename",network_name="namespace/firstNAD",pod="podname"} 0
```

네트워크 이름 레이블은 **Multus**에서 추가한 주석을 사용하여 생성됩니다. 네트워크 연결 정의가 속하는 네임스페이스와 네트워크 연결 정의 이름을 연결하는 것입니다.

새 지표 단독으로는 많은 가치를 제공하지 않지만 네트워크 관련 **container\_network\_\*** 지표와 결합되는 경우 보조 네트워크 모니터링을 더 잘 지원합니다.

다음과 같은 **promql** 쿼리를 사용하면 **k8s.v1.cni.cncf.io/networks-status** 주석에서 검색된 값과 네트워크 이름이 포함된 새 지표를 가져올 수 있습니다.

```
(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)
```

## 31장. 네트워크 관찰 기능

### 31.1. NETWORK OBSERVABILITY OPERATOR 릴리스 노트

**Network Observability Operator**를 사용하면 관리자가 **OpenShift Container Platform** 클러스터의 네트워크 트래픽 흐름을 관찰하고 분석할 수 있습니다.

이 릴리스 노트에서는 **OpenShift Container Platform**에서 **Network Observability Operator**의 개발을 추적합니다.

**Network Observability Operator**에 대한 개요는 [Network Observability Operator](#) 정보를 참조하십시오.

#### 31.1.1. Network Observability Operator 1.1.0

**Network Observability Operator 1.1.0**에서 다음 권고를 사용할 수 있습니다.

- [RHSA-2023:0786 Network Observability Operator 보안 권고 업데이트](#)

**Network Observability Operator**가 안정적이며 릴리스 채널이 **v1.1.0**으로 업그레이드되었습니다.

#### 31.1.1.1. 버그 수정

- 이전 버전에서는 **Loki authToken** 구성을 **FORWARD** 모드로 설정하지 않으면 인증이 더 이상 적용되지 않으므로 **OpenShift Container Platform** 클러스터의 **OpenShift Container Platform** 콘솔에 연결할 수 있는 모든 사용자가 인증 없이 흐름을 검색할 수 있었습니다. 이제 **Loki authToken** 모드에 관계없이 클러스터 관리자만 흐름을 검색할 수 있습니다. ([BZ#2169468](#))

### 31.2. 네트워크 관찰 정보

**Red Hat**은 클러스터 관리자에게 **Network Observability Operator**를 제공하여 **OpenShift Container Platform** 클러스터의 네트워크 트래픽을 관찰합니다. 네트워크 관찰 기능은 **eBPF** 기술을 사용하여 네트워크 흐름을 생성합니다. 그러면 네트워크 흐름이 **OpenShift Container Platform** 정보로 보강되고 로키에 저장됩니다. 추가 정보 및 문제 해결을 위해 **OpenShift Container Platform** 콘솔에서 저장된 네트워크 흐름 정보를 보고 분석할 수 있습니다.



### 31.2.1. Network Observability Operator의 종속성

**Network Observability Operator**에는 다음 **Operator**가 필요합니다.

- 로키: 로키를 설치해야 합니다. 로키는 수집된 모든 흐름을 저장하는 데 사용되는 백엔드입니다. **Network Observability Operator** 설치를 위해 **Red Hat Loki Operator**를 설치하여 **Loki**를 설치하는 것이 좋습니다.

### 31.2.2. Network Observability Operator의 선택적 종속 항목

- Grafana:** **Grafana Operator**를 사용하여 사용자 정의 대시보드 및 쿼리 기능을 사용하기 위해 **Grafana**를 설치할 수 있습니다. **Red Hat**은 **Grafana Operator**를 지원하지 않습니다.
- Kafka:** **OpenShift Container Platform** 클러스터에서 확장성, 복원력 및 고가용성을 제공합니다. 대규모 배포에는 **AMQ Streams Operator**를 사용하여 **Kafka**를 설치하는 것이 좋습니다.

### 31.2.3. Network Observability Operator

**Network Observability Operator**는 **Flow Collector API** 사용자 정의 리소스 정의를 제공합니다. **Flow Collector** 인스턴스는 설치 중에 생성되며 네트워크 흐름 컬렉션의 구성을 활성화합니다. **Flow Collector** 인스턴스는 로키에 저장하기 전에 네트워크 흐름이 수집되고 **Kubernetes** 메타데이터로 보강되는 모니터링 파이프라인을 형성하는 **Pod** 및 서비스를 배포합니다. 데몬 세트 오브젝트로 배포되는 **eBPF** 에이전트는 네트워크 흐름을 생성합니다.

### 31.2.4. OpenShift Container Platform 콘솔 통합

**OpenShift Container Platform** 콘솔 통합은 개요, 토폴로지 보기 및 트래픽 흐름 테이블을 제공합니다.

#### 31.2.4.1. 네트워크 관찰 지표

**OpenShift Container Platform** 콘솔은 클러스터의 네트워크 트래픽 흐름에 대한 전체 집계 메트릭을 표시하는 개요 탭을 제공합니다. 정보는 노드, 네임스페이스, 소유자, **Pod**, 서비스에서 표시할 수 있습니다. 필터 및 디스플레이 옵션은 메트릭을 더욱 구체화할 수 있습니다.

#### 31.2.4.2. 네트워크 관찰 토폴로지 보기

**OpenShift Container Platform** 콘솔은 네트워크 흐름의 그래픽 표현과 트래픽 양을 표시하는 **Topology** 탭을 제공합니다. 토폴로지 보기는 **OpenShift Container Platform** 구성 요소 간 트래픽을 네

트위크 그래프로 나타냅니다. 필터 및 표시 옵션을 사용하여 그래프를 구체화할 수 있습니다. 노드, 네임스페이스, 소유자, Pod 및 서비스에 대한 정보에 액세스할 수 있습니다.

### 31.2.4.3. 트래픽 흐름 테이블

트래픽 흐름 테이블 뷰는 원시 흐름, 집계되지 않은 필터링 옵션 및 구성 가능한 열에 대한 보기를 제공합니다. OpenShift Container Platform 콘솔은 네트워크 흐름의 데이터와 트래픽 양을 표시하는 트래픽 흐름 탭을 제공합니다.

## 31.3. NETWORK OBSERVABILITY OPERATOR 설치

VMDK 설치에 Network Observability Operator를 사용하기 위한 사전 요구 사항입니다. 따라서 Network Observability Operator 설치 전에 이러한 단계가 아래에 설명되어 있습니다.

CloudEvent Operator는 데이터 흐름 저장을 위해 멀티 테넌시 및 인증을 통해 CloudEvent와 함께 구현하는 게이트웨이를 통합합니다. CloudEventStack 리소스는 확장 가능하고 가용성이 높은 다중 테넌트 로그 집계 시스템인, OpenShift Container Platform 인증을 사용하는 웹 프록시를 관리합니다. CloudEvent Stack 프록시는 OpenShift Container Platform 인증을 사용하여 멀티 테넌시를 적용하고 CloudEvent 로그 저장소의 데이터를 저장 및 인덱싱할 수 있습니다.



참고

또한, Operator를 사용하여 로깅에 사용할 수도 있습니다. Network Observability Operator에는 로깅과는 별도로 전용 CloudEventStack이 필요합니다.

### 31.3.1. CloudEvent Operator 설치

iPXE Operator 버전 5.6 을 사용하여 installation하는 것이 좋습니다. 이 버전은 openshift-network-tenant 구성 모드를 사용하여 CloudEventStack 인스턴스를 생성할 수 있는 기능을 제공합니다. 또한 Network Observability에 대한 완전 자동 클러스터 내 인증 및 권한 부여를 제공합니다.

사전 요구 사항

- 지원되는 로그 저장소(AWS S3, Google Cloud Storage, Azure, Swift, Minio, OpenShift Data Foundation)
- OpenShift Container Platform 4.10 이상.

- **Linux Kernel 4.18+.**

사용자가 *installation* 할 수 있는 방법은 여러 가지가 있습니다. **CloudEvent Operator**를 설치할 수 있는 한 가지 방법은 **OpenShift Container Platform** 웹 콘솔 **Operator Hub**를 사용하는 것입니다.

### 절차

1. **CloudEvent Operator** 를 설치합니다.
  - a. **OpenShift Container Platform** 웹 콘솔에서 **Operator** → **OperatorHub**를 클릭합니다.
  - b. 사용 가능한 **Operator** 목록에서 **or Operator**를 선택하고 설치를 클릭합니다.
  - c. 설치 모드에서 클러스터의 모든 네임스페이스를 선택합니다.
  - d. **CloudEvent Operator**를 설치했는지 확인합니다. **Operator** → 설치된 **Operator** 페이지를 방문하여 **test Operator**를 찾습니다.
  - e. 모든 프로젝트에서 **Status** 가 **Succeeded** 로 나열되어 있는지 확인합니다.
2. **Secret YAML** 파일을 생성합니다. 웹 콘솔 또는 **CLI**에서 이 시크릿을 생성할 수 있습니다.
  - a. 웹 콘솔을 사용하여 프로젝트 → 모든 프로젝트 드롭다운으로 이동하여 프로젝트 생성을 선택합니다. 프로젝트 이름을 **netobserv** 로 지정하고 생성을 클릭합니다.
  - b. 오른쪽 상단에 있는 **Import(가져오기)** 아이콘+ 로 이동합니다. **YAML** 파일을 편집기에 놓습니다. **access\_key\_id** 및 **access\_key\_secret** 을 사용하여 인증 정보를 지정하는 **netobserv** 네임스페이스에 이 **YAML** 파일을 생성하는 것이 중요합니다.
  - c. 시크릿을 생성하면 웹 콘솔의 워크로드 → 시크릿 아래에 나열되어야 합니다.
 

다음은 보안 **YAML** 파일의 예를 보여줍니다.

```

apiVersion: v1
kind: Secret
metadata:
  name: loki-s3
  namespace: netobserv
stringData:
  access_key_id: QUtJQUIPU0ZPRE5ON0VYQU1QTEUK
  access_key_secret:
d0phbHJYVXRuRkVNSS9LN01ERU5HL2JQeFJmaUNZRvhBTvBMRUtFWQo=
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1
    
```



**중요**

**CloudEvent**를 설치 제거하려면 **created**를 설치하는 데 사용한 메서드에 해당하는 제거 프로세스를 참조하십시오. 나머지 **ClusterRoles** 및 **ClusterRoleBindings**, 오브젝트 저장소에 저장된 데이터, 제거해야 하는 영구 불륨이 있을 수 있습니다.

**31.3.1.1. SelectStack 사용자 정의 리소스 생성**

**FlowCollector** 사양, **spec.namespace** 에서 참조하는 동일한 네임스페이스에 **CloudEventStack**을 배포하는 것이 좋습니다. 웹 콘솔 또는 **CLI**를 사용하여 네임스페이스 또는 새 프로젝트를 생성할 수 있습니다.

**절차**

1. **Operators** → 설치된 **Operator**로 이동합니다.
2. **Details**의 **Provided APIs under Provided APIs**, select **SelectStack** and click **CreateECDHEStack**.
- 3.

```

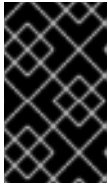
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: loki
  namespace: netobserv
spec:
  size: 1x.small
  storage:
    schemas:
      - version: v12
    
```

```

effectiveDate: '2022-06-01'
secret:
  name: loki-s3
  type: s3
storageClassName: gp3 ①
tenants:
  mode: openshift-network

```

①



중요

### 31.3.1.1.1.



참고

표 31.1.

|              | 1x.extra-small | 1x.small          |                   |
|--------------|----------------|-------------------|-------------------|
|              | 데모에서만 사용합니다.   | 500GB/day         | 2TB/day           |
| 초당 쿼리 수(QPS) | 데모에서만 사용합니다.   | 200ms에서 25-50 QPS | 200ms에서 25-75 QPS |
| 복제 요인        | 없음             | 2                 | 3                 |
| 총 CPU 요청     | 5개의 vCPU       | 36개의 vCPU         | 54 vCPU           |
| 총 메모리 요청     | 7.5Gi          | 63Gi              | 139Gi             |
| 총 디스크 요청     | 150Gi          | 300Gi             | 450Gi             |

### 31.3.1.2. CloudEventStack ingestion 구성

**CloudEventStack** 인스턴스는 구성된 크기에 따라 기본 설정으로 제공됩니다. 수집 및 쿼리 제한과 같은 이러한 설정 중 일부를 재정의할 수 있습니다. **Console** 플러그인 또는 **flowlog-pipeline** 로그에 **CloudEvent** 오류가 표시되면 업데이트할 수 있습니다.

다음은 구성된 제한의 예입니다.

