



OpenShift Container Platform 4.9

ネットワーク

クラスターネットワークの設定および管理

法律上の通知

Copyright © 2023 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

この文書では、DNS、ingress および Pod ネットワークを含む、OpenShift Container Platform のクラスターネットワークを設定し、管理する方法を説明します。

目次

第1章 ネットワークについて	6
1.1. OPENSIFT CONTAINER PLATFORM DNS	6
1.2. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	6
1.3. OPENSIFT CONTAINER PLATFORM ネットワーキングの一般用語集	7
第2章 ホストへのアクセス	10
2.1. インストーラーでプロビジョニングされるインフラストラクチャクラスターでの AMAZON WEB SERVICES のホストへのアクセス	10
第3章 ネットワーキング OPERATOR の概要	11
3.1. CLUSTER NETWORK OPERATOR	11
3.2. DNS OPERATOR	11
3.3. INGRESS OPERATOR	11
第4章 OPENSIFT CONTAINER PLATFORM における CLUSTER NETWORK OPERATOR	12
4.1. CLUSTER NETWORK OPERATOR	12
4.2. クラスターネットワーク設定の表示	12
4.3. CLUSTER NETWORK OPERATOR のステータス表示	13
4.4. CLUSTER NETWORK OPERATOR ログの表示	13
4.5. CLUSTER NETWORK OPERATOR (CNO) の設定	14
4.6. 関連情報	19
第5章 OPENSIFT CONTAINER PLATFORM の DNS OPERATOR	20
5.1. DNS OPERATOR	20
5.2. DNS OPERATOR MANAGEMENTSTATE の変更	20
5.3. DNS POD 配置の制御	21
5.4. デフォルト DNS の表示	22
5.5. DNS 転送の使用	23
5.6. DNS OPERATOR のステータス	24
5.7. DNS OPERATOR ログ	25
第6章 OPENSIFT CONTAINER PLATFORM の INGRESS OPERATOR	26
6.1. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR	26
6.2. INGRESS 設定アセット	26
6.3. INGRESS コントローラー設定パラメーター	26
6.4. デフォルト INGRESS コントローラーの表示	41
6.5. INGRESS OPERATOR ステータスの表示	41
6.6. INGRESS コントローラーログの表示	42
6.7. INGRESS コントローラーステータスの表示	42
6.8. INGRESS コントローラーの設定	42
6.9. 関連情報	65
第7章 エンドポイントへの接続の確認	66
7.1. 実行する接続ヘルスチェック	66
7.2. 接続ヘルスチェックの実装	66
7.3. PODNETWORKCONNECTIVITYCHECK オブジェクトフィールド	66
7.4. エンドポイントのネットワーク接続の確認	69
第8章 ノードポートサービス範囲の設定	74
8.1. 前提条件	74
8.2. ノードのポート範囲の拡張	74
8.3. 関連情報	75
第9章 IP フェイルオーバーの設定	76

9.1. IP フェイルオーバーの環境変数	77
9.2. IP フェイルオーバーの設定	78
9.3. 仮想 IP アドレスについて	81
9.4. CHECK スクリプトおよび NOTIFY スクリプトの設定	82
9.5. VRRP プリエンプションの設定	84
9.6. VRRP ID オフセットについて	85
9.7. 254 を超えるアドレスについての IP フェイルオーバーの設定	85
9.8. INGRESSIP の高可用性	86
9.9. IP フェイルオーバーの削除	87
第10章 ベアメタルクラスターでの SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の使用	89
10.1. OPENSIFT CONTAINER PLATFORM での SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) のサポート	89
10.2. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の有効化	90
10.3. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) が有効になっていることの確認	91
第11章 PTP ハードウェアの使用	94
11.1. PTP ハードウェアについて	94
11.2. PTP について	94
11.3. CLI を使用した PTP OPERATOR のインストール	95
11.4. WEB コンソールを使用した PTP OPERATOR のインストール	97
11.5. PTP ネットワークデバイスの自動検出	97
11.6. LINUXPTP サービスを通常のクロックとして設定	98
11.7. LINUXPTP サービスを境界クロックとして設定	101
11.8. PTP ハードウェアの FIFO 優先スケジューリングの設定	105
11.9. 一般的な PTP OPERATOR の問題のトラブルシューティング	106
11.10. PTP ハードウェアの高速イベント通知フレームワーク	109
第12章 ネットワークポリシー	118
12.1. ネットワークポリシーについて	118
12.2. ネットワークポリシーイベントのロギング	121
12.3. ネットワークポリシーの作成	129
12.4. ネットワークポリシーの表示	131
12.5. ネットワークポリシーの編集	133
12.6. ネットワークポリシーの削除	135
12.7. プロジェクトのデフォルトネットワークポリシーの定義	136
12.8. ネットワークポリシーを使用したマルチテナント分離の設定	139
第13章 複数ネットワーク	142
13.1. 複数ネットワークについて	142
13.2. 追加のネットワークの設定	143
13.3. 仮想ルーティングおよび転送について	155
13.4. マルチネットワークポリシーの設定	156
13.5. POD の追加のネットワークへの割り当て	162
13.6. 追加ネットワークからの POD の削除	167
13.7. 追加ネットワークの編集	168
13.8. 追加ネットワークの削除	169
13.9. VRF へのセカンダリーネットワークの割り当て	169
第14章 ハードウェアネットワーク	173
14.1. SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) ハードウェアネットワークについて	173
14.2. SR-IOV NETWORK OPERATOR のインストール	179
14.3. SR-IOV NETWORK OPERATOR の設定	182
14.4. SR-IOV ネットワークデバイスの設定	187

14.5. SR-IOV イーサネットネットワーク割り当ての設定	196
14.6. SR-IOV INFINIBAND ネットワーク割り当ての設定	202
14.7. POD の SR-IOV の追加ネットワークへの追加	208
14.8. 高パフォーマンスのマルチキャストの使用	214
14.9. DPDK および RDMA の使用	216
14.10. SR-IOV NETWORK OPERATOR のインストール	225
第15章 OPENSIFT SDN デフォルト CNI ネットワークプロバイダー	227
15.1. OPENSIFT SDN デフォルト CNI ネットワークプロバイダーについて	227
15.2. プロジェクトの EGRESS IP の設定	228
15.3. プロジェクトの EGRESS ファイアウォールの設定	233
15.4. プロジェクトの EGRESS ファイアウォールの編集	238
15.5. プロジェクトの EGRESS ファイアウォールの編集	239
15.6. プロジェクトからの EGRESS ファイアウォールの削除	239
15.7. EGRESS ルーター POD の使用についての考慮事項	240
15.8. リダイレクトモードでの EGRESS ルーター POD のデプロイ	243
15.9. HTTP プロキシモードでの EGRESS ルーター POD のデプロイ	245
15.10. DNS プロキシモードでの EGRESS ルーター POD のデプロイ	248
15.11. CONFIGMAP からの EGRESS ルーター POD 宛先一覧の設定	251
15.12. プロジェクトのマルチキャストの有効化	253
15.13. プロジェクトのマルチキャストの無効化	256
15.14. OPENSIFT SDN を使用したネットワーク分離の設定	256
15.15. KUBE-PROXY の設定	258
第16章 OVN-KUBERNETES デフォルト CNI ネットワークプロバイダー	261
16.1. OVN-KUBERNETES デフォルト CONTAINER NETWORK INTERFACE (CNI) ネットワークプロバイダーについて	261
16.2. OPENSIFT SDN クラスタネットワークプロバイダーからの移行	263
16.3. OPENSIFT SDN ネットワークプロバイダーへのロールバック	273
16.4. IPV4/IPV6 デュアルスタックネットワークへの変換	277
16.5. IPSEC 暗号化の設定	279
16.6. プロジェクトの EGRESS ファイアウォールの設定	281
16.7. プロジェクトの EGRESS ファイアウォールの表示	286
16.8. プロジェクトの EGRESS ファイアウォールの編集	287
16.9. プロジェクトからの EGRESS ファイアウォールの削除	288
16.10. EGRESS IP アドレスの設定	288
16.11. EGRESS IP アドレスの割り当て	294
16.12. EGRESS ルーター POD の使用についての考慮事項	295
16.13. リダイレクトモードでの EGRESS ルーター POD のデプロイ	298
16.14. プロジェクトのマルチキャストの有効化	303
16.15. プロジェクトのマルチキャストの無効化	305
16.16. ネットワークフローの追跡	305
16.17. ハイブリッドネットワークの設定	309
第17章 ルートの作成	312
17.1. ルート設定	312
17.2. セキュリティー保護されたルート	335
第18章 INGRESS クラスタトラフィックの設定	340
18.1. INGRESS クラスタトラフィックの設定の概要	340
18.2. サービスの EXTERNALIP の設定	341
18.3. INGRESS コントローラーを使用した INGRESS クラスタの設定	347
18.4. ロードバランサーを使用した INGRESS クラスタの設定	352
18.5. ネットワークロードバランサーを使用した AWS での INGRESS クラスタトラフィックの設定	356

18.6. サービスの外部 IP を使用した INGRESS クラスタートラフィックの設定	359
18.7. NODEPORT を使用した INGRESS クラスタートラフィックの設定	361
第19章 KUBERNETES NMSTATE	364
19.1. KUBERNETES NMSTATE OPERATOR について	364
19.2. ノードのネットワーク状態の確認	365
19.3. ノードのネットワーク設定の更新	366
19.4. ノードのネットワーク設定のトラブルシューティング	377
第20章 クラスタ全体のプロキシの設定	383
20.1. 前提条件	383
20.2. クラスタ全体のプロキシの有効化	383
20.3. クラスタ全体のプロキシの削除	385
第21章 カスタム PKI の設定	387
21.1. インストール時のクラスタ全体のプロキシの設定	387
21.2. クラスタ全体のプロキシの有効化	389
21.3. OPERATOR を使用した証明書の挿入	391
第22章 RHOSP での負荷分散	393
22.1. KURYR SDN を使用した OCTAVIA OVN ロードバランサープロバイダードライバーの使用	393
22.2. OCTAVIA を使用したアプリケーショントラフィック用のクラスタのスケーリング	394
22.3. RHOSP OCTAVIA を使用した INGRESS トラフィックのスケーリング	396
22.4. 外部ロードバランサーの設定	398
第23章 METALLB を使用した負荷分散	402
23.1. METALLB および METALLB OPERATOR について	402
23.2. METALLB OPERATOR のインストール	407
23.3. METALLB アドレスプールの設定	411
23.4. METALLB を使用するためのサービスの設定	414
第24章 セカンダリーインターフェイスメトリクスのネットワーク割り当てへの関連付け	419
24.1. モニタリングのためのセカンダリーネットワークメトリックの拡張	419

第1章 ネットワークについて

クラスター管理者は、クラスターで実行されるアプリケーションを外部トラフィックに公開し、ネットワーク接続のセキュリティを保護するための複数のオプションがあります。

- ノードポートやロードバランサーなどのサービスタイプ
- **Ingress** や **Route** などの API リソース

デフォルトで、Kubernetes は各 Pod に、Pod 内で実行しているアプリケーションの内部 IP アドレスを割り当てます。Pod とそのコンテナはネットワーク接続が可能ですが、クラスター外のクライアントにはネットワークアクセスがありません。アプリケーションを外部トラフィックに公開する場合、各 Pod に IP アドレスを割り当てると、ポートの割り当て、ネットワーク、名前の指定、サービス検出、負荷分散、アプリケーション設定、移行などの点で、Pod を物理ホストや仮想マシンのように扱うことができます。



注記

一部のクラウドプラットフォームでは、169.254.169.254 IP アドレスでリッスンするメタデータ API があります。これは、IPv4 **169.254.0.0/16** CIDR ブロックのリンクローカル IP アドレスです。

