OpenShift Dedicated 4

네트워킹

OpenShift Dedicated 네트워킹 구성

Last Updated: 2024-09-10

OpenShift Dedicated 4 네트워킹

OpenShift Dedicated 네트워킹 구성

법적 공지

초록

이 문서에서는 OpenShift Dedicated 클러스터의 네트워킹에 대한 정보를 제공합니다.

차례

1장. 네트워킹 정보

Red Hat OpenShift Networking은 클러스터가 하나 이상의 하이브리드 클러스터의 네트워크 트래픽을 관리하는 데 필요한 고급 네트워킹 관련 기능을 사용하여 Kubernetes 네트워킹을 개선하는 기능, 플러그인 및 고급 네트워킹 기능으로 구성된 에코시스템입니다. 이 네트워킹 기능의 에코시스템은 수신, 송신, 로드 밸런싱, 고성능 처리량, 보안 및 클러스터 내부 트래픽 관리를 통합합니다. Red Hat OpenShift Networking 에코시스템은 또한 역할 기반 관찰 툴을 제공하여 자연의 복잡성을 줄일 수 있습니다.

다음은 클러스터에서 사용할 수 있는 가장 일반적으로 사용되는 Red Hat OpenShift Networking 기능 중 일부입니다.

- 다음 CNI(Container Network Interface) 플러그인에서 제공하는 기본 클러스터 네트워크입니다.
  - OVN-Kubernetes 네트워크 플러그인 - 기본 플러그인
  - OpenShift SDN 네트워크 플러그인 - OpenShift 4.14부터는 클러스터에 더 이상 사용되지 않음
- 네트워크 플러그인 관리를 위한 Cluster Network Operator

OpenShift 4.11 이상으로 생성된 OpenShift Dedicated 클러스터는 기본적으로 OVN-Kubernetes 네트워크 플러그인을 사용합니다. OpenShift 버전 4.11 이전에 생성된 OpenShift Dedicated 클러스터는 OpenShift SDN 플러그인을 OpenShift 버전 4.11 이상으로 업그레이드한 후 사용합니다.

중요

OpenShift Dedicated는 OpenShift Core Platform에 따른 SDN의 라이프 사이클을 따릅니다.

- OpenShift 버전 4.14부터는 SDN이 클러스터에서 더 이상 사용되지 않습니다.
- OpenShift SDN 플러그인을 이미 사용하는 클러스터는 OpenShift 버전 4.11 이상으로 업그레이드된 후에도 플러그인을 계속 사용합니다.
- 클러스터는 OpenShift 버전 4.16으로 업그레이드할 수 있습니다.
- OpenShift 4.16에서 실행되는 클러스터:
  - OpenShift 버전 4.16을 사용하는 클러스터는 클러스터가 SDN 플러그인을 사용하는 경우 업그레이드할 수 없습니다.
- SDN 플러그인은 OpenShift 버전 4.17에서 중단됩니다.

OpenShift 버전 4.15 이상에서 실행되는 클러스터의 경우 OpenShift SDN에서 OVN으로 마이그레이션할 수 있습니다. 이 마이그레이션 툴은 현재 사용할 수 없습니다. OpenShift SDN 사용 중단 및 OVN 마이그레이션에 대한 자세한 내용은 OCP 4.17의 OpenShift SDN CNI 제거 관련 KCS 문서를 참조하십시오.

2장. OPENSHIFT DEDICATED의 DNS OPERATOR

OpenShift Dedicated에서 DNS Operator는 CoreDNS 인스턴스를 배포 및 관리하여 클러스터 내부 Pod에 이름 확인 서비스를 제공하고 DNS 기반 Kubernetes 서비스 검색을 활성화하고 내부 **cluster.local** 이름을 확인합니다.

2.1. DNS OPERATOR의 상태 확인

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  MESSAGE
dns   4.1.15-0.11  True       False        False     92m
```

AVAILABLE,**PROGRESSING** 및 **DEGRADED** 는 Operator 상태에 대한 정보를 제공합니다. **AVAILABLE** 은 CoreDNS 데몬 세트에서 1개 이상의 Pod가 **Available** 상태 조건을 보고하고 DNS 서비스에 클러스터 IP 주소가 있는 경우 **True** 입니다.

2.2. 기본 DNS보기

모든 새로운 OpenShift Dedicated 설치에는 이름이 **default** 인 **dns.operator** 가 있습니다.

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)
...
```

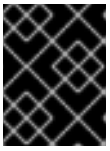
(1) Cluster Domain 필드는 정규화된 pod 및 service 도메인 이름을 구성하는 데 사용되는 기본 DNS 도메인입니다.

(2) Cluster IP는 이름을 확인하기 위한 주소 Pod 쿼리입니다. IP는 service CIDR 범위에서 10번째 주소로 정의됩니다.

2.3. DNS 전달 사용

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

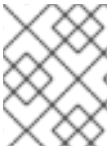
- 모든 영역에 대해 이름 서버(**spec.servers**)를 지정합니다. 전달된 영역이 OpenShift Dedicated에서 관리하는 인그레스 도메인인 경우 도메인에 대해 업스트림 이름 서버를 인증해야 합니다.



중요

하나 이상의 영역을 지정해야 합니다. 그렇지 않으면 클러스터가 기능을 손실할 수 있습니다.

- 업스트림 DNS 서버 목록 제공(**spec.upstreamResolvers**).
- 기본 전달 정책을 변경합니다.



참고

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

절차

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

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

이전 명령을 실행한 후 Operator는 **spec.servers** 를 기반으로 추가 서버 구성 블록을 사용하여 **dns-default** 라는 구성 맵을 생성하고 업데이트합니다.

중요

zones 매개 변수의 값을 지정할 때 인트라넷과 같은 특정 영역으로만 전달해야 합니다. 하나 이상의 영역을 지정해야 합니다. 그렇지 않으면 클러스터가 기능을 손실할 수 있습니다.

서버에 쿼리와 일치하는 영역이 없는 경우 이름 확인은 업스트림 DNS 서버로 대체됩니다.

DNS 전달 구성

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    negativeTTL: 0s
    positiveTTL: 0s
  logLevel: Normal
  nodePlacement: {}
  operatorLogLevel: Normal
  servers:
  - name: example-server (1)
    zones:
    - example.com (2)
    forwardPlugin:
      policy: Random (3)
      upstreams: (4)
      - 1.1.1.1
      - 2.2.2.2:5353
  upstreamResolvers: (5)
    policy: Random (6)
    protocolStrategy: "" (7)
    transportConfig: {} (8)
    upstreams:
    - type: SystemResolvConf (9)
    - type: Network
      address: 1.2.3.4 (10)
      port: 53 (11)
    status:
      clusterDomain: cluster.local
      clusterIP: x.y.z.10
      conditions:
    ...
```

(1) **rfc6335** 서비스 이름 구문을 준수해야 합니다.

(2) **rfc1123** 서비스 이름 구문의 하위 도메인 정의를 준수해야 합니다. 클러스터 도메인인 **cluster.local** 은 **zones** 필드에 대해 유효하지 않은 하위 도메인입니다.

(3) **forwardPlugin** 에 나열된 업스트림 리졸버를 선택하는 정책을 정의합니다. 기본값은 **Random** 입니다. **RoundRobin** 및 **Sequential** 값을 사용할 수도 있습니다.

(4) **forwardPlugin**당 최대 15개의 **업스트림**이 허용됩니다.

(5) **upstreamResolvers** 를 사용하여 기본 전달 정책을 재정의하고 기본 도메인의 지정된 DNS 확인자(업스트림 확인자)로 DNS 확인을 전달할 수 있습니다. 업스트림 확인자를 제공하지 않으면 DNS 이름 쿼리는 **/etc/resolv.conf** 에 선언된 서버로 이동합니다.

(6) 쿼리용으로 업스트림에 나열된 **업스트림** 서버의 순서를 결정합니다. 이러한 값 중 하나를 지정할 수 있습니다. **Random**,**RoundRobin** 또는 **Sequential**. 기본값은 **Sequential** 입니다.

7 생략하면 플랫폼은 원래 클라이언트 요청의 프로토콜인 기본값을 선택합니다. 클라이언트 요청이 UDP를 사용하는 경우에도 플랫폼에서 모든 업스트림 DNS 요청에 **TCP** 를 사용하도록 지정하려면 TCP로 설정합니다.

8 DNS 요청을 업스트림 확인기로 전달할 때 사용할 전송 유형, 서버 이름 및 선택적 사용자 정의 CA 또는 CA 번들을 구성하는 데 사용됩니다.

9 두 가지 유형의 **업스트림** 을 지정할 수 있습니다 : **SystemResolvConf** 또는 **Network**. **SystemResolvConf** 는 **/etc/resolv.conf** 를 사용하도록 업스트림을 구성하고 **Network** 는 **Networkresolver** 를 정의합니다. 하나 또는 둘 다를 지정할 수 있습니다.

10 지정된 유형이 **Network** 인 경우 IP 주소를 제공해야 합니다. **address** 필드는 유효한 IPv4 또는 IPv6 주소여야 합니다.

11 지정된 유형이 **네트워크** 인 경우 선택적으로 포트를 제공할 수 있습니다. **port** 필드에는 **1** 에서 **65535** 사이의 값이 있어야 합니다. 업스트림에 대한 포트를 지정하지 않으면 기본 포트는 853입니다.

추가 리소스

- DNS 전달에 대한 자세한 내용은 CoreDNS 전달 설명서를 참조하십시오.

2.4. DNS OPERATOR 상태 확인

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

절차

- DNS Operator의 상태를 확인하려면 다음을 실행합니다.

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

메시지와 철자는 특정 릴리스에서 다를 수 있지만 예상되는 상태 출력은 다음과 같습니다.

```
Status:
  Conditions:
    Last Transition Time:  <date>
    Message:               DNS "default" is available.
    Reason:                AsExpected
    Status:                True
    Type:                  Available
    Last Transition Time:  <date>
    Message:               Desired and current number of DNSes are equal
    Reason:                AsExpected
    Status:                False
    Type:                  Progressing
    Last Transition Time:  <date>
    Reason:                DNSNotDegraded
    Status:                False
    Type:                  Degraded
    Last Transition Time:  <date>
    Message:               DNS default is upgradeable: DNS Operator can be upgraded
```

```
Reason:              DNSUpgradeable
Status:              True
Type:                Upgradeable
```

2.5. DNS OPERATOR 로그 보기

oc logs 명령을 사용하여 DNS Operator 로그를 확인할 수 있습니다.

절차

- DNS Operator의 로그를 확인합니다.

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

2.6. COREDNS 로그 수준 설정

CoreDNS 및 CoreDNS Operator의 로그 수준은 다른 방법을 사용하여 설정됩니다. CoreDNS 로그 수준을 구성하여 기록된 오류 메시지의 세부 정보를 확인할 수 있습니다. CoreDNS 로그 수준에 유효한 값은 **Normal**,**Debug** 및 **Trace** 입니다. 기본 **logLevel** 은 **Normal** 입니다.

참고

CoreDNS 오류 로그 수준은 항상 활성화됩니다. 다음 로그 수준 설정은 다른 오류 응답을 보고합니다.

- **loglevel**:**Normal** 은 "errors" 클래스를 활성화합니다. **log . { class error }**.
- **loglevel**:**Debug** 는 "denial" 클래스를 활성화합니다. **log . { class denial error}** } .
- **loglevel**:**Trace** 는 "all" 클래스를 활성화합니다. **log . { class all }**.

절차

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

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

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

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

검증

- 원하는 로그 수준이 설정되었는지 확인하려면 구성 맵을 확인합니다.

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

예를 들어 **logLevel** 을 **Trace** 로 설정한 후 각 server 블록에 이 스탠자가 표시되어야 합니다.

```
errors
log . {
    class all
```

```
}
```

2.7. COREDNS OPERATOR 로그 수준 설정

CoreDNS 및 CoreDNS Operator의 로그 수준은 다른 방법을 사용하여 설정됩니다. 클러스터 관리자는 OpenShift DNS 문제를 더 신속하게 추적하도록 Operator 로그 수준을 구성할 수 있습니다. **operatorLogLevel** 에 유효한 값은 **Normal**,**Debug** 및 **Trace** 입니다. **추적** 에는 가장 자세한 정보가 있습니다. 기본 **operatorlogLevel** 은 **Normal** 입니다. Operator 문제의 로깅 수준은 추적, 디버그, 정보, 경고, 오류, Fatal 및 Panic입니다. 로깅 수준이 설정되면 해당 심각도가 있는 로그 항목 또는 그 이상의 로그 항목이 기록됩니다.

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

절차

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

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

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

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

검증

1. 결과 변경 사항을 검토하려면 다음 명령을 입력합니다.

```
$ oc get dnses.operator -A -oyaml
```

두 개의 로그 수준 항목이 표시되어야 합니다. **operatorLogLevel** 은 OpenShift DNS Operator 문제에 적용되며 **logLevel** 은 CoreDNS Pod의 데몬 세트에 적용됩니다.

```
logLevel: Trace
operatorLogLevel: Debug
```

2. 데몬 세트의 로그를 검토하려면 다음 명령을 입력합니다.

```
$ oc logs -n openshift-dns ds/dns-default
```

2.8. COREDNS 캐시 튜닝

CoreDNS의 경우 각각 양수 또는 음수 캐싱이라고도 하는 성공 또는 실패한 캐싱의 최대 기간을 구성할 수 있습니다. DNS 쿼리 응답의 캐시 기간을 조정하면 업스트림 DNS 확인자의 부하를 줄일 수 있습니다.

주의

TTL 필드를 낮은 값으로 설정하면 클러스터의 부하, 업스트림 확인자 또는 둘 다 증가할 수 있습니다.

절차

1. 다음 명령을 실행하여 **default** 라는 DNS Operator 오브젝트를 편집합니다.

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

2. TTL(Time-to-live) 캐싱 값을 수정합니다.

DNS 캐싱 구성

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  cache:
    positiveTTL: 1h 1
    negativeTTL: 0.5h10m 2
```

1. 문자열 값 **1h** 는 CoreDNS에 의해 해당 시간(초)으로 변환됩니다. 이 필드를 생략하면 값이 **0s** 로 간주되고 클러스터는 내부 기본값 **900s** 를 폴백으로 사용합니다.
2. 문자열 값은 **0.5h10m** 과 같은 단위의 조합일 수 있으며 CoreDNS를 통해 해당 시간(초)으로 변환됩니다. 이 필드를 생략하면 값이 **0s** 로 간주되고 클러스터는 내부 기본값 **30s** 를 폴백으로 사용합니다.

검증

1. 변경 사항을 검토하려면 다음 명령을 실행하여 구성 맵을 다시 확인합니다.

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

2. 다음 예와 같은 항목이 표시되는지 확인합니다.

```
cache 3600 {
    denial 9984 2400
}
```

추가 리소스

캐싱에 대한 자세한 내용은 CoreDNS 캐시 를 참조하십시오.

2.9. 고급 작업

2.9.1. DNS Operator managementState 변경

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

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

- 사용자가 개발자이며 구성 변경을 테스트하여 CoreDNS의 문제가 해결되었는지 확인하려고 합니다. **managementState** 를 **Unmanaged** 로 설정하여 DNS Operator가 구성 변경 사항을 덮어쓰지 않도록 할 수 있습니다.
- 클러스터 관리자이며 CoreDNS 관련 문제를 보고했지만 문제가 해결될 때까지 해결 방법을 적용해야 합니다. DNS Operator의 **managementState** 필드를 **Unmanaged**로 설정하여 해결 방법을 적용할 수 있습니다.

절차

1. DNS Operator에서 **managementState** 를 **Unmanaged** 로 변경합니다.

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

2. **jsonpath** 명령줄 JSON 구문 분석을 사용하여 DNS Operator의 **managementState** 를 검토합니다.

```
$ oc get dns.operator.openshift.io default -ojsonpath='{.spec.managementState}'
```

출력 예

```
"Unmanaged"
```

참고

managementState 가 **Unmanaged** 로 설정된 동안은 업그레이드할 수 없습니다.

2.9.2. DNS Pod 배치 제어

DNS Operator에는 **dns-default** 라는 두 개의 데몬 세트가 있으며 하나는 **node-resolver** 라는 **/etc/hosts** 파일을 관리하는 데 사용됩니다.

일반적인 작업은 아니지만 CoreDNS Pod가 할당 및 실행 중인 노드를 제어해야 할 수도 있습니다. 예를 들어 클러스터 관리자가 노드 쌍 간 통신을 금지할 수 있는 보안 정책을 구성한 경우 CoreDNS의 데몬 세트가 실행되는 노드 세트를 제한해야 합니다. 클러스터의 일부 노드에서 DNS pod가 실행 중이고 DNS pod가 실행되고 있지 않은 노드에 DNS pod가 실행 중인 노드에 대한 네트워크 연결이 있는 경우 모든 Pod에서 DNS 서비스를 사용할 수 있습니다.

node-resolver 데몬 세트는 이미지 가져오기를 지원하는 클러스터 이미지 레지스트리의 항목을 추가하므로 모든 노드 호스트에서 실행해야 합니다. **node-resolver** Pod에는 하나의 작업만 있습니다. **image-registry.openshift-image-registry.svc** 서비스의 클러스터 IP 주소를 조회하고 컨테이너 런타임에서 서

비스 이름을 확인할 수 있도록 노드 호스트의 **/etc/hosts** 에 추가합니다.

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

사전 요구 사항

- **oc** CLI를 설치했습니다.
- **cluster-admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- DNS Operator **managementState** 는 **Managed** 로 설정됩니다.

절차

- CoreDNS의 데몬 세트가 특정 노드에서 실행되도록 테인트 및 허용 오차를 구성합니다.
  1. 이름이 **default**인 DNS Operator 오브젝트를 수정합니다.

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

  2. 테인트 키와 테인트에 대한 허용 오차를 지정합니다.

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

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

2.9.3. TLS를 사용하여 DNS 전달 구성

고도로 규제된 환경에서 작업하는 경우 추가 DNS 트래픽 및 데이터 개인 정보를 보장할 수 있도록 요청을 업스트림 해석기로 전달할 때 DNS 트래픽을 보호할 수 있는 기능이 필요할 수 있습니다.

CoreDNS 캐시가 10초 동안 전달된 연결을 확인합니다. CoreDNS는 요청이 발행되지 않은 경우 해당 10초 동안 TCP 연결을 열린 상태로 유지합니다. 대규모 클러스터에서는 노드당 연결을 시작할 수 있으므로 DNS 서버에서 많은 새 연결을 유지할 수 있음을 알고 있는지 확인합니다. 성능 문제를 방지하기 위해 DNS 계층 구조를 적절하게 설정합니다.

중요

zones 매개 변수의 값을 지정할 때 인트라넷과 같은 특정 영역으로만 전달해야 합니다. 하나 이상의 영역을 지정해야 합니다. 그렇지 않으면 클러스터가 기능을 손실할 수 있습니다.

절차

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

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

클러스터 관리자는 전달된 DNS 쿼리에 대해 TLS(전송 계층 보안)를 구성할 수 있습니다.

TLS를 사용하여 DNS 전달 구성

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

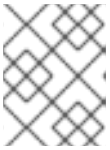
1. **rfc6335** 서비스 이름 구문을 준수해야 합니다.
2. **rfc1123** 서비스 이름 구문의 하위 도메인 정의를 준수해야 합니다. 클러스터 도메인인 **cluster.local** 은 **zones** 필드에 대해 유효하지 않은 하위 도메인입니다. 클러스터 도메인에 해당하는 **cluster.local**은 **영역**에 유효하지 않은 **하위 도메인**입니다.
3. 전달된 DNS 쿼리에 대해 TLS를 구성할 때 값 **TLS** 를 갖도록 **transport** 필드를 설정합니다.
4. 전달된 DNS 쿼리에 대해 TLS를 구성할 때 이는 업스트림 TLS 서버 인증서의 유효성을 확인하기 위해 SNI(서버 이름 표시)의 일부로 사용되는 필수 서버 이름입니다.
5. 업스트림 리졸버를 선택하는 정책을 정의합니다. 기본값은 **Random** 입니다. **RoundRobin** 및 **Sequential** 값을 사용할 수도 있습니다.
6. 필수 항목입니다. 이를 사용하여 업스트림 리졸버를 제공합니다. **forwardPlugin** 항목당 최대 15 **개의 업스트림** 항목이 허용됩니다.

7 선택 사항: 이를 사용하여 기본 정책을 재정의하고 기본 도메인의 지정된 DNS 확인자(업스트림 확인자)로 DNS 확인을 전달할 수 있습니다. 업스트림 확인자를 제공하지 않으면 DNS 이름 쿼리는 **/etc/resolv.conf** 의 서버로 이동합니다.

8 TLS를 사용할 때 **네트워크** 유형만 허용되며 IP 주소를 제공해야 합니다. **네트워크** 유형은 이 업스트림 리졸버가 **/etc/resolv.conf** 에 나열된 업스트림 해석기와 별도로 전달된 요청을 처리해야 함을 나타냅니다.

9 **address** 필드는 유효한 IPv4 또는 IPv6 주소여야 합니다.

10 선택적으로 포트를 제공할 수 있습니다. **포트** 는 **1** 에서 **65535** 사이의 값이 있어야 합니다. 업스트림에 대한 포트를 지정하지 않으면 기본 포트는 853입니다.



참고

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

검증

1. 구성 맵을 표시합니다.