```
spec:
  limits:
    global:
      ingestion:
        ingestionBurstSize: 40
        ingestionRate: 20
        maxGlobalStreamsPerTenant: 25000
      queries:
        maxChunksPerQuery: 2000000
        maxEntriesLimitPerQuery: 10000
        maxQuerySeries: 3000
```

이러한 설정에 대한 자세한 내용은 **CloudEventStack API** 참조를 참조하십시오.

이러한 제한에 도달하면 알림을 받기 위해 경고를 정의하는 것이 좋습니다. 아래 예에서 경고는 **CloudEvent Operator**, **loki\_request\_duration\_seconds\_count** 에서 제공하는 지표를 사용합니다.

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: loki-alerts
  namespace: openshift-operators-redhat
spec:
  groups:
  - name: LokiRateLimitAlerts
    rules:
    - alert: LokiTenantRateLimit
      annotations:
        message: |-
          {{ $labels.job }} {{ $labels.route }} is experiencing 429 errors.
        summary: "At any number of requests are responded with the rate limit error code."
        expr: sum(irate(loki_request_duration_seconds_count{status_code="429"}[1m])) by (job, namespace, route) / sum(irate(loki_request_duration_seconds_count[1m])) by (job, namespace, route) * 100 > 0
        for: 10s
        labels:
          severity: warning
```

### 31.3.2. 인증 및 권한 부여를 위한 역할 생성

**ClusterRole** 및 **ClusterRole Binding** 을 정의하여 인증 및 권한 부여 구성을 지정합니다. **YAML** 파일

을 생성하여 이러한 역할을 정의할 수 있습니다.

#### 절차

- 1.
2. **YAML 파일을 편집기에 놓고 생성을 클릭합니다.**

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: loki-netobserv-tenant
rules:
- apiGroups:
  - 'loki.grafana.com'
resources:
  - network
resourceNames:
  - logs
verbs:
  - 'get'
  - 'create'
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: loki-netobserv-tenant
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: loki-netobserv-tenant
subjects:
- kind: ServiceAccount
  name: flowlogs-pipeline
  namespace: netobserv

```

1

**flowlogs-pipeline** 은 CloudEvent에 씁니다. Kafka를 사용하는 경우 이 값은 **flowlogs-pipeline-transformer** 입니다.

#### 31.3.3. Kafka 설치(선택 사항)

**Kafka Operator**는 대규모 환경에서 지원됩니다. **Kafka Operator** 및 **Network Observability Operator**가 설치된 것과 마찬가지로 **Operator Hub**에서 **Red Hat AMQ Streams** 로 **Kafka Operator**를 설치할 수 있습니다.



## 참고

**Kafka**를 설치 제거하려면 설치에 사용한 메서드에 해당하는 제거 프로세스를 참조하십시오.

### 31.3.4. Network Observability Operator 설치

OpenShift Container Platform 웹 콘솔 Operator Hub를 사용하여 Network Observability Operator를 설치할 수 있습니다. Operator를 설치하면 FlowCollector CRD(사용자 정의 리소스 정의)가 제공됩니다. FlowCollector를 만들 때 웹 콘솔에서 사양을 설정할 수 있습니다.

#### 사전 요구 사항

- **test**가 설치되어 있어야 합니다. **iPXE Operator 버전 5.6**을 사용하여 설치하는 것이 좋습니다.

#### 절차

- a. OpenShift Container Platform 웹 콘솔에서 Operator → OperatorHub를 클릭합니다.
- b. OperatorHub의 사용 가능한 Operator 목록에서 Network Observability Operator를 선택하고 설치를 클릭합니다.
- c. Operators → 설치된 Operator로 이동합니다. 네트워크 관찰성을 위한 제공된 API에서 흐름 수집기 링크를 선택합니다.
  1. **Flow Collector** 탭으로 이동하여 **FlowCollector** 만들기를 클릭합니다. 폼 보기에서 다음 항목을 선택합니다.
    - **spec.agent.ebpf.Sampling** : 흐름에 대한 샘플링 크기를 지정합니다. 샘플링 크기가 줄어들면 리소스 사용률에 더 큰 영향을 미칩니다. 자세한 내용은 **spec.agent.ebpf** 아래의 **FlowCollector API** 참조를 참조하십시오.
    - **spec.deploymentModel**: Kafka를 사용하는 경우 **Kafka**가 선택되어 있는지 확인합니다.
    - **url**: 인증이 별도로 지정되어 있으므로 이 URL을 <https://loki-gateway->



<http://netobserv.svc:8080/api/logs/v1/network> 로 업데이트해야 합니다. URL의 첫 번째 부분인 "loki"는 해당 스택의 이름과 일치해야 합니다.

- **.statusUrl:** 이 값을 <https://loki-query-frontend-http.netobserv.svc:3100/> 로 설정합니다.
- **iPXE.authToken:** FORWARD 값을 선택합니다.
- **TLS.enable:** 상자가 활성화되어 있는지 확인합니다.

2. 생성을 클릭합니다.

## 검증

이 작업이 성공했는지 확인하려면 **Observe** 로 이동할 때 옵션에 네트워크 트래픽 목록이 표시되어야 합니다.

**OpenShift Container Platform** 클러스터 내에 애플리케이션 트래픽이 없으면 기본 필터에 "결과 없음"이 표시되어 시각적 흐름이 발생하지 않습니다. 필터 선택 항목 옆에 있는 모든 필터 지우기를 선택하여 흐름을 확인합니다.



## 중요

**iPXE Operator**를 사용하여 설치한 경우 콘솔 액세스를 중단할 수 있으므로 **querierUrl** 을 사용하지 않는 것이 좋습니다. 다른 유형의 **installation**을 사용하여 **ScanSetting**을 설치한 경우 이는 적용되지 않습니다.

## 추가 리소스

**Flow Collector** 사양에 대한 자세한 내용은 [Flow Collector API Reference](#) 및 [Flow Collector 샘플 리소스](#)를 참조하십시오.

### 31.3.5. Network Observability Operator 설치 제거

**Operator** → 설치된 **Operator** 영역에서 작업하는 **OpenShift Container Platform** 웹 콘솔 **Operator Hub**를 사용하여 **Network Observability Operator**를 설치 제거할 수 있습니다.

절차

1. **FlowCollector 사용자 지정 리소스를 제거합니다.**
  - a. 제공된 API 열에서 **Network Observability Operator** 옆에 있는 흐름 수집기를 클릭합니다.
  - b.
  
2. **Network Observability Operator를 설치 제거합니다.**
  - a. **Operator** → 설치된 **Operator** 영역으로 다시 이동합니다.
  - b.
  
3. **FlowCollector CRD(사용자 정의 리소스 정의)를 제거합니다.**
  - a. **Administration** → **CustomResourceDefinitions** 로 이동합니다.
  - b.
  - c. **Delete CustomResourceDefinition** 을 선택합니다.



중요

**CloudEvent Operator** 및 **Kafka**는 설치된 경우에도 남아 있으며 별도로 제거해야 합니다. 또한 오브젝트 저장소에 남아 있는 데이터와 제거해야 하는 영구 불륨이 있을 수 있습니다.

31.4. OPENSIFT CONTAINER PLATFORM의 NETWORK OBSERVABILITY OPERATOR

네트워크 관찰 기능은 네트워크 관찰 기능 **eBPF** 에이전트에서 생성하는 네트워크 트래픽 흐름을 수집하고 보강하기 위해 모니터링 파이프라인을 배포하는 **OpenShift** 운영자입니다.

### 31.4.1. 상태 보기

**Network Observability Operator**는 **Flow Collector API**를 제공합니다. 흐름 수집기 리소스가 생성되면 포트 및 서비스를 배포하여 **CloudEvent** 로그 저장소에 네트워크 흐름을 생성 및 저장하고 **OpenShift Container Platform** 웹 콘솔에서 대시보드, 메트릭 및 흐름을 표시합니다.

#### 절차

1. 다음 명령을 실행하여 **FlowCollector** 상태를 확인합니다.

```
$ oc get flowcollector/cluster
```

#### 출력 예

```
NAME      AGENT  SAMPLING (EBPF)  DEPLOYMENT MODEL  STATUS
cluster  EBPF   50                DIRECT             Ready
```

2. 다음 명령을 입력하여 **netobserv** 네임스페이스에서 실행 중인 **Pod**의 상태를 확인합니다.

```
$ oc get pods -n netobserv
```

#### 출력 예

```
NAME                                READY  STATUS  RESTARTS  AGE
flowlogs-pipeline-56hbp             1/1   Running  0          147m
flowlogs-pipeline-9plvv             1/1   Running  0          147m
flowlogs-pipeline-h5gkb             1/1   Running  0          147m
flowlogs-pipeline-hh6kf             1/1   Running  0          147m
flowlogs-pipeline-w7vv5             1/1   Running  0          147m
netobserv-plugin-cdd7dc6c-j8ggp     1/1   Running  0          147m
```

**FlowLogs-pipeline Pod**는 흐름을 수집하고 수집된 흐름을 보강한 다음 **flows**를 **CloudEvent** 스토리지로 보냅니다. **netobserv-plugin Pod**는 **OpenShift Container Platform** 콘솔을 위한 시각화 플러그인을 생성합니다.

1.

다음 명령을 입력하여 네임스페이스 **netobserv-privileged** 에서 실행 중인 **Pod**의 상태를 확인합니다.

```
$ oc get pods -n netobserv-privileged
```

출력 예

| NAME                       | READY | STATUS  | RESTARTS | AGE  |
|----------------------------|-------|---------|----------|------|
| netobserv-ebpf-agent-4lpp6 | 1/1   | Running | 0        | 151m |
| netobserv-ebpf-agent-6gbrk | 1/1   | Running | 0        | 151m |
| netobserv-ebpf-agent-klpl9 | 1/1   | Running | 0        | 151m |
| netobserv-ebpf-agent-vrcnf | 1/1   | Running | 0        | 151m |
| netobserv-ebpf-agent-xf5jh | 1/1   | Running | 0        | 151m |

**netobserv-ebpf-agent Pod**는 노드의 네트워크 인터페이스를 모니터링하여 흐름을 가져와서 **flowlog-pipeline pod**로 보냅니다.

1.

**CloudEvent Operator**를 사용하는 경우 다음 명령을 입력하여 **openshift-operators-redhat** 네임스페이스에서 실행 중인 **Pod**의 상태를 확인합니다.

```
$ oc get pods -n openshift-operators-redhat
```

출력 예

| NAME                                              | READY | STATUS  | RESTARTS | AGE |
|---------------------------------------------------|-------|---------|----------|-----|
| loki-operator-controller-manager-5f6cff4f9d-jq25h | 2/2   | Running | 0        | 18h |
| lokistack-compactor-0                             | 1/1   | Running | 0        | 18h |
| lokistack-distributor-654f87c5bc-qhkhv            | 1/1   | Running | 0        | 18h |
| lokistack-distributor-654f87c5bc-skxgm            | 1/1   | Running | 0        | 18h |
| lokistack-gateway-796dc6ff7-c54gz                 | 2/2   | Running | 0        | 18h |
| lokistack-index-gateway-0                         | 1/1   | Running | 0        | 18h |
| lokistack-index-gateway-1                         | 1/1   | Running | 0        | 18h |
| lokistack-ingester-0                              | 1/1   | Running | 0        | 18h |
| lokistack-ingester-1                              | 1/1   | Running | 0        | 18h |

|                                           |     |         |   |     |
|-------------------------------------------|-----|---------|---|-----|
| lokistack-ingester-2                      | 1/1 | Running | 0 | 18h |
| lokistack-querier-66747dc666-6vh5x        | 1/1 | Running | 0 | 18h |
| lokistack-querier-66747dc666-cjr45        | 1/1 | Running | 0 | 18h |
| lokistack-querier-66747dc666-xh8rq        | 1/1 | Running | 0 | 18h |
| lokistack-query-frontend-85c6db4fbd-b2xfb | 1/1 | Running | 0 | 18h |
| lokistack-query-frontend-85c6db4fbd-jm94f | 1/1 | Running | 0 | 18h |

### 31.4.2. Network Observability Operator 상태 및 구성 보기

`oc describe` 명령을 사용하여 상태를 검사하고 **FlowCollector**의 세부 정보를 볼 수 있습니다.

#### 절차

1. 다음 명령을 실행하여 **Network Observability Operator**의 상태 및 구성을 확인합니다.