この CIDR ブロックは Pod ネットワークから到達できません。これらの IP アドレスへのアクセスを必要とする Pod には、Pod 仕様の **spec.hostNetwork** フィールドを **true** に設定して、ホストのネットワークアクセスが付与される必要があります。

Pod ホストのネットワークアクセスを許可する場合、Pod に基礎となるネットワークインフラストラクチャーへの特権アクセスを付与します。

1.1. OPENSIFT CONTAINER PLATFORM DNS

フロントエンドサービスやバックエンドサービスなど、複数のサービスを実行して複数の Pod で使用している場合、フロントエンド Pod がバックエンドサービスと通信できるように、ユーザー名、サービス IP などの環境変数を作成します。サービスが削除され、再作成される場合には、新規の IP アドレスがそのサービスに割り当てられるので、フロントエンド Pod がサービス IP の環境変数の更新された値を取得するには、これを再作成する必要があります。さらに、バックエンドサービスは、フロントエンド Pod を作成する前に作成し、サービス IP が正しく生成され、フロントエンド Pod に環境変数として提供できるようにする必要があります。

そのため、OpenShift Container Platform には DNS が組み込まれており、これにより、サービスは、サービス IP/ポートと共にサービス DNS によって到達可能になります。

1.2. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR

OpenShift Container Platform クラスターを作成すると、クラスターで実行している Pod およびサービスにはそれぞれ独自の IP アドレスが割り当てられます。IP アドレスは、近くで実行されている他の Pod やサービスからアクセスできますが、外部クライアントの外部からはアクセスできません。Ingress Operator は **IngressController** API を実装し、OpenShift Container Platform クラスターサービスへの外部アクセスを可能にするコンポーネントです。

Ingress Operator を使用すると、ルーティングを処理する 1 つ以上の HAProxy ベースの **Ingress コントローラー** をデプロイおよび管理することにより、外部クライアントがサービスにアクセスできるようになります。OpenShift Container Platform **Route** および Kubernetes **Ingress** リソースを指定して、トラ

フィックをルーティングするために Ingress Operator を使用します。**endpointPublishingStrategy** タイプおよび内部負荷分散を定義する機能などの Ingress コントローラー内の設定は、Ingress コントローラーエンドポイントを公開する方法を提供します。

1.2.1. ルートと Ingress の比較

OpenShift Container Platform の Kubernetes Ingress リソースは、クラスター内で Pod として実行される共有ルーターサービスと共に Ingress コントローラーを実装します。Ingress トラフィックを管理する最も一般的な方法は Ingress コントローラーを使用することです。他の通常の Pod と同様にこの Pod をスケーリングし、複製できます。このルーターサービスは、オープンソースのロードバランサーソリューションである [HAProxy](#) をベースとしています。

OpenShift Container Platform ルートは、クラスターのサービスに Ingress トラフィックを提供します。ルートは、Blue-Green デプロイメント向けに TLS 再暗号化、TLS パススルー、分割トラフィックなどの標準の Kubernetes Ingress コントローラーでサポートされない可能性のある高度な機能を提供します。

Ingress トラフィックは、ルートを介してクラスターのサービスにアクセスします。ルートおよび Ingress は、Ingress トラフィックを処理する主要なリソースです。Ingress は、外部要求を受け入れ、ルートに基づいてそれらを委譲するなどのルートと同様の機能を提供します。ただし、Ingress では、特定タイプの接続 (HTTP/2、HTTPS およびサーバー名 ID(SNI)、ならび証明書を使用した TLS のみを許可できます。OpenShift Container Platform では、ルートは、Ingress リソースで指定される各種の条件を満たすために生成されます。

1.3. OPENSIFT CONTAINER PLATFORM ネットワーキングの一般用語集

この用語集では、ネットワーキングコンテンツで使用される一般的な用語を定義します。

authentication

OpenShift Container Platform クラスターへのアクセスを制御するために、クラスター管理者はユーザー認証を設定し、承認されたユーザーのみがクラスターにアクセスできます。OpenShift Container Platform クラスターと対話するには、OpenShift Container Platform API に対して認証する必要があります。Open Shift Container Platform API へのリクエストで、OAuth アクセストークンまたは X.509 クライアント証明書を提供することで認証できます。

AWS Load Balancer Operator

AWS Load Balancer (ALB) Operator は、**aws-load-balancer-controller** のインスタンスをデプロイおよび管理します。

Cluster Network Operator

Cluster Network Operator (CNO) は、OpenShift Container Platform クラスター内のクラスターネットワークコンポーネントをデプロイおよび管理します。これには、インストール中にクラスター用に選択された Container Network Interface (CNI) のデフォルトネットワークプロバイダープラグインのデプロイメントが含まれます。

設定マップ

ConfigMap は、設定データを Pod に注入する方法を提供します。タイプ **ConfigMap** のボリューム内の ConfigMap に格納されたデータを参照できます。Pod で実行しているアプリケーションは、このデータを使用できます。

カスタムリソース (CR)

CR は Kubernetes API の拡張です。カスタムリソースを作成できます。

DNS

クラスター DNS は、Kubernetes サービスの DNS レコードを提供する DNS サーバーです。Kubernetes により開始したコンテナは、DNS 検索にこの DNS サーバーを自動的に含めます。

DNS Operator

DNS Operator は、CoreDNS をデプロイして管理し、Pod に名前解決サービスを提供します。これにより、OpenShift Container Platform で DNS ベースの Kubernetes サービス検出が可能になります。

deployment

アプリケーションのライフサイクルを維持する Kubernetes リソースオブジェクト。

domain

ドメインは、Ingress Controller によってサービスされる DNS 名です。

egress

Pod からのネットワークのアウトバウンドトラフィックを介して外部とデータを共有するプロセス。

外部 DNS Operator

外部 DNS Operator は、ExternalDNS をデプロイして管理し、外部 DNS プロバイダーから OpenShift Container Platform へのサービスおよびルートの名前解決を提供します。

HTTP ベースのルート

HTTP ベースのルートとは、セキュアではないルートで、基本的な HTTP ルーティングプロトコルを使用してセキュリティ保護されていないアプリケーションポートでサービスを公開します。

Ingress

OpenShift Container Platform の Kubernetes Ingress リソースは、クラスター内で Pod として実行される共有ルーターサービスと共に Ingress コントローラーを実装します。

Ingress コントローラー

Ingress Operator は Ingress Controller を管理します。Ingress コントローラーの使用は、OpenShift Container Platform クラスターへの外部アクセスを許可するための最も一般的な方法です。

インストーラーでプロビジョニングされるインフラストラクチャー

インストールプログラムは、クラスターが実行されるインフラストラクチャーをデプロイして設定します。

kubelet

コンテナが Pod で実行されていることを確認するために、クラスター内の各ノードで実行されるプライマリーノードエージェント。

Kubernetes NMState Operator

Kubernetes NMState Operator は、NMState の OpenShift Container Platform クラスターのノード間でステートドリブンのネットワーク設定を実行するための Kubernetes API を提供します。

kube-proxy

Kube-proxy は、各ノードで実行するプロキシサービスであり、外部ホストがサービスを利用できるようにするのに役立ちます。リクエストを正しいコンテナに転送するのに役立ち、基本的な負荷分散を実行できます。

ロードバランサー

OpenShift Container Platform は、ロードバランサーを使用して、クラスターの外部からクラスターで実行されているサービスと通信します。

Metal LB オペレーター

クラスター管理者は、MetalLB Operator をクラスターに追加し、タイプ **LoadBalancer** のサービスがクラスターに追加されると、MetalLB はサービスの外部 IP アドレスを追加できます。

multicast

IP マルチキャストを使用すると、データが多数の IP アドレスに同時に配信されます。

namespace

namespace は、すべてのプロセスから見える特定のシステムリソースを分離します。namespace 内では、その namespace のメンバーであるプロセスのみがそれらのリソースを参照できます。

networking

OpenShift Container Platform クラスターのネットワーク情報。

node

OpenShift Container Platform クラスター内のワーカーマシン。ノードは、仮想マシン (VM) または物理マシンのいずれかです。

OpenShift Container Platform Ingress Operator

Ingress Operator は **IngressController** API を実装し、OpenShift Container Platform サービスへの外部アクセスを可能にするコンポーネントです。

Pod

OpenShift Container Platform クラスターで実行されている、ボリュームや IP アドレスなどの共有リソースを持つ1つ以上のコンテナ。Pod は、定義、デプロイ、および管理される最小のコンピュータ単位です。

PTP Operator

PTP Operator は、**linuxptp** サービスを作成し、管理します。

route

OpenShift Container Platform ルートは、クラスターのサービスに Ingress トラフィックを提供します。ルートは、Blue-Green デプロイメント向けに TLS 再暗号化、TLS パススルー、分割トラフィックなどの標準の Kubernetes Ingress コントローラーでサポートされない可能性のある高度な機能を提供します。

スケーリング

リソース容量の増減。

サービス

一連の Pod で実行中のアプリケーションを公開します。

シングルルート I/O 仮想化 (SR-IOV) Network Operator

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator は、クラスターで SR-IOV ネットワークデバイスおよびネットワーク割り当てを管理します。

ソフトウェア定義ネットワーク (SDN)

OpenShift Container Platform は、Software Defined Networking (SDN) アプローチを使用して、クラスターのネットワークを統合し、OpenShift Container Platform クラスターの Pod 間の通信を可能にします。

SCTP (Stream Control Transmission Protocol)

SCTP は、IP ネットワークの上部で実行される信頼できるメッセージベースのプロトコルです。

taint

テイントと容認により、Pod が適切なノードに確実にスケジューラされます。ノードに1つ以上のテイントを適用できます。

容認

Pod に容認を適用できます。Tolerations を使用すると、スケジューラーは、テイントが一致する Pod をスケジューラできます。

Web コンソール

OpenShift Container Platform を管理するためのユーザーインターフェイス (UI)。

第2章 ホストへのアクセス

OpenShift Container Platform インスタンスにアクセスして、セキュアシェル (SSH) アクセスでコントロールプレーンノードにアクセスするために bastion ホストを作成する方法を学びます。

2.1. インストーラーでプロビジョニングされるインフラストラクチャクラスターでの AMAZON WEB SERVICES のホストへのアクセス

OpenShift Container Platform インストーラーは、OpenShift Container Platform クラスターにプロビジョニングされる Amazon Elastic Compute Cloud (Amazon EC2) インスタンスのパブリック IP アドレスを作成しません。OpenShift Container Platform ホストに対して SSH を実行できるようにするには、以下の手順を実行する必要があります。

手順

1. **openshift-install** コマンドで作成される仮想プライベートクラウド (VPC) に対する SSH アクセスを可能にするセキュリティグループを作成します。
2. インストーラーが作成したパブリックサブネットのいずれかに Amazon EC2 インスタンスを作成します。
3. パブリック IP アドレスを、作成した Amazon EC2 インスタンスに関連付けます。
OpenShift Container Platform のインストールとは異なり、作成した Amazon EC2 インスタンスを SSH キーペアに関連付ける必要があります。これにはインターネットを OpenShift Container Platform クラスターの VPC にブリッジ接続するための SSH bastion としてのみの単純な機能しかないので、このインスタンスにどのオペレーティングシステムを選択しても問題ありません。どの Amazon Machine Image (AMI) を使用するかについては、注意が必要です。たとえば、Red Hat Enterprise Linux CoreOS (RHCOS) では、インストーラーと同様に、Ignition でキーを指定することができます。
4. Amazon EC2 インスタンスをプロビジョニングし、これに対して SSH を実行した後に、OpenShift Container Platform インストールに関連付けた SSH キーを追加する必要があります。このキーは bastion インスタンスのキーとは異なる場合がありますが、異なるキーにしなければならない訳ではありません。