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

TLS 전달을 기반으로 하는 샘플 DNS ConfigMap 예

```
apiVersion: v1
data:
  Corefile: |
    example.com:5353 {
        forward . 1.1.1.1 2.2.2.2:5353
    }
    bar.com:5353 example.com:5353 {
        forward . 3.3.3.3 4.4.4.4:5454 1
    }
    .:5353 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
            pods insecure
            upstream
            fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        forward . /etc/resolv.conf 1.2.3.4:53 {
            policy Random
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
```

```
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

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

추가 리소스

- DNS 전달에 대한 자세한 내용은 CoreDNS 전달 설명서를 참조하십시오.

3장. OPENSHIFT DEDICATED의 INGRESS OPERATOR

3.1. OPENSHIFT DEDICATED INGRESS OPERATOR

OpenShift Dedicated 클러스터를 생성할 때 클러스터에서 실행 중인 Pod 및 서비스에 각각 고유한 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 Dedicated 클러스터 서비스에 대한 외부 액세스를 활성화하는 구성 요소입니다.

Ingress Operator를 사용하면 라우팅을 처리하기 위해 하나 이상의 HAProxy 기반 Ingress 컨트롤러를 배포하고 관리하여 외부 클라이언트가 서비스에 액세스할 수 있습니다. Red Hat 사이트 안정성 엔지니어(SRE)는 OpenShift Dedicated 클러스터용 Ingress Operator를 관리합니다. Ingress Operator의 설정을 변경할 수는 없지만 기본 Ingress 컨트롤러 구성, 상태 및 로그와 Ingress Operator 상태를 볼 수 있습니다.

3.2. 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.openshiftdemos.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.

3.3. INGRESS 컨트롤러 구성 매개변수

Infrastructure CR(사용자 정의 리소스)에는 조직에 대한 특정 요구 사항을 충족하도록 구성할 수 있는 선택적 구성 매개변수가 포함되어 있습니다.

매개변수	설명

매개변수	설명
domain	**domain**은 Ingress 컨트롤러에서 제공하는 DNS 이름이며 여러 기능을 구성하는 데 사용됩니다. ● **LoadBalancerService** 끝점 게시 방식에서는 **domain**을 사용하여 DNS 레코드를 구성합니다. **endpointPublishingStrategy**를 참조하십시오. ● 생성된 기본 인증서를 사용하는 경우, 인증서는 **domain** 및 해당 **subdomains**에 유효합니다. **defaultCertificate**를 참조하십시오. ● 사용자가 외부 DNS 레코드의 대상 위치를 확인할 수 있도록 이 값이 개별 경로 상태에 게시됩니다. **domain** 값은 모든 Ingress 컨트롤러에서 고유해야 하며 업데이트할 수 없습니다. 비어 있는 경우 기본값은 **ingress.config.openshift.io/cluster .spec.domain**입니다.
replicas	**replicas** 는 Ingress 컨트롤러 복제본 수입니다. 설정되지 않은 경우, 기본값은 **2**입니다.

매개변수	설명
endpointPublishingStrategy	**endpointPublishingStrategy**는 Ingress 컨트롤러 끝점을 다른 네트워크에 게시하고 로드 밸런서 통합을 활성화하며 다른 시스템에 대한 액세스를 제공하는 데 사용됩니다. 클라우드 환경의 경우 **loadBalancer** 필드를 사용하여 Ingress 컨트롤러에 대한 끝점 게시 전략을 구성합니다. 다음 **endpointPublishingStrategy** 필드를 구성할 수 있습니다. ● **loadBalancer.scope** ● **loadBalancer.allowedSourceRanges** 설정되지 않은 경우, 기본값은 **infrastructure.config.openshift.io/cluster .status.platform**을 기반으로 다음과 같습니다. ● AWS(Amazon Web Services): **LoadBalancerService** (외부 범위 포함) ● GCP(Google Cloud Platform): **LoadBalancerService** (외부 범위 포함) 대부분의 플랫폼의 경우 **endpointPublishingStrategy** 값을 업데이트할 수 있습니다. GCP에서 다음 **endpointPublishingStrategy** 필드를 구성할 수 있습니다. ● **loadBalancer.scope** ● **loadbalancer.providerParameters.gcp.clientAccess** 클러스터가 배포된 후 **endpointPublishingStrategy** 값을 업데이트해야 하는 경우 다음 **endpointPublishingStrategy** 필드를 구성할 수 있습니다. ● **hostNetwork.protocol** ● **nodePort.protocol** ● **private.protocol**
defaultCertificate	**defaultCertificate** 값은 Ingress 컨트롤러가 제공하는 기본 인증서가 포함된 보안에 대한 참조입니다. 경로가 고유한 인증서를 지정하지 않으면 **defaultCertificate**가 사용됩니다. 보안에는 키와 데이터, 즉 ***tls.crt**: 인증서 파일 내용 ***tls.key**: 키 파일 내용이 포함되어야 합니다. 설정하지 않으면 와일드카드 인증서가 자동으로 생성되어 사용됩니다. 인증서는 Ingress 컨트롤러 **도메인** 및 **하위 도메인**에 유효하며 생성된 인증서의 CA는 클러스터의 신뢰 저장소와 자동으로 통합됩니다. 생성된 인증서 또는 사용자 지정 인증서는 OpenShift Dedicated 기본 제공 OAuth 서버와 자동으로 통합됩니다.

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	**nodePlacement**를 사용하면 Ingress 컨트롤러의 스케줄링을 명시적으로 제어할 수 있습니다.
	
	설정하지 않으면 기본값이 사용됩니다.
	
	 
	참고
	
	nodePlacement 매개변수는 **nodeSelector** 및 **tolerations**의 두 부분으로 구성됩니다. 예를 들면 다음과 같습니다.
	
	`nodePlacement:`
	`nodeSelector:`
	`matchLabels:`
	`kubernetes.io/os: linux`
	`tolerations:`
	`- effect: NoSchedule`
	`operator: Exists`

tlsSecurityProfile	**tlsSecurityProfile**은 Ingress 컨트롤러의 TLS 연결 설정을 지정합니다. 설정되지 않으면, 기본값은 **apiservers.config.openshift.io/cluster** 리소스를 기반으로 설정됩니다. **Old**, **Intermediate** 및 **Modern** 프로파일 유형을 사용하는 경우 유효한 프로파일 구성은 릴리스마다 변경될 수 있습니다. 예를 들어, 릴리스 **X.Y.Z**에 배포된 **Intermediate** 프로파일을 사용하도록 설정한 경우 **X.Y.Z+1** 릴리스로 업그레이드하면 새 프로파일 구성이 Ingress 컨트롤러에 적용되어 롤아웃이 발생할 수 있습니다. Ingress 컨트롤러의 최소 TLS 버전은 **1.1**이며 최대 TLS 버전은 **1.3**입니다.  **참고** 구성된 보안 프로파일의 암호 및 최소 TLS 버전은 **TLSProfile** 상태에 반영됩니다.  **중요** Ingress Operator는 **Old** 또는 **Custom** 프로파일의 TLS **1.0** 을 **1.1** 로 변환합니다.

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

매개변수	설명
IngressControllerLogging	**logging**은 어디에서 무엇이 기록되는지에 대한 매개변수를 정의합니다. 이 필드가 비어 있으면 작동 로그는 활성화되지만 액세스 로그는 비활성화됩니다. • **access**는 클라이언트 요청이 기록되는 방법을 설명합니다. 이 필드가 비어 있으면 액세스 로깅이 비활성화됩니다. ◦ **destination**은 로그 메시지의 대상을 설명합니다. ▪ **type**은 로그 대상의 유형입니다. • **Container** 는 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. • **Syslog**는 로그가 Syslog 끝점으로 전송되도록 지정합니다. 관리자는 Syslog 메시지를 수신할 수 있는 끝점을 지정해야 합니다. 관리자가 사용자 정의 Syslog 인스턴스를 구성하는 것이 좋습니다. ▪ **컨테이너**는 **Container** 로깅 대상 유형의 매개변수를 설명합니다. 현재는 컨테이너 로깅에 대한 매개변수가 없으므로 이 필드는 비어 있어야 합니다. ▪ **syslog**는 **Syslog** 로깅 대상 유형의 매개변수를 설명합니다. • **address**는 로그 메시지를 수신하는 syslog 끝점의 IP 주소입니다. • **port**는 로그 메시지를 수신하는 syslog 끝점의 UDP 포트 번호입니다. • **MaxLength** 는 syslog 메시지의 최대 길이입니다.**480** 에서 **4096** 바이트 사이여야 합니다. 이 필드가 비어 있으면 최대 길이는 기본값인 **1024** 바이트로 설정됩니다. • **facility**는 로그 메시지의 syslog 기능을 지정합니다. 이 필드가 비어 있으면 장치가 **local1**이 됩니다. 아니면 **kern**, **user**, **mail**, **daemon**, **auth**, **syslog**, **lpr**, **news**, **uucp**, **cron**, **auth2**, **ftp**, **ntp**, **audit**, **alert**, **cron2**, **local0**, **local1**, **local2**, **local3**, **local4**, **local5**, **local6** 또는 **local7** 중에서 유효한 syslog 장치를 지정해야 합니다. ◦ **httpLogFormat**은 HTTP 요청에 대한 로그 메시지의 형식을 지정합니다. 이 필드가 비어 있으면 로그 메시지는 구현의 기본 HTTP 로그 형식을 사용합니다. HAProxy의 기본 HTTP 로그 형식과 관련한 내용은 HAProxy 문서를 참조하십시오.

매개변수	설명
httpHeaders	**httpHeaders**는 HTTP 헤더에 대한 정책을 정의합니다. **IngressControllerHTTPHeaders** 에 **forwardedHeaderPolicy** 를 설정하여 Ingress 컨트롤러가 **Forwarded**,**X-Forwarded-For**,**X-Forwarded-Host**,**X-Forwarded-Port**, X-Forwarded-Proto , X-Forwarded-Proto **X-Forwarded- Proto -Version** HTTP 헤더를 설정하는 시기와 방법을 지정합니다. 기본적으로 정책은 **Append**로 설정됩니다. ● **Append**는 Ingress 컨트롤러에서 기존 헤더를 유지하면서 헤더를 추가하도록 지정합니다. ● **Replace**는 Ingress 컨트롤러에서 헤더를 설정하고 기존 헤더를 제거하도록 지정합니다. ● **IfNone**은 헤더가 아직 설정되지 않은 경우 Ingress 컨트롤러에서 헤더를 설정하도록 지정합니다. ● **Never**는 Ingress 컨트롤러에서 헤더를 설정하지 않고 기존 헤더를 보존하도록 지정합니다. **headerNameCaseAdjustments**를 설정하여 HTTP 헤더 이름에 적용할 수 있는 대/소문자 조정을 지정할 수 있습니다. 각 조정은 원하는 대문자를 사용하여 HTTP 헤더 이름으로 지정됩니다. 예를 들어 **X-Forwarded-For**를 지정하면 지정된 대문자를 사용하도록 **x-forwarded-for** HTTP 헤더를 조정해야 합니다. 이러한 조정은 HTTP/1을 사용하는 경우에만 일반 텍스트, 에지 종료 및 재암호화 경로에 적용됩니다. 요청 헤더의 경우 이러한 조정은 **haproxy.router.openshift.io/h1-adjust-case=true** 주석이 있는 경로에만 적용됩니다. 응답 헤더의 경우 이러한 조정이 모든 HTTP 응답에 적용됩니다. 이 필드가 비어 있으면 요청 헤더가 조정되지 않습니다. **작업은** 헤더에서 특정 작업을 수행하기 위한 옵션을 지정합니다. TLS 통과 연결에 대해 헤더를 설정하거나 삭제할 수 없습니다. **actions** 필드에 추가 하위 필드 **spec.httpHeader.actions.response** 및 **spec.httpHeader.actions.request**: ● **응답** 하위 필드는 설정하거나 삭제할 HTTP 응답 헤더 목록을 지정합니다. ● **요청** 하위 필드는 설정하거나 삭제할 HTTP 요청 헤더 목록을 지정합니다.

매개변수	설명
httpCompression	**httpCompression** 은 HTTP 트래픽 압축 정책을 정의합니다. ● **mimetypes** 는 압축을 적용해야 하는 MIME 유형 목록을 정의합니다. 예를 들어 **text/css; charset=utf-8**,**text/html**,**text/***, **image/svg+xml**,**application/octet-stream**,**X-custom/customsub**, 형식 패턴,**type/subtype;[;attribute=value]**. **유형은** 다음과 같습니다: application, image, message, multipart, text, video, or a custom type prefaced by **X-**; 예를 들어 MIME 유형 및 하위 유형에 대한 전체 표기법을 보려면 RFC1341을 참조하십시오.
httpErrorCodePages	**httpErrorCodePages**는 사용자 정의 HTTP 오류 코드 응답 페이지를 지정합니다. 기본적으로 IngressController는 IngressController 이미지에 빌드된 오류 페이지를 사용합니다.

httpCaptureCookies	**httpCaptureCookies** 는 액세스 로그에서 캡처하려는 HTTP 쿠키를 지정합니다. **httpCaptureCookies** 필드가 비어 있으면 액세스 로그에서 쿠키를 캡처하지 않습니다.
	
	캡처하려는 모든 쿠키의 경우 다음 매개변수가 **IngressController** 구성에 있어야 합니다.
	
	● **name** 은 쿠키의 이름을 지정합니다.
	
	● **MaxLength** 는 쿠키의 최대 길이를 지정합니다.
	
	● **matchType** 은 쿠키의 필드**이름이** 캡처 쿠키 설정과 정확히 일치하는지 또는 캡처 쿠키 설정의 접두사인지 지정합니다. **matchType** 필드는 **Exact** 및 **Prefix** 매개변수를 사용합니다.
	
	예를 들면 다음과 같습니다.
	
	`httpCaptureCookies:`
	`- matchType: Exact`
	`  maxLength: 128`
	`  name: MYCOOKIE`

매개변수	설명

httpCaptureHeaders	**httpCaptureHeaders** 는 액세스 로그에서 캡처할 HTTP 헤더를 지정합니다. **httpCaptureHeaders** 필드가 비어 있으면 액세스 로그에서 헤더를 캡처하지 않습니다.
	
	httpCaptureHeaders 에는 액세스 로그에서 캡처할 두 개의 헤더 목록이 포함되어 있습니다. 두 개의 헤더 필드 목록은 **request** 및 **response** 입니다. 두 목록 모두에서 **name** 필드는 헤더 이름을 지정하고 **maxlength** 필드는 헤더의 최대 길이를 지정해야 합니다. 예를 들면 다음과 같습니다.
	
	`httpCaptureHeaders:`
	`  request:`
	`  - maxLength: 256`
	`    name: Connection`
	`  - maxLength: 128`
	`    name: User-Agent`
	`  response:`
	`  - maxLength: 256`
	`    name: Content-Type`
	`  - maxLength: 256`
	`    name: Content-Length`

tuningOptions	**tuningOptions**는 Ingress 컨트롤러 Pod의 성능을 조정하는 옵션을 지정합니다. ● **clientFinTimeout**은 연결을 닫는 서버에 대한 클라이언트 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 **1s** 입니다. ● **ClientTimeout**은 클라이언트 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 **30s** 입니다. ● **headerBufferBytes**는 Ingress 컨트롤러 연결 세션에 대해 예약된 메모리 양을 바이트 단위로 지정합니다. Ingress 컨트롤러에 HTTP/2가 활성화된 경우 이 값은 **16384** 이상이어야 합니다. 설정되지 않은 경우, 기본값은 **32768** 바이트입니다. **headerBufferBytes** 값이 너무 작으면 Ingress 컨트롤러가 손상될 수 있으며 **headerBufferBytes** 값이 너무 크면 Ingress 컨트롤러가 필요 이상으로 많은 메모리를 사용할 수 있기 때문에 이 필드를 설정하지 않는 것이 좋습니다. ● **headerBufferMaxRewriteBytes**는 HTTP 헤더 재작성 및 Ingress 컨트롤러 연결 세션에 대한 **headerBufferBytes**의 바이트 단위로 예약해야 하는 메모리 양을 지정합니다. **headerBufferMaxRewriteBytes**의 최소 값은 **4096입니다**. **headerBufferBytes**는 들어오는 HTTP 요청에 대해 **headerBufferMaxRewriteBytes**보다 커야 합니다. 설정되지 않은 경우, 기본값은 **8192** 바이트입니다. **headerBufferMaxRewriteBytes** 값이 너무 작으면 Ingress 컨트롤러가 손상될 수 있으며 **headerBufferMaxRewriteBytes** 값이 너무 크면 Ingress 컨트롤러가 필요 이상으로 많은 메모리를 사용할 수 있기 때문에 이 필드를 설정하지 않는 것이 좋습니다. ● **healthCheckInterval** 은 라우터가 상태 점검 사이에 대기하는 시간을 지정합니다. 기본값은 **5s**입니다. ● **serverFinTimeout**은 연결을 종료하는 클라이언트에 대한 서버 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 **1s** 입니다.

매개변수	설명
	• **ServerTimeout**은 서버 응답을 기다리는 동안 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 **30s** 입니다. • **threadCount**는 HAProxy 프로세스별로 생성할 스레드 수를 지정합니다. 더 많은 스레드를 생성하면 각 Ingress 컨트롤러 Pod가 더 많은 시스템 리소스 비용으로 더 많은 연결을 처리할 수 있습니다. HAProxy는 최대 **64**개의 스레드를 지원합니다. 이 필드가 비어 있으면 Ingress 컨트롤러는 기본값 **4** 스레드를 사용합니다. 기본값은 향후 릴리스에서 변경될 수 있습니다. HAProxy 스레드 수를 늘리면 Ingress 컨트롤러 Pod에서 부하에 더 많은 CPU 시간을 사용할 수 있고 다른 Pod에서 수행해야 하는 CPU 리소스를 수신하지 못하므로 이 필드를 설정하지 않는 것이 좋습니다. 스레드 수를 줄이면 Ingress 컨트롤러가 제대로 작동하지 않을 수 있습니다. • **tlsInspectDelay**는 라우터에서 일치하는 경로를 찾기 위해 데이터를 보유할 수 있는 기간을 지정합니다. 이 값을 너무 짧게 설정하면 일치하는 인증서를 사용하는 경우에도 라우터가 에지 종료, 재암호화 또는 패스스루 경로의 기본 인증서로 대체될 수 있습니다. 기본 검사 지연은 **5s** 입니다. • **tunnelTimeout**은 터널이 유휴 상태일 때 웹소켓을 포함한 터널 연결이 열린 상태로 유지되는 시간을 지정합니다. 기본 제한 시간은 **1h** 입니다. • **maxConnections** 는 HAProxy 프로세스별로 설정할 수 있는 최대 동시 연결 수를 지정합니다. 이 값을 늘리면 각 Ingress 컨트롤러 Pod에서 추가 시스템 리소스 비용으로 더 많은 연결을 처리할 수 있습니다. 허용되는 값은 **0**,**-1**, 범위 **2000** 및 **2000000** 범위 내의 모든 값 또는 필드를 비워 둘 수 있습니다. ◦ 이 필드가 비어 있거나 값이 **0** 인 경우 Ingress 컨트롤러는 기본값인 **50000** 을 사용합니다. 이 값은 향후 릴리스에서 변경될 수 있습니다. ◦ 필드에 **-1** 값이 있는 경우 HAProxy는 실행 중인 컨테이너에서 사용 가능한 **ulimits** 를 기반으로 값을 동적으로 계산합니다. 이 프로세스에서는 현재 기본값인 **50000** 에 비해 상당한 메모리 사용량이 발생하는 큰 계산 값이 생성됩니다. ◦ 필드에 현재 운영 체제 제한보다 큰 값이 있는 경우 HAProxy 프로세스가 시작되지 않습니다. ◦ 개별 값을 선택하고 라우터 Pod가 새 노드로 마이그레이션되면 새 노드에 동일한 **ulimit** 가 구성되어 있지 않을 수 있습니다. 이러한 경우 Pod가 시작되지 않습니다. ◦ 다른 **ulimits** 가 구성된 노드가 있고 불연속 값을 선택하는 경우 런타임 시 최대 연결 수를 계산하도록 이 필드에 **-1** 값을 사용하는 것이 좋습니다.

매개변수	설명
logEmptyRequests	**logEmptyRequests**는 요청이 수신 및 기록되지 않은 연결을 지정합니다. 이러한 빈 요청은 로드 밸런서 상태 프로브 또는 웹 브라우저 추측 연결(사전 연결)에서 발생하며 이러한 요청을 로깅하는 것은 바람직하지 않을 수 있습니다. 이러한 빈 요청은 로드 밸런서 상태 프로브 또는 웹 브라우저 추측 연결(사전 연결)에서 발생하며 이러한 요청을 로깅하는 것은 바람직하지 않을 수 있습니다. 이러한 요청은 포트 검색으로 인해 발생할 수 있으며 빈 요청을 로깅하면 침입 시도를 감지하는 데 도움이 될 수 있습니다. 이 필드에 허용되는 값은 **Log** 및 **Ignore** 입니다. 기본값은 **Log** 입니다. **LoggingPolicy** 유형은 다음 두 값 중 하나를 허용합니다. ● **Log**: 이 값을 **Log**로 설정하면 이벤트가 로깅되어야 함을 나타냅니다. ● **ignore**: 이 값을 **Ignore** 로 설정하면 HAproxy 구성에서 **dontlognull** 옵션이 설정됩니다.
HTTPEmptyRequestsPolicy	**HTTPEmptyRequestsPolicy**는 요청을 수신하기 전에 연결 시간이 초과된 경우 HTTP 연결이 처리되는 방법을 설명합니다.. 이 필드에 허용되는 값은 **Respond** 및 **Ignore**입니다. 기본값은 **Respond**입니다. **HTTPEmptyRequestsPolicy** 유형은 다음 두 값 중 하나를 허용합니다. ● **Response**: 필드가 **Respond**로 설정된 경우 Ingress 컨트롤러는 HTTP **400** 또는 **408** 응답을 전송하고 액세스 로깅이 활성화된 경우 연결을 로깅한 다음 적절한 메트릭의 연결을 계산합니다. ● **ignore**: 이 옵션을 **Ignore**로 설정하면 HAproxy 구성에 **http-ignore-probes** 매개변수가 추가됩니다. 필드가 **Ignore** 로 설정된 경우 Ingress 컨트롤러는 응답을 전송하지 않고 연결을 종료한 다음 연결을 기록하거나 메트릭을 늘립니다. 이러한 연결은 로드 밸런서 상태 프로브 또는 웹 브라우저 추측 연결(preconnect)에서 제공되며 무시해도 됩니다. 그러나 이러한 요청은 네트워크 오류로 인해 발생할 수 있으므로 이 필드를 **Ignore**로 설정하면 문제를 탐지하고 진단할 수 있습니다. 이러한 요청은 포트 검색으로 인해 발생할 수 있으며, 이 경우 빈 요청을 로깅하면 침입 시도를 탐지하는 데 도움이 될 수 있습니다.

3.3.1. Ingress 컨트롤러 TLS 보안 프로필

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

3.3.1.1. TLS 보안 프로필 이해

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

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

표 3.1. TLS 보안 프로필

Profile	설명
Old	이 프로필은 레거시 클라이언트 또는 라이브러리와 함께 사용하기 위한 것입니다. 프로필은 이전 버전과의 호환성 권장 구성을 기반으로 합니다. **Old** 프로파일에는 최소 TLS 버전 1.0이 필요합니다.  **참고** Ingress 컨트롤러의 경우 최소 TLS 버전이 1.0에서 1.1로 변환됩니다.
Intermediate	이 프로필은 대부분의 클라이언트에서 권장되는 구성입니다. Ingress 컨트롤러, kubelet 및 컨트롤 플레인의 기본 TLS 보안 프로필입니다. 프로필은 중간 호환성 권장 구성을 기반으로 합니다. **Intermediate** 프로필에는 최소 TLS 버전이 1.2가 필요합니다.
Modern	이 프로필은 이전 버전과의 호환성이 필요하지 않은 최신 클라이언트와 사용하기 위한 것입니다. 이 프로필은 최신 호환성 권장 구성을 기반으로 합니다. **Modern** 프로필에는 최소 TLS 버전 1.3이 필요합니다.
사용자 지정	이 프로필을 사용하면 사용할 TLS 버전과 암호를 정의할 수 있습니다.  주의 **Custom** 프로파일을 사용할 때는 잘못된 구성으로 인해 문제가 발생할 수 있으므로 주의해야 합니다.

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.

3.3.1.2. Ingress 컨트롤러의 TLS 보안 프로필 구성

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

Old TLS 보안 프로파일을 구성하는 샘플 IngressController CR

```
apiVersion: operator.openshift.io/v1
```

```
kind: IngressController
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

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

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

참고

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

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

사전 요구 사항

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

절차

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 프로필 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
 ...
```

3.3.1.3. 상호 TLS 인증 구성

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

clientCA 값이 X509v3 인증서 취소 목록(CRL) 배포 지점을 지정하는 경우 Ingress Operator는 각 인증서에 지정된 HTTP URI X509v3 **CRL Distribution Point** 를 기반으로 CRL 구성 맵을 다운로드하고 관리합니다. Ingress 컨트롤러는 mTLS/TLS 협상 중에 이 구성 맵을 사용합니다. 유효한 인증서를 제공하지 않는 요청은 거부됩니다.

사전 요구 사항

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

- PEM 인코딩 CA 인증서 번들이 있습니다.
- CA 번들이 CRL 배포 지점을 참조하는 경우 클라이언트 CA 번들에 엔드센티 또는 리프 인증서도 포함해야 합니다. RFC 5280에 설명된 대로 이 인증서에는 **CRL 배포 지점** 아래에 HTTP URI가 포함되어 있어야 합니다. 예를 들면 다음과 같습니다.

```
Issuer: C=US, O=Example Inc, CN=Example Global G2 TLS RSA SHA256 2020 CA1
        Subject: SOME SIGNED CERT            X509v3 CRL Distribution Points:
              Full Name:
                URI:http://crl.example.com/example.crl
```

절차

1. **openshift-config** 네임스페이스에서 CA 번들에서 구성 맵을 생성합니다.

```
$ oc create configmap \
  router-ca-certs-default \
  --from-file=ca-bundle.pem=client-ca.crt \(1)
  -n openshift-config
```

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

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

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

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

패턴 필터링을 지정하는 clientTLS 프로필에 대한 IngressController CR 샘플

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:
      name: router-ca-certs-default
    allowedSubjectPatterns:
    - "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"
```

4. 선택 사항, 다음 명령을 입력하여 **allowedSubjectPatterns** 에 대한 Distinguished Name (DN)을 가져옵니다.

```
$ openssl  x509 -in custom-cert.pem  -noout -subject
subject= /CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift
```

3.4. 기본 INGRESS 컨트롤러 보기

Ingress Operator는 OpenShift Dedicated의 핵심 기능이며 즉시 사용할 수 있습니다.

모든 새로운 OpenShift Dedicated 설치에는 이름이 default인 **ingresscontroller** 가 있습니다. 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
```

3.5. INGRESS OPERATOR 상태 보기

Ingress Operator의 상태를 확인 및 조사할 수 있습니다.

프로세스

- Ingress Operator 상태를 확인합니다.

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

3.6. INGRESS 컨트롤러 로그 보기

Ingress 컨트롤러의 로그를 확인할 수 있습니다.

프로세스

- Ingress 컨트롤러 로그를 확인합니다.

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

3.7. INGRESS 컨트롤러 상태 보기

특정 Ingress 컨트롤러의 상태를 확인할 수 있습니다.

프로세스

- Ingress 컨트롤러의 상태를 확인합니다.

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

3.8. 사용자 정의 INGRESS 컨트롤러 생성

클러스터 관리자는 새 사용자 정의 Ingress 컨트롤러를 생성할 수 있습니다. 기본 Ingress 컨트롤러는 OpenShift Dedicated 업데이트 중에 변경될 수 있으므로 사용자 정의 Ingress 컨트롤러를 생성하면 클러스터 업데이트 시 구성을 수동으로 유지 관리할 때 유용할 수 있습니다.

이 예에서는 사용자 정의 Ingress 컨트롤러에 대한 최소 사양을 제공합니다. 사용자 정의 Ingress 컨트롤러를 추가로 사용자 지정하려면 " Ingress 컨트롤러 구성"을 참조하십시오.

사전 요구 사항

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

절차

1. 사용자 지정 **IngressController** 오브젝트를 정의하는 YAML 파일을 생성합니다.

   custom-ingress-controller.yaml 파일 예

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
   name: <custom_name> 1
   namespace: openshift-ingress-operator
spec:
   defaultCertificate:
      name: <custom-ingress-custom-certs> 2
   replicas: 1 3
   domain: <custom_domain> 4
```

   1. **IngressController** 오브젝트의 사용자 지정 **이름을** 지정합니다.
   2. 사용자 정의 와일드카드 인증서를 사용하여 보안 이름을 지정합니다.
   3. 최소 복제본이 하나여야 합니다.
   4. 도메인 이름에 도메인을 지정합니다. IngressController 오브젝트에 지정된 도메인과 인증서에 사용된 도메인이 일치해야 합니다. 예를 들어 도메인 값이 "custom_domain.mycompany.com"인 경우 인증서에 SAN *.custom_domain.mycompany.com(***.** 도메인에 추가됨)이 있어야 합니다.