```
$ oc describe flowcollector/cluster
```

## 31.5. NETWORK OBSERVABILITY OPERATOR 구성

**Flow Collector API** 리소스를 업데이트하여 **Network Observability Operator** 및 관리되는 구성 요소를 구성할 수 있습니다. **Flow Collector**는 설치 중에 명시적으로 생성됩니다. 이 리소스는 클러스터 전체에서 작동하기 때문에 단일 **FlowCollector**만 허용되며 **cluster**라는 이름으로 지정해야 합니다.

### 31.5.1. FlowCollector 리소스 보기

**OpenShift Container Platform** 웹 콘솔에서 직접 **YAML**을 보고 편집할 수 있습니다.

#### 절차

1. 웹 콘솔에서 **Operator** → 설치된 **Operator**로 이동합니다.
2. **NetObserv Operator**의 제공된 **API** 제목에서 **Flow Collector**를 선택합니다.
3. **cluster**를 선택한 다음 **YAML** 탭을 선택합니다. 여기에서 **FlowCollector** 리소스를 수정하여 **Network Observability Operator**를 구성할 수 있습니다.

다음 예제는 OpenShift Container Platform Network Observability Operator의 샘플 FlowCollector 리소스를 보여줍니다.

### FlowCollector 리소스 샘플

```

apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: DIRECT
  agent:
    type: EBPF 1
    ebpf:
      sampling: 50 2
      logLevel: info
      privileged: false
    resources:
      requests:
        memory: 50Mi
        cpu: 100m
      limits:
        memory: 800Mi
  processor:
    logLevel: info
    resources:
      requests:
        memory: 100Mi
        cpu: 100m
      limits:
        memory: 800Mi
  loki: 3
    url: 'http://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network'
    statusUrl: 'https://loki-query-frontend-http.netobserv.svc:3100/'
    authToken: FORWARD
    tls:
      enable: true
      caCert:
        type: configmap
        name: loki-gateway-ca-bundle
        certFile: service-ca.crt
  consolePlugin:
    register: true
    logLevel: info
    portNaming:
      enable: true
      portNames:
        "3100": loki
  quickFilters: 4
    - name: Applications

```

```

filter:
  src_namespace!: 'openshift-,netobserv'
  dst_namespace!: 'openshift-,netobserv'
default: true
- name: Infrastructure
  filter:
    src_namespace: 'openshift-,netobserv'
    dst_namespace: 'openshift-,netobserv'
- name: Pods network
  filter:
    src_kind: 'Pod'
    dst_kind: 'Pod'
default: true
- name: Services network
  filter:
    dst_kind: 'Service'

```

1

에이전트 사양인 `spec.agent.type` 은 `skopeo PF` 여야 합니다. `eBPF`는 `OpenShift Container Platform`에서 지원되는 유일한 옵션입니다.

2

`Sampling` 사양인 `spec.agent.ebpf.sampling` 을 설정하여 리소스를 관리할 수 있습니다. 더 낮은 샘플링 값은 많은 양의 곱셈, 메모리 및 스토리지 리소스를 소비할 수 있습니다. 샘플링 비율 값을 지정하여 이 문제를 완화할 수 있습니다. 값 `100`은 `100`마다 `1` 흐름이 샘플링됨을 의미합니다. 값이 `0` 또는 `1`이면 모든 흐름이 캡처됩니다. 값이 낮을수록 반환된 흐름이 증가하고 파생된 지표의 정확도가 높아집니다. 기본적으로 `eBPF` 샘플링은 `50` 값으로 설정되므로 `50`마다 `1`개의 흐름이 샘플링됩니다. 더 많은 샘플 흐름도 더 많은 스토리지가 필요하다는 것을 의미합니다. 클러스터에서 관리할 수 있는 설정을 결정하려면 기본값으로 시작하고 경험적으로 구체화하는 것이 좋습니다.

3

`spec.loki`의 `spec.loki` 는 `client`를 지정합니다. 기본값은 `Installing the iPXE Operator` 섹션에 언급된 `iPXE` 설치 경로와 일치합니다. `CloudEvent`에 대한 다른 설치 방법을 사용한 경우 설치에 적합한 클라이언트 정보를 지정합니다.

4

`spec.quickFilters` 사양은 웹 콘솔에 표시되는 필터를 정의합니다. 애플리케이션 필터 키인 `10.0.0.1_namespace` 및 `dst_namespace` 는 부정(!)이므로 애플리케이션 필터는 시작되지 않거나 `openshift-` 또는 `netobserv` 네임스페이스의 대상이 아닌 모든 트래픽을 표시합니다. 자세한 내용은 아래 빠른 필터 구성을 참조하십시오.

### 31.5.2. Kafka를 사용하여 흐름 수집기 리소스 구성

Kafka를 사용하도록 **FlowCollector** 리소스를 구성할 수 있습니다. Kafka 인스턴스가 실행 중이어야 하며 해당 인스턴스에서 **OpenShift Container Platform Network Observability** 전용 Kafka 주제를 생성해야 합니다. 자세한 내용은 [AMQ Streams의 Kafka 설명서](#)와 같은 [Kafka 설명서](#)를 참조하십시오.

다음 예제에서는 Kafka를 사용하도록 **OpenShift Container Platform Network Observability Operator**의 **FlowCollector** 리소스를 수정하는 방법을 보여줍니다.

### FlowCollector 리소스의 Kafka 구성 샘플

```
deploymentModel: KAFKA
kafka:
  address: "kafka-cluster-kafka-bootstrap.netobserv"
  topic: network-flows
  tls:
    enable: false
```

1

**Kafka** 배포 모델을 활성화하려면 **DIRECT** 대신 **spec.deploymentModel** 을 **KAFKA** 로 설정합니다.

2

**spec.kafka.address** 는 Kafka 부트스트랩 서버 주소를 나타냅니다. 필요한 경우 포트 **9093**에서 TLS를 사용하기 위해 **kafka-cluster-kafka-bootstrap.netobserv:9093** 과 같은 포트를 지정할 수 있습니다.

3

**spec.kafka.topic** 은 Kafka에서 생성된 주제의 이름과 일치해야 합니다.

4

**spec.kafka.tls** 는 TLS 또는 mTLS를 사용하여 Kafka와의 모든 통신을 암호화하는 데 사용할 수 있습니다. 활성화된 경우 Kafka CA 인증서를 **ConfigMap** 또는 **Secret**으로 사용할 수 있어야 합니다. **flowlogs-pipeline** 프로세서 구성 요소가 배포된 네임스페이스(**default: netobserv**)와 **eBPF** 에이전트가 배포된 네임스페이스(기본값: **netobserv-privileged**). **spec.kafka.tls.caCert** 를 사용하여 참조해야 합니다. mTLS를 사용하는 경우 이러한 네임스페이스에서 클라이언트 시크릿을 사용할 수 있어야 하며 ( **AMQ Streams User Operator**를 사용하여 인스턴스의 경우 생성할 수 있음) **spec.kafka.tls.userCert** 로 참조되어야 합니다.



### 31.5.3. Flow Collector 리소스 업데이트

OpenShift Container Platform 웹 콘솔에서 YAML을 편집하는 대신 `flowcollector CR`(사용자 정의 리소스)에 패치를 적용하여 `eBPF` 샘플링과 같은 사양을 구성할 수 있습니다.

#### 절차

1. 다음 명령을 실행하여 `flowcollector CR`을 패치하고 `spec.agent.ebpf.sampling` 값을 업데이트합니다.