注記

直接の SSH アクセスは、障害復旧を目的とする場合にのみ推奨されます。Kubernetes API が応答する場合、特権付き Pod を代わりに実行します。

5. **oc get nodes** を実行し、出力を検査し、マスターであるノードのいずれかを選択します。ホスト名は **ip-10-0-1-163.ec2.internal** に類似したものになります。
6. Amazon EC2 に手動でデプロイした bastion SSH ホストから、そのコントロールプレーンホストに SSH を実行します。インストール時に指定したものと同一 SSH キーを使用するようにします。

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

第3章 ネットワーキング OPERATOR の概要

OpenShift Container Platform は、複数のタイプのネットワーキング Operator をサポートします。これらのネットワーク Operator を使用して、クラスターネットワークを管理できます。

3.1. CLUSTER NETWORK OPERATOR

Cluster Network Operator (CNO) は、OpenShift Container Platform クラスター内のクラスターネットワークコンポーネントをデプロイおよび管理します。これには、インストール中にクラスター用に選択された Container Network Interface (CNI) のデフォルトネットワークプロバイダープラグインのデプロイメントが含まれます。詳細は、[OpenShift Container Platform の Cluster Network Operator](#) を参照してください。

3.2. DNS OPERATOR

DNS Operator は、CoreDNS をデプロイして管理し、Pod に名前解決サービスを提供します。これにより、OpenShift Container Platform で DNS ベースの Kubernetes サービス検出が可能になります。詳細は、[OpenShift Container Platform の DNS Operator](#) を参照してください。

3.3. INGRESS OPERATOR

OpenShift Container Platform クラスターを作成すると、クラスターで実行している Pod およびサービスにはそれぞれの IP アドレスが割り当てられます。IP アドレスは、近くで実行されている他の Pod やサービスからアクセスできますが、外部クライアントの外部からはアクセスできません。Ingress Operator は IngressController API を実装し、OpenShift Container Platform クラスターサービスへの外部アクセスを可能にします。詳細は、[OpenShift Container Platform の Ingress Operator](#) を参照してください。

第4章 OPENSIFT CONTAINER PLATFORM における CLUSTER NETWORK OPERATOR

Cluster Network Operator (CNO) は、インストール時にクラスター用に選択される Container Network Interface (CNI) デフォルトネットワークプロバイダープラグインを含む、OpenShift Container Platform クラスターの各種のクラスターネットワークコンポーネントをデプロイし、これらを管理します。

4.1. CLUSTER NETWORK OPERATOR

Cluster Network Operator は、**operator.openshift.io** API グループから **network** API を実装します。Operator は、デーモンセットを使用して OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグイン、またはクラスターのインストール時に選択したデフォルトネットワークプロバイダープラグインをデプロイします。

手順

Cluster Network Operator は、インストール時に Kubernetes **Deployment** としてデプロイされます。

1. 以下のコマンドを実行して Deployment のステータスを表示します。

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

出力例

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

2. 以下のコマンドを実行して、Cluster Network Operator の状態を表示します。

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

出力例

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

以下のフィールドは、Operator のステータス (**AVAILABLE**、**PROGRESSING**、および **DEGRADED**) についての情報を提供します。**AVAILABLE** フィールドは、Cluster Network Operator が Available ステータス条件を報告する場合に **True** になります。

4.2. クラスターネットワーク設定の表示

すべての新規 OpenShift Container Platform インストールには、**cluster** という名前の **network.config** オブジェクトがあります。

手順

- **oc describe** コマンドを使用して、クラスターネットワーク設定を表示します。

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


出力例

```

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

```

- ❶ **Spec** フィールドは、クラスターネットワークの設定済みの状態を表示します。
- ❷ **Status** フィールドは、クラスターネットワークの現在の状態を表示します。

4.3. CLUSTER NETWORK OPERATOR のステータス表示

oc describe コマンドを使用して、Cluster Network Operator のステータスを検査し、その詳細を表示することができます。

手順

- 以下のコマンドを実行して、Cluster Network Operator のステータスを表示します。

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

4.4. CLUSTER NETWORK OPERATOR ログの表示

oc logs コマンドを使用して、Cluster Network Operator ログを表示できます。

手順

- 以下のコマンドを実行して、Cluster Network Operator のログを表示します。

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

4.5. CLUSTER NETWORK OPERATOR (CNO) の設定

クラスターネットワークの設定は、Cluster Network Operator (CNO) 設定の一部として指定され、**cluster** という名前のカスタムリソース (CR) オブジェクトに保存されます。CR は **operator.openshift.io** API グループの **Network** API のフィールドを指定します。

CNO 設定は、**Network.config.openshift.io** API グループの **Network** API からクラスターのインストール時に以下のフィールドを継承し、これらのフィールドは変更できません。

clusterNetwork

Pod IP アドレスの割り当てに使用する IP アドレスプール。

serviceNetwork

サービスの IP アドレスプール。

defaultNetwork.type

OpenShift SDN または OVN-Kubernetes などのクラスターネットワークプロバイダー。



注記

クラスターのインストール後に、直前のセクションで一覧表示されているフィールドを変更することはできません。

defaultNetwork オブジェクトのフィールドを **cluster** という名前の CNO オブジェクトに設定することにより、クラスターのクラスターネットワークプロバイダー設定を指定できます。

4.5.1. Cluster Network Operator 設定オブジェクト

Cluster Network Operator (CNO) のフィールドは以下の表で説明されています。

表4.1 Cluster Network Operator 設定オブジェクト


フィールド	型	説明
metadata.name	string	CNO オブジェクトの名前。この名前は常に cluster です。
spec.clusterNetwork	array	<p>Pod ID アドレスの割り当て、サブネット接頭辞の長さのクラスター内の個別ノードへの割り当てに使用される IP アドレスのブロックを指定する一覧です。以下に例を示します。</p> <pre>spec: clusterNetwork: - cidr: 10.128.0.0/19 hostPrefix: 23 - cidr: 10.128.32.0/19 hostPrefix: 23</pre> <p>この値は読み取り専用であり、クラスターのインストール時に cluster という名前の Network.config.openshift.io オブジェクトから継承されます。</p>

フィールド	型	説明
spec.serviceNetwork	array	<p>サービスの IP アドレスのブロック。OpenShift SDN および OVN-Kubernetes Container Network Interface (CNI) ネットワークプロバイダーは、サービスネットワークの単一 IP アドレスブロックのみをサポートします。以下に例を示します。</p> <pre>spec: serviceNetwork: - 172.30.0.0/14</pre> <p>この値は読み取り専用であり、クラスターのインストール時に cluster という名前の Network.config.openshift.io オブジェクトから継承されます。</p>
spec.defaultNetwork	object	<p>クラスターネットワークの Container Network Interface (CNI) ネットワークプロバイダーを設定します。</p>
spec.kubeProxyConfig	object	<p>このオブジェクトのフィールドは、kube-proxy 設定を指定します。OVN-Kubernetes クラスターネットワークプロバイダーを使用している場合、kube-proxy 設定は機能しません。</p>

defaultNetwork オブジェクト設定

defaultNetwork オブジェクトの値は、以下の表で定義されます。

表4.2 defaultNetwork オブジェクト

フィールド	型	説明
type	string	<p>OpenShiftSDN または OVNKubernetes のいずれか。クラスターネットワークプロバイダーはインストール時に選択されます。この値は、クラスターのインストール後は変更できません。</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;">  </div> <div style="flex: 1;"> <p>注記</p> <p>OpenShift Container Platform はデフォルトで、OpenShift SDN Container Network Interface (CNI) クラスターネットワークプロバイダーを使用します。</p> </div> </div>
openshiftSDNConfig	object	<p>このオブジェクトは OpenShift SDN クラスターネットワークプロバイダーにのみ有効です。</p>
ovnKubernetesConfig	object	<p>このオブジェクトは OVN-Kubernetes クラスターネットワークプロバイダーにのみ有効です。</p>

OpenShift SDN CNI クラスターネットワークプロバイダーの設定

以下の表は、OpenShift SDN Container Network Interface (CNI) クラスタネットワークプロバイダーの設定フィールドについて説明しています。

表4.3 openshiftSDNConfig オブジェクト

フィールド	型	説明
mode	string	OpenShiftSDN のネットワーク分離モード。
mtu	integer	VXLAN オーバーレイネットワークの最大転送単位 (MTU)。通常、この値は自動的に設定されます。
vxlanPort	integer	すべての VXLAN パケットに使用するポート。デフォルト値は 4789 です。



注記

クラスタのインストール時にのみクラスタネットワークプロバイダーの設定を変更することができます。

OpenShift SDN 設定の例

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

OVN-Kubernetes CNI クラスタネットワークプロバイダーの設定

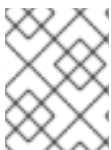
以下の表は OVN-Kubernetes CNI クラスタネットワークプロバイダーの設定フィールドについて説明しています。

表4.4 ovnKubernetesConfig object

フィールド	型	説明
mtu	integer	Geneve (Generic Network Virtualization Encapsulation) オーバーレイネットワークの MTU (maximum transmission unit)。通常、この値は自動的に設定されます。
genevePort	integer	Geneve オーバーレイネットワークの UDP ポート。
ipsecConfig	object	フィールドがある場合、IPsec はクラスタに対して有効にされます。
policyAuditConfig	object	ネットワークポリシー監査ロギングをカスタマイズする設定オブジェクトを指定します。指定されていない場合は、デフォルトの監査ログ設定が使用されます。

表4.5 policyAuditConfig object

フィールド	型	説明
rateLimit	integer	ノードごとに毎秒生成されるメッセージの最大数。デフォルト値は、1秒あたり 20 メッセージです。
maxFileSize	integer	監査ログの最大サイズ (バイト単位)。デフォルト値は 50000000 または 50MB です。
destination	string	以下の追加の監査ログターゲットのいずれかになります。 libc ホスト上の journald プロセスの libc syslog() 関数。 udp:<host>:<port> syslog サーバー。<host>:<port> を syslog サーバーのホストおよびポートに置き換えます。 unix:<file> <file> で指定された Unix ドメインソケットファイル。 null 監査ログを追加のターゲットに送信しないでください。
syslogFacility	string	RFC5424 で定義される kern などの syslog ファシリティ。デフォルト値は local0 です。



注記

クラスターのインストール時にのみクラスターネットワークプロバイダーの設定を変更することができます。

OVN-Kubernetes 設定の例

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

kubeProxyConfig オブジェクト設定

kubeProxyConfig オブジェクトの値は以下の表で定義されます。

表4.6 kubeProxyConfig オブジェクト

フィールド	型	説明
-------	---	----

フィールド	型	説明
iptablesSyncPeriod	string	<p>iptables ルールの更新期間。デフォルト値は 30s です。有効な接尾辞には、s、m、および h などが含まれ、これらについては、Go time パッケージ ドキュメントで説明されています。</p> <div style="display: flex; align-items: flex-start;">  <div> <p>注記</p> <p>OpenShift Container Platform 4.3 以降で強化されたパフォーマンスの向上により、iptablesSyncPeriod パラメーターを調整する必要はなくなりました。</p> </div> </div>
proxyArguments.iptables-min-sync-period	array	<p>iptables ルールを更新する前の最小期間。このフィールドにより、更新の頻度が高くなり過ぎないようにできます。有効な接尾辞には、s、m、および h などが含まれ、これらについては、Go time パッケージ で説明されています。デフォルト値:</p> <pre>kubeProxyConfig: proxyArguments: iptables-min-sync-period: - 0s</pre>

4.5.2. Cluster Network Operator の設定例

以下の例では、詳細な CNO 設定が指定されています。

Cluster Network Operator オブジェクトのサンプル

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

```
proxyArguments:  
  iptables-min-sync-period:  
    - 0s
```

1 2 3 クラスターのインストール時にのみ設定されます。

4.6. 関連情報

- [operator.openshift.io API グループの Network API](#)