2. 다음 명령을 실행하여 오브젝트를 생성합니다.

```
$ oc create -f custom-ingress-controller.yaml
```

3.9. INGRESS 컨트롤러 구성

3.9.1. 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.com과** 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.

절차

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

참고

이 작업을 수행하면 롤링 배포 전략에 따라 Ingress 컨트롤러가 재배포됩니다.

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

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

2. 새 인증서 보안 키를 참조하도록 IngressController CR을 업데이트합니다.

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

3. 업데이트가 적용되었는지 확인합니다.

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

다음과 같습니다.

<domain>

클러스터의 기본 도메인 이름을 지정합니다.

출력 예

```
-
```

```
subject=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = *.apps.example.com
issuer=C = US, ST = NC, L = Raleigh, O = RH, OU = OCP4, CN = example.com
notAfter=May 10 08:32:45 2022 GM
```

작은 정보

다음 YAML을 적용하여 사용자 지정 기본 인증서를 설정할 수 있습니다.

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

인증서 보안 이름은 CR을 업데이트하는 데 사용된 값과 일치해야 합니다.

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

3.9.2. 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 구성했습니다.

절차

- 사용자 정의 인증서를 제거하고 OpenShift Dedicated와 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
```

3.9.3. Ingress 컨트롤러 자동 스케일링

처리량 증가 요구와 같은 라우팅 성능 또는 가용성 요구 사항을 동적으로 충족하도록 Ingress 컨트롤러를 자동으로 확장합니다. 다음 절차는 기본 **IngressController**를 확장하는 예제입니다.

사전 요구 사항

1. OpenShift CLI(**oc**)가 설치되어 있어야 합니다.
2. **cluster-admin** 역할의 사용자로 OpenShift Dedicated 클러스터에 액세스할 수 있습니다.
3. Custom Metrics Autoscaler Operator가 설치되어 있습니다.
4. **openshift-ingress-operator** 프로젝트 네임스페이스에 있습니다.

절차

1. 다음 명령을 실행하여 Thanos로 인증할 서비스 계정을 생성합니다.

```
$ oc create serviceaccount thanos && oc describe serviceaccount thanos
```

출력 예

```
Name:                thanos
Namespace:           openshift-ingress-operator
Labels:              <none>
Annotations:         <none>
Image pull secrets:  thanos-dockercfg-b4l9s
Mountable secrets:   thanos-dockercfg-b4l9s
Tokens:              thanos-token-c422q
Events:              <none>
```

2. 다음 명령을 사용하여 서비스 계정 시크릿 토큰을 수동으로 생성합니다.

```
$ oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: thanos-token
  annotations:
```

```
    kubernetes.io/service-account.name: thanos
  type: kubernetes.io/service-account-token
EOF
```

3. 서비스 계정의 토큰을 사용하여 **openshift-ingress-operator** 네임스페이스에 **TriggerAuthentication** 오브젝트를 정의합니다.

   a. 다음 명령을 실행하여 **시크릿** 이 포함된 변수 보안을 정의합니다.

   ```
   $ secret=$(oc get secret | grep thanos-token | head -n 1 | awk '{ print $1 }')
   ```

   b. **TriggerAuthentication** 오브젝트를 생성하고 **secret** 변수 값을 **TOKEN** 매개변수에 전달합니다.

   ```
   $ oc process TOKEN="$secret" -f - <<EOF | oc apply -f -
   apiVersion: template.openshift.io/v1
   kind: Template
   parameters:
   - name: TOKEN
   objects:
   - apiVersion: keda.sh/v1alpha1
     kind: TriggerAuthentication
     metadata:
       name: keda-trigger-auth-prometheus
     spec:
       secretTargetRef:
       - parameter: bearerToken
         name: \${TOKEN}
         key: token
       - parameter: ca
         name: \${TOKEN}
         key: ca.crt
   EOF
   ```

4. Thanos에서 메트릭을 읽는 역할을 생성하고 적용합니다.

   a. Pod 및 노드에서 지표를 읽는 새 역할 **thanos-metrics-reader.yaml** 을 생성합니다.

   thanos-metrics-reader.yaml

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: Role
   metadata:
     name: thanos-metrics-reader
   rules:
   - apiGroups:
     - ""
     resources:
     - pods
     - nodes
     verbs:
     - get
   - apiGroups:
     - metrics.k8s.io
     resources:
   ```

```
    - pods
    - nodes
    verbs:
    - get
    - list
    - watch
  - apiGroups:
    - ""
    resources:
    - namespaces
    verbs:
    - get
```

b. 다음 명령을 실행하여 새 역할을 적용합니다.

```
$ oc apply -f thanos-metrics-reader.yaml
```

5. 다음 명령을 입력하여 서비스 계정에 새 역할을 추가합니다.

```
$ oc adm policy add-role-to-user thanos-metrics-reader -z thanos --role-
namespace=openshift-ingress-operator
```

```
$ oc adm policy -n openshift-ingress-operator add-cluster-role-to-user cluster-monitoring-view
-z thanos
```

참고

add-cluster-role-to-user 인수는 네임스페이스 간 쿼리를 사용하는 경우에만 필요합니다. 다음 단계에서는 이 인수가 필요한 **kube-metrics** 네임스페이스의 쿼리를 사용합니다.

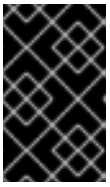
6. 기본 Ingress 컨트롤러 배포를 대상으로 하는 새 scaled **Object** YAML 파일 **ingress-autoscaler.yaml** 을 만듭니다.

scaled Object 정의의 예

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: ingress-scaler
spec:
  scaleTargetRef: 1
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 2
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
  - type: prometheus
    metricType: AverageValue
```

```
  metadata:
    serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 3
    namespace: openshift-ingress-operator 4
    metricName: 'kube-node-role'
    threshold: '1'
    query: 'sum(kube_node_role{role="worker",service="kube-state-metrics"})' 5
    authModes: "bearer"
  authenticationRef:
    name: keda-trigger-auth-prometheus
```

1. 대상으로 하는 사용자 정의 리소스입니다. 이 경우 Ingress 컨트롤러입니다.
2. 선택사항: 최대 복제본 수입니다. 이 필드를 생략하면 기본 최대값이 100개의 복제본으로 설정됩니다.
3. **openshift-monitoring** 네임스페이스의 Thanos 서비스 끝점입니다.
4. Ingress Operator 네임스페이스입니다.
5. 이 표현식은 배포된 클러스터에 많은 작업자 노드가 있는 것으로 평가됩니다.



중요

네임스페이스 간 쿼리를 사용하는 경우 **serverAddress** 필드에서 포트 9091이 아닌 포트 9091을 대상으로 지정해야 합니다. 또한 이 포트에서 메트릭을 읽을 수 있는 높은 권한이 있어야 합니다.

7. 다음 명령을 실행하여 사용자 정의 리소스 정의를 적용합니다.

```
$ oc apply -f ingress-autoscaler.yaml
```

검증

- 다음 명령을 실행하여 기본 Ingress 컨트롤러가 **kube-state-metrics** 쿼리에서 반환된 값과 일치하도록 확장되었는지 확인합니다.
  - **grep** 명령을 사용하여 Ingress 컨트롤러 YAML 파일에서 복제본을 검색합니다.

```
$ oc get ingresscontroller/default -o yaml | grep replicas:
```

출력 예

```
replicas: 3
```

  - **openshift-ingress** 프로젝트에서 Pod를 가져옵니다.

```
$ oc get pods -n openshift-ingress
```

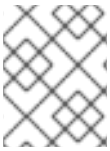
출력 예

```
NAME                              READY   STATUS    RESTARTS   AGE
router-default-7b5df44ff-l9pmm   2/2     Running   0          17h
```

```
router-default-7b5df44ff-s5sl5   2/2     Running   0          3d22h
router-default-7b5df44ff-wwsth   2/2     Running   0          66s
```

3.9.4. Ingress 컨트롤러 확장

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



참고

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

절차

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

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

출력 예

```
2
```

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

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

출력 예

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

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

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

출력 예

```
3
```

작은 정보

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

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 3 1
```

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

3.9.5. 수신 액세스 로깅 구성

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

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

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

사전 요구 사항

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

절차

사이드카에 Ingress 액세스 로깅을 구성합니다.

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

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

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

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

출력 예

```
2020-05-11T19:11:50.135710+00:00 router-default-57dfc6cd95-bpmk6 router-default-57dfc6cd95-bpmk6 haproxy[108]: 174.19.21.82:39654 [11/May/2020:19:11:50.133] public be_http:hello-openshift:hello-openshift/pod:hello-openshift:hello-openshift:10.128.2.12:8080 0/0/1/0/1 200 142 - - --NI 1/1/0/0/0 0/0 "GET / HTTP/1.1"
```

Syslog 끝점에 대한 Ingress 액세스 로깅을 구성합니다.

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

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
```

참고

syslog 대상 포트는 UDP여야 합니다.

syslog 대상 주소는 IP 주소여야 합니다. DNS 호스트 이름을 지원하지 않습니다.

특정 로그 형식으로 Ingress 액세스 로깅을 구성합니다.

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

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

```
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          port: 10514
      httpLogFormat: '%ci:%cp [%t] %ft %b/%s %B %bq %HM %HU %HV'
```

Ingress 액세스 로깅을 비활성화합니다.

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

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

사이드카를 사용할 때 Ingress 컨트롤러에서 HAProxy 로그 길이를 수정할 수 있도록 허용합니다.

- **spec.logging.access.destination.destination. type: Syslog를 사용하는 경우 spec.logging.access.destination. maxLength** 를 사용합니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Syslog
        syslog:
          address: 1.2.3.4
          maxLength: 4096
          port: 10514
```

- **spec.logging.access.destination.destination. type: Container를 사용하는 경우 spec.logging.access.destination. maxLength** 를 사용합니다.

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

```
logging:
  access:
    destination:
      type: Container
      container:
        maxLength: 8192
```

3.9.6. Ingress 컨트롤러 스레드 수 설정

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

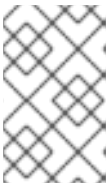
사전 요구 사항

- 다음은 Ingress 컨트롤러를 이미 생성했다고 가정합니다.

절차

- 스레드 수를 늘리도록 Ingress 컨트롤러를 업데이트합니다.

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



참고

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

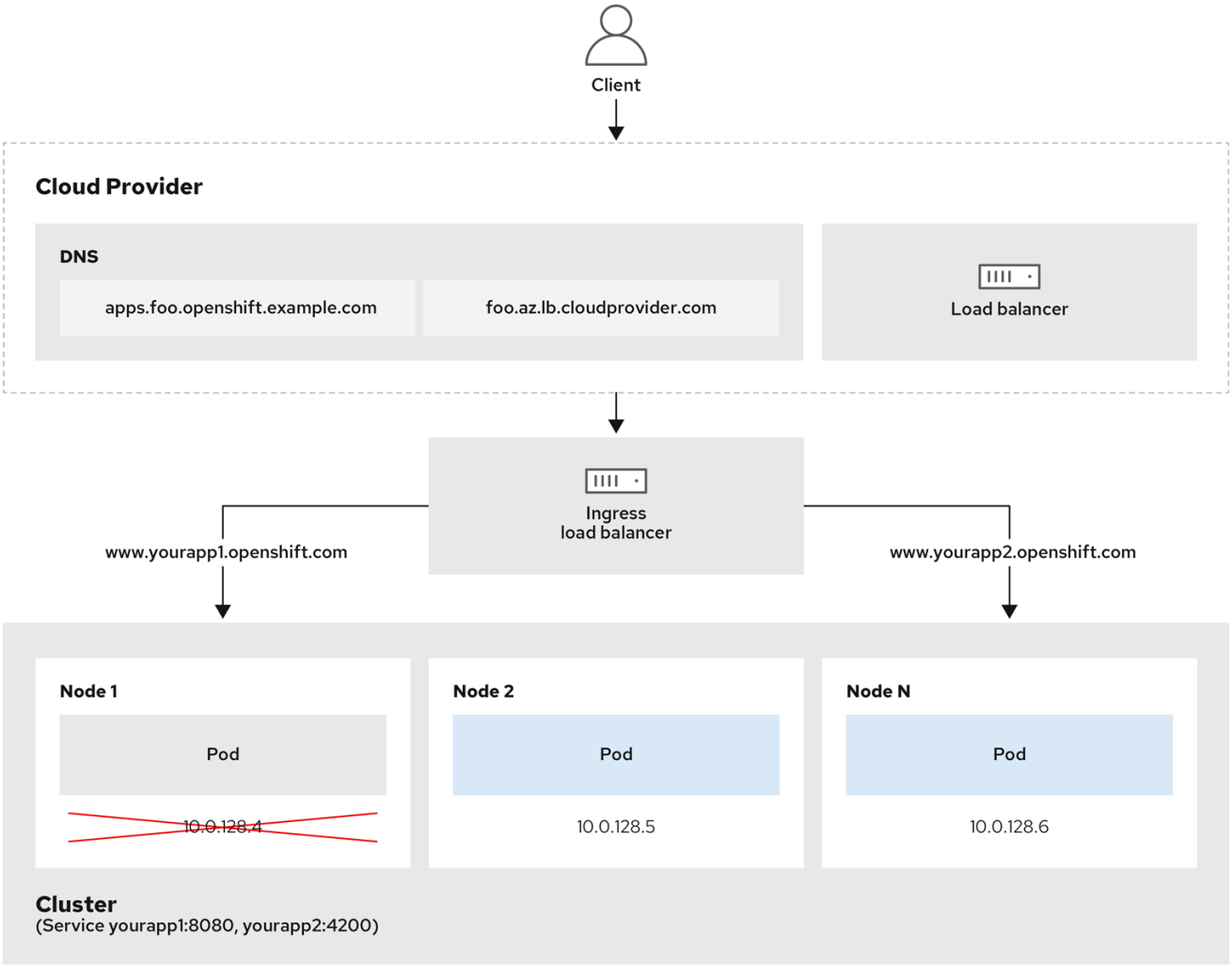
3.9.7. 내부 로드 밸런서를 사용하도록 Ingress 컨트롤러 구성

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.

중요

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

그림 3.1. LoadBalancer 다이어그램



이전 그림에서는 OpenShift Dedicated Ingress LoadBalancerService 끝점 게시 전략과 관련된 다음 개념을 보여줍니다.

- 클라우드 공급자 로드 밸런서를 사용하거나 내부적으로 OpenShift Ingress 컨트롤러 로드 밸런서를 사용하여 외부에서 부하를 분산할 수 있습니다.
- 그래픽에 표시된 클러스터에 표시된 대로 로드 밸런서의 단일 IP 주소와 8080 및 4200과 같은 더 친숙한 포트를 사용할 수 있습니다.
- 외부 로드 밸런서의 트래픽은 Pod에서 전달되고 다운 노드의 인스턴스에 표시된 대로 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.

절차

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

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> 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: Ingress 컨트롤러가 생성되었는지 확인하려면 다음 명령을 실행합니다.

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

3.9.8. Ingress 컨트롤러 상태 점검 간격 설정

클러스터 관리자는 상태 점검 간격을 설정하여 라우터가 연속 상태 점검 사이에 대기하는 시간을 정의할 수 있습니다. 이 값은 모든 경로에 대해 전역적으로 적용됩니다. 기본값은 5초입니다.

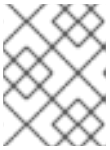
사전 요구 사항

- 다음은 Ingress 컨트롤러를 이미 생성했다고 가정합니다.

프로세스

- 백엔드 상태 점검 간 간격을 변경하도록 Ingress 컨트롤러를 업데이트합니다.

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

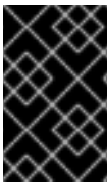


참고

단일 경로의 **healthCheckInterval** 을 재정의하려면 경로 주석 **router.openshift.io/haproxy.health.check.interval**을 사용합니다.

3.9.9. 클러스터의 기본 Ingress 컨트롤러를 내부로 구성

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



중요

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

사전 요구 사항

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

프로세스

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

```
$ oc replace --force --wait --filename - <<EOF
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: default
spec:
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal
EOF
```

3.9.10. 경로 허용 정책 구성

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

주의

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

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.

프로세스

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

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

샘플 Ingress 컨트롤러 구성

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

작은 정보

다음 YAML을 적용하여 경로 승인 정책을 구성할 수 있습니다.

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

3.9.11. 와일드카드 경로 사용

HAProxy Ingress 컨트롤러는 와일드카드 경로를 지원합니다. Ingress Operator는 **wildcardPolicy**를 사용하여 Ingress 컨트롤러의 **ROUTER_ALLOW_WILDCARD_ROUTES** 환경 변수를 구성합니다.

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

프로세스

1. 와일드카드 정책을 구성합니다.
   a. 다음 명령을 사용하여 **IngressController** 리소스를 편집합니다.

```
$ oc edit IngressController
```

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

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

3.9.12. HTTP 헤더 구성

OpenShift Dedicated는 HTTP 헤더로 작업하는 다양한 방법을 제공합니다. 헤더를 설정하거나 삭제할 때 Ingress 컨트롤러의 특정 필드를 사용하거나 개별 경로를 사용하여 요청 및 응답 헤더를 수정할 수 있습니다. 경로 주석을 사용하여 특정 헤더를 설정할 수도 있습니다. 헤더를 구성하는 다양한 방법은 함께 작업할 때 문제가 발생할 수 있습니다.

참고

IngressController 또는 **Route** CR 내에서 헤더만 설정하거나 삭제할 수 있으므로 추가할 수 없습니다. HTTP 헤더가 값으로 설정된 경우 해당 값은 완료되어야 하며 나중에 추가할 필요가 없습니다. X-Forwarded-For 헤더와 같은 헤더를 추가하는 것이 적합한 경우 **spec.httpHeaders.actions** 대신 **spec.httpHeaders.forwardedHeaderPolicy** 필드를 사용합니다.

3.9.12.1. 우선순위 순서

Ingress 컨트롤러와 경로에서 동일한 HTTP 헤더를 수정하는 경우 HAProxy는 요청 또는 응답 헤더인지 여부에 따라 특정 방식으로 작업에 우선순위를 부여합니다.

- HTTP 응답 헤더의 경우 경로에 지정된 작업 후에 Ingress 컨트롤러에 지정된 작업이 실행됩니다. 즉, Ingress 컨트롤러에 지정된 작업이 우선합니다.
- HTTP 요청 헤더의 경우 경로에 지정된 작업은 Ingress 컨트롤러에 지정된 작업 후에 실행됩니다. 즉, 경로에 지정된 작업이 우선합니다.

예를 들어 클러스터 관리자는 다음 구성을 사용하여 Ingress 컨트롤러에서 값이 **DENY** 인 X-Frame-Options 응답 헤더를 설정합니다.

IngressController 사양 예

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: DENY
```

경로 소유자는 클러스터 관리자가 Ingress 컨트롤러에 설정한 것과 동일한 응답 헤더를 설정하지만 다음 구성을 사용하여 **SAMEORIGIN** 값이 사용됩니다.

Route 사양의 예

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
```

```
      action:
        type: Set
        set:
          value: SAMEORIGIN
```

IngressController 사양과 **Route** 사양 모두에서 X-Frame-Options 응답 헤더를 구성하는 경우 특정 경로에서 프레임을 허용하는 경우에도 Ingress 컨트롤러의 글로벌 수준에서 이 헤더에 설정된 값이 우선합니다. 요청 헤더의 경우 **Route** spec 값은 **IngressController** 사양 값을 재정의합니다.

이 우선순위는 **haproxy.config** 파일에서 다음 논리를 사용하므로 Ingress 컨트롤러가 프런트 엔드로 간주되고 개별 경로가 백엔드로 간주되기 때문입니다. 프런트 엔드 구성에 적용된 헤더 값 **DENY** 는 백엔드에 설정된 **SAMEORIGIN** 값으로 동일한 헤더를 재정의합니다.

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

또한 Ingress 컨트롤러 또는 경로 주석을 사용하여 설정된 경로 덮어쓰기 값에 정의된 모든 작업입니다.

3.9.12.2. 특수 케이스 헤더

다음 헤더는 완전히 설정되거나 삭제되지 않거나 특정 상황에서 허용되지 않습니다.

표 3.2. 특수 케이스 헤더 구성 옵션

헤더 이름	IngressController 사양을 사용하여 구성 가능	Route 사양을 사용하여 구성 가능	허용하지 않는 이유	다른 방법을 사용하여 구성 가능
proxy	없음	없음	**프록시** HTTP 요청 헤더는 **HTTP_PROXY** 환경 변수에 헤더 값을 삽입하여 취약한 CGI 애플리케이션을 활용하는 데 사용할 수 있습니다. **프록시** HTTP 요청 헤더는 비표준이며 구성 중에 오류가 발생하기 쉽습니다.	없음

헤더 이름	IngressController 사양을 사용하여 구성 가능	Route 사양을 사용하여 구성 가능	허용하지 않는 이유	다른 방법을 사용하여 구성 가능
host	없음	제공됨	**IngressController** CR을 사용하여 **호스트** HTTP 요청 헤더를 설정하면 올바른 경로를 찾을 때 HAProxy가 실패할 수 있습니다.	없음
strict-transport-security	없음	없음	**strict-transport-security** HTTP 응답 헤더는 경로 주석을 사용하여 이미 처리되었으며 별도의 구현이 필요하지 않습니다.	제공됨: **haproxy.router.openshift.io/hsts_header** 경로 주석
쿠키 및 **설정 쿠키**	없음	없음	HAProxy가 클라이언트 연결을 특정 백엔드 서버에 매핑하는 세션 추적에 사용되는 쿠키입니다. 이러한 헤더를 설정하도록 허용하면 HAProxy의 세션 선호도를 방해하고 HAProxy의 쿠키 소유권을 제한할 수 있습니다.	예: • **haproxy.router.openshift.io/disable_cookie** 경로 주석 • **haproxy.router.openshift.io/cookie_name** 경로 주석

3.9.13. Ingress 컨트롤러에서 HTTP 요청 및 응답 헤더 설정 또는 삭제

규정 준수 목적 또는 기타 이유로 특정 HTTP 요청 및 응답 헤더를 설정하거나 삭제할 수 있습니다. Ingress 컨트롤러에서 제공하는 모든 경로 또는 특정 경로에 대해 이러한 헤더를 설정하거나 삭제할 수 있습니다.

예를 들어 상호 TLS를 사용하기 위해 클러스터에서 실행 중인 애플리케이션을 마이그레이션할 수 있습니다. 이 경우 애플리케이션에서 X-Forwarded-Client-Cert 요청 헤더를 확인해야 하지만 OpenShift Dedicated 기본 Ingress 컨트롤러는 X-SSL-Client-Der 요청 헤더를 제공합니다.

다음 절차에서는 X-Forwarded-Client-Cert 요청 헤더를 설정하도록 Ingress 컨트롤러를 수정하고 X-SSL-Client-Der 요청 헤더를 삭제합니다.

사전 요구 사항

- OpenShift CLI(**oc**)가 설치되어 있습니다.

- **cluster-admin** 역할의 사용자로 OpenShift Dedicated 클러스터에 액세스할 수 있습니다.

프로세스

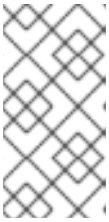
1. Ingress 컨트롤러 리소스를 편집합니다.

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

2. X-SSL-Client-Der HTTP 요청 헤더를 X-Forwarded-Client-Cert HTTP 요청 헤더로 바꿉니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpHeaders:
    actions: 1
      request: 2
      - name: X-Forwarded-Client-Cert 3
        action:
          type: Set 4
          set:
            value: "%{+Q}[ssl_c_der,base64]" 5
      - name: X-SSL-Client-Der
        action:
          type: Delete
```

1. HTTP 헤더에서 수행할 작업 목록입니다.
2. 변경할 헤더 유형입니다. 이 경우 요청 헤더가 있습니다.
3. 변경할 헤더의 이름입니다. 설정하거나 삭제할 수 있는 사용 가능한 헤더 목록은 *HTTP 헤더 구성* 을 참조하십시오.
4. 헤더에서 수행되는 작업 유형입니다. 이 필드에는 **Set** 또는 **Delete** 값이 있을 수 있습니다.
5. HTTP 헤더를 설정할 때 **값을** 제공해야 합니다. 값은 해당 헤더에 사용 가능한 지시문 목록(예: **DENY**)의 문자열이거나 HAProxy의 동적 값 구문을 사용하여 해석되는 동적 값이 될 수 있습니다. 이 경우 동적 값이 추가됩니다.



참고

HTTP 응답에 대한 동적 헤더 값을 설정하기 위해 허용되는 샘플 페이퍼는 **res.hdr** 및 **ssl_c_der** 입니다. HTTP 요청에 대한 동적 헤더 값을 설정하는 경우 허용되는 샘플 페더는 **req.hdr** 및 **ssl_c_der** 입니다. request 및 response 동적 값은 모두 **lower** 및 **base64** 컨버터를 사용할 수 있습니다.

3. 파일을 저장하여 변경 사항을 적용합니다.

3.9.14. 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.

절차

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
```

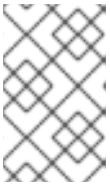
사용 사례 예
클러스터 관리자는 다음을 수행할 수 있습니다.

- 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.

