



OpenShift Container Platform 4.6

네트워킹

클러스터 네트워킹 구성 및 관리

OpenShift Container Platform 4.6 네트워킹

클러스터 네트워킹 구성 및 관리

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

법적 공지

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Networking.ent file |.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

초록

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

차례

1장. 네트워킹 이해	10
1.1. OPENSIFT CONTAINER PLATFORM DNS	10
1.2. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	10
1.2.1. 경로와 Ingress 비교	10
2장. 호스트에 액세스	12
2.1. 설치 관리자 프로비저닝 인프라 클러스터에서 AMAZON WEB SERVICES의 호스트에 액세스	12
3장. 네트워킹 OPERATOR 개요	13
3.1. CNO(CLUSTER NETWORK OPERATOR)	13
3.2. DNS OPERATOR	13
3.3. INGRESS OPERATOR	13
4장. OPENSIFT 컨테이너 플랫폼의 CLUSTER NETWORK OPERATOR	14
4.1. CNO(CLUSTER NETWORK OPERATOR)	14
4.2. 클러스터 네트워크 구성 보기	14
4.3. CNO(CLUSTER NETWORK OPERATOR) 상태 보기	15
4.4. CNO(CLUSTER NETWORK OPERATOR) 로그 보기	15
4.5. CNO(CLUSTER NETWORK OPERATOR) 구성	15
4.5.1. CNO(Cluster Network Operator) 구성 오브젝트	16
defaultNetwork 오브젝트 구성	17
OpenShift SDN CNI 네트워크 공급자에 대한 구성	17
OVN-Kubernetes CNI 클러스터 네트워크 공급자에 대한 구성	18
kubeProxyConfig 오브젝트 구성	19
4.5.2. CNO(Cluster Network Operator) 구성 예시	19
4.6. 추가 리소스	20
5장. OPENSIFT CONTAINER PLATFORM에서의 DNS OPERATOR	21
5.1. DNS OPERATOR	21
5.2. 기본 DNS보기	21
5.3. DNS 전달 사용	22
5.4. DNS OPERATOR 상태	24
5.5. DNS OPERATOR 로그	24
6장. OPENSIFT CONTAINER PLATFORM에서의 INGRESS OPERATOR	25
6.1. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	25
6.2. INGRESS 구성 자산	25
6.3. INGRESS 컨트롤러 구성 매개변수	25
6.3.1. Ingress 컨트롤러 TLS 보안 프로파일	31
6.3.1.1. TLS 보안 프로파일 이해	31
6.3.1.2. Ingress 컨트롤러의 TLS 보안 프로파일 구성	33
6.3.2. Ingress 컨트롤러 끝점 게시 전략	35
6.4. 기본 INGRESS 컨트롤러 보기	36
6.5. INGRESS OPERATOR 상태 보기	36
6.6. INGRESS 컨트롤러 로그 보기	36
6.7. INGRESS 컨트롤러 상태 보기	37
6.8. INGRESS 컨트롤러 구성	37
6.8.1. 사용자 정의 기본 인증서 설정	37
6.8.2. 사용자 정의 기본 인증서 제거	38
6.8.3. Ingress 컨트롤러 확장	39
6.8.4. 수신 액세스 로깅 구성	40
6.8.5. Ingress 컨트롤러 분할	42

6.8.5.1. 경로 라벨을 사용하여 Ingress 컨트롤러 분할 구성	43
6.8.5.2. 네임스페이스 라벨을 사용하여 Ingress 컨트롤러 분할 구성	43
6.8.6. 내부 로드 밸런서를 사용하도록 Ingress 컨트롤러 구성	44
6.8.7. 클러스터의 기본 Ingress 컨트롤러를 내부로 구성	46
6.8.8. 경로 허용 정책 구성	47
6.8.9. 와일드카드 경로 사용	48
6.8.10. X-Forwarded 헤더 사용	49
사용 사례 예	49
6.8.11. HTTP/2 수신 연결 사용	50
6.9. 추가 리소스	51
7장. 노드 포트 서비스 범위 구성	52
7.1. 사전 요구 사항	52
7.2. 노드 포트 범위 확장	52
7.3. 추가 리소스	52
8장. 베어 메탈 클러스터에서 SCTP(STREAM CONTROL TRANSMISSION PROTOCOL) 사용	54
8.1. OPENSIFT CONTAINER PLATFORM에서의 SCTP(스트림 제어 전송 프로토콜)	54
8.1.1. SCTP 프로토콜을 사용하는 구성의 예	54
8.2. SCTP(스트림 제어 전송 프로토콜) 활성화	55
8.3. SCTP(STREAM CONTROL TRANSMISSION PROTOCOL)의 활성화 여부 확인	56
9장. PTP 하드웨어 구성	59
9.1. PTP 하드웨어 정보	59
9.2. PTP 네트워크 장치의 자동 검색	59
9.3. PTP OPERATOR 설치	60
9.3.1. CLI: PTP Operator 설치	60
9.3.2. 웹 콘솔: PTP Operator 설치	61
9.4. LINUXPTP 서비스 구성	62
10장. 네트워크 정책	65
10.1. 네트워크 정책 정의	65
10.1.1. 네트워크 정책 정의	65
10.1.2. 네트워크 정책 최적화	67
10.1.3. 다음 단계	68
10.1.4. 추가 리소스	68
10.2. 네트워크 정책 생성	68
10.2.1. 네트워크 정책 생성	68
10.2.2. NetworkPolicy 오브젝트 예	69
10.3. 네트워크 정책 보기	70
10.3.1. 네트워크 정책 보기	70
10.3.2. NetworkPolicy 오브젝트 예	71
10.4. 네트워크 정책 편집	72
10.4.1. 네트워크 정책 편집	72
10.4.2. NetworkPolicy 오브젝트 예	73
10.4.3. 추가 리소스	74
10.5. 네트워크 정책 삭제	74
10.5.1. 네트워크 정책 삭제	74
10.6. 프로젝트의 기본 네트워크 정책 정의	75
10.6.1. 새 프로젝트의 템플릿 수정	75
10.6.2. 새 프로젝트 템플릿에 네트워크 정책 추가	76
10.7. 네트워크 정책으로 다중 테넌트 격리 구성	77
10.7.1. 네트워크 정책을 사용하여 다중 테넌트 격리 구성	77
10.7.2. 다음 단계	79

10.7.3. 추가 리소스	79
11장. 다중 네트워크	80
11.1. 다중 네트워크 이해하기	80
11.1.1. 추가 네트워크 사용 시나리오	80
11.1.2. OpenShift Container Platform의 그룹은 중첩되지 않습니다.	80
11.2. 추가 네트워크 구성	81
11.2.1. 추가 네트워크 관리 접근법	81
11.2.2. 추가 네트워크 연결 구성	81
11.2.2.1. Cluster Network Operator를 통한 추가 네트워크 구성	81
11.2.2.2. YAML 매니페스트에서 추가 네트워크 구성	82
11.2.3. 추가 네트워크 유형에 대한 구성	82
11.2.3.1. 브리지 추가 네트워크 구성	82
11.2.3.1.1. 브릿지 구성 예	83
11.2.3.2. 호스트 장치 추가 네트워크 구성	84
11.2.3.2.1. 호스트 장치 구성 예	84
11.2.3.3. IPVLAN 추가 네트워크 구성	85
11.2.3.3.1. ipvlan 구성 예	85
11.2.3.4. MACVLAN 추가 네트워크 구성	86
11.2.3.4.1. macvlan 구성 예	86
11.2.4. 추가 네트워크에 대한 IP 주소 할당 구성	86
11.2.4.1. 고정 IP 주소 할당 구성	87
11.2.4.2. DHCP(동적 IP 주소) 할당 구성	88
11.2.4.3. Whereabouts를 사용한 동적 IP 주소 할당 구성	89
11.2.5. Cluster Network Operator로 추가 네트워크 연결 생성	90
11.2.6. YAML 매니페스트를 적용하여 추가 네트워크 연결 생성	91
11.3. 추가 네트워크에 POD 연결	92
11.3.1. 추가 네트워크에 Pod 추가	92
11.3.1.1. Pod별 주소 지정 및 라우팅 옵션 지정	94
11.4. 추가 네트워크에서 POD 제거	97
11.4.1. 추가 네트워크에서 Pod 제거	97
11.5. 추가 네트워크 편집	98
11.5.1. 추가 네트워크 연결 정의 수정	98
11.6. 추가 네트워크 제거	99
11.6.1. 추가 네트워크 연결 정의 제거	99
12장. 하드웨어 네트워크	100
12.1. SR-IOV(SINGLE ROOT I/O VIRTUALIZATION) 하드웨어 네트워크 정보	100
12.1.1. SR-IOV 네트워크 장치를 관리하는 구성 요소	100
12.1.1.1. 지원되는 플랫폼	101
12.1.1.2. 지원되는 장치	101
12.1.1.3. SR-IOV 네트워크 장치의 자동 검색	101
12.1.1.3.1. SrioNetworkNodeState 오브젝트의 예	102
12.1.1.4. Pod에서 가상 함수 사용 예	103
12.1.2. 다음 단계	104
12.2. SR-IOV NETWORK OPERATOR 설치	104
12.2.1. SR-IOV Network Operator 설치	105
12.2.1.1. CLI: SR-IOV Network Operator 설치	105
12.2.1.2. 웹 콘솔: SR-IOV Network Operator 설치	106
12.2.2. 다음 단계	107
12.3. SR-IOV NETWORK OPERATOR 구성	107
12.3.1. SR-IOV Network Operator 구성	107
12.3.1.1. Network Resources Injector 정보	108

12.3.1.2. SR-IOV Operator Admission Controller webhook 정보	108
12.3.1.3. 사용자 정의 노드 선택기 정보	109
12.3.1.4. Network Resources Injector 비활성화 또는 활성화	109
12.3.1.5. SR-IOV Operator Admission Controller webhook 비활성화 또는 활성화	109
12.3.1.6. SR-IOV Network Config 데몬에 대한 사용자 정의 NodeSelector 구성	109
12.3.2. 다음 단계	110
12.4. SR-IOV 네트워크 장치 구성	110
12.4.1. SR-IOV 네트워크 노드 구성 오브젝트	110
12.4.1.1. SR-IOV 네트워크 노드 구성 예	112
12.4.1.2. SR-IOV 장치의 VF(가상 기능) 파티셔닝	112
12.4.2. SR-IOV 네트워크 장치 구성	113
12.4.3. SR-IOV 구성 문제 해결	114
12.4.4. 다음 단계	115
12.5. SR-IOV 이더넷 네트워크 연결 구성	115
12.5.1. 이더넷 장치 구성 오브젝트	115
12.5.1.1. 추가 네트워크에 대한 IP 주소 할당 구성	116
12.5.1.1.1. 고정 IP 주소 할당 구성	117
12.5.1.1.2. DHCP(동적 IP 주소) 할당 구성	118
12.5.1.1.3. Whereabouts를 사용한 동적 IP 주소 할당 구성	119
12.5.2. SR-IOV 추가 네트워크 구성	120
12.5.3. 다음 단계	121
12.5.4. 추가 리소스	121
12.6. SR-IOV INFINIBAND 네트워크 연결 구성	121
12.6.1. InfiniBand 장치 구성 오브젝트	121
12.6.1.1. 추가 네트워크에 대한 IP 주소 할당 구성	122
12.6.1.1.1. 고정 IP 주소 할당 구성	122
12.6.1.1.2. DHCP(동적 IP 주소) 할당 구성	123
12.6.1.1.3. Whereabouts를 사용한 동적 IP 주소 할당 구성	124
12.6.2. SR-IOV 추가 네트워크 구성	125
12.6.3. 다음 단계	126
12.6.4. 추가 리소스	126
12.7. SR-IOV 추가 네트워크에 POD 추가	126
12.7.1. 네트워크 연결을 위한 런타임 구성	126
12.7.1.1. 이더넷 기반 SR-IOV 연결을 위한 런타임 구성	126
12.7.1.2. InfiniBand 기반 SR-IOV 연결을 위한 런타임 구성	127
12.7.2. 추가 네트워크에 Pod 추가	128
12.7.3. NUMA(Non-Uniform Memory Access) 정렬 SR-IOV Pod 생성	130
12.7.4. 추가 리소스	131
12.8. 고성능 멀티 캐스트 사용	132
12.8.1. 고성능 멀티 캐스트	132
12.8.2. 멀티 캐스트에 대한 SR-IOV 인터페이스 구성	132
12.9. DPDK 및 RDMA 모드와 함께 VF(가상 기능) 사용	133
12.9.1. DPDK 및 RDMA 모드에서 가상 기능을 사용하는 예	133
12.9.2. 사전 요구 사항	134
12.9.3. Intel NIC와 함께 DPDK 모드에서 VF(가상 기능)를 사용하는 예	134
12.9.4. Mellanox NIC와 함께 DPDK 모드에서 가상 기능을 사용하는 예	137
12.9.5. Mellanox NIC와 함께 RDMA 모드에서 가상 기능을 사용하는 예	140
12.10. SR-IOV NETWORK OPERATOR 설치 제거	142
12.10.1. SR-IOV Network Operator 설치 제거	142
13장. OPENSIFT SDN 기본 CNI 네트워크 공급자	144
13.1. OPENSIFT SDN 기본 CNI 네트워크 공급자 정보	144
13.1.1. OpenShift SDN 네트워크 격리 모드	144

13.1.2. 지원되는 기본 CNI 네트워크 공급자 기능 매트릭스	144
13.2. 프로젝트의 송신 IP 구성	144
13.2.1. 프로젝트 송신 트래픽에 대한 송신 IP 주소 할당	145
13.2.1.1. 자동 할당된 송신 IP 주소 사용 시 고려사항	146
13.2.1.2. 수동으로 할당된 송신 IP 주소 사용 시 고려사항	146
13.2.2. 네임스페이스에 자동으로 할당된 송신 IP 주소 구성	146
13.2.3. 네임스페이스에 수동으로 할당된 송신 IP 주소 구성	147
13.3. 프로젝트에 대한 송신 방화벽 구성	149
13.3.1. 프로젝트에서 송신 방화벽이 작동하는 방식	149
13.3.1.1. 송신 방화벽의 제한	150
13.3.1.2. 송신 방화벽 정책 규칙에 대한 일치 순서	151
13.3.1.3. DNS(Domain Name Server) 확인 작동 방식	151
13.3.2. EgressNetworkPolicy CR(사용자 정의 리소스) 오브젝트	151
13.3.2.1. EgressNetworkPolicy 규칙	152
13.3.2.2. EgressNetworkPolicy CR 오브젝트의 예	152
13.3.3. 송신 방화벽 정책 오브젝트 생성	153
13.4. 프로젝트의 송신 방화벽 편집	153
13.4.1. EgressNetworkPolicy 오브젝트 보기	153
13.5. 프로젝트의 송신 방화벽 편집	154
13.5.1. EgressNetworkPolicy 오브젝트 편집	154
13.6. 프로젝트에서 송신 방화벽 제거	155
13.6.1. EgressNetworkPolicy 오브젝트 제거	155
13.7. 송신 라우터 POD 사용에 대한 고려 사항	155
13.7.1. 송신 라우터 Pod 정보	155
13.7.1.1. 송신 라우터 모드	156
13.7.1.2. 송신 라우터 Pod 구현	156
13.7.1.3. 배포 고려 사항	156
13.7.1.4. 장애 조치 구성	157
13.7.2. 추가 리소스	158
13.8. 리디렉션 모드에서 송신 라우터 POD 배포	158
13.8.1. 리디렉션 모드에 대한 송신 라우터 Pod 사양	158
13.8.2. 송신 대상 구성 형식	159
13.8.3. 리디렉션 모드에서 송신 라우터 Pod 배포	160
13.8.4. 추가 리소스	160
13.9. HTTP 프록시 모드에서 송신 라우터 POD 배포	161
13.9.1. HTTP 모드에 대한 송신 라우터 Pod 사양	161
13.9.2. 송신 대상 구성 형식	162
13.9.3. HTTP 프록시 모드에서 송신 라우터 Pod 배포	162
13.9.4. 추가 리소스	163
13.10. DNS 프록시 모드에서 송신 라우터 POD 배포	163
13.10.1. DNS 모드에 대한 송신 라우터 Pod 사양	163
13.10.2. 송신 대상 구성 형식	164
13.10.3. DNS 프록시 모드에서 송신 라우터 Pod 배포	165
13.10.4. 추가 리소스	166
13.11. 구성 맵에서 송신 라우터 POD 대상 목록 구성	166
13.11.1. 구성 맵을 사용하여 송신 라우터 대상 매핑 구성	166
13.11.2. 추가 리소스	167
13.12. 프로젝트에 멀티 캐스트 사용	167
13.12.1. 멀티 캐스트 정보	168
13.12.2. Pod 간 멀티 캐스트 활성화	168
13.13. 프로젝트에 대한 멀티 캐스트 비활성화	170
13.13.1. Pod 간 멀티 캐스트 비활성화	170
13.14. OPENSIFT SDN을 사용하여 네트워크 격리 구성	170

13.14.1. 사전 요구 사항	170
13.14.2. 프로젝트 참여	171
13.14.3. 프로젝트 격리	171
13.14.4. 프로젝트의 네트워크 격리 비활성화	171
13.15. KUBE-PROXY 설정	172
13.15.1. iptables 규칙 동기화 정보	172
13.15.2. kube-proxy 구성 매개변수	172
13.15.3. kube-proxy 구성 수정	173
14장. OVN-KUBERNETES 기본 CNI 네트워크 공급자	175
14.1. OVN-KUBERNETES 기본 CNI(CONTAINER NETWORK INTERFACE) 네트워크 공급자 정보	175
14.1.1. OVN-Kubernetes 기능	175
14.1.2. 지원되는 기본 CNI 네트워크 공급자 기능 매트릭스	175
14.1.3. OVN-Kubernetes 제한 사항	175
14.2. OPENSIFT SDN 클러스터 네트워크 공급자에서 마이그레이션	176
14.2.1. OVN-Kubernetes 네트워크 공급자로 마이그레이션	176
14.2.1.1. OVN-Kubernetes 네트워크 공급자로 마이그레이션에 대한 고려 사항	176
네임스페이스 격리	177
송신 IP 주소	177
송신 네트워크 정책	177
송신 라우터 Pod	177
멀티 캐스트	177
네트워크 정책	178
14.2.1.2. 마이그레이션 프로세스의 작동 방식	178
14.2.2. OVN-Kubernetes 기본 CNI 네트워크 공급자로 마이그레이션	178
14.2.3. 추가 리소스	182
14.3. OPENSIFT SDN 네트워크 공급자로 롤백	183
14.3.1. 기본 CNI 네트워크 공급자를 OpenShift SDN으로 롤백	183
14.4. 프로젝트에 대한 송신 방화벽 구성	187
14.4.1. 프로젝트에서 송신 방화벽이 작동하는 방식	187
14.4.1.1. 송신 방화벽의 제한	188
14.4.1.2. 송신 방화벽 정책 규칙에 대한 일치 순서	189
14.4.2. EgressFirewall CR(사용자 정의 리소스) 오브젝트	189
14.4.2.1. EgressFirewall 규칙	189
14.4.2.2. EgressFirewall CR 오브젝트의 예	190
14.4.3. 송신 방화벽 정책 오브젝트 생성	191
14.5. 프로젝트의 송신 방화벽 보기	191
14.5.1. EgressFirewall 오브젝트 보기	191
14.6. 프로젝트의 송신 방화벽 편집	192
14.6.1. EgressFirewall 오브젝트 편집	192
14.7. 프로젝트에서 송신 방화벽 제거	193
14.7.1. EgressFirewall 오브젝트 제거	193
14.8. 송신 IP 주소 구성	193
14.8.1. 송신 IP 주소 아키텍처 설계 및 구현	193
14.8.1.1. 플랫폼 지원	194
14.8.1.2. Pod에 송신 IP 할당	194
14.8.1.3. 노드에 송신 IP 할당	195
14.8.1.4. 송신 IP 주소 구성에 대한 아키텍처 다이어그램	195
14.8.2. EgressIP 오브젝트	196
14.8.3. 송신 IP 주소 호스팅을 위해 노드에 레이블 지정	198
14.8.4. 다음 단계	198
14.8.5. 추가 리소스	198
14.9. 송신 IP 주소 할당	199

14.9.1. 네임스페이스에 송신 IP 주소 할당	199
14.9.2. 추가 리소스	200
14.10. 프로젝트에 멀티 캐스트 사용	200
14.10.1. 멀티 캐스트 정보	200
14.10.2. Pod 간 멀티 캐스트 활성화	200
14.11. 프로젝트에 대한 멀티 캐스트 비활성화	202
14.11.1. Pod 간 멀티 캐스트 비활성화	202
14.12. 하이브리드 네트워킹 구성	202
14.12.1. OVN-Kubernetes로 하이브리드 네트워킹 구성	202
14.12.2. 추가 리소스	204
15장. 경로 구성	205
15.1. 경로 구성	205
15.1.1. HTTP 기반 경로 생성	205
15.1.2. 경로 시간 초과 구성	206
15.1.3. HTTP 엄격한 전송 보안 활성화	206
15.1.4. 처리량 트리블슈팅	207
15.1.5. 쿠키를 사용하여 경로 상태 유지	208
15.1.5.1. 쿠키를 사용하여 경로에 주석 달기	208
15.1.6. 경로 기반 라우터	209
15.1.7. 경로별 주석	210
15.1.8. 경로 허용 정책 구성	216
15.1.9. Ingress 오브젝트를 통해 경로 생성	217
15.2. 보안 경로	219
15.2.1. 사용자 정의 인증서를 사용하여 재암호화 경로 생성	219
15.2.2. 사용자 정의 인증서를 사용하여 엣지 경로 생성	221
15.2.3. 패스스루 라우팅 생성	222
16장. 수신 클러스터 트래픽 구성	223
16.1. 수신 클러스터 트래픽 구성 개요	223
16.2. 서비스의 EXTERNALIP 구성	223
16.2.1. 사전 요구 사항	223
16.2.2. ExternalIP 정보	223
16.2.2.1. ExternalIP 구성	224
16.2.2.2. 외부 IP 주소 할당 제한 사항	225
16.2.2.3. 정책 오브젝트의 예	226
16.2.3. ExternalIP 주소 블록 구성	227
외부 IP 구성의 예	228
16.2.4. 클러스터에 대한 외부 IP 주소 블록 구성	228
16.2.5. 다음 단계	229
16.3. INGRESS 컨트롤러를 사용한 수신 클러스터 트래픽 구성	229
16.3.1. Ingress 컨트롤러 및 경로 사용	229
16.3.2. 사전 요구 사항	230
16.3.3. 프로젝트 및 서비스 생성	230
16.3.4. 경로를 생성하여 서비스 노출	231
16.3.5. 경로 라벨을 사용하여 Ingress 컨트롤러 분할 구성	232
16.3.6. 네임스페이스 라벨을 사용하여 Ingress 컨트롤러 분할 구성	232
16.3.7. 추가 리소스	233
16.4. 로드 밸런서를 사용하여 수신 클러스터 트래픽 구성	234
16.4.1. 로드 밸런서를 사용하여 클러스터로 트래픽 가져오기	234
16.4.2. 사전 요구 사항	234
16.4.3. 프로젝트 및 서비스 생성	234
16.4.4. 경로를 생성하여 서비스 노출	235

16.4.5. 로드 밸런서 서비스 생성	236
16.5. 네트워크 로드 밸런서를 사용하여 AWS에서 수신 클러스터 트래픽 구성	238
16.5.1. Ingress 컨트롤러 Classic 로드 밸런서를 네트워크 로드 밸런서로 교체	238
16.5.2. 기존 AWS 클러스터에서 Ingress 컨트롤러 네트워크 로드 밸런서 생성	239
16.5.3. 새 AWS 클러스터에서 Ingress 컨트롤러 네트워크 로드 밸런서 생성	240
16.5.4. 추가 리소스	241
16.6. 서비스 외부 IP에 대한 수신 클러스터 트래픽 구성	241
16.6.1. 사전 요구 사항	241
16.6.2. 서비스에 ExternalIP 연결	241
16.6.3. 추가 리소스	242
16.7. NODEPORT를 사용하여 수신 클러스터 트래픽 구성	242
16.7.1. NodePort를 사용하여 클러스터로 트래픽 가져오기	243
16.7.2. 사전 요구 사항	243
16.7.3. 프로젝트 및 서비스 생성	243
16.7.4. 경로를 생성하여 서비스 노출	244
16.7.5. 추가 리소스	245
17장. 클러스터 전체 프록시 구성	246
17.1. 사전 요구 사항	246
17.2. 클러스터 전체 프록시 사용	246
17.3. 클러스터 전체 프록시 제거	248
추가 리소스	249
18장. 사용자 정의 PKI 구성	250
18.1. 설치 중 클러스터 전체 프록시 구성	250
18.2. 클러스터 전체 프록시 사용	251
18.3. OPERATOR를 사용한 인증서 주입	253
19장. RHOSP의 로드 밸런싱	256
19.1. KURYR SDN으로 OCTAVIA OVN 로드 밸런서 공급자 드라이버 사용	256
19.2. OCTAVIA를 사용하여 애플리케이션 트래픽의 클러스터 확장	257
19.2.1. Octavia를 사용하여 클러스터 스케일링	257
19.2.2. Octavia를 사용하여 Kuryr를 사용하는 클러스터 스케일링	259
19.3. RHOSP OCTAVIA를 사용하여 수신 트래픽 스케일링	259
20장. 보조 인터페이스 지표와 네트워크 연결 연관 짓기	262
20.1. 보조 인터페이스 지표와 네트워크 연결 연관 짓기	262
20.1.1. 네트워크 지표 데몬	262
20.1.2. 네트워크 이름이 있는 지표	263

1장. 네트워킹 이해

클러스터 관리자에게는 클러스터 내부에서 실행되는 애플리케이션을 외부 트래픽에 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 특정 유형의 연결만 허용할 수 있습니다. HTTP/2, HTTPS 및 SNI(서버 이름 식별) 및 인증서가 있는 TLS. OpenShift Container Platform에서는 Ingress 리소스에서 지정하는 조건을 충족하기 위해 경로가 생성됩니다.

2장. 호스트에 액세스

SSH(Secure Shell) 액세스를 통해 OpenShift Container Platform 인스턴스에 액세스하고 컨트롤 플레인 노드(마스터 노드라고도 함)에 액세스하기 위한 베스천 호스트를 생성하는 방법을 알아봅니다.

2.1. 설치 관리자 프로비저닝 인프라 클러스터에서 AMAZON WEB SERVICES 의 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 관련된 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 클레인 호스트(마스터 호스트라고도 함)에 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 선택된 CNI(Container Network Interface) 기본 네트워크 공급자 플러그인의 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.

4장. OPENSIFT 컨테이너 플랫폼의 CLUSTER NETWORK OPERATOR

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

4.1. CNO(CLUSTER NETWORK OPERATOR)

Cluster Network Operator는 **operator.openshift.io** API 그룹에서 **네트워크 API**를 구현합니다. Operator는 데몬 세트를 사용하여 OpenShift SDN 기본 컨테이너 네트워크 인터페이스(CNI) 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

다음 필드는 Operator 상태에 대한 정보를 제공합니다. **AVAILABLE**, **PROGRESSING** 및 **DEGRADED**. 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

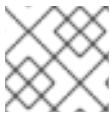
Pod IP 주소가 할당되는 IP 주소 풀입니다.

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.

cluster라는 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
metadata.name	string	CNO 개체 이름입니다. 이 이름은 항상 cluster 입니다.
spec.clusterNetwork	array	Pod IP 주소가 할당되는 IP 주소 블록과 클러스터의 각 개별 노드에 할당된 서브넷 접두사 길이를 지정하는 목록입니다. 예를 들면 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 cluster 라는 Network.config.openshift.io 개체에서 상속됩니다.</p>

필드	유형	설명
spec.serviceNetwork	array	<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 들면 다음과 같습니다.</p> <pre>spec: serviceNetwork: - 172.30.0.0/14</pre> <p>이 값은 준비 전용이며 클러스터 설치 중에 cluster 라는 Network.config.openshift.io 개체에서 상속됩니다.</p>
spec.defaultNetwork	object	클러스터 네트워크의 CNI(Container Network Interface) 클러스터 네트워크 공급자를 구성합니다.
spec.kubeProxyConfig	object	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
type	string	<p>OpenShiftSDN 또는 OVNKubernetes 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>참고</p> <p>OpenShift Container Platform은 기본적으로 OpenShift SDN CNI(Container Network Interface) 클러스터 네트워크 공급자를 사용합니다.</p> </div> </div>
openshiftSDNConfig	object	이 오브젝트는 OpenShift SDN 클러스터 네트워크 공급자에만 유효합니다.
ovnKubernetesConfig	object	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
mode	string	OpenShift SDN의 네트워크 격리 모드입니다.
mtu	integer	VXLAN 오버레이 네트워크의 최대 전송 단위(MTU)입니다. 이 값은 일반적으로 자동 구성됩니다.
vxlanPort	integer	모든 VXLAN 패킷에 사용할 포트입니다. 기본값은 4789 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
mtu	integer	Geneve(Generic Network Virtualization Encapsulation) 오버레이 네트워크의 MTU(최대 전송 단위)입니다. 이 값은 일반적으로 자동 구성됩니다.
genevePort	integer	Geneve 오버레이 네트워크용 UDP 포트입니다.



참고

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

OVN-Kubernetes 구성 예


```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
```

```
mtu: 1400
genevePort: 6081
```

kubeProxyConfig 오브젝트 구성

kubeProxyConfig 오브젝트의 값은 다음 표에 정의되어 있습니다.

표 4.5. kubeProxyConfig object

필드	유형	설명
iptablesSyncPeriod	string	<p>iptables 규칙의 새로 고침 간격입니다. 기본값은 30s입니다. 유효 접미사로 s, m, h가 있으며, 자세한 설명은 Go time 패키지 문서를 참조하십시오.</p> <div style="display: flex; align-items: flex-start;">  <div> <p>참고</p> <p>OpenShift Container Platform 4.3 이상에서는 성능이 개선되어 더 이상 iptablesSyncPeriod 매개변수를 조정할 필요가 없습니다.</p> </div> </div>
proxyArguments.iptables-min-sync-period	array	<p>iptables 규칙을 새로 고치기 전 최소 기간입니다. 이 필드를 통해 새로 고침 간격이 너무 짧지 않도록 조정할 수 있습니다. 유효 접미사로 s, m, h가 있으며, 자세한 설명은 Go time 패키지를 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 **Network API**

5장. OPENSIFT CONTAINER PLATFORM에서의 DNS OPERATOR

DNS Operator는 CoreDNS를 배포 및 관리하고 Pod에 이름 확인 서비스를 제공하여 OpenShift에서 DNS 기반 Kubernetes 서비스 검색을 사용할 수 있도록 합니다.