第5章 OPENSIFT CONTAINER PLATFORM の DNS OPERATOR

DNS Operator は、Pod に対して名前解決サービスを提供するために CoreDNS をデプロイし、これを管理し、OpenShift Container Platform での DNS ベースの Kubernetes サービス検出を可能にします。

5.1. DNS OPERATOR

DNS Operator は、**operator.openshift.io** API グループから **dns** API を実装します。この Operator は、デーモンセットを使用して CoreDNS をデプロイし、デーモンセットのサービスを作成し、kubelet を Pod に対して名前解決に CoreDNS サービス IP を使用するよう指示するように設定します。

手順

DNS Operator は、インストール時に **Deployment** オブジェクトを使用してデプロイされます。

1. **oc get** コマンドを使用してデプロイメントのステータスを表示します。

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

出力例

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

2. **oc get** コマンドを使用して DNS Operator の状態を表示します。

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

出力例

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

AVAILABLE、**PROGRESSING** および **DEGRADED** は、Operator のステータスについての情報を提供します。**AVAILABLE** は、CoreDNS デーモンセットからの1つ以上の Pod が **Available** ステータス条件を報告する場合は **True** になります。

5.2. DNS OPERATOR MANAGEMENTSTATE の変更

DNS は CoreDNS コンポーネントを管理し、クラスター内の Pod およびサービスの名前解決サービスを提供します。DNS Operator の **managementState** はデフォルトで **Managed** に設定されます。これは、DNS Operator がそのリソースをアクティブに管理できることを意味します。これを **Unmanaged** に変更できます。つまり、DNS Operator がそのリソースを管理していないことを意味します。

以下は、DNS Operator **managementState** を変更するためのユースケースです。

- 開発者であり、CoreDNS の問題が修正されているかどうかを確認するために設定変更をテストする必要があります。**managementState** を **Unmanaged** に設定すると、DNS Operator により修正が上書きされないようになります。

- クラスター管理者であり、CoreDNS の問題が報告されていますが、問題が修正されるまで回避策を適用する必要があります。DNS Operator の **managementState** フィールドを **Unmanaged** に設定して、回避策を適用できます。

手順

- **managementState** DNS Operator を変更します。

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

5.3. DNS POD 配置の制御

DNS Operator には、CoreDNS 用と **/etc/hosts** ファイルを管理するための2つのデーモンセットがあります。**/etc/hosts** に設定されたデーモンは、イメージのプルをサポートするクラスターイメージレジストリーのエントリーを追加するために、すべてのノードホストで実行する必要があります。セキュリティポリシーにより、ノードのペア間の通信が禁止され、CoreDNS のデーモンセットがすべてのノードで実行できなくなります。

クラスター管理者は、カスタムノードセレクターを使用して、CoreDNS のデーモンセットを特定のノードで実行するか、または実行しないように設定できます。

前提条件

- **oc** CLI をインストールしていること。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていること。

手順

- 特定のノード間の通信を防ぐには、**spec.nodePlacement.nodeSelector** API フィールドを設定します。

1. **default** という名前の DNS Operator オブジェクトを変更します。

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

2. **spec.nodePlacement.nodeSelector** API フィールドにコントロールプレーンノードのみが含まれるノードセレクターを指定します。

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

- CoreDNS のデーモンセットをノードで実行されるようにするには、テイントおよび容認を設定します。

1. **default** という名前の DNS Operator オブジェクトを変更します。

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

2. テイントのテイントキーおよび容認を指定します。

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

- ❶ ティントが **dns-only** である場合、それは無制限に許容できます。 **tolerationSeconds** は省略できます。

5.4. デフォルト DNS の表示

すべての新規 OpenShift Container Platform インストールには、**default** という名前の **dns.operator** があります。

手順

1. **oc describe** コマンドを使用してデフォルトの **dns** を表示します。

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

出力例

```
Name:      default
Namespace:
Labels:    <none>
Annotations: <none>
API Version: operator.openshift.io/v1
Kind:      DNS
...
Status:
  Cluster Domain: cluster.local ❶
  Cluster IP:     172.30.0.10 ❷
...
```

- ❶ Cluster Domain フィールドは、完全修飾 Pod およびサービスドメイン名を作成するために使用されるベース DNS ドメインです。
- ❷ クラスタ IP は、Pod が名前解決のためにクエリーするアドレスです。IP は、サービス CIDR 範囲の 10 番目のアドレスで定義されます。

2. クラスタのサービス CIDR を見つけるには、**oc get** コマンドを使用します。

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

出力例

```
[172.30.0.0/16]
```

5.5. DNS 転送の使用

DNS 転送を使用すると、指定のゾーンにどのネームサーバーを使用するかを指定することで、ゾーンごとに **/etc/resolv.conf** で特定される転送設定をオーバーライドできます。転送されるゾーンが OpenShift Container Platform によって管理される Ingress ドメインである場合、アップストリームネームサーバーがドメインについて認証される必要があります。

手順

1. **default** という名前の DNS Operator オブジェクトを変更します。

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

これにより、**Server** に基づく追加のサーバー設定ブロックを使用して **dns-default** という名前の ConfigMap を作成し、更新できます。クエリーに一致するゾーンを持つサーバーがない場合、名前解決は **/etc/resolv.conf** で指定されたネームサーバーにフォールバックします。