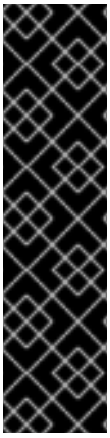
3.9.15. HTTP/2 수신 연결 사용

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.



중요

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

절차

단일 Ingress 컨트롤러에서 HTTP/2를 활성화합니다.

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

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

  <ingresscontroller_name>을 주석 처리할 Ingress 컨트롤러의 이름으로 변경합니다.

전체 클러스터에서 HTTP/2를 활성화합니다.

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

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

작은 정보

다음 YAML을 적용하여 주석을 추가할 수도 있습니다.

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

3.9.16. Ingress 컨트롤러에 대한 PROXY 프로토콜 구성

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

주의

Keepalived Ingress 가상 IP(VIP)를 사용하는 클라우드 플랫폼에 설치 관리자 프로비저닝 클러스터가 있는 기본 Ingress 컨트롤러는 PROXY 프로토콜을 지원하지 않습니다.

PROXY 프로토콜을 사용하면 로드 밸런서에서 Ingress 컨트롤러가 수신하는 연결에 대한 원래 클라이언트 주소를 유지할 수 있습니다. 원래 클라이언트 주소는 HTTP 헤더를 로깅, 필터링 및 삽입하는 데 유용합니다. 기본 구성에서 Ingress 컨트롤러에서 수신하는 연결에는 로드 밸런서와 연결된 소스 IP 주소만 포함됩니다.

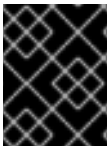
중요

패스스루 경로 구성의 경우 OpenShift Dedicated 클러스터의 서버는 원래 클라이언트 소스 IP 주소를 관찰할 수 없습니다. 원래 클라이언트 소스 IP 주소를 알아야 하는 경우 클라이언트 소스 IP 주소를 볼 수 있도록 Ingress 컨트롤러에 대한 Ingress 액세스 로깅을 구성합니다.

재암호화 및 에지 경로의 경우 OpenShift Dedicated 라우터는 애플리케이션 워크로드가 클라이언트 소스 IP 주소를 확인하도록 **Forwarded** 및 **X-Forwarded-For** 헤더를 설정합니다.

Ingress 액세스 로깅에 대한 자세한 내용은 " Ingress 액세스 로깅 구성"을 참조하십시오.

LoadBalancerService 끝점 게시 전략 유형을 사용하는 경우 Ingress 컨트롤러에 대한 PROXY 프로토콜 구성은 지원되지 않습니다. 이 제한은 OpenShift Dedicated가 클라우드 플랫폼에서 실행되고 Ingress 컨트롤러에서 서비스 로드 밸런서를 사용해야 함을 지정하기 때문에 Ingress Operator는 로드 밸런서 서비스를 구성하고 소스 주소를 유지하기 위한 플랫폼 요구 사항에 따라 PROXY 프로토콜을 활성화하기 때문입니다.



중요

PROXY 프로토콜 또는 TCP를 사용하도록 OpenShift Dedicated 및 외부 로드 밸런서를 모두 구성해야 합니다.

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

중요

PROXY 프로토콜을 사용하거나 TCP(Transmission Control Protocol)를 사용하도록 OpenShift Dedicated 및 외부 로드 밸런서를 모두 구성해야 합니다.

사전 요구 사항

- Ingress 컨트롤러가 생성되어 있습니다.

절차

1. CLI에 다음 명령을 입력하여 Ingress 컨트롤러 리소스를 편집합니다.

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

2. PROXY 구성을 설정합니다.

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

     PROXY에 대한 hostNetwork 구성 샘플

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

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

     PROXY에 대한 nodePort 구성 샘플

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

- Ingress 컨트롤러에서 **Private** 끝점 게시 전략 유형을 사용하는 경우 **spec.endpointPublishingStrategy.private.protocol** 하위 필드를 **PROXY** 로 설정합니다.

  PROXY에 대한 개인 구성 샘플

```
# ...
  spec:
    endpointPublishingStrategy:
      private:
        protocol: PROXY
    type: Private
# ...
```

추가 리소스

- 수신 액세스 로깅 구성

3.9.17. appsDomain 옵션을 사용하여 대체 클러스터 도메인 지정

클러스터 관리자는 **appsDomain** 필드를 구성하여 사용자가 생성한 경로에 대한 기본 클러스터 도메인의 대안을 지정할 수 있습니다. **appsDomain** 필드는 도메인 필드에 지정된 기본값 대신 사용할 OpenShift Dedicated의 선택적 **도메인** 입니다. 대체 도메인을 지정하면 새 경로의 기본 호스트를 결정하기 위해 기본 클러스터 도메인을 덮어씁니다.

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

사전 요구 사항

- OpenShift Dedicated 클러스터를 배포했습니다.
- **oc** 명령줄 인터페이스를 설치했습니다.

절차

1. 사용자 생성 경로에 대한 대체 기본 도메인을 지정하여 **appsDomain** 필드를 구성합니다.

   a. ingress **클러스터** 리소스를 편집합니다.

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

   b. YAML 파일을 편집합니다.

   test.example.com에 대한 appsDomain 구성 샘플

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

1. 기본 도메인을 지정합니다. 설치 후에는 기본 도메인을 수정할 수 없습니다.
2. 선택 사항: 애플리케이션 경로에 사용할 OpenShift Dedicated 인프라의 도메인입니다. 기본 접두사인 **앱** 대신 **test** 와 같은 대체 접두사를 사용할 수 있습니다.

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

참고

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

a. 경로를 노출합니다.

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

출력 예

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

3.9.18. HTTP 헤더 대소문자 변환

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

중요

OpenShift Dedicated에는 HAProxy 2.8이 포함되어 있습니다. 이 웹 기반 로드 밸런서 버전으로 업데이트하려면 **spec.httpHeaders.headerNameCaseAdjustments** 섹션을 클러스터의 구성 파일에 추가해야 합니다.

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

사전 요구 사항

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

프로세스

- **oc patch** 명령을 사용하여 HTTP 헤더를 대문자로 설정합니다.

a. 다음 명령을 실행하여 HTTP 헤더를 **host** 에서 **Host** 로 변경합니다.

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

b. 주석을 애플리케이션에 적용할 수 있도록 **Route** 리소스 YAML 파일을 생성합니다.

my-application이라는 경로 예

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true (1)
  name: <application_name>
  namespace: <application_name>
# ...
```

(1) Ingress 컨트롤러가 지정된 대로 **호스트** 요청 헤더를 조정할 수 있도록 **haproxy.router.openshift.io/h1-adjust-case** 를 설정합니다.

- Ingress 컨트롤러 YAML 구성 파일에서 **HeaderNameCaseAdjustments** 필드를 구성하여 조정을 지정합니다.

a. 다음 예제 Ingress 컨트롤러 YAML 파일은 적절한 주석이 달린 경로에 대해 HTTP/1 요청에 대해 **호스트** 헤더를 **Host** 로 조정합니다.

Ingress 컨트롤러 YAML 예시

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

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

경로 YAML의 예

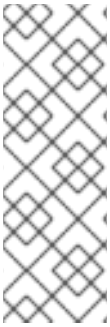
```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/h1-adjust-case: true (1)
  name: my-application
  namespace: my-application
spec:
```

```
  to:
    kind: Service
    name: my-application
```

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

3.9.19. 라우터 압축 사용

특정 MIME 유형에 대해 전역적으로 라우터 압축을 지정하도록 HAProxy Ingress 컨트롤러를 구성합니다. **mimeTypes** 변수를 사용하여 압축이 적용되는 MIME 유형의 형식을 정의할 수 있습니다. 유형은 application, image, message, multipart, text, video 또는 "X-"가 붙은 사용자 지정 유형입니다. MIME 유형 및 하위 유형에 대한 전체 표기법을 보려면 RFC1341 을 참조하십시오.



참고

압축에 할당된 메모리는 최대 연결에 영향을 미칠 수 있습니다. 또한 큰 버퍼를 압축하면 많은 regex 또는 긴 regex 목록과 같은 대기 시간이 발생할 수 있습니다.

모든 MIME 유형이 압축의 이점은 아니지만 HAProxy는 여전히 리소스를 사용하여 다음을 지시한 경우 압축합니다. 일반적으로 html, css, js와 같은 텍스트 형식은 압축할 수 있지만 이미 압축한 형식(예: 이미지, 오디오, 비디오 등)은 압축에 소요되는 시간과 리소스를 거의 교환하지 못합니다.

프로세스

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

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

   b. **spec** 에서 **httpCompression** 정책 필드를 **mimeTypes** 로 설정하고 압축이 적용되어야 하는 MIME 유형 목록을 지정합니다.

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  httpCompression:
    mimeTypes:
    - "text/html"
    - "text/css; charset=utf-8"
    - "application/json"
     ...
```

3.9.20. 라우터 지표 노출

기본 통계 포트인 1936에서 Prometheus 형식으로 기본적으로 HAProxy 라우터 지표를 노출할 수 있습니다. Prometheus와 같은 외부 메트릭 컬렉션 및 집계 시스템은 HAProxy 라우터 지표에 액세스할 수 있습니다. 브라우저에서 HAProxy 라우터 메트릭을 HTML 및 쉼표로 구분된 값(CSV) 형식으로 볼 수 있습니다.

사전 요구 사항

- 기본 통계 포트인 1936에 액세스하도록 방화벽을 구성했습니다.

프로세스

1. 다음 명령을 실행하여 라우터 Pod 이름을 가져옵니다.

```
$ oc get pods -n openshift-ingress
```

출력 예

```
NAME                              READY   STATUS    RESTARTS   AGE
router-default-76bfffb66c-46qwp   1/1     Running   0          11h
```

2. 라우터 Pod가 **/var/lib/haproxy/conf/metrics-auth/statsUsername** 및 **/var/lib/haproxy/conf/metrics-auth/statsPassword** 파일에 저장하는 라우터의 사용자 이름과 암호를 가져옵니다.

   a. 다음 명령을 실행하여 사용자 이름을 가져옵니다.

```
$ oc rsh <router_pod_name> cat metrics-auth/statsUsername
```

   b. 다음 명령을 실행하여 암호를 가져옵니다.

```
$ oc rsh <router_pod_name> cat metrics-auth/statsPassword
```

3. 다음 명령을 실행하여 라우터 IP 및 메트릭 인증서를 가져옵니다.

```
$ oc describe pod <router_pod>
```

4. 다음 명령을 실행하여 Prometheus 형식으로 원시 통계를 가져옵니다.

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

5. 다음 명령을 실행하여 메트릭에 안전하게 액세스합니다.

```
$ curl -u user:password https://<router_IP>:<stats_port>/metrics -k
```

6. 다음 명령을 실행하여 기본 stats 포트 1936에 액세스합니다.

```
$ curl -u <user>:<password> http://<router_IP>:<stats_port>/metrics
```

예 3.1. 출력 예

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route-alt"} 0
```

```
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""} 0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. 브라우저에 다음 URL을 입력하여 통계 창을 시작합니다.

```
http://<user>:<password>@<router_IP>:<stats_port>
```

8. 선택 사항: 브라우저에 다음 URL을 입력하여 CSV 형식으로 통계를 가져옵니다.

```
http://<user>:<password>@<router_ip>:1936/metrics;csv
```

3.9.21. HAProxy 오류 코드 응답 페이지 사용자 정의

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

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

사용자 지정 HTTP 오류 코드 응답 페이지는 HAProxy HTTP 오류 페이지 구성 지침 을 따라야 합니다. 다음은 기본 OpenShift Dedicated HAProxy 라우터 http 503 오류 코드 응답 페이지의 예입니다. 기본 콘텐츠를 고유한 사용자 지정 페이지를 생성하기 위한 템플릿으로 사용할 수 있습니다.

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

참고

OpenShift Dedicated 기본 503 오류 코드 페이지를 사용자 지정 템플릿으로 사용하는 경우 파일의 헤더에는 CRLF 줄 끝을 사용할 수 있는 편집기가 필요합니다.

프로세스

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

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

중요

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

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

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

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

3. 복사본을 표시합니다.

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

출력 예

```
NAME                 DATA   AGE
default-errorpages     2    25s  1
```

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

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

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

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

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

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

검증

사용자 정의 오류 코드 HTTP 응답을 확인합니다.

1. 테스트 프로젝트 및 애플리케이션을 생성합니다.

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

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

2. 503 사용자 정의 http 오류 코드 응답의 경우:
   a. 애플리케이션의 모든 pod를 중지합니다.
   b. 다음 curl 명령을 실행하거나 브라우저에서 경로 호스트 이름을 방문합니다.

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

3. 404 사용자 정의 http 오류 코드 응답의 경우:
   a. 존재하지 않는 경로 또는 잘못된 경로를 방문합니다.
   b. 다음 curl 명령을 실행하거나 브라우저에서 경로 호스트 이름을 방문합니다.

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

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

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

3.9.22. Ingress 컨트롤러 최대 연결 설정

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

사전 요구 사항

- 다음은 Ingress 컨트롤러를 이미 생성했다고 가정합니다.

프로세스

- HAProxy의 최대 연결 수를 변경하도록 Ingress 컨트롤러를 업데이트합니다.

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

주의

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

3.10. OPENSHIFT DEDICATED INGRESS OPERATOR 구성

다음 표에서는 Ingress Operator의 구성 요소를 자세히 설명하고 Red Hat 사이트 안정성 엔지니어(SRE)가 OpenShift Dedicated 클러스터에서 이 구성 요소를 유지 관리하는 경우입니다.

표 3.3. Ingress Operator 책임 차트

Ingress 구성 요소	관리 대상	기본 설정?
스케일링 Ingress 컨트롤러	SRE	제공됨
Ingress Operator 스레드 수	SRE	제공됨
Ingress 컨트롤러 액세스 로깅	SRE	제공됨
Ingress 컨트롤러 분할	SRE	제공됨
Ingress 컨트롤러 경로 허용 정책	SRE	제공됨
Ingress 컨트롤러 와일드카드 경로	SRE	제공됨
Ingress 컨트롤러 X-Forwarded 헤더	SRE	제공됨
Ingress 컨트롤러 경로 압축	SRE	제공됨

4장. OPENSHIFT SDN 기본 CNI 네트워크 공급자

4.1. 프로젝트에 멀티 캐스트 사용

참고

OpenShift SDN CNI는 OpenShift Dedicated 4.14에서 더 이상 사용되지 않습니다. OpenShift Dedicated 4.15부터 네트워크 플러그인은 새 설치를 위한 옵션이 아닙니다. 향후 릴리스에서 OpenShift SDN 네트워크 플러그인은 제거될 예정이며 더 이상 지원되지 않습니다. Red Hat은 제거될 때까지 이 기능에 대한 버그 수정 및 지원을 제공하지만 이 기능은 더 이상 개선 사항을 받지 않습니다. OpenShift SDN CNI 대신 OVN Kubernetes CNI를 대신 사용할 수 있습니다.

4.1.1. 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.
- 기본적으로 네트워크 정책은 네임스페이스의 모든 연결에 영향을 미칩니다. 그러나 멀티캐스트는 네트워크 정책의 영향을 받지 않습니다. 네트워크 정책과 동일한 네임스페이스에서 멀티 캐스트를 활성화하면 **모든 네트워크 정책이 거부** 된 경우에도 항상 허용됩니다. 클러스터 관리자는 활성화하기 전에 네트워크 정책에서 멀티 캐스트에 미치는 영향을 고려해야 합니다.

OpenShift Dedicated Pod 간 멀티 캐스트 트래픽은 기본적으로 비활성화되어 있습니다. OpenShift SDN 네트워크 플러그인을 사용하는 경우 프로젝트별로 멀티 캐스트를 활성화할 수 있습니다.

networkpolicy 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.

4.1.2. Pod 간 멀티 캐스트 활성화

프로젝트의 Pod 간 멀티 캐스트를 활성화할 수 있습니다.

사전 요구 사항

- OpenShift CLI(**oc**)를 설치합니다.
- **cluster-admin** 또는 **dedicated-admin** 역할이 있는 사용자로 클러스터에 로그인해야 합니다.

절차

- 다음 명령을 실행하여 프로젝트에 대한 멀티 캐스트를 활성화합니다. 멀티 캐스트를 활성화하려는 프로젝트의 네임스페이스로 **<namespace>**를 바꿉니다.

```
$ oc annotate netnamespace <namespace> \
    netnamespace.network.openshift.io/multicast-enabled=true
```

검증

프로젝트에 멀티 캐스트가 활성화되어 있는지 확인하려면 다음 절차를 완료합니다.

1. 멀티 캐스트를 활성화한 프로젝트로 현재 프로젝트를 변경합니다. **<project>**를 프로젝트 이름으로 바꿉니다.

```
$ oc project <project>
```

2. 멀티 캐스트 수신자 역할을 할 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/ubi9
      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/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. 새 터미널 창 또는 탭에서 멀티 캐스트 리스너를 시작합니다.

   a. Pod의 IP 주소를 가져옵니다.

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

   b. 다음 명령을 입력하여 멀티 캐스트 리스너를 시작합니다.

   ```
   $ oc exec mlistener -i -t -- \
       socat UDP4-RECVFROM:30102,ip-add-membership=224.1.0.1:$POD_IP,fork
   EXEC:hostname
   ```

5. 멀티 캐스트 송신기를 시작합니다.

   a. Pod 네트워크 IP 주소 범위를 가져옵니다.

   ```
   $ CIDR=$(oc get Network.config.openshift.io cluster \
       -o jsonpath='{.status.clusterNetwork[0].cidr}')
   ```

   b. 멀티 캐스트 메시지를 보내려면 다음 명령을 입력합니다.

   ```
   $ oc exec msender -i -t -- \
       /bin/bash -c "echo | socat STDIO UDP4-
   DATAGRAM:224.1.0.1:30102,range=$CIDR,ip-multicast-ttl=64"
   ```

   멀티 캐스트가 작동하는 경우 이전 명령은 다음 출력을 반환합니다.

   ```
   mlistener
   ```

5장. OPENSHIFT DEDICATED 클러스터에 대한 네트워크 확인

네트워크 확인 검사는 기존 VPC(Virtual Private Cloud)에 OpenShift Dedicated 클러스터를 배포하거나 클러스터에 새로 추가된 서브넷을 사용하여 추가 머신 풀을 생성할 때 자동으로 실행됩니다. 이 검사에서는 네트워크 구성을 검증하고 오류를 강조 표시하므로 배포 전에 구성 문제를 해결할 수 있습니다.

네트워크 확인 검사를 수동으로 실행하여 기존 클러스터의 구성을 검증할 수도 있습니다.

5.1. OPENSHIFT DEDICATED 클러스터에 대한 네트워크 확인 이해

OpenShift Dedicated 클러스터를 기존 VPC(Virtual Private Cloud)에 배포하거나 클러스터에 새로 추가된 서브넷을 사용하여 추가 머신 풀을 생성하면 네트워크 확인이 자동으로 실행됩니다. 이를 통해 배포 전에 구성 문제를 식별하고 해결할 수 있습니다.

Red Hat OpenShift Cluster Manager를 사용하여 클러스터 설치를 준비하면 **VPC(Virtual Private Cloud) 서브넷 설정 페이지의 서브넷 ID 필드에 서브넷** ID를 입력한 후 자동 검사가 실행됩니다.

서브넷이 새로 추가된 서브넷으로 머신 풀을 추가하면 자동 네트워크 확인에서 서브넷을 확인하여 머신 풀을 프로비저닝하기 전에 네트워크 연결을 사용할 수 있는지 확인합니다.

자동 네트워크 확인이 완료되면 서비스 로그에 레코드가 전송됩니다. 레코드는 네트워크 구성 오류를 포함하여 확인 확인 결과를 제공합니다. 배포 전에 확인된 문제를 해결할 수 있으며 배포의 성공 가능성이 더 높습니다.

기존 클러스터에 대해 네트워크 확인을 수동으로 실행할 수도 있습니다. 이를 통해 구성을 변경한 후 클러스터의 네트워크 구성을 확인할 수 있습니다. 네트워크 확인 검사를 수동으로 실행하는 단계는 *네트워크 확인 수동 실행을* 참조하십시오.

5.2. 네트워크 검증 검사 범위

네트워크 확인에는 다음 요구사항 각각에 대한 검사가 포함됩니다.

- 상위 VPC(Virtual Private Cloud)가 있습니다.
- 지정된 모든 서브넷이 VPC에 속합니다.
- VPC에 **enableDnsSupport** 가 활성화되어 있습니다.
- VPC에 **enableDnsHostnames** 가 활성화되어 있습니다.
- egress는 AWS 방화벽 사전 요구 사항 섹션에 지정된 필수 도메인 및 포트 조합에서 사용할 수 있습니다.

5.3. 자동 네트워크 확인 우회

알려진 네트워크 구성 문제가 있는 OpenShift Dedicated 클러스터를 기존 VPC(Virtual Private Cloud)에 배포하려면 자동 네트워크 확인을 바이패스할 수 있습니다.

클러스터를 생성할 때 네트워크 확인을 바이패스하면 클러스터의 지원 상태가 제한됩니다. 설치 후 문제를 해결한 다음 네트워크 확인을 수동으로 실행할 수 있습니다. 제한된 지원 상태는 검증에 성공한 후 제거됩니다.

Red Hat OpenShift Cluster Manager를 사용하여 기존 VPC에 클러스터를 설치하는 경우 **VPC(Virtual Private Cloud) 서브넷 설정** 페이지에서 **Bypass network verification** 을 선택하여 자동 확인을 바이패스할 수 있습니다.

5.4. 수동으로 네트워크 확인 실행

Red Hat OpenShift Cluster Manager를 사용하여 기존 OpenShift Dedicated 클러스터에 대한 네트워크 확인 검사를 수동으로 실행할 수 있습니다.

사전 요구 사항

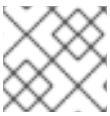
- 기존 OpenShift Dedicated 클러스터가 있어야 합니다.
- 클러스터 소유자이거나 클러스터 편집기 역할이 있습니다.

절차

1. OpenShift Cluster Manager 로 이동하여 클러스터를 선택합니다.
2. **작업** 드롭다운 메뉴에서 네트워크 **확인을** 선택합니다.

6장. 클러스터 전체 프록시 구성

기존 VPC(Virtual Private Cloud)를 사용하는 경우 OpenShift Dedicated 클러스터 설치 중 또는 클러스터가 설치된 후 클러스터 전체 프록시를 구성할 수 있습니다. 프록시를 활성화하면 코어 클러스터 구성 요소가 인터넷에 대한 직접 액세스가 거부되지만 프록시는 사용자 워크로드에 영향을 미치지 않습니다.



참고

클라우드 공급자 API에 대한 호출을 포함하여 시스템 송신 트래픽만 프록시됩니다.

CCO(Customer Cloud Subscription) 모델을 사용하는 OpenShift Dedicated 클러스터에 대해서만 프록시를 활성화할 수 있습니다.

클러스터 전체 프록시를 사용하는 경우 클러스터에 대한 프록시의 가용성을 유지 관리해야 합니다. 프록시를 사용할 수 없게 되면 클러스터의 상태 및 지원 가능성에 영향을 미칠 수 있습니다.

6.1. 클러스터 전체 프록시 구성을 위한 사전 요구 사항

클러스터 전체 프록시를 구성하려면 다음 요구 사항을 충족해야 합니다. 이러한 요구 사항은 설치 중 또는 설치 후 프록시를 구성할 때 유효합니다.

일반 요구 사항

- 클러스터 소유자입니다.
- 계정에는 충분한 권한이 있습니다.
- 클러스터의 기존 VPC(Virtual Private Cloud)가 있습니다.
- 클러스터의 CCS(Customer Cloud Subscription) 모델을 사용하고 있습니다.
- 프록시는 클러스터의 VPC 및 VPC의 프라이빗 서브넷에 액세스할 수 있습니다. 이 프록시는 클러스터의 VPC 및 VPC의 프라이빗 서브넷에서도 액세스할 수 있습니다.
- VPC 끝점에 다음 끝점을 추가했습니다.
  - **ec2.<aws_region>.amazonaws.com**
  - **elasticloadbalancing.<aws_region>.amazonaws.com**
  - **s3.<aws_region>.amazonaws.com**
    이러한 끝점은 노드에서 AWS EC2 API로 요청을 완료하는 데 필요합니다. 프록시는 노드 수준이 아닌 컨테이너 수준에서 작동하기 때문에 이러한 요청을 AWS 사설 네트워크를 통해 AWS EC2 API로 라우팅해야 합니다. 프록시 서버의 허용 목록에 EC2 API의 공용 IP 주소를 추가하는 것만으로는 충분하지 않습니다.

중요

클러스터 전체 프록시를 사용하는 경우 **게이트웨이** 유형으로 **s3.<aws_region>.amazonaws.com** 끝점을 구성해야 합니다.

네트워크 요구 사항

- 프록시에서 송신 트래픽을 다시 암호화하는 경우 도메인 및 포트 조합에 대한 제외를 생성해야 합니다. 다음 표에서는 이러한 예외에 대한 지침을 제공합니다.

- 프록시는 다음 OpenShift URL을 다시 암호화하도록 제외해야 합니다.

address	프로토콜/포트	함수
observatorium-mst.api.openshift.com	https/443	필수 항목입니다. Managed OpenShift별 Telemetry에 사용됩니다.
sso.redhat.com	https/443	https://cloud.redhat.com/openshift 사이트에서는 sso.redhat.com의 인증을 사용하여 클러스터 풀 시크릿을 다운로드하고 Red Hat SaaS 솔루션을 사용하여 서브스크립션, 클러스터 인벤토리 및 관련 보고를 원활하게 모니터링할 수 있습니다.