```
$ oc patch flowcollector cluster --type=json -p [{"op": "replace", "path": "/spec/agent/ebpf/sampling", "value": <new value>}] -n netobserv
```

### 31.5.4. 쿼리 필터 구성

`FlowCollector` 리소스에서 필터를 수정할 수 있습니다. 정확한 일치는 값의 `double-quotes`를 사용할 수 있습니다. 그렇지 않으면 부분 일치 항목이 텍스트 값에 사용됩니다. 키의 끝에 배치된 `bang(!)` 문자는 부정을 의미합니다. `YAML` 수정에 대한 자세한 내용은 샘플 `FlowCollector` 리소스를 참조하십시오.



#### 참고

"all of" 또는 "any of"라는 필터 일치하는 사용자가 쿼리 옵션에서 수정할 수 있는 UI 설정입니다. 이 리소스 구성은 포함되지 않습니다.

다음은 사용 가능한 모든 필터 키 목록입니다.

표 31.2. 필터 키

| Universe* | 소스                         | 대상                         | 설명                                                                                                        |
|-----------|----------------------------|----------------------------|-----------------------------------------------------------------------------------------------------------|
| namespace | <code>src_namespace</code> | <code>dst_namespace</code> | 특정 네임스페이스와 관련된 트래픽을 필터링합니다.                                                                               |
| name      | <code>src_name</code>      | <code>dst_name</code>      | 특정 Pod, 서비스 또는 노드(호스트 네트워크 트래픽용)와 같은 지정된 리프 리소스 이름과 관련된 트래픽을 필터링합니다.                                      |
| kind      | <code>src_kind</code>      | <code>dst_kind</code>      | 지정된 리소스 종류와 관련된 트래픽을 필터링합니다. 리소스 종류에는 리프 리소스(Pod, 서비스 또는 노드) 또는 소유자 리소스(Deployment 및 StatefulSet)가 포함됩니다. |

| Universal*     | 소스               | 대상               | 설명                                                                                                                                                                                      |
|----------------|------------------|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| owner_name     | src_owner_name   | dst_owner_name   | 지정된 리소스 소유자, 즉 워크로드 또는 Pod 세트와 관련된 트래픽을 필터링합니다. 예를 들어 배포 이름, StatefulSet 이름 등을 사용할 수 있습니다.                                                                                              |
| resource       | src_resource     | dst_resource     | 고유하게 식별하는 표준 이름으로 표시되는 특정 리소스와 관련된 트래픽을 필터링합니다. 정식 표기법은 네임스페이스가 지정된 종류의 <b>kind.namespace.name</b> 또는 노드의 <b>node.name</b> 입니다. 예를 들면 <b>Deployment.my-namespace.my-web-server</b> 입니다. |
| address        | src_address      | dst_address      | IP 주소와 관련된 트래픽을 필터링합니다. IPv4 및 IPv6가 지원됩니다. CIDR 범위도 지원됩니다.                                                                                                                             |
| mac            | src_mac          | dst_mac          | MAC 주소와 관련된 트래픽을 필터링합니다.                                                                                                                                                                |
| port           | src_port         | dst_port         | 특정 포트와 관련된 트래픽을 필터링합니다.                                                                                                                                                                 |
| host_addresses | src_host_address | dst_host_address | Pod가 실행 중인 호스트 IP 주소와 관련된 트래픽을 필터링합니다.                                                                                                                                                  |
| 프로토콜           | 해당 없음            | 해당 없음            | TCP 또는 UDP와 같은 프로토콜과 관련된 트래픽을 필터링합니다.                                                                                                                                                   |

- 소스 또는 대상에 대한 범용 키 필터입니다. 예를 들어, 필터링 **name: 'my-pod'** 는 일치하는 유형(**all** 또는 **Match any**) 여부에 관계없이 **my-pod** 및 모든 트래픽의 모든 트래픽을 의미합니다.

### 31.6. 네트워크 트래픽 관찰

관리자는 자세한 문제 해결 및 분석을 위해 **OpenShift Container Platform** 콘솔에서 네트워크 트래픽을 확인할 수 있습니다. 이 기능을 사용하면 트래픽 흐름의 다양한 그래픽 표현에서 통찰력을 얻을 수 있습니다. 네트워크 트래픽을 관찰하는 데 사용할 수 있는 몇 가지 보기가 있습니다.

#### 31.6.1. 개요 보기에서 네트워크 트래픽 관찰

개요 보기에는 클러스터에서 네트워크 트래픽 흐름의 전체 집계 메트릭이 표시됩니다. 관리자는 사용

가능한 디스플레이 옵션을 사용하여 통계를 모니터링할 수 있습니다.

### 31.6.1.1. 개요 보기 작업

관리자는 개요 보기로 이동하여 흐름 속도 통계의 그래픽 표시를 볼 수 있습니다.

#### 절차

1.       모니터링 → 네트워크 트래픽 으로 이동합니다.
2.       네트워크 트래픽 페이지에서 개요 탭을 클릭합니다.

메뉴 아이콘을 클릭하여 각 흐름 속도 데이터의 범위를 구성할 수 있습니다.

### 31.6.1.2. 개요 보기에 대한 고급 옵션 구성

고급 옵션을 사용하여 그래픽 보기를 사용자 지정할 수 있습니다. 고급 옵션에 액세스하려면 고급 옵션 표시를 클릭합니다. 표시 옵션 드롭다운 메뉴를 사용하여 그래프에서 세부 정보를 구성할 수 있습니다. 사용 가능한 옵션은 다음과 같습니다.

- **메트릭 유형: 10.0.0.1s** 또는 패킷에 표시되는 메트릭입니다. 기본값은 **10.0.0.1s**입니다.
- **범위:** 네트워크 트래픽이 이동하는 구성 요소의 세부 정보를 선택합니다. 범위를 노드,네임스페이스,소유자 또는 리소스로 설정할 수 있습니다. 소유자는 리소스 집계입니다. **resource**는 호스트 네트워크 트래픽의 경우 **pod**, 서비스, 노드 또는 알 수 없는 **IP** 주소일 수 있습니다. 기본값은 **Namespace**입니다.
- **truncate labels:** 드롭다운 목록에서 레이블의 필요한 너비를 선택합니다. 기본값은 **M**입니다.

#### 31.6.1.2.1. 패널 관리

필요한 통계를 표시하고 다시 정렬할 수 있습니다. 열을 관리하려면 패널 관리를 클릭합니다.

### 31.6.2. 트래픽 흐름 보기에서 네트워크 트래픽 관찰

트래픽 흐름 보기에는 네트워크 흐름의 데이터와 테이블의 트래픽 양이 표시됩니다. 관리자는 트래픽 흐름 테이블을 사용하여 애플리케이션 전체의 트래픽 양을 모니터링할 수 있습니다.

### 31.6.2.1. 트래픽 흐름 보기 작업

관리자는 **Traffic flows** 테이블로 이동하여 네트워크 흐름 정보를 볼 수 있습니다.

#### 절차

1. 모니터링 → 네트워크 트래픽 으로 이동합니다.
2. 네트워크 트래픽 페이지에서 트래픽 흐름 탭을 클릭합니다.

각 행을 클릭하여 해당 흐름 정보를 가져올 수 있습니다.

### 31.6.2.2. 트래픽 흐름 보기에 대한 고급 옵션 구성

고급 옵션 표시를 사용하여 보기를 사용자 지정하고 내보낼 수 있습니다. 표시 옵션 드롭다운 메뉴를 사용하여 행 크기를 설정할 수 있습니다. 기본값은 **Normal** 입니다.

#### 31.6.2.2.1. 열 관리

표시되는 데 필요한 열을 선택하고 순서를 다시 정렬할 수 있습니다. 열을 관리하려면 열 관리를 클릭합니다.

#### 31.6.2.2.2. 트래픽 흐름 데이터 내보내기

트래픽 흐름 보기에서 데이터를 내보낼 수 있습니다.

#### 절차

1. 데이터 내보내기 를 클릭합니다.
2. 팝업 창에서 모든 데이터를 내보내려면 모든 데이터 내보내기 확인란을 선택하고, 내보낼 필수 필드를 선택하려면 확인란의 선택을 취소할 수 있습니다.

3. **내보내기** 를 클릭합니다.

### 31.6.3. 토폴로지 보기에서 네트워크 트래픽 관찰

토폴로지 보기는 네트워크 흐름과 트래픽 양을 그래픽으로 나타냅니다. 관리자는 토폴로지 보기를 사용하여 애플리케이션 전체의 트래픽 데이터를 모니터링할 수 있습니다.

#### 31.6.3.1. 토폴로지 보기 작업

관리자는 토폴로지 보기로 이동하여 구성 요소의 세부 정보 및 지표를 확인할 수 있습니다.

#### 절차

1. **모니터링** → **네트워크 트래픽** 으로 이동합니다.
2. **네트워크 트래픽 페이지**에서 **토폴로지** 탭을 클릭합니다.

토폴로지 에서 각 구성 요소를 클릭하면 구성 요소의 세부 정보 및 지표를 확인할 수 있습니다.

#### 31.6.3.2. 토폴로지 보기의 고급 옵션 구성

고급 옵션 표시를 사용하여 보기를 사용자 지정하고 내보낼 수 있습니다. 고급 옵션 보기에는 다음과 같은 기능이 있습니다.

- **보기:** 보기에서 필요한 구성 요소를 검색하려면 다음을 수행하십시오.
- **표시 옵션:** 다음 옵션을 구성하려면 다음을 수행합니다.
  - **layout -** 그래픽 표현의 레이아웃을 선택합니다. 기본값은 **ColaNoForce** 입니다.
  - **범위:** 네트워크 트래픽이 이동하는 구성 요소의 범위를 선택합니다. 기본값은 **Namespace** 입니다.

- **groups:** 구성 요소를 그룹화하여 소유권에 대한 이해를 부여합니다. 기본값은 **None**입니다.
- **그룹 축소:** 그룹을 확장하거나 축소하려면 다음을 수행합니다. 그룹은 기본적으로 확장됩니다. **Groups** 값이 **None** 인 경우 이 옵션은 비활성화되어 있습니다.
- **Display:** 표시해야 하는 세부 사항을 선택하려면 다음을 수행하십시오. 모든 옵션은 기본적으로 확인됩니다. 사용 가능한 옵션은 **Edges** 레이블, **Edges** 레이블 및 **10.0.0.1**입니다.
- **truncate labels:** 드롭다운 목록에서 라벨의 필요한 너비를 선택하려면. 기본값은 **M**입니다.

### 31.6.3.2.1. 토폴로지 뷰 내보내기

보기를 내보내려면 토폴로지 내보내기 보기를 클릭합니다. 보기는 **PNG** 형식으로 다운로드됩니다.

### 31.6.4. 네트워크 트래픽 필터링

기본적으로 네트워크 트래픽 페이지에는 **FlowCollector** 인스턴스에 구성된 기본 필터를 기반으로 클러스터의 트래픽 흐름 데이터가 표시됩니다. 필터 옵션을 사용하여 사전 설정 필터를 변경하여 필요한 데이터를 관찰할 수 있습니다.

#### 퀵 필터

빠른 필터 드롭다운 메뉴의 기본값은 **FlowCollector** 구성에 정의됩니다. 콘솔에서 옵션을 수정할 수 있습니다.

#### 고급 필터

필터링할 매개변수와 해당 텍스트 값을 제공하여 고급 필터를 설정할 수 있습니다. 매개 변수 드롭다운 목록의 **Common** 섹션은 **Source** 또는 **Destination** 과 일치하는 결과를 필터링합니다. 적용된 필터를 활성화하거나 비활성화하려면 필터 옵션 아래에 나열된 적용된 필터를 클릭하면 됩니다.



#### 참고

텍스트 값을 지정하는 규칙을 이해하려면 **learn More** 을 클릭합니다.

#### 쿼리 옵션

쿼리 옵션을 사용하여 아래와 같이 검색 결과를 최적화할 수 있습니다.

- reporter Node:** 모든 흐름은 소스 노드와 대상 노드 모두에서 보고할 수 있습니다. 클러스터 **Ingress**의 경우 대상 노드와 클러스터 송신의 흐름이 보고되면 소스 노드에서 흐름이 보고됩니다. **Source** 또는 **Destination** 을 선택할 수 있습니다. 개요 및 토폴로지 보기에 대해 옵션 **Both** 가 비활성화되어 있습니다. 선택한 기본값은 **Destination** 입니다.
- 일치 필터:** 고급 필터에서 선택한 다양한 필터 매개변수 간 관계를 확인할 수 있습니다. 사용 가능한 옵션은 모두 일치하고 모든 항목에 대응합니다. 일치하는 모든 값과 일치하는 결과를 제공하고 입력한 값과 일치하는 결과를 제공합니다. 기본값은 **all**과 일치합니다.
- limit:** 내부 백엔드 쿼리에 대한 데이터 제한입니다. 일치 및 필터 설정에 따라 지정된 제한 내에 트래픽 흐름 데이터 수가 표시됩니다.

기본 필터 재설정 을 클릭하여 기존 필터를 제거하고 **FlowCollector** 구성에 정의된 필터를 적용할 수 있습니다.

또는 네임스페이스, 서비스, 경로, 노드 및 워크로드 페이지의 네트워크 트래픽 탭에서 해당 집계의 필터링된 데이터를 제공하는 트래픽 흐름 데이터에 액세스할 수 있습니다.

추가 리소스

**FlowCollector** 의 빠른 필터 구성에 대한 자세한 내용은 [빠른 필터](#) 및 [흐름 수집기 샘플 리소스](#) 구성을 참조하십시오.

## 31.7. FLOWCOLLECTOR 구성 매개변수

**FlowCollector**는 **netflow** 컬렉션을 개발하고 구성하는 **flowcollectors API**의 스키마입니다.

### 31.7.1. FlowCollector API 사양

유형

**object**

| 속성                | 유형                | 설명                                                                                                                                                                                                                                                                                                                     |
|-------------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>apiVersion</b> | <b>string</b>     | APIVersion은 버전이 지정된 이 오브젝트 표현의 스키마를 정의합니다. 서버는 인식된 스키마를 최신 내부 값으로 변환해야 하며 인식되지 않은 값을 거부할 수 있습니다. 자세한 내용은 <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources</a>     |
| <b>kind</b>       | <b>string</b>     | kind는 이 오브젝트가 나타내는 REST 리소스에 해당하는 문자열 값입니다. 서버는 클라이언트가 요청을 제출한 끝점에서 이를 유추할 수 있습니다. 업데이트할 수 없습니다. 자세한 내용은 <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds</a> |
| <b>metadata</b>   | <b>ObjectMeta</b> | 표준 오브젝트의 메타데이터입니다. 자세한 내용은 <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata</a>                                                                                     |
| <b>spec</b>       | <b>object</b>     | FlowCollectorSpec은 원하는 FlowCollector 상태를 정의합니다.                                                                                                                                                                                                                                                                        |
| <b>status</b>     | <b>object</b>     | FlowCollectorStatus는 FlowCollector의 관찰 상태를 정의합니다.                                                                                                                                                                                                                                                                      |

### 31.7.1.1. .spec

#### 설명

**FlowCollectorSpec**은 원하는 **FlowCollector** 상태를 정의합니다.

#### 유형

**object**

#### 필수 항목



- 에이전트
- *deploymentModel*

| 속성                     | 유형            | 설명                                                                                                                                                                                                                                                                                                                                                          |
|------------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 에이전트                   | <b>object</b> | 흐름 추출을 위한 에이전트입니다.                                                                                                                                                                                                                                                                                                                                          |
| <b>consolePlugin</b>   | <b>object</b> | ConsolePlugin은 사용 가능한 경우 OpenShift Container Platform 콘솔 플러그인과 관련된 설정을 정의합니다.                                                                                                                                                                                                                                                                               |
| <b>deploymentModel</b> | <b>string</b> | DeploymentModel은 흐름 처리를 위해 원하는 배포 유형을 정의합니다. 가능한 값은 에이전트에서 직접 흐름 프로세서를 수신 대기하도록 "DIRECT"(기본값) 또는 프로세서에서 사용하기 전에 Kafka 파이프라인으로 전송되는 "KAFKA"입니다. Kafka는 향상된 확장성, 복원력 및 고가용성을 제공할 수 있습니다 (자세한 내용은 <a href="https://www.redhat.com/en/topics/integration/what-is-apache-kafka">https://www.redhat.com/en/topics/integration/what-is-apache-kafka</a> 을 참조하십시오). |
| <b>exporters</b>       | <b>array</b>  | 내보내기 사용자는 사용자 지정 사용 또는 저장을 위해 추가 선택적 내보내기를 정의합니다. 이것은 실험적인 기능입니다. 현재는 KAFKA 내보내기만 사용할 수 있습니다.                                                                                                                                                                                                                                                               |
| <b>exporters[]</b>     | <b>object</b> | FlowCollectorExporter는 보강된 흐름을 보낼 수 있는 추가 내보내기를 정의합니다.                                                                                                                                                                                                                                                                                                      |
| <b>kafka</b>           | <b>object</b> | Kafka 구성을 사용하여 흐름 컬렉션 파이프라인의 일부로 Kafka를 브로커로 사용할 수 있습니다. "spec.deploymentModel"이 "KAFKA"인 경우 사용 가능합니다.                                                                                                                                                                                                                                                      |
| <b>loki</b>            | <b>object</b> | 흐름 저장소, 클라이언트 설정입니다.                                                                                                                                                                                                                                                                                                                                        |

| 속성               | 유형            | 설명                                                                         |
|------------------|---------------|----------------------------------------------------------------------------|
| <b>namespace</b> | <b>string</b> | NetObserv Pod가 배포되는 네임스페이스입니다. 비어 있으면 Operator의 네임스페이스가 사용됩니다.             |
| <b>프로세서</b>      | <b>object</b> | 프로세서는 에이전트에서 흐름을 수신하는 구성 요소의 설정을 정의하고, 이를 보강하고, CloudEvent 지속성 계층으로 전달합니다. |

### 31.7.1.2. .spec.agent

#### 설명

흐름 추출을 위한 에이전트입니다.

#### 유형

**object**

#### 필수 항목

- **type**

| 속성           | 유형            | 설명                                                                      |
|--------------|---------------|-------------------------------------------------------------------------|