DNS の例

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
    - name: foo-server ①
      zones: ②
        - example.com
      forwardPlugin:
        upstreams: ③
          - 1.1.1.1
          - 2.2.2.2:5353
    - name: bar-server
      zones:
        - bar.com
        - example.com
      forwardPlugin:
        upstreams:
          - 3.3.3.3
          - 4.4.4.4:5454
```

- ① **name** は、**rfc6335** サービス名の構文に準拠する必要があります。
- ② **zones** は、**rfc1123** の **subdomain** の定義に準拠する必要があります。クラスタードメインの **cluster.local** は、**zones** の無効な **subdomain** です。
- ③ **forwardPlugin** ごとに最大 15 の **upstreams** が許可されます。



注記

servers が定義されていないか、または無効な場合、ConfigMap にはデフォルトサーバーのみが含まれます。

2. ConfigMap を表示します。

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

以前のサンプル DNS に基づく DNS ConfigMap の例

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

- ① **forwardPlugin** への変更により、CoreDNS デモンセットのローリング更新がトリガーされます。

関連情報

- DNS 転送の詳細は、[CoreDNS forward のドキュメント](#) を参照してください。

5.6. DNS OPERATOR のステータス

oc describe コマンドを使用して、DNS Operator のステータスを検査し、その詳細を表示することができます。

手順

DNS Operator のステータスを表示します。

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

5.7. DNS OPERATOR ログ

oc logs コマンドを使用して、DNS Operator ログを表示できます。

手順

DNS Operator のログを表示します。

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

第6章 OPENSIFT CONTAINER PLATFORM の INGRESS OPERATOR

6.1. OPENSIFT CONTAINER PLATFORM INGRESS OPERATOR

OpenShift Container Platform クラスターを作成すると、クラスターで実行している Pod およびサービスにはそれぞれ独自の IP アドレスが割り当てられます。IP アドレスは、近くで実行されている他の Pod やサービスからアクセスできますが、外部クライアントの外部からはアクセスできません。Ingress Operator は **IngressController** API を実装し、OpenShift Container Platform クラスターサービスへの外部アクセスを可能にするコンポーネントです。

Ingress Operator を使用すると、ルーティングを処理する1つ以上の HAProxy ベースの **Ingress コントローラー** をデプロイおよび管理することにより、外部クライアントがサービスにアクセスできるようになります。OpenShift Container Platform **Route** および Kubernetes **Ingress** リソースを指定して、トラフィックをルーティングするために Ingress Operator を使用します。**endpointPublishingStrategy** タイプおよび内部負荷分散を定義する機能などの Ingress コントローラー内の設定は、Ingress コントローラーエンドポイントを公開する方法を提供します。

6.2. INGRESS 設定アセット

インストールプログラムでは、**config.openshift.io** API グループの **Ingress** リソースでアセットを生成します (**cluster-ingress-02-config.yml**)。

Ingress リソースの YAML 定義

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

インストールプログラムは、このアセットを **manifests/** ディレクトリーの **cluster-ingress-02-config.yml** ファイルに保存します。この **Ingress** リソースは、Ingress のクラスター全体の設定を定義します。この Ingress 設定は、以下のように使用されます。

- Ingress Operator は、クラスター Ingress 設定のドメインを、デフォルト Ingress コントローラーのドメインとして使用します。
- OpenShift API Server Operator は、クラスター Ingress 設定からのドメインを使用します。このドメインは、明示的なホストを指定しない **Route** リソースのデフォルトホストを生成する際にも使用されます。

6.3. INGRESS コントローラー設定パラメーター

ingresscontrollers.operator.openshift.io リソースは以下の設定パラメーターを提供します。

パラメーター

説明

パラメーター	説明
domain	<p>domain は Ingress コントローラーによって提供される DNS 名で、複数の機能を設定するために使用されます。</p> <ul style="list-style-type: none"> ● LoadBalancerService エンドポイント公開ストラテジーの場合、domain は DNS レコードを設定するために使用されます。endpointPublishingStrategy を参照してください。 ● 生成されるデフォルト証明書を使用する場合、証明書は domain およびその subdomains で有効です。defaultCertificate を参照してください。 ● この値は個別の Route ステータスに公開され、ユーザーは外部 DNS レコードのターゲット先を認識できるようにします。 <p>domain 値はすべての Ingress コントローラーの中でも固有の値であり、更新できません。</p> <p>空の場合、デフォルト値は ingress.config.openshift.io/cluster.spec.domain です。</p>
replicas	<p>replicas は Ingress コントローラーレプリカの必要な数です。設定されていない場合、デフォルト値は 2 になります。</p>
endpointPublishingStrategy	<p>endpointPublishingStrategy は Ingress コントローラーエンドポイントを他のネットワークに公開し、ロードバランサーの統合を有効にし、他のシステムへのアクセスを提供するために使用されます。</p> <p>設定されていない場合、デフォルト値は infrastructure.config.openshift.io/cluster.status.platform をベースとします。</p> <ul style="list-style-type: none"> ● AWS: LoadBalancerService (外部スコープあり) ● Azure: LoadBalancerService (外部スコープあり) ● GCP: LoadBalancerService (外部スコープあり) ● Bare metal: NodePortService ● その他: HostNetwork <p>ほとんどのプラットフォームの場合、endpointPublishingStrategy 値は更新できません。ただし、GCP では、loadbalancer.providerParameters.gcp.clientAccess サブフィールドを設定できます。</p>

パラメーター	説明
<p>defaultCertificate</p>	<p>defaultCertificate 値は、Ingress コントローラーによって提供されるデフォルト証明書が含まれるシークレットへの参照です。ルートが独自の証明書を指定しない場合、defaultCertificate が使用されます。</p> <p>シークレットには以下のキーおよびデータが含まれる必要があります: *tls.crt: 証明書ファイルコンテンツ *tls.key: キーファイルコンテンツ</p> <p>設定されていない場合、ワイルドカード証明書は自動的に生成され、使用されます。証明書は Ingress コントローラーの domain および subdomains で有効であり、生成された証明書 CA はクラスターの信頼ストアに自動的に統合されます。</p> <p>使用中の証明書 (生成されるか、ユーザー指定の場合かを問わない) は OpenShift Container Platform のビルトイン OAuth サーバーに自動的に統合されます。</p>
<p>namespaceSelector</p>	<p>namespaceSelector は、Ingress コントローラーによって提供される namespace セットをフィルターするために使用されます。これはシャードの実装に役立ちます。</p>
<p>routeSelector</p>	<p>routeSelector は、Ingress コントローラーによって提供される Routes のセットをフィルターするために使用されます。これはシャードの実装に役立ちます。</p>
<p>nodePlacement</p>	<p>nodePlacement は、Ingress コントローラーのスケジュールに対する明示的な制御を有効にします。</p> <p>設定されていない場合は、デフォルト値が使用されます。</p> <div data-bbox="517 1323 625 1794" style="border: 1px solid #ccc; padding: 5px; margin: 10px 0;">  </div> <p>注記</p> <p>nodePlacement パラメーターには、nodeSelector と tolerations の2つの部分が含まれます。以下に例を示します。</p> <pre data-bbox="703 1518 1069 1794"> nodePlacement: nodeSelector: matchLabels: kubernetes.io/os: linux tolerations: - effect: NoSchedule operator: Exists </pre>

パラメーター	説明
<p>tlsSecurityProfile</p>	<p>tlsSecurityProfile は、Ingress コントローラーの TLS 接続の設定を指定します。</p> <p>これが設定されていない場合、デフォルト値は apiservers.config.openshift.io/cluster リソースをベースとして設定されます。</p> <p>Old、Intermediate、および Modern のプロファイルタイプを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース X.Y.Z にデプロイされた Intermediate プロファイルを使用する仕様がある場合、リリース X.Y.Z+1 へのアップグレードにより、新規のプロファイル設定が Ingress コントローラーに適用され、ロールアウトが生じる可能性があります。</p> <p>Ingress コントローラーの最小 TLS バージョンは 1.1 で、最大 TLS バージョンは 1.3 です。</p> <p> 注記</p> <p>設定されたセキュリティープロファイルの暗号および最小 TLS バージョンが TLSProfile ステータスに反映されます。</p> <p> 重要</p> <p>Ingress Operator は TLS 1.0 の Old または Custom プロファイルを 1.1 に変換します。</p>
<p>clientTLS</p>	<p>clientTLS は、クラスターおよびサービスへのクライアントアクセスを認証します。その結果、相互 TLS 認証が有効になります。設定されていない場合、クライアント TLS は有効になっていません。</p> <p>ClientTLS には、必要なサブフィールド spec.clientTLS.clientCertificatePolicy および spec.clientTLS.ClientCA があります。</p> <p>ClientCertificatePolicy サブフィールドは、Required または Optional の 2 つの値のいずれかを受け入れます。ClientCA サブフィールドは、openshift-config namespace にある ConfigMap を指定します。ConfigMap には CA 証明書バンドルが含まれている必要があります。AllowedSubjectPatterns は、要求をフィルターするために有効なクライアント証明書の識別名と照合される正規表現の一覧を指定する任意の値です。正規表現は PCRE 構文を使用する必要があります。1 つ以上のパターンがクライアント証明書の識別名と一致している必要があります。一致しない場合、Ingress コントローラーは証明書を拒否し、接続を拒否します。指定しないと、Ingress コントローラーは識別名に基づいて証明書を拒否しません。</p>

パラメーター	説明
routeAdmission	<p>routeAdmission は、複数の namespace での要求の許可または拒否など、新規ルート要求を処理するためのポリシーを定義します。</p> <p>namespaceOwnership は、namespace 間でホスト名の要求を処理する方法を記述します。デフォルトは Strict です。</p> <ul style="list-style-type: none">● Strict: ルートが複数の namespace 間で同じホスト名を要求することを許可しません。● InterNamespaceAllowed: ルートが複数の namespace 間で同じホスト名の異なるパスを要求することを許可します。 <p>wildcardPolicy は、ワイルドカードポリシーを使用するルートが Ingress コントローラーによって処理される方法を記述します。</p> <ul style="list-style-type: none">● WildcardsAllowed: ワイルドカードポリシーと共にルートが Ingress コントローラーによって許可されていることを示します。● WildcardsDisallowed: ワイルドカードポリシーの None を持つルートのみが Ingress コントローラーによって許可されることを示します。 wildcardPolicy を WildcardsAllowed から WildcardsDisallowed に更新すると、ワイルドカードポリシーの Subdomain を持つ許可されたルートが機能を停止します。これらのルートは、Ingress コントローラーによって許可されるように None のワイルドカードポリシーに対して再作成される必要があります。 WildcardsDisallowed はデフォルト設定です。

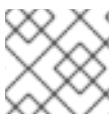
パラメーター	説明
IngressControllerLogging	<p>logging はログに記録される内容および場所のパラメーターを定義します。このフィールドが空の場合、操作ログは有効になりますが、アクセスログは無効になります。</p> <ul style="list-style-type: none"> ● access は、クライアント要求をログに記録する方法を記述します。このフィールドが空の場合、アクセスロギングは無効になります。 <ul style="list-style-type: none"> ○ destination はログメッセージの宛先を記述します。 <ul style="list-style-type: none"> ■ type はログの宛先のタイプです。 <ul style="list-style-type: none"> ● Container は、ログがサイドカーコンテナに移動することを指定します。Ingress Operator は Ingress コントローラー Pod で logs という名前のコンテナを設定し、Ingress コントローラーがログをコンテナに書き込むように設定します。管理者がこのコンテナからログを読み取るカスタムロギングソリューションを設定することが予想されます。コンテナログを使用すると、ログの割合がコンテナランタイムの容量やカスタムロギングソリューションの容量を超えるとログがドロップされることがあります。 ● Syslog は、ログが Syslog エンドポイントに送信されることを指定します。管理者は、Syslog メッセージを受信できるエンドポイントを指定する必要があります。管理者がカスタム Syslog インスタンスを設定していることが予想されます。 ■ container は Container ロギング宛先タイプのパラメーターを記述します。現在、コンテナロギングのパラメーターはないため、このフィールドは空である必要があります。 ■ syslog は、Syslog ロギング宛先タイプのパラメーターを記述します。 <ul style="list-style-type: none"> ● address は、ログメッセージを受信する syslog エンドポイントの IP アドレスです。 ● port は、ログメッセージを受信する syslog エンドポイントの UDP ポート番号です。 ● facility はログメッセージの syslog ファシリティーを指定します。このフィールドが空の場合、ファシリティーは local1 になります。それ以外の場合、有効な syslog ファシリティー (kern, user, mail, daemon, auth, syslog, lpr, news, uucp, cron, auth2, ftp, ntp, audit, alert, cron2, local0, local1, local2, local3) を指定する必要があります。 local4, local5, local6, または local7。 ○ httpLogFormat は、HTTP 要求のログメッセージの形式を指定します。このフィールドが空の場合、ログメッセージは実装のデフォルト HTTP ログ形式を使用します。HAProxy のデフォルトの HTTP ログ形式については、HAProxy ドキュメント を参照してください。

パラメーター	説明
<p>httpHeaders</p>	<p>httpHeaders は HTTP ヘッダーのポリシーを定義します。</p> <p>IngressControllerHTTPHeaders の forwardedHeaderPolicy を設定することで、Ingress コントローラーが Forwarded、X-Forwarded-For、X-Forwarded-Host、X-Forwarded-Port、X-Forwarded-Proto、および X-Forwarded-Proto-Version HTTP ヘッダーをいつどのように設定するか指定します。</p> <p>デフォルトでは、ポリシーは Append に設定されます。</p> <ul style="list-style-type: none"> ● Append は、Ingress コントローラーがヘッダーを追加するように指定し、既存のヘッダーを保持します。 ● Replace は、Ingress コントローラーがヘッダーを設定するように指定し、既存のヘッダーを削除します。 ● IfNone は、ヘッダーがまだ設定されていない場合に、Ingress コントローラーがヘッダーを設定するように指定します。 ● Never は、Ingress コントローラーがヘッダーを設定しないように指定し、既存のヘッダーを保持します。 <p>headerNameCaseAdjustments を設定して、HTTP ヘッダー名に適用できるケースの調整を指定できます。それぞれの調整は、必要な大文字化を指定して HTTP ヘッダー名として指定されます。たとえば、X-Forwarded-For を指定すると、指定された大文字化を有効にするために x-forwarded-for HTTP ヘッダーを調整する必要があることを示唆できます。</p> <p>これらの調整は、クリアテキスト、edge terminationd、および re-encrypt ルートにのみ適用され、HTTP/1 を使用する場合にのみ適用されます。</p> <p>要求ヘッダーの場合、これらの調整は haproxy.router.openshift.io/h1-adjust-case=true アノテーションを持つルートについてのみ適用されます。応答ヘッダーの場合、これらの調整はすべての HTTP 応答に適用されます。このフィールドが空の場合、要求ヘッダーは調整されません。</p>
<p>httpErrorCodePages</p>	<p>httpErrorCodePages は、カスタムの HTTP エラーコードの応答ページを指定します。デフォルトで、IngressController は IngressController イメージにビルドされたエラーページを使用します。</p>

パラメーター	説明
<p>httpCaptureCookies</p>	<p>httpCaptureCookies は、アクセスログにキャプチャーする HTTP Cookie を指定します。httpCaptureCookies フィールドが空の場合、アクセスログは Cookie をキャプチャーしません。</p> <p>キャプチャーするすべての Cookie について、次のパラメーターが IngressController 設定に含まれている必要があります。</p> <ul style="list-style-type: none"> ● name は、Cookie の名前を指定します。 ● maxLength は、Cookie の最大長を指定します。 ● matchType は、Cookie のフィールドの name が、キャプチャー Cookie 設定と完全に一致するか、キャプチャー Cookie 設定の接頭辞であるかを指定します。matchType フィールドは Exact および Prefix パラメーターを使用します。 <p>以下に例を示します。</p> <pre>httpCaptureCookies: - matchType: Exact maxLength: 128 name: MYCOOKIE</pre>
<p>httpCaptureHeaders</p>	<p>httpCaptureHeaders は、アクセスログにキャプチャーする HTTP ヘッダーを指定します。httpCaptureHeaders フィールドが空の場合、アクセスログはヘッダーをキャプチャーしません。</p> <p>httpCaptureHeaders には、アクセスログにキャプチャーするヘッダーの 2 つのリストが含まれています。ヘッダーフィールドの 2 つのリストは request と response です。どちらのリストでも、name フィールドはヘッダー名を指定し、maxLength フィールドはヘッダーの最大長を指定する必要があります。以下に例を示します。</p> <pre>httpCaptureHeaders: request: - maxLength: 256 name: Connection - maxLength: 128 name: User-Agent response: - maxLength: 256 name: Content-Type - maxLength: 256 name: Content-Length</pre>
<p>tuningOptions</p>	<p>tuningOptions は、Ingress コントローラー Pod のパフォーマンスを調整するためのオプションを指定します。</p> <ul style="list-style-type: none"> ● headerBufferBytes は、Ingress コントローラー接続セッション用に予約されるメモリーの量をバイト単位で指定します。Ingress コントローラーで HTTP/2 が有効になっている場合、この値は少なくとも 16384 である必要があります。設定されていない場合、デフォルト値は 32768 バイトになります。このフィールドを設定することは

パラメーター	説明
	<p>お勧めしません。headerBufferBytes 値が小さすぎると Ingress コントローラーが破損する可能性があり、headerBufferBytes 値が大きすぎると、Ingress コントローラーが必要以上のメモリーを使用する可能性があるためです。</p> <ul style="list-style-type: none"> ● headerBufferMaxRewriteBytes は、HTTP ヘッダーの書き換えと Ingress コントローラー接続セッションの追加のために headerBufferBytes から予約するメモリーの量をバイト単位で指定します。headerBufferMaxRewriteBytes の最小値は 4096 です。受信 HTTP 要求には、headerBufferBytes は headerBufferMaxRewriteBytes よりも大きくなければなりません。設定されていない場合、デフォルト値は 8192 バイトになります。このフィールドを設定することはお勧めしません。headerBufferMaxRewriteBytes 値が小さすぎると Ingress コントローラーが破損する可能性があり、headerBufferMaxRewriteBytes 値が大きすぎると、Ingress コントローラーが必要以上のメモリーを使用する可能性があるためです。 ● threadCount は、HAProxy プロセスごとに作成するスレッドの数を指定します。より多くのスレッドを作成すると、使用されるシステムリソースを増やすことで、各 Ingress コントローラー Pod がより多くの接続を処理できるようになります。HAProxy は最大 64 のスレッドをサポートします。このフィールドが空の場合、Ingress コントローラーはデフォルト値の 4 スレッドを使用します。デフォルト値は、将来のリリースで変更される可能性があります。このフィールドを設定することはお勧めしません。HAProxy スレッドの数を増やすと、Ingress コントローラー Pod が負荷時に CPU 時間をより多く使用できるようになり、他の Pod が実行に必要な CPU リソースを受け取れないようになるためです。スレッドの数を減らすと、Ingress コントローラーのパフォーマンスが低下する可能性があります。 ● clientTimeout は、クライアント応答の待機中に接続が開かれる期間を指定します。未設定の場合、デフォルトのタイムアウトは 30s です。 ● serverFinTimeout は、接続を閉じるクライアントへの応答を待つ間、接続が開かれる期間を指定します。未設定の場合、デフォルトのタイムアウトは 1s です。 ● serverTimeout は、サーバーの応答を待機している間に接続が開かれる期間を指定します。未設定の場合、デフォルトのタイムアウトは 30s です。 ● clientFinTimeout は、クライアントの応答が接続を閉じるのを待機している間に接続が開かれる期間を指定します。未設定の場合、デフォルトのタイムアウトは 1s です。 ● tlsInspectDelay は、一致するルートを見つけるためにルーターがデータを保持する期間を指定します。この値の設定が短すぎると、より一致する証明書を使用している場合でも、ルーターがエッジ終端、再暗号化された、またはパススルーのルートのデフォルトの証明書にフォールバックする可能性があります。未設定の場合、デフォルトの検査遅延は 5s です。 ● tunnelTimeout は、トンネルがアイドル状態の間、WebSocket などのトンネル接続期間を開いた期間を指定します。未設定の場合、デフォルトのタイムアウトは 1h です。

パラメーター	説明
logEmptyRequests	<p>logEmptyRequests は、リクエストを受け取らず、ログに記録されない接続を指定します。これらの空の要求は、ロードバランサーヘルスプローブまたは Web ブラウザーの投機的接続 (事前接続) から送信され、これらの要求をログに記録することは望ましくない場合があります。ただし、これらの要求はネットワークエラーによって引き起こされる可能性があります。この場合は、空の要求をログに記録すると、エラーの診断に役立ちます。これらの要求はポートスキャンによって引き起こされ、空の要求をログに記録すると、侵入の試行が検出されなくなります。このフィールドに使用できる値は Log および Ignore です。デフォルト値は Log です。</p> <p>LoggingPolicy タイプは、以下のいずれかの値を受け入れます。</p> <ul style="list-style-type: none"> ● ログ: この値を Log に設定すると、イベントがログに記録される必要があることを示します。 ● Ignore: この値を Ignore に設定すると、HAproxy 設定の dontlognull オプションを設定します。
HTTPEmptyRequestsPolicy	<p>HTTPEmptyRequestsPolicy は、リクエストを受け取る前に接続がタイムアウトした場合に HTTP 接続を処理する方法を記述します。このフィールドに使用できる値は Respond および Ignore です。デフォルト値は Respond です。</p> <p>HTTPEmptyRequestsPolicy タイプは、以下のいずれかの値を受け入れます。</p> <ul style="list-style-type: none"> ● 応答: フィールドが Respond に設定されている場合、Ingress コントローラーは HTTP 400 または 408 応答を送信する場合、アクセスログが有効な場合に接続をログに記録し、適切なメトリクスで接続をカウントします。 ● ignore: このオプションを Ignore に設定すると HAproxy 設定に http-ignore-probes パラメーターが追加されます。フィールドが Ignore に設定されている場合、Ingress コントローラーは応答を送信せずに接続を閉じると、接続をログに記録するか、メトリクスを増分します。 <p>これらの接続は、ロードバランサーのヘルスプローブまたは Web ブラウザーの投機的接続 (事前接続) から取得され、無視しても問題はありません。ただし、これらの要求はネットワークエラーによって引き起こされる可能性があります。そのため、このフィールドを Ignore に設定すると問題の検出と診断が妨げられる可能性があります。これらの要求はポートスキャンによって引き起こされ、空の要求をログに記録すると、侵入の試行が検出されなくなります。</p>



注記

すべてのパラメーターはオプションです。

6.3.1. Ingress コントローラーの TLS セキュリティープロファイル

TLS セキュリティープロファイルは、サーバーに接続する際に接続クライアントが使用できる暗号を規制する方法をサーバーに提供します。

6.3.1.1. TLS セキュリティープロファイルについて

TLS (Transport Layer Security) セキュリティープロファイルを使用して、さまざまな OpenShift Container Platform コンポーネントに必要な TLS 暗号を定義できます。OpenShift Container Platform の TLS セキュリティープロファイルは、[Mozilla が推奨する設定](#) に基づいています。

コンポーネントごとに、以下の TLS セキュリティープロファイルのいずれかを指定できます。

表6.1 TLS セキュリティープロファイル

プロファイル	説明
Old	<p>このプロファイルは、レガシークライアントまたはライブラリーでの使用を目的としています。このプロファイルは、Old 後方互換性 の推奨設定に基づいています。</p> <p>Old プロファイルには、最小 TLS バージョン 1.0 が必要です。</p> <div style="display: flex; align-items: center;">  <div> <p>注記</p> <p>Ingress コントローラーの場合、TLS の最小バージョンは 1.0 から 1.1 に変換されます。</p> </div> </div>
Intermediate	<p>このプロファイルは、大多数のクライアントに推奨される設定です。これは、Ingress コントローラー、kubelet、およびコントロールプレーンのデフォルトの TLS セキュリティープロファイルです。このプロファイルは、Intermediate 互換性 の推奨設定に基づいています。</p> <p>Intermediate プロファイルには、最小 TLS バージョン 1.2 が必要です。</p>
Modern	<p>このプロファイルは、後方互換性を必要としない Modern のクライアントでの使用を目的としています。このプロファイルは、Modern 互換性 の推奨設定に基づいています。</p> <p>Modern プロファイルには、最小 TLS バージョン 1.3 が必要です。</p>
カスタム	<p>このプロファイルを使用すると、使用する TLS バージョンと暗号を定義できます。</p> <div style="background-color: #fff9c4; padding: 10px; border: 1px solid #ccc;"> <div style="display: flex; align-items: center;">  <div> <p>警告</p> <p>無効な設定により問題が発生する可能性があるため、Custom プロファイルを使用するには注意してください。</p> </div> </div> </div>



注記

事前定義されたプロファイルタイプのいずれかを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース X.Y.Z にデプロイされた Intermediate プロファイルを使用する仕様がある場合、リリース X.Y.Z+1 へのアップグレードにより、新規のプロファイル設定が適用され、ロールアウトが生じる可能性があります。

6.3.1.2. Ingress コントローラーの TLS セキュリティープロファイルの設定

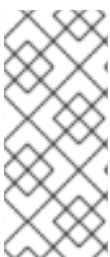
Ingress コントローラーの TLS セキュリティープロファイルを設定するには、**IngressController** カスタムリソース (CR) を編集して、事前定義済みまたはカスタムの TLS セキュリティープロファイルを指定します。TLS セキュリティープロファイルが設定されていない場合、デフォルト値は API サーバーに設定された TLS セキュリティープロファイルに基づいています。

Old TLS のセキュリティプロファイルを設定するサンプル IngressController CR

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

TLS セキュリティープロファイルは、Ingress コントローラーの TLS 接続の最小 TLS バージョンと TLS 暗号を定義します。

設定された TLS セキュリティープロファイルの暗号と最小 TLS バージョンは、**Status.Tls Profile** 配下の **IngressController** カスタムリソース (CR) と **Spec.Tls Security Profile** 配下の設定された TLS セキュリティープロファイルで確認できます。**Custom** TLS セキュリティープロファイルの場合、特定の暗号と最小 TLS バージョンは両方のパラメーターの下に一覧表示されます。



注記

HAProxy Ingress Controller イメージは、TLS1.3 と **Modern** プロファイルをサポートしています。

また、Ingress Operator は TLS 1.0 の **Old** または **Custom** プロファイルを 1.1 に変換します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **openshift-ingress-operator** プロジェクトの **IngressController** CR を編集して、TLS セキュリティープロファイルを設定します。

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

2. **spec.tlsSecurityProfile** フィールドを追加します。

Custom プロファイルのサンプル IngressController CR