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**에서는 Operator 상태에 대한 정보를 제공합니다. **AVAILABLE**은 CoreDNS 데몬 세트에서 1개 이상의 포트가 **Available** 상태 조건을 보고할 때 **True**입니다.

5.2. 기본 DNS 보기

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]
```

5.3. DNS 전달 사용

지정된 구역에 사용해야 하는 네임 서버를 지정하는 방식으로 DNS 전달을 사용하여 **etc/resolv.conf**에서 식별된 영역별 전달 구성을 덮어쓸 수 있습니다. 전달된 영역이 OpenShift Container Platform에서 관리하는 Ingress 도메인인 경우 도메인에 대한 업스트림 이름 서버를 승인해야 합니다.

프로세스

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

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

이를 통해 Operator는 **Server** 기반의 추가 서버 구성 블록으로 **dns-default**라는 ConfigMap을 생성 및 업데이트할 수 있습니다. 서버에 쿼리와 일치하는 영역이 없는 경우 이름 확인은 **/etc/resolv.conf**에 지정된 네임 서버로 대체됩니다.

샘플 DNS

```

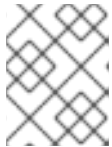
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
    - name: foo-server 1
  zones: 2
    - example.com
    
```

```

forwardPlugin:
  upstreams: ③
    - 1.1.1.1
    - 2.2.2.2:5353
- name: bar-server
  zones:
    - bar.com
    - example.com
forwardPlugin:
  upstreams:
    - 3.3.3.3
    - 4.4.4.4:5454

```

- ① **name**은 **rfc6335** 서비스 이름 구문을 준수해야 합니다.
- ② **zones**는 **rfc1123**의 하위 도메인 정의를 준수해야 합니다. 클러스터 도메인에 해당하는 **cluster.local**은 영역에 유효하지 않은 하위 도메인입니다.
- ③ **forwardPlugin**당 최대 15개의 업스트림이 허용됩니다.



참고

servers가 정의되지 않았거나 유효하지 않은 경우 ConfigMap에는 기본 서버만 포함됩니다.

2. ConfigMap을 확인합니다.

```
$ 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 ①
    }
    .:5353 {
      errors
      health
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        upstream
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      forward . /etc/resolv.conf {
        policy sequential
      }
      cache 30

```

```

      reload
    }
  kind: ConfigMap
  metadata:
    labels:
      dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns

```

- 1 **forwardPlugin**을 변경하면 CoreDNS 데몬 세트의 롤링 업데이트가 트리거됩니다.

추가 리소스

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

5.4. 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
```

5.5. 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
```

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** / 디렉터리의 **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
<p>domain</p>	<p>domain은 Ingress 컨트롤러에서 제공하는 DNS 이름이며 여러 기능을 구성하는데 사용됩니다.</p> <ul style="list-style-type: none"> ● LoadBalancerService 끝점 게시 방식에서는 domain을 사용하여 DNS 레코드를 구성합니다. endpointPublishingStrategy를 참조하십시오. ● 생성된 기본 인증서를 사용하는 경우, 인증서는 domain 및 해당 subdomains에 유효합니다. defaultCertificate를 참조하십시오. ● 사용자가 외부 DNS 레코드의 대상 위치를 확인할 수 있도록 이 값이 개별 경로 상태에 게시됩니다. <p>domain 값은 모든 Ingress 컨트롤러에서 고유해야 하며 업데이트할 수 없습니다.</p> <p>비어 있는 경우 기본값은 ingress.config.openshift.io/cluster.spec.domain입니다.</p>
<p>replicas</p>	<p>replicas는 원하는 개수의 Ingress 컨트롤러 복제본입니다. 설정되지 않은 경우, 기본값은 2입니다.</p>
<p>endpointPublishingStrategy</p>	<p>endpointPublishingStrategy는 Ingress 컨트롤러 끝점을 다른 네트워크에 게시하고 로드 밸런서 통합을 활성화하며 다른 시스템에 대한 액세스를 제공하는 데 사용됩니다.</p> <p>설정되지 않은 경우, 기본값은 infrastructure.config.openshift.io/cluster.status.platform을 기반으로 다음과 같습니다.</p> <ul style="list-style-type: none"> ● AWS: LoadBalancerService (외부 범위 포함) ● Azure: LoadBalancerService (외부 범위 포함) ● GCP: LoadBalancerService (외부 범위 포함) ● 베어 메탈: NodePortService ● 기타: HostNetwork <p>endpointPublishingStrategy 값은 업데이트할 수 없습니다.</p>
<p>defaultCertificate</p>	<p>defaultCertificate 값은 Ingress 컨트롤러가 제공하는 기본 인증서가 포함된 보안에 대한 참조입니다. 경로가 고유한 인증서를 지정하지 않으면 defaultCertificate가 사용됩니다.</p> <p>보안에는 키와 데이터, 즉 *tls.crt: 인증서 파일 내용 *tls.key: 키 파일 내용이 포함되어야 합니다.</p> <p>설정하지 않으면 와일드카드 인증서가 자동으로 생성되어 사용됩니다. 인증서는 Ingress 컨트롤러 도메인 및 하위 도메인에 유효하며 생성된 인증서의 CA는 클러스터의 신뢰 저장소와 자동으로 통합됩니다.</p> <p>생성된 인증서 또는 사용자 정의 인증서는 OpenShift Container Platform 내장 OAuth 서버와 자동으로 통합됩니다.</p>

매개변수	설명
namespaceSelector	namespaceSelector 는 Ingress 컨트롤러가 서비스를 제공하는 네임스페이스 집합을 필터링하는 데 사용됩니다. 이는 분할을 구현하는 데 유용합니다.
routeSelector	routeSelector 는 Ingress 컨트롤러가 서비스를 제공하는 경로 집합을 필터링하는 데 사용됩니다. 이는 분할을 구현하는 데 유용합니다.
nodePlacement	<p>nodePlacement를 사용하면 Ingress 컨트롤러의 스케줄링을 명시적으로 제어할 수 있습니다.</p> <p>설정하지 않으면 기본값이 사용됩니다.</p> <div data-bbox="517 645 625 1115" style="border: 1px solid black; padding: 5px; width: fit-content;">  </div> <p>참고</p> <p>nodePlacement 매개변수는 nodeSelector 및 tolerations의 두 부분으로 구성됩니다. 예를 들면 다음과 같습니다.</p> <pre data-bbox="703 835 1069 1115"> nodePlacement: nodeSelector: matchLabels: kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists </pre>

매개변수	설명
<p>tlsSecurityProfile</p>	<p>tlsSecurityProfile은 Ingress 컨트롤러의 TLS 연결 설정을 지정합니다.</p> <p>설정되지 않으면, 기본값은 apiservers.config.openshift.io/cluster 리소스를 기반으로 설정됩니다.</p> <p>Old, Intermediate 및 Modern 프로파일 유형을 사용하는 경우 유효한 프로파일 구성은 릴리스마다 변경될 수 있습니다. 예를 들어, 릴리스 X.Y.Z에 배포된 Intermediate 프로파일을 사용하도록 설정한 경우 X.Y.Z+1 릴리스로 업그레이드하면 새 프로파일 구성이 Ingress 컨트롤러에 적용되어 롤아웃이 발생할 수 있습니다.</p> <p>Ingress 컨트롤러의 최소 TLS 버전은 1.1이며 최대 TLS 버전은 1.2입니다.</p> <div data-bbox="518 678 624 1301" style="background-color: black; width: 66px; height: 278px; margin: 10px 0;"></div> <p>중요</p> <p>HAProxy Ingress 컨트롤러 이미지는 TLS 1.3을 지원하지 않으며 Modern 프로파일에는 TLS 1.3이 필요하므로 이는 지원되지 않습니다. Ingress Operator는 Modern 프로파일을 Intermediate로 변환합니다.</p> <p>또한, Ingress Operator는 Old 또는 Custom 프로파일의 TLS 1.0을 1.1로 변환하고 Custom 프로파일의 TLS 1.3을 1.2로 변환합니다.</p> <p>OpenShift Container Platform 라우터를 사용하면 TLS_AES_128_CCM_SHA256, TLS_CHACHA20_POLY1305_SHA256, TLS_AES_256_GCM_SHA384, TLS_AES_128_GCM_SHA256, TLS_AES_128_GCM_SHA256을 사용하는 Red Hat 배포 OpenSSL 기본 세트가 사용됩니다. OpenShift Container Platform 4.6, 4.7, 4.8에서는 TLS 1.3 연결 및 암호 제품군을 사용할 수 없습니다.</p> <div data-bbox="518 1469 624 1686" style="background-color: black; width: 66px; height: 97px; margin: 10px 0;"></div> <p>참고</p> <p>구성된 보안 프로파일의 암호 및 최소 TLS 버전은 TLSPProfile 상태에 반영됩니다.</p>

매개변수	설명
routeAdmission	<p>routeAdmission은 네임스페이스에서 클레임을 허용 또는 거부하는 등 새로운 경로 클레임을 처리하기 위한 정책을 정의합니다.</p> <p>namespaceOwnership은 네임스페이스에서 호스트 이름 클레임을 처리하는 방법을 설명합니다. 기본값은 Strict입니다.</p> <ul style="list-style-type: none"> ● Strict: 경로가 네임스페이스에서 동일한 호스트 이름을 요청하는 것을 허용하지 않습니다. ● InterNamespaceAllowed: 경로가 네임스페이스에서 동일한 호스트 이름의 다른 경로를 요청하도록 허용합니다. <p>wildcardPolicy는 Ingress 컨트롤러에서 와일드카드 정책이 포함된 경로를 처리하는 방법을 설명합니다.</p> <ul style="list-style-type: none"> ● WildcardsAllowed: 와일드카드 정책이 있는 경로가 Ingress 컨트롤러에 의해 허용됨을 나타냅니다. ● WildcardsDisallowed: 와일드카드 정책이 None인 경로만 Ingress 컨트롤러에서 허용됨을 나타냅니다. WildcardsAllowed에서 WildcardsDisallowed로 wildcardPolicy를 업데이트하면 와일드카드 정책이 Subdomain인 허용되는 경로의 작동이 중지됩니다. Ingress 컨트롤러에서 이러한 경로를 다시 허용하려면 이 경로를 설정이 None인 와일드카드 정책으로 다시 생성해야 합니다. 기본 설정은 WildcardsDisallowed입니다.

매개변수	설명
<p>IngressControllerLogging</p>	<p>logging은 어디에서 무엇이 기록되는지에 대한 매개변수를 정의합니다. 이 필드가 비어 있으면 작동 로그는 활성화되지만 액세스 로그는 비활성화됩니다.</p> <ul style="list-style-type: none"> ● access는 클라이언트 요청이 기록되는 방법을 설명합니다. 이 필드가 비어 있으면 액세스 로깅이 비활성화됩니다. <ul style="list-style-type: none"> ○ destination은 로그 메시지의 대상을 설명합니다. <ul style="list-style-type: none"> ■ type은 로그 대상의 유형입니다. <ul style="list-style-type: none"> ● Container는 로그가 사이드카 컨테이너로 이동하도록 지정합니다. Ingress Operator는 Ingress 컨트롤러 pod에서 logs 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. ● Syslog는 로그가 Syslog 끝점으로 전송되도록 지정합니다. 관리자는 Syslog 메시지를 수신할 수 있는 끝점을 지정해야 합니다. 관리자가 사용자 정의 Syslog 인스턴스를 구성하는 것이 좋습니다. ■ 컨테이너는 Container 로깅 대상 유형의 매개변수를 설명합니다. 현재는 컨테이너 로깅에 대한 매개변수가 없으므로 이 필드는 비어 있어야 합니다. ■ syslog는 Syslog 로깅 대상 유형의 매개변수를 설명합니다. <ul style="list-style-type: none"> ● address는 로그 메시지를 수신하는 syslog 끝점의 IP 주소입니다. ● port는 로그 메시지를 수신하는 syslog 끝점의 UDP 포트 번호입니다. ● facility는 로그 메시지의 syslog 기능을 지정합니다. 이 필드가 비어 있으면 장치가 local1이 됩니다. 그렇지 않으면 유효한 syslog 기능을 지정해야 합니다. kern,사용자,메일,데몬,auth,syslog,lpr,뉴스,uucp,cron,auth2,ftp,ntp,audit,alert,cron2,local0,local1,local2,local3,local4,local5,local6 또는 local7. ○ httpLogFormat은 HTTP 요청에 대한 로그 메시지의 형식을 지정합니다. 이 필드가 비어 있으면 로그 메시지는 구현의 기본 HTTP 로그 형식을 사용합니다. HAProxy의 기본 HTTP 로그 형식과 관련한 내용은 HAProxy 문서를 참조하십시오.

매개변수	설명
httpHeaders	<p>httpHeaders는 HTTP 헤더에 대한 정책을 정의합니다.</p> <p>IngressControllerHTTPHeaders에 forwardedHeaderPolicy를 설정하여 Ingress 컨트롤러에서 Forwarded, X-Forwarded-For, X-Forwarded-Host, X-Forwarded-Port, X-Forwarded-Proto, X-Forwarded-Proto-Version HTTP 헤더를 설정하는 시기와 방법을 지정합니다.</p> <p>기본적으로 정책은 Append로 설정됩니다.</p> <ul style="list-style-type: none"> ● Append는 Ingress 컨트롤러에서 기존 헤더를 유지하면서 헤더를 추가하도록 지정합니다. ● Replace는 Ingress 컨트롤러에서 헤더를 설정하고 기존 헤더를 제거하도록 지정합니다. ● IfNone은 헤더가 아직 설정되지 않은 경우 Ingress 컨트롤러에서 헤더를 설정하도록 지정합니다. ● Never는 Ingress 컨트롤러에서 헤더를 설정하지 않고 기존 헤더를 보존하도록 지정합니다.



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 프로파일을 사용하여 다양한 OpenShift Container Platform 구성 요소에 필요한 TLS 암호를 정의할 수 있습니다. OpenShift Container Platform TLS 보안 프로파일은 [Mozilla 권장 구성](#)을 기반으로 합니다.

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

표 6.1. TLS 보안 프로파일

Profile	설명
Old	<p>이 프로파일은 레거시 클라이언트 또는 라이브러리와 함께 사용하기 위한 것입니다. 프로파일은 이전 버전과의 호환성 권장 구성을 기반으로 합니다.</p> <p>Old 프로파일에는 최소 TLS 버전 1.0이 필요합니다.</p> <p> 참고</p> <p>Ingress 컨트롤러의 경우 최소 TLS 버전이 1.0에서 1.1로 변환됩니다.</p>

Profile	설명
<p>Intermediate</p>	<p>이 프로파일은 대부분의 클라이언트에서 권장되는 구성입니다. Ingress 컨트롤러 및 컨트롤 플레인의 기본 TLS 보안 프로파일입니다. 프로파일은 중간 호환성 권장 구성을 기반으로 합니다.</p> <p>Intermediate 프로파일에는 최소 TLS 버전이 1.2가 필요합니다.</p>
<p>Modern</p>	<p>이 프로파일은 이전 버전과의 호환성이 필요하지 않은 최신 클라이언트와 사용하기 위한 것입니다. 이 프로파일은 최신 호환성 권장 구성을 기반으로 합니다.</p> <p>Modern 프로파일에는 최소 TLS 버전 1.3이 필요합니다.</p> <div data-bbox="593 660 702 824">  <p>참고</p> <p>OpenShift Container Platform 4.6, 4.7, 4.8에서는 Modern 프로파일 지원되지 않습니다. 선택하면 Intermediate 프로파일 활성화됩니다.</p> </div> <div data-bbox="593 873 702 974">  <p>중요</p> <p>현재 Modern 프로파일은 지원되지 않습니다.</p> </div>
<p>사용자 지정</p>	<p>이 프로파일을 사용하면 사용할 TLS 버전과 암호를 정의할 수 있습니다.</p> <div data-bbox="593 1137 1428 1429" style="background-color: #fff9c4; padding: 10px;"> <div data-bbox="646 1214 746 1303">  <p>주의</p> <p>Custom 프로파일을 사용할 때는 잘못된 구성으로 인해 문제가 발생할 수 있으므로 주의해야 합니다.</p> </div> </div> <div data-bbox="593 1473 702 1729">  <p>참고</p> <p>OpenShift Container Platform 라우터를 사용하면 Red Hat에서 배포한 OpenSSL 기본 TLS 1.3 암호화 제품군 세트를 사용할 수 있습니다. OpenShift Container Platform 4.6, 4.7, 4.8에서는 TLS 1.3 연결 및 암호 제품군을 사용할 수 없습니다.</p> </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 프로필 구성이 적용되어 롤아웃이 발생할 수 있습니다.

6.3.1.2. Ingress 컨트롤러의 TLS 보안 프로필 구성

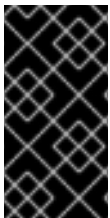
Ingress 컨트롤러에 대한 TLS 보안 프로필을 구성하려면 **IngressController** CR(사용자 정의 리소스)을 편집하여 사전 정의된 또는 사용자 지정 TLS 보안 프로필을 지정합니다. TLS 보안 프로필이 구성되지 않은 경우 기본값은 API 서버에 설정된 TLS 보안 프로필을 기반으로 합니다.

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.



중요

HAProxy Ingress 컨트롤러 이미지는 TLS **1.3**을 지원하지 않으며 **Modern** 프로파일에는 TLS **1.3**이 필요하므로 이는 지원되지 않습니다. Ingress Operator는 **Modern** 프로파일을 **Intermediate**로 변환합니다. 또한, Ingress Operator는 **Old** 또는 **Custom** 프로파일의 TLS **1.0**을 **1.1**로 변환하고 **Custom** 프로파일의 TLS **1.3**을 **1.2**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

...

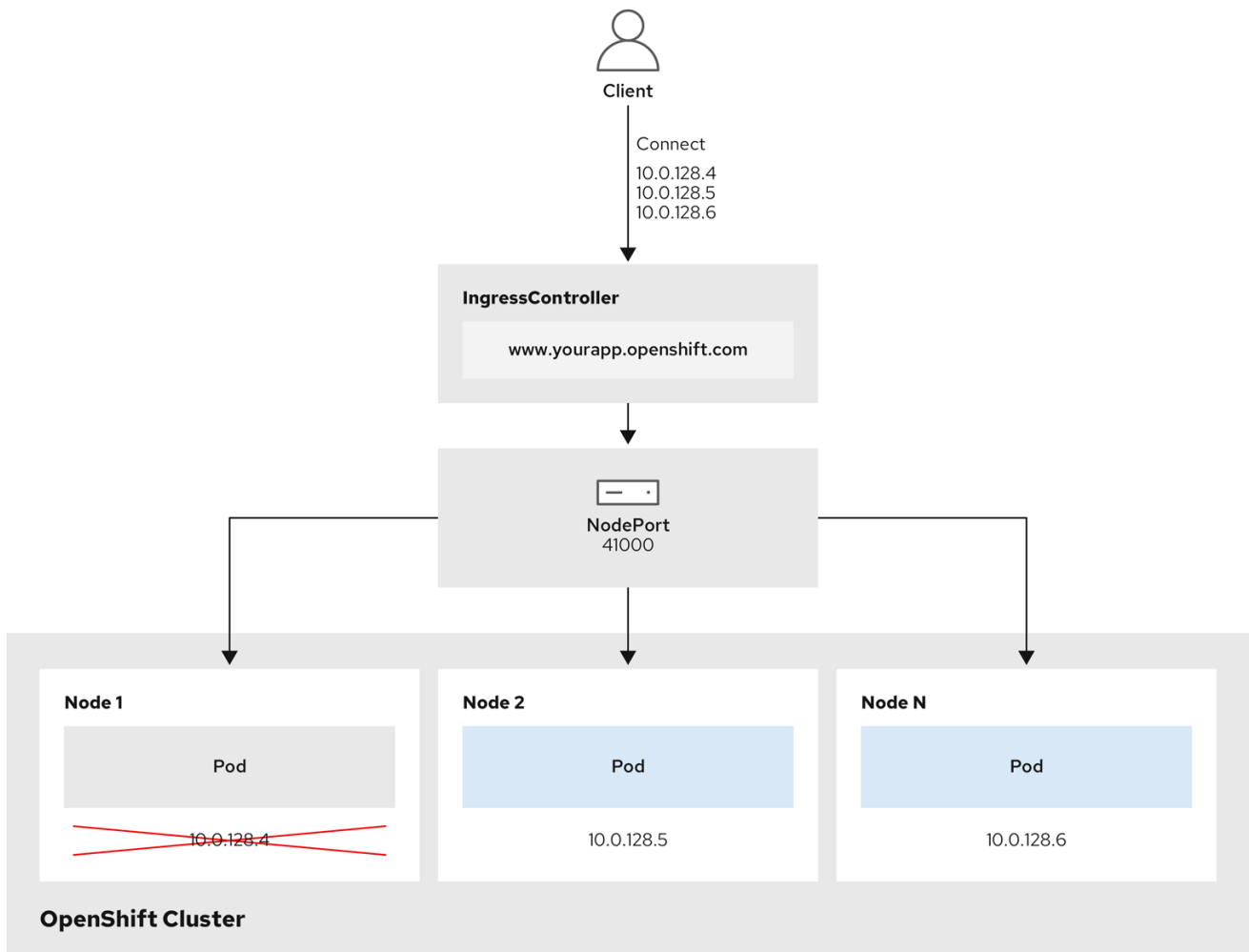
6.3.2. Ingress 컨트롤러 끝점 게시 전략

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 Platform에 의해 동적으로 할당됩니다. 그러나 정적 포트 할당을 지원하기 위해 관리형 **NodePortService**의 노드 포트 필드에 대한 변경 사항은 유지됩니다.

그림 6.1. NodePortService 다이어그램



202_OpenShift_0222

앞의 그래픽에서는 OpenShift Container Platform Ingress NodePort 끝점 게시 전략과 관련된 다음 개념을 보여줍니다.

- 클러스터에서 사용 가능한 모든 노드에는 외부적으로 액세스할 수 있는 자체 노드가 있습니다. 클러스터에서 실행 중인 서비스는 모든 노드에 대해 고유한 NodePort에 바인딩됩니다.
- 클라이언트가 그래픽에서 **10.0.128.4** IP 주소를 연결하여 다운된 노드에 연결할 때 노드 포트는

클라이언트를 서비스를 실행하는 사용 가능한 노드에 직접 연결합니다. 이 시나리오에서는 로드 밸런싱이 필요하지 않습니다. 이미지에 **10.0.128.4** 주소가 다운되고 다른 IP 주소를 대신 사용해야 합니다.



참고

Ingress Operator는 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 pod가 해당 포트를 사용하는 경우 복제본을 노드에 예약할 수 없습니다.

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
```

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
```

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.
- 이전에는 Ingress 컨트롤러에 대한 사용자 정의 기본 인증서를 구성하셨습니다.

절차

- 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

처리량을 늘리기 위한 요구 사항과 같은 라우팅 성능 또는 가용성 요구 사항을 충족하도록 Ingress 컨트롤러를 수동으로 확장합니다. **IngressController** 리소스를 확장하는 데 **oc** 명령이 사용됩니다. 다음 절차는 기본 **IngressController**를 확장하는 예제입니다.

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
```



참고

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

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 클러스터 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
  endpointPublishingStrategy:
    type: NodePortService 1
```

```
logging:
  access:
    destination:
      type: Container
```

- 1 사이드카에 Ingress 액세스 로깅을 구성하는 데 **NodePortService**가 필요하지 않습니다. 수신 로깅은 모든 **endpointPublishingStrategy**와 호환됩니다.

- 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
  endpointPublishingStrategy:
    type: NodePortService
  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
  endpointPublishingStrategy:
    type: NodePortService
  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
  endpointPublishingStrategy:
    type: NodePortService
  logging:
    access: null

```

6.8.5. Ingress 컨트롤러 분할

Ingress 컨트롤러 또는 라우터는 트래픽이 클러스터로 유입되는 기본 메커니즘이므로 수요가 매우 클 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

Ingress 컨트롤러는 라우팅 라벨 또는 네임스페이스 라벨을 분할 방법으로 사용할 수 있습니다.

6.8.5.1. 경로 라벨을 사용하여 Ingress 컨트롤러 분할 구성

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

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.

6.8.5.2. 네임스페이스 라벨을 사용하여 Ingress 컨트롤러 분할 구성

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

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.



주의

Keepalived Ingress VIP를 배포하는 경우 **endpointPublishingStrategy** 매개변수에 값이 **HostNetwork** 인 기본이 아닌 Ingress 컨트롤러를 배포하지 마십시오. 이렇게 하면 문제가 발생할 수 있습니다. **endpointPublishingStrategy** 에 대해 **HostNetwork** 대신 **NodePort** 값을 사용합니다.

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.

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

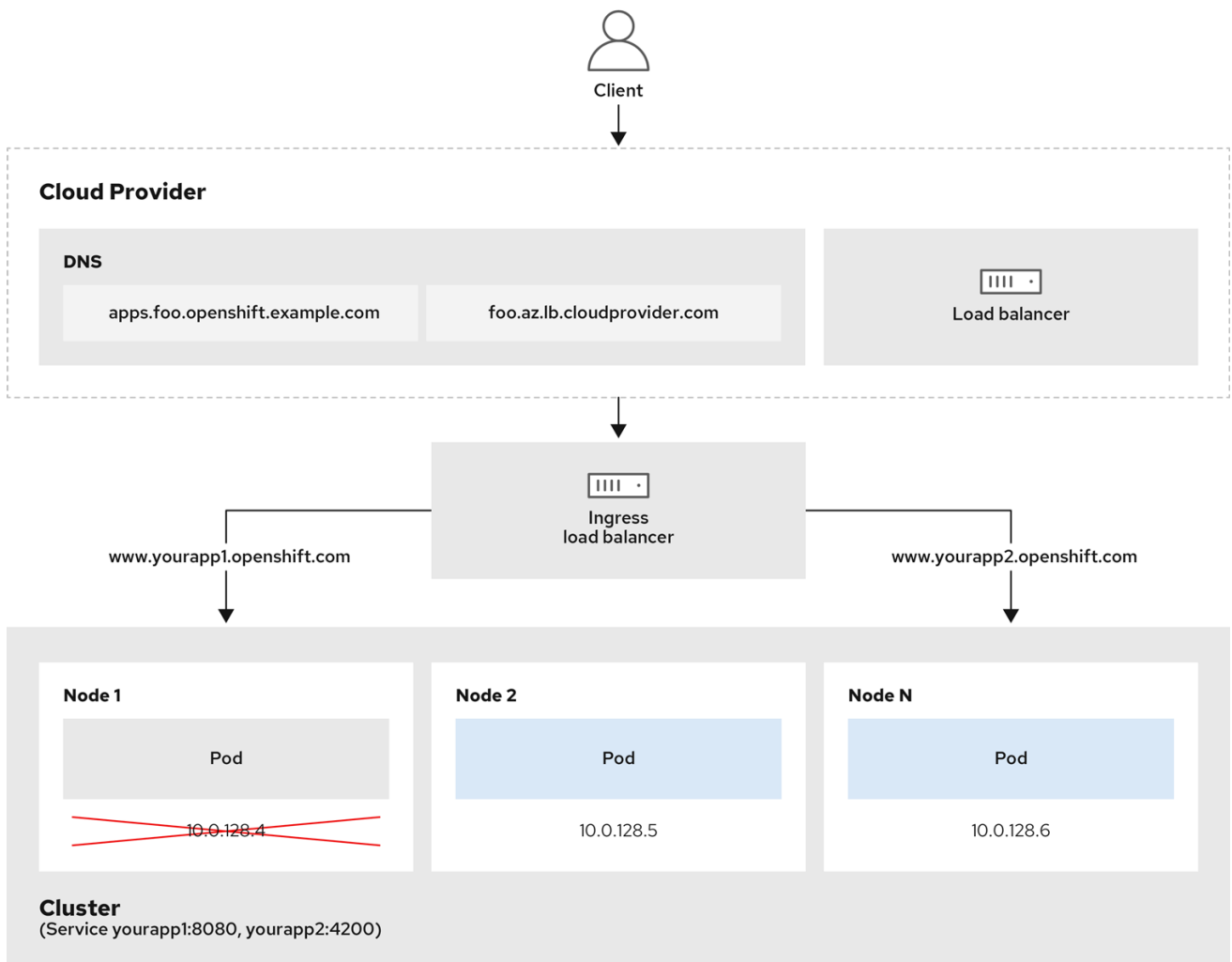
클라우드 공급자가 Microsoft Azure인 경우 노드를 가리키는 퍼블릭 로드 밸런서가 하나 이상 있어야 합니다. 그렇지 않으면 모든 노드의 인터넷 연결이 끊어집니다.



중요

IngressController 오브젝트의 **scope**를 변경하려면, 해당 **IngressController** 오브젝트를 삭제한 후 다시 생성해야 합니다. CR(사용자 정의 리소스)을 생성한 후에는 **.spec.endpointPublishingStrategy.loadBalancer.scope** 매개변수를 변경할 수 없습니다.

그림 6.2. LoadBalancer 다이어그램



202_OpenShift_0222

앞의 그래픽에서는 OpenShift Container Platform Ingress LoadBalancerService 끝점 게시 전략과 관련된 다음 개념을 보여줍니다.

- OpenShift Ingress 컨트롤러 로드 밸런서를 사용하여 클라우드 공급자 로드 밸런서를 사용하거나 내부적으로 로드 밸런싱을 로드할 수 있습니다.

- 그래픽에 표시된 클러스터에 표시된 것처럼 로드 밸런서의 단일 IP 주소와 더 친숙한 포트(예: 8080 및 4200)를 사용할 수 있습니다.
- 외부 로드 밸런서의 트래픽은 다운 노드 인스턴스에 표시된 대로 Pod에 전달되고 로드 밸런서에 의해 관리됩니다. 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.yaml** 파일에 **IngressController** CR(사용자 정의 리소스)을 생성합니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal 3
```

- 1 **<name>**을 **IngressController** 오브젝트의 이름으로 변경합니다.
- 2 컨트롤러가 게시한 애플리케이션의 **domain**을 지정합니다.
- 3 내부 로드 밸런서를 사용하려면 **Internal** 값을 지정합니다.

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

```
$ oc create -f <name>-ingress-controller.yaml 1
```

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

3. 선택 사항: 다음 명령을 실행하여 Ingress 컨트롤러가 생성되었는지 확인합니다.

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

6.8.7. 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 오브젝트의 **scope**를 변경하려면, 해당 **IngressController** 오브젝트를 삭제한 후 다시 생성해야 합니다. CR(사용자 정의 리소스)을 생성한 후에는 **.spec.endpointPublishingStrategy.loadBalancer.scope** 매개변수를 변경할 수 없습니다.

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
```

6.8.8. 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
  ...
```

6.8.9. 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

- 와일드카드 정책을 구성합니다.
 - 다음 명령을 사용하여 **IngressController** 리소스를 편집합니다.