- 프록시는 다음 사이트 안정성 엔지니어링(SRE) 및 관리 URL을 재암호화해야 합니다.

address	프로토콜/포트	함수
***.osdsecuritylogs.splunkcloud.com** 또는 **inputs1.osdsecuritylogs.splunkcloud.cominputs2**.osdsecuritylogs.complunkcloud.com**inputs4.osdsecuritylogs.splunkcloud.cominputs5.osdsecuritylogs.splunkcloud.cominputs6.osdsecuritylogs.splunkcloud.com**input6.**osdsecuritylogs.splunkcloud.comcomputes.cloudoscomstoredcloud-complunks.com computesfcloudoscomcomputecomfcloudos의 입력**	tcp/9997	splunk-forwarder-operator에서 로그 기반 경고에 사용할 로그 전달 끝점으로 사용합니다.
http-inputs-osdsecuritylogs.splunkcloud.com	https/443	splunk-forwarder-operator에서 로그 기반 경고에 사용할 로그 전달 끝점으로 사용합니다.

추가 리소스

- CCO(Customer Cloud Subscription) 모델을 사용하는 OpenShift Dedicated 클러스터의 설치 사전 요구 사항은 AWS의 Customer Cloud Subscription 또는 GCP의 Customer Cloud 서브스크립션을 참조하십시오.

6.2. 추가 신뢰 번들에 대한 책임

추가 신뢰 번들을 제공하는 경우 다음 요구 사항을 담당합니다.

- 추가 신뢰 번들의 콘텐츠가 유효한지 확인
- 추가 신뢰 번들에 포함된 중간 인증서를 포함하여 인증서가 만료되지 않았는지 확인
- 만료 추적 및 추가 신뢰 번들에 포함된 인증서에 필요한 갱신 수행
- 업데이트된 추가 신뢰 번들로 클러스터 구성 업데이트

6.3. 설치 중 프록시 구성

기존 VPC(Virtual Private Cloud) 클러스터에 CCO(Customer Cloud Subscription) 클러스터를 사용하여 OpenShift Dedicated를 설치할 때 HTTP 또는 HTTPS 프록시를 구성할 수 있습니다. Red Hat OpenShift Cluster Manager를 사용하여 설치 중에 프록시를 구성할 수 있습니다.

6.4. OPENSHIFT CLUSTER MANAGER를 사용하여 설치 중에 프록시 구성

기존 VPC(Virtual Private Cloud)에 OpenShift Dedicated 클러스터를 설치하는 경우 Red Hat OpenShift Cluster Manager를 사용하여 설치 중에 클러스터 전체 HTTP 또는 HTTPS 프록시를 활성화할 수 있습니다. CCS(Customer Cloud Subscription) 모델을 사용하는 클러스터에 대해서만 프록시를 활성화할 수 있습니다.

설치하기 전에 클러스터를 설치하는 VPC에서 프록시에 액세스할 수 있는지 확인해야 합니다. VPC의 프라이빗 서브넷에서도 프록시에 액세스할 수 있어야 합니다.

OpenShift Cluster Manager를 사용하여 설치 중에 클러스터 전체 프록시를 구성하는 방법에 대한 자세한 단계는 CCS를 사용하여 *AWS에서 클러스터 생성* 또는 *CCS 를 사용하여 GCP에서 클러스터 생성*을 참조하십시오.

추가 리소스

- CCS를 사용하여 AWS에서 클러스터 생성
- CCS를 사용하여 GCP에 클러스터 생성

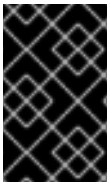
6.5. 설치 후 프록시 구성

CCO(Customer Cloud Subscription) 클러스터를 사용하여 OpenShift Dedicated를 기존 VPC(Virtual Private Cloud)에 설치한 후 HTTP 또는 HTTPS 프록시를 구성할 수 있습니다. Red Hat OpenShift Cluster Manager를 사용하여 설치 후 프록시를 구성할 수 있습니다.

6.6. OPENSHIFT CLUSTER MANAGER를 사용하여 설치 후 프록시 구성

Red Hat OpenShift Cluster Manager를 사용하여 VPC(Virtual Private Cloud)의 기존 OpenShift Dedicated 클러스터에 클러스터 전체 프록시 구성을 추가할 수 있습니다. CCS(Customer Cloud Subscription) 모델을 사용하는 클러스터에 대해서만 프록시를 활성화할 수 있습니다.

OpenShift Cluster Manager를 사용하여 기존 클러스터 전체 프록시 구성을 업데이트할 수도 있습니다. 예를 들어 프록시의 인증 기관이 만료되면 프록시의 네트워크 주소를 업데이트하거나 추가 신뢰 번들을 교체해야 할 수 있습니다.



중요

클러스터는 컨트롤 플레인 및 컴퓨팅 노드에 프록시 설정을 적용합니다. 구성을 적용하는 동안 각 클러스터 노드는 일시적으로 스케줄링할 수 없는 상태로 배치되어 워크로드를 드레인합니다. 각 노드는 프로세스의 일부로 재시작됩니다.

사전 요구 사항

- CCO(Customer Cloud Subscription) 모델을 사용하는 OpenShift Dedicated 클러스터가 있어야 합니다.
- 클러스터는 VPC에 배포되어 있습니다.

절차

1. OpenShift Cluster Manager 로 이동하여 클러스터를 선택합니다.
2. **네트워킹** 페이지의 **VPC(Virtual Private Cloud)** 섹션에서 **Edit cluster-wide proxy** 를 클릭합니다.
3. **클러스터 전체 프록시 편집 페이지에서 프록시** 구성 세부 정보를 입력합니다.
   a. 다음 필드 중 하나 이상에 값을 입력합니다.
      - 유효한 **HTTP 프록시 URL** 을 지정합니다.
      - 유효한 **HTTPS 프록시 URL** 을 지정합니다.
      - **추가 신뢰 번들** 필드에서 PEM 인코딩 X.509 인증서 번들을 제공합니다. 기존 신뢰 번들 파일을 교체하는 경우 **파일 교체를** 선택하여 필드를 확인합니다. 번들은 클러스터 노드의 신뢰할 수 있는 인증서 저장소에 추가됩니다. 프록시의 ID 인증서가 RHCOS(Red Hat Enterprise Linux CoreOS) 신뢰 번들의 기관에서 서명되지 않는 한 TLS 지정 프록시를 사용하는 경우 추가 신뢰 번들 파일이 필요합니다. 이 요구 사항은 프록시가 투명했는지 또는 **http-proxy** 인수 및 **https-proxy** 인수를 사용하여 명시적 구성이 필요한지 여부와 관계없이 적용됩니다.
   b. **Confirm** 을 클릭합니다.

검증

- **네트워킹** 페이지의 **VPC(Virtual Private Cloud)** 섹션에서 클러스터의 프록시 구성이 예상대로 구성되어 있는지 확인합니다.

7장. CIDR 범위 정의

다음 CIDR 범위에 대해 겹치지 않는 범위를 지정해야 합니다.

참고

머신 CIDR 범위는 클러스터를 생성한 후에는 변경할 수 없습니다.

서브넷 CIDR 범위를 지정할 때 서브넷 CIDR 범위가 정의된 Machine CIDR 내에 있는지 확인합니다. 서브넷 CIDR 범위가 클러스터가 호스팅되는 플랫폼에 따라 모든 의도한 워크로드에 대해 충분한 IP 주소를 허용하는지 확인해야 합니다.

중요

OpenShift Dedicated 4.14 이상 버전의 기본 네트워크 공급자인 OVN-Kubernetes는 내부적으로 다음 IP 주소 범위를 사용합니다.
100.64.0.0/16,**169.254.169.0/29**,**100.88.0.0/16**,**fd98::/64**,**fd69::/125**, **fd69::** /125 . 클러스터에서 OVN-Kubernetes를 사용하는 경우 클러스터 또는 인프라의 다른 CIDR 정의에 이러한 IP 주소 범위를 포함하지 마십시오.

7.1. MACHINE CIDR

CIDR(Machine classless inter-domain routing) 필드에서 시스템 또는 클러스터 노드의 IP 주소 범위를 지정해야 합니다. 이 범위는 VPC(가상 프라이빗 클라우드) 서브넷에 대한 모든 CIDR 주소 범위를 포함해야 합니다. 서브넷이 연속되어야 합니다. 단일 가용성 영역 배포에 대해 서브넷 접두사 **/25** 를 사용하는 최소 128 주소 범위가 지원됩니다. 여러 가용 영역을 사용하는 배포에는 서브넷 접두사 **/24** 를 사용하는 최소 주소 범위 256 주소가 지원됩니다.

기본값은 **10.0.0.0/16** 입니다. 이 범위는 연결된 네트워크와 충돌해서는 안 됩니다.

7.2. SERVICE CIDR

Service CIDR 필드에서 서비스의 IP 주소 범위를 지정해야 합니다. 클러스터 간에 주소 블록이 동일하지만 필수는 아닙니다. 이는 IP 주소 충돌을 생성하지 않습니다. 범위는 워크로드를 수용할 수 있을 만큼 충분히 커야 합니다. address 블록은 클러스터 내에서 액세스한 외부 서비스와 겹치지 않아야 합니다. 기본값은 **172.30.0.0/16**입니다.

7.3. POD CIDR

Pod CIDR 필드에서 Pod의 IP 주소 범위를 지정해야 합니다.

클러스터 간에 주소 블록이 동일하지만 필수는 아닙니다. 이는 IP 주소 충돌을 생성하지 않습니다. 범위는 워크로드를 수용할 수 있을 만큼 충분히 커야 합니다. address 블록은 클러스터 내에서 액세스한 외부 서비스와 겹치지 않아야 합니다. 기본값은 **10.128.0.0/14** 입니다.

7.4. 호스트 접두사

Host Prefix 필드에서 개별 머신에 예약된 Pod에 할당된 서브넷 접두사 길이를 지정해야 합니다. 호스트 접두사는 각 머신의 Pod IP 주소 풀을 결정합니다.

예를 들어 호스트 접두사를 **/23** 으로 설정하면 Pod CIDR 주소 범위의 **/23** 서브넷이 각 시스템에 할당됩니다. 기본값은 **/23** 입니다. 이로 인해 512개의 클러스터 노드와 노드당 512개의 포드가 허용됩니다(두 가지 모두 최대 지원 범위를 초과함).

8장. 네트워크 보안

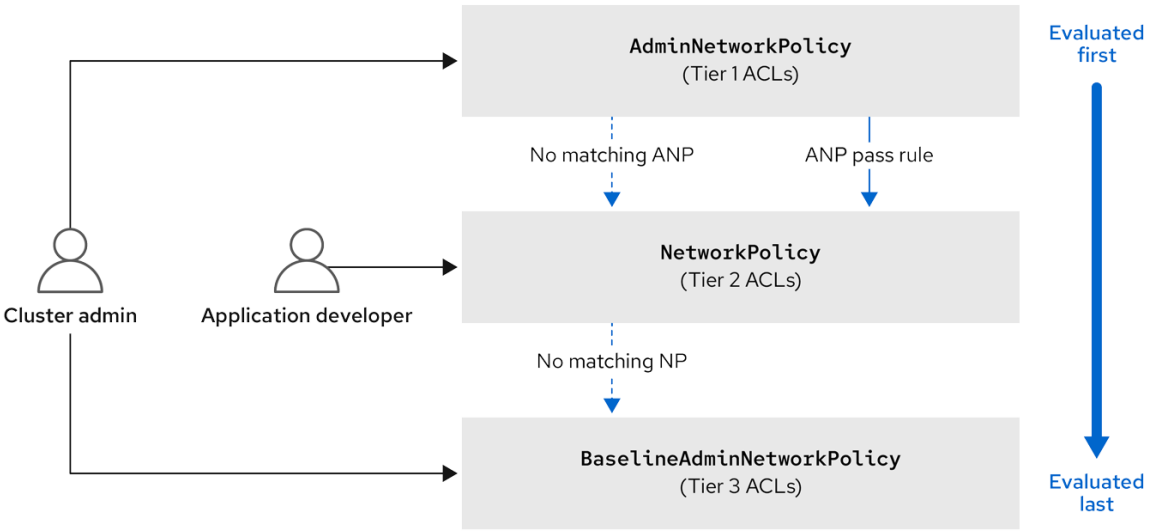
8.1. 네트워크 정책 API 이해

Kubernetes는 사용자가 네트워크 보안을 적용하는 데 사용할 수 있는 두 가지 기능을 제공합니다. 사용자가 네트워크 정책을 적용할 수 있는 한 가지 기능은 애플리케이션 개발자 및 네임스페이스 테넌트가 네임스페이스 범위 정책을 생성하여 네임스페이스를 보호하기 위해 주로 설계된 **NetworkPolicy** API입니다.

두 번째 기능은 **AdminNetworkPolicy** (ANP) API와 **Baseline AdminNetworkPolicy** (BANP) API의 두 API로 구성된 AdminNetworkPolicy입니다. ANP 및 BANP는 클러스터 범위 정책을 생성하여 클러스터 및 네트워크 관리자가 전체 클러스터를 보호하도록 설계되었습니다. 클러스터 관리자는 ANPs를 사용하여 **NetworkPolicy** 오브젝트보다 우선하는 복구 불가능한 정책을 적용할 수 있습니다. 관리자는 BANP를 사용하여 필요한 경우 **NetworkPolicy** 오브젝트를 사용하는 사용자가 덮어쓸 수 있는 선택적 클러스터 범위 네트워크 정책 규칙을 설정하고 적용할 수 있습니다. ANP, BANP 및 네트워크 정책을 함께 사용하면 관리자가 클러스터를 보호하는 데 사용할 수 있는 완전한 다중 테넌트 격리를 수행할 수 있습니다.

OpenShift Dedicated의 OVN-Kubernetes CNI는 ACL(Access Control List) 계층을 사용하여 이러한 네트워크 정책을 구현하여 이를 평가하고 적용합니다. ACL은 계층 1에서 계층 3까지 내림차순으로 평가됩니다.

계층 1은 **관리NetworkPolicy** (ANP) 오브젝트를 평가합니다. 계층 2는 **NetworkPolicy** 오브젝트를 평가합니다. 계층 3은 **BaselineAdminNetworkPolicy** (BANP) 오브젝트를 평가합니다.



ANP가 먼저 평가됩니다. 일치 항목이 ANP **허용** 또는 **거부** 규칙인 경우 클러스터의 기존 **NetworkPolicy** 및 BANP(**BaselineAdminNetworkPolicy**) 오브젝트는 평가에서 건너뜁니다. 일치 항목이 ANP **통과** 규칙인 경우 평가는 ACL의 계층 1에서 **NetworkPolicy** 정책이 평가되는 계층 2로 이동합니다. **NetworkPolicy** 가 트래픽과 일치하지 않으면 평가는 계층 2 ACL에서 BANP가 평가되는 계층 3 ACL로 이동합니다.

8.1.1. AdminNetworkPolicy와 NetworkPolicy 사용자 정의 리소스의 주요 차이점

다음 표에서는 클러스터 범위 **AdminNetworkPolicy** API와 네임스페이스 범위 **NetworkPolicy** API 간의 주요 차이점을 설명합니다.

정책 요소	AdminNetworkPolicy	NetworkPolicy
적용 가능한 사용자	클러스터 관리자 또는 이에 상응하는	네임스페이스 소유자
범위	Cluster	namespaced
트래픽 드롭	명시적 **거부** 작업 세트에서 규칙으로 지원됩니다.	정책 생성 시 암시적 **거부** 격리를 통해 지원됩니다.
트래픽 위임	규칙으로 **Pass** 작업 세트에서 지원됩니다.	해당 없음
트래픽 허용	명시적 **허용** 작업 세트를 규칙으로 사용하여 지원됩니다.	모든 규칙에 대한 기본 작업은 허용하는 것입니다.
정책 내에서 규칙 우선순위	ANP 내에 표시되는 순서에 따라 달라집니다. 규칙의 위치가 높을수록 우선순위가 높습니다.	규칙이 추가됩니다.
정책 우선순위	ANPs 중 **우선순위** 필드는 평가 순서를 설정합니다. 정책 우선 순위가 높은 우선 순위 번호가 적습니다.	정책 간에는 정책 순서가 없습니다.
기능 우선 순위	계층 1 ACL 및 BANP를 통해 먼저 평가되는 것은 계층 3 ACL을 통해 마지막으로 평가됩니다.	ANP 및 BANP 전에 시행되며 ACL의 계층 2에서 평가됩니다.
Pod 선택 일치	네임스페이스에 다른 규칙을 적용할 수 있습니다.	단일 네임스페이스의 Pod에 다른 규칙을 적용할 수 있습니다.
클러스터 송신 트래픽	**노드** 및 **네트워크** 피어를 통해 지원	허용되는 CIDR 구문과 함께 **ipBlock** 필드를 통해 지원됩니다.
클러스터 인그레스 트래픽	지원되지 않음	지원되지 않음
FQDN(정규화된 도메인 이름) 피어 지원	지원되지 않음	지원되지 않음
네임스페이스 선택기	**namespaces.matchLabels** 필드를 사용하여 고급 네임스페이스 선택 지원	**namespaceSelector** 필드를 사용하여 라벨 기반 네임스페이스 선택 지원

8.2. 네트워크 정책

8.2.1. 네트워크 정책 정의

개발자는 클러스터의 Pod로 트래픽을 제한하는 네트워크 정책을 정의할 수 있습니다.

8.2.1.1. 네트워크 정책 정의

Kubernetes 네트워크 정책을 지원하는 네트워크 플러그인을 사용하는 클러스터에서 네트워크 격리는 **NetworkPolicy** 오브젝트에 의해 완전히 제어됩니다. OpenShift Dedicated 4에서 OpenShift SDN은 기본 네트워크 격리 모드에서 네트워크 정책 사용을 지원합니다.

주의

네트워크 정책은 호스트 네트워크 네임스페이스에 적용되지 않습니다. 호스트 네트워킹이 활성화된 Pod는 네트워크 정책 규칙의 영향을 받지 않습니다. 그러나 호스트 네트워크 pod에 연결하는 Pod는 네트워크 정책 규칙의 영향을 받을 수 있습니다.

네트워크 정책은 localhost 또는 상주 노드의 트래픽을 차단할 수 없습니다.

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.

네트워크 정책은 TCP, UDP, ICMP 및 SCTP 프로토콜에만 적용됩니다. 다른 프로토콜은 영향을 받지 않습니다.

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 Dedicated Ingress 컨트롤러의 연결만 허용합니다.
  프로젝트에서 OpenShift Dedicated Ingress 컨트롤러의 연결만 허용하도록 하려면 다음 **NetworkPolicy** 오브젝트를 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:

중요

동일한 네임스페이스에 있는 **hostNetwork** Pod의 수신 연결을 허용하려면 **allow-from-hostnetwork** 정책을 **allow-same-namespace** 정책과 함께 적용해야 합니다.

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.

8.2.1.1.1. allow-from-router 네트워크 정책 사용

다음 **NetworkPolicy** 를 사용하여 라우터 구성에 관계없이 외부 트래픽을 허용합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-router
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: "" (1)
  podSelector: {}
  policyTypes:
  - Ingress
```

(1) **policy-group.network.openshift.io/ingress:""** 레이블은 OpenShift-SDN 및 OVN-Kubernetes를 모두 지원합니다.

8.2.1.1.2. allow-from-hostnetwork 네트워크 정책 사용

다음 **allow-from-hostnetwork NetworkPolicy** 오브젝트를 추가하여 호스트 네트워크 Pod에서 트래픽을 전달합니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-hostnetwork
```

```
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/host-network: ""
  podSelector: {}
  policyTypes:
  - Ingress
```

8.2.1.2. OpenShift SDN을 사용한 네트워크 정책 최적화

네트워크 정책을 사용하여 네임스페이스 내의 라벨에 따라 서로 다른 포드를 분리합니다.

NetworkPolicy 오브젝트를 단일 네임스페이스에서 개별 포드의 많은 수에 적용하는 것은 비효율적입니다. 포드 라벨은 IP 주소 수준에 존재하지 않으므로 네트워크 정책은 **podSelector**로 선택한 모든 포드 간에 가능한 모든 링크에 대한 별도의 OVS(Open vSwitch) 흐름 규칙을 생성합니다.

예를 들어 **NetworkPolicy** 오브젝트 내의 spec **podSelector** 및 ingress **podSelector**가 각각 200개의 포드와 일치하는 경우 40,000(200*200)개의 OVS 흐름 규칙이 생성됩니다. 이 경우 노드가 느려질 수 있습니다.

네트워크 정책을 설계할 때 다음 지침을 참조하십시오.

- 분리해야 하는 포드 그룹을 포함하도록 네임스페이스를 사용하여 OVS 흐름 규칙의 수를 줄입니다.
  namespaceSelector 또는 빈 **podSelector**를 사용하여 전체 네임스페이스를 선택하는 **NetworkPolicy** 오브젝트는 네임스페이스의 VXLAN 가상 네트워크 ID(VNID)와 일치하는 단일 OVS 흐름 규칙만 생성합니다.

- 원래 네임스페이스에서 분리할 필요가 없는 포드를 유지하고, 분리해야 하는 포드를 하나 이상의 네임스페이스로 이동합니다.

- 분리된 포드에서 허용하려는 특정 트래픽을 허용하도록 추가 대상의 네임스페이스 간 네트워크 정책을 생성합니다.

8.2.1.3. OVN-Kubernetes 네트워크 플러그인을 사용하여 네트워크 정책 최적화

네트워크 정책을 설계할 때 다음 지침을 참조하십시오.

- 동일한 **spec.podSelector** 사양이 있는 네트워크 정책의 경우 수신 또는 송신 규칙의 서브 세트가 있는 여러 네트워크 정책보다 여러 **수신** 또는 **송신** 규칙이 있는 하나의 네트워크 정책을 사용하는 것이 더 효율적입니다.

-
  podSelector 또는 namespaceSelector 사양을 기반으로 하는 모든 수신 또는 송신 규칙은 수신 또는 송신 규칙에서 선택한 네트워크 정책 + Pod 수에 비례하여 OVS 흐름 수를 생성합니다. 따라서 모든 Pod에 대해 개별 규칙을 생성하는 대신 하나의 규칙에서 필요한 만큼의 Pod를 선택할 수 있는 podSelector 또는 namespaceSelector 사양을 사용하는 것이 좋습니다.

  예를 들어 다음 정책에는 두 개의 규칙이 있습니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
  - from:
    - podSelector:
        matchLabels:
          role: backend
```

다음 정책은 다음과 같은 두 가지 규칙을 나타냅니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector: {}
  ingress:
  - from:
    - podSelector:
        matchExpressions:
        - {key: role, operator: In, values: [frontend, backend]}
```

spec.podSelector 사양에 동일한 지침이 적용됩니다. 다른 네트워크 정책에 대해 동일한 수신 또는 송신 규칙이 있는 경우 공통 **spec.podSelector** 사양을 사용하여 하나의 네트워크 정책을 생성하는 것이 더 효과적일 수 있습니다. 예를 들어 다음 두 정책에는 다른 규칙이 있습니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy1
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
```

```
  name: policy2
spec:
  podSelector:
    matchLabels:
      role: client
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

다음 네트워크 정책은 다음과 같은 두 가지 규칙을 나타냅니다.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy3
spec:
  podSelector:
    matchExpressions:
    - {key: role, operator: In, values: [db, client]}
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

이 최적화는 여러 개의 선택기만 하나의 선택기로 표시되는 경우에만 적용할 수 있습니다. 선택기가 다른 레이블을 기반으로 하는 경우 이 최적화를 적용하지 못할 수 있습니다. 이러한 경우 네트워크 정책 최적화에 새로운 레이블을 적용하는 것이 좋습니다.