| <b>ebpf</b>  | <b>object</b> | eBPF는 "agent.type" 속성이 "EBPF"로 설정된 경우 eBPF 기반 흐름 보고기와 관련된 설정을 설명합니다.    |
| <b>ipfix</b> | <b>object</b> | IPFIX는 "agent.type" 속성이 "IPFIX"로 설정된 경우 IPFIX 기반 흐름 보고기와 관련된 설정을 설명합니다. |

| 속성          | 유형            | 설명                                                                                                                                                                                                                                                       |
|-------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b> | <b>string</b> | type은 추적 에이전트를 선택합니다. 가능한 값은 NetObserv eBPF 에이전트, "IPFIX"를 사용하여 레거시 IPFIX 컬렉터를 사용하는 "EBPF"(기본값)입니다. "EBPF"는 대부분의 경우 더 나은 성능을 제공하며 클러스터에 설치된 CNI와 관계없이 작동해야 합니다. "IPFIX"는 OVN-Kubernetes CNI에서 작동합니다(IPFIX 내보내기를 지원하는 경우 다른 CNI가 작동할 수 있지만 수동 구성이 필요합니다). |

### 31.7.1.2.1. .spec.agent.ebpf

#### 설명

**eBPF**는 "agent.type" 속성이 "EBPF"로 설정된 경우 **eBPF** 기반 흐름 보고기와 관련된 설정을 설명합니다.

#### 유형

**object**

| 속성                        | 유형             | 설명                                                                                                                                                                                  |
|---------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>cacheActiveTimeout</b> | <b>string</b>  | cacheActiveTimeout은 보고자가 전송하기 전에 흐름을 집계하는 최대 기간입니다. <b>cacheMaxFlows</b> 및 <b>cacheActiveTimeout</b> 을 늘리면 네트워크 트래픽 오버헤드와 CPU 로드를 줄일 수 있지만 메모리 사용량이 증가하고 흐름 컬렉션의 대기 시간이 증가할 수 있습니다. |
| <b>cacheMaxFlows</b>      | <b>integer</b> | cacheMaxFlows는 집계의 최대 흐름 수이며, 도달하면 보고자가 흐름을 보냅니다. <b>cacheMaxFlows</b> 및 <b>cacheActiveTimeout</b> 을 늘리면 네트워크 트래픽 오버헤드와 CPU 로드를 줄일 수 있지만 메모리 사용량이 증가하고 흐름 컬렉션의 대기 시간이 증가할 수 있습니다.   |

| 속성                       | 유형             | 설명                                                                                                                                                                                        |
|--------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>debug</b>             | <b>object</b>  | debug를 사용하면 eBPF 에이전트의 내부 구성의 일부 측면을 설정할 수 있습니다. 이 섹션은 GOGC 및 GOMAXPROCS env vars와 같은 디버깅 및 세분화된 성능 최적화만을 위한 것입니다. 값을 설정하면 사용자가 자신의 위험을 감수해야 합니다.                                         |
| <b>excludeInterfaces</b> | 배열(문자열)        | excludeInterfaces에는 흐름 추적에서 제외될 인터페이스 이름이 포함되어 있습니다. 항목이 <b>/br-/</b> 과 같이 슬래시로 묶이면 정규식과 일치하지만 그렇지 않으면 대소문자를 구분하지 않는 문자열로 일치합니다.                                                          |
| <b>imagePullPolicy</b>   | <b>string</b>  | imagePullPolicy는 위에서 정의한 이미지의 Kubernetes 가져오기 정책입니다.                                                                                                                                      |
| 인터페이스                    | 배열(문자열)        | 인터페이스에는 흐름이 수집되는 인터페이스 이름이 포함됩니다. 비어 있는 경우 에이전트는 ExcludeInterfaces에 나열된 인터페이스를 제외하고 시스템의 모든 인터페이스를 가져옵니다. 항목을 슬래시로 묶은 경우(예: <b>/br-/</b> )는 정규식과 일치합니다. 그렇지 않으면 대소문자를 구분하지 않는 문자열로 일치합니다. |
| <b>kafkaBatchSize</b>    | <b>integer</b> | kafkaBatchSize는 파티션에 보내기 전에 요청의 최대 크기를 바이트 단위로 제한합니다. Kafka를 사용하지 않는 경우 무시됩니다. 기본값: 10MB.                                                                                                 |
| <b>logLevel</b>          | <b>string</b>  | loglevel은 NetObserv eBPF 에이전트의 로그 수준을 정의합니다.                                                                                                                                              |

| 속성                | 유형             | 설명                                                                                                                                                                                                                                                                |
|-------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>privileged</b> | <b>boolean</b> | eBPF 에이전트 컨테이너에 대한 권한 모드입니다. 일반적으로 이 설정은 무시하거나 false로 설정할 수 있습니다. 이 경우 Operator는 세분화된 기능(BPF, PERFMON, NET_ADMIN, SYS_RESOURCE)을 컨테이너에 설정하여 올바른 작업을 활성화합니다. 어떤 이유로 CAP_BPF를 인식하지 못하는 이전 커널 버전이 사용 중인 경우와 같이 이러한 기능을 설정할 수 없는 경우 더 많은 글로벌 권한을 위해 이 모드를 해제할 수 있습니다. |
| <b>resources</b>  | <b>object</b>  | resources는 이 컨테이너에 필요한 컴퓨팅 리소스입니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a>                                            |
| <b>sampling</b>   | <b>integer</b> | 흐름 보고자의 샘플링 비율입니다. 100은 100에 하나의 흐름이 전송되었음을 의미합니다. 0 또는 1은 모든 흐름이 샘플링됨을 의미합니다.                                                                                                                                                                                    |

### 31.7.1.2.2. .spec.agent.ebpf.debug

#### 설명

**debug**를 사용하면 eBPF 에이전트의 내부 구성의 일부 측면을 설정할 수 있습니다. 이 섹션은 **GOGC** 및 **GOMAXPROCS env vars**와 같은 디버깅 및 세분화된 성능 최적화만을 위한 것입니다. 값을 설정하면 사용자가 자신의 위험을 감수해야 합니다.

#### 유형

**object**

| 속성 | 유형 | 설명 |
|----|----|----|
|----|----|----|

| 속성         | 유형      | 설명                                                                                                                                                                           |
|------------|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>env</b> | 개체(문자열) | env는 사용자 지정 환경 변수를 NetObserv 에이전트에 전달할 수 있습니다. Edge debug 또는 지원 시나리오에서만 유용하기 때문에 FlowCollector 설명자의 일부로 공개적으로 노출되지 않는 GOGC GOMAXPROCS 옵션과 같은 매우 포괄적인 성능 튜닝 옵션을 전달하는 데 유용합니다. |

### 31.7.1.2.3. .spec.agent.ebpf.resources

**설명**

**resources**는 이 컨테이너에 필요한 컴퓨팅 리소스입니다. 자세한 내용은 <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

**유형**

**object**

| 속성              | 유형                | 설명                                                                                                                                                                                                                                                                                       |
|-----------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>limits</b>   | integer-or-string | limits는 허용되는 최대 컴퓨팅 리소스 양을 설명합니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a>                                                                    |
| <b>requests</b> | integer-or-string | requests는 필요한 최소 컴퓨팅 리소스 양을 설명합니다. 컨테이너에 대한 요청이 생략된 경우 해당 요청은 명시적으로 지정되고 그렇지 않으면 구현 정의 값으로 기본 설정됩니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> |

### 31.7.1.2.4. .spec.agent.ipfix

**설명**

**IPFIX**는 "agent.type" 속성이 "IPFIX"로 설정된 경우 IPFIX 기반 흐름 보고기와 관련된 설정을 설명합니다.

## 유형

## object

| 속성                            | 유형             | 설명                                                                                                                                                                                                             |
|-------------------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>cacheActiveTimeout</b>     | <b>string</b>  | cacheActiveTimeout은 보고자가 보내기 전에 흐름을 집계하는 최대 기간입니다.                                                                                                                                                             |
| <b>cacheMaxFlows</b>          | <b>integer</b> | cacheMaxFlows는 집계의 최대 흐름 수입니다;에 도달하면 보고자가 흐름을 보냅니다.                                                                                                                                                            |
| <b>clusterNetworkOperator</b> | <b>object</b>  | clusterNetworkOperator는 사용 가능한 경우 OpenShift Container Platform Cluster Network Operator와 관련된 설정을 정의합니다.                                                                                                        |
| <b>forceSampleAll</b>         | <b>boolean</b> | forceSampleAll는 IPFIX 기반 흐름 보고기에서 샘플링을 비활성화할 수 있습니다. 클러스터 불안정을 생성할 수 있으므로 IPFIX로 모든 트래픽을 샘플링하지 않는 것이 좋습니다. 실제로 이 플래그를 true로 설정합니다. at your own risk 입니다. true로 설정하면 "sampling" 값이 무시됩니다.                       |
| <b>ovnKubernetes</b>          | <b>object</b>  | OVNKubernetes는 사용 가능한 경우 OVN-Kubernetes CNI의 설정을 정의합니다. 이 구성은 OpenShift Container Platform 없이 OVN의 IPFIX 내보내기를 사용할 때 사용됩니다. OpenShift Container Platform을 사용하는 경우 대신 <b>clusterNetworkOperator</b> 속성을 참조하십시오. |
| <b>sampling</b>               | <b>integer</b> | 샘플링은 보고자의 샘플링 비율입니다. 100은 100에 하나의 흐름이 전송되었음을 의미합니다. 클러스터 안정성을 보장하기 위해 2 미만의 값을 설정할 수 없습니다. 클러스터 안정성에 영향을 줄 수 있는 모든 패킷을 샘플링하려면 "forceSampleAll"를 참조하십시오. 또는 IPFIX 대신 eBPF 에이전트를 사용할 수 있습니다.                    |

### 31.7.1.2.5. `.spec.agent.ipfix.clusterNetworkOperator`

**설명**

`clusterNetworkOperator`는 사용 가능한 경우 **OpenShift Container Platform Cluster Network Operator**와 관련된 설정을 정의합니다.

**유형**

`object`

| 속성                     | 유형                  | 설명                   |
|------------------------|---------------------|----------------------|
| <code>namespace</code> | <code>string</code> | 구성 맵을 배포할 네임스페이스입니다. |

### 31.7.1.2.6. `.spec.agent.ipfix.ovnKubernetes`

**설명**

`OVNKubernetes`는 사용 가능한 경우 **OVN-Kubernetes CNI**의 설정을 정의합니다. 이 구성은 **OpenShift Container Platform** 없이 **OVN의 IPFIX** 내보내기를 사용할 때 사용됩니다. **OpenShift Container Platform**을 사용하는 경우 대신 `clusterNetworkOperator` 속성을 참조하십시오.

**유형**

`object`

| 속성                         | 유형                  | 설명                                                                          |
|----------------------------|---------------------|-----------------------------------------------------------------------------|
| <code>containerName</code> | <code>string</code> | <code>containerName</code> 은 IPFIX에 대해 구성할 컨테이너의 이름을 정의합니다.                 |
| <code>daemonSetName</code> | <code>string</code> | <code>daemonSetName</code> 은 OVN-Kubernetes Pod를 제어하는 DaemonSet의 이름을 정의합니다. |
| <code>namespace</code>     | <code>string</code> | OVN-Kubernetes Pod가 배포되는 네임스페이스입니다.                                         |

### 31.7.1.3. `.spec.consolePlugin`

**설명**

`consolePlugin`은 사용 가능한 경우 **OpenShift Container Platform 콘솔 플러그인**과 관련된 설정을 정의합니다.



유형

**object**

필수 항목

•

**register**

| 속성                     | 유형             | 설명                                                   |
|------------------------|----------------|------------------------------------------------------|
| <b>autoscaler</b>      | <b>object</b>  | 플러그인 배포에 대해 설정할 수평 Pod 자동 스케일러 사양의 자동 스케일러 사양입니다.    |
| <b>imagePullPolicy</b> | <b>string</b>  | imagePullPolicy는 위에서 정의한 이미지의 Kubernetes 가져오기 정책입니다. |
| <b>logLevel</b>        | <b>string</b>  | 콘솔 플러그인 백엔드의 로그 수준                                   |
| <b>port</b>            | <b>integer</b> | port는 plugin 서비스 포트입니다.                              |
| <b>portNaming</b>      | <b>object</b>  | portNaming은 포트-투-서비스 이름 변환의 구성을 정의합니다.               |
| <b>quickFilters</b>    | <b>array</b>   | quickFilters는 Console 플러그인의 빠른 필터 사전 설정을 구성합니다.      |
| <b>quickFilters[]</b>  | <b>object</b>  | QuickFilter는 Console의 빠른 필터에 대한 사전 설정 구성을 정의합니다.     |

| 속성               | 유형             | 설명                                                                                                                                                                                                                                                                                                                                                            |
|------------------|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>register</b>  | <b>boolean</b> | 레지스터를 사용하면 true로 설정하면 제공된 콘솔 플러그인을 OpenShift Container Platform Console Operator에 자동으로 등록할 수 있습니다. false로 설정하면 <b>oc patch console.operator.OpenShift Container Platform.io cluster --type=json -p '{"op": "path": "/spec/plugins/-", "netobserv-plugin"}'</b> 을 편집하여 <b>console.operator.OpenShift Container Platform.io/cluster</b> 를 편집하여 수동으로 등록할 수 있습니다. |
| <b>replicas</b>  | <b>integer</b> | 복제는 시작할 복제본(Pod) 수를 정의합니다.                                                                                                                                                                                                                                                                                                                                    |
| <b>resources</b> | <b>object</b>  | 이 컨테이너에 필요한 컴퓨팅 리소스 측면에서 리소스입니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a>                                                                                                                                          |

### 31.7.1.3.1. .spec.consolePlugin.autoscaler

#### 설명

플러그인 배포에 대해 설정할 수 있는 Pod 자동 스케일러 사양의 자동 스케일러 사양입니다. **HorizontalPodAutoscaler** 문서 (자동 확장/v2)를 참조하십시오.

### 31.7.1.3.2. .spec.consolePlugin.portNaming

#### 설명

**portNaming**은 포트-투-서비스 이름 변환의 구성을 정의합니다.

#### 유형

**object**

| 속성               | 유형             | 설명                                                                   |
|------------------|----------------|----------------------------------------------------------------------|
| <b>enable</b>    | <b>boolean</b> | 콘솔 플러그인 포트-서비스 이름 변환을 활성화                                            |
| <b>portNames</b> | 개체(문자열)        | portNames는 콘솔에서 사용할 추가 포트 이름을 정의합니다(예: portNames: {"3100": "loki"}). |

### 31.7.1.3.3. `.spec.consolePlugin.quickFilters`

#### 설명

`quickFilters`는 **Console** 플러그인의 빠른 필터 사전 설정을 구성합니다.

#### 유형

**array**

### 31.7.1.3.4. `.spec.consolePlugin.quickFilters[]`

#### 설명

**QuickFilter**는 **Console**의 빠른 필터에 대한 사전 설정 구성을 정의합니다.

#### 유형

**object**

#### 필수 항목

- **filter**
- **name**

| 속성             | 유형             | 설명                                                       |
|----------------|----------------|----------------------------------------------------------|
| <b>default</b> | <b>boolean</b> | <code>default</code> 는 이 필터가 기본적으로 활성화되어야 하는지 여부를 정의합니다. |

| 속성            | 유형      | 설명                                                                                                                                  |
|---------------|---------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>filter</b> | 개체(문자열) | Filter는 이 필터를 선택할 때 설정할 키와 값 집합입니다. 각 키는 쉼표로 구분된 문자열을 사용하여 값 목록(예: filter: {"src_namespace": "namespace1,namespace2"})과 관련될 수 있습니다. |