```
$ oc edit IngressController
```

- spec**에서 **wildcardPolicy** 필드를 **WildcardDisallowed** 또는 **WildcardAllowed**로 설정합니다.

```
spec:
  routeAdmission:
    wildcardPolicy: WildcardDisallowed # or WildcardAllowed
```

6.8.10. 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.

6.8.11. 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. 그렇지 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
```

6.9. 추가 리소스

- [사용자 정의 PKI 구성](#)

7장. 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.

7.1. 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.

7.2. 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>" }
  }
```

출력 예

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

- 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"]
```

7.3. 추가 리소스

- NodePort를 사용하여 수신 클러스터 트래픽 구성
- Network [config.openshift.io/v1]
- 서비스 [core/v1]

8장. 베어 메탈 클러스터에서 SCTP(STREAM CONTROL TRANSMISSION PROTOCOL) 사용

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

8.1. 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 **Service**를 정의해야 합니다.

8.1.1. 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 Pod의 포트 **80**에서 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
```

8.2. 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: 3.1.0
    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 Operator가 구성 변경 사항을 적용하는 동안 노드의 상태를 보려면 다음 명령을 입력합니다. 노드 상태가 **Ready**로 전환되면 구성 업데이트가 적용됩니다.

```
$ oc get nodes
```

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

SCTP 트래픽을 수신하는 애플리케이션으로 pod를 만들고 서비스와 연결한 다음, 노출된 서비스에 연결하여 SCTP가 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 YAML로 pod를 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 pod를 생성합니다.

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

2. SCTP 리스너 pod에 대한 서비스를 생성합니다.
 - a. 다음 YAML을 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 pod를 생성합니다.

- a. 다음 YAML을 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 Pod에 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 프로그래머에서 새 터미널 창 따는 태으 여이다

a. `oc get services sctp` 명령을 사용하여 클러스터 IP를 얻습니다.

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
```

9장. PTP 하드웨어 구성



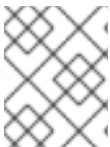
중요

Precision Time Protocol(PTP) 하드웨어는 기술 프리뷰 기능만 해당합니다. 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 <https://access.redhat.com/support/offerings/techpreview/>를 참조하십시오.

9.1. PTP 하드웨어 정보

OpenShift Container Platform에는 노드에서 Precision Time Protocol(PTP) 하드웨어를 사용하는 기능이 포함되어 있습니다. PTP 지원 하드웨어가 있는 클러스터의 노드에서 linuxptp 서비스를 구성할 수 있습니다.



참고

PTP Operator는 베어 메탈 인프라에서만 프로비저닝된 클러스터에서 PTP 가능 장치와 함께 작동합니다.

PTP Operator를 배포하여 OpenShift Container Platform 콘솔을 사용하여 PTP를 설치할 수 있습니다. PTP Operator는 linuxptp 서비스를 생성하고 관리합니다. Operator는 다음 기능을 제공합니다.

- 클러스터에서 PTP 지원 장치 검색.
- linuxptp 서비스의 구성 관리.

9.2. PTP 네트워크 장치의 자동 검색

PTP Operator는 **NodePtpDevice.ptp.openshift.io** CRD(custom resource definition)를 OpenShift Container Platform에 추가합니다. PTP Operator는 각 노드에서 PTP 가능 네트워크 장치를 클러스터에서 검색합니다. Operator는 호환 가능한 PTP 장치를 제공하는 각 노드에 대해 **NodePtpDevice** CR(사용자 정의 리소스)을 생성하고 업데이트합니다.

각 노드마다 하나의 CR이 작성되며 노드와 동일한 이름을 공유합니다. **.status.devices** 목록은 노드의 PTP 장치에 대한 정보를 제공합니다.

다음은 PTP Operator가 생성한 **NodePtpDevice** CR의 예입니다.

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2019-11-15T08:57:11Z"
  generation: 1
  name: dev-worker-0 1
  namespace: openshift-ptp 2
  resourceVersion: "487462"
  selfLink: /apis/ptp.openshift.io/v1/namespaces/openshift-ptp/nodeptpdevices/dev-worker-0
  uid: 08d133f7-aae2-403f-84ad-1fe624e5ab3f
```

```
spec: {}
status:
  devices: 3
  - name: eno1
  - name: eno2
  - name: ens787f0
  - name: ens787f1
  - name: ens801f0
  - name: ens801f1
  - name: ens802f0
  - name: ens802f1
  - name: ens803
```

- 1 **name** 매개변수의 값은 노드 이름과 같습니다.
- 2 CR은 PTP Operator에 의해 **openshift-ptp** 네임스페이스에 생성됩니다.
- 3 **devices** 컬렉션에는 노드에서 Operator가 검색한 모든 PTP 가능 장치 목록이 포함됩니다.

9.3. PTP OPERATOR 설치

클러스터 관리자는 OpenShift Container Platform CLI 또는 웹 콘솔을 사용하여 PTP Operator를 설치할 수 있습니다.

9.3.1. CLI: PTP Operator 설치

클러스터 관리자는 CLI를 사용하여 Operator를 설치할 수 있습니다.

사전 요구 사항

- PTP를 지원하는 하드웨어가 있는 노드로 베어 메탈 하드웨어에 설치된 클러스터
- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.

프로세스

1. PTP Operator 네임스페이스를 생성하려면 다음 명령을 입력합니다.

```
$ cat << EOF | oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  labels:
    name: openshift-ptp
    openshift.io/cluster-monitoring: "true"
EOF
```

2. 해당 Operator에 대한 Operator group을 생성하려면 다음 명령을 입력합니다.

```
$ cat << EOF | oc create -f -
```



```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
  - openshift-ptp
EOF

```

3. PTP Operator에 등록합니다.

- a. 다음 명령을 실행하여 OpenShift Container Platform의 주 버전과 부 버전을 환경 변수로 설정합니다. 이 변수는 다음 단계에서 **channel** 값으로 사용됩니다.

```

$ OC_VERSION=$(oc version -o yaml | grep openshiftVersion | \
  grep -o '[0-9]*[.][0-9]*' | head -1)

```

- b. PTP Operator에 대한 서브스크립션을 만들려면 다음 명령을 입력합니다.

```

$ cat << EOF | oc create -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "${OC_VERSION}"
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF

```

4. Operator가 설치되었는지 확인하려면 다음 명령을 입력합니다.

```

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

```

출력 예

Name	Phase
ptp-operator.4.4.0-202006160135	Succeeded

9.3.2. 웹 콘솔: PTP Operator 설치

클러스터 관리자는 웹 콘솔을 사용하여 Operator를 설치할 수 있습니다.



참고

이전 섹션에서 언급한 것처럼 네임스페이스 및 Operator group을 생성해야 합니다.

프로세스

1. OpenShift Container Platform 웹 콘솔을 사용하여 PTP Operator를 설치합니다.
 - a. OpenShift Container Platform 웹 콘솔에서 **Operator → OperatorHub**를 클릭합니다.
 - b. 사용 가능한 Operator 목록에서 **PTP Operator**를 선택한 다음 **설치**를 클릭합니다.
 - c. **Operator 설치** 페이지의 클러스터의 특정 네임스페이스에서 **openshift-ptp**를 선택합니다. 그런 다음, **설치**를 클릭합니다.
2. 선택 사항: PTP Operator가 성공적으로 설치되었는지 확인합니다.
 - a. **Operator → 설치된 Operator** 페이지로 전환합니다.
 - b. **PTP Operator**가 **openshift-ptp** 프로젝트에 **InstallSucceeded** 상태로 나열되어 있는지 확인합니다.



참고

설치 중에 Operator는 **실패** 상태를 표시할 수 있습니다. 나중에 **InstallSucceeded** 메시지와 함께 설치에 성공하면 이 **실패** 메시지를 무시할 수 있습니다.

Operator가 설치된 것으로 나타나지 않으면 다음과 같이 추가 트러블슈팅을 수행하십시오.

- **Operator → 설치된 Operator** 페이지로 이동하고 **Operator** 서브스크립션 및 **설치 계획** 탭의 **상태**에 장애나 오류가 있는지 검사합니다.
- **Workloads → Pod** 페이지로 이동하여 **openshift-ptp** 프로젝트에서 Pod 로그를 확인합니다.

9.4. LINUXPTP 서비스 구성

PTP Operator는 **PtpConfig.ptp.openshift.io** CRD(custom resource definition)를 OpenShift Container Platform에 추가합니다. **PtpConfig** CR(사용자 정의 리소스)을 생성하여 Linuxptp 서비스(ptp4l, phc2sys)를 구성할 수 있습니다.

사전 요구 사항

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

프로세스

1. 다음 **PtpConfig** CR을 생성한 다음 YAML을 **<name>-ptp-config.yaml** 파일에 저장합니다. **<name>**을 이 구성의 이름으로 바꿉니다.

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <name> 1
  namespace: openshift-ptp 2
spec:
    
```

```

profile: 3
- name: "profile1" 4
  interface: "ens787f1" 5
  ptp4lOpts: "-s -2" 6
  phc2sysOpts: "-a -r" 7
recommend: 8
- profile: "profile1" 9
  priority: 10 10
  match: 11
- nodeLabel: "node-role.kubernetes.io/worker" 12
  nodeName: "dev-worker-0" 13

```

- 1 **PtpConfig** CR의 이름을 지정합니다.
- 2 PTP Operator가 설치된 네임스페이스를 지정합니다.
- 3 하나 이상의 **profile** 오브젝트의 배열을 지정합니다.
- 4 프로필 오브젝트를 고유하게 식별하는 데 사용되는 프로필 오브젝트의 이름을 지정합니다.
- 5 **ptp4l** 서비스에서 사용할 네트워크 인터페이스 이름을 지정합니다(예: **ens787f1**).
- 6 **ptp4l** 서비스에 대한 시스템 구성 옵션을 지정합니다(예: **-s -2**). 인터페이스 이름 **-i <interface>** 및 서비스 구성 파일 **-f /etc/ptp4l.conf**는 자동으로 추가되므로 포함하지 않아야 합니다.
- 7 **phc2sys** 서비스에 대한 시스템 구성 옵션을 지정합니다(예: **-a -r**).
- 8 **profile**이 노드에 적용되는 방법에 대한 규칙을 정의하는 하나 이상의 **recommend** 오브젝트 배열을 지정합니다.
- 9 **profile** 섹션에 정의된 **profile** 오브젝트 이름을 지정합니다.
- 10 0에서 99 사이의 정수 값으로 **priority**를 지정합니다. 숫자가 클수록 우선순위가 낮으므로 우선순위 99는 우선순위 10보다 낮습니다. **match** 필드에 정의된 규칙에 따라 노드를 여러 프로필과 일치시킬 수 있는 경우 우선순위가 높은 프로필이 해당 노드에 적용됩니다.
- 11 **nodeLabel** 또는 **nodeName**으로 일치 규칙을 지정합니다.
- 12 노드 오브젝트에서 **node.Labels**의 **key**로 **nodeLabel**을 지정합니다.
- 13 노드 오브젝트에서 **node.Name**으로 **nodeName**을 지정합니다.

2. 다음 명령을 실행하여 CR을 생성합니다.

```
$ oc create -f <filename> 1
```

- 1 **<filename>**을 이전 단계에서 생성한 파일 이름으로 바꿉니다.

3. 선택 사항: **PtpConfig** 프로필이 **nodeLabel** 또는 **node Name** 과 일치하는 노드에 적용되는지 확인합니다.

```
$ 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	192.168.111.15	dev-worker-0
<none>	<none>					
linuxptp-daemon-tdspf	1/1	Running	0	43m	192.168.111.11	dev-master-0
<none>	<none>					
ptp-operator-657bbb64c8-2f8sj	1/1	Running	0	43m	10.128.0.116	dev-master-0
<none>	<none>					

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp
l1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
l1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
l1115 09:41:17.117607 4143292 daemon.go:110] -----
l1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1 1
l1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1 2
l1115 09:41:17.117620 4143292 daemon.go:102] Ptp4IOpts: -s -2 3
l1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r 4
l1115 09:41:17.117626 4143292 daemon.go:116] -----
l1115 09:41:18.117934 4143292 daemon.go:186] Starting phc2sys...
l1115 09:41:18.117985 4143292 daemon.go:187] phc2sys cmd: &{Path:/usr/sbin/phc2sys
Args:[/usr/sbin/phc2sys -a -r] Env:[] Dir: Stdin:<nil> Stdout:<nil> Stderr:<nil> ExtraFiles:[]
SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil> lookPathErr:<nil> finished:false
childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[] errch:<nil> waitDone:<nil>}
l1115 09:41:19.118175 4143292 daemon.go:186] Starting ptp4l...
l1115 09:41:19.118209 4143292 daemon.go:187] ptp4l cmd: &{Path:/usr/sbin/ptp4l Args:
[/usr/sbin/ptp4l -m -f /etc/ptp4l.conf -i ens787f1 -s -2] Env:[] Dir: Stdin:<nil> Stdout:<nil>
Stderr:<nil> ExtraFiles:[] SysProcAttr:<nil> Process:<nil> ProcessState:<nil> ctx:<nil>
lookPathErr:<nil> finished:false childFiles:[] closeAfterStart:[] closeAfterWait:[] goroutine:[]
errch:<nil> waitDone:<nil>}
ptp4l[102189.864]: selected /dev/ptp5 as PTP clock
ptp4l[102189.886]: port 1: INITIALIZING to LISTENING on INIT_COMPLETE
ptp4l[102189.886]: port 0: INITIALIZING to LISTENING on INIT_COMPLETE
```

- 1 Profile Name은 dev-worker-0 노드에 적용되는 이름입니다.
- 2 Interface는 profile1 인터페이스 필드에 지정된 PTP 장치입니다. ptp4l 서비스는 이 인터페이스에서 실행됩니다.
- 3 Ptp4IOpts는 profile1 Ptp4IOpts 필드에 지정된 ptp4l sysconfig 옵션입니다.
- 4 Phc2sysOpts는 profile1 Phc2sysOpts 필드에 지정된 phc2sys sysconfig 옵션입니다.

10장. 네트워크 정책

10.1. 네트워크 정책 정의

클러스터 관리자는 클러스터의 pod로 트래픽을 제한하는 네트워크 정책을 정의할 수 있습니다.

10.1.1. 네트워크 정책 정의

Kubernetes 네트워크 정책을 지원하는 CNI(Kubernetes Container Network Interface) 플러그인을 사용하는 클러스터에서 네트워크 격리는 **NetworkPolicy** 개체에 의해서만 제어됩니다. OpenShift Container Platform 4.6에서 OpenShift SDN은 기본 네트워크 격리 모드에서 네트워크 정책의 사용을 지원합니다.



참고

OpenShift SDN 클러스터 네트워크 공급자를 사용할 경우 네트워크 정책과 관련하여 다음과 같은 제한 사항이 적용됩니다.

- 송신 필드에서 지정한 **egress** 네트워크 정책은 지원되지 않습니다.
- IPBlock은 네트워크 정책에서 지원되지만 **except** 절에는 지원되지 않습니다. **except** 절이 포함된 IPBlock 섹션이 포함된 정책을 생성하면 SDN Pod 로그가 경고를 생성하고 해당 정책의 전체 IPBlock 섹션이 무시됩니다.



주의

네트워크 정책은 호스트 네트워크 네임스페이스에 적용되지 않습니다. 호스트 네트워크가 활성화된 Pod는 네트워크 정책 규칙의 영향을 받지 않습니다.

기본적으로 네트워크 정책 모드에서는 다른 Pod 및 네트워크 끝점에서 프로젝트의 모든 Pod에 액세스할 수 있습니다. 프로젝트에서 하나 이상의 Pod를 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 Pod가 일치하면 Pod는 해당 **NetworkPolicy** 오브젝트 중 하나 이상에서 허용되는 연결만 허용합니다. **NetworkPolicy** 오브젝트가 선택하지 않은 Pod에 완전히 액세스할 수 있습니다.

다음 예제 **NetworkPolicy** 오브젝트는 다양한 시나리오 지원을 보여줍니다.

- 모든 트래픽 거부:
기본적으로 프로젝트를 거부하려면 모든 Pod와 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: Pod가 동일한 프로젝트 내 다른 Pod의 연결은 수락하지만 다른 프로젝트에 속하는 Pod의 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 Pod에 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 **443**에 대한 연결이 허용됩니다.

10.1.2. 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. 포트 라벨은 IP 주소 수준에 존재하지 않으므로 네트워크 정책은 **podSelector**로 선택한 모든 포트 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.

10.1.3. 다음 단계

- [네트워크 정책 생성](#)
- 선택 사항: [기본 네트워크 정책 정의](#)

10.1.4. 추가 리소스

- [프로젝트 및 네임스페이스](#)
- [다중 테넌트 네트워크 정책 구성](#)
- [NetworkPolicy API](#)

10.2. 네트워크 정책 생성

admin 역할이 있는 사용자는 네임스페이스에 대한 네트워크 정책을 생성할 수 있습니다.

10.2.1. 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 **mode**를 사용하여 **OVN-Kubernetes** 네트워크 공급자 또는 **OpenShift SDN** 네트워크 공급자와 같은 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자를 사용합니다. **NetworkPolicy**가 설정되어 있습니다. 이 모드는 OpenShift SDN의 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 Pod에서 수신 거부

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
spec:
  podSelector:
    ingress: []
```

동일한 네임 스페이스에 있는 모든 Pod의 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 네임스페이스를 지정합니다.

출력 예

```
networkpolicy "default-deny" created
```

10.2.2. NetworkPolicy 오브젝트 예

다음은 예제 NetworkPolicy 오브젝트에 대한 주석입니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
```

```

name: allow-27107 ❶
spec:
  podSelector: ❷
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: ❸
      matchLabels:
        app: app
  ports: ❹
  - protocol: TCP
    port: 27017
    
```

- ❶ NetworkPolicy 오브젝트의 이름입니다.
- ❷ 정책이 적용되는 Pod를 설명하는 선택기입니다. 정책 오브젝트는 NetworkPolicy 오브젝트를 정의하는 프로젝트에서 Pod만 선택할 수 있습니다.
- ❸ 정책 오브젝트에서 수신 트래픽을 허용하는 Pod와 일치하는 선택기입니다. 선택기는 NetworkPolicy와 동일한 네임스페이스의 Pod와 일치합니다.
- ❹ 트래픽을 허용할 하나 이상의 대상 포트 목록입니다.

10.3. 네트워크 정책 보기

admin 역할이 있는 사용자는 네임스페이스에 대한 네트워크 정책을 볼 수 있습니다.

10.3.1. 네트워크 정책 보기

네임스페이스에서 네트워크 정책을 검사할 수 있습니다.



참고

cluster-admin 역할을 가진 사용자로 로그인하면 클러스터의 모든 네트워크 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 **NetworkPolicy** 오브젝트를 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 네임스페이스를 지정합니다.

예를 들면 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
```

10.3.2. 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
```

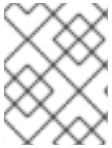
- 1 NetworkPolicy 오브젝트의 이름입니다.
- 2 정책이 적용되는 Pod를 설명하는 선택기입니다. 정책 오브젝트는 NetworkPolicy 오브젝트를 정의하는 프로젝트에서 Pod만 선택할 수 있습니다.
- 3 정책 오브젝트에서 수신 트래픽을 허용하는 Pod와 일치하는 선택기입니다. 선택기는 NetworkPolicy와 동일한 네임스페이스의 Pod와 일치합니다.
- 4 트래픽을 허용할 하나 이상의 대상 포트 목록입니다.

10.4. 네트워크 정책 편집

관리자 역할이 있는 사용자는 네임스페이스에 대한 기존 네트워크 정책을 편집할 수 있습니다.

10.4.1. 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 **mode**를 사용하여 **OVN-Kubernetes** 네트워크 공급자 또는 **OpenShift SDN** 네트워크 공급자와 같은 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자를 사용합니다. **NetworkPolicy**가 설정되어 있습니다. 이 모드는 OpenShift SDN의 기본값입니다.
- OpenShift CLI(**oc**)를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 존재하는 네임스페이스에서 작업하고 있습니다.

절차

1. 선택 사항: 네임스페이스의 네트워크 정책 오브젝트를 나열하려면 다음 명령을 입력합니다.

```
$ oc get networkpolicy -n <namespace>
```

다음과 같습니다.

<namespace>

선택 사항: 오브젝트가 현재 네임스페이스와 다른 네임스페이스에 정의된 경우 네임스페이스를 지정합니다.

2. **NetworkPolicy** 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 네임스페이스를 지정합니다.

<policy_file>

네트워크 정책이 포함된 파일의 이름을 지정합니다.

- **NetworkPolicy** 오브젝트를 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 네임스페이스를 지정합니다.

3. **NetworkPolicy** 오브젝트가 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 네임스페이스를 지정합니다.

10.4.2. 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 Pod를 설명하는 선택기입니다. 정책 오브젝트는 NetworkPolicy 오브젝트를 정의하는 프로젝트에서 Pod만 선택할 수 있습니다.
- 3 정책 오브젝트에서 수신 트래픽을 허용하는 Pod와 일치하는 선택기입니다. 선택기는 NetworkPolicy와 동일한 네임스페이스의 Pod와 일치합니다.
- 4 트래픽을 허용할 하나 이상의 대상 포트 목록입니다.

10.4.3. 추가 리소스

- [네트워크 정책 생성](#)

10.5. 네트워크 정책 삭제

admin 역할이 있는 사용자는 네임스페이스에서 네트워크 정책을 삭제할 수 있습니다.

10.5.1. 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 **mode**를 사용하여 **OVN-Kubernetes** 네트워크 공급자 또는 **OpenShift SDN** 네트워크 공급자와 같은 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자를 사용합니다. **NetworkPolicy**가 설정되어 있습니다. 이 모드는 OpenShift SDN의 기본값입니다.
- OpenShift CLI(**oc**)를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 존재하는 네임스페이스에서 작업하고 있습니다.

절차

- **NetworkPolicy** 오브젝트를 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 네임스페이스를 지정합니다.

출력 예

```
networkpolicy.networking.k8s.io/allow-same-namespace deleted
```

10.6. 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.

10.6.1. 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 CLI를 사용하여 프로젝트 구성 리소스를 편집합니다.

- 웹 콘솔에 액세스:
 - i. **관리** → **클러스터 설정**으로 이동합니다.
 - ii. **전역 구성**을 클릭하여 모든 구성 리소스를 확인합니다.
 - iii. **프로젝트 항목**을 찾아 **YAML 편집**을 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.

10.6.2. 새 프로젝트 템플릿에 네트워크 정책 추가

클러스터 관리자는 네트워크 정책을 새 프로젝트의 기본 템플릿에 추가할 수 있습니다. OpenShift Container Platform은 프로젝트의 템플릿에 지정된 모든 **NetworkPolicy** 개체를 자동으로 생성합니다.

사전 요구 사항

- 클러스터는 **NetworkPolicy** 오브젝트를 지원하는 기본 CNI 네트워크 공급자(예: **mode**)를 사용하는 **OpenShift SDN** 네트워크 공급자를 사용합니다. **NetworkPolicy** 가 설정되어 있습니다. 이 모드는 OpenShift SDN의 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 구성된 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.



중요

OVN-Kubernetes 네트워크 공급자 플러그인의 경우 Ingress 컨트롤러가 **HostNetwork** 끝점 게시 전략을 사용하도록 구성된 경우 Ingress 트래픽이 허용되고 기타 모든 트래픽이 거부되도록 네트워크 정책을 적용하는 방법은 지원되지 않습니다.

```

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
...

```

3. 선택 사항: 다음 명령을 실행하여 새 프로젝트를 생성하여 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

```

10.7. 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 네트워킹 정책 모드가 설정된 네트워크 격리를 제공합니다.

10.7.1. 네트워크 정책을 사용하여 다중 테넌트 격리 구성

다른 프로젝트 네임스페이스의 Pod 및 서비스에서 격리하도록 프로젝트를 구성할 수 있습니다.

사전 요구 사항

- 클러스터는 **mode**를 사용하여 **OVN-Kubernetes** 네트워크 공급자 또는 **OpenShift SDN** 네트워크 공급자와 같은 **NetworkPolicy** 오브젝트를 지원하는 클러스터 네트워크 공급자를 사용합니다. **NetworkPolicy**가 설정되어 있습니다. 이 모드는 OpenShift SDN의 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
```

- 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
```

2. 선택 사항: 네트워크 정책이 현재 프로젝트에 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
```

10.7.2. 다음 단계

- [기본 네트워크 정책 정의](#)

10.7.3. 추가 리소스

- [OpenShift SDN 네트워크 격리 모드](#)

11장. 다중 네트워킹

11.1. 다중 네트워킹 이해하기

Kubernetes에서 컨테이너 네트워킹은 컨테이너 네트워킹 인터페이스(CNI)를 구현하는 네트워킹 플러그인에 위임됩니다.

OpenShift Container Platform은 Multus CNI 플러그인을 사용하여 CNI 플러그인 체인을 허용합니다. 클러스터 설치 중에 기본 pod 네트워킹을 구성합니다. 기본 네트워킹은 클러스터의 모든 일반 네트워킹 트래픽을 처리합니다. 사용 가능한 CNI 플러그인을 기반으로 추가 네트워킹을 정의하고 이러한 네트워킹 중 하나 이상을 pod에 연결할 수 있습니다. 필요에 따라 클러스터에 2개 이상의 추가 네트워킹을 정의할 수 있습니다. 따라서 스위칭 또는 라우팅과 같은 네트워킹 기능을 제공하는 pod를 구성할 때 유연성이 제공됩니다.

11.1.1. 추가 네트워킹 사용 시나리오

데이터 플레인 및 컨트롤 플레인 분리를 포함하여 네트워킹 격리가 필요한 상황에서 추가 네트워킹을 사용할 수 있습니다. 네트워킹 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 네트워킹 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 네트워킹을 사용하여 클러스터 전체의 연결을 유지합니다. 모든 pod에는 클러스터 전체 pod 네트워킹에 연결된 **eth0** 인터페이스가 있습니다. **oc exec -it <pod_name> -- ip a** 명령을 사용하여 pod의 인터페이스를 확인할 수 있습니다. Multus CNI를 사용하는 네트워킹 인터페이스를 추가하는 경우 이름은 **net1, net2, ..., netN**.

Pod에 추가 네트워킹 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.

11.1.2. OpenShift Container Platform의 그룹은 중첩되지 않습니다.

OpenShift Container Platform은 클러스터에서 추가 네트워킹을 생성하기 위해 다음과 같은 CNI 플러그인을 제공합니다.

- **bridge**: 동일한 호스트의 포트가 서로 및 호스트와 통신할 수 있도록 **브리지 기반 추가 네트워킹**을 구성합니다.
- **host-device**: 포트가 호스트 시스템의 물리적 이더넷 네트워킹 장치에 액세스할 수 있도록 호스트 장치 추가 네트워킹을 구성합니다.
- **ipvlan**: macvlan 기반 추가 네트워킹과 유사하게 호스트의 pod가 해당 호스트의 다른 호스트 및 Pod와 통신할 수 있도록 **ipvlan** 기반 추가 네트워킹을 구성합니다. macvlan 기반 추가 네트워킹과 달리 각 pod는 상위 물리적 네트워킹 인터페이스와 동일한 MAC 주소를 공유합니다.
- **macvlan**: 호스트의 pod가 실제 네트워킹 인터페이스를 사용하여 해당 호스트의 다른 호스트 및 포트와 통신할 수 있도록 **macvlan** 기반 추가 네트워킹을 구성합니다. macvlan 기반 추가 네트워킹에 연결된 각 pod에는 고유한 MAC 주소가 제공됩니다.

- **SR-IOV**: Pod 가 호스트 시스템의 SR-IOV 가능 하드웨어에서 VF(가상 기능) 인터페이스에 연결할 수 있도록 SR-IOV 기반 추가 네트워크를 구성합니다.

11.2. 추가 네트워크 구성

클러스터 관리자는 클러스터에 대한 추가 네트워크를 구성할 수 있습니다. 지원되는 네트워크 유형은 다음과 같습니다.

- [브릿지](#)
- [호스트 장치](#)
- [IPVLAN](#)
- [MACVLAN](#)

11.2.1. 추가 네트워크 관리 접근법

두 가지 방법으로 추가 네트워크의 라이프사이클을 관리할 수 있습니다. 각 접근 방식은 상호 배타적이며 한 번에 추가 네트워크를 관리하는 데 한 가지 접근 방식만 사용할 수 있습니다. 두 방법 모두의 경우 추가 네트워크는 구성하는 CNI(컨테이너 네트워크 인터페이스) 플러그인에 의해 관리됩니다.

추가 네트워크의 경우 IP 주소가 추가 네트워크의 일부로 구성하는 IPAM(IP 주소 관리) CNI 플러그인을 통해 프로비저닝됩니다. IPAM 플러그인은 DHCP 및 정적 할당을 비롯한 다양한 IP 주소 할당 방법을 지원합니다.

- CNO(Cluster Network Operator) 구성을 수정합니다. CNO는 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성하고 관리합니다. CNO는 개체 라이프사이클을 관리하는 것 외에도 DHCP에 할당된 IP 주소를 사용하는 추가 네트워크에 DHCP를 사용할 수 있습니다.
- YAML 매니페스트 적용: **NetworkAttachmentDefinition** 오브젝트를 생성하여 추가 네트워크를 직접 관리할 수 있습니다. 이 접근 방식을 사용하면 CNI 플러그인의 체인을 사용할 수 있습니다.

11.2.2. 추가 네트워크 연결 구성

추가 네트워크는 **k8s.cni.cncf.io** API 그룹의 **NetworkAttachmentDefinition** API를 통해 구성됩니다. API의 구성은 다음 표에 설명되어 있습니다.

표 11.1. **NetworkAttachmentDefinition** API fields

필드	유형	설명
metadata.name	string	추가 네트워크의 이름입니다.
metadata.namespace	string	오브젝트와 연결된 네임스페이스입니다.
spec.config	string	JSON 형식의 CNI 플러그인 구성입니다.

11.2.2.1. Cluster Network Operator를 통한 추가 네트워크 구성

추가 네트워크 연결 구성은 CNO(Cluster Network Operator) 구성의 일부로 지정됩니다.