8.2.1.4. 다음 단계

- 네트워크 정책 생성

8.2.2. 네트워크 정책 생성

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

8.2.2.1. NetworkPolicy 오브젝트 예

다음은 예제 NetworkPolicy 오브젝트에 대한 주석입니다.

```
kind: NetworkPolicy
```

```
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

1 NetworkPolicy 오브젝트의 이름입니다.

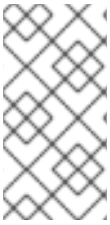
2 정책이 적용되는 Pod를 설명하는 선택기입니다. 정책 오브젝트는 NetworkPolicy 오브젝트를 정의하는 프로젝트에서 Pod만 선택할 수 있습니다.

3 정책 오브젝트가 수신 트래픽을 허용하는 Pod와 일치하는 선택기입니다. 선택기는 NetworkPolicy와 동일한 네임스페이스의 Pod와 일치합니다.

4 트래픽을 수락할 하나 이상의 대상 포트 목록입니다.

8.2.2.2. CLI를 사용하여 네트워크 정책 생성

클러스터의 네임스페이스에서 허용된 수신 또는 송신 네트워크 트래픽을 설명하는 세분화된 규칙을 정의하기 위해 네트워크 정책을 생성할 수 있습니다.



참고

cluster-admin 역할로 사용자로 로그인하는 경우 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

사전 요구 사항

- 클러스터는 **mode: NetworkPolicy** 로 설정된 **OVN-Kubernetes** 네트워크 플러그인 또는 **OpenShift SDN** 네트워크 플러그인과 같은 **NetworkPolicy** 오브젝트를 지원하는 네트워크 플러그인을 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

절차

1. 다음과 같이 정책 규칙을 생성합니다.

   a. **<policy_name>.yaml** 파일을 생성합니다.

   ```
   $ touch <policy_name>.yaml
   ```

   다음과 같습니다.

   <policy_name>

   네트워크 정책 파일 이름을 지정합니다.

   b. 방금 만든 파일에서 다음 예와 같이 네트워크 정책을 정의합니다.

   모든 네임스페이스의 모든 **Pod**에서 수신 거부

   이는 다른 네트워크 정책 구성에서 허용하는 크로스 포드 트래픽 이외의 모든 크로스 포드 네트워킹을 차단하는 기본 정책입니다.

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

```
  podSelector: {}
  policyTypes:
  - Ingress
  ingress: []
```

동일한 네임 스페이스에 있는 모든 **Pod**의 수신 허용

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

특정 네임스페이스에서 하나의 **Pod**로 들어오는 트래픽 허용

이 정책을 사용하면 **namespace-y** 에서 실행되는 **Pod**의 **pod-a** 레이블이 지정된 **Pod** 로의 트래픽을 수행할 수 있습니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-traffic-pod
spec:
  podSelector:
   matchLabels:
     pod: pod-a
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: namespace-y
```

2. 다음 명령을 실행하여 네트워크 정책 오브젝트를 생성합니다.

```
$ oc apply -f <policy_name>.yaml -n <namespace>
```

다음과 같습니다.

<policy_name>

네트워크 정책 파일 이름을 지정합니다.

<namespace>

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

출력 예

```
networkpolicy.networking.k8s.io/deny-by-default created
```

참고

cluster-admin 권한을 사용하여 웹 콘솔에 로그인하는 경우 **YAML** 또는 웹 콘솔의 양식에서 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

8.2.2.3. 기본 거부 모든 네트워크 정책 생성

이는 배포된 다른 네트워크 정책 구성에서 허용하는 네트워크 트래픽 이외의 모든 크로스-**Pod** 네트워킹을 차단하는 기본 정책입니다. 이 절차에서는 기본 거부 정책을 적용합니다.

참고

cluster-admin 역할로 사용자로 로그인하는 경우 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

사전 요구 사항

- 클러스터는 **mode: NetworkPolicy** 로 설정된 **OVN-Kubernetes** 네트워크 플러그인 또는 **OpenShift SDN** 네트워크 플러그인과 같은 **NetworkPolicy** 오브젝트를 지원하는 네트워크 플러그인을 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.

- OpenShift CLI(oc)를 설치합니다.
- admin 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

절차

1. 모든 네임스페이스의 모든 Pod에서 수신을 거부하는 거부-별-기본 정책을 정의하는 다음 YAML을 생성합니다. YAML을 deny-by-default.yaml 파일에 저장합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: deny-by-default
  namespace: default 1
spec:
  podSelector: {} 2
  ingress: [] 3
```

1. namespace: default 는 이 정책을 default 네임스페이스에 배포합니다.
2. podSelector: 비어 있습니다. 즉, 모든 Pod와 일치합니다. 따라서 이 정책은 default 네임스페이스의 모든 Pod에 적용됩니다.
3. Ingress 규칙이 지정되지 않았습니다. 이로 인해 들어오는 트래픽이 모든 Pod로 삭제됩니다.

2. 다음 명령을 입력하여 정책을 적용합니다.

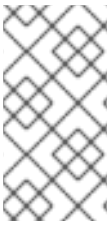
```
$ oc apply -f deny-by-default.yaml
```

출력 예

```
networkpolicy.networking.k8s.io/deny-by-default created
```

8.2.2.4. 외부 클라이언트의 트래픽을 허용하는 네트워크 정책 생성

deny-by-default 정책을 사용하여 외부 클라이언트에서 라벨이 **app=web** 인 **Pod**로의 트래픽을 허용하는 정책을 구성할 수 있습니다.



참고

cluster-admin 역할로 사용자로 로그인하는 경우 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

다음 절차에 따라 공용 인터넷의 외부 서비스를 직접 또는 로드 밸런서를 사용하여 **Pod**에 액세스할 수 있는 정책을 구성합니다. 트래픽은 **app=web** 레이블이 있는 **Pod**에만 허용됩니다.

사전 요구 사항

- 클러스터는 **mode: NetworkPolicy** 로 설정된 **OVN-Kubernetes** 네트워크 플러그인 또는 **OpenShift SDN** 네트워크 플러그인과 같은 **NetworkPolicy** 오브젝트를 지원하는 네트워크 플러그인을 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

절차

1. 공용 인터넷의 트래픽을 직접 또는 로드 밸런서를 사용하여 **Pod**에 액세스할 수 있는 정책을 생성합니다. **web-allow-external.yaml** 파일에 **YAML**을 저장합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-external
  namespace: default
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

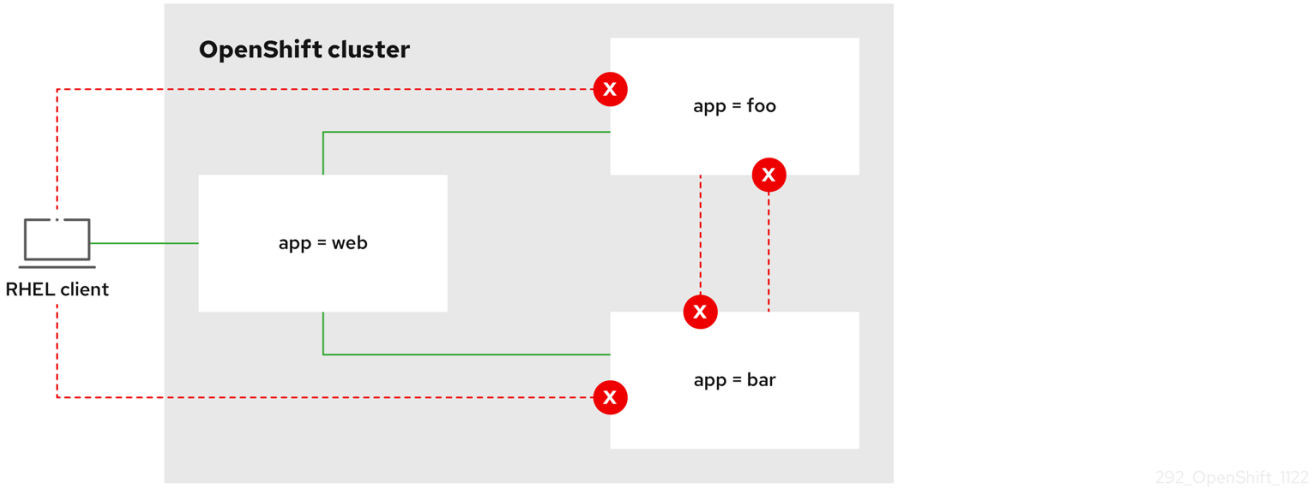
2. 다음 명령을 입력하여 정책을 적용합니다.

```
$ oc apply -f web-allow-external.yaml
```

출력 예

```
networkpolicy.networking.k8s.io/web-allow-external created
```

이 정책은 다음 다이어그램에 설명된 외부 트래픽을 포함하여 모든 리소스의 트래픽을 허용합니다.



8.2.2.5. 모든 네임스페이스에서 애플리케이션으로의 트래픽을 허용하는 네트워크 정책 생성

참고

cluster-admin 역할로 사용자로 로그인하는 경우 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

다음 절차에 따라 모든 네임스페이스의 모든 **Pod**에서 특정 애플리케이션으로의 트래픽을 허용하는 정책을 구성합니다.

사전 요구 사항

- 클러스터는 **mode: NetworkPolicy** 로 설정된 **OVN-Kubernetes** 네트워크 플러그인 또는 **OpenShift SDN** 네트워크 플러그인과 같은 **NetworkPolicy** 오브젝트를 지원하는 네트워크 플러그인을 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

프로세스

1. 모든 네임스페이스의 모든 **Pod**에서 특정 애플리케이션으로의 트래픽을 허용하는 정책을 생성합니다. **YAML**을 **web-allow-all-namespaces.yaml** 파일에 저장합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-all-namespaces
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector: {} 2
```

1

2

모든 네임스페이스의 모든 Pod를 선택합니다.

참고

기본적으로 namespaceSelector 를 지정하지 않으면 정책이 네트워크 정책이 배포된 네임스페이스에서만 트래픽을 허용하는 네임스페이스를 선택하지 않습니다.

2. 다음 명령을 입력하여 정책을 적용합니다.

```
$ oc apply -f web-allow-all-namespaces.yaml
```

출력 예

```
networkpolicy.networking.k8s.io/web-allow-all-namespaces created
```

검증

1. 다음 명령을 입력하여 default 네임스페이스에서 웹 서비스를 시작합니다.

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```

2. 다음 명령을 실행하여 보조 네임스페이스에 alpine 이미지를 배포하고 쉘을 시작합니다.

```
$ oc run test-$RANDOM --namespace=secondary --rm -i -t --image=alpine -- sh
```

3. 쉘에서 다음 명령을 실행하고 요청이 허용되는지 확인합니다.

```
# wget -qO- --timeout=2 http://web.default
```

예상 출력

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

8.2.2.6. 네임스페이스에서 애플리케이션으로의 트래픽을 허용하는 네트워크 정책 생성

참고

cluster-admin 역할로 사용자로 로그인하는 경우 클러스터의 모든 네임스페이스에서 네트워크 정책을 생성할 수 있습니다.

다음 절차에 따라 특정 네임스페이스에서 **app=web** 레이블이 있는 **Pod**로의 트래픽을 허용하는 정책을 구성합니다. 다음 작업을 수행할 수 있습니다.

- 프로덕션 워크로드가 배포된 네임스페이스로만 프로덕션 데이터베이스로 트래픽을 제한합니다.
- 특정 네임스페이스에 배포된 모니터링 툴을 활성화하여 현재 네임스페이스에서 메트릭을 스크랩할 수 있습니다.

사전 요구 사항

- 클러스터는 **mode: NetworkPolicy** 로 설정된 **OVN-Kubernetes** 네트워크 플러그인 또는 **OpenShift SDN** 네트워크 플러그인과 같은 **NetworkPolicy** 오브젝트를 지원하는 네트워크 플러그인을 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.

- **OpenShift CLI(oc)**를 설치합니다.

- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

- 네트워크 정책이 적용되는 네임스페이스에서 작업하고 있습니다.

프로세스

1. **purpose=production** 레이블이 있는 특정 네임스페이스의 모든 **Pod**의 트래픽을 허용하는 정책을 생성합니다. **YAML**을 **web-allow-prod.yaml** 파일에 저장합니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-prod
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web 1
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production 2
```

1. 정책을 **default** 네임스페이스의 **app:web Pod**에만 적용합니다.

2. **purpose=production** 레이블이 있는 네임스페이스의 **Pod**로만 트래픽을 제한합니다.

2. 다음 명령을 입력하여 정책을 적용합니다.

```
$ oc apply -f web-allow-prod.yaml
```

출력 예

```
networkpolicy.networking.k8s.io/web-allow-prod created
```

검증

1. 다음 명령을 입력하여 **default** 네임스페이스에서 웹 서비스를 시작합니다.

```
$ oc run web --namespace=default --image=nginx --labels="app=web" --expose --port=80
```

2. 다음 명령을 실행하여 **prod** 네임스페이스를 생성합니다.

```
$ oc create namespace prod
```

3. 다음 명령을 실행하여 **prod** 네임스페이스에 레이블을 지정합니다.

```
$ oc label namespace/prod purpose=production
```

4. 다음 명령을 실행하여 **dev** 네임스페이스를 생성합니다.

```
$ oc create namespace dev
```

5. 다음 명령을 실행하여 **dev** 네임스페이스에 레이블을 지정합니다.

```
$ oc label namespace/dev purpose=testing
```

6. 다음 명령을 실행하여 **dev** 네임스페이스에 **alpine** 이미지를 배포하고 쉘을 시작합니다.

```
$ oc run test-$RANDOM --namespace=dev --rm -i -t --image=alpine -- sh
```

7. 쉘에서 다음 명령을 실행하고 요청이 차단되었는지 확인합니다.

```
# wget -qO- --timeout=2 http://web.default
```

예상 출력

```
wget: download timed out
```

8. 다음 명령을 실행하여 **prod** 네임스페이스에 **alpine** 이미지를 배포하고 쉘을 시작합니다.

```
$ oc run test-$RANDOM --namespace=prod --rm -i -t --image=alpine -- sh
```

9. 쉘에서 다음 명령을 실행하고 요청이 허용되는지 확인합니다.

```
# wget -qO- --timeout=2 http://web.default
```

예상 출력

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
```

```
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

8.2.2.7. OpenShift Cluster Manager를 사용하여 네트워크 정책 생성

클러스터의 네임스페이스에서 허용된 수신 또는 송신 네트워크 트래픽을 설명하는 세분화된 규칙을 정의하기 위해 네트워크 정책을 생성할 수 있습니다.

사전 요구 사항

- OpenShift Cluster Manager 에 로그인했습니다.
- OpenShift Dedicated 클러스터를 생성하셨습니다.
- 클러스터의 ID 공급자를 구성했습니다.
- 구성된 ID 공급자에 사용자 계정을 추가했습니다.
- OpenShift Dedicated 클러스터 내에 프로젝트를 생성하셨습니다.

프로세스

1. OpenShift Cluster Manager 에서 액세스할 클러스터를 클릭합니다.
2. Open console 을 클릭하여 OpenShift 웹 콘솔로 이동합니다.
3. ID 공급자를 클릭하고 클러스터에 로그인할 수 있는 인증 정보를 제공합니다.

4. 관리자 관점에서 **Networking** 에서 **NetworkPolicies** 를 클릭합니다.

5. **NetworkPolicy** 생성을 클릭합니다.

6. 정책 이름 필드에 정책의 이름을 입력합니다.

7. 선택 사항: 이 정책이 하나 이상의 특정 **Pod**에만 적용되는 경우 특정 **Pod**에 대한 라벨 및 선택기를 제공할 수 있습니다. 특정 **Pod**를 선택하지 않으면 이 정책은 클러스터의 모든 **Pod**에 적용됩니다.

8. 선택 사항: 모든 수신 트래픽 거부를 사용하거나 모든 송신 트래픽 확인란을 거부하여 모든 수신 및 송신 트래픽 을 차단할 수 있습니다.

9. 또한 수신 및 송신 규칙의 조합을 추가하여 승인하려는 포트, 네임스페이스 또는 **IP** 블록을 지정할 수 있습니다.

10. 정책에 수신 규칙을 추가합니다.

    a. 수신 규칙 추가 를 선택하여 새 규칙을 구성합니다. 이 작업은 인바운드 트래픽을 제한하려는 방법을 지정할 수 있는 **Add allowed source** 드롭다운 메뉴를 사용하여 새 **Ingress** 규칙 행을 생성합니다. 드롭다운 메뉴에는 인그레스 트래픽을 제한하는 세 가지 옵션이 있습니다.

    - 동일한 네임스페이스의 **Pod**는 동일한 네임스페이스 내의 **Pod**로 트래픽을 제한합니다. 네임스페이스에서 **Pod**를 지정할 수 있지만 이 옵션을 비워 두면 네임스페이스의 **Pod**의 모든 트래픽이 허용됩니다.

    - 클러스터 내부에서 포드가 정책과 동일한 클러스터 내의 **Pod**로 트래픽을 제한할 수 있습니다. 인바운드 트래픽을 허용하려는 네임스페이스 및 **Pod**를 지정할 수 있습니다. 이 옵션을 비워 두면 이 클러스터 내의 모든 네임스페이스 및 **Pod**의 인바운드 트래픽이 허용됩니다.

    - **IP** 블록별 피어 는 지정된 **CIDR(Classless Inter-Domain Routing) IP** 블록의 트래픽을 제한합니다. 예외 옵션으로 특정 **IP**를 차단할 수 있습니다. **CIDR** 필드를 비워 두면 모든 외부 소스의 인바운드 트래픽이 모두 허용됩니다.

b.

모든 인바운드 트래픽을 포트로 제한할 수 있습니다. 포트를 추가하지 않으면 모든 포트에 트래픽에 액세스할 수 있습니다.

11.

네트워크 정책에 송신 규칙을 추가합니다.

a.

송신 규칙 추가 를 선택하여 새 규칙을 구성합니다. 이 작업은 아웃바운드 트래픽을 제한하려는 방법을 지정할 수 있는 **Add allowed destination"*** 드롭다운 메뉴를 사용하여 새 **Egress** 규칙 행을 생성합니다. 드롭다운 메뉴에서는 송신 트래픽을 제한하는 세 가지 옵션을 제공합니다.

- 동일한 네임스페이스의 **Pod**는 동일한 네임스페이스 내의 **pod**로 아웃바운드 트래픽을 제한합니다. 네임스페이스에서 **Pod**를 지정할 수 있지만 이 옵션을 비워 두면 네임스페이스의 **Pod**의 모든 트래픽이 허용됩니다.

- 클러스터 내부에서 포드가 정책과 동일한 클러스터 내의 **Pod**로 트래픽을 제한할 수 있습니다. 아웃바운드 트래픽을 허용하려는 네임스페이스 및 **Pod**를 지정할 수 있습니다. 이 옵션을 비워 두면 이 클러스터 내의 모든 네임스페이스 및 **Pod**의 아웃바운드 트래픽이 허용됩니다.

- **IP** 블록별 피어를 허용 하면 지정된 **CIDR IP** 블록의 트래픽을 제한합니다. 예외 옵션으로 특정 **IP**를 차단할 수 있습니다. **CIDR** 필드를 비워 두면 모든 외부 소스의 아웃바운드 트래픽이 모두 허용됩니다.

b.

모든 아웃바운드 트래픽을 포트로 제한할 수 있습니다. 포트를 추가하지 않으면 모든 포트에 트래픽에 액세스할 수 있습니다.

8.2.3. 네트워크 정책 보기

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

8.2.3.1. NetworkPolicy 오브젝트 예

다음은 예제 **NetworkPolicy** 오브젝트에 대한 주석입니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 1
```

```
spec:
  podSelector: 2
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: 3
        matchLabels:
          app: app
    ports: 4
    - protocol: TCP
      port: 27017
```

1

NetworkPolicy 오브젝트의 이름입니다.

2

정책이 적용되는 Pod를 설명하는 선택기입니다. 정책 오브젝트는 NetworkPolicy 오브젝트를 정의하는 프로젝트에서 Pod만 선택할 수 있습니다.

3

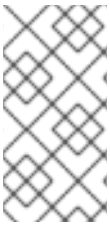
정책 오브젝트가 수신 트래픽을 허용하는 Pod와 일치하는 선택기입니다. 선택기는 NetworkPolicy와 동일한 네임스페이스의 Pod와 일치합니다.

4

트래픽을 수락할 하나 이상의 대상 포트 목록입니다.

8.2.3.2. CLI를 사용하여 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.

절차

- 네임스페이스의 네트워크 정책을 나열합니다.
  - 네임스페이스에 정의된 네트워크 정책 개체를 보려면 다음 명령을 입력합니다.

```
$ oc get networkpolicy
```

  - 선택 사항: 특정 네트워크 정책을 검사하려면 다음 명령을 입력합니다.

```
$ oc describe networkpolicy <policy_name> -n <namespace>
```

    다음과 같습니다.

    <policy_name>

    검사할 네트워크 정책의 이름을 지정합니다.

    <namespace>

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

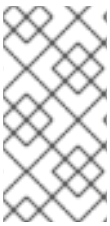
    예를 들면 다음과 같습니다.

```
$ 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
```



참고

cluster-admin 권한을 사용하여 웹 콘솔에 로그인하는 경우 YAML에서 직접 또는 웹 콘솔의 양식에서 클러스터의 모든 네임스페이스에서 네트워크 정책을 볼 수 있습니다.

8.2.3.3. OpenShift Cluster Manager를 사용하여 네트워크 정책 보기

Red Hat OpenShift Cluster Manager에서 네트워크 정책의 구성 세부 정보를 볼 수 있습니다.

사전 요구 사항

- OpenShift Cluster Manager 에 로그인했습니다.
- OpenShift Dedicated 클러스터를 생성하셨습니다.
- 클러스터의 ID 공급자를 구성했습니다.
- 구성된 ID 공급자에 사용자 계정을 추가했습니다.
- 네트워크 정책을 생성했습니다.

프로세스

1.

OpenShift Cluster Manager 웹 콘솔의 Administrator 관점에서 Networking 에서 NetworkPolicies 를 클릭합니다.

2. 확인할 네트워크 정책을 선택합니다.

3. 네트워크 정책 세부 정보 페이지에서 연결된 모든 수신 및 송신 규칙을 볼 수 있습니다.

4. 네트워크 정책 세부 정보에서 YAML 을 선택하여 YAML 형식으로 정책 구성을 확인합니다.

참고

이러한 정책의 세부 사항만 볼 수 있습니다. 이러한 정책을 편집할 수 없습니다.

8.2.4. 네트워크 정책 삭제

admin 역할이 있는 사용자는 네임스페이스에서 네트워크 정책을 삭제할 수 있습니다.

8.2.4.1. CLI를 사용하여 네트워크 정책 삭제

네임스페이스에서 네트워크 정책을 삭제할 수 있습니다.

참고

cluster-admin 역할을 가진 사용자로 로그인하면 클러스터의 모든 네트워크 정책을 삭제할 수 있습니다.

사전 요구 사항

- 클러스터는 mode: NetworkPolicy 로 설정된 OVN-Kubernetes 네트워크 플러그인 또는 OpenShift SDN 네트워크 플러그인과 같은 NetworkPolicy 오브젝트를 지원하는 네트워크 플러그인을 사용합니다. 이 모드는 OpenShift SDN의 기본값입니다.

- OpenShift CLI(oc)를 설치합니다.

- admin 권한이 있는 사용자로 클러스터에 로그인합니다.
- 네트워크 정책이 존재하는 네임스페이스에서 작업하고 있습니다.

절차

- 네트워크 정책 개체를 삭제하려면 다음 명령을 입력합니다.

```
$ oc delete networkpolicy <policy_name> -n <namespace>
```

다음과 같습니다.

<policy_name>

네트워크 정책의 이름을 지정합니다.

<namespace>

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

출력 예

```
networkpolicy.networking.k8s.io/default-deny deleted
```

참고

cluster-admin 권한을 사용하여 웹 콘솔에 로그인하는 경우 작업 메뉴를 통해 클러스터의 네임스페이스에서 직접 또는 웹 콘솔의 정책에서 네트워크 정책을 삭제할 수 있습니다.

8.2.4.2. OpenShift Cluster Manager를 사용하여 네트워크 정책 삭제

네임스페이스에서 네트워크 정책을 삭제할 수 있습니다.

사전 요구 사항

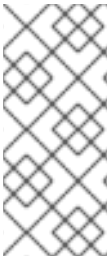
- OpenShift Cluster Manager 에 로그인했습니다.
- OpenShift Dedicated 클러스터를 생성하셨습니다.
- 클러스터의 ID 공급자를 구성했습니다.
- 구성된 ID 공급자에 사용자 계정을 추가했습니다.

절차

1. OpenShift Cluster Manager 웹 콘솔의 Administrator 관점에서 Networking 에서 NetworkPolicies 를 클릭합니다.
2. 다음 방법 중 하나를 사용하여 네트워크 정책을 삭제합니다.
   - 네트워크 정책 표에서 정책을 삭제합니다.
     a. Network Policies (네트워크 정책) 표에서 삭제할 네트워크 정책 행의 스택 메뉴를 선택한 다음 Delete NetworkPolicy 를 클릭합니다.
   - 개별 네트워크 정책 세부 정보에서 작업 드롭다운 메뉴를 사용하여 정책을 삭제합니다.
     a. 네트워크 정책의 작업 드롭다운 메뉴를 클릭합니다.
     b. 메뉴에서 NetworkPolicy 삭제 를 선택합니다.

8.2.5. 네트워크 정책으로 다중 테넌트 격리 구성

클러스터 관리자는 다중 테넌트 네트워크 격리를 제공하도록 네트워크 정책을 구성할 수 있습니다.



참고

OpenShift SDN 네트워크 플러그인을 사용하는 경우 이 섹션에 설명된 대로 네트워크 정책을 구성하는 경우 다중 테넌트 모드와 유사하지만 네트워크 정책 모드가 설정된 네트워크 격리를 제공합니다.

8.2.5.1. 네트워크 정책을 사용하여 다중 테넌트 격리 구성

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

사전 요구 사항

- 클러스터는 **mode: NetworkPolicy** 로 설정된 **OVN-Kubernetes** 네트워크 플러그인 또는 **OpenShift SDN** 네트워크 플러그인과 같은 **NetworkPolicy** 오브젝트를 지원하는 네트워크 플러그인을 사용합니다. 이 모드는 **OpenShift SDN**의 기본값입니다.
- **OpenShift CLI(oc)**를 설치합니다.
- **admin** 권한이 있는 사용자로 클러스터에 로그인합니다.

절차

1. 다음 **NetworkPolicy** 오브젝트를 생성합니다.
   a. 이름이 **allow-from-openshift-ingress**인 정책입니다.