| <b>name</b>   | string  | Console에 표시될 필터의 이름                                                                                                                 |

### 31.7.1.3.5. .spec.consolePlugin.resources

#### 설명

이 컨테이너에 필요한 컴퓨팅 리소스 측면에서 리소스입니다. 자세한 내용은 <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

#### 유형

**object**

| 속성              | 유형                | 설명                                                                                                                                                                                                                                                                                       |
|-----------------|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>limits</b>   | integer-or-string | limits는 허용되는 최대 컴퓨팅 리소스 양을 설명합니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a>                                                                    |
| <b>requests</b> | integer-or-string | requests는 필요한 최소 컴퓨팅 리소스 양을 설명합니다. 컨테이너에 대한 요청이 생략된 경우 해당 요청은 명시적으로 지정되고 그렇지 않으면 구현 정의 값으로 기본 설정됩니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> |

### 31.7.1.4. .spec.exporters

#### 설명

내보내기는 사용자 정의 사용 또는 저장을 위해 추가 선택적 내보내기를 정의합니다. 이는 실험적인 기능입니다. 현재 KAFKA 내보내기만 사용할 수 있습니다.

유형

**array**

#### 31.7.1.4.1. `.spec.exporters[]`

설명

**FlowCollectorExporter**는 보강된 흐름을 보낼 추가 내보내기를 정의합니다.

유형

**object**

필수 항목

- **type**

| 속성           | 유형            | 설명                                                    |
|--------------|---------------|-------------------------------------------------------|
| <b>kafka</b> | <b>object</b> | Kafka는 보강된 흐름을 보낼 kafka 구성(address, topic...)을 설명합니다. |
| <b>type</b>  | <b>string</b> | type은 내보내기 유형을 선택합니다. 현재 "KAFKA"만 사용할 수 있습니다.         |

#### 31.7.1.4.2. `.spec.exporters[].kafka`

설명

보강된 흐름을 전송하기 위해 주소 또는 주제와 같은 **kafka** 구성에 대해 설명합니다.

유형

**object**

필수 항목

- 주소
- 주제

| 속성  | 유형     | 설명                                                                                                                                                               |
|-----|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 주소  | string | Kafka 서버의 주소                                                                                                                                                     |
| tls | object | TLS 클라이언트 구성입니다. TLS를 사용하는 경우 주소가 TLS에 사용되는 Kafka 포트(일반적으로 9093)와 일치하는지 확인합니다. eBPF 에이전트를 사용하는 경우 Kafka 인증서를 에이전트 네임스페이스에서 복사해야 합니다(기본적으로 netobserv-privileged). |
| 주제  | string | 사용할 Kafka 주제입니다. NetObserv가 이를 생성하지 않아야 합니다.                                                                                                                     |

### 31.7.1.4.3. .spec.exporters[].kafka.tls

#### 설명

**TLS 클라이언트 구성입니다. TLS를 사용하는 경우 주소가 TLS에 사용되는 Kafka 포트(일반적으로 9093)와 일치하는지 확인합니다. eBPF 에이전트를 사용하는 경우 Kafka 인증서를 에이전트 네임스페이스에서 복사해야 합니다(기본적으로 netobserv-privileged).**

#### 유형

**object**

| 속성                 | 유형      | 설명                                                                                       |
|--------------------|---------|------------------------------------------------------------------------------------------|
| caCert             | object  | cacert는 인증 기관의 인증서 참조를 정의합니다.                                                            |
| enable             | boolean | TLS 활성화                                                                                  |
| insecureSkipVerify | boolean | insecureSkipVerify를 사용하면 서버 인증서에 대한 클라이언트 측 확인을 건너뛸 수 있습니다. true로 설정되면 CACert 필드가 무시됩니다. |
| userCert           | object  | UserCert는 mTLS에 사용되는 사용자 인증서 참조를 정의합니다(일반 단방향 TLS를 사용할 때 무시 가능)                          |

### 31.7.1.4.4. .spec.exporters[].kafka.tls.caCert

**설명**

**cacert**는 인증 기관의 인증서 참조를 정의합니다.

**유형**

**object**

| 속성              | 유형            | 설명                                                                         |
|-----------------|---------------|----------------------------------------------------------------------------|
| <b>certFile</b> | <b>string</b> | certfile은 구성 맵 / Secret 내에서 인증서 파일 이름의 경로를 정의합니다.                          |
| <b>certKey</b>  | <b>string</b> | certKey는 구성 맵 / Secret 내에서 인증서 개인 키 파일 이름의 경로를 정의합니다. 키가 필요하지 않은 경우 생략합니다. |
| <b>name</b>     | <b>string</b> | 인증서가 포함된 구성 맵 또는 보안의 이름                                                    |
| <b>type</b>     | <b>string</b> | 인증서 참조의 경우 유형: 구성 맵 또는 secret                                              |

**31.7.1.4.5. .spec.exporters[].kafka.tls.userCert****설명**

**UserCert**는 mTLS에 사용되는 사용자 인증서 참조를 정의합니다(일반 단방향 TLS를 사용할 때 무시 가능)

**유형**

**object**

| 속성              | 유형            | 설명                                                                         |
|-----------------|---------------|----------------------------------------------------------------------------|
| <b>certFile</b> | <b>string</b> | certfile은 구성 맵 / Secret 내에서 인증서 파일 이름의 경로를 정의합니다.                          |
| <b>certKey</b>  | <b>string</b> | certKey는 구성 맵 / Secret 내에서 인증서 개인 키 파일 이름의 경로를 정의합니다. 키가 필요하지 않은 경우 생략합니다. |

| 속성          | 유형            | 설명                            |
|-------------|---------------|-------------------------------|
| <b>name</b> | <b>string</b> | 인증서가 포함된 구성 맵 또는 보안의 이름       |
| <b>type</b> | <b>string</b> | 인증서 참조의 경우 유형: 구성 맵 또는 secret |

### 31.7.1.5. .spec.kafka

#### 설명

**Kafka 구성으로 Kafka를 흐름 컬렉션 파이프라인의 일부로 브로커로 사용할 수 있습니다. "spec.deploymentModel"이 "KAFKA"인 경우 사용할 수 있습니다.**

#### 유형

**object**

#### 필수 항목

- 주소
- 주제

| 속성         | 유형            | 설명                                                                                                                                                               |
|------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 주소         | <b>string</b> | Kafka 서버의 주소                                                                                                                                                     |
| <b>tls</b> | <b>object</b> | TLS 클라이언트 구성입니다. TLS를 사용하는 경우 주소가 TLS에 사용되는 Kafka 포트(일반적으로 9093)와 일치하는지 확인합니다. eBPF 에이전트를 사용하는 경우 Kafka 인증서를 에이전트 네임스페이스에서 복사해야 합니다(기본적으로 netobserv-privileged). |
| 주제         | <b>string</b> | 사용할 Kafka 주제입니다. NetObserv가 이를 생성하지 않아야 합니다.                                                                                                                     |

### 31.7.1.5.1. .spec.kafka.tls

#### 설명



**TLS 클라이언트 구성입니다. TLS를 사용하는 경우 주소가 TLS에 사용되는 Kafka 포트(일반적으로 9093)와 일치하는지 확인합니다. eBPF 에이전트를 사용하는 경우 Kafka 인증서를 에이전트 네임스페이스에서 복사해야 합니다(기본적으로 netobserv-privileged).**

유형

**object**

| 속성                        | 유형             | 설명                                                                                       |
|---------------------------|----------------|------------------------------------------------------------------------------------------|
| <b>caCert</b>             | <b>object</b>  | cacert는 인증 기관의 인증서 참조를 정의합니다.                                                            |
| <b>enable</b>             | <b>boolean</b> | TLS 활성화                                                                                  |
| <b>insecureSkipVerify</b> | <b>boolean</b> | insecureSkipVerify를 사용하면 서버 인증서에 대한 클라이언트 측 확인을 건너뛸 수 있습니다. true로 설정되면 CACert 필드가 무시됩니다. |
| <b>userCert</b>           | <b>object</b>  | UserCert는 mTLS에 사용되는 사용자 인증서 참조를 정의합니다(일반 단방향 TLS를 사용할 때 무시 가능)                          |

### 31.7.1.5.2. .spec.kafka.tls.caCert

설명

**cacert는 인증 기관의 인증서 참조를 정의합니다.**

유형

**object**

| 속성              | 유형            | 설명                                                                         |
|-----------------|---------------|----------------------------------------------------------------------------|
| <b>certFile</b> | <b>string</b> | certfile은 구성 맵 / Secret 내에서 인증서 파일 이름의 경로를 정의합니다.                          |
| <b>certKey</b>  | <b>string</b> | certKey는 구성 맵 / Secret 내에서 인증서 개인 키 파일 이름의 경로를 정의합니다. 키가 필요하지 않은 경우 생략합니다. |

| 속성          | 유형            | 설명                            |
|-------------|---------------|-------------------------------|
| <b>name</b> | <b>string</b> | 인증서가 포함된 구성 맵 또는 보안의 이름       |
| <b>type</b> | <b>string</b> | 인증서 참조의 경우 유형: 구성 맵 또는 secret |

### 31.7.1.5.3. .spec.kafka.tls.userCert

**설명**

*UserCert*는 mTLS에 사용되는 사용자 인증서 참조를 정의합니다(일반 단방향 TLS를 사용할 때 무시 가능)

**유형**

**object**

| 속성              | 유형            | 설명                                                                         |
|-----------------|---------------|----------------------------------------------------------------------------|
| <b>certFile</b> | <b>string</b> | certfile은 구성 맵 / Secret 내에서 인증서 파일 이름의 경로를 정의합니다.                          |
| <b>certKey</b>  | <b>string</b> | certKey는 구성 맵 / Secret 내에서 인증서 개인 키 파일 이름의 경로를 정의합니다. 키가 필요하지 않은 경우 생략합니다. |
| <b>name</b>     | <b>string</b> | 인증서가 포함된 구성 맵 또는 보안의 이름                                                    |
| <b>type</b>     | <b>string</b> | 인증서 참조의 경우 유형: 구성 맵 또는 secret                                              |

### 31.7.1.6. .spec.loki

**설명**

로키, 흐름 저장소, 클라이언트 설정

**유형**

**object**

| 속성                  | 유형             | 설명                                                                                                                                                                                                                                                             |
|---------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>authToken</b>    | <b>string</b>  | authToken은 로키 DISABLED에 인증하는 토큰을 얻는 방법을 설명합니다. HOST는 로컬 Pod 서비스 계정을 사용하여 로컬 Pod 서비스 계정을 사용하여 사용자 토큰을 전달합니다. 이 모드에서 프로세서와 같은 사용자 요청을 수신하지 않는 Pod는 로컬 Pod 서비스 계정을 사용합니다. 호스트 모드와 유사합니다.                                                                          |
| <b>batchSize</b>    | <b>integer</b> | batchSize는 보내기 전에 누적할 최대 배치 크기(바이트 단위)입니다.                                                                                                                                                                                                                     |
| <b>batchWait</b>    | <b>string</b>  | batchWait는 배치를 전송하기 전에 대기하는 최대 시간입니다.                                                                                                                                                                                                                          |
| <b>maxBackoff</b>   | <b>string</b>  | maxBackoff는 재시도 사이의 클라이언트 연결의 최대 백오프 시간입니다.                                                                                                                                                                                                                    |
| <b>maxRetries</b>   | <b>integer</b> | maxRetries는 클라이언트 연결의 최대 재시도 횟수입니다.                                                                                                                                                                                                                            |
| <b>minBackoff</b>   | <b>string</b>  | minBackoff는 재시도 사이의 클라이언트 연결 초기 백오프 시간입니다.                                                                                                                                                                                                                     |
| <b>querierUrl</b>   | <b>string</b>  | querierURL은 로키 쿼리어 서비스의 주소를 지정합니다. 비어 있는 경우 (Roki ingester 및 querier가 동일한 서버에 있는 것으로 가정). + [IMPORTANT] ===== lostki Operator를 사용하여 로키를 설치한 경우 로키에 대한 콘솔 액세스를 중단시킬 수 있으므로 <b>querierUrl</b> 을 사용하지 않는 것이 좋습니다. 다른 유형의 로키 설치를 사용하여 로키를 설치한 경우, 적용되지 않습니다. ===== |
| <b>staticLabels</b> | 개체(문자열)        | staticLabels는 각 흐름에 설정할 공통 레이블의 맵입니다.                                                                                                                                                                                                                          |

| 속성               | 유형            | 설명                                                                                                                                                                 |
|------------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>statusUrl</b> | <b>string</b> | statusURL은 Loki /ready /metrics /config 엔드 포인트의 주소를 지정합니다. 이 URL은 Loki querier URL 과 다릅니다. 비어 있는 경우 QuerierURL 값이 사용됩니다. 이는 오류 메시지와 프런트 엔드의 일부 컨텍스트를 표시하는 데 유용합니다. |
| <b>tenantID</b>  | <b>string</b> | tenantId는 각 요청의 테넌트를 식별하는 Loki X-Scope-OrgID입니다. instanceSpec이 지정된 경우 무시됩니다.                                                                                       |
| <b>timeout</b>   | <b>string</b> | timeout은 최대 시간 연결 / 요청 제한 A Timeout of zero means no timeout입니다.                                                                                                   |
| <b>tls</b>       | <b>object</b> | TLS 클라이언트 구성입니다.                                                                                                                                                   |
| <b>url</b>       | <b>string</b> | URL은 흐름을 푸시하는 기존 로키 서비스의 주소입니다.                                                                                                                                    |

**31.7.1.6.1. .spec.loki.tls**

**설명**

**TLS 클라이언트 구성입니다.**

**유형**

**object**

| 속성                        | 유형             | 설명                                                                                       |
|---------------------------|----------------|------------------------------------------------------------------------------------------|
| <b>caCert</b>             | <b>object</b>  | cacert는 인증 기관의 인증서 참조를 정의합니다.                                                            |
| <b>enable</b>             | <b>boolean</b> | TLS 활성화                                                                                  |
| <b>insecureSkipVerify</b> | <b>boolean</b> | insecureSkipVerify를 사용하면 서버 인증서에 대한 클라이언트 측 확인을 건너뛸 수 있습니다. true로 설정되면 CACert 필드가 무시됩니다. |