다음 YAML은 CNO로 추가 네트워크를 관리하기 위한 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
    rawCNConfig: |- 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.

11.2.2.2. YAML 매니페스트에서 추가 네트워크 구성

추가 네트워크의 구성은 다음 예와 같이 YAML 구성 파일에서 지정합니다.

```

apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: <name> ①
spec:
  config: |- ②
    {
      ...
    }

```

- ① 생성 중인 추가 네트워크 연결의 이름입니다.
- ② JSON 형식의 CNI 플러그인 구성입니다.

11.2.3. 추가 네트워크 유형에 대한 구성

추가 네트워크의 특정 구성 필드는 다음 섹션에 설명되어 있습니다.

11.2.3.1. 브리지 추가 네트워크 구성

다음 오브젝트는 브릿지 CNI 플러그인의 구성 매개변수를 설명합니다.

표 11.2. 브릿지 CNI 플러그인 JSON 구성 오브젝트

필드	유형	설명
cniVersion	string	CNI 사양 버전입니다. redhat.1 값이 필요합니다.
name	string	CNO 구성에 대해 이전에 제공한 name 매개변수의 값입니다.
type	string	
bridge	string	사용할 가상 브릿지의 이름을 지정합니다. 브릿지 인터페이스가 호스트에 없으면 생성됩니다. 기본값은 cni0 입니다.
ipam	object	IPAM CNI 플러그인의 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.
ipMasq	boolean	가상 네트워크에서 전송되는 트래픽에 IP 마스커레이딩을 사용하려면 true 로 설정합니다. 모든 트래픽의 소스 IP 주소가 브리지의 IP 주소로 다시 작성됩니다. 브리지에 IP 주소가 없으면 이 설정이 적용되지 않습니다. 기본값은 false 입니다.
isGateway	boolean	브리지에 IP 주소를 할당하려면 true 로 설정합니다. 기본값은 false 입니다.
isDefaultGateway	boolean	브릿지를 가상 네트워크의 기본 게이트웨이로 구성하려면 true 로 설정합니다. 기본값은 false 입니다. isDefaultGateway 가 true 로 설정되면 isGateway 도 자동으로 true 로 설정됩니다.
forceAddress	boolean	이전에 할당된 IP 주소를 가상 브리지에 할당할 수 있도록 하려면 true 로 설정합니다. false 로 설정하면 중첩되는 하위 집합의 IPv4 주소 또는 IPv6 주소가 가상 브릿지에 지정되는 경우 오류가 발생합니다. 기본값은 false 입니다.
hairpinMode	boolean	가상 브리지가 수신한 가상 포트를 통해 이더넷 프레임을 다시 보낼 수 있도록 하려면 true 로 설정합니다. 이 모드를 반사 릴레이 라고도 합니다. 기본값은 false 입니다.
promiscMode	boolean	브릿지에서 무차별 모드를 사용하려면 true 로 설정합니다. 기본값은 false 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.

11.2.3.1.1. 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"
  }
}
```

11.2.3.2. 호스트 장치 추가 네트워크 구성



참고

device, **hwaddr**, **kernelpath** 또는 **pciBusID** 매개변수 중 하나만 설정하여 네트워크 장치를 지정합니다.

다음 오브젝트는 호스트 장치 CNI 플러그인의 구성 매개변수를 설명합니다.

표 11.3. 호스트 장치 CNI 플러그인 JSON 구성 오브젝트

필드	유형	설명
cniVersion	string	CNI 사양 버전입니다. redhat.1 값이 필요합니다.
name	string	CNO 구성에 대해 이전에 제공한 name 매개변수의 값입니다.
type	string	구성할 CNI 플러그인의 이름: host-device 입니다.
device	string	선택 사항: 장치 이름(예: eth0).
hwaddr	string	선택 사항: 장치 하드웨어 MAC 주소입니다.
kernelpath	string	선택 사항: Linux 커널 장치 경로(예: /sys/devices/pci0000:00/0000:00:1f.6).
pciBusID	string	선택 사항: 네트워크 장치의 PCI 주소(예: 0000:00:1f.6).
ipam	object	IPAM CNI 플러그인의 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.

11.2.3.2.1. 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",
"ipam": {
  "type": "dhcp"
}
}

```

11.2.3.3. IPVLAN 추가 네트워크 구성

다음 오브젝트는 IPVLAN CNI 플러그인의 구성 매개변수를 설명합니다.

표 11.4. IPVLAN CNI 플러그인 JSON 구성 오브젝트

필드	유형	설명
cniVersion	string	CNI 사양 버전입니다. redhat.1 값이 필요합니다.
name	string	CNO 구성에 대해 이전에 제공한 name 매개변수의 값입니다.
type	string	구성할 CNI 플러그인의 이름: ipvlan .
mode	string	가상 네트워크의 작동 모드입니다. 값은 I2 , I3 또는 I3s 여야 합니다. 기본값은 I2 입니다.
master	string	네트워크 연결과 연결할 이더넷 인터페이스입니다. 마스터 를 지정하지 않으면 기본 네트워크 경로에 대한 인터페이스가 사용됩니다.
mtu	integer	최대 전송 단위(MTU)를 지정된 값으로 설정합니다. 기본값은 커널에 의해 자동으로 설정됩니다.
ipam	object	IPAM CNI 플러그인의 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다. dhcp 를 지정하지 마십시오. IPVLAN 인터페이스가 호스트 인터페이스와 MAC 주소를 공유하므로 DHCP를 사용하여 IPVLAN 구성은 지원되지 않습니다.

11.2.3.3.1. 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"
      }
    ]
  }
}

```

```

    ]
  }
}

```

11.2.3.4. MACVLAN 추가 네트워크 구성

다음 오브젝트는 macvlan CNI 플러그인의 구성 매개변수를 설명합니다.

표 11.5. MACVLAN CNI 플러그인 JSON 구성 오브젝트

필드	유형	설명
cniVersion	string	CNI 사양 버전입니다. redhat.1 값이 필요합니다.
name	string	CNO 구성에 대해 이전에 제공한 name 매개변수의 값입니다.
type	string	구성할 CNI 플러그인의 이름: macvlan .
mode	string	가상 네트워크에서 트래픽 가시성을 구성합니다. bridge , passthru , private 또는 vepa 중 하나여야 합니다. 값을 입력하지 않으면 기본값은 bridge 입니다.
master	string	가상 인터페이스와 연결할 이더넷, 본딩 또는 VLAN 인터페이스입니다. 값을 지정하지 않으면 호스트 시스템의 기본 이더넷 인터페이스가 사용됩니다.
mtu	string	지정된 값으로 MTU(최대 전송 단위)입니다. 기본값은 커널에 의해 자동으로 설정됩니다.
ipam	object	IPAM CNI 플러그인의 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.

11.2.3.4.1. 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"
  }
}

```

11.2.4. 추가 네트워크에 대한 IP 주소 할당 구성

IPAM(IP 주소 관리) CNI(Container Network Interface) 플러그인은 다른 CNI 플러그인의 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

11.2.4.1. 고정 IP 주소 할당 구성

다음 표에서는 고정 IP 주소 할당 구성에 대해 설명합니다.

표 11.6. IPAM 정적 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 static 이 필요합니다.
주소	array	가상 인터페이스에 할당할 IP 주소를 지정하는 개체 배열입니다. IPv4 및 IPv6 IP 주소가 모두 지원됩니다.
routes	array	Pod 내부에서 구성할 경로를 지정하는 오브젝트 배열입니다.
DNS	array	선택 사항: DNS 구성을 지정하는 개체 배열입니다.

addresses 배열에는 다음 필드가 있는 오브젝트가 필요합니다.

표 11.7. ipam.addresses[] 배열

필드	유형	설명
address	string	지정하는 IP 주소 및 네트워크 접두사입니다. 예를 들어 10.10.21.10/24 를 지정하면 추가 네트워크에 IP 주소 10.10.21.10 이 할당되고 넷마스크는 255.255.255.0 입니다.
gateway	string	송신 네트워크 트래픽을 라우팅할 기본 게이트웨이입니다.

표 11.8. ipam.routes[] 배열

필드	유형	설명
dst	string	CIDR 형식의 IP 주소 범위(예: 기본 경로의 경우 192.168.17.0/24 또는 0.0.0.0/0)입니다.
gw	string	네트워크 트래픽이 라우팅되는 게이트웨이입니다.

표 11.9. ipam.dns 오브젝트

필드	유형	설명
nameservers	array	DNS 쿼리를 보낼 하나 이상의 IP 주소 배열입니다.
domain	array	호스트 이름에 추가할 기본 도메인입니다. 예를 들어 도메인이 example.com 으로 설정되면 example-host 에 대한 DNS 조회 쿼리가 example-host.example.com 으로 다시 작성됩니다.
search	array	DNS 조회 쿼리 중에 규정되지 않은 호스트 이름(예: example-host)에 추가할 도메인 이름 배열입니다.

고정 IP 주소 할당 구성 예

```
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}
```

11.2.4.2. DHCP(동적 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 11.10. IPAM DHCP 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 dhcp 가 필요합니다.

DHCP(동적 IP 주소) 할당 구성 예

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

11.2.4.3. 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 Whereabouts를 사용한 동적 IP 주소 할당 구성을 설명합니다.

표 11.11. IPAM whereabouts 구성 개체

필드	유형	설명
type	string	IPAM 주소 유형입니다. whereabouts 가 필요한 값입니다.
범위	string	CIDR 표기법의 IP 주소 및 범위입니다. IP 주소는 이 주소 범위 내에서 할당됩니다.
제외	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"
    ]
  }
}
```

11.2.5. Cluster Network Operator로 추가 네트워크 연결 생성

CNO(Cluster Network Operator)는 추가 네트워크 정의를 관리합니다. 생성할 추가 네트워크를 지정하면 CNO가 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성합니다.



중요

Cluster Network Operator가 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. CNO 구성을 편집하려면 다음 명령을 입력합니다.

```
$ oc edit networks.operator.openshift.io cluster
```

2. 다음 예제 CR과 같이 생성할 추가 네트워크의 구성을 추가하여 생성 중인 CR을 수정합니다.

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
```

```
spec:
  # ...
  additionalNetworks:
  - name: tertiary-net
    namespace: project2
    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"
            }
          ]
        }
      }
    }
```

3. 변경 사항을 저장하고 텍스트 편집기를 종료하여 변경 사항을 커밋합니다.

검증

- CNO가 다음 명령을 실행하여 NetworkAttachmentDefinition 오브젝트를 생성했는지 확인합니다. 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

```
NAME          AGE
test-network-1 14m
```

11.2.6. 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.

11.3. 추가 네트워크에 POD 연결

클러스터 사용자는 pod를 추가 네트워크에 연결할 수 있습니다.

11.3.1. 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>,...] ❶

```

- ❶ 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>", ❶
          "namespace": "<namespace>", ❷
          "default-route": ["<default-route>"] ❸
        }
      ]

```

- ❶ **NetworkAttachmentDefinition** 오브젝트에서 정의한 추가 네트워크의 이름을 지정합니다.
- ❷ **NetworkAttachmentDefinition** 오브젝트가 정의된 네임스페이스를 지정합니다.
- ❸ 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. 선택 사항: **Pod** CR에 주석이 있는지 확인하려면 다음 명령을 입력하고 **<name>**을 Pod 이름으로 바꿉니다.

```
$ 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: |- ❶
      [
        {
          "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** 매개변수는 JSON 오브젝트 배열입니다. 각 오브젝트는 Pod에 연결된 추가 네트워크의 상태를 설명합니다. 주석 값은 일반 텍스트 값으로 저장됩니다.

11.3.1.1. Pod별 주소 지정 및 라우팅 옵션 지정

추가 네트워크에 Pod를 연결할 때 특정 Pod에서 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

- Pod는 추가 네트워크와 동일한 네임스페이스에 있어야 합니다.
- OpenShift 명령줄 인터페이스(**oc**)를 설치합니다.
- 클러스터에 로그인해야 합니다.

프로세스

주소 지정 및/또는 라우팅 옵션을 지정하는 동안 추가 네트워크에 Pod를 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 라우팅될 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.

Pod의 고정 IP 주소 또는 MAC 주소를 설정하려면 JSON 형식의 주석을 사용하면 됩니다. 이를 위해서는 이러한 기능을 특별하게 허용하는 네트워크를 생성해야 합니다. 이는 다음과 같이 CNO의 rawCNICConfig에서 지정할 수 있습니다.

1. 다음 명령을 실행하여 CNO CR을 편집합니다.

```
$ oc edit networks.operator.openshift.io cluster
```

다음 YAML은 CNO의 구성 매개변수를 설명합니다.

CNO(Cluster Network Operator) YAML 구성

```
name: <name> 1
namespace: <namespace> 2
rawCNIConfig: '{ 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 기능 사용을 요청하도록 지정합니다.

- 4 macvlan 플러그인이 사용하는 인터페이스를 지정합니다.
- 5 CNI 플러그인의 정적 MAC 주소 기능을 활성화하기 위한 요청이 이루어지도록 지정합니다.

그런 다음 위의 네트워크 연결을 키와 함께 JSON 형식 주석에서 참조하여 지정된 Pod에 할당할 고정 IP 및 MAC 주소를 지정할 수 있습니다.

다음을 사용하여 Pod를 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 **rawCNIConfig**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 Pod의 IP 주소 및 MAC 속성을 확인하려면 **oc** 명령을 사용하여 Pod에서 **ip** 명령을 실행합니다.

```
$ oc exec -it <pod_name> -- ip a
```

11.4. 추가 네트워크에서 POD 제거

클러스터 사용자는 추가 네트워크에서 Pod를 제거할 수 있습니다.

11.4.1. 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.

11.5. 추가 네트워크 편집

클러스터 관리자는 기존 추가 네트워크의 구성을 수정할 수 있습니다.

11.5.1. 추가 네트워크 연결 정의 수정

클러스터 관리자는 기존 추가 네트워크를 변경할 수 있습니다. 추가 네트워크에 연결된 기존 Pod는 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) CR을 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: CNO가 다음 명령을 실행하여 **NetworkAttachmentDefinition** 오브젝트를 업데이트했는지 확인합니다. **<network-name>**을 표시할 추가 네트워크의 이름으로 변경합니다. CNO가 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"]}}} }
```

11.6. 추가 네트워크 제거

클러스터 관리자는 추가 네트워크의 연결을 제거할 수 있습니다.

11.6.1. 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.

프로세스

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 CR을 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: 다음 명령을 실행하여 추가 네트워크 CR이 삭제되었는지 확인합니다.

```
$ oc get network-attachment-definition --all-namespaces
```

12장. 하드웨어 네트워킹

12.1. SR-IOV(SINGLE ROOT I/O VIRTUALIZATION) 하드웨어 네트워킹 정보

SR-IOV(Single Root I/O Virtualization) 사양은 단일 장치를 여러 Pod와 공유할 수 있는 PCI 장치 할당 유형의 표준입니다.

SR-IOV를 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 장치 드라이버는 컨테이너에서 VF가 노출되는 방식을 결정합니다.

- **netdevice** 드라이버: 컨테이너의 **netns**에 있는 일반 커널 네트워크 장치
- **vfio-pci** 드라이버: 컨테이너에 마운트된 문자 장치

높은 대역폭 또는 짧은 대기 시간이 필요한 애플리케이션의 베어 메탈 또는 RHOSP(Red Hat OpenStack Platform) 인프라에 OpenShift Container Platform 클러스터에서 추가 네트워크와 함께 SR-IOV 네트워크 장치를 사용할 수 있습니다.

다음 명령을 사용하여 노드에서 SR-IOV를 활성화할 수 있습니다.

```
$ oc label node <node_name> feature.node.kubernetes.io/network-sriov.capable="true"
```

12.1.1. 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 구성 생성 및 업데이트
- 노드별 **SriovNetworkNodeState** 사용자 정의 리소스 생성
- 각 **SriovNetworkNodeState** 사용자 정의 리소스에서 **spec.interfaces** 필드 업데이트

Operator는 다음 구성 요소를 프로비저닝합니다.

SR-IOV 네트워크 구성 데몬

SR-IOV Operator가 시작될 때 작업자 노드에 배포되는 DaemonSet. 데몬은 클러스터에서 SR-IOV 네트워크 장치를 검색하고 초기화합니다.

SR-IOV Operator 웹 후크

Operator 사용자 정의 리소스의 유효성을 검증하고 설정되지 않은 필드에 적절한 기본값을 설정하는 동적 승인 컨트롤러 webhook.

SR-IOV 네트워크 리소스 인젝터

SR-IOV VF와 같은 사용자 정의 네트워크 리소스에 대한 요청 및 제한으로 Kubernetes pod 사양을 패치하는 기능을 제공하는 동적 승인 컨트롤러 webhook. SR-IOV 네트워크 리소스 인젝터는 리소스 필드를 Pod의 첫 번째 컨테이너에 자동으로 추가합니다.

SR-IOV 네트워크 장치 플러그인

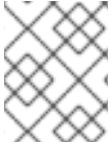
SR-IOV 네트워크 VF(가상 기능) 리소스를 검색, 보급 및 할당하는 장치 플러그인입니다. Kubernetes에서는 장치 플러그인을 사용하여 일반적으로 물리적 장치에서 제한된 리소스를 사용할 수 있습니다. 장치 플러그인은 Kubernetes 스케줄러에 리소스 가용성을 알려 스케줄러가 충분한 리소스가 있는 노드에서 pod를 스케줄링할 수 있도록 합니다.

SR-IOV CNI 플러그인

SR-IOV 장치 플러그인에서 할당된 VF 인터페이스를 pod에 직접 연결하는 CNI 플러그인입니다.

SR-IOV InfiniBand CNI 플러그인

SR-IOV 장치 플러그인에서 할당된 IB(InfiniBand) VF 인터페이스를 pod에 직접 연결하는 CNI 플러그인입니다.



참고

SR-IOV 네트워크 리소스 인젝터 및 SR-IOV 네트워크 Operator webhook는 기본적으로 활성화되어 있으며 기본 **SriovOperatorConfig** CR을 편집하여 비활성화할 수 있습니다.

12.1.1.1. 지원되는 플랫폼

SR-IOV Network Operator는 다음 플랫폼에서 지원됩니다.

- 베어 메탈
- Red Hat OpenStack Platform (RHOSP)

12.1.1.2. 지원되는 장치

OpenShift Container Platform에서는 다음 네트워크 인터페이스 컨트롤러를 지원합니다.

표 12.1. 지원되는 네트워크 인터페이스 컨트롤러

제조업체	모델	벤더 ID	장치 ID
Intel	X710	8086	1572
Intel	XXV710	8086	158b
Mellanox	MT27700 제품군 [ConnectX-4]	15b3	1013
Mellanox	MT27710 제품군 [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 제품군 [ConnectX-5]	15b3	1017
Mellanox	MT28908 제품군 [ConnectX-6]	15b3	101b

12.1.1.3. SR-IOV 네트워크 장치의 자동 검색

SR-IOV Network Operator는 작업자 노드에서 SR-IOV 가능 네트워크 장치를 클러스터에서 검색합니다. Operator는 호환되는 SR-IOV 네트워크 장치를 제공하는 각 작업자 노드에 대해 **SriovNetworkNodeState** CR(사용자 정의 리소스)을 생성하고 업데이트합니다.

CR에는 작업자 노드와 동일한 이름이 할당됩니다. **status.interfaces** 목록은 노드의 네트워크 장치에 대한 정보를 제공합니다.



중요

SriovNetworkNodeState 오브젝트를 수정하지 마십시오. Operator는 이러한 리소스를 자동으로 생성하고 관리합니다.

12.1.1.3.1. 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 Operator가 감지한 모든 SR-IOV 장치 목록이 포함되어 있습니다.

12.1.1.4. Pod에서 가상 함수 사용 예

SR-IOV VF가 연결된 pod에서 RDMA(Remote Direct Memory Access) 또는 DPDK(Data Plane Development Kit) 애플리케이션을 실행할 수 있습니다.

이 예는 RDMA 모드에서 VF(가상 기능)를 사용하는 pod를 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

```

Pod와 관련된 네트워크 정보를 수집할 때 컨테이너에서 실행되는 애플리케이션을 돕기 위해 선택적 라이브러리를 사용할 수 있습니다. 이 라이브러리를 'app-netutil'이라고 합니다. [app-netutil GitHub repo](#)에서 라이브러리의 소스 코드를 참조하십시오.

이 라이브러리는 DPDK 모드의 SR-IOV VF를 컨테이너에 쉽게 통합할 수 있도록 고안되었습니다. 라이브러리는 GO API 및 C API와 두 언어 사용 예를 모두 제공합니다.

pod-spec의 환경 변수 l2fwd, l3wd 또는 testpmd와 같은 DPDK 샘플 애플리케이션 중 하나를 실행할 수 있는 샘플 Docker 이미지 'dpdk-app-centos'도 있습니다. 이 Docker 이미지는 'app-netutil'을 컨테이너 이미지 자체에 통합하는 예를 제공합니다. 라이브러리는 원하는 데이터를 수집하고 기존 DPDK 워크로드로 데이터를 전달하는 초기화 컨테이너에 통합할 수도 있습니다.

12.1.2. 다음 단계

- [SR-IOV Network Operator 설치](#)
- 선택 사항: [SR-IOV Network Operator 구성](#)
- [SR-IOV 네트워크 장치 구성](#)
- OpenShift Virtualization을 사용하는 경우: [가상 머신의 SR-IOV 네트워크 장치 구성](#)
- [SR-IOV 네트워크 연결 구성](#)
- [SR-IOV 추가 네트워크에 pod 추가](#)

12.2. SR-IOV NETWORK OPERATOR 설치

SR-IOV(Single Root I/O Virtualization) Network Operator를 클러스터에 설치하여 SR-IOV 네트워크 장치 및 네트워크 연결을 관리할 수 있습니다.

12.2.1. SR-IOV Network Operator 설치

클러스터 관리자는 OpenShift Container Platform CLI 또는 웹 콘솔을 사용하여 SR-IOV Network Operator를 설치할 수 있습니다.

12.2.1.1. CLI: SR-IOV Network Operator 설치

클러스터 관리자는 CLI를 사용하여 Operator를 설치할 수 있습니다.

사전 요구 사항

- SR-IOV를 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
EOF
```

2. OperatorGroup CR을 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 Operator를 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 Operator에 대한 서브스크립션 CR을 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.4.4.0-202006160135 Succeeded
    
```

12.2.1.2. 웹 콘솔: SR-IOV Network Operator 설치

클러스터 관리자는 웹 콘솔을 사용하여 Operator를 설치할 수 있습니다.



참고

CLI를 사용하여 Operator group을 생성해야 합니다.

사전 요구 사항

- SR-IOV를 지원하는 하드웨어가 있는 노드로 베어 메탈 하드웨어에 설치된 클러스터.
- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 계정.

프로세스

1. SR-IOV Network Operator의 네임스페이스를 만듭니다.
 - a. OpenShift Container Platform 웹 콘솔에서 **관리** → **네임스페이스**를 클릭합니다.
 - b. **네임스페이스 생성**을 클릭합니다.
 - c. 이름 필드에 **openshift-sriov-network-operator**를 입력한 후 **생성**을 클릭합니다.
2. 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 **설치**를 클릭합니다.
 - c. **Operator** 설치 페이지의 클러스터의 특정 네임스페이스에서 **openshift-sriov-network-operator**를 선택합니다.

- d. 설치를 클릭합니다.
3. SR-IOV Network Operator가 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 Operator는 **실패** 상태를 표시할 수 있습니다. 나중에 **InstallSucceeded** 메시지와 함께 설치에 성공하면 이 **실패** 메시지를 무시할 수 있습니다.

Operator가 설치된 것으로 나타나지 않으면 다음과 같이 추가 트러블슈팅을 수행하십시오.

- **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.

12.2.2. 다음 단계

- 선택 사항: [SR-IOV Network Operator 구성](#)

12.3. SR-IOV NETWORK OPERATOR 구성

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

12.3.1. 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 만듭니다.



참고

default CR에는 클러스터에 대한 SR-IOV Network Operator 구성이 포함됩니다. Operator 구성을 변경하려면 이 CR을 수정해야 합니다.

SriovOperatorConfig 오브젝트는 Operator 구성을 위한 몇 가지 필드를 제공합니다.

- **enableInjector**를 사용하면 프로젝트 관리자가 Network Resources Injector 데몬 세트를 활성화하거나 비활성화할 수 있습니다.

- **enableOperatorWebhook**을 사용하면 프로젝트 관리자가 Operator Admission Controller 웹 후크 데몬 세트를 활성화하거나 비활성화할 수 있습니다.
- **configDaemonNodeSelector**를 사용하면 프로젝트 관리자가 선택된 노드에서 SR-IOV Network Config Daemon을 예약할 수 있습니다.

12.3.1.1. Network Resources Injector 정보

Network Resources Injector는 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
- 하향식 API 볼륨으로 **Pod** 사양을 변경하여 Pod 주석 및 레이블을 실행 중인 컨테이너에 **/etc/podnetinfo** 경로의 파일로 노출합니다.

기본적으로 Network Resources Injector는 SR-IOV Operator에 의해 활성화되며 모든 컨트롤 플레인 노드(마스터 노드라고도 함)에서 데몬 세트로 실행됩니다. 다음은 3개의 컨트롤 플레인 노드가 있는 클러스터에서 실행 중인 Network Resources Injector Pod의 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

12.3.1.2. SR-IOV Operator Admission Controller webhook 정보

SR-IOV Operator Admission Controller webhook은 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 Operator Admission Controller webhook은 Operator에서 활성화하며 모든 컨트롤 플레인 노드에서 데몬 세트로 실행됩니다. 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

12.3.1.3. 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.

12.3.1.4. Network Resources Injector 비활성화 또는 활성화

기본적으로 활성화되어 있는 Network Resources Injector를 비활성화하거나 활성화하려면 다음 절차를 완료하십시오.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- SR-IOV Operator가 설치되어 있어야 합니다.

프로세스

- **enableInjector** 필드를 설정합니다. 기능을 비활성화하려면 **<value>**를 **false**로 바꾸고 기능을 활성화하려면 **true**로 바꿉니다.

```
$ oc patch sriovoperatorconfig default \
  --type=merge -n openshift-sriov-network-operator \
  --patch '{"spec": {"enableInjector": <value> } }'
```

12.3.1.5. SR-IOV 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 Operator가 설치되어 있어야 합니다.

프로세스

- **enableOperatorWebhook** 필드를 설정합니다. 기능을 비활성화하려면 **<value>**를 **false**로 바꾸고 활성화하려면 **true**로 바꿉니다.

```
$ oc patch sriovoperatorconfig default --type=merge \
  -n openshift-sriov-network-operator \
  --patch '{"spec": {"enableOperatorWebhook": <value> } }'
```

12.3.1.6. 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 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 Pod를 만들 수 없습니다.

프로세스

- Operator의 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.

12.3.2. 다음 단계

- [SR-IOV 네트워크 장치 구성](#)

12.4. SR-IOV 네트워크 장치 구성

클러스터에서 SR-IOV(Single Root I/O Virtualization) 장치를 구성할 수 있습니다.

12.4.1. SR-IOV 네트워크 노드 구성 오브젝트

SriovNetworkNodePolicy 오브젝트를 정의하여 노드의 SR-IOV 네트워크 장치 구성을 지정합니다. 오브젝트는 **sriovnetwork.openshift.io** API 그룹의 일부입니다.

다음 YAML은 **SriovNetworkNodePolicy** 오브젝트를 설명합니다.

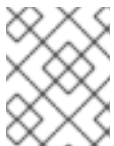
```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> ①
  namespace: openshift-sriov-network-operator ②
spec:
  resourceName: <sriov_resource_name> ③
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" ④
  priority: <priority> ⑤
  mtu: <mtu> ⑥
```

```

numVfs: <num> 7
nicSelector: 8
  vendor: "<vendor_code>" 9
  deviceID: "<device_id>" 10
  pfNames: ["<pf_name>", ...] 11
  rootDevices: ["<pci_bus_id>", "..."] 12
deviceType: <device_type> 13
isRdma: false 14
linkType: <link_type> 15

```

- 1 CR 오브젝트의 이름입니다.
- 2 SR-IOV Operator가 설치된 네임스페이스입니다.
- 3 SR-IOV 장치 플러그인의 리소스 이름입니다. 리소스 이름에 대해 여러 **SriovNetworkNodePolicy** 오브젝트를 생성할 수 있습니다.
- 4 구성할 노드를 선택하는 노드 선택기입니다. 선택한 노드의 SR-IOV 네트워크 장치만 구성됩니다. SR-IOV CNI(Container Network Interface) 플러그인 및 장치 플러그인은 선택한 노드에만 배포됩니다.
- 5 선택 사항: 0 에서 99 사이의 정수 값입니다. 숫자가 작을수록 우선 순위가 높아지므로 우선 순위 10은 우선 순위 99보다 높습니다. 기본값은 99입니다.
- 6 선택 사항: 가상 기능의 최대 전송 단위(MTU). 최대 MTU 값은 NIC 모델마다 다를 수 있습니다.
- 7 SR-IOV 물리적 네트워크 장치에 생성할 VF(가상 기능) 수입니다. Intel NIC(Network Interface Card)의 경우 VF 수는 장치에서 지원하는 총 VF보다 클 수 없습니다. Mellanox NIC의 경우 VF 수는 128보다 클 수 없습니다.
- 8 **nicSelector** 매핑은 Operator가 구성할 장치를 선택합니다. 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 가리키는지 확인하십시오.
- 9 선택 사항: SR-IOV 네트워크 장치의 공급업체 16진수 코드입니다. 허용되는 값은 **8086** 및 **15b3**입니다.
- 10 선택 사항: SR-IOV 네트워크 장치의 장치 16진수 코드입니다. 허용되는 값은 **158b**, **1015**, **1017**입니다.
- 11 선택 사항: 장치에 대한 하나 이상의 물리적 기능(PF) 이름으로 이루어진 배열입니다.
- 12 장치의 PF에 대한 하나 이상의 PCI 버스 주소로 구성된 배열입니다. 다음 형식으로 주소를 입력합니다. **0000:02:00.1**.
- 13 선택 사항: 가상 기능의 드라이버 유형입니다. 허용되는 값은 **netdevice** 및 **vfio-pci**입니다. 기본값은 **netdevice**입니다.



참고

베어 메탈 노드의 DPDK(Data Plane Development Kit) 모드에서 Mellanox 카드를 작동시키려면 **netdevice** 드라이버 유형을 사용하고 **isRdma**를 **true**로 설정합니다.

- 14 선택 사항: 원격 직접 메모리 액세스(RDMA) 모드 사용 여부. 기본값은 **false**입니다.