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-ingress
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          policy-group.network.openshift.io/ingress: ""
```

```
  podSelector: {}
  policyTypes:
  - Ingress
EOF
```

참고

policy-group.network.openshift.io/ingress: "" 는 OpenShift SDN에 권장되는 네임스페이스 선택기 레이블입니다. network.openshift.io/policy-group: ingress 네임스페이스 선택기 레이블을 사용할 수 있지만 레거시 레이블입니다.

b.

이름이 allow-from-openshift-monitoring인 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-namespace인 정책:

```
$ cat << EOF| oc create -f -
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-same-namespace
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
EOF
```

d.

이름이 allow-from-kube-apiserver-operator 인 정책:

```
$ cat << EOF| oc create -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-kube-apiserver-operator
      podSelector:
        matchLabels:
          app: kube-apiserver-operator
  policyTypes:
  - Ingress
EOF
```

자세한 내용은 웹 후크 의 상태를 검증하는 **New kube-apiserver-operator webhook 컨트롤러**를 참조하십시오.

2. 선택 사항: 현재 프로젝트에 네트워크 정책이 있는지 확인하려면 다음 명령을 입력합니다.

```
$ oc describe networkpolicy
```

출력 예

```
Name:         allow-from-openshift-ingress
Namespace:    example1
Created on:   2020-06-09 00:28:17 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From:
      NamespaceSelector: network.openshift.io/policy-group: ingress
  Not affecting egress traffic
  Policy Types: Ingress


Name:         allow-from-openshift-monitoring
Namespace:    example1
Created on:   2020-06-09 00:29:57 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
```

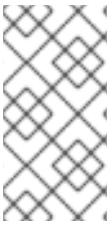
```
PodSelector:     <none> (Allowing the specific traffic to all pods in this namespace)
Allowing ingress traffic:
  To Port: <any> (traffic allowed to all ports)
  From:
    NamespaceSelector: network.openshift.io/policy-group: monitoring
Not affecting egress traffic
Policy Types: Ingress
```

9장. OVN-KUBERNETES 네트워크 플러그인

9.1. OVN-KUBERNETES 네트워크 플러그인 정보

OpenShift Dedicated 클러스터는 Pod 및 서비스 네트워크에 가상화된 네트워크를 사용합니다.

Red Hat OpenShift Networking의 일부인 OVN-Kubernetes 네트워크 플러그인은 OpenShift Dedicated의 기본 네트워크 공급자입니다. OVN-Kubernetes는 OVN(Open Virtual Network)을 기반으로 하며 오버레이 기반 네트워킹 구현을 제공합니다. OVN-Kubernetes 플러그인을 사용하는 클러스터는 각 노드에서 OVS(Open vSwitch)도 실행합니다. OVN은 각 노드에서 선언된 네트워크 구성을 구현하도록 OVS를 구성합니다.



참고

OVN-Kubernetes는 OpenShift Dedicated 및 단일 노드 OpenShift 배포의 기본 네트워킹 솔루션입니다.

OVS 프로젝트에서 발생한 OVN-Kubernetes는 공개 흐름 규칙과 같은 많은 동일한 구성을 사용하여 패킷을 네트워크를 통과하는 방법을 결정합니다. 자세한 내용은 Open Virtual Network 웹 사이트를 참조하십시오.

OVN-Kubernetes는 가상 네트워크 구성을 OpenFlow 규칙으로 변환하는 OVS의 일련의 데몬입니다. OpenFlow 는 네트워크 스위치 및 라우터와 통신하기 위한 프로토콜로, 네트워크 장치에서 네트워크 트래픽 흐름을 원격으로 제어하는 수단을 제공하여 네트워크 트래픽 흐름을 구성, 관리 및 모니터링할 수 있습니다.

OVN-Kubernetes는 OpenFlow 에서 사용할 수 없는 고급 기능을 더 많이 제공합니다. OVN은 분산 가상 라우팅, 분산 논리 스위치, 액세스 제어, DHCP 및 DNS를 지원합니다. OVN은 흐름을 여는 논리 흐름 내에서 분산 가상 라우팅을 구현합니다. 예를 들어 네트워크에서 DHCP 요청을 보내는 Pod가 있는 경우 DHCP 주소를 찾고 있는 브로드캐스트에서 해당 패킷과 일치하는 논리 흐름 규칙이 있으며 게이트웨이, DNS 서버에 IP 주소를 부여합니다.

OVN-Kubernetes는 각 노드에서 데몬을 실행합니다. 데이터베이스와 모든 노드에서 실행되는 OVN 컨트롤러에 대한 데몬 세트가 있습니다. OVN 컨트롤러는 네트워크 공급자 기능을 지원하기 위해 노드에서 Open vSwitch 데몬을 프로그래밍합니다. 송신 IP, 방화벽, 라우터, 하이브리드 네트워킹, IPSEC 암호화, IPv6, 네트워크 정책 로그, 네트워크 정책 로그, 하드웨어 오프로드 및 멀티 캐스트.

9.1.1. OVN-Kubernetes 용도

OVN-Kubernetes 네트워크 플러그인은 **OVN(Open Virtual Network)**을 사용하여 네트워크 트래픽 흐름을 관리하는 오픈 소스 완전한 기능을 갖춘 **Kubernetes CNI** 플러그인입니다. **OVN**은 커뮤니티에서 개발한 벤더와 무관한 네트워크 가상화 솔루션입니다. **OVN-Kubernetes** 네트워크 플러그인

- **OVN(Open Virtual Network)**을 사용하여 네트워크 트래픽 흐름을 관리합니다. **OVN**은 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.

OVN-Kubernetes 네트워크 플러그인은 **OpenShift SDN**에 비해 다음과 같은 이점을 제공합니다.

- 지원되는 플랫폼에서 **IPv6** 단일 스택 및 **IPv4/IPv6** 듀얼 스택 네트워킹에 대한 전체 지원
- **Linux** 및 **Microsoft Windows** 워크로드를 모두 사용하여 하이브리드 클러스터 지원
- 클러스터 내부 통신의 선택적 **IPsec** 암호화
- 호스트 **CPU**에서 호환 가능한 네트워크 카드 및 **DPDK(**데이터 처리 장치**)**로 네트워크 데이터 처리 오프로드

9.1.2. 지원되는 네트워크 플러그인 기능 매트릭스

Red Hat OpenShift Networking은 네트워크 플러그인에 대한 두 가지 옵션인 **OpenShift SDN** 및 **OVN-Kubernetes**를 제공합니다. 다음 표에는 두 네트워크 플러그인 모두에 대한 현재 기능 지원이 요약되어 있습니다.

표 9.1. 기본 CNI 네트워크 플러그인 기능 비교

기능	OVN-Kubernetes	OpenShift SDN
송신 IP	지원됨	지원됨
송신 방화벽 [1]	지원됨	지원됨

기능	OVN-Kubernetes	OpenShift SDN
송신 라우터	지원됨 [2]	지원됨
하이브리드 네트워킹	지원됨	지원되지 않음
클러스터 내부 통신을 위한 IPsec 암호화	지원됨	지원되지 않음
IPv6	지원됨 [3]	지원되지 않음
Kubernetes 네트워크 정책	지원됨	지원됨
Kubernetes 네트워크 정책 로그	지원됨	지원되지 않음
하드웨어 오프로드	지원됨	지원되지 않음
멀티 캐스트	지원됨	지원됨

1. **송신 방화벽은 OpenShift SDN에서 송신 네트워크 정책이라고도 합니다. 이것은 네트워크 정책 송신과 동일하지 않습니다.**

2. **OVN-Kubernetes용 송신 라우터는 리디렉션 모드만 지원합니다.**

3. **IPv6는 베어 메탈, vSphere, IBM Power®, IBM Z® 및 Red Hat OpenStack 클러스터에서만 지원됩니다.**

4. **IBM Power®, IBM Z® 및 Red Hat OpenStack 클러스터에서는 IPv6 단일 스택이 지원되지 않습니다.**

9.1.3. OVN-Kubernetes IPv6 및 듀얼 스택 제한 사항

OVN-Kubernetes 네트워크 플러그인에는 다음과 같은 제한 사항이 있습니다.

- **듀얼 스택 네트워킹을 위해 구성된 클러스터의 경우 IPv4 및 IPv6 트래픽 모두 기본 게이트웨이와 동일한 네트워크 인터페이스를 사용해야 합니다. 이 요구 사항이 충족되지 않으면 ovnkube-node 데몬 세트의 호스트의 Pod가 CrashLoopBackOff 상태가 됩니다. oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml 과 같은 명령으로 Pod를 표시하는**

경우 **status** 필드에 다음 출력에 표시된 대로 기본 게이트웨이에 대한 메시지가 두 개 이상 포함됩니다.

```
I1006 16:09:50.985852   60651 helper_linux.go:73] Found default gateway interface br-ex 192.168.127.1
I1006 16:09:50.985923   60651 helper_linux.go:73] Found default gateway interface ens4 fe80::5054:ff:febe:bcd4
F1006 16:09:50.985939   60651 ovnkube.go:130] multiple gateway interfaces detected: br-ex ens4
```

유일한 해결 방법은 두 IP 제품군이 기본 게이트웨이에 동일한 네트워크 인터페이스를 사용하도록 호스트 네트워킹을 재구성하는 것입니다.

- 듀얼 스택 네트워킹을 위해 구성된 클러스터의 경우 IPv4 및 IPv6 라우팅 테이블 모두 기본 게이트웨이를 포함해야 합니다. 이 요구 사항이 충족되지 않으면 **ovnkube-node** 데몬 세트의 호스트의 Pod가 **CrashLoopBackOff** 상태가 됩니다. **oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml** 과 같은 명령으로 Pod를 표시하는 경우 **status** 필드에 다음 출력에 표시된 대로 기본 게이트웨이에 대한 메시지가 두 개 이상 포함됩니다.

```
I0512 19:07:17.589083  108432 helper_linux.go:74] Found default gateway interface br-ex 192.168.123.1
F0512 19:07:17.589141  108432 ovnkube.go:133] failed to get default gateway interface
```

유일한 해결 방법은 두 IP 제품군에 기본 게이트웨이가 포함되도록 호스트 네트워킹을 재구성하는 것입니다.

9.1.4. 세션 선호도

세션 선호도는 Kubernetes Service 오브젝트에 적용되는 기능입니다. **<service_VIP>:<Port>**에 연결할 때마다 트래픽이 항상 동일한 백엔드에 분산되도록 하려면 *세션 선호도*를 사용할 수 있습니다. 클라이언트의 IP 주소를 기반으로 세션 선호도를 설정하는 방법을 포함한 자세한 내용은 세션 선호도를 참조하십시오.

세션 선호도에 대한 고정 시간 제한

OpenShift Dedicated의 OVN-Kubernetes 네트워크 플러그인은 마지막 패킷을 기반으로 클라이언트에서 세션의 고정 시간 초과를 계산합니다. 예를 들어 **curl** 명령을 10번 실행하면 고정 세션 타이머가 첫 번째 패킷이 아닌 10번째 패킷에서 시작됩니다. 결과적으로 클라이언트가 지속적으로 서비스에 문의하는 경우 세션이 시간 초과되지 않습니다. 서비스가 **timeoutSeconds** 매개변수에 설정된 시간 양에 대한 패킷을 수신하지 않으면 시간 초과가 시작됩니다.

10장. 경로 구성

10.1. 경로 구성

10.1.1. HTTP 기반 경로 생성

경로를 사용하면 공용 URL에서 애플리케이션을 호스팅할 수 있습니다. 애플리케이션의 네트워크 보안 구성에 따라 안전하거나 안전하지 않을 수 있습니다. HTTP 기반 경로는 기본 HTTP 라우팅 프로토콜을 사용하고 보안되지 않은 애플리케이션 포트에서 서비스를 노출하는 비보안 라우팅입니다.

다음 절차에서는 hello-openshift 애플리케이션을 예로 사용하여 웹 애플리케이션에 대한 간단한 HTTP 기반 경로를 생성하는 방법을 설명합니다.

사전 요구 사항

- OpenShift CLI(oc)를 설치합니다.
- 관리자로 로그인되어 있습니다.
- 포트 및 포트의 트래픽을 수신하는 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
```

검증

- 생성한 경로 리소스가 있는지 확인하려면 다음 명령을 실행합니다.

```
$ oc get routes -o yaml <name of resource> 1
```

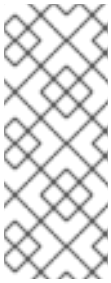
1 이 예제에서 경로 이름은 **hello-openshift** 입니다.

생성된 비보안 경로에 대한 **YAML** 정의 샘플

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
  host: hello-openshift-hello-openshift.<Ingress_Domain> 1
  port:
    targetPort: 8080 2
  to:
    kind: Service
    name: hello-openshift
```

1 **<Ingress_Domain >**은 기본 수신 도메인 이름입니다. **ingresses.config/cluster** 오브젝트는 설치 중에 생성되며 변경할 수 없습니다. 다른 도메인을 지정하려면 **appsDomain** 옵션을 사용하여 대체 클러스터 도메인을 지정할 수 있습니다.

2 **targetPort** 는 이 경로가 가리키는 서비스에서 선택한 **Pod**의 대상 포트입니다.



참고

기본 수신 도메인을 표시하려면 다음 명령을 실행합니다.

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

10.1.2. 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 myroute인 경로에서 2초의 시간 초과를 설정합니다.

```
$ oc annotate route myroute --overwrite haproxy.router.openshift.io/timeout=2s
```

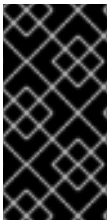
10.1.3. HSTS(HTTP Strict Transport Security)

HSTS(HTTP Strict Transport Security) 정책은 라우트 호스트에서 HTTPS 트래픽만 허용됨을 브라우저 클라이언트에 알리는 보안 강화 정책입니다. 또한 HSTS는 HTTP 리디렉션을 사용하지 않고 HTTPS 전송 신호를 통해 웹 트래픽을 최적화합니다. HSTS는 웹사이트와의 상호 작용을 가속화하는 데 유용합니다.

HSTS 정책이 적용되면 HSTS는 사이트의 HTTP 및 HTTPS 응답에 Strict Transport Security 헤더를 추가합니다. 경로에서 insecureEdgeTerminationPolicy 값을 사용하여 HTTP를 HTTPS로 리디렉션할 수 있습니다. HSTS를 적용하면 클라이언트는 요청을 전송하기 전에 HTTP URL의 모든 요청을 HTTPS로 변경하여 리디렉션이 필요하지 않습니다.

클러스터 관리자는 다음을 수행하도록 HSTS를 구성할 수 있습니다.

- 경로당 HSTS 활성화
- 라우팅당 HSTS 비활성화
- 도메인당 HSTS 시행, 도메인 집합 또는 도메인과 함께 네임스페이스 라벨 사용



중요

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

10.1.3.1. 라우팅당 HSTS(HTTP Strict Transport Security) 활성화

HSTS(HTTP Strict Transport Security)는 HAProxy 템플릿에 구현되고 haproxy.router.openshift.io/hsts_header 주석이 있는 에지 및 재암호화 경로에 적용됩니다.

사전 요구 사항

- 프로젝트에 대한 관리자 권한이 있는 사용자로 클러스터에 로그인합니다.
- OpenShift CLI(oc)를 설치합니다.

절차

- 경로에서 HSTS를 활성화하려면 haproxy.router.openshift.io/hsts_header 값을 에지 종료 또는 재암호화 경로에 추가합니다. oc annotate 툴을 사용하여 다음 명령을 실행하여 이 작업을 수행할 수 있습니다.

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000;\ 1
includeSubDomains;preload"
```

1

이 예에서 최대 나이는 **31536000 ms**로 설정되며 이는 약 **8.5**시간입니다.

참고

이 예제에서는 등호(=)는 따옴표로 묶습니다. 이 작업은 **annotate** 명령을 올바르게 실행하는 데 필요합니다.

주석으로 구성된 경로 예

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-
age=31536000;includeSubDomains;preload 1 2 3
...
spec:
  host: def.abc.com
  tls:
    termination: "reencrypt"
    ...
  wildcardPolicy: "Subdomain"
```

1

필수 항목입니다. **Max-age**는 **HSTS** 정책이 적용되는 시간(초)을 측정합니다. **0**으로 설정하면 정책이 무효화됩니다.

2

선택 사항: 포함되는 경우 **includeSubDomains**는 호스트의 모든 하위 도메인에 호스트와 동일한 **HSTS** 정책이 있어야 함을 알려줍니다.

3

10.1.3.2. 라우팅당 HSTS(HTTP Strict Transport Security) 비활성화

경로당 HSTS(HTTP Strict Transport Security)를 비활성화하려면 경로 주석에서 max-age 값을 0으로 설정할 수 있습니다.

사전 요구 사항

- 프로젝트에 대한 관리자 권한이 있는 사용자로 클러스터에 로그인합니다.
- OpenShift CLI(oc)를 설치합니다.

절차

- HSTS를 비활성화하려면 다음 명령을 입력하여 경로 주석의 max-age 값을 0으로 설정합니다.

```
$ oc annotate route <route_name> -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

작은 정보

다음 YAML을 적용하여 구성 맵을 만들 수 있습니다.

경로당 HSTS 비활성화 예

```
metadata:
  annotations:
    haproxy.router.openshift.io/hsts_header: max-age=0
```

- 네임스페이스의 모든 경로에 대해 HSTS를 비활성화하려면 다음 명령을 입력합니다.

```
$ oc annotate route --all -n <namespace> --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=0"
```

검증

1. 모든 경로에 대한 주석을 쿼리하려면 다음 명령을 입력합니다.

```
$ oc get route  --all-namespaces -o go-template='{{range .items}}{{if
.metadata.annotations}}{{$a := index .metadata.annotations
"haproxy.router.openshift.io/hsts_header"}}{{$n := .metadata.name}}{{with $a}}Name:
{{$n}} HSTS: {{$a}}{{"\n"}}{{else}}{{""}}{{end}}{{end}}{{end}}'
```

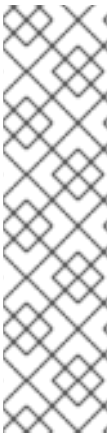
출력 예

```
Name: routename HSTS: max-age=0
```

10.1.4. 쿠키를 사용하여 경로 상태 유지

OpenShift Dedicated는 모든 트래픽이 동일한 끝점에 도달하도록 하여 상태 저장 애플리케이션 트래픽을 활성화하는 고정 세션을 제공합니다. 그러나 재시작, 스케일링 또는 구성 변경 등으로 인해 끝점 **pod**가 종료되면 이러한 상태 저장 특성이 사라질 수 있습니다.

OpenShift Dedicated에서는 쿠키를 사용하여 세션 지속성을 구성할 수 있습니다. 수신 컨트롤러는 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 **pod**로 라우팅되도록 합니다.



참고

HTTP 트래픽을 볼 수 없기 때문에 패스스루 경로에서 쿠키를 설정할 수 없습니다. 대신 백엔드를 결정하는 소스 **IP** 주소를 기반으로 숫자가 계산됩니다.

백엔드가 변경되면 트래픽을 잘못된 서버로 전달하여 덜 고정시킬 수 있습니다. 소스 **IP**를 숨기는 로드 밸런서를 사용하는 경우 모든 연결에 대해 동일한 번호가 설정되고 트래픽이 동일한 **Pod**로 전송됩니다.

10.1.4.1. 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 삭제하면 다음 요청이 끝점을 다시 선택하도록 강제할 수 있습니다. 결과적으로 서버가 과부하된 경우 해당 서버에서 클라이언트의 요청을 제거하고 재배포하려고 합니다.

프로세스

1. 지정된 쿠키 이름으로 경로에 주석을 답니다.

```
$ 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
```

10.1.5. 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.

다음 표에서는 경로 및 액세스 가능성을 보여줍니다.

표 10.1. 경로 가용성

경로	비교 대상	액세스 가능
www.example.com/test	*www.example.com/test*	제공됨
	www.example.com	없음
www.example.com/test 및 *www.example.com*	*www.example.com/test*	제공됨
	www.example.com	제공됨
www.example.com	*www.example.com/text*	예 (경로가 아닌 호스트에 의해 결정됨)
	www.example.com	제공됨

경로가 있는 보안되지 않은 라우터

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-unsecured
spec:
  host: www.example.com
  path: "/test" 1
```

```
to:
  kind: Service
  name: service-name
```

1

경로는 경로 기반 라우터에 대해 추가된 유일한 속성입니다.

참고

라우터가 해당 경우 TLS를 종료하지 않고 요청 콘텐츠를 읽을 수 없기 때문에 패스스루 TLS를 사용할 때 경로 기반 라우팅을 사용할 수 없습니다.

10.1.6. HTTP 헤더 구성

OpenShift Dedicated는 HTTP 헤더로 작업하는 다양한 방법을 제공합니다. 헤더를 설정하거나 삭제할 때 Ingress 컨트롤러의 특정 필드를 사용하거나 개별 경로를 사용하여 요청 및 응답 헤더를 수정할 수 있습니다. 경로 주석을 사용하여 특정 헤더를 설정할 수도 있습니다. 헤더를 구성하는 다양한 방법은 함께 작업할 때 문제가 발생할 수 있습니다.

참고

IngressController 또는 Route CR 내에서 헤더만 설정하거나 삭제할 수 있으므로 추가할 수 없습니다. HTTP 헤더가 값으로 설정된 경우 해당 값은 완료되어야 하며 나중에 추가할 필요가 없습니다. X-Forwarded-For 헤더와 같은 헤더를 추가하는 것이 적합한 경우 spec.httpHeaders.actions 대신 spec.httpHeaders.forwardedHeaderPolicy 필드를 사용합니다.

10.1.6.1. 우선순위 순서

Ingress 컨트롤러와 경로에서 동일한 HTTP 헤더를 수정하는 경우 HAProxy는 요청 또는 응답 헤더인지 여부에 따라 특정 방식으로 작업에 우선순위를 부여합니다.

- HTTP 응답 헤더의 경우 경로에 지정된 작업 후에 Ingress 컨트롤러에 지정된 작업이 실행됩니다. 즉, Ingress 컨트롤러에 지정된 작업이 우선합니다.
- HTTP 요청 헤더의 경우 경로에 지정된 작업은 Ingress 컨트롤러에 지정된 작업 후에 실행됩니다. 즉, 경로에 지정된 작업이 우선합니다.

예를 들어 클러스터 관리자는 다음 구성을 사용하여 **Ingress** 컨트롤러에서 값이 **DENY** 인 **X-Frame-Options** 응답 헤더를 설정합니다.

IngressController 사양 예

```
apiVersion: operator.openshift.io/v1
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: DENY
```

경로 소유자는 클러스터 관리자가 **Ingress** 컨트롤러에 설정한 것과 동일한 응답 헤더를 설정하지만 다음 구성을 사용하여 **SAMEORIGIN** 값이 사용됩니다.

Route 사양의 예

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: SAMEORIGIN
```

IngressController 사양과 Route 사양 모두에서 X-Frame-Options 응답 헤더를 구성하는 경우 특정 경로에서 프레임을 허용하는 경우에도 Ingress 컨트롤러의 글로벌 수준에서 이 헤더에 설정된 값이 우선합니다. 요청 헤더의 경우 Route spec 값은 IngressController 사양 값을 재정의합니다.

이 우선순위는 haproxy.config 파일에서 다음 논리를 사용하므로 Ingress 컨트롤러가 프런트 엔드로 간주되고 개별 경로가 백엔드로 간주되기 때문입니다. 프런트 엔드 구성에 적용된 헤더 값 DENY 는 백엔드에 설정된 SAMEORIGIN 값으로 동일한 헤더를 재정의합니다.

```
frontend public
  http-response set-header X-Frame-Options 'DENY'

frontend fe_sni
  http-response set-header X-Frame-Options 'DENY'

frontend fe_no_sni
  http-response set-header X-Frame-Options 'DENY'

backend be_secure:openshift-monitoring:alertmanager-main
  http-response set-header X-Frame-Options 'SAMEORIGIN'
```

또한 Ingress 컨트롤러 또는 경로 주석을 사용하여 설정된 경로 덮어쓰기 값에 정의된 모든 작업입니다.

10.1.6.2. 특수 케이스 헤더

다음 헤더는 완전히 설정되거나 삭제되지 않거나 특정 상황에서 허용되지 않습니다.