| 속성              | 유형            | 설명                                                              |
|-----------------|---------------|-----------------------------------------------------------------|
| <b>userCert</b> | <b>object</b> | UserCert는 mTLS에 사용되는 사용자 인증서 참조를 정의합니다(일반 단방향 TLS를 사용할 때 무시 가능) |

### 31.7.1.6.2. .spec.loki.tls.caCert

#### 설명

**cacert**는 인증 기관의 인증서 참조를 정의합니다.

#### 유형

**object**

| 속성              | 유형            | 설명                                                                         |
|-----------------|---------------|----------------------------------------------------------------------------|
| <b>certFile</b> | <b>string</b> | certfile은 구성 맵 / Secret 내에서 인증서 파일 이름의 경로를 정의합니다.                          |
| <b>certKey</b>  | <b>string</b> | certKey는 구성 맵 / Secret 내에서 인증서 개인 키 파일 이름의 경로를 정의합니다. 키가 필요하지 않은 경우 생략합니다. |
| <b>name</b>     | <b>string</b> | 인증서가 포함된 구성 맵 또는 보안의 이름                                                    |
| <b>type</b>     | <b>string</b> | 인증서 참조의 경우 유형: 구성 맵 또는 secret                                              |

### 31.7.1.6.3. .spec.loki.tls.userCert

#### 설명

**UserCert**는 mTLS에 사용되는 사용자 인증서 참조를 정의합니다(일반 단방향 TLS를 사용할 때 무시 가능)

#### 유형

**object**

| 속성              | 유형            | 설명                                                                         |
|-----------------|---------------|----------------------------------------------------------------------------|
| <b>certFile</b> | <b>string</b> | certfile은 구성 맵 / Secret 내에서 인증서 파일 이름의 경로를 정의합니다.                          |
| <b>certKey</b>  | <b>string</b> | certKey는 구성 맵 / Secret 내에서 인증서 개인 키 파일 이름의 경로를 정의합니다. 키가 필요하지 않은 경우 생략합니다. |
| <b>name</b>     | <b>string</b> | 인증서가 포함된 구성 맵 또는 보안의 이름                                                    |
| <b>type</b>     | <b>string</b> | 인증서 참조의 경우 유형: 구성 맵 또는 secret                                              |

### 31.7.1.7. .spec.processor

#### 설명

프로세서는 에이전트에서 흐름을 수신하고 보강된 구성 요소의 설정을 정의하고 이를 로키 지속성 계층으로 전달합니다.

#### 유형

*object*

| 속성                      | 유형             | 설명                                                                                                                                            |
|-------------------------|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>debug</b>            | <b>object</b>  | 디버그를 사용하면 흐름 프로세서의 내부 구성의 일부 측면을 설정할 수 있습니다. 이 섹션은 GOGC 및 GOMAXPROCS env vars와 같은 디버깅 및 세분화된 성능 최적화만을 위한 것입니다. 값을 설정하면 사용자가 자신의 위험을 감수해야 합니다. |
| <b>dropUnusedFields</b> | <b>boolean</b> | dropUnusedFields를 사용하면 true로 설정하면 OVS에서 사용하지 않는 것으로 알려진 필드를 삭제하여 스토리지 공간을 절약할 수 있습니다.                                                         |
| <b>enableKubeProbes</b> | <b>boolean</b> | enableKubeProbes는 Kubernetes 활동성 및 준비 상태 프로브를 활성화 또는 비활성화하는 플래그입니다.                                                                           |

| 속성                                | 유형             | 설명                                                                                                                                  |
|-----------------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>healthPort</b>                 | <b>integer</b> | healthPort는 상태 점검 API를 노출하는 Pod의 수집기 HTTP 포트입니다.                                                                                    |
| <b>imagePullPolicy</b>            | <b>string</b>  | imagePullPolicy는 위에서 정의한 이미지의 Kubernetes 가져오기 정책입니다.                                                                                |
| <b>kafkaConsumerAutoscaler</b>    | <b>object</b>  | Kafka 메시지를 사용하는 flowlogs-pipeline-transformer에 대해 설정하기 위해 수평 Pod 자동 스케일러의 kafkaConsumerAutoscaler 사양입니다. Kafka가 비활성화되면 이 설정이 무시됩니다. |
| <b>kafkaConsumerBatchSize</b>     | <b>integer</b> | kafkaConsumerBatchSize는 브로커에게 최대 배치 크기를 바이트 단위로 나타냅니다. Kafka를 사용하지 않는 경우 무시됩니다. 기본값: 10MB.                                          |
| <b>kafkaConsumerQueueCapacity</b> | <b>integer</b> | kafkaConsumerQueueCapacity는 Kafka 소비자 클라이언트에 사용되는 내부 메시지 큐의 용량을 정의합니다. Kafka를 사용하지 않는 경우 무시됩니다.                                     |
| <b>kafkaConsumerReplicas</b>      | <b>integer</b> | kafkaConsumerReplicas는 Kafka 메시지를 사용하는 flowlogs-pipeline-transformer에 대해 시작할 복제본 수(Pod)를 정의합니다. Kafka가 비활성화되면 이 설정이 무시됩니다.          |
| <b>logLevel</b>                   | <b>string</b>  | 수집기 런타임의 로그 수준                                                                                                                      |
| <b>메트릭</b>                        | <b>object</b>  | 메트릭은 메트릭에 대한 프로세서 구성을 정의합니다.                                                                                                        |
| <b>port</b>                       | <b>integer</b> | 규칙에 따라 일부 값은 1024 미만이어야 하며 4789,6081,500 및 4500의 값과 일치하지 않아야 합니다.                                                                   |
| <b>profilePort</b>                | <b>integer</b> | profilePort를 사용하면 이 포트를 수신하는 Go pprof 프로파일러를 설정할 수 있습니다.                                                                            |

| 속성               | 유형            | 설명                                                                                                                                                                                                                     |
|------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>resources</b> | <b>object</b> | resources는 이 컨테이너에 필요한 컴퓨팅 리소스입니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> |

### 31.7.1.7.1. .spec.processor.debug

#### 설명

디버그를 사용하면 흐름 프로세서의 내부 구성의 일부 측면을 설정할 수 있습니다. 이 섹션은 **GOGC** 및 **GOMAXPROCS env vars**와 같은 디버깅 및 세분화된 성능 최적화만을 위한 것입니다. 값을 설정하면 사용자가 자신의 위험을 감수해야 합니다.

#### 유형

**object**

| 속성         | 유형      | 설명                                                                                                                                                                      |
|------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>env</b> | 개체(문자열) | env는 사용자 지정 환경 변수를 NetObserv 에이전트에 전달할 수 있습니다. edge debug 및 지원 시나리오에서만 유용하므로 FlowCollector 설명자의 일부로 공개적으로 노출되지 않는 GOGC 및 GOMAXPROCS와 같은 매우 포괄적인 성능 튜닝 옵션을 전달하는 데 유용합니다. |

### 31.7.1.7.2. .spec.processor.kafkaConsumerAutoscaler

#### 설명

**Kafka** 메시지를 사용하는 **flowlogs-pipeline-transformer**에 대해 설정하기 위해 수평 **Pod** 자동 스케일러의 **kafkaConsumerAutoscaler** 사양입니다. **Kafka**가 비활성화되면 이 설정이 무시됩니다. **HorizontalPodAutoscaler** 문서 (자동 확장/v2)를 참조하십시오.

### 31.7.1.7.3. .spec.processor.metrics

#### 설명

메트릭은 메트릭에 대한 프로세서 구성을 정의합니다.



유형

**object**

| 속성                | 유형            | 설명                                      |
|-------------------|---------------|-----------------------------------------|
| <b>ignoreTags</b> | 배열(문자열)       | ignoreTags는 무시할 메트릭을 지정하는 태그 목록입니다.     |
| <b>server</b>     | <b>object</b> | Prometheus 스크래퍼에 대한 metricsServer 끝점 구성 |

**31.7.1.7.4. .spec.processor.metrics.server**

설명

**Prometheus 스크래퍼에 대한 metricsServer 끝점 구성**

유형

**object**

| 속성          | 유형             | 설명                 |
|-------------|----------------|--------------------|
| <b>port</b> | <b>integer</b> | prometheus HTTP 포트 |
| <b>tls</b>  | <b>object</b>  | TLS 구성입니다.         |

**31.7.1.7.5. .spec.processor.metrics.server.tls**

설명

**TLS 구성입니다.**

유형

**object**

| 속성  | 유형            | 설명         |
|-----|---------------|------------|
| 제공됨 | <b>object</b> | TLS 구성입니다. |

| 속성          | 유형            | 설명                                                                                                                                             |
|-------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b> | <b>string</b> | FTP를 사용하여 OpenShift Container Platform 자동 생성 인증서를 사용하도록 TLS 구성 "DISABLED"를 선택하여 엔드포인트에 TLS를 구성하지 않고 "PROVIDED"를 선택하여 인증서 파일과 키 파일을 수동으로 제공합니다. |

**31.7.1.7.6. .spec.processor.metrics.server.tls.provided**

**설명**

**TLS 구성입니다.**

**유형**

**object**

| 속성              | 유형            | 설명                                                                         |
|-----------------|---------------|----------------------------------------------------------------------------|
| <b>certFile</b> | <b>string</b> | certfile은 구성 맵 / Secret 내에서 인증서 파일 이름의 경로를 정의합니다.                          |
| <b>certKey</b>  | <b>string</b> | certKey는 구성 맵 / Secret 내에서 인증서 개인 키 파일 이름의 경로를 정의합니다. 키가 필요하지 않은 경우 생략합니다. |
| <b>name</b>     | <b>string</b> | 인증서가 포함된 구성 맵 또는 보안의 이름                                                    |
| <b>type</b>     | <b>string</b> | 인증서 참조의 경우 유형: 구성 맵 또는 secret                                              |

**31.7.1.7.7. .spec.processor.resources**

**설명**

**resources**는 이 컨테이너에 필요한 컴퓨팅 리소스입니다. 자세한 내용은 <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

**유형**

**object**

| 속성              | 유형                       | 설명                                                                                                                                                                                                                                                                                       |
|-----------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>limits</b>   | <b>integer-or-string</b> | limits는 허용되는 최대 컴퓨팅 리소스 양을 설명합니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a>                                                                    |
| <b>requests</b> | <b>integer-or-string</b> | requests는 필요한 최소 컴퓨팅 리소스 양을 설명합니다. 컨테이너에 대한 요청이 생략된 경우 해당 요청은 명시적으로 지정되고 그렇지 않으면 구현 정의 값으로 기본 설정됩니다. 자세한 내용은 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> |

**31.7.1.8. .status****설명**

**FlowCollectorStatus**가 관찰된 **FlowCollectorStatus**를 정의합니다.

**유형****object****필수 항목**

- **conditions**

| 속성                | 유형           | 설명                                           |
|-------------------|--------------|----------------------------------------------|
| <b>conditions</b> | <b>array</b> | conditions는 오브젝트 상태에 대한 사용 가능한 최신 관찰을 나타냅니다. |

| 속성                  | 유형            | 설명                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>conditions[]</b> | <b>object</b> | <p>condition에는 이 API 리소스의 현재 상태의 한 측면에 대한 세부 정보가 포함되어 있습니다. --- 이 구조는 필드 경로 <code>.status.conditions</code>에서 배열로 직접 사용하기 위한 것입니다. 예를 들어, <code>foo</code>의 현재 상태에 대한 관찰을 나타냅니다. // <code>known .status.conditions.type</code>은: <code>"Available", "Progressing", "Degraded" // +patchMergeKey=type // +patchStrategy=merge // +listType=map=map.</code></p> <p><b>Conditions:Conditions: 비어 있는" patchStrategy:"merge" patchMergeKey:"type" protobuf:"bytes,1,rep,name=conditions" // 기타 필드 }</b></p> |
| <b>namespace</b>    | <b>string</b> | <p>콘솔 플러그인 및 <code>flowlogs-pipeline</code>이 배포된 네임스페이스입니다.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                        |

**31.7.1.8.1. .status.conditions**

설명

**conditions**는 오브젝트 상태에 대한 사용 가능한 최신 관찰을 나타냅니다.

유형

**array**

**31.7.1.8.2. .status.conditions[]**

설명

**condition**에는 이 API 리소스의 현재 상태의 한 측면에 대한 세부 정보가 포함되어 있습니다. --- 이 구조는 필드 경로 `.status.conditions`에서 배열로 직접 사용하기 위한 것입니다. 예를 들어, `foo`의 현재 상태에 대한 관찰을 나타냅니다. // `known .status.conditions.type`은: `"Available", "Progressing", "Degraded" // +patchMergeKey=type // +patchStrategy=merge // +listType=map=map.` **Conditions:Conditions: 비어 있는" patchStrategy:"merge" patchMergeKey:"type" protobuf:"bytes,1,rep,name=conditions" // 기타 필드 }**

유형

**object**

## 필수 항목

- ***lastTransitionTime***
- ***message***
- ***reason***
- ***status***
- ***type***

| 속성                        | 유형             | 설명                                                                                                                                                                                     |
|---------------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>lastTransitionTime</b> | <b>string</b>  | lastTransitionTime은 하나의 상태에서 다른 상태로 전환된 조건이 마지막 시간입니다. 기본 조건이 변경될 때여야 합니다. 이를 모르는 경우 API 필드가 변경된 시간을 사용할 수 있습니다.                                                                       |
| <b>message</b>            | <b>string</b>  | message는 전환에 대한 세부 정보를 나타내는 사람이 읽을 수 있는 메시지입니다. 빈 문자열일 수 있습니다.                                                                                                                         |
| <b>observedGeneration</b> | <b>integer</b> | observedGeneration은 조건을 기반으로 설정한 .metadata.generation을 나타냅니다. 예를 들어 .metadata.generation이 현재 12이지만 .status.conditions[x].observedGeneration이 9인 경우 해당 조건은 인스턴스의 현재 상태에 대한 최신 상태가 아닙니다. |

| 속성            | 유형            | 설명                                                                                                                                                                                                                              |
|---------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>reason</b> | <b>string</b> | reason에는 조건의 마지막 전환 이유를 나타내는 프로그래밍 식별자가 포함되어 있습니다. 특정 조건 유형의 생산자는 이 필드에 예상 값과 의미가 정의될 수 있으며 값이 보장된 API로 간주되는지 여부를 정의할 수 있습니다. 값은 CamelCase 문자열이어야 합니다. 이 필드는 비어 있지 않을 수 있습니다.                                                   |
| <b>status</b> | <b>string</b> | 조건의 상태, True, False, Unknown 중 하나입니다.                                                                                                                                                                                           |
| <b>type</b>   | <b>string</b> | CamelCase 또는 foo.example.com/CamelCase. --- Many .condition.type 값은 Available와 같은 리소스 간에 일관되지만 임의의 조건이 유용할 수 있기 때문에 (.node.status.conditions 참조) 분리할 수 있는 기능이 중요합니다. 일치하는 regex는 (dns1123SubdomainFmt/)? (qualifiedNameFmt)입니다. |

### 31.8. 네트워킹 관찰 문제 해결

네트워킹 관찰 문제 해결을 지원하기 위해 몇 가지 문제 해결 작업을 수행할 수 있습니다.

#### 31.8.1. OpenShift Container Platform 콘솔에서 네트워킹 트래픽 메뉴 항목 구성

네트워킹 트래픽 메뉴 항목이 OpenShift Container Platform 콘솔의 **Observe** 메뉴에 나열되지 않은 경우 OpenShift Container Platform 콘솔에서 네트워킹 트래픽 메뉴 항목을 수동으로 구성합니다.

##### 사전 요구 사항

- **OpenShift Container Platform 버전 4.10 이상이 설치되어 있습니다.**

##### 절차

1. 다음 명령을 실행하여 **spec.consolePlugin.register** 필드가 **true** 로 설정되어 있는지 확인합니다.