```

apiVersion: operator.openshift.io/v1
kind: IngressController
...
spec:
  tlsSecurityProfile:
    type: Custom ❶
    custom: ❷
      ciphers: ❸
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
    minTLSVersion: VersionTLS11
  ...

```

- ❶ TLS セキュリティプロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。
- ❷ 選択したタイプに適切なフィールドを指定します。
 - **old:** {}
 - **intermediate:** {}
 - **custom:**
- ❸ **custom** タイプには、TLS 暗号の一覧と最小許容 TLS バージョンを指定します。

3. 変更を適用するためにファイルを保存します。

検証

- **IngressController** CR にプロファイルが設定されていることを確認します。

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

出力例

```

Name:      default
Namespace: openshift-ingress-operator
Labels:    <none>
Annotations: <none>
API Version: operator.openshift.io/v1
Kind:      IngressController
...
Spec:
...
Tls Security Profile:
  Custom:
    Ciphers:
      ECDHE-ECDSA-CHACHA20-POLY1305
      ECDHE-RSA-CHACHA20-POLY1305

```

```

ECDHE-RSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
Min TLS Version: VersionTLS11
Type:          Custom
...

```

6.3.1.3. 相互 TLS 認証の設定

spec.clientTLS 値を設定して、相互 TLS (mTLS) 認証を有効にするように Ingress コントローラーを設定できます。**clientTLS** 値は、クライアント証明書を検証するように Ingress コントローラーを設定します。この設定には、ConfigMap の参照である **clientCA** 値の設定が含まれます。ConfigMap には、クライアントの証明書を検証するために使用される PEM でエンコードされた CA 証明書バンドルが含まれます。必要に応じて、証明書サブジェクトフィルターの一覧を設定できます。

clientCA の値が X509v3 証明書失効リスト (CRL) の分散ポイントを指定する場合、Ingress Operator は CRL をダウンロードし、Ingress コントローラーがこれを認識するように設定します。有効な証明書を提供しない要求は拒否されます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **openshift-config** namespace にある ConfigMap を作成します。

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



注記

ConfigMap データキーは **ca-bundle.pem** で、data の値は PEM 形式の CA 証明書である必要があります。

2. **openshift-ingress-operator** プロジェクトで **IngressController** リソースを編集します。

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

3. **spec.clientTLS** フィールドおよびサブフィールドを追加して相互 TLS を設定します。

フィルターリングパターンを指定する clientTLS プロファイルのサンプル IngressController CR

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  clientTLS:
    clientCertificatePolicy: Required
    clientCA:

```

```
name: router-ca-certs-default
allowedSubjectPatterns:
- "^/CN=example.com/ST=NC/C=US/O=Security/OU=OpenShift$"
```