참고

isRDMA 매개변수가 **true**로 설정된 경우 RDMA 가능 VF를 일반 네트워크 장치로 계속 사용할 수 있습니다. 어느 모드에서나 장치를 사용할 수 있습니다.

- 15** 선택 사항: VF의 링크 유형입니다. **eth** 또는 **ib** 값 중 하나를 지정할 수 있습니다. **eth** 는 이더넷이고 **ib** 는 InfiniBand입니다. 명시적으로 설정되지 않은 경우 기본값은 **eth**입니다. **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.

12.4.1.1. SR-IOV 네트워크 노드 구성 예

다음 예제에서는 IB 장치의 구성을 설명합니다.

IB 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
```

12.4.1.2. 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 **2**에서 **7**까지의 **netpf0** 인터페이스에 대한 선택기를 보여줍니다.

```
pfNames: ["netpf0#2-7"]
```

- **netpf0**은 PF 인터페이스 이름입니다.
- **2**는 범위에 포함된 첫 번째 VF 인덱스(0 기반)입니다.
- **7**은 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>-1까지의 범위 내에 있어야 합니다. 예를 들어, numVfs가 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 **0**을 포함하는 리소스 풀 **net-1**을 정의합니다. 정책 **policy-net-1-dpdk**는 **vfio** VF 드라이버와 함께 PF **netpf0**의 VF **8 ~ 15**를 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: net1 dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#8-15"]
  deviceType: vfio-pci
```

12.4.2. SR-IOV 네트워크 장치 구성

SR-IOV Network Operator는 **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition을 OpenShift Container Platform에 추가합니다. SriovNetworkNodePolicy CR(사용자 정의 리소스)을 만들어 SR-IOV 네트워크 장치를 구성할 수 있습니다.



참고

SriovNetworkNodePolicy 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 Operator가 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 YAML을 **<name>-sriov-node-network.yaml** 파일에 저장합니다. **<name>**을 이 구성의 이름으로 바꿉니다.
2. 선택 사항: **SriovNetworkNodePolicy.Spec.NodeSelector** 로 SR-IOV 가능 클러스터 노드에 레이블을 지정하지 않은 경우 레이블을 지정합니다. 노드 레이블링에 대한 자세한 내용은 "노드에서 라벨을 업데이트하는 방법"을 참조하십시오.
2. **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 Pod가 **Running** 상태로 전환됩니다.

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}'
```

추가 리소스

- [노드에서 라벨을 업데이트하는 방법 이해](#)

12.4.3. 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 없습니다

```
"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.

12.4.4. 다음 단계

- [SR-IOV 네트워크 연결 구성](#)

12.5. SR-IOV 이더넷 네트워크 연결 구성

클러스터에서 SR-IOV(Single Root I/O Virtualization) 장치에 대한 이더넷 네트워크 연결을 구성할 수 있습니다.

12.5.1. 이더넷 장치 구성 오브젝트

SriovNetwork 오브젝트를 정의하여 이더넷 네트워크 장치를 구성할 수 있습니다.

다음 YAML은 **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 Operator는 동일한 이름으로 **NetworkAttachmentDefinition** 오브젝트를 생성합니다.

2 SR-IOV Network Operator가 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 선택 사항: 추가 네트워크의 VLAN(Virtual LAN) ID입니다. 정수 값은 **0**에서 **4095** 사이여야 합니다. 기본값은 **0**입니다.
- 6 선택 사항: VF의 스푸핑 검사 모드입니다. 허용되는 값은 문자열 **"on"** 및 **"off"**입니다.



중요

SR-IOV Network Operator가 지정한 값을 따옴표로 묶거나 오브젝트를 거부해야 합니다.

- 7 YAML 블록 스칼라인 IPAM CNI 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.
- 8 선택 사항: VF(가상 기능)의 링크 상태입니다. 허용되는 값은 **enable**, **disable** 및 **auto**입니다.
- 9 선택 사항: VF의 경우 최대 전송 속도(Mbps).
- 10 선택 사항: VF의 경우 최소 전송 속도(Mbps). 이 값은 최대 전송 속도보다 작거나 같아야 합니다.



참고

인텔 NIC는 **minTxRate** 매개변수를 지원하지 않습니다. 자세한 내용은 [BZ#1772847](#)에서 참조하십시오.

- 11 선택 사항: VF의 IEEE 802.1p 우선 순위 수준입니다. 기본값은 **0**입니다.
- 12 선택 사항: VF의 신뢰 모드입니다. 허용되는 값은 문자열 **"on"** 및 **"off"**입니다.



중요

지정한 값을 따옴표로 묶어야 합니다. 그렇지 않으면 SR-IOV Network Operator에서 오브젝트를 거부합니다.

- 13 선택 사항: 이 추가 네트워크에 대해 구성할 수 있는 기능입니다. **"ips": true** 를 지정하여 IP 주소 지원을 활성화하거나 **"mac": true** 를 지정하여 MAC 주소 지원을 활성화할 수 있습니다.

12.5.1.1. 추가 네트워크에 대한 IP 주소 할당 구성

IPAM(IP 주소 관리) CNI(Container Network Interface) 플러그인은 다른 CNI 플러그인의 IP 주소를 제공 합니다.

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

12.5.1.1.1. 고정 IP 주소 할당 구성

다음 표에서는 고정 IP 주소 할당 구성에 대해 설명합니다.

표 12.2. IPAM 정적 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 static 이 필요합니다.
주소	array	가상 인터페이스에 할당할 IP 주소를 지정하는 개체 배열입니다. IPv4 및 IPv6 IP 주소가 모두 지원됩니다.
routes	array	Pod 내부에서 구성할 경로를 지정하는 오브젝트 배열입니다.
DNS	array	선택 사항: DNS 구성을 지정하는 개체 배열입니다.

addresses 배열에는 다음 필드가 있는 오브젝트가 필요합니다.

표 12.3. `ipam.addresses[]` 배열

필드	유형	설명
address	string	지정하는 IP 주소 및 네트워크 접두사입니다. 예를 들어 10.10.21.10/24 를 지정하면 추가 네트워크에 IP 주소 10.10.21.10 이 할당되고 넷마스크는 255.255.255.0 입니다.
gateway	string	송신 네트워크 트래픽을 라우팅할 기본 게이트웨이입니다.

표 12.4. `ipam.routes[]` 배열

필드	유형	설명
dst	string	CIDR 형식의 IP 주소 범위(예: 기본 경로의 경우 192.168.17.0/24 또는 0.0.0.0/0)입니다.
gw	string	네트워크 트래픽이 라우팅되는 게이트웨이입니다.

표 12.5. `ipam.dns` 오브젝트

필드	유형	설명
nameservers	array	DNS 쿼리를 보낼 하나 이상의 IP 주소 배열입니다.

필드	유형	설명
domain	array	호스트 이름에 추가할 기본 도메인입니다. 예를 들어 도메인이 example.com 으로 설정되면 example-host 에 대한 DNS 조회 쿼리가 example-host.example.com 으로 다시 작성됩니다.
search	array	DNS 조회 쿼리 중에 규정되지 않은 호스트 이름(예: example-host)에 추가할 도메인 이름 배열입니다.

고정 IP 주소 할당 구성 예

```
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}
```

12.5.1.1.2. DHCP(동적 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.

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 12.6. IPAM DHCP 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 dhcp 가 필요합니다.

DHCP(동적 IP 주소) 할당 구성 예

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

12.5.1.1.3. 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 Whereabouts를 사용한 동적 IP 주소 할당 구성을 설명합니다.

표 12.7. IPAM whereabouts 구성 개체

필드	유형	설명
type	string	IPAM 주소 유형입니다. whereabouts 가 필요한 값입니다.
범위	string	CIDR 표기법의 IP 주소 및 범위입니다. IP 주소는 이 주소 범위 내에서 할당됩니다.
제외	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"
    ]
  }
}
```

12.5.2. SR-IOV 추가 네트워크 구성

SriovNetwork 오브젝트를 생성하여 SR-IOV 하드웨어를 사용하는 추가 네트워크를 구성할 수 있습니다. **SriovNetwork** 오브젝트를 생성하면 SR-IOV Operator에서 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성합니다.



참고

SriovNetwork 오브젝트가 **running** 상태의 pod에 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 YAML을 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"
    }
}
```

- 오브젝트를 생성하려면 다음 명령을 입력합니다:

```
$ oc create -f <name>.yaml
```

여기서 **<name>**은 추가 네트워크의 이름을 지정합니다.

- 선택 사항: 이전 단계에서 생성한 **SriovNetwork** 오브젝트와 연결된 **NetworkAttachmentDefinition** 오브젝트가 존재하는지 확인하려면 다음 명령을 입력합니다. **<namespace>**를 **SriovNetwork** 오브젝트에 지정한 **networkNamespace**로 바꿉니다.

```
$ oc get net-attach-def -n <namespace>
```

12.5.3. 다음 단계

- [SR-IOV 추가 네트워크에 pod 추가](#)

12.5.4. 추가 리소스

- [SR-IOV 네트워크 장치 구성](#)

12.6. SR-IOV INFINIBAND 네트워크 연결 구성

클러스터에서 SR-IOV(Single Root I/O Virtualization) 장치에 대한 IB(InfiniBand) 네트워크 연결을 구성할 수 있습니다.

12.6.1. InfiniBand 장치 구성 오브젝트

SriovIBNetwork 오브젝트를 정의하여 IB(InfiniBand) 네트워크 장치를 구성할 수 있습니다.

다음 YAML은 **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 Operator는 동일한 이름으로 **NetworkAttachmentDefinition** 오브젝트를 생성합니다.
- 2 SR-IOV Operator가 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 선택 사항: YAML 블록 스칼라인 IPAM CNI 플러그인에 대한 구성 오브젝트입니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.
- 6 선택 사항: VF(가상 기능)의 링크 상태입니다. 허용되는 값은 **enable, disable, auto**입니다.
- 7 선택 사항: 이 네트워크에 구성할 수 있는 기능. IP 주소 지원을 사용하려면 "{ **"ips": true }**"를 지정하고 IB GUID(Global Unique Identifier) 지원을 사용하려면 "{ **"infinibandGUID": true }**"를 지정하면 됩니다.

12.6.1.1. 추가 네트워크에 대한 IP 주소 할당 구성

IPAM(IP 주소 관리) CNI(Container Network Interface) 플러그인은 다른 CNI 플러그인의 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

12.6.1.1.1. 고정 IP 주소 할당 구성

다음 표에서는 고정 IP 주소 할당 구성에 대해 설명합니다.

표 12.8. IPAM 정적 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 static 이 필요합니다.
주소	array	가상 인터페이스에 할당할 IP 주소를 지정하는 개체 배열입니다. IPv4 및 IPv6 IP 주소가 모두 지원됩니다.
routes	array	Pod 내부에서 구성할 경로를 지정하는 오브젝트 배열입니다.
DNS	array	선택 사항: DNS 구성을 지정하는 개체 배열입니다.

addresses 배열에는 다음 필드가 있는 오브젝트가 필요합니다.

표 12.9. **ipam.addresses[]** 배열

필드	유형	설명
address	string	지정하는 IP 주소 및 네트워크 접두사입니다. 예를 들어 10.10.21.10/24 를 지정하면 추가 네트워크에 IP 주소 10.10.21.10 이 할당되고 넷마스크는 255.255.255.0 입니다.
gateway	string	송신 네트워크 트래픽을 라우팅할 기본 게이트웨이입니다.

표 12.10. **ipam.routes[]** 배열

필드	유형	설명
dst	string	CIDR 형식의 IP 주소 범위(예: 기본 경로의 경우 192.168.17.0/24 또는 0.0.0.0/0)입니다.
gw	string	네트워크 트래픽이 라우팅되는 게이트웨이입니다.

표 12.11. **ipam.dns** 오브젝트

필드	유형	설명
nameservers	array	DNS 쿼리를 보낼 하나 이상의 IP 주소 배열입니다.
domain	array	호스트 이름에 추가할 기본 도메인입니다. 예를 들어 도메인이 example.com 으로 설정되면 example-host 에 대한 DNS 조회 쿼리가 example-host.example.com 으로 다시 작성됩니다.
search	array	DNS 조회 쿼리 중에 규정되지 않은 호스트 이름(예: example-host)에 추가할 도메인 이름 배열입니다.

고정 IP 주소 할당 구성 예

```
{
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.7/24"
      }
    ]
  }
}
```

12.6.1.1.2. DHCP(동적 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 12.12. IPAM DHCP 구성 오브젝트

필드	유형	설명
type	string	IPAM 주소 유형입니다. 값 dhcp 가 필요합니다.

DHCP(동적 IP 주소) 할당 구성 예

```
{
  "ipam": {
    "type": "dhcp"
  }
}
```

12.6.1.1.3. 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 Whereabouts를 사용한 동적 IP 주소 할당 구성을 설명합니다.

표 12.13. IPAM whereabouts 구성 개체

필드	유형	설명
type	string	IPAM 주소 유형입니다. whereabouts 가 필요한 값입니다.
범위	string	CIDR 표기법의 IP 주소 및 범위입니다. IP 주소는 이 주소 범위 내에서 할당됩니다.
제외	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"
    ]
  }
}
```

12.6.2. SR-IOV 추가 네트워크 구성

SriovIBNetwork 오브젝트를 생성하여 SR-IOV 하드웨어를 사용하는 추가 네트워크를 구성할 수 있습니다. **SriovIBNetwork** 오브젝트를 생성하면 SR-IOV Operator에서 **NetworkAttachmentDefinition** 오브젝트를 자동으로 생성합니다.



참고

SriovIBNetwork 오브젝트가 **running** 상태의 pod에 연결된 경우 이 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 YAML을 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** 오브젝트와 연결된 **NetworkAttachmentDefinition** 오브젝트가 존재하는지 확인하려면 다음 명령을 입력합니다. **<namespace>**를 **SriovIBNetwork** 오브젝트에 지정한 **networkNamespace**로 바꿉니다.

```
$ oc get net-attach-def -n <namespace>
```

12.6.3. 다음 단계

- [SR-IOV 추가 네트워크에 pod 추가](#)

12.6.4. 추가 리소스

- [SR-IOV 네트워크 장치 구성](#)

12.7. SR-IOV 추가 네트워크에 POD 추가

기존 SR-IOV(Single Root I/O Virtualization) 네트워크에 pod를 추가할 수 있습니다.

12.7.1. 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.

12.7.1.1. 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 선택 사항: SR-IOV 네트워크 연결 정의 CR에 정의된 리소스 유형에서 할당된 SR-IOV 장치의 MAC 주소입니다. 이 기능을 사용하려면 **SriovNetwork** 오브젝트에 { **"mac": true** }도 지정해야 합니다.
- 3 선택 사항: SR-IOV 네트워크 연결 정의 CR에 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.

런타임 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"]
```

12.7.1.2. 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.

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"]

```

12.7.2. 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 Injector는 리소스 필드를 포드의 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 구성됩니다.

SriovNetworkNodePolicy 오브젝트에서 **deviceType** 이 **vfio-pci**로 설정된 경우 SR-IOV 추가 네트워크가 DPDK 모드에 대해 구성됩니다.

NIC에 액세스해야 하는 컨테이너가 **Pod** 오브젝트에 정의된 첫 번째 컨테이너인지 또는 Network Resource Injector를 비활성화하여 이 문제를 해결할 수 있습니다. 자세한 내용은 [BZ#1990953](#) 에서 참조하십시오.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- 클러스터에 로그인합니다.
- SR-IOV Operator를 설치합니다.
- Pod를 연결할 **SriovNetwork** 오브젝트 또는 **SriovIBNetwork** 오브젝트를 생성합니다.

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. 선택 사항: **Pod** CR에 주석이 있는지 확인하려면 다음 명령을 입력하고 **<name>**을 Pod 이름으로 바꿉니다.

```
$ 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:
  ...

```

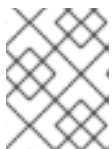
1 **k8s.v1.cni.cncf.io/networks-status** 매개변수는 JSON 오브젝트 배열입니다. 각 오브젝트는 Pod에 연결된 추가 네트워크의 상태를 설명합니다. 주석 값은 일반 텍스트 값으로 저장됩니다.

12.7.3. 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 Pod를 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-node가 요청을 충족할 수 없는 경우 Topology Manager 정책을 **restricted**로 구성할 수 있습니다.

절차

1. 다음과 같은 SR-IOV Pod 사양을 생성한 다음 YAML을 **<name>-sriov-pod.yaml** 파일에 저장합니다. **<name>**을 이 Pod의 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> ❶
spec:
  containers:
  - name: sample-container
    image: <image> ❷
    command: ["sleep", "infinity"]
    resources:
      limits:
        memory: "1Gi" ❸
        cpu: "2" ❹
      requests:
        memory: "1Gi"
        cpu: "2"

```

- ❶ <name>을 SR-IOV 네트워크 첨부 파일 정의 CR의 이름으로 바꿉니다.
- ❷ <image>를 **sample-pod** 이미지의 이름으로 바꿉니다.
- ❸ 보장된 QoS로 SR-IOV Pod를 생성하려면 **메모리 제한을 메모리 요청**과 동일하게 설정합니다.
- ❹ 보장된 QoS로 SR-IOV Pod를 생성하려면 **cpu 제한을 CPU 요청**과 동일하게 설정합니다.

2. 다음 명령을 실행하여 샘플 SR-IOV Pod를 만듭니다.

```
$ oc create -f <filename> ❶
```

- ❶ <filename>을 이전 단계에서 생성한 파일 이름으로 바꿉니다.

3. **sample-pod**가 보장된 QoS로 구성되어 있는지 확인하십시오.

```
$ oc describe pod sample-pod
```

4. **sample-pod**에 전용 CPU가 할당되어 있는지 확인하십시오.

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

5. **sample-pod**에 할당된 SR-IOV 장치 및 CPU가 동일한 NUMA 노드에 있는지 확인하십시오.

```
$ oc exec sample-pod -- cat /sys/fs/cgroup/cpuset/cpuset.cpus
```

12.7.4. 추가 리소스

- [SR-IOV 이더넷 네트워크 연결 구성](#)
- [SR-IOV InfiniBand 네트워크 연결 구성](#)
- [CPU 관리자 사용](#)

12.8. 고성능 멀티 캐스트 사용

SR-IOV(Single Root I/O Virtualization) 하드웨어 네트워킹에서 멀티 캐스트를 사용할 수 있습니다.

12.8.1. 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 pod에서 보내거나 받아야 합니다.
- SR-IOV 인터페이스를 연결하는 물리적 네트워크는 멀티 캐스트 라우팅 및 토폴로지를 결정하며 OpenShift Container Platform에서 제어하지 않습니다.

12.8.2. 멀티 캐스트에 대한 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 선택하는 경우 DHCP 서버를 통해 다음 기본 경로를 프로비저닝해야 합니다. **224.0.0.0/5** 및 **232.0.0.0/5**. 이는 기본 네트워크 공급자가 설정한 정적 멀티캐스트 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.

12.9. DPDK 및 RDMA 모드와 함께 VF(가상 기능) 사용

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.

12.9.1. DPDK 및 RDMA 모드에서 가상 기능을 사용하는 예

중요

DPDK(Data Plane Development Kit)는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수도 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 <https://access.redhat.com/support/offerings/techpreview/>를 참조하십시오.

중요

RDMA(Remote Direct Memory Access)는 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수도 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 <https://access.redhat.com/support/offerings/techpreview/>를 참조하십시오.

12.9.2. 사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 권한이 있는 사용자로 로그인합니다.
- SR-IOV Network Operator가 설치되어 있어야 합니다.

12.9.3. Intel NIC와 함께 DPDK 모드에서 VF(가상 기능)를 사용하는 예**절차**

1. 다음 **SriovNetworkNodePolicy** 오브젝트를 생성한 다음 YAML을 **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

SriovNetworkNodePolicy의 각 옵션에 대한 자세한 설명은 **SR-IOV 네트워크 장치 구성** 섹션을 참조하십시오.

SriovNetworkNodePolicy 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. 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 YAML을 **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 플러그인에 빈 오브젝트 "{}"를 지정합니다. DPDK는 사용자 공간 모드에서 작동하며 IP 주소가 필요하지 않습니다.



참고

SriovNetwork의 각 옵션에 대한 자세한 설명은 "SR-IOV 추가 네트워크 구성" 섹션을 참조하십시오.

4. 다음 명령을 실행하여 **SriovNetwork** 오브젝트를 생성합니다.

```
$ oc create -f intel-dpdk-network.yaml
```

5. 다음 **Pod** 사양을 생성한 다음 YAML을 **intel-dpdk-pod.yaml** 파일에 저장합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
```

```

namespace: <target_namespace> ❶
annotations:
  k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> ❷
    securityContext:
      runAsUser: 0
    capabilities:
      add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"] ❸
    volumeMounts:
    - mountPath: /dev/hugepages ❹
      name: hugepage
    resources:
      limits:
        openshift.io/intelNics: "1" ❺
        memory: "1Gi"
        cpu: "4" ❻
        hugepages-1Gi: "4Gi" ❼
      requests:
        openshift.io/intelNics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
      command: ["sleep", "infinity"]
    volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages

```

- ❶ **SriovNetwork** 오브젝트 **intel-dpdk-network**가 생성되는 동일한 **target_namespace**를 지정합니다. 다른 네임스페이스에서 포드를 생성하려면 **Pod** 사양과 **SriovNetwork** 오브젝트 모두에서 **target_namespace**를 변경합니다.
- ❷ 애플리케이션 및 애플리케이션이 사용하는 DPDK 라이브러리를 포함하는 DPDK 이미지를 지정합니다.
- ❸ hugepage 할당, 시스템 리소스 할당 및 네트워크 인터페이스 액세스를 위해 컨테이너 내부의 애플리케이션에 필요한 추가 기능을 지정합니다.
- ❹ **/dev/hugepages** 아래 DPDK pod에 hugepage 볼륨을 마운트합니다. hugepage 볼륨은 매체가 **HugePages**인 emptyDir 볼륨 유형으로 지원됩니다.
- ❺ 선택 사항: DPDK Pod에 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 Operator에서 관리하는 승인 컨트롤러 구성 요소입니다. 기본적으로 활성화되어 있으며 기본 **SriovOperatorConfig** CR에서 **enableInjector** 옵션을 **false**로 설정하여 비활성화할 수 있습니다.
- ❻ CPU 수를 지정합니다. DPDK pod는 일반적으로 kubelet에서 배타적 CPU를 할당해야 합니다. 이를 위해 CPU 관리자 정책을 **static**으로 설정하고 QoS가 **보장된** Pod를 생성합니다.
- ❼

hugepage 크기 **hugepages-1Gi** 또는 **hugepages-2Mi**를 지정하고 DPDK Pod에 할당할 hugepage 수량을 지정합니다. **2Mi** 및 **1Gi** hugepage를 별도로 구성합니다. **1Gi** hugepage를

- 다음 명령을 실행하여 DPDK Pod를 생성합니다.

```
$ oc create -f intel-dpdk-pod.yaml
```

12.9.4. Mellanox NIC와 함께 DPDK 모드에서 가상 기능을 사용하는 예

절차

- 다음 **SriovNetworkNodePolicy** 오브젝트를 생성한 다음 YAML을 **mlx-dpdk-node-policy.yaml** 파일에 저장합니다.

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
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. Mellanox 카드에 허용되는 유일한 값은 **1015**, **1017**입니다.
- ② **netdevice**에 가상 기능의 드라이버 유형을 지정합니다. Mellanox SR-IOV VF는 **vfio-pci** 장치 유형을 사용하지 않고도 DPDK 모드에서 작동할 수 있습니다. VF 장치는 컨테이너 내부에 커널 네트워크 인터페이스로 나타납니다.
- ③ RDMA 모드를 활성화합니다. 이는 DPDK 모드에서 작동하기 위해 Mellanox 카드에 필요합니다.



참고

SriovNetworkNodePolicy의 각 옵션에 대한 자세한 설명은 **Configuring SR-IOV network devices** 섹션을 참조하십시오.

SriovNetworkNodePolicy 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. 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.

- 다음 명령을 실행하여 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f mlx-dpdk-node-policy.yaml
```

- 다음 **SriovNetwork** 오브젝트를 생성한 다음 YAML을 **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: |- ❶
    ...
  vlan: <vlan>
  resourceName: mlxnic
```

- ❶ ipam CNI 플러그인의 구성 오브젝트를 YAML 블록 스칼라로 지정합니다. 플러그인은 연결 정의에 대한 IP 주소 할당을 관리합니다.



참고

SriovNetwork의 각 옵션에 대한 자세한 설명은 "SR-IOV 추가 네트워크 구성" 섹션을 참조하십시오.

- 다음 명령을 실행하여 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f mlx-dpdk-network.yaml
```

- 다음 **Pod** 사양을 생성한 다음 YAML을 **mlx-dpdk-pod.yaml** 파일에 저장합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  namespace: <target_namespace> ❶
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/mlxnlcs: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/mlxnlcs: "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 포드를 생성하려면 **Pod** 사양과 **SriovNetwork** 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 hugepage 볼륨을 **/dev/hugepages** 아래의 DPDK Pod에 마운트합니다. hugepage 볼륨은 매체가 **Hugepages**인 emptyDir 볼륨 유형으로 지원됩니다.
- 5 선택 사항: DPDK Pod에 할당된 DPDK 장치 수를 지정합니다. SR-IOV 네트워크 리소스 인젝터에서 명시적으로 지정하지 않은 경우 이 리소스 요청 및 제한이 자동으로 추가됩니다. SR-IOV 네트워크 리소스 인젝터는 SR-IOV Operator에서 관리하는 승인 컨트롤러 구성 요소입니다. 기본적으로 활성화되어 있으며 기본 **SriovOperatorConfig** CR에서 **enableInjector** 옵션을 **false**로 설정하여 비활성화할 수 있습니다.
- 6 CPU 수를 지정합니다. DPDK Pod에서는 일반적으로 kubelet에서 전용 CPU를 할당해야 합니다. 이를 위해 CPU 관리자 정책을 **static**으로 설정하고 QoS가 **Guaranteed** Pod를 생성합니다.
- 7 hugepage 크기 **hugepages-1Gi** 또는 **hugepages-2Mi**를 지정하고 DPDK Pod에 할당할 hugepage 수량을 지정합니다. **2Mi** 및 **1Gi** hugepage를 별도로 구성합니다. **1Gi** hugepage를 구성하려면 커널 인수를 노드에 추가해야 합니다.

6. 다음 명령을 실행하여 DPDK Pod를 생성합니다.

```
$ oc create -f mlx-dpdk-pod.yaml
```

12.9.5. Mellanox NIC와 함께 RDMA 모드에서 가상 기능을 사용하는 예

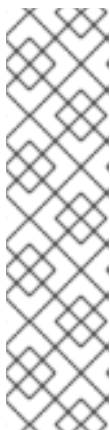
OpenShift Container Platform에서 RDMA를 사용할 때 RoCE(RDMA over Converged Ethernet)가 지원되는 유일한 모드입니다.

절차

1. 다음 **SriovNetworkNodePolicy** 오브젝트를 생성한 다음 YAML을 **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 16진수 코드를 지정합니다. Mellanox 카드에 허용되는 유일한 값은 **1015, 1017**입니다.
- 2 **netdevice**에 가상 기능의 드라이버 유형을 지정합니다.
- 3 RDMA 모드를 활성화합니다.



참고

SriovNetworkNodePolicy의 각 옵션에 대한 자세한 설명은 **SR-IOV 네트워크 장치 구성** 섹션을 참조하십시오.

SriovNetworkNodePolicy 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. 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 YAML을 **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

SriovNetwork의 각 옵션에 대한 자세한 설명은 "SR-IOV 추가 네트워크 구성" 섹션을 참조하십시오.

4. 다음 명령을 실행하여 **SriovNetworkNodePolicy** 오브젝트를 생성합니다.

```
$ oc create -f mlx-rdma-network.yaml
```

5. 다음 **Pod** 사양을 생성한 다음 YAML을 **mlx-rdma-pod.yaml** 파일에 저장합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  namespace: <target_namespace> ❶
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
    - name: testpmd
      image: <RDMA_image> ❷
      securityContext:
        runAsUser: 0
        capabilities:
          add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"] ❸
      volumeMounts:
        - mountPath: /dev/hugepages ❹
          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 라이브러리를 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 Pod에 마운트합니다. hugepage 볼륨은 매체가 **Hugepages**인 emptyDir 볼륨 유형으로 지원됩니다.
- 5** CPU 수를 지정합니다. RDMA Pod는 일반적으로 kubelet에서 전용 CPU를 할당해야 합니다. 이를 위해 CPU 관리자 정책을 **static**으로 설정하고 QoS가 **Guaranteed** Pod를 생성합니다.
- 6** hugepage 크기 **hugepages-1Gi** 또는 **hugepages-2Mi**를 지정하고 RDMA Pod에 할당할 hugepage 수량을 지정합니다. **2Mi** 및 **1Gi** hugepage를 별도로 구성합니다. **1Gi** hugepage를 구성하려면 커널 인수를 노드에 추가해야 합니다.

6. 다음 명령을 실행하여 RDMA Pod를 생성합니다.

```
$ oc create -f mlx-rdma-pod.yaml
```

12.10. SR-IOV NETWORK OPERATOR 설치 제거

SR-IOV Network Operator를 설치 제거하려면 실행 중인 SR-IOV 워크로드를 삭제하고 Operator를 제거한 다음 Operator에서 사용하는 Webhook를 삭제해야 합니다.

12.10.1. SR-IOV Network Operator 설치 제거

클러스터 관리자는 SR-IOV Network Operator를 제거할 수 있습니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 OpenShift Container Platform 클러스터에 액세스할 수 있습니다.
- SR-IOV Network Operator가 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 Operator를 제거합니다.
3. SR-IOV Network Operator가 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](#)

13장. OPENSIFT SDN 기본 CNI 네트워크 공급자

13.1. 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.

13.1.1. 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 4.6의 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.

13.1.2. 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 13.1. 기본 CNI 네트워크 공급자 기능 비교

기능	OpenShift SDN	OVN-Kubernetes
송신 IP	지원됨	지원됨
송신 방화벽 [1]	지원됨	지원됨
송신 라우터	지원됨	지원되지 않음
Kubernetes 네트워크 정책	부분적으로 지원됨 [2]	지원됨
멀티 캐스트	지원됨	지원됨

1. 송신 방화벽은 OpenShift SDN에서 송신 네트워크 정책이라고도 합니다. 이것은 네트워크 정책 송신과 동일하지 않습니다.
2. 송신 규칙 및 일부 **ipBlock** 규칙을 지원하지 않습니다.