```
$ oc -n netobserv get flowcollector cluster -o yaml
```

출력 예

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  consolePlugin:
    register: false
```

2. 선택 사항: **Console Operator** 구성을 수동으로 편집하여 **netobserv-plugin** 플러그인을 추가합니다.

```
$ oc edit console.operator.openshift.io cluster
```

출력 예

```
...
spec:
  plugins:
  - netobserv-plugin
...
```

3. 선택 사항: 다음 명령을 실행하여 **spec.consolePlugin.register** 필드를 **true** 로 설정합니다.

```
$ oc -n netobserv edit flowcollector cluster -o yaml
```

출력 예

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
```

```

name: cluster
spec:
  consolePlugin:
    register: true

```

4.

다음 명령을 실행하여 콘솔 포드의 상태가 실행 중인지 확인합니다.

```
$ oc get pods -n openshift-console -l app=console
```

5.

다음 명령을 실행하여 콘솔 Pod를 다시 시작합니다.

```
$ oc delete pods -n openshift-console -l app=console
```

6.

브라우저 캐시 및 기록을 지웁니다.

7.

다음 명령을 실행하여 Network Observability 플러그인 Pod의 상태를 확인합니다.

```
$ oc get pods -n netobserv -l app=netobserv-plugin
```

출력 예

```

NAME                                READY STATUS RESTARTS AGE
netobserv-plugin-68c7bbb9bb-b69q6  1/1   Running  0      21s

```

8.

다음 명령을 실행하여 Network Observability 플러그인 Pod의 로그를 확인합니다.

```
$ oc logs -n netobserv -l app=netobserv-plugin
```

출력 예



```
time="2022-12-13T12:06:49Z" level=info msg="Starting netobserv-console-plugin
[build version: , build date: 2022-10-21 15:15] at log level info" module=main
time="2022-12-13T12:06:49Z" level=info msg="listening on https://:9001"
module=server
```

**31.8.2. FlowLogs-Pipeline**은 Kafka를 설치한 후 네트워크 흐름을 소비하지 않습니다.

**deploymentModel: KAFKA** 를 사용하여 흐름 수집기를 먼저 배포한 후 Kafka를 배포한 경우 흐름 수집기가 Kafka에 올바르게 연결되지 않을 수 있습니다. **Flowlogs-pipeline**에서 Kafka의 네트워크 흐름을 사용하지 않는 **flow-pipeline pod**를 수동으로 다시 시작합니다.

절차

1.

다음 명령을 실행하여 **flow-pipeline Pod**를 삭제하여 재시작합니다.

```
$ oc delete pods -n netobserv -l app=flowlogs-pipeline-transformer
```

**31.8.3. br-int** 및 **br-ex** 인터페이스에서 네트워크 흐름을 보지 못함

**br-ex** 및 **br-int** 는 OSI 계층 2에서 작동하는 가상 브리지 장치입니다. **eBPF** 에이전트는 각각 IP 및 TCP 수준, 계층 3 및 4에서 작동합니다. **eBPF** 에이전트는 물리적 호스트 또는 가상 Pod 인터페이스와 같은 다른 인터페이스에서 네트워크 트래픽을 처리할 때 **br-ex** 및 **br-int** 를 통해 전달되는 네트워크 트래픽을 캡처할 수 있습니다. **br-ex** 및 **br-int** 에만 연결하도록 **eBPF** 에이전트 네트워크 인터페이스를 제한하면 네트워크 흐름이 표시되지 않습니다.

네트워크 인터페이스를 **br-int** 및 **br-ex** 로 제한하는 인터페이스 또는 **excludeInterfaces** 의 부분을 수동으로 제거합니다.

절차

1.

인터페이스 제거: **[br-int', 'br-ex']** 필드. 이를 통해 에이전트는 모든 인터페이스에서 정보를 가져올 수 있습니다. 또는 **Layer-3** 인터페이스를 지정할 수도 있습니다(예: **eth0**). 다음 명령을 실행합니다.

```
$ oc edit -n netobserv flowcollector.yaml -o yaml
```

출력 예

```

apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
    type: EBPF
  ebpf:
    interfaces: [ 'br-int', 'br-ex' ] 1

```

**1**

네트워크 인터페이스를 지정합니다.

#### 31.8.4. Network Observability 컨트롤러 관리자 Pod가 메모리 부족

CSV(Cluster Service Version)에 패치를 적용하여 Network Observability Operator의 메모리 제한을 늘릴 수 있습니다. 여기서 Network Observability 컨트롤러 관리자 Pod는 메모리가 부족합니다.

##### 절차

1.

다음 명령을 실행하여 CSV를 패치합니다.

```

$ oc -n netobserv patch csv network-observability-operator.v1.0.0 --type='json' -
p='[{"op": "replace",
"path": "/spec/install/spec/deployments/0/spec/template/spec/containers/0/resources/li
mits/memory", value: "1Gi"}]'

```

##### 출력 예

```

clusterserviceversion.operators.coreos.com/network-observability-operator.v1.0.0 patched

```

2.

다음 명령을 실행하여 업데이트된 CSV를 확인합니다.

```

$ oc -n netobserv get csv network-observability-operator.v1.0.0 -o

```

```
jsonpath='{.spec.install.spec.deployments[0].spec.template.spec.containers[0].resources.limits.memory}'  
1Gi
```