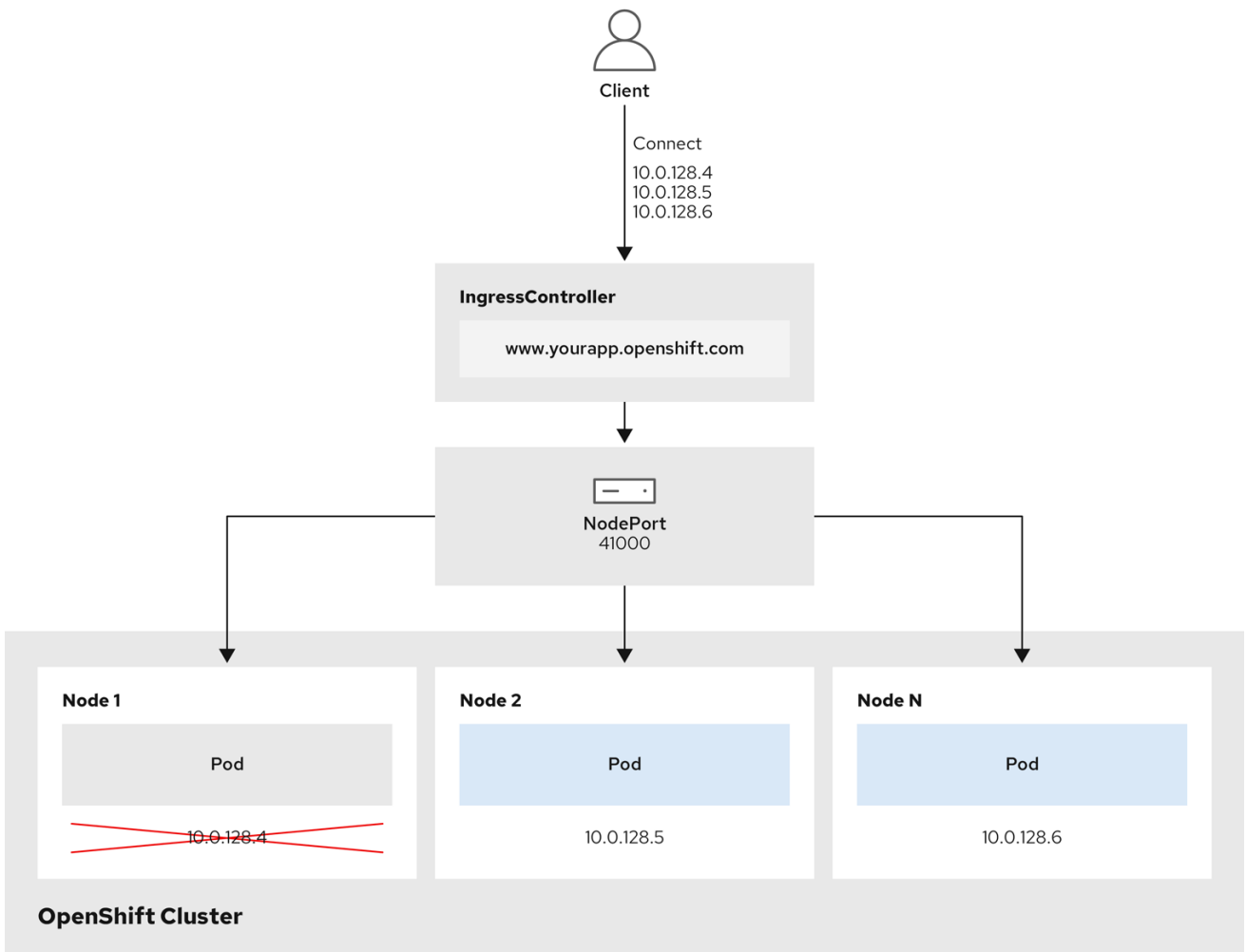
6.3.2. Ingress コントローラーエンドポイントの公開ストラテジー

NodePortService エンドポイントの公開ストラテジー

NodePortService エンドポイントの公開ストラテジーは、Kubernetes NodePort サービスを使用して Ingress コントローラーを公開します。

この設定では、Ingress コントローラーのデプロイメントはコンテナのネットワークを使用します。**NodePortService** はデプロイメントを公開するために作成されます。特定のノードポートは OpenShift Container Platform によって動的に割り当てられますが、静的ポートの割り当てをサポートするために、管理される **NodePortService** のノードポートフィールドへの変更が保持されます。

図6.1 NodePortService の図



202_OpenShift_0222

前述の図では、OpenShift Container Platform Ingress NodePort エンドポイントの公開戦略に関する以下のような概念を示しています。

- クラスタで利用可能なノードにはすべて、外部からアクセス可能な独自の IP アドレスが割り当てられています。クラスタ内で動作するサービスは、全ノードに固有の NodePort にバインドされます。

- たとえば、クライアントが図中の IP アドレス **10.0.128.4** に接続してダウンしているノードに接続した場合に、ノードポートは、サービスを実行中で利用可能なノードにクライアントを直接接続します。このシナリオでは、ロードバランシングは必要ありません。イメージが示すように、**10.0.128.4** アドレスがダウンしており、代わりに別の IP アドレスを使用する必要があります。



注記

Ingress Operator は、サービスの `.spec.ports[].nodePort` フィールドへの更新を無視します。

デフォルトで、ポートは自動的に割り当てられ、各種の統合用のポート割り当てにアクセスできます。ただし、既存のインフラストラクチャーと統合するために静的ポートの割り当てが必要になることがあります。これは動的ポートに対応して簡単に再設定できない場合があります。静的ノードポートとの統合を実行するには、管理対象のサービスリソースを直接更新できます。

詳細は、[NodePort についての Kubernetes サービスについてのドキュメント](#) を参照してください。

HostNetwork エンドポイントの公開ストラテジー

HostNetwork エンドポイントの公開ストラテジーは、Ingress コントローラーがデプロイされるノードポートで Ingress コントローラーを公開します。

HostNetwork エンドポイント公開ストラテジーを持つ Ingress コントローラーには、ノードごとに単一の Pod レプリカのみを設定できます。n のレプリカを使用する場合、それらのレプリカをスケジュールできる n 以上のノードを使用する必要があります。各 Pod はスケジュールされるノードホストでポート **80** および **443** を要求するので、同じノードで別の Pod がそれらのポートを使用している場合、レプリカをノードにスケジュールすることはできません。

6.4. デフォルト INGRESS コントローラーの表示

Ingress Operator は、OpenShift Container Platform の中核となる機能であり、追加の設定なしに有効にできます。

すべての新規 OpenShift Container Platform インストールには、**ingresscontroller** の名前付きのデフォルトがあります。これは、追加の Ingress コントローラーで補足できます。デフォルトの **ingresscontroller** が削除される場合、Ingress Operator は 1 分以内にこれを自動的に再作成します。

手順

- デフォルト Ingress コントローラーを表示します。

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

6.5. INGRESS OPERATOR ステータスの表示

Ingress Operator のステータスを表示し、検査することができます。

手順

- Ingress Operator ステータスを表示します。

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

■

6.6. INGRESS コントローラーログの表示

Ingress コントローラーログを表示できます。

手順

- Ingress コントローラーログを表示します。

```
$ oc logs --namespace=openshift-ingress-operator deployments/ingress-operator
```

6.7. INGRESS コントローラーステータスの表示

特定の Ingress コントローラーのステータスを表示できます。

手順

- Ingress コントローラーのステータスを表示します。

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

6.8. INGRESS コントローラーの設定

6.8.1. カスタムデフォルト証明書の設定

管理者として、Secret リソースを作成し、**IngressController** カスタムリソース (CR) を編集して Ingress コントローラーがカスタム証明書を使用するように設定できます。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアがなければなりません。ここで、証明書は信頼される認証局またはカスタム PKI で設定されたプライベートの信頼される認証局で署名されます。
- 証明書が以下の要件を満たしている必要があります。
 - 証明書が Ingress ドメインに対して有効化されている必要があります。
 - 証明書は拡張を使用して、**subjectAltName** 拡張を使用して、***.apps.ocp4.example.com** などのワイルドカードドメインを指定します。
- **IngressController** CR がなければなりません。デフォルトの CR を使用できます。

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

出力例

```
NAME    AGE
default 10m
```

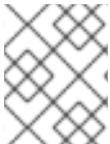


注記

Intermediate 証明書がある場合、それらはカスタムデフォルト証明書が含まれるシークレットの **tls.crt** ファイルに組み込まれる必要があります。証明書を指定する際の順序は重要になります。サーバー証明書の後に Intermediate 証明書を一覧表示します。

手順

以下では、カスタム証明書とキーのペアが、現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提とします。**tls.crt** および **tls.key** を実際のパス名に置き換えます。さらに、Secret リソースを作成し、これを IngressController CR で参照する際に、**custom-certs-default** を別の名前に置き換えます。



注記

このアクションにより、Ingress コントローラーはデプロイメントストラテジーを使用して再デプロイされます。

1. **tls.crt** および **tls.key** ファイルを使用して、カスタム証明書を含む Secret リソースを **openshift-ingress** namespace に作成します。

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

2. IngressController CR を、新規証明書シークレットを参照するように更新します。

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

3. 更新が正常に行われていることを確認します。

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

ここでは、以下ようになります。

<domain>

クラスターのベースドメイン名を指定します。

出力例

```
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
```

ヒント

または、以下の YAML を適用してカスタムのデフォルト証明書を設定できます。

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

証明書シークレットの名前は、CR を更新するために使用された値に一致する必要があります。

IngressController CR が変更された後に、Ingress Operator はカスタム証明書を使用できるように Ingress コントローラーのデプロイメントを更新します。

6.8.2. カスタムデフォルト証明書の削除

管理者は、使用する Ingress Controller を設定したカスタム証明書を削除できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Ingress Controller のカスタムデフォルト証明書を設定している。

手順

- カスタム証明書を削除し、OpenShift Container Platform に同梱されている証明書を復元するには、以下のコマンドを入力します。

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

クラスターが新しい証明書設定を調整している間、遅延が発生する可能性があります。

検証

- 元のクラスター証明書が復元されたことを確認するには、次のコマンドを入力します。

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

ここでは、以下ようになります。

<domain>

クラスターのベースドメイン名を指定します。

出力例

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

6.8.3. Ingress コントローラーのスケーリング

Ingress コントローラーは、スループットを増大させるための要件を含む、ルーティングのパフォーマンスや可用性に関する各種要件に対応するために手動でスケーリングできます。**oc** コマンドは、**IngressController** リソースのスケーリングに使用されます。以下の手順では、デフォルトの **IngressController** をスケールアップする例を示します。



注記

スケーリングは、必要な数のレプリカを作成するのに時間がかかるため、すぐに実行できるアクションではありません。

手順

1. デフォルト **IngressController** の現在の利用可能なレプリカ数を表示します。

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

出力例

```
2
```

2. **oc patch** コマンドを使用して、デフォルトの **IngressController** を必要なレプリカ数にスケーリングします。以下の例では、デフォルトの **IngressController** を3つのレプリカにスケーリングしています。

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

出力例

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

3. デフォルトの **IngressController** が指定したレプリカ数にスケーリングされていることを確認します。

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

出力例

```
3
```

ヒント

または、以下の YAML を適用して Ingress コントローラーを 3 つのレプリカにスケールすることもできます。

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

- ① 異なる数のレプリカが必要な場合は **replicas** 値を変更します。

6.8.4. Ingress アクセスロギングの設定

アクセスログを有効にするように Ingress コントローラーを設定できます。大量のトラフィックを受信しないクラスターがある場合、サイドカーにログインできます。クラスターのトラフィックが多い場合、ロギングスタックの容量を超えないようにしたり、OpenShift Container Platform 外のロギングインフラストラクチャーと統合したりするために、ログをカスタム syslog エンドポイントに転送することができます。アクセスログの形式を指定することもできます。

コンテナロギングは、既存の Syslog ロギングインフラストラクチャーがない場合や、Ingress コントローラーで問題を診断する際に短期間使用する場合に、低トラフィックのクラスターのアクセスログを有効にするのに役立ちます。

アクセスログが OpenShift Logging スタックの容量を超える可能性があるトラフィックの多いクラスターや、ロギングソリューションが既存の Syslog ロギングインフラストラクチャーと統合する必要がある環境では、syslog が必要です。Syslog のユースケースは重複する可能性があります。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