13.2. 프로젝트의 송신 IP 구성

클러스터 관리자는 OpenShift SDN 기본 컨테이너 네트워크 인터페이스(CNI) 네트워크 공급자를 구성하여 하나 이상의 송신 IP 주소를 프로젝트에 할당할 수 있습니다.

13.2.1. 프로젝트 송신 트래픽에 대한 송신 IP 주소 할당

프로젝트의 송신 IP 주소를 구성하면 지정된 프로젝트의 모든 발신 외부 연결이 동일한 고정 소스 IP 주소를 공유합니다. 외부 리소스는 송신 IP 주소를 기반으로 특정 프로젝트의 트래픽을 인식할 수 있습니다. 프로젝트에 할당된 송신 IP 주소는 특정 목적지로 트래픽을 보내는 데 사용되는 송신 라우터와 다릅니다.

송신 IP 주소는 노드의 기본 네트워크 인터페이스에서 추가 IP 주소로 구현되며 노드의 기본 IP 주소와 동일한 서브넷에 있어야 합니다.

중요

송신 IP 주소는 **ifcfg-eth0**과 같은 Linux 네트워크 구성 파일에서 구성하지 않아야 합니다.

AWS(Amazon Web Services), GCP(Google Cloud Platform)의 송신 IP는 OpenShift Container Platform 버전 4.10 이상에서만 지원됩니다.

기본 네트워크 인터페이스에서 추가 IP 주소를 허용하려면 일부 가상 머신 솔루션을 사용할 때 추가 구성이 필요할 수 있습니다.

NetNamespace 오브젝트의 **egressIPs** 매개변수를 설정하여 네임스페이스에 송신 IP 주소를 지정할 수 있습니다. 송신 IP가 프로젝트와 연결된 후 OpenShift SDN을 사용하면 다음 두 가지 방법으로 송신 IP를 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.

중요

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.

13.2.1.1. 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.
- 자동 할당 모드를 사용하는 경우 네임스페이스당 하나의 송신 IP 주소만 지원됩니다.

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.

13.2.1.2. 수동으로 할당된 송신 IP 주소 사용 시 고려사항

이 방법은 추가 IP 주소를 퍼블릭 클라우드 환경과 같은 노드와 연결하는 데 제한이 있을 수 있는 클러스터에 사용됩니다.

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 개의 송신 IP 주소가 있는 경우 첫 번째 송신 IP 주소를 호스팅하는 노드에 도달할 수 없으면 OpenShift Container Platform은 첫 번째 송신 IP 주소에 다시 도달할 때까지 사용 가능한 다음 송신 IP 주소를 사용하도록 자동 전환합니다.

13.2.2. 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 \ 1
{
```

```
"egressIPs": [
  "<ip_address>" 2
]
```

- 1 프로젝트 이름을 지정합니다.
- 2 단일 송신 IP 주소를 지정합니다. 다중 IP 주소 사용은 지원되지 않습니다.

예를 들어 **project1**을 IP 주소 192.168.1.100에 할당하고 **project2**를 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 \ 1
{
  "egressCIDRs": [
    "<ip_address_range_1>", "<ip_address_range_2>" 2
  ]
}
```

- 1 노드 이름을 지정합니다.
- 2 CIDR 형식으로 하나 이상의 IP 주소 범위를 지정합니다.

예를 들어, **node1** 및 **node2**를 192.168.1.0에서 192.168.1.255 범위의 송신 IP 주소를 호스팅하도록 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.

13.2.3. 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> --type=merge -p \ 1
{
  "egressIPs": [ 2
    "<ip_address>"
  ]
}
```

- 1** 프로젝트 이름을 지정합니다.
- 2** 하나 이상의 송신 IP 주소를 지정합니다. **egressIPs** 매개변수는 배열입니다.

예를 들어, **project1** 프로젝트를 IP 주소 **192.168.1.100**에 할당하려면 다음을 수행합니다.

```
$ oc patch netnamespace project1 --type=merge \
-p '{"egressIPs": ["192.168.1.100"]}'
```

고가용성을 제공하기 위해 **egressIP**를 다른 노드에서 둘 이상의 IP 주소로 설정할 수 있습니다. 여러 송신 IP 주소가 설정되어 있으면 Pod는 목록의 첫 번째 IP를 송신에 사용하지만, 해당 IP 주소를 호스팅하는 노드가 실패하면 Pod는 잠시 후 목록의 다음 IP를 사용하도록 전환됩니다.



참고

OpenShift SDN은 **NetNamespace** 오브젝트를 관리하므로 기존 **NetNamespace** 오브젝트를 수정하기만 하면 됩니다. 새 **NetNamespace** 오브젝트를 생성하지 마십시오.

2. 송신 IP를 노드 호스트에 수동으로 할당합니다. 노드 호스트의 **HostSubnet** 오브젝트에서 **egressIPs** 매개변수를 설정합니다. 다음 JSON을 사용하여 해당 노드 호스트에 할당하려는 IP 수를 포함합니다.

```
$ oc patch hostsubnet <node_name> --type=merge -p \ 1
{
  "egressIPs": [ 2
    "<ip_address_1>",
    "<ip_address_N>"
  ]
}
```

- 1** 노드 이름을 지정합니다.
- 2** 하나 이상의 송신 IP 주소를 지정합니다. **egressIPs** 필드는 배열입니다.

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 hostsubnet 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 IP를 호스팅하는 노드로 라우팅된 다음 NAT를 사용하여 해당 IP 주소에 연결됩니다.

13.3. 프로젝트에 대한 송신 방화벽 구성

클러스터 관리자는 OpenShift Container Platform 클러스터에서 나가는 송신 트래픽을 제한하는 프로젝트에 대한 송신 방화벽을 생성할 수 있습니다.

13.3.1. 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.

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 **0.0.0.0/0**에 대한 거부 규칙이 포함된 경우 OpenShift Container Platform API 서버에 대한 액세스 권한이 차단됩니다. Pod에서 OpenShift Container Platform API 서버에 계속 액세스할 수 있도록 하려면 다음 예와 같이 API 서버가 송신 방화벽 규칙에서 수신 대기하는 IP 주소 범위를 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.

13.3.1.1. 송신 방화벽의 제한

송신 방화벽에는 다음과 같은 제한이 있습니다.

- EgressNetworkPolicy 오브젝트를 두 개 이상 보유할 수 있는 프로젝트는 없습니다.

- 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 프로젝트에서 송신 방화벽을 사용할 수 없습니다.

이러한 제한 사항을 위반하면 프로젝트의 송신 방화벽이 손상되고 모든 외부 네트워크 트래픽이 삭제될 수 있습니다.

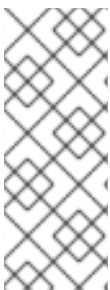
13.3.1.2. 송신 방화벽 정책 규칙에 대한 일치 순서

송신 방화벽 정책 규칙은 정의된 순서대로 처음부터 마지막까지 평가됩니다. Pod의 송신 연결과 일치하는 첫 번째 규칙이 적용됩니다. 해당 연결에 대한 모든 후속 규칙은 무시됩니다.

13.3.1.3. 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 로컬 권한이 없는 서버가 반환한 도메인의 TTL(Time to Live) 값을 기준으로 폴링됩니다.
- Pod는 필요한 경우 동일한 로컬 이름 서버에서 도메인을 확인해야 합니다. 확인하지 않으면 송신 방화벽 컨트롤러와 Pod에 의해 알려진 도메인의 IP 주소가 다를 수 있습니다. 호스트 이름의 IP 주소가 다르면 송신 방화벽이 일관되게 적용되지 않을 수 있습니다.
- 송신 방화벽 컨트롤러와 Pod는 동일한 로컬 이름 서버를 비동기적으로 폴링하기 때문에 Pod가 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.

13.3.2. 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> ❶
spec:
  egress: ❷
  ...
```

- ❶ 송신 방화벽 정책의 이름입니다.
- ❷ 다음 섹션에서 설명하는 하나 이상의 송신 네트워크 정책 규칙 컬렉션입니다.

13.3.2.1. EgressNetworkPolicy 규칙

다음 YAML은 송신 방화벽 규칙 오브젝트를 설명합니다. 송신 스탠자는 하나 이상의 오브젝트 배열을 예시합니다.

송신 정책 규칙 스탠자

```
egress:
- type: <type> ❶
  to: ❷
    cidrSelector: <cidr> ❸
    dnsName: <dns_name> ❹
```

- ❶ 규칙 유형입니다. 값은 허용 또는 거부여야 합니다.
- ❷ 송신 트래픽 일치 규칙을 설명하는 스탠자입니다. 규칙에 대한 **cidrSelector** 필드 또는 **dnsName** 필드의 값입니다. 동일한 규칙에서 두 필드를 모두 사용할 수 없습니다.
- ❸ CIDR 형식의 IP 주소 범위입니다,
- ❹ 도메인 이름입니다.

13.3.2.2. 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: ❶
  - 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.

13.3.3. 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 사용하는 클러스터
- 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. 선택 사항: 나중에 변경할 수 있도록 **<policy_name>.yaml** 파일을 저장합니다.

13.4. 프로젝트의 송신 방화벽 편집

클러스터 관리자는 기존 송신 방화벽에 대한 네트워크 트래픽 규칙을 수정할 수 있습니다.

13.4.1. EgressNetworkPolicy 오브젝트 보기

클러스터의 EgressNetworkPolicy 오브젝트를 확인할 수 있습니다.

사전 요구 사항

- OpenShift SDN CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용하는 클러스터입니다.
- **oc**로 알려진 OpenShift 명령 인터페이스 (CLI)를 설치합니다.
- 클러스터에 로그인해야 합니다.

절차

1. 선택 사항: 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
```

13.5. 프로젝트의 송신 방화벽 편집

클러스터 관리자는 기존 송신 방화벽에 대한 네트워크 트래픽 규칙을 수정할 수 있습니다.

13.5.1. EgressNetworkPolicy 오브젝트 편집

클러스터 관리자는 프로젝트의 송신 방화벽을 업데이트할 수 있습니다.

사전 요구 사항

- OpenShift SDN 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. 선택 사항: 송신 네트워크 방화벽을 생성할 때 EgressNetworkPolicy 오브젝트의 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. YAML을 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
```

13.6. 프로젝트에서 송신 방화벽 제거

클러스터 관리자는 프로젝트에서 송신 방화벽을 제거하여 OpenShift Container Platform 클러스터를 나가는 프로젝트에서 네트워크 트래픽에 대한 모든 제한을 제거할 수 있습니다.

13.6.1. EgressNetworkPolicy 오브젝트 제거

클러스터 관리자는 프로젝트에서 송신 방화벽을 제거할 수 있습니다.

사전 요구 사항

- OpenShift SDN 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>
```

13.7. 송신 라우터 POD 사용에 대한 고려 사항

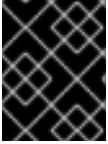
13.7.1. 송신 라우터 Pod 정보

OpenShift Container Platform 송신 라우터 Pod는 다른 용도로 사용되지 않는 프라이빗 소스 IP 주소를 사용하여 지정된 원격 서버로 트래픽을 리디렉션합니다. 이를 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.

13.7.1.1. 송신 라우터 모드

*리디렉션 모드*에서는 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.

*HTTP 프록시 모드*에서는 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 모드*에서는 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 HTTPS를 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.

13.7.1.2. 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.

13.7.1.3. 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)

RHOSP에 OpenShift Container Platform을 배포하는 경우 OpenStack 환경에서 IP 및 MAC 주소를 화이트리스트에 추가해야 합니다. 그렇지 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 **네트워크 필터 없음**을 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](#)

13.7.1.4. 장애 조치 구성

다운타임을 방지하기 위해 다음 예와 같이 **Deployment** 리소스를 사용하여 송신 라우터 Pod를 배포할 수 있습니다. 예제 배포를 위해 새 **Service** 오브젝트를 생성하려면 **oc expose deployment/egress-demo-controller** 명령을 사용하십시오.

```
apiVersion: v1
kind: Deployment
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 1
  selector:
    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 Pod만 지정된 송신 소스 IP 주소를 사용할 수 있으므로 복제본이 **1**로 설정되어 있는지 확인합니다. 이는 하나의 라우터 사본만 노드에서 실행됨을 의미합니다.

- 2 송신 라우터 Pod에 대한 **Pod** 오브젝트 템플릿을 지정합니다.

13.7.2. 추가 리소스

- [리디렉션 모드에서 송신 라우터 배포](#)
- [HTTP 프록시 모드에서 송신 라우터 배포](#)
- [DNS 프록시 모드에서 송신 라우터 배포](#)

13.8. 리디렉션 모드에서 송신 라우터 POD 배포

클러스터 관리자는 트래픽을 지정된 대상 IP 주소로 리디렉션하도록 구성된 송신 라우터 Pod를 배포할 수 있습니다.

13.8.1. 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 이 주석은 OpenShift Container Platform에 기본 네트워크 인터페이스 컨트롤러(NIC)에서 macvlan 네트워크 인터페이스를 생성하고 해당 macvlan 인터페이스를 Pod의 네트워크 네임스페이스로 이동하도록 지시합니다. **"true"** 값을 따옴표로 묶어야 합니다. OpenShift Container Platform이 다른 NIC 인터페이스에서 macvlan 인터페이스를 생성하도록 하려면 주석 값을 해당 인터페이스 이름으로 설정합니다. 예를 들면 **eth1**입니다.

- 2 송신 라우터 Pod에서 사용하도록 예약된 노드가 있는 물리적 네트워크의 IP 주소입니다. 선택 사항: 서브넷 길이인 /24 접미사를 포함하여 로컬 서브넷의 적절한 경로가 설정되도록 할 수 있습니다. 서
- 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

```

13.8.2. 송신 대상 구성 형식

송신 라우터 Pod가 리디렉션 모드로 배포되면 다음 형식 중 하나 이상을 사용하여 리디렉션 규칙을 지정할 수 있습니다.

- **<port> <protocol> <ip_address>** - 지정된 **<port>** 에 대한 수신 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 포트로 리디렉션됩니다. 대체 IP 주소가 없으면 기타 포트의 연결이 거부됩니다.

이어지는 예제에서는 몇 가지 규칙이 정의됩니다.

- 첫 번째 줄에서는 트래픽을 로컬 포트 **80**에서 **203.0.113.25**의 포트 **80**으로 리디렉션합니다.
- 두 번째 및 세 번째 줄에서는 로컬 포트 **8080** 및 **8443**을 **203.0.113.26**의 원격 포트 **80** 및 **443**으로 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
```

13.8.3. 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 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
      port: 80
    - name: https
      port: 443
  type: ClusterIP
selector:
  name: egress-1
```

이제 Pod에서 이 서비스에 연결할 수 있습니다. 이러한 연결은 예약된 송신 IP 주소를 사용하여 외부 서버의 해당 포트로 리디렉션됩니다.

13.8.4. 추가 리소스

- ConfigMap을 사용하여 송신 라우터 대상 매핑 구성

13.9. HTTP 프록시 모드에서 송신 라우터 POD 배포

클러스터 관리자는 지정된 HTTP 및 HTTPS 기반 서비스로 트래픽을 프록시하도록 구성된 송신 라우터 Pod를 배포할 수 있습니다.

13.9.1. HTTP 모드에 대한 송신 라우터 Pod 사양

Pod 오브젝트에서 송신 라우터 Pod에 대한 구성을 정의합니다. 다음 YAML은 HTTP 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_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 이 주석은 OpenShift Container Platform에 기본 네트워크 인터페이스 컨트롤러(NIC)에서 macvlan 네트워크 인터페이스를 생성하고 해당 macvlan 인터페이스를 Pod의 네트워크 네임스페이스로 이동하도록 지시합니다. **"true"** 값을 따옴표로 묶어야 합니다. OpenShift Container Platform이 다른 NIC 인터페이스에서 macvlan 인터페이스를 생성하도록 하려면 주석 값을 해당 인터페이스 이름으로 설정합니다. 예를 들면 **eth1**입니다.
- 2 송신 라우터 Pod에서 사용하도록 예약된 노드가 있는 물리적 네트워크의 IP 주소입니다. 선택 사항: 서브넷 길이인 /24 접미사를 포함하여 로컬 서브넷의 적절한 경로가 설정되도록 할 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.

13.9.2. 송신 대상 구성 형식

송신 라우터 Pod가 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.com과 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
*
```

13.9.3. HTTP 프록시 모드에서 송신 라우터 Pod 배포

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.

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 ❶
  type: ClusterIP
  selector:
    name: egress-1

```

❶ 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/ ❶
  - name: https_proxy
    value: http://egress-1:8080/
  ...

```

❶ 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 툴이나 소프트웨어에 대한 설명서를 참조하십시오.

13.9.4. 추가 리소스

- [ConfigMap을 사용하여 송신 라우터 대상 매핑 구성](#)

13.10. DNS 프록시 모드에서 송신 라우터 POD 배포

클러스터 관리자는 지정된 DNS 이름 및 IP 주소로 트래픽을 프록시하도록 구성된 송신 라우터 Pod를 배포할 수 있습니다.

13.10.1. 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 이 주석은 OpenShift Container Platform에 기본 네트워크 인터페이스 컨트롤러(NIC)에서 macvlan 네트워크 인터페이스를 생성하고 해당 macvlan 인터페이스를 Pod의 네트워크 네임스페이스로 이동하도록 지시합니다. **"true"** 값을 따옴표로 묶어야 합니다. OpenShift Container Platform이 다른 NIC 인터페이스에서 macvlan 인터페이스를 생성하도록 하려면 주석 값을 해당 인터페이스 이름으로 설정합니다. 예를 들면 **eth1**입니다.
- ❷ 송신 라우터 Pod에서 사용하도록 예약된 노드가 있는 물리적 네트워크의 IP 주소입니다. 선택 사항: 서브넷 길이인 **/24** 접미사를 포함하여 로컬 서브넷의 적절한 경로가 설정되도록 할 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.
- ❹ 하나 이상의 프록시 대상 목록을 지정합니다.
- ❺ 선택 사항: DNS 프록시 로그 출력을 **stdout**에 출력하도록 를 지정합니다.

13.10.2. 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 Pod는 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 DNS를 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
```

13.10.3. 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 Pod에서 이 서비스에 연결할 수 있습니다. 이러한 연결은 예약된 송신 IP 주소를 사용하여 외부 서버의 해당 포트에 프록시로 연결됩니다.

13.10.4. 추가 리소스

- [ConfigMap을 사용하여 송신 라우터 대상 매핑 구성](#)

13.11. 구성 맵에서 송신 라우터 POD 대상 목록 구성

클러스터 관리자는 송신 라우터 Pod에 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 Pod에 대한 설명서를 참조하십시오.

13.11.1. 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 Pod에는 권한 있는 컨테이너가 필요하기 때문에 **cluster-admin** 권한이 없는 사용자는 Pod 정의를 직접 편집할 수 없습니다.



참고

송신 라우터 Pod는 구성 맵이 변경될 때 자동으로 업데이트되지 않습니다. 업데이트하려면 송신 라우터 Pod를 재시작해야 합니다.

사전 요구 사항

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

프로세스

1. 다음 예와 같이 송신 라우터 Pod에 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.

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
...
```

13.11.2. 추가 리소스

- [리디렉션 모드](#)
- [HTTP 프록시 모드](#)
- [DNS 프록시 모드](#)

13.12. 프로젝트에 멀티 캐스트 사용

13.12.1. 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.

13.12.2. 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>
```

- 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
```

- 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
```

- 새 터미널 창 또는 탭에서 멀티캐스트 리스너를 시작합니다.

- Pod의 IP 주소를 가져옵니다.

```
$ POD_IP=$(oc get pods mlistener -o jsonpath='{.status.podIP}')
```

- 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
```

- 멀티 캐스트 송신기를 시작합니다.

- 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
```

13.13. 프로젝트에 대한 멀티 캐스트 비활성화

13.13.1. 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.

13.14. OPENSIFT SDN을 사용하여 네트워크 격리 구성

OpenShift SDN CNI 플러그인에 멀티 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 pod와 service 간에 네트워크 트래픽을 허용할 수 있습니다.
- 프로젝트의 네트워크 격리를 비활성화할 수 있습니다. 다른 모든 프로젝트에서 pod 및 service의 네트워크 트래픽을 수락하여 전역에서 액세스할 수 있습니다. 전역에서 액세스 가능한 프로젝트는 다른 모든 프로젝트의 pod 및 service에 액세스할 수 있습니다.

13.14.1. 사전 요구 사항

- 다중 테넌트 격리 모드에서 OpenShift SDN CNI(Container Network Interface) 플러그인을 사용하도록 구성된 클러스터가 있어야 합니다.

13.14.2. 프로젝트 참여

두 개 이상의 프로젝트에 참여하여 다른 프로젝트의 Pod와 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 함께 참여한 Pod 네트워크를 확인합니다.

```
$ oc get netnamespaces
```

동일한 Pod 네트워크에 있는 프로젝트는 **NETID** 열에서 동일한 네트워크 ID를 보유합니다.

13.14.3. 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.

13.14.4. 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.

13.15. KUBE-PROXY 설정

Kubernetes 네트워크 프록시(kube-proxy)는 각 노드에서 실행되며 CNO(Cluster Network Operator)에 의해 관리됩니다. kube-proxy는 서비스와 관련된 끝점에 대한 연결을 전달하기 위한 네트워크 규칙을 유지 관리합니다.

13.15.1. iptables 규칙 동기화 정보

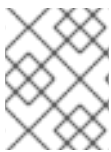
동기화 기간은 Kubernetes 네트워크 프록시(kube-proxy)가 노드에서 iptables 규칙을 동기화하는 빈도를 결정합니다.

다음 이벤트 중 하나가 발생하면 동기화가 시작됩니다.

- 서비스 또는 끝점과 같은 이벤트가 클러스터에 추가되거나 클러스터에서 제거됩니다.
- 마지막 동기화 이후 시간이 kube-proxy에 대해 정의된 동기화 기간을 초과합니다.

13.15.2. 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 13.2. 매개변수

매개변수	설명	값	기본
iptablesSyncPeriod	iptables 규칙의 새로 고침 간격으로,	30s 또는 2m 과 같은 시간 간격입니다. 유효 접미사로 s , m , h 가 있으며, 자세한 설명은 Go time 패키지 문서를 참조하십시오.	30s

매개변수	설명	값	기본
proxyArguments.iptables-min-sync-period	iptables 규칙을 새로 고치기 전 최소 기간입니다. 이 매개변수를 이용하면 새로 고침 간격이 너무 짧지 않도록 조정할 수 있습니다. 기본적으로 새로 고침은 iptables 규칙에 영향을 주는 변경이 발생하는 즉시 시작됩니다.	30s 또는 2m 과 같은 시간 간격입니다. 유효 접미사로 s, m 및 h 가 있으며, 자세한 설명은 Go time 패키지 를 참조하십시오.	0s

13.15.3. 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 CR과 같이 kube-proxy 구성을 변경하여 CR의 **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: 다음 명령을 입력하여 Cluster Network Operator가 구성 변경을 승인했는지 확인합니다.

```
$ oc get clusteroperator network
```

출력 예

```

NAME      VERSION  AVAILABLE  PROGRESSING  DEGRADED  SINCE
network  4.1.0-0.9  True       False        False     1m

```

구성 업데이트가 성공적으로 적용되면 **AVAILABLE** 필드는 **True**입니다.

14장. OVN-KUBERNETES 기본 CNI 네트워크 공급자

14.1. 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.

14.1.1. 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.

14.1.2. 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 14.1. 기본 CNI 네트워크 공급자 기능 비교

기능	OVN-Kubernetes	OpenShift SDN
송신 IP	지원됨	지원됨
송신 방화벽 [1]	지원됨	지원됨
송신 라우터	지원되지 않음	지원됨
Kubernetes 네트워크 정책	지원됨	부분적으로 지원됨 [2]
멀티 캐스트	지원됨	지원됨

1. 송신 방화벽은 OpenShift SDN에서 송신 네트워크 정책이라고도 합니다. 이것은 네트워크 정책 송신과 동일하지 않습니다.
2. 송신 규칙 및 일부 **ipBlock** 규칙을 지원하지 않습니다.

14.1.3. OVN-Kubernetes 제한 사항

OVN-Kubernetes CNI(Container Network Interface) 클러스터 네트워크 공급자에는 트래픽 정책과 관련된 제한 사항이 있습니다. 네트워크 프로바이더는 Kubernetes 서비스의 외부 트래픽 정책 또는 내부 트래픽 정책 설정을 **local**로 설정할 수 없습니다. 두 매개변수 모두에서 기본값인 **cluster**가 지원됩니다. 이 제한은 **LoadBalancer,NodePort** 유형의 서비스를 추가하거나 외부 IP를 사용하여 서비스를 추가할 때 영향을 줄 수 있습니다.

추가 리소스

- [프로젝트에 대한 송신 방화벽 구성](#)
- [네트워크 정책 정의](#)
- [프로젝트에 멀티 캐스트 사용](#)
- [Network \[operator.openshift.io/v1\]](#)

14.2. OPENSIFT SDN 클러스터 네트워크 공급자에서 마이그레이션

클러스터 관리자는 OpenShift SDN CNI 클러스터 네트워크 공급자에서 OVN-Kubernetes CNI (Container Network Interface) 클러스터 네트워크 공급자로 마이그레이션할 수 있습니다.

OVN-Kubernetes에 대한 자세한 내용은 [OVN-Kubernetes 네트워크 공급자 정보](#)를 참조하십시오.

14.2.1. OVN-Kubernetes 네트워크 공급자로 마이그레이션

OVN-Kubernetes CNI(Container Network Interface) 기본 네트워크 공급자로 마이그레이션하는 것은 클러스터에 연결할 수 없는 일부 중단 시간이 포함된 수동 프로세스입니다. 롤백 절차가 제공되지만 마이그레이션은 단방향 프로세스로 설정됩니다.



참고

베어 메탈 하드웨어의 설치 관리자 프로비저닝 클러스터에서만 OVN-Kubernetes 네트워크 공급자로의 마이그레이션이 지원됩니다.

베어 메탈 하드웨어에서 사용자 프로비저닝 클러스터에서 마이그레이션을 수행하는 것은 지원되지 않습니다.

14.2.1.1. OVN-Kubernetes 네트워크 공급자로 마이그레이션에 대한 고려 사항

노드에 할당된 서브넷과 개별 포드에 할당된 IP 주소는 마이그레이션 중에 유지되지 않습니다.

OVN-Kubernetes 네트워크 공급자는 OpenShift SDN 네트워크 공급자에 있는 많은 기능을 구현하지만 구성은 동일하지 않습니다.

- 클러스터에서 다음 OpenShift SDN 기능을 사용하는 경우 OVN-Kubernetes에서 동일한 기능을 수동으로 구성해야 합니다.
 - 네임스페이스 격리
 - 송신 IP 주소
 - 송신 네트워크 정책
 - 송신 라우터 포트

- 멀티 캐스트
- 클러스터에서 **100.64.0.0/16** IP 주소 범위의 모든 부분을 사용하는 경우 이 IP 주소 범위를 내부적으로 사용하므로 OVN-Kubernetes로 마이그레이션할 수 없습니다.

다음 섹션에서는 OVN-Kubernetes와 OpenShift SDN의 앞서 언급한 기능 간 구성의 차이점을 설명합니다.

네임스페이스 격리

OVN-Kubernetes는 네트워크 정책 격리 모드만 지원합니다.



중요

클러스터가 다중 테넌트 또는 서브넷 격리 모드에서 구성된 OpenShift SDN을 사용하는 경우 OVN-Kubernetes 네트워크 공급자로 마이그레이션할 수 없습니다.

송신 IP 주소

OVN-Kubernetes와 OpenShift SDN 간의 송신 IP 주소를 구성하는 데 있어서 차이점은 다음 표에 설명되어 있습니다.

표 14.2. 송신 IP 주소 구성의 차이점

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● EgressIPs 오브젝트 생성 ● Node 오브젝트에 주석 추가 	<ul style="list-style-type: none"> ● NetNamespace 오브젝트 패치 ● HostSubnet 오브젝트 패치

OVN-Kubernetes에서 송신 IP 주소를 사용하는 방법에 대한 자세한 내용은 "송신 IP 주소 구성"을 참조하십시오.

송신 네트워크 정책

OVN-Kubernetes와 OpenShift SDN 간의 송신 방화벽이라고도 하는 송신 네트워크 정책 구성의 차이점은 다음 표에 설명되어 있습니다.

표 14.3. 송신 네트워크 정책 구성의 차이점

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● 네임스페이스에 EgressFirewall 오브젝트 생성 	<ul style="list-style-type: none"> ● 네임스페이스에서 EgressNetworkPolicy 오브젝트 생성

OVN-Kubernetes에서 송신 방화벽을 사용하는 방법에 대한 자세한 내용은 "프로젝트에 대한 송신 방화벽 구성"을 참조하십시오.

송신 라우터 Pod

OVN-Kubernetes는 OpenShift Container Platform 4.6에서 송신 라우터 Pod 사용을 지원하지 않습니다.

멀티 캐스트

OVN-Kubernetes 및 OpenShift SDN에서 멀티 캐스트 트래픽 활성화의 차이점은 다음 표에 설명되어 있습니다.

표 14.4. 멀티 캐스트 구성의 차이점

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● Namespace 오브젝트에 주석 추가 	<ul style="list-style-type: none"> ● NetNamespace 오브젝트에 주석 추가

OVN-Kubernetes에서 멀티 캐스트를 사용하는 방법에 대한 자세한 내용은 "프로젝션에 멀티 캐스트 사용"을 참조하십시오.

네트워크 정책

OVN-Kubernetes는 **networking.k8s.io/v1** API 그룹에서 Kubernetes **NetworkPolicy** API를 완전히 지원합니다. OpenShift SDN에서 마이그레이션할 때 네트워크 정책에 변경 사항이 필요하지 않습니다.

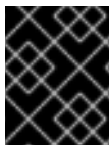
14.2.1.2. 마이그레이션 프로세스의 작동 방식

마이그레이션 프로세스는 다음과 같이 작동합니다.

1. CNO(Cluster Network Operator) 구성 오브젝트에 설정된 임시 주석을 설정합니다. 이 주석은 CNO를 트리거하여 **defaultNetwork** 필드에 대한 변경 사항을 확인합니다.
2. MCO(Machine Config Operator)가 마이그레이션을 중단하지 않도록 일시 중지합니다.
3. **defaultNetwork** 필드를 업데이트합니다. 업데이트로 CNO가 OpenShift SDN 컨트롤 플레인 포드를 제거하고 OVN-Kubernetes 컨트롤 플레인 포드를 배포합니다. 또한 새 클러스터 네트워크 공급자를 반영하도록 Multus 오브젝트를 업데이트합니다.
4. 클러스터의 각 노드를 재부팅합니다. 클러스터의 기존 포드는 클러스터 네트워크 공급자로의 변경을 인식하지 못하므로 각 노드를 재부팅하면 각 노드가 포드를 드레인합니다. 새 포드는 OVN-Kubernetes에서 제공하는 새 클러스터 네트워크에 연결되어 있습니다.
5. 클러스터 재부팅의 모든 노드 후에 MCO를 활성화합니다. MCO는 마이그레이션을 완료하는 데 필요한 **systemd** 구성에 대한 업데이트를 롤아웃합니다. MCO는 기본적으로 한 번에 풀당 단일 머신을 업데이트하므로 마이그레이션이 걸리는 총 시간이 클러스터 크기와 함께 증가합니다.

14.2.2. OVN-Kubernetes 기본 CNI 네트워크 공급자로 마이그레이션

클러스터 관리자는 클러스터의 기본 CNI(Container Network Interface) 네트워크 공급자를 OVN-Kubernetes로 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

- 베어 메탈 설치 관리자 프로비저닝 인프라에 설치되고 네트워크 정책 격리 모드에서 OpenShift SDN 기본 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.
- 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 사용하려면 다음 명령을 입력하여 Cluster Network Operator 구성 오브젝트에 주석을 설정합니다.