표 10.2. 특수 케이스 헤더 구성 옵션

헤더 이름	IngressController 사양을 사용하여 구성 가능	Route 사양을 사용하여 구성 가능	허용하지 않는 이유	다른 방법을 사용하여 구성 가능

헤더 이름	IngressController 사양을 사용하여 구성 가능	Route 사양을 사용하여 구성 가능	허용하지 않는 이유	다른 방법을 사용하여 구성 가능
proxy	없음	없음	**프록시** HTTP 요청 헤더는 **HTTP_PROXY** 환경 변수에 헤더 값을 삽입하여 취약한 CGI 애플리케이션을 활용하는 데 사용할 수 있습니다. **프록시** HTTP 요청 헤더는 비표준이며 구성 중에 오류가 발생하기 쉽습니다.	없음
host	없음	제공됨	**IngressController** CR을 사용하여 **호스트** HTTP 요청 헤더를 설정하면 올바른 경로를 찾을 때 HAProxy가 실패할 수 있습니다.	없음
strict-transport-security	없음	없음	**strict-transport-security** HTTP 응답 헤더는 경로 주석을 사용하여 이미 처리되었으며 별도의 구현이 필요하지 않습니다.	제공됨: **haproxy.router.openshift.io/hsts_header** 경로 주석
쿠키 및 **설정 쿠키**	없음	없음	HAProxy가 클라이언트 연결을 특정 백엔드 서버에 매핑하는 세션 추적에 사용되는 쿠키입니다. 이러한 헤더를 설정하도록 허용하면 HAProxy의 세션 선호도를 방해하고 HAProxy의 쿠키 소유권을 제한할 수 있습니다.	예: • **haproxy.router.openshift.io/disable_cookie** 경로 주석 • **haproxy.router.openshift.io/cookie_name** 경로 주석

10.1.7. 경로에서 HTTP 요청 및 응답 헤더 설정 또는 삭제

규정 준수 목적 또는 기타 이유로 특정 HTTP 요청 및 응답 헤더를 설정하거나 삭제할 수 있습니다. Ingress 컨트롤러에서 제공하는 모든 경로 또는 특정 경로에 대해 이러한 헤더를 설정하거나 삭제할 수 있습니다.

예를 들어 경로를 제공하는 Ingress 컨트롤러에서 지정하는 기본 글로벌 위치가 있더라도 해당 콘텐츠가 여러 언어로 작성된 경우 웹 애플리케이션에서 특정 경로에 대한 콘텐츠를 제공할 수 있도록 할 수 있습니다.

다음 절차에서는 애플리케이션 https://app.example.com 과 연결된 URL이 위치 https://app.example.com/lang/en-us 로 전달되도록 Content-Location HTTP 요청 헤더를 설정하는 경로를 생성합니다. 애플리케이션 트래픽을 이 위치로 전달한다는 것은 특정 경로를 사용하는 사람이 미국 영어로 작성된 웹 콘텐츠에 액세스하는 것을 의미합니다.

사전 요구 사항

- OpenShift CLI(oc)가 설치되어 있습니다.
- OpenShift Dedicated 클러스터에 프로젝트 관리자로 로그인되어 있습니다.
- 포트에서 트래픽을 수신하는 포트와 HTTP 또는 TLS 끝점을 노출하는 웹 애플리케이션이 있습니다.

프로세스

1. 경로 정의를 생성하고 app-example-route.yaml 이라는 파일에 저장합니다.

HTTP 헤더 지시문을 사용하여 생성된 경로의 YAML 정의

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  host: app.example.com
  tls:
    termination: edge
  to:
    kind: Service
```

```
  name: app-example
httpHeaders:
  actions: 1
    response: 2
    - name: Content-Location 3
      action:
        type: Set 4
        set:
          value: /lang/en-us 5
```

1

2

변경할 헤더 유형입니다. 이 경우 응답 헤더입니다.

3

변경할 헤더의 이름입니다. 설정하거나 삭제할 수 있는 사용 가능한 헤더 목록은 *HTTP 헤더 구성* 을 참조하십시오.

4

헤더에서 수행되는 작업 유형입니다. 이 필드에는 **Set** 또는 **Delete** 값이 있을 수 있습니다.

5

HTTP 헤더를 설정할 때 값을 제공해야 합니다. 값은 해당 헤더에 사용 가능한 지시문 목록(예: **DENY**)의 문자열이거나 **HAProxy**의 동적 값 구문을 사용하여 해석되는 동적 값이 될 수 있습니다. 이 경우 값은 콘텐츠의 상대 위치로 설정됩니다.

2. 새로 생성된 경로 정의를 사용하여 기존 웹 애플리케이션에 대한 경로를 생성합니다.

```
$ oc -n app-example create -f app-example-route.yaml
```

HTTP 요청 헤더의 경우 경로 정의에 지정된 작업이 **Ingress** 컨트롤러의 **HTTP** 요청 헤더에 실행된 후 실행됩니다. 즉, 경로의 해당 요청 헤더에 설정된 모든 값이 **Ingress** 컨트롤러에 설정된 값보다 우선합니다. **HTTP** 헤더 처리 순서에 대한 자세한 내용은 *HTTP 헤더 구성* 을 참조하십시오.

10.1.8. 경로별 주석

Ingress 컨트롤러는 노출하는 모든 경로에 기본 옵션을 설정할 수 있습니다. 개별 경로는 주석에 특정 구성을 제공하는 방식으로 이러한 기본값 중 일부를 덮어쓸 수 있습니다. Red 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 10.3. 경로 주석

변수	설명	기본값으로 사용되는 환경 변수
haproxy.router.openshift.io/balance	로드 밸런싱 알고리즘을 설정합니다. 사용 가능한 옵션은 **random**, **source**, **roundrobin**, **leastconn**입니다. 기본값은 TLS 패스스루 경로의 **source** 입니다. 다른 모든 경로의 경우 기본값은 **random** 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 수신되는 각 HTTP 요청에 대해 어떤 백엔드 연결을 제공하는지 밸런싱 알고리즘이 사용됩니다.	
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 개인 경우 각각 이 수만큼의 연결이 있을 수 있습니다. 라우터가 여러 개 있고 조정이 이루어지지 않는 경우에는 각각 이 횟수만큼 연결할 수 있습니다. 설정하지 않거나 0으로 설정하면 제한이 없습니다.	

변수	설명	기본값으로 사용되는 환경 변수
haproxy.router.openshift.io/rate-limit-connections	**'true'** 또는 **'TRUE'** 를 설정하면 경로당 특정 백엔드에서 고정 테이블을 통해 구현되는 속도 제한 기능을 사용할 수 있습니다. 참고: 이 주석을 사용하면 서비스 거부 공격에 대한 기본 보호 기능이 제공됩니다.	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	동일한 소스 IP 주소를 통해 만든 동시 TCP 연결 수를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 서비스 거부 공격에 대한 기본 보호 기능이 제공됩니다.	
haproxy.router.openshift.io/rate-limit-connections.rate-http	동일한 소스 IP 주소가 있는 클라이언트에서 HTTP 요청을 수행할 수 있는 속도를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 서비스 거부 공격에 대한 기본 보호 기능이 제공됩니다.	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	동일한 소스 IP 주소가 있는 클라이언트에서 TCP 연결을 수행할 수 있는 속도를 제한합니다. 숫자 값을 허용합니다. 참고: 이 주석을 사용하면 서비스 거부 공격에 대한 기본 보호 기능이 제공됩니다.	
haproxy.router.openshift.io/timeout	경로에 대한 서버 쪽 타임아웃을 설정합니다. (TimeUnits)	**ROUTER_DEFAULT_SERVER_TIMEOUT**
haproxy.router.openshift.io/timeout-tunnel	이 시간 초과는 일반 텍스트, 에지, 재암호화 또는 패스스루 경로와 같은 터널 연결에 적용됩니다. 일반 텍스트, 에지 또는 재암호화 경로 유형을 사용하면 이 주석이 기존 시간 초과 값을 사용하여 시간 제한 터널로 적용됩니다. passthrough 경로 유형의 경우 주석은 기존의 시간 초과 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 전체 소프트 중지를 수행하는 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 허용 목록을 설정합니다. 허용 목록은 승인된 소스 주소에 대한 IP 주소 및 CIDR 범위로 이루어진 공백으로 구분된 목록입니다. 허용 목록에 없는 IP 주소의 요청은 삭제됩니다. **haproxy.config** 파일에 직접 표시되는 최대 IP 주소 및 CIDR 범위는 61입니다. [1]	
haproxy.router.openshift.io/hsts_header	엣지 종단 경로 또는 재암호화 경로에 Strict-Transport-Security 헤더를 설정합니다.	
haproxy.router.openshift.io/rewrite-target	백엔드의 요청 재작성 경로를 설정합니다.	
router.openshift.io/cookie-same-site	쿠키를 제한하는 값을 설정합니다. 값은 다음과 같습니다. **LAX**: 브라우저는 사이트 간 요청에 쿠키를 보내지 않지만 사용자가 외부 사이트에서 원본 사이트로 이동할 때 쿠키를 보냅니다. 이는 **SameSite** 값이 지정되지 않은 경우 기본 브라우저 동작입니다. **Strict**: 브라우저가 동일한 사이트 요청에 대해서만 쿠키를 보냅니다. **none**: 브라우저가 교차 사이트 요청과 동일한 사이트 요청에 대해 쿠키를 보냅니다. 이 값은 재암호화 및 엣지 경로에만 적용됩니다. 자세한 내용은 SameSite 쿠키 설명서를 참조하십시오.	

변수	설명	기본값으로 사용되는 환경 변수
haproxy.router.openshift.io/set-forwarded-headers	라우터당 **Forwarded** 및 **X-Forwarded-For** HTTP 헤더를 처리하기 위한 정책을 설정합니다. 값은 다음과 같습니다. **append**: 기존 헤더를 유지하면서 헤더를 추가합니다. 이는 기본값입니다. **replace**: 헤더를 설정하고 기존 헤더를 제거합니다. **never**: 헤더를 설정하지 않고 기존 헤더를 유지합니다. **if-none**: 아직 설정되지 않은 경우 헤더를 설정합니다.	**ROUTER_SET_FORWARDED_HEADERS**

1.

허용 목록의 IP 주소 및 CIDR 범위가 61를 초과하면 haproxy.config 에서 참조되는 별도의 파일에 작성됩니다. 이 파일은 var/lib/haproxy/router/whitelists 폴더에 저장됩니다.

참고

주소가 허용 목록에 작성되도록 하려면 전체 CIDR 범위가 Ingress 컨트롤러 구성 파일에 나열되어 있는지 확인합니다. etcd 오브젝트 크기 제한은 경로 주석의 크기를 제한합니다. 이로 인해 허용 목록에 포함할 수 있는 최대 IP 주소 및 CIDR 범위에 대한 임계값이 생성됩니다.

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**	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. 일부 유효한 시간 제한 값은 예상되는 특정 시간 초과가 아니라 특정 변수의 합계일 수 있습니다. 예를 들어 **ROUTER_SLOWLORIS_HTTP_KEEPALIVE**는 **timeout http-keep-alive**를 조정합니다. 기본적으로 **300s**로 설정되지만 HAProxy는 **5s**로 설정된 **tcp-request inspect-delay**도 대기합니다. 이 경우 전체 시간 초과는 **300s** + **5s**입니다.
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: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/timeout: 5500ms (1)
...
```

(1) HAProxy 지원 단위(us, ms, s, m, h, d)를 사용하여 새 타임아웃을 지정합니다. 단위가 제공되지 않는 경우 ms가 기본값입니다.

참고

패스스루(passthrough) 경로에 대한 서버 쪽 타임아웃 값을 너무 낮게 설정하면 해당 경로에서 WebSocket 연결이 자주 시간 초과될 수 있습니다.

하나의 특정 IP 주소만 허용하는 경로

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

여러 IP 주소를 허용하는 경로

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

IP 주소 CIDR 네트워크를 허용하는 경로

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

IP 주소 및 IP 주소 CIDR 네트워크를 둘 다 허용하는 경로

```
metadata:
  annotations:
    haproxy.router.openshift.io/ip_whitelist: 180.5.61.153 192.168.1.0/24 10.0.0.0/8
```

재작성 대상을 지정하는 경로

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / (1)
...
```

(1) /를 백엔드의 요청 재작성 경로로 설정합니다.

경로에 **haproxy.router.openshift.io/rewrite-target** 주석을 설정하면 **Ingress Controller**에서 요청을 백엔드 애플리케이션으로 전달하기 전에 이 경로를 사용하여 **HTTP** 요청의 경로를 재작성해야 합니다. **spec.path**에 지정된 경로와 일치하는 요청 경로 부분은 주석에 지정된 재작성 대상으로 교체됩니다.

다음 표에 **spec.path**, 요청 경로, 재작성 대상의 다양한 조합에 따른 경로 재작성 동작의 예가 있습니다.

표 10.4. 재작성 대상 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

haproxy.router.openshift.io/rewrite-target 의 특정 특수 문자는 올바르게 이스케이프해야 하므로 특수 처리가 필요합니다. 이러한 문자를 처리하는 방법을 알아보려면 다음 표를 참조하십시오.

표 10.5. 특수 문자 처리

문자의 경우	문자 사용	참고
#	\#	#을 사용하지 마십시오. 재작성 표현식이 종료되기 때문입니다.

문자의 경우	문자 사용	참고
%	% 또는 %%	%%와 같은 홀수 시퀀스를 사용하지 마십시오.
‘	\’	무시될 수 있기 때문에 방지

다른 모든 유효한 URL 문자는 이스케이프 없이 사용할 수 있습니다.

10.1.9. Ingress 오브젝트를 통해 기본 인증서를 사용하여 경로 생성

TLS 구성을 지정하지 않고 Ingress 오브젝트를 생성하면 OpenShift Dedicated에서 비보안 경로를 생성합니다. 기본 수신 인증서를 사용하여 에지 종료 보안 경로를 생성하는 Ingress 오브젝트를 생성하려면 다음과 같이 빈 TLS 구성을 지정할 수 있습니다.

사전 요구 사항

- 노출하려는 서비스가 있습니다.
- OpenShift CLI(oc)에 액세스할 수 있습니다.

프로세스

1. Ingress 오브젝트에 대한 YAML 파일을 생성합니다. 이 예제에서는 파일을 example-ingress.yaml 이라고 합니다.

   Ingress 오브젝트의 YAML 정의

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  ...
spec:
  rules:
    ...
  tls:
  - {} 1
```

1

이 정확한 구문을 사용하여 사용자 정의 인증서를 지정하지 않고 **TLS**를 지정합니다.

2. 다음 명령을 실행하여 **Ingress** 오브젝트를 생성합니다.

```
$ oc create -f example-ingress.yaml
```

검증

- 다음 명령을 실행하여 **OpenShift Dedicated**에서 **Ingress** 오브젝트에 대한 예상 경로를 생성했는지 확인합니다.

```
$ oc get routes -o yaml
```

출력 예

```
apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend-j9sdd 1
    ...
  spec:
  ...
    tls: 2
      insecureEdgeTerminationPolicy: Redirect
      termination: edge 3
  ...
```

1

경로 이름에는 **Ingress** 오브젝트의 이름과 임의의 접미사가 포함됩니다.

2

기본 인증서를 사용하려면 경로에서 **spec.certificate** 를 지정하지 않아야 합니다.

3

경로는 엣지 종료 정책을 지정해야 합니다.

10.1.10. Ingress 주석의 대상 CA 인증서를 사용하여 경로 생성

route.openshift.io/destination-ca-certificate-secret 주석을 Ingress 오브젝트에서 사용하여 사용자 정의 대상 CA 인증서로 경로를 정의할 수 있습니다.

사전 요구 사항

- PEM 인코딩 파일에 인증서/키 쌍이 있고, 인증서가 경로 호스트에 유효할 수 있습니다.
- 인증서 체인을 완성하는 PEM 인코딩 파일에 별도의 CA 인증서가 있을 수 있습니다.
- PEM 인코딩 파일에 별도의 대상 CA 인증서가 있어야 합니다.
- 노출하려는 서비스가 있어야 합니다.

프로세스

1. Ingress 주석에 **route.openshift.io/destination-ca-certificate-secret** 을 추가합니다.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt"
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert 1
...
```

1

주석은 kubernetes 보안을 참조합니다.

2. 이 주석에서 참조하는 보안은 생성된 경로에 삽입됩니다.

출력 예

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: reencrypt
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert
spec:
...
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: reencrypt
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
...
```

추가 리소스

- appsDomain 옵션을 사용하여 대체 클러스터 도메인 지정

10.2. 보안 경로

보안 경로는 여러 유형의 TLS 종료를 사용하여 클라이언트에 인증서를 제공하는 기능을 제공합니다. 다음 섹션에서는 사용자 정의 인증서를 사용하여 재암호화 에지 및 패스스루 경로를 생성하는 방법을 설명합니다.

중요

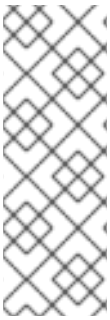
공용 끝점을 통해 Microsoft Azure에서 경로를 생성하는 경우 리소스 이름에 제한이 적용됩니다. 특정 용어를 사용하는 리소스를 생성할 수 없습니다. Azure에서 제한하는 용어 목록은 Azure 설명서의 예약된 리소스 이름 오류 해결을 참조하십시오.

10.2.1. 사용자 정의 인증서를 사용하여 재암호화 경로 생성

oc create route 명령을 사용하면 재암호화 **TLS** 종료와 사용자 정의 인증서로 보안 경로를 구성할 수 있습니다.

사전 요구 사항

- **PEM** 인코딩 파일에 인증서/키 쌍이 있고 해당 인증서가 경로 호스트에 유효해야 합니다.
- 인증서 체인을 완성하는 **PEM** 인코딩 파일에 별도의 **CA** 인증서가 있을 수 있습니다.
- **PEM** 인코딩 파일에 별도의 대상 **CA** 인증서가 있어야 합니다.
- 노출하려는 서비스가 있어야 합니다.



참고

암호로 보호되는 키 파일은 지원되지 않습니다. 키 파일에서 암호를 제거하려면 다음 명령을 사용하십시오.

```
$ openssl rsa -in password_protected_tls.key -out tls.key
```

프로세스

이 절차에서는 사용자 정의 인증서를 사용하여 **Route** 리소스를 생성하고 **TLS** 종료를 재암호화합니다. 다음 예에서는 인증서/키 쌍이 현재 작업 디렉터리의 **tls.crt** 및 **tls.key** 파일에 있다고 가정합니다. **Ingress** 컨트롤러에서 서비스의 인증서를 신뢰하도록 하려면 대상 **CA** 인증서도 지정해야 합니다. 인증서 체인을 완료하는 데 필요한 경우 **CA** 인증서를 지정할 수도 있습니다. **tls.crt, tls.key, cacert.crt, ca.crt(옵션)**에 실제 경로 이름을 사용하십시오. **frontend**에는 노출하려는 서비스 리소스 이름을 사용합니다. **www.example.com**을 적절한 호스트 이름으로 바꿉니다.

- 재암호화 **TLS** 종료 및 사용자 정의 인증서를 사용하여 보안 **Route** 리소스를 생성합니다.

```
$ oc create route reencrypt --service=frontend --cert=tls.crt --key=tls.key --dest-ca-cert=destca.crt --ca-cert=ca.crt --hostname=www.example.com
```

생성된 **Route** 리소스는 다음과 유사합니다.

보안 경로의 **YAML** 정의

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: reencrypt
    key: |-
      -----BEGIN PRIVATE KEY-----
      [...]
      -----END PRIVATE KEY-----
    certificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    caCertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    destinationCACertificate: |-
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

자세한 옵션은 **oc create route reencrypt --help**를 참조하십시오.

10.2.2. 사용자 정의 인증서를 사용하여 엣지 경로 생성

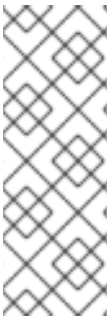
oc create route 명령을 사용하면 엣지 **TLS** 종료와 사용자 정의 인증서로 보안 경로를 구성할 수 있습니다. 엣지 경로를 사용하면 **Ingress** 컨트롤러에서 트래픽을 대상 **Pod**로 전달하기 전에 **TLS** 암호화를 종료합니다. 이 경로는 **Ingress** 컨트롤러가 경로에 사용하는 **TLS** 인증서 및 키를 지정합니다.

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. frontend에는 노출하려는 서비스 이름을 사용합니다. www.example.com을 적절한 호스트 이름으로 바꿉니다.

- 엣지 TLS 종료 및 사용자 정의 인증서를 사용하여 보안 Route 리소스를 생성합니다.

```
$ oc create route edge --service=frontend --cert=tls.crt --key=tls.key --ca-cert=ca.crt --
hostname=www.example.com
```

생성된 Route 리소스는 다음과 유사합니다.

보안 경로의 YAML 정의

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend
spec:
  host: www.example.com
  to:
    kind: Service
    name: frontend
  tls:
    termination: edge
    key: |-
```

```
    -----BEGIN PRIVATE KEY-----
    [...]
    -----END PRIVATE KEY-----
  certificate: |-
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
  caCertificate: |-
    -----BEGIN CERTIFICATE-----
    [...]
    -----END CERTIFICATE-----
```

추가 옵션은 **oc create route edge --help**를 참조하십시오.

10.2.3. 패스스루 라우팅 생성

oc create route 명령을 사용하면 패스스루 종료와 사용자 정의 인증서로 보안 경로를 구성할 수 있습니다. 패스스루 종료를 사용하면 암호화된 트래픽이 라우터에서 **TLS** 종료를 제공하지 않고 바로 대상으로 전송됩니다. 따라서 라우터에 키 또는 인증서가 필요하지 않습니다.

사전 요구 사항

- 노출하려는 서비스가 있어야 합니다.

프로세스

- **Route** 리소스를 생성합니다.

```
$ oc create route passthrough route-passthrough-secured --service=frontend --
port=8080
```

생성된 **Route** 리소스는 다음과 유사합니다.

패스스루 종료를 사용하는 보안 경로

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
```

```
  name: route-passthrough-secured 1
spec:
  host: www.example.com
  port:
    targetPort: 8080
  tls:
    termination: passthrough 2
    insecureEdgeTerminationPolicy: None 3
  to:
    kind: Service
    name: frontend
```

1

63자로 제한되는 개체의 이름입니다.

2

termination 필드는 **passthrough**로 설정됩니다. 이 필드는 유일한 필수 **tls** 필드입니다.

3

insecureEdgeTerminationPolicy는 선택 사항입니다. 비활성화경우 유효한 값은 **None, Redirect** 또는 빈 값입니다.

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

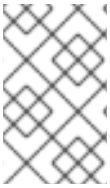
10.2.4. 외부 관리 인증서를 사용하여 경로 생성

중요

TLS 시크릿에서 외부 인증서로 경로를 보호하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

경로 API의 .spec.tls.externalCertificate 필드를 사용하여 타사 인증서 관리 솔루션으로 OpenShift Dedicated 경로를 구성할 수 있습니다. 시크릿을 통해 외부 관리형 TLS 인증서를 참조할 수 있으므로 수동 인증서 관리가 필요하지 않습니다. 외부 관리형 인증서를 사용하면 인증서 업데이트를 원활하게 롤아웃하는 오류를 줄일 수 있으므로 OpenShift 라우터에서 업데이트된 인증서를 즉시 제공할 수 있습니다.



참고

이 기능은 에지 경로와 재암호화 경로에 모두 적용됩니다.

사전 요구 사항

- RouteExternalCertificate 기능 게이트를 활성화해야 합니다.
- routes/custom-host 에 대한 생성 및 업데이트 권한이 있어야 합니다.
- tls.key 및 tls.crt 키가 모두 포함된 kubernetes.io/tls 유형의 PEM 인코딩 형식의 유효한 인증서/키 쌍이 포함된 시크릿이 있어야 합니다.
- 참조된 보안을 보안하려는 경로와 동일한 네임스페이스에 배치해야 합니다.

프로세스

1. 다음 명령을 실행하여 라우터 서비스 계정을 읽을 수 있도록 시크릿과 동일한 네임스페이스에 역할을 생성합니다.

```
$ oc create role secret-reader --verb=get,list,watch --resource=secrets --resource-
name=<secret-name> \ 1
--namespace=<current-namespace> 2
```

1 시크릿의 실제 이름을 지정합니다.

2 시크릿과 경로가 모두 있는 네임스페이스를 지정합니다.

2.

보안과 동일한 네임스페이스에 **rolebinding** 을 생성하고 다음 명령을 실행하여 라우터 서비스 계정을 새로 생성된 역할에 바인딩합니다.

```
$ oc create rolebinding secret-reader-binding --role=secret-reader --
serviceaccount=openshift-ingress:router --namespace=<current-namespace> 1
```

1

시크릿과 경로가 모두 있는 네임스페이스를 지정합니다.

3.

경로를 정의하는 **YAML** 파일을 생성하고 다음 예제를 사용하여 인증서가 포함된 보안을 지정합니다.

보안 경로에 대한 **YAML** 정의

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: myedge
  namespace: test
spec:
  host: myedge-test.apps.example.com
  tls:
    externalCertificate:
      name: <secret-name> 1
    termination: edge
    [...]
[...]
```

1

시크릿의 실제 이름을 지정합니다.

4.

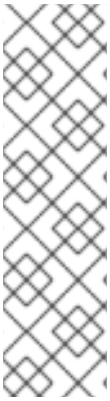
다음 명령을 실행하여 경로 리소스를 생성합니다.

```
$ oc apply -f <route.yaml> 1
```

1

생성된 YAML 파일 이름을 지정합니다.

보안이 존재하고 인증서/키 쌍이 있는 경우 라우터는 모든 사전 요구 사항을 충족하는 경우 생성된 인증서를 제공합니다.



참고

.spec.tls.externalCertificate 를 제공하지 않으면 라우터에서 기본 생성된 인증서를 사용합니다.

.spec.tls.externalCertificate 필드를 사용하는 경우 .spec.tls.certificate 필드 또는 .spec.tls.key 필드를 지정할 수 없습니다.

추가 리소스

- 외부 관리형 인증서가 있는 경로 문제 해결은 OpenShift Dedicated 라우터 Pod 로그에 오류가 있는지 확인하십시오. Pod 문제 수집을 참조하십시오.