サイドカーへの Ingress アクセスロギングを設定します。

- Ingress アクセスロギングを設定するには、**spec.logging.access.destination** を使用して宛先を指定する必要があります。サイドカーコンテナへのロギングを指定するには、**Container spec.logging.access.destination.type** を指定する必要があります。以下の例は、コンテナ **Container** の宛先に対してログ記録する Ingress コントローラー定義です。

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

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

- Ingress コントローラーをサイドカーに対してログを記録するように設定すると、Operator は Ingress コントローラー Pod 内に **logs** という名前のコンテナを作成します。

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

出力例

```
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 エンドポイントへの Ingress アクセスロギングを設定します。

- Ingress アクセスロギングを設定するには、**spec.logging.access.destination** を使用して宛先を指定する必要があります。Syslog エンドポイント宛先へのロギングを指定するには、**spec.logging.access.destination.type** に **Syslog** を指定する必要があります。宛先タイプが **Syslog** の場合、**spec.logging.access.destination.syslog.endpoint** を使用して宛先エンドポイントも指定する必要があります。また、**spec.logging.access.destination.syslog.facility** を使用してファシリティを指定できます。以下の例は、**Syslog** 宛先に対してログを記録する Ingress コントローラーの定義です。

```
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
```



注記

syslog 宛先ポートは UDP である必要があります。

特定のログ形式で Ingress アクセスロギングを設定します。

- **spec.logging.access.httpLogFormat** を指定して、ログ形式をカスタマイズできます。以下の例は、IP アドレスが 1.2.3.4 およびポート 10514 の **syslog** エンドポイントに対してログを記録する Ingress コントローラーの定義です。

```
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 アクセスロギングを無効にします。

- Ingress アクセスロギングを無効にするには、**spec.logging** または **spec.logging.access** を空のままにします。

```

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

```

6.8.5. Ingress コントローラー スレッド数の設定

クラスター管理者は、スレッド数を設定して、クラスターが処理できる受信接続の量を増やすことができます。既存の Ingress コントローラーにパッチを適用して、スレッドの数を増やすことができます。

前提条件

- 以下では、Ingress コントローラーがすでに作成されていることを前提とします。

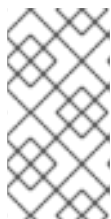
手順

- Ingress コントローラーを更新して、スレッド数を増やします。

```

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

```



注記

大量のリソースを実行できるノードがある場合、**spec.nodePlacement.nodeSelector** を、意図されているノードの容量に一致するラベルで設定し、**spec.tuningOptions.threadCount** を随時高い値に設定します。

6.8.6. Ingress コントローラーのシャード化

トラフィックがクラスターに送信される主要なメカニズムとして、Ingress コントローラーまたはルーターへの要求が大きくなる可能性があります。クラスター管理者は、以下を実行するためにルートをシャード化できます。

- Ingress コントローラーまたはルーターを複数のルートに分散し、変更に対する応答を加速します。
- 特定のルートを他のルートとは異なる信頼性の保証を持つように割り当てます。
- 特定の Ingress コントローラーに異なるポリシーを定義することを許可します。
- 特定のルートのみが追加機能を使用することを許可します。
- たとえば、異なるアドレスで異なるルートを公開し、内部ユーザーおよび外部ユーザーが異なるルートを認識できるようにします。

Ingress コントローラーは、ルートラベルまたは namespace ラベルのいずれかをシャード化の方法として使用できます。

6.8.6.1. ルートラベルを使用した Ingress コントローラーのシャード化の設定

ルートラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーがルートセレクターによって選択される任意 namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

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

- Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

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

Ingress コントローラーは、**type: sharded** というラベルのある namespace のルートを選択します。

6.8.6.2. namespace ラベルを使用した Ingress コントローラーのシャード化の設定

namespace ラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーが namespace セレクターによって選択される任意の namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。



警告

Keepalived Ingress VIP をデプロイする場合は、**endpoint Publishing Strategy** パラメーターに **Host Network** の値が割り当てられた、デフォルト以外の Ingress Controller をデプロイしないでください。デプロイしてしまうと、問題が発生する可能性があります。**endpoint Publishing Strategy** に **Host Network** ではなく、**Node Port** という値を使用してください。

手順

- router-internal.yaml** ファイルを編集します。

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

出力例

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

```

status: {}
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""

```

- Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

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

Ingress コントローラーは、**type: sharded** というラベルのある namespace セレクターによって選択される namespace のルートを選択します。

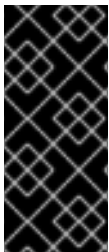
6.8.7. 内部ロードバランサーを使用するように Ingress コントローラーを設定する

クラウドプラットフォームで Ingress コントローラーを作成する場合、Ingress コントローラーはデフォルトでパブリッククラウドロードバランサーによって公開されます。管理者は、内部クラウドロードバランサーを使用する Ingress コントローラーを作成できます。



警告

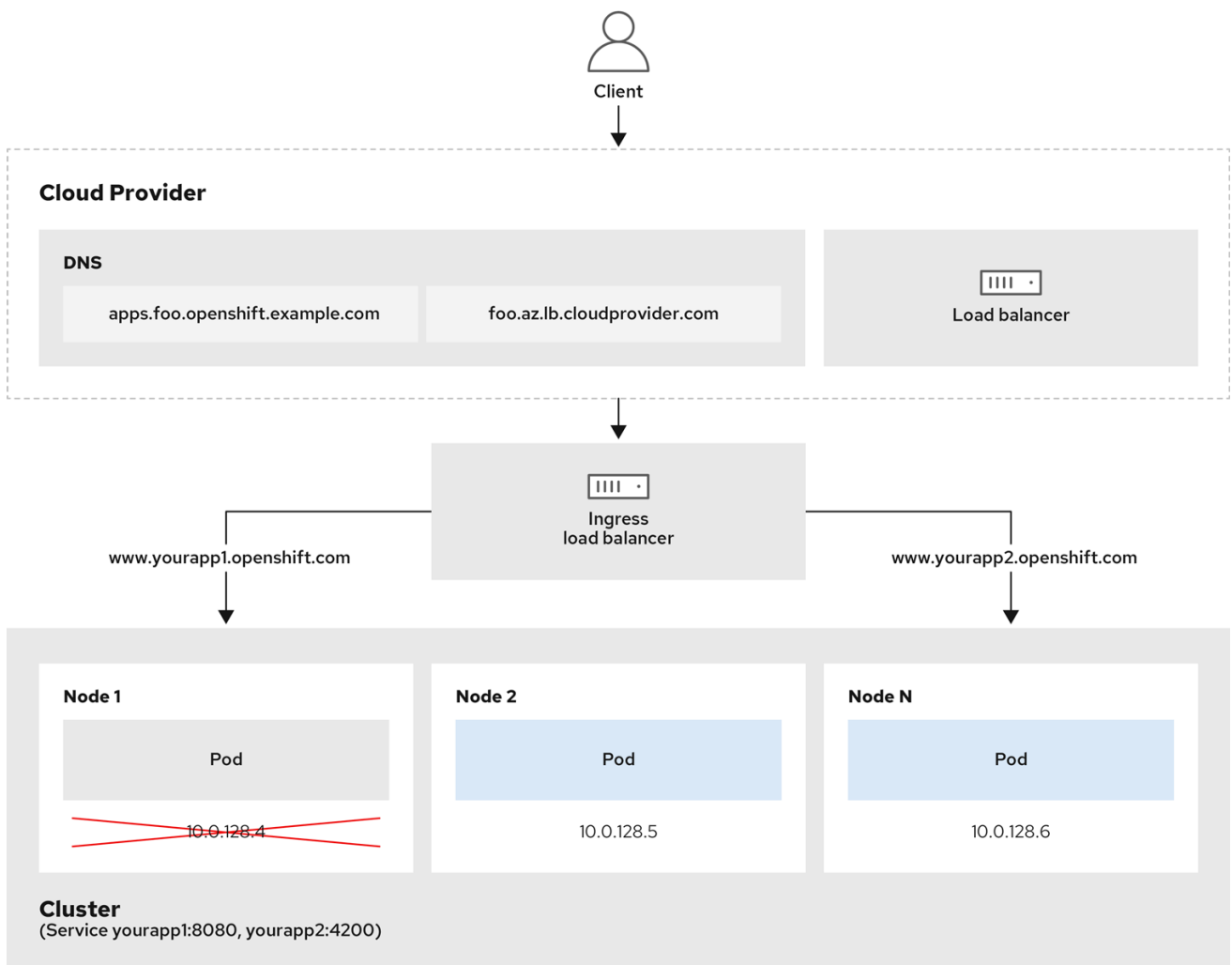
クラウドプロバイダーが Microsoft Azure の場合、ノードを参照するパブリックロードバランサーが少なくとも1つ必要です。これがない場合、すべてのノードがインターネットへの egress 接続を失います。



重要

IngressController オブジェクトの **scope** を変更する必要がある場合、**IngressController** オブジェクトを削除してから、これを再作成する必要があります。カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更することはできません。

図6.2 ロードバランサーの図



202_OpenShift_0222

前述の図では、OpenShift Container Platform Ingress LoadBalancerService エンドポイントの公開戦略に関する以下のような概念を示しています。

- 負荷は、外部からクラウドプロバイダーのロードバランサーを使用するか、内部から OpenShift Ingress Controller Load Balancer を使用して、分散できます。
- ロードバランサーのシングル IP アドレスと、図にあるクラスターのように、8080 や 4200 といった馴染みのあるポートを使用することができます。
- 外部のロードバランサーからのトラフィックは、ダウンしたノードのインスタンスで記載されているように、Pod の方向に進められ、ロードバランサーが管理します。実装の詳細については、[Kubernetes サービスドキュメント](#) を参照してください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の例のように、**<name>-ingress-controller.yaml** という名前のファイルに **IngressController** カスタムリソース (CR) を作成します。


```

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

```

- ❶ <name> を **IngressController** オブジェクトの名前に置き換えます。
- ❷ コントローラーによって公開されるアプリケーションの **ドメイン** を指定します。
- ❸ 内部ロードバランサーを使用するために **Internal** の値を指定します。

2. 以下のコマンドを実行して、直前の手順で定義された Ingress コントローラーを作成します。

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

- ❶ <name> を **IngressController** オブジェクトの名前に置き換えます。

3. オプション: 以下のコマンドを実行して Ingress コントローラーが作成されていることを確認します。

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

6.8.8. GCP での Ingress コントローラーのグローバルアクセスの設定

内部ロードバランサーで GCP で作成された Ingress コントローラーは、サービスの内部 IP アドレスを生成します。クラスター管理者は、グローバルアクセスオプションを指定できます。これにより、同じ VPC ネットワーク内の任意のリージョンでクラスターを有効にし、ロードバランサーとしてコンピューターリージョンを有効にして、クラスターで実行されるワークロードに到達できるようにできます。

詳細情報は、GCP ドキュメントの [グローバルアクセス](#) について参照してください。

前提条件

- OpenShift Container Platform クラスターを GCP インフラストラクチャーにデプロイしている。
- 内部ロードバランサーを使用するように Ingress コントローラーを設定している。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. グローバルアクセスを許可するように Ingress コントローラーリソースを設定します。



注記

Ingress コントローラーを作成し、グローバルアクセスのオプションを指定することもできます。

- a. Ingress コントローラーリソースを設定します。

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

- b. YAML ファイルを編集します。

サンプル clientAccess 設定を Global に設定します。

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

- 1** **gcp.clientAccess** を **Global** に設定します。

- c. 変更を適用するためにファイルを保存します。

2. 以下のコマンドを実行して、サービスがグローバルアクセスを許可することを確認します。

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

この出力では、グローバルアクセスがアノテーション **networking.gke.io/internal-load-balancer-allow-global-access** で GCP について有効にされていることを示しています。