```
$ oc annotate Network.operator.openshift.io cluster \
'networkoperator.openshift.io/network-migration='"
```

3. 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 구성 풀을 중지합니다.

```
$ oc patch MachineConfigPool worker --type='merge' --patch \
'{"spec":{"paused": true }}'
```

4. OVN-Kubernetes 클러스터 네트워크 공급자를 구성하려면 다음 명령을 입력합니다.

```
$ oc patch Network.config.openshift.io cluster \
--type='merge' --patch '{"spec":{"networkType": "OVNKubernetes" }}'
```

5. 선택 사항: OVN-Kubernetes에 대해 네트워크 인프라 요구 사항을 충족하도록 다음 설정을 사용자 지정할 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. 기본 값을 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 MTU입니다. MTU 값은 일반적으로 자동으로 지정되지만 클러스터의 모든 노드가 동일한 MTU를 사용하지 않을 때는 최소 노드 MTU 값에서 **100**을 뺀 값으로 명시적으로 설정해야 합니다.

port

Geneve 오버레이 네트워크용 UDP 포트입니다.

mtu 필드를 업데이트하는 패치 명령 예

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
  --patch '{
  "spec":{
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "mtu":1200
      }
    }
  }
}'
```

- 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
```

- 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.

- 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 }}'
```

MCO는 각 구성 풀에서 머신을 업데이트하므로 각 노드를 재부팅합니다.

기본적으로 MCO는 한 번에 풀당 단일 머신을 업데이트하므로 마이그레이션이 완료하는 데 필요한 시간은 클러스터 크기와 함께 증가합니다.

9. 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
```

10. 마이그레이션이 성공했는지 확인합니다.

- a. 기본 CNI 네트워크 공급자가 OVN-Kubernetes인지 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. 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 임의의 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.

d. Pod가 오류 상태가 아닌지 확인하려면 다음 명령을 입력합니다.

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

노드의 Pod가 오류 상태인 경우 해당 노드를 재부팅합니다.

11. 마이그레이션이 성공하고 클러스터가 양호한 상태인 경우에만 다음 단계를 완료합니다.

a. Cluster Network Operator 구성 오브젝트에서 마이그레이션 주석을 제거하려면 다음 명령을 입력합니다.

```
$ oc annotate Network.operator.openshift.io cluster \
networkoperator.openshift.io/network-migration-
```

b. OpenShift SDN 네트워크 공급자 네임스페이스를 제거하려면 다음 명령을 입력합니다.

```
$ oc delete namespace openshift-sdn
```

14.2.3. 추가 리소스

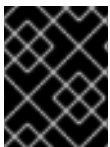
- OVN-Kubernetes 기본 CNI 네트워크 공급자에 대한 구성 매개변수
- etcd 백업
- 네트워크 정책 정의
- OVN-Kubernetes 기능
 - 송신 IP 주소 구성
 - 프로젝트에 대한 송신 방화벽 구성
 - 프로젝트에 멀티 캐스트 사용
- OpenShift SDN 기능
 - 프로젝트의 송신 IP 구성
 - 프로젝트에 대한 송신 방화벽 구성
 - 프로젝트에 멀티 캐스트 사용
- Network [operator.openshift.io/v1]

14.3. 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.

14.3.1. 기본 CNI 네트워크 공급자를 OpenShift SDN으로 롤백

클러스터 관리자는 클러스터를 OpenShift SDN 기본 CNI(Container Network Interface) 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 기본 CNI 네트워크 공급자로 구성된 베어 메탈 인프라에 설치된 클러스터입니다.

절차

1. 마이그레이션을 사용하려면 다음 명령을 입력하여 Cluster Network Operator 구성 오브젝트에 주석을 설정합니다.

```
$ oc annotate Network.operator.openshift.io cluster \
'networkoperator.openshift.io/network-migration='"
```

2. 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 구성 풀을 중지합니다.

```
$ oc patch MachineConfigPool worker --type='merge' --patch \
'{"spec":{"paused": true }}'
```

3. OpenShift SDN 클러스터 네트워크 공급자를 구성하려면 다음 명령을 입력합니다.

```
$ oc patch Network.config.openshift.io cluster \
--type='merge' --patch '{"spec":{"networkType": "OpenShiftSDN"} }'
```

4. 선택 사항: OpenShift SDN에 대해 네트워크 인프라 요구 사항을 충족하도록 다음 설정을 사용자 지정할 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>,
        "vxlانPort":<port>
      }
    }
  }
}'
```

mtu

Geneve 오버레이 네트워크용 MTU입니다. MTU 값은 일반적으로 자동으로 지정되지만 클러스터의 모든 노드가 동일한 MTU를 사용하지 않을 때는 최소 노드 MTU 값에서 **100**을 뺀 값으로 명시적으로 설정해야 합니다.

port

Geneve 오버레이 네트워크용 UDP 포트입니다.

패치 명령 예

```
$ oc patch Network.operator.openshift.io cluster --type=merge \
--patch '{
  "spec":{
    "defaultNetwork":{
```

```
"openshiftSDNConfig":{
  "mtu":1200
}}}'
```

5. 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
```

6. 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
```

7. 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 }}'
```

MCO는 각 구성 풀에서 머신을 업데이트하므로 각 노드를 재부팅합니다.

기본적으로 MCO는 한 번에 풀당 단일 머신을 업데이트하므로 마이그레이션이 완료하는 데 필요한 시간은 클러스터 크기와 함께 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
```

여기서 **<config_name>**은 **machineconfiguration.openshift.io/currentConfig** 필드에서 머신 구성의 이름입니다.

9. 마이그레이션이 성공했는지 확인합니다.

a. 기본 CNI 네트워크 공급자가 OVN-Kubernetes인지 확인하려면 다음 명령을 입력합니다. **status.networkType** 값은 **OpenShiftSDN**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. 노드가 **NotReady** 상태에 있는 경우 머신 구성 데몬 포트 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 임의의 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. Pod가 오류 상태가 아닌지 확인하려면 다음 명령을 입력합니다.

```
$ oc get pods --all-namespaces -o wide --sort-by='{.spec.nodeName}'
```

노드의 Pod가 오류 상태인 경우 해당 노드를 재부팅합니다.

10. 마이그레이션이 성공하고 클러스터가 양호한 상태인 경우에만 다음 단계를 완료합니다.

- a. Cluster Network Operator 구성 오브젝트에서 마이그레이션 주석을 제거하려면 다음 명령을 입력합니다.

```
$ oc annotate Network.operator.openshift.io cluster \
networkoperator.openshift.io/network-migration-
```

- b. OVN-Kubernetes 네트워크 공급자 네임스페이스를 제거하려면 다음 명령을 입력합니다.

```
$ oc delete namespace openshift-ovn-kubernetes
```

14.4. 프로젝트에 대한 송신 방화벽 구성

클러스터 관리자는 OpenShift Container Platform 클러스터에서 나가는 송신 트래픽을 제한하는 프로젝트에 대한 송신 방화벽을 생성할 수 있습니다.

14.4.1. 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.

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
- 포트 번호
- 다음 프로토콜 중 하나인 프로토콜: TCP, UDP 및 SCTP

중요

송신 방화벽에 **0.0.0.0/0**에 대한 거부 규칙이 포함된 경우 OpenShift Container Platform API 서버에 대한 액세스 권한이 차단됩니다. Pod에서 OpenShift Container Platform API 서버에 계속 액세스할 수 있도록 하려면 다음 예와 같이 API 서버가 송신 방화벽 규칙에서 수신 대기하는 IP 주소 범위를 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.

14.4.1.1. 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.

이러한 제한 사항을 위반하면 프로젝트의 송신 방화벽이 손상되고 모든 외부 네트워크 트래픽이 삭제될 수 있습니다.

14.4.1.2. 송신 방화벽 정책 규칙에 대한 일치 순서

송신 방화벽 정책 규칙은 정의된 순서대로 처음부터 마지막까지 평가됩니다. Pod의 송신 연결과 일치하는 첫 번째 규칙이 적용됩니다. 해당 연결에 대한 모든 후속 규칙은 무시됩니다.

14.4.2. 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 YAML은 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.

14.4.2.1. EgressFirewall 규칙

다음 YAML은 송신 방화벽 규칙 오브젝트를 설명합니다. 송신 스탠자는 하나 이상의 오브젝트 배열을 예상합니다.

송신 정책 규칙 스탠자

```
egress:
- type: <type> ①
  to: ②
  cidrSelector: <cidr> ③
  ports: ④
  ...
```

- ① 규칙 유형입니다. 값은 허용 또는 거부여야 합니다.

- 2 송신 트래픽 일치 규칙을 설명하는 스탠자입니다.
- 3 CIDR 형식의 IP 주소 범위입니다,
- 4 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 **protocol**의 값도 지정해야 합니다.
- 2 네트워크 프로토콜. 값은 **TCP**, **UDP** 또는 **SCTP**여야 합니다.

14.4.2.2. 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
```

14.4.3. 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. 선택 사항: 나중에 변경할 수 있도록 **<policy_name>.yaml** 파일을 저장합니다.

14.5. 프로젝트의 송신 방화벽 보기

클러스터 관리자는 기존 송신 방화벽의 이름을 나열하고 특정 송신 방화벽에 대한 트래픽 규칙을 볼 수 있습니다.

14.5.1. EgressFirewall 오브젝트 보기

클러스터의 EgressFirewall 오브젝트를 볼 수 있습니다.

사전 요구 사항

- OVN-Kubernetes 기본 CNI(Container Network Interface) 네트워크 공급자 플러그인을 사용하는 클러스터입니다.
- **oc**로 알려진 OpenShift 명령 인터페이스 (CLI)를 설치합니다.
- 클러스터에 로그인해야 합니다.

절차

1. 선택 사항: 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
```

14.6. 프로젝트의 송신 방화벽 편집

클러스터 관리자는 기존 송신 방화벽에 대한 네트워크 트래픽 규칙을 수정할 수 있습니다.

14.6.1. 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 생성할 때 EgressFirewall 오브젝트 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. YAML을 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
```

14.7. 프로젝트에서 송신 방화벽 제거

클러스터 관리자는 프로젝트에서 송신 방화벽을 제거하여 OpenShift Container Platform 클러스터를 나가는 프로젝트에서 네트워크 트래픽에 대한 모든 제한을 제거할 수 있습니다.

14.7.1. 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>
```

14.8. 송신 IP 주소 구성

클러스터 관리자는 OVN-Kubernetes 기본 컨테이너 네트워크 인터페이스(CNI) 네트워크 공급자를 구성하여 하나 이상의 송신 IP 주소를 네임스페이스 또는 네임스페이스내 특정 Pod에 할당할 수 있습니다.

14.8.1. 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.

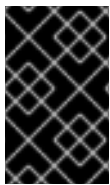
예를 들어 클러스터 외부 서버에서 호스팅되는 데이터베이스를 주기적으로 쿼리하는 Pod가 있을 수 있습니다. 서버에 대한 액세스 요구 사항을 적용하기 위해 패킷 필터링 장치는 특정 IP 주소의 트래픽만 허용하도록 구성됩니다. 특정 Pod에서만 서버에 안정적으로 액세스할 수 있도록 허용하려면 서버에 요청하는 Pod에 대해 특정 송신 IP 주소를 구성하면 됩니다.

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.

일부 클러스터 구성에서 애플리케이션 Pod 및 인그레스 라우터 Pod가 동일한 노드에서 실행됩니다. 이 시나리오에서 애플리케이션 프로젝트에 대한 송신 IP를 구성하면 애플리케이션 프로젝트에서 경로에 요청을 보낼 때 IP가 사용되지 않습니다.

14.8.1.1. 플랫폼 지원

다음 표에는 다양한 플랫폼의 송신 IP 주소 기능에 대한 지원이 요약되어 있습니다.



중요

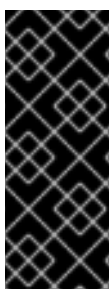
송신 IP 주소 구현은 AWS(Amazon Web Services), Azure Cloud 또는 송신 IP 기능에 필요한 자동 계층 2 네트워크 조작과 호환되지 않는 기타 퍼블릭 클라우드 플랫폼과 호환되지 않습니다.

플랫폼	지원됨
베어 메탈	예
vSphere	예
Red Hat OpenStack Platform (RHOSP)	아니요
퍼블릭 클라우드	아니요

14.8.1.2. 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 Platform에서 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.

14.8.1.3. 노드에 송신 IP 할당

EgressIP 오브젝트를 생성할 때 `k8s.ovn.org/egress-assignable: ""` 레이블이 지정된 노드에 다음 조건이 적용됩니다.

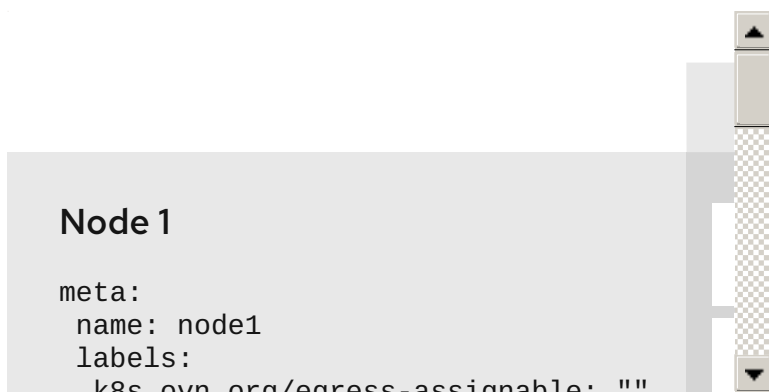
- 송신 IP 주소는 한 번에 두 개 이상의 노드에 할당되지 않습니다.
- 송신 IP 주소는 송신 IP 주소를 호스팅할 수 있는 사용 가능한 노드 간에 균형을 이룹니다.
- **EgressIP** 오브젝트의 `spec.EgressIPs` 배열에서 두 개 이상의 IP 주소를 지정해도 노드에서는 지정된 주소 중 두 개 이상을 호스팅하지 않습니다.
- 노드를 사용할 수 없게 되면 할당된 모든 송신 IP 주소가 이전에 설명한 조건에 따라 자동으로 재할당됩니다.

Pod가 여러 **EgressIP** 오브젝트의 선택기와 일치하는 경우 **EgressIP** 오브젝트에 지정된 송신 IP 주소 중 어느 것이 Pod의 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 Pod가 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.

14.8.1.4. 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 `192.168.126.102`입니다.

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:
  assignments:
    - node: node1
      egressIP: 192.168.126.10
    - node: node3
      egressIP: 192.168.126.102

```

이전 예제의 구성에 대해 OpenShift Container Platform은 두 송신 IP 주소를 사용 가능한 노드에 할당합니다. **status** 필드는 송신 IP 주소가 할당되었는지 여부와 위치를 반영합니다.

14.8.2. EgressIP 오브젝트

다음 YAML에서는 **EgressIP** 오브젝트의 API를 설명합니다. 오브젝트의 범위는 클러스터 전체이며 네임 스페이스에 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 선택 사항: 송신 IP 주소를 연결할 지정된 네임스페이스의 Pod에 대한 하나 이상의 선택기입니다. 이러한 선택기를 적용하면 네임스페이스 내에서 Pod 하위 집합을 선택할 수 있습니다.

다음 YAML은 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 스탠자를 설명합니다.

Pod 선택기 스탠자

```
podSelector: ①
matchLabels:
  <label_name>: <label_value>
```

- ① 선택 사항: 지정된 namespace **Selector** 규칙과 일치하는 네임스페이스의 Pod에 대해 일치하는 하나 이상의 규칙입니다. 지정된 경우 일치하는 Pod만 선택됩니다. 네임스페이스의 다른 Pod는 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 Pod와 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
```

14.8.3. 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.

14.8.4. 다음 단계

- [송신 IP 할당](#)

14.8.5. 추가 리소스

- [LabelSelector meta/v1](#)

- [LabelSelectorRequirement meta/v1](#)

14.9. 송신 IP 주소 할당

클러스터 관리자는 네임스페이스 또는 네임스페이스의 특정 Pod에서 클러스터를 떠나는 트래픽에 송신 IP 주소를 할당할 수 있습니다.

14.9.1. 네임스페이스에 송신 IP 주소 할당

하나 이상의 송신 IP 주소를 네임스페이스 또는 네임스페이스의 특정 Pod에 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. 선택 사항: 나중에 변경할 수 있도록 **<egressips_name>.yaml** 파일을 저장합니다.

4. 송신 IP 주소가 필요한 네임스페이스에 레이블을 추가합니다. 1단계에 정의된 **EgressIP** 오브젝트의 네임스페이스에 레이블을 추가하려면 다음 명령을 실행합니다.

```
$ oc label ns <namespace> env=qa 1
```

- 1 **<namespace>** 를 송신 IP 주소가 필요한 네임스페이스로 바꿉니다.

14.9.2. 추가 리소스

- 송신 IP 주소 구성

14.10. 프로젝트에 멀티 캐스트 사용

14.10.1. 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 프로젝트별로 멀티 캐스트를 활성화할 수 있습니다.

14.10.2. 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
```

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 멀티캐스트 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
```

14.11. 프로젝트에 대한 멀티 캐스트 비활성화

14.11.1. 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-
```

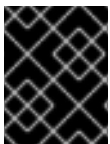
1 멀티 캐스트를 비활성화하려는 프로젝트의 **namespace**입니다.

14.12. 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.

14.12.1. OVN-Kubernetes로 하이브리드 네트워킹 구성

OVN-Kubernetes에서 하이브리드 네트워킹을 사용하도록 클러스터를 구성할 수 있습니다. 이를 통해 다양한 노드 네트워킹 구성을 지원하는 하이브리드 클러스터를 사용할 수 있습니다. 예를 들어 클러스터에서 Linux 및 Windows 노드를 모두 실행하려면 이 작업이 필요합니다.



중요

클러스터를 설치하는 동안 OVN-Kubernetes를 사용하여 하이브리드 네트워킹을 구성해야 합니다. 설치 프로세스 후에는 하이브리드 네트워킹으로 전환할 수 없습니다.

사전 요구 사항

- **install-config.yaml** 파일에 **networking.networkType** 매개 변수의 **OVNKubernetes**가 정의되어 있어야 합니다. 자세한 내용은 선택한 클라우드 공급자에서 OpenShift Container Platform 네트워크 사용자 정의 설정에 필요한 설치 문서를 참조하십시오.

프로세스

1. 설치 프로그램이 포함된 디렉터리로 변경하고 매니페스트를 생성합니다.

```
$ ./openshift-install create manifests --dir <installation_directory>
```

다음과 같습니다.

<installation_directory>

클러스터의 **install-config.yaml** 파일이 포함된 디렉터리의 이름을 지정합니다.

2. **<installation_directory>/manifests/** 디렉터리에 **cluster-network-03-config.yaml**이라는 stub 매니페스트 파일을 만듭니다.

```
$ cat <<EOF > <installation_directory>/manifests/cluster-network-03-config.yaml
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.yaml** 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: 1
        - cidr: 10.132.0.0/14
          hostPrefix: 23
        hybridOverlayVXLANPort: 9898 2
```

- 1 추가 오버레이 네트워크의 노드에 사용되는 CIDR 구성을 지정합니다. **hybridClusterNetwork** CIDR은 **clusterNetwork** CIDR과 중복될 수 없습니다.

- 2 추가 오버레이 네트워크에 대한 사용자 정의 VXLAN 포트를 지정합니다. 이는 vSphere에 설치된 클러스터에서 Windows 노드를 실행해야 하며 다른 클라우드 공급자에 대해 구성해서

4. **cluster-network-03-config.yml** 파일을 저장하고 텍스트 편집기를 종료합니다.오.
5. 선택 사항: **manifests/cluster-network-03-config.yml** 파일을 백업합니다. 설치 프로그램은 클러스터를 생성할 때 **manifests/** 디렉터리를 삭제합니다.

추가 설치 구성을 완료한 후 클러스터를 생성합니다. 설치 프로세스가 완료되면 하이브리드 네트워킹이 활성화됩니다.

14.12.2. 추가 리소스

- [Windows 컨테이너 워크로드 이해](#)
- [Windows 컨테이너 워크로드 활성화](#)
- [네트워크 사용자 지정으로 AWS에 클러스터 설치](#)
- [네트워크 사용자 지정 설정을 사용하여 Azure에 클러스터 설치](#)

15장. 경로 구성

15.1. 경로 구성

15.1.1. HTTP 기반 경로 생성

경로를 사용하면 공용 URL에서 애플리케이션을 호스팅할 수 있습니다. 애플리케이션의 네트워크 보안 구성에 따라 보안 또는 비보안이 될 수 있습니다. HTTP 기반 경로는 기본 HTTP 라우팅 프로토콜을 사용하고 보안되지 않은 애플리케이션 포트에서 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 로그인했습니다.
- 포트와 포트에서 트래픽을 수신 대기하는 TCP 엔드포인트를 노출하는 웹 애플리케이션이 있습니다.

절차

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

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

2. 다음 명령을 실행하여 프로젝트에 Pod를 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
```

생성된 **Route** 리소스는 다음과 유사합니다.

생성된 보안되지 않은 경로에 대한 **YAML** 정의입니다.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: hello-openshift-hello-openshift.<Ingress_Domain> 1
  port:
```

```
targetPort: 8080
to:
  kind: Service
  name: hello-openshift
```

1 <Ingress_Domain> 은 기본 수신 도메인 이름입니다.



참고

기본 수신 도메인을 표시하려면 다음 명령을 실행합니다.

```
$ oc get ingresses.config/cluster -o jsonpath={.spec.domain}
```

15.1.2. 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
```

15.1.3. HTTP 엄격한 전송 보안 활성화

HSTS(HTTP Strict Transport Security) 정책은 보안 향상으로 호스트에서 HTTPS 트래픽만 허용합니다. 모든 HTTP 요청은 기본적으로 삭제됩니다. 이는 웹 사이트와의 안전한 상호 작용을 보장하거나 사용자의 이익을 위해 안전한 애플리케이션을 제공하는 데 유용합니다.

HSTS가 활성화되면 HSTS는 엄격한 전송 보안 헤더를 사이트의 HTTPS 응답에 추가합니다. 경로에서 **insecureEdgeTerminationPolicy** 값을 사용하여 HTTP를 HTTPS로 보내도록 경로를 재지정할 수 있습니다. 그러나 HSTS를 사용하면 클라이언트는 요청을 보내기 전에 HTTP URL의 모든 요청을 HTTPS로 변경하므로 리디렉션이 필요하지 않습니다. 클라이언트가 이를 지원할 필요는 없으며 **max-age=0**을 설정하여 비활성화할 수 있습니다.



중요

HSTS는 보안 경로(엣지 종료 또는 재암호화)에서만 작동합니다. HTTP 또는 패스스루(passthrough) 경로에서는 구성이 유효하지 않습니다.

프로세스

- 경로에서 HSTS를 사용하려면 `haproxy.router.openshift.io/hsts_header` 값을 엣지 종료에 추가하거나 경로를 다시 암호화합니다.

```
apiVersion: v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=31536000;includeSubDomains;preload
```

1 2 3

- 1 **max-age**는 유일한 필수 매개변수입니다. HSTS 정책이 적용되는 시간(초)을 측정합니다. HSTS 헤더가 있는 응답이 호스트에서 수신될 때마다 클라이언트는 **max-age**를 업데이트합니다. **max-age** 시간이 초과되면 클라이언트는 정책을 삭제합니다.
- 2 **includeSubDomains**는 선택 사항입니다. 포함되면 클라이언트에게 호스트의 모든 하위 도메인이 호스트와 동일하게 취급되도록 지시합니다.
- 3 사전 로드는 선택 사항입니다. **max-age**가 0보다 큰 경우 `haproxy.router.openshift.io/hsts_header`에 **preload**를 포함하면 외부 서비스가 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. 사전 로드를 설정하지 않으면 브라우저가 HTTPS를 통해 사이트와 상호 작용하여 헤더를 가져와야 합니다.

15.1.4. 처리량 트러블슈팅

OpenShift Container Platform을 통해 애플리케이션을 배포하면 특정 서비스 간에 대기 시간이 비정상적으로 길어지는 등 네트워크 처리량 문제가 발생하는 경우가 있습니다.

Pod 로그에 문제의 원인이 드러나지 않는 경우 다음 방법으로 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 **podip**은 Pod의 IP 주소입니다. `oc get pod <pod_name> -o wide` 명령을 실행하여 Pod의 IP 주소를 가져옵니다.

`tcpdump`는 `/tmp/dump.pcap`에 이 두 Pod 간의 모든 트래픽을 포함하는 파일을 생성합니다. 문제가 재현되기 직전에 Analyzer를 실행하고 문제 재현이 종료된 직후 Analyzer를 중지하여 파일 크기를 최소화하는 것이 좋습니다. 다음을 사용하여 노드 간에 패킷 Analyzer를 실행할 수도 있습니다(방정식에서 SDN 제거).

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap port 4789
```

- 스트리밍 처리량 및 UDP 처리량을 측정하려면 iperf와 같은 대역폭 측정 도구를 사용합니다. 먼저 Pod에서 툴을 실행한 다음 노드에서 툴을 실행하여 병목 현상이 발생하는지 확인합니다.
 - iperf 설치 및 사용에 대한 정보는 이 [Red Hat 솔루션](#) 을 참조하십시오.

15.1.5. 쿠키를 사용하여 경로 상태 유지

OpenShift Container Platform은 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 Platform에서는 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 경로에서는 쿠키를 설정할 수 없습니다. 대신, 백엔드를 결정하는 소스 IP 주소를 기반으로 숫자를 계산합니다.

백엔드가 변경되면 트래픽이 잘못된 서버로 전달되어 스티키를 줄일 수 있습니다. 소스 IP를 숨기는 로드 밸런서를 사용하는 경우 모든 연결에 대해 동일한 번호가 설정되고 트래픽이 동일한 포드로 전송됩니다.

15.1.5.1. 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
```

15.1.6. 경로 기반 라우터

경로 기반 라우터는 URL과 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 다른 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 15.1. 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: 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.

15.1.7. 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 Hat은 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 15.2. 경로 주석

변수	설명	기본값으로 사용되는 환경 변수
haproxy.router.openshift.io/balance	로드 밸런싱 알고리즘을 설정합니다. 사용 가능한 옵션은 source , roundrobin , leastconn 입니다.	경유 경로의 경우 ROUTER_TCP_BALANCE_SCHEME 입니다. 그 외에는 ROUTER_LOAD_BALANCE_ALGORITHM 을 사용하십시오.
haproxy.router.openshift.io/disable_cookies	쿠키를 사용하여 관련 연결을 추적하지 않도록 설정합니다. true 또는 TRUE 로 설정하면 들어오는 각 HTTP 요청에 사용할 백엔드 연결을 밸런스 알고리즘으로 선택합니다.	
router.openshift.io/cookie_name	이 경로에 사용할 선택적 쿠키를 지정합니다. 이름은 대문자와 소문자, 숫자, '_', '-'의 조합으로 구성해야 합니다. 기본값은 경로의 해시된 내부 키 이름입니다.	
haproxy.router.openshift.io/pod-concurrent-connections	라우터에서 백업 pod로 허용되는 최대 연결 수를 설정합니다. 참고: Pod가 여러 개 있는 경우 각각이 수의 연결이 있을 수 있습니다. 라우터가 여러 개 있고 조정이 이루어지지 않는 경우에는 각각이 횟수만큼 연결할 수 있습니다. 설정하지 않거나 0으로 설정하면 제한이 없습니다.	

변수	설명	기본값으로 사용되는 환경 변수
haproxy.router.openshift.io/ate-limit-connections	true 또는 TRUE 를 설정하면 경로당 특정 백엔드에서 고정 테이블을 통해 구현되는 속도 제한 기능이 활성화됩니다. 참고: 이 주석을 사용하면 DDoS(Distributy-of-service) 공격에 대한 기본 보호 기능을 제공합니다.	
haproxy.router.openshift.io/ate-limit-connections.concurrent-tcp	동일한 소스 IP 주소를 통해 만든 동시 TCP 연결 수를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 DDoS(Distributy-of-service) 공격에 대한 기본 보호 기능을 제공합니다.	
haproxy.router.openshift.io/ate-limit-connections.rate-http	동일한 소스 IP 주소가 있는 클라이언트에서 HTTP 요청을 수행할 수 있는 속도를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 DDoS(Distributy-of-service) 공격에 대한 기본 보호 기능을 제공합니다.	
haproxy.router.openshift.io/ate-limit-connections.rate-tcp	동일한 소스 IP 주소가 있는 클라이언트에서 TCP 연결을 수행할 수 있는 속도를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 DDoS(Distributy-of-service) 공격에 대한 기본 보호 기능을 제공합니다.	
haproxy.router.openshift.io/timeout	경로에 대한 서버 쪽 타임아웃을 설정합니다. (TimeUnits)	ROUTER_DEFAULT_SERVER_TIMEOUT
haproxy.router.openshift.io/timeout-tunnel	이 시간 제한은 터널 연결에 적용됩니다(예: 일반 텍스트, 에지, 재암호화 또는 패스스루 라우팅을 통한 WebSocket). 일반 텍스트, 에지 또는 재암호화 경로 유형을 사용하면 이 주석이 기존 시간 제한 값이 있는 시간 제한 터널로 적용됩니다. passthrough 경로 유형의 경우 주석이 설정된 기존 시간 제한 값보다 우선합니다.	ROUTER_DEFAULT_TUNNEL_TIMEOUT

변수	설명	기본값으로 사용되는 환경 변수
ingresses.config/cluster ingress.operator.openshift.io/ hard-stop-after	IngressController 또는 Ingress 구성을 설정할 수 있습니다. 이 주석은 라우터를 재배포하고, 일반 소프트웨어 중지 실행하는 데 허용되는 최대 시간을 정의하는 haproxy hard-stop-after 글로벌 옵션을 내보내도록 HA 프록시를 구성합니다.	ROUTER_HARD_STOP_AFTER
router.openshift.io/haproxy.health.check.interval	백엔드 상태 점검 간격을 설정합니다. (TimeUnits)	ROUTER_BACKEND_CHECK_INTERVAL
haproxy.router.openshift.io/ip_whitelist	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.	
haproxy.router.openshift.io/header	엣지 종단 경로 또는 재암호화 경로에 Strict-Transport-Security 헤더를 설정합니다.	
haproxy.router.openshift.io/log-send-hostname	Syslog 헤더에 hostname 필드를 설정합니다. 시스템의 호스트 이름을 사용합니다. 라우터에 대해 사이드카 또는 Syslog 기능과 같은 Ingress API 로깅 방법이 활성화된 경우 log-send-hostname 은 기본적으로 활성화됩니다.	
haproxy.router.openshift.io/rewrite-target	백엔드의 요청 재작성 경로를 설정합니다.	

변수	설명	기본값으로 사용되는 환경 변수
<code>router.openshift.io/cookie-same-site</code>	<p>쿠키를 제한하는 값을 설정합니다. 값은 다음과 같습니다.</p> <p>Lax: 방문한 사이트와 타사 사이트 간에 쿠키가 전송됩니다.</p> <p>Strict: 쿠키가 방문한 사이트로 제한됩니다.</p> <p>None: 쿠키가 방문한 사이트로 제한됩니다.</p> <p>이 값은 재암호화 및 엣지 경로에만 적용됩니다. 자세한 내용은 SameSite 쿠키 설명서를 참조하십시오.</p>	
<code>haproxy.router.openshift.io/set-forwarded-headers</code>	<p>라우터당 Forwarded 및 X-Forwarded-For HTTP 헤더를 처리하기 위한 정책을 설정합니다. 값은 다음과 같습니다.</p> <p>append: 기존 헤더를 유지하면서 헤더를 추가합니다. 이는 기본값입니다.</p> <p>replace: 헤더를 설정하고 기존 헤더를 제거합니다.</p> <p>never: 헤더를 설정하지 않고 기존 헤더를 유지합니다.</p> <p>if-none: 아직 설정되지 않은 경우 헤더를 설정합니다.</p>	<code>ROUTER_SET_FORWARDED_HEADERS</code>



참고

환경 변수는 편집할 수 없습니다.

라우터 시간 제한 변수

TimeUnits는 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.

Variable	Default	설명
ROUTER_BACKEND_CHECK_INTERVAL	5000ms	백엔드에서 후속 활성 검사 사이의 시간입니다.
ROUTER_CLIENT_FIN_TIMEOUT	1s	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.
ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	클라이언트가 데이터를 승인하거나 보내야 하는 시간입니다.
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	최대 연결 시간입니다.
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	라우터에서 경로를 지원하는 pod로의 TCP FIN 시간 초과를 제어합니다.
ROUTER_DEFAULT_SERVER_TIMEOUT	30s	서버에서 데이터를 승인하거나 보내야 하는 시간입니다.
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	TCP 또는 WebSocket 연결이 열린 상태로 유지되는 동안의 시간입니다. 이 시간 제한 기간은 HAProxy를 다시 로드할 때마다 재설정됩니다.
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	300s	<p>새 HTTP 요청이 표시될 때까지 대기할 최대 시간을 설정합니다. 이 값을 너무 낮게 설정하면 작은 keepalive 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 ROUTER_SLOWLORIS_HTTP_KEEPALIVE는 timeout http-keep-alive를 조정합니다. 기본적으로 300s로 설정되지만 HAProxy는 5s로 설정된 tcp-request inspect-delay도 대기합니다. 이 경우 전체 시간 초과는 300s + 5s입니다.</p>
ROUTER_SLOWLORIS_TIMEOUT	10s	HTTP 요청 전송에 걸리는 시간입니다.
RELOAD_INTERVAL	5s	라우터의 최소 빈도가 새 변경 사항을 다시 로드하고 승인하도록 허용합니다.

Variable	Default	설명
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	HAProxy 메트릭 수집에 대한 시간 제한입니다.

경로 설정 사용자 정의 타임아웃

```

apiVersion: 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 **ms**가 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: 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 Controller에서 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 15.3. 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

15.1.8. 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
  ...
```

15.1.9. Ingress 오브젝트를 통해 경로 생성

일부 에코시스템 구성 요소는 Ingress 리소스와 통합되지만 경로 리소스와는 통합되지 않습니다. 이러한 경우를 처리하기 위해 OpenShift Container Platform에서는 Ingress 오브젝트가 생성될 때 관리형 경로 오브젝트를 자동으로 생성합니다. 이러한 경로 오브젝트는 해당 Ingress 오브젝트가 삭제될 때 삭제됩니다.

프로세스

- OpenShift Container Platform 콘솔에서 또는 **oc create** 명령을 입력하여 Ingress 오브젝트를 정의합니다.

Ingress의 YAML 정의

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt" 1
spec:
  rules:
  - host: www.example.com
```

```

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`가 기본 경로입니다. 기본 엣지 경로를 구현하려면 TLS 인증서 세부 정보를 템플릿 파일에 정의해야 합니다.

- a. `route.openshift.io/termination` 주석에 `passthrough` 값을 지정하는 경우 `path`를 ""로 설정하고 spec에서 `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. 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
to:
  kind: Service
  name: frontend

```

15.2. 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 Azure에서 경로를 생성하는 경우 리소스 이름에 제한이 적용됩니다. 특정 용어를 사용하는 리소스를 생성할 수 없습니다. Azure에서 제한하는 용어 목록은 Azure 설명서의 [예약된 리소스 이름 오류 해결](#)을 참조하십시오.

15.2.1. 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.com을 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: 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.

15.2.2. 사용자 정의 인증서를 사용하여 엣지 경로 생성

oc create route 명령을 사용하면 엣지 TLS 종료를와 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: 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.

15.2.3. 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: 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
```

- 63자로 제한되는 개체의 이름입니다.
- termination** 필드는 **passthrough**로 설정됩니다. 이 필드는 유일한 필수 **tls** 필드입니다.
- insecureEdgeTerminationPolicy**는 선택 사항입니다. 비활성화 경우 유효한 값은 **None**, **Redirect** 또는 빈 값입니다.

대상 Pod는 끝점의 트래픽에 대한 인증서를 제공해야 합니다. 현재 이 방법은 양방향 인증이라고도 하는 클라이언트 인증서도 지원할 수 있는 유일한 방법입니다.

16장. 수신 클러스터 트래픽 구성

16.1. 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
Ingress 컨트롤러 사용	HTTPS 이외의 HTTP/HTTPS 트래픽 및 TLS 암호화 프로토콜(예: SNI 헤더가 있는 TLS)에 액세스할 수 있습니다.
로드 밸런서 서비스를 사용하여 외부 IP 자동 할당	풀에서 할당된 IP 주소를 통해 비표준 포트로의 트래픽을 허용합니다.
서비스에 외부 IP를 수동으로 할당	특정 IP 주소를 통해 비표준 포트로의 트래픽을 허용합니다.
NodePort 구성	클러스터의 모든 노드에 서비스를 노출합니다.

16.2. 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.

16.2.1. 사전 요구 사항

- 네트워크 인프라는 외부 IP 주소에 대한 트래픽을 클러스터로 라우팅해야 합니다.

16.2.2. ExternalIP 정보

클라우드 환경이 아닌 경우 OpenShift Container Platform에서는 **ExternalIP** 기능을 통해 **Service** 오브젝트 **spec.externalIPs[]** 필드에 외부 IP 주소 할당을 지원합니다. 이 필드를 설정하면 OpenShift Container Platform에서 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 위험이 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=LoadBalancer**가 설정된 **Service** 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

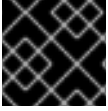
OpenShift Container Platform에서는 **Service** 오브젝트를 생성할 때 **spec.externalIPs[]** 배열에 할당된 IP 주소를 사용합니다. 다른 서비스에서 이미 사용 중인 IP 주소는 지정할 수 없습니다.

16.2.2.1. 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 할당할 때 제한된 수의 공유 IP 주소로 구성된 포트 공간을 관리하는 것보다 더 간단할 수 있습니다. 자동 할당을 사용하는 경우 **spec.type=LoadBalancer**인 **Service**에 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

할당하는 IP 주소 블록이 해당 클러스터의 노드 1개 이상에서 종료되어야 합니다.

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

```

16.2.2.2. 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 Platform의 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.

16.2.2.3. 정책 오브젝트의 예

다음 예제에서는 다양한 정책 구성을 보여줍니다.

- 다음 예에서 정책은 OpenShift Container Platform에서 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
...
```

16.2.3. ExternalIP 주소 블록 구성

ExternalIP 주소 블록에 대한 구성은 **cluster**라는 네트워크 CR(사용자 정의 리소스)에 의해 정의됩니다. 네트워크 CR은 **config.openshift.io** API 그룹의 일부입니다.



중요

CVO(Cluster Version Operator)는 클러스터를 설치하는 동안 **cluster**라는 네트워크 CR을 자동으로 생성합니다. 이 유형의 다른 CR 오브젝트는 생성할 수 없습니다.

다음 YAML에서는 ExternalIP 구성을 설명합니다.

cluster라는 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 YAML에서는 **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 YAML에서는 자동으로 할당된 외부 IP 주소를 사용하는 구성을 설명합니다.

spec.externalIP.autoAssignCIDRs가 설정된 구성의 예

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP:
    autoAssignCIDRs:
      - 192.168.132.254/29
```

- 다음 YAML에서는 허용되거나 거부된 CIDR 범위에 대한 정책 규칙을 구성합니다.

spec.externalIP.policy가 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
```

16.2.4. 클러스터에 대한 외부 IP 주소 블록 구성

클러스터 관리자는 다음 ExternalIP 설정을 구성할 수 있습니다.

- **Service** 오브젝트의 **spec.clusterIP** 필드를 자동으로 채우도록 OpenShift Container Platform에서 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.

절차

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** 스탠자에 대한 구성을 지정합니다.

4. 업데이트된 ExternalIP 구성을 확인하려면 다음 명령을 입력합니다.

```
$ oc get networks.config cluster -o go-template='{{.spec.externalIP}}{"\n"}'
```

16.2.5. 다음 단계

- [서비스 외부 IP에 대한 수신 클러스터 트래픽 구성](#)

16.3. INGRESS 컨트롤러를 사용한 수신 클러스터 트래픽 구성

OpenShift Container Platform에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법에서는 Ingress 컨트롤러를 사용합니다.

16.3.1. Ingress 컨트롤러 및 경로 사용

Ingress Operator에서는 Ingress 컨트롤러 및 와일드카드 DNS를 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, SNI를 사용하는 HTTPS, SNI를 사용하는 TLS로 제한되며, SNI를 사용하는 TLS를 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 수신 컨트롤러는 클러스터의 모든 프로젝트에서 생성된 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.

16.3.2. 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 시스템이 클러스터와 동일한 서브넷에 있다고 가정합니다. 다른 서브넷에 있는 외부 시스템에 필요한 추가 네트워킹은 이 주제에서 다루지 않습니다.

16.3.3. 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** CLI를 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.

16.3.4. 경로를 생성하여 서비스 노출

oc expose 명령을 사용하여 서비스를 경로로 노출할 수 있습니다.

프로세스

서비스를 노출하려면 다음을 수행하십시오.

1. OpenShift Container Platform 4에 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 cURL과 같은 도구를 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. cURL을 사용하여 호스트가 GET 요청에 응답하는지 확인합니다.

```
$ curl --head nodejs-ex-myproject.example.com
```

출력 예

```
HTTP/1.1 200 OK
```

```
...
```

16.3.5. 경로 라벨을 사용하여 Ingress 컨트롤러 분할 구성

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

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.

16.3.6. 네임스페이스 라벨을 사용하여 Ingress 컨트롤러 분할 구성

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

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.



주의

Keepalived Ingress VIP를 배포하는 경우 **endpointPublishingStrategy** 매개변수에 값이 **HostNetwork** 인 기본이 아닌 Ingress 컨트롤러를 배포하지 마십시오. 이렇게 하면 문제가 발생할 수 있습니다. **endpointPublishingStrategy** 에 대해 **HostNetwork** 대신 **NodePort** 값을 사용합니다.

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.

16.3.7. 추가 리소스

- Ingress Operator는 와일드카드 DNS를 관리합니다. 자세한 내용은 [OpenShift Container Platform의 Ingress Operator](#), [베어 메탈에 클러스터 설치](#), [vSphere에 클러스터 설치](#)를 참조하십시오.

16.4. 로드 밸런서를 사용하여 수신 클러스터 트래픽 구성

OpenShift Container Platform에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법에서는 로드 밸런서를 사용합니다.

16.4.1. 로드 밸런서를 사용하여 클러스터로 트래픽 가져오기

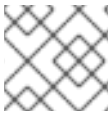
특정 외부 IP 주소가 필요하지 않은 경우 OpenShift Container Platform 클러스터에 대한 외부 액세스를 허용하도록 로드 밸런서 서비스를 구성할 수 있습니다.

로드 밸런서 서비스에서는 고유 IP를 할당합니다. 로드 밸런서에는 VIP(가상 IP)일 수 있는 단일 엣지 라우터 IP가 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.

16.4.2. 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 시스템이 클러스터와 동일한 서브넷에 있다고 가정합니다. 다른 서브넷에 있는 외부 시스템에 필요한 추가 네트워킹은 이 주제에서 다루지 않습니다.

16.4.3. 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** CLI를 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.

16.4.4. 경로를 생성하여 서비스 노출

oc expose 명령을 사용하여 서비스를 경로로 노출할 수 있습니다.

프로세스

서비스를 노출하려면 다음을 수행하십시오.

1. OpenShift Container Platform 4에 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 cURL과 같은 도구를 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. cURL을 사용하여 호스트가 GET 요청에 응답하는지 확인합니다.

```
$ curl --head nodejs-ex-myproject.example.com
```

출력 예

```
HTTP/1.1 200 OK
...
```

16.4.5. 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 4에 로그인합니다.
2. 노출하려는 서비스가 있는 프로젝트를 로드합니다.

```
$ oc project project1
```

3. 필요한 경우 컨트롤 플레인 노드 (마스터 노드라고도 함)에서 텍스트 파일을 열고 다음 텍스트를 붙여넣어 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 API로 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 cURL과 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 잘못됨**이라는 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)]>
```

16.5. 네트워크 로드 밸런서를 사용하여 AWS에서 수신 클러스터 트래픽 구성

OpenShift Container Platform에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법에서는 클라이언트의 IP 주소를 노드로 전달하는 NLB(Network Load Balancer)를 사용합니다. 신규 또는 기존 AWS 클러스터에서 NLB를 구성할 수 있습니다.

16.5.1. Ingress 컨트롤러 Classic 로드 밸런서를 네트워크 로드 밸런서로 교체

Classic Load Balancer(CLB)를 사용하는 Ingress 컨트롤러를 AWS의 NLB(Network Load Balancer)를 사용하는 컨트롤러로 교체할 수 있습니다.



주의

이 절차에서는 새 DNS 레코드 전파, 새 로드 밸런서 프로비저닝 및 기타 요인으로 인해 몇 분 정도 지속될 수 있는 예상 중단이 발생합니다. 이 절차를 적용한 후 Ingress 컨트롤러 로드 밸런서의 IP 주소 및 정식 이름이 변경될 수 있습니다.

프로세스

1. 새 기본 Ingress 컨트롤러로 파일을 생성합니다. 다음 예제에서는 default Ingress 컨트롤러에 외부 범위가 있고 다른 사용자 지정이 없는 것으로 가정합니다.

ingresscontroller.yml 파일 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

```

default Ingress 컨트롤러에 다른 사용자 지정이 있는 경우 그에 따라 파일을 수정해야 합니다.

- Ingress 컨트롤러 YAML 파일을 강제 교체합니다.

```
$ oc replace --force --wait -f ingresscontroller.yml
```

Ingress 컨트롤러가 교체될 때까지 기다립니다. 서버 운영 중단 시간 예상.

16.5.2. 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 AWS여야 합니다.
 - PlatformStatus**가 AWS인지 확인하려면 다음을 실행하십시오.

```
$ oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.type}'
AWS
```

프로세스

기존 클러스터에서 AWS NLB가 지원하는 Ingress 컨트롤러를 생성합니다.

- 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_controller**를 Ingress 컨트롤러에 대해 고유한 이름으로 교체합니다.
- 2 **\$my_unique_ingress_domain**을 클러스터의 모든 Ingress 컨트롤러 간에 고유한 도메인 이름으로 교체합니다.
- 3 내부 NLB를 사용하려면 **External**을 **Internal**로 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.

16.5.3. 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.

프로세스

새 클러스터에서 AWS NLB가 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 CR을 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/** 디렉토리를 삭제합니다.

16.5.4. 추가 리소스

- [네트워크 사용자 지정으로 AWS에 클러스터 설치](#)
- 자세한 내용은 [AWS에서 네트워크 로드 밸런서 지원](#) 을 참조하십시오.

16.6. 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 네트워크 인프라를 올바르게 구성해야 합니다.

16.6.1. 사전 요구 사항

- 클러스터는 ExternalIP가 활성화된 상태로 구성됩니다. 자세한 내용은 [서비스에 대한 ExternalIP 구성](#) 을 참조하십시오.

16.6.2. 서비스에 ExternalIP 연결

서비스에 ExternalIP를 연결할 수 있습니다. 클러스터가 ExternalIP를 자동으로 할당하도록 구성된 경우, ExternalIP를 서비스에 수동으로 연결할 필요가 없습니다.

절차

1. 선택 사항: ExternalIP와 함께 사용하도록 구성된 IP 주소 범위를 확인하려면 다음 명령을 입력합니다.

```
$ oc get networks.config cluster -o jsonpath='{.spec.externalIP}'
```

autoAssignCIDRs가 설정된 경우 **spec.externalIPs** 필드가 지정되지 않으면 OpenShift Container Platform에서 새 **Service** 오브젝트에 ExternalIP를 자동으로 할당합니다.

2. 서비스에 ExternalIP를 연결합니다.

- a. 새 서비스를 생성하는 경우 **spec.externalIPs** 필드를 지정하고 하나 이상의 유효한 IP 주소 배열을 제공합니다. 예를 들면 다음과 같습니다.

```
apiVersion: v1
kind: Service
metadata:
  name: svc-with-externalip
spec:
  ...
  externalIPs:
  - 192.174.120.10
```

- b. ExternalIP를 기존 서비스에 연결하는 경우 다음 명령을 입력합니다. **<name>**을 서비스 이름으로 교체합니다. **<ip_address>**를 유효한 ExternalIP 주소로 교체합니다. 쉼표로 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 ExternalIP를 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
```

16.6.3. 추가 리소스

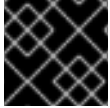
- 서비스의 ExternalIP 구성

16.7. NODEPORT를 사용하여 수신 클러스터 트래픽 구성

OpenShift Container Platform에서는 클러스터에서 실행되는 서비스와 클러스터 외부에서 통신할 수 있습니다. 이 방법에서는 **NodePort**를 사용합니다.

16.7.1. 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.

NodePort는 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 **8080**은 노드에서 포트 **31020**으로 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.

16.7.2. 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.

16.7.3. 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** CLI를 설치하고 클러스터 관리자로 로그인합니다.

프로세스

- 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.

16.7.4. 경로를 생성하여 서비스 노출

oc expose 명령을 사용하여 서비스를 경로로 노출할 수 있습니다.

프로세스

서비스를 노출하려면 다음을 수행하십시오.

1. OpenShift Container Platform 4에 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 상태에서 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 자동으로 생성된 서비스를 제거하려면 다음 명령을 입력합니다.

```
$ oc delete svc nodejs-ex
```

16.7.5. 추가 리소스

- [노드 포트 서비스 범위 구성](#)

17장. 클러스터 전체 프록시 구성

프로덕션 환경에서는 인터넷에 대한 직접 액세스를 거부하고 대신 HTTP 또는 HTTPS 프록시를 제공합니다. 기존 클러스터의 프록시 오브젝트를 수정하거나 새 클러스터의 **install-config.yaml** 파일에 프록시 설정을 구성하면 OpenShift Container Platform을 프록시를 사용하도록 구성할 수 있습니다.

17.1. 사전 요구 사항

- 클러스터에서 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.

17.2. 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 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 Platform의 프록시를 구성할 수 있습니다.



참고

cluster라는 프록시 오브젝트만 지원되며 추가 프록시는 생성할 수 없습니다.

사전 요구 사항

- 클러스터 관리자 권한
- OpenShift Container Platform **oc** CLI 도구 설치

프로세스

1. HTTPS 연결 프록시에 필요한 추가 CA 인증서가 포함된 ConfigMap을 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** Proxy 오브젝트에서 참조할 ConfigMap 이름입니다.
- 4** ConfigMap은 **openshift-config** 네임스페이스에 있어야 합니다.

- b. 이 파일에서 ConfigMap을 생성합니다.

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

2. **oc edit** 명령을 사용하여 프록시 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: http://<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 URL입니다. URL 스키마는 **http**여야 합니다.
- 2 클러스터 외부에서 HTTPS 연결을 구축하는 데 사용할 프록시 URL입니다.
- 3 대상 도메인 이름, 도메인, IP 주소 또는 프록시를 제외할 기타 네트워크 CIDR로 이루어진 집합으로 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.

httpProxy와 **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 URL입니다.
- 5 HTTPS 연결을 프록시하는 데 필요한 추가 CA 인증서가 포함된 **openshift-config** 네임스페이스의 ConfigMap에 대한 참조입니다. 여기서 ConfigMap을 참조하기 전에 ConfigMap이 이미 존재해야 합니다. 프록시의 ID 인증서를 RHCOS 트러스트 번들에 있는 기관에서 서명하지 않은 경우 이 필드가 있어야 합니다.

4. 파일을 저장하여 변경 사항을 적용합니다.

17.3. 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: {}
status: {}
```

3. 파일을 저장하여 변경 사항을 적용합니다.

추가 리소스

- CA 번들 인증서 교체
- 프록시 인증서 사용자 정의

18장. 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 구성 합니다. **install-config.yaml** 파일의 **additionalTrustBundle** 설정에 개인 서명 CA 인증서를 정의해야 합니다. 설치 프로그램에서 사용자가 정의한 추가 CA 인증서가 포함된 **user-ca-bundle**이라는 ConfigMap을 생성합니다. 그러면 CNO(Cluster Network Operator)에서 이러한 CA 인증서를 RHCOS(Red Hat Enterprise Linux CoreOS) 신뢰 번들과 병합하는 **trusted-ca-bundle** ConfigMap을 생성합니다. 이 ConfigMap은 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 ConfigMap을 생성한 다음 개인 서명 인증서의 ConfigMap을 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.

18.1. 설치 중 클러스터 전체 프록시 구성

프로덕션 환경에서는 인터넷에 대한 직접 액세스를 거부하고 대신 HTTP 또는 HTTPS 프록시를 제공합니다. **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.

절차

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: example.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 쉼표로 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.
- 4 이 값을 제공하면 설치 프로그램에서 추가 CA 인증서를 보유할 **openshift-config** 네임스페이스에 **user-ca-bundle**이라는 구성 맵을 생성합니다. **additionalTrustBundle** 및 하나 이상의 프록시 설정을 제공하는 경우 프록시 오브젝트는 **trustedCA** 필드의 **user-ca-bundle** 구성 맵을 참조하도록 구성됩니다. 그러면 Cluster Network Operator에서 **trustedCA** 매개 변수에 대해 지정된 콘텐츠를 RHCOS 신뢰 번들과 병합하는 **trusted-ca-bundle** 구성 맵을 생성합니다. 프록시의 ID 인증서를 RHCOS 트러스트 번들에 있는 기관에서 서명하지 않은 경우 **additionalTrustBundle** 필드가 있어야 합니다.



참고

설치 프로그램에서 프록시 **adinessEndpoints** 필드를 지원하지 않습니다.

2. 파일을 저장해 놓고 OpenShift Container Platform을 설치할 때 참조하십시오.

제공되는 **install-config.yaml** 파일의 프록시 설정을 사용하는 **cluster**라는 이름의 클러스터 전체 프록시가 설치 프로그램에 의해 생성됩니다. 프록시 설정을 제공하지 않아도 **cluster Proxy** 오브젝트는 계속 생성되지만 **spec**은 **nil**이 됩니다.



참고

cluster라는 **Proxy** 오브젝트만 지원되며 추가 프록시는 생성할 수 없습니다.

18.2. 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

cluster라는 프록시 오브젝트만 지원되며 추가 프록시는 생성할 수 없습니다.

사전 요구 사항

- 클러스터 관리자 권한
- OpenShift Container Platform **oc** CLI 도구 설치

프로세스

1. HTTPS 연결 프록시에 필요한 추가 CA 인증서가 포함된 ConfigMap을 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** Proxy 오브젝트에서 참조할 ConfigMap 이름입니다.
- 4** ConfigMap은 **openshift-config** 네임스페이스에 있어야 합니다.

- b. 이 파일에서 ConfigMap을 생성합니다.


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

2. **oc edit** 명령을 사용하여 프록시 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: http://<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 URL입니다. URL 스키마는 **http**여야 합니다.
- 2 클러스터 외부에서 HTTPS 연결을 구축하는 데 사용할 프록시 URL입니다.
- 3 대상 도메인 이름, 도메인, IP 주소 또는 프록시를 제외할 기타 네트워크 CIDR로 이루어진 쉼표로 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.

httpProxy와 **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 URL입니다.
- 5 HTTPS 연결을 프록시하는 데 필요한 추가 CA 인증서가 포함된 **openshift-config** 네임스페이스의 ConfigMap에 대한 참조입니다. 여기서 ConfigMap을 참조하기 전에 ConfigMap이 이미 존재해야 합니다. 프록시의 ID 인증서를 RHCOS 트러스트 번들에 있는 기관에서 서명하지 않은 경우 이 필드가 있어야 합니다.

4. 파일을 저장하여 변경 사항을 적용합니다.

18.3. OPERATOR를 사용한 인증서 주입

ConfigMap을 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.

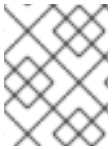
```
config.openshift.io/inject-trusted-cabundle="true"
```

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 들면 다음과 같습니다.

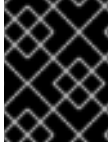
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
  spec:
    ...
    containers:
      - name: my-container-that-needs-custom-ca
        volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            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.

19장. RHOSP의 로드 밸런싱

19.1. 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 확인합니다.

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 OVN으로 변경하려면 다음을 수행하십시오.

1. **kuryr-config** ConfigMap을 엽니다. 명령줄에 다음을 입력합니다.

```
$ oc -n openshift-kuryr edit cm kuryr-config
```

2. ConfigMap에서 **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** Pod를 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. RHOSP에서 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
```

19.2. 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 Kuryr를 사용하는 경우 CNO(Cluster Network Operator)가 배포 시 내부 Octavia 로드 밸런서를 생성했습니다. 이 로드 밸런서를 애플리케이션 네트워크 스케일링에 사용할 수 있습니다.

클러스터가 Kuryr를 사용하지 않는 경우 애플리케이션 네트워크 확장에 사용할 자체 Octavia 로드 밸런서를 생성해야 합니다.

19.2.1. Octavia를 사용하여 클러스터 스케일링

여러 API 로드 밸런서를 사용하거나 클러스터가 Kuryr를 사용하지 않는 경우 Octavia 로드 밸런서를 생성하고 이를 사용할 클러스터를 구성합니다.

사전 요구 사항

- Octavia는 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.

19.2.2. 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.
- Octavia는 RHOSP(Red Hat OpenStack Platform) 배포에서 사용할 수 있습니다.

절차

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 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.

19.3. RHOSP OCTAVIA를 사용하여 수신 트래픽 스케일링

Octavia 로드 밸런서를 사용하여 Kuryr를 사용하는 클러스터에서 Ingress 컨트롤러를 스케일링할 수 있습니다.

사전 요구 사항

- OpenShift Container Platform 클러스터는 Kuryr을 사용합니다.
- RHOSP 배포에서 Octavia를 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
```

- 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.

- 명령줄에서 **external_router.yaml** 파일의 서비스를 생성합니다.

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

- 서비스의 외부 IP 주소가 로드 밸런서와 연결된 항목과 동일한지 확인합니다.
 - 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 |
```

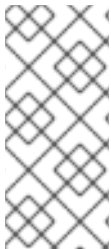
- c. 이전 단계에서 검색한 주소가 유동 IP 목록에서 서로 연결되어 있는지 확인합니다.

```
$ 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

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.

20장. 보조 인터페이스 지표와 네트워크 연결 연관 짓기

20.1. 보조 인터페이스 지표와 네트워크 연결 연관 짓기

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. 이 방법은 추가 인터페이스가 없을 때는 사용 가능하지만 보조 인터페이스를 추가하는 경우 인터페이스 이름만으로 인터페이스를 구분하기 힘들기 때문에 지표를 사용하기 어렵습니다.

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 확인하는 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 다른 클래스를 구별하는 데 사용됩니다. 예를 들어 서로 다른 네트워크에 속하거나 서로 다른 CNI를 사용하는 서로 다른 인터페이스는 서로 다른 네트워크 연결 정의 이름을 사용합니다.

20.1.1. 네트워크 지표 데몬

네트워크 지표 데몬은 네트워크 관련 지표를 수집하고 게시하는 데몬 구성 요소입니다.

kubelet은 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.

- 포트 이름
- 포트 네임스페이스
- 인터페이스 이름(예: `eth0`)

이러한 지표는 예를 들면 `Multus`를 통해 Pod에 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.

20.1.2. 네트워크 이름이 있는 지표

이 daemonset는 고정 값이 0인 `pod_network_name_info` 게이지 지표를 게시합니다.

```
pod_network_name_info{interface="net0",namespace="namespacename",network_name="nadspace/firstNAD",pod="podname"} 0
```

네트워크 이름 레이블은 Multus에서 추가한 주석을 사용하여 생성됩니다. 네트워크 연결 정의가 속하는 네임스페이스와 네트워크 연결 정의 이름이 연결된 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)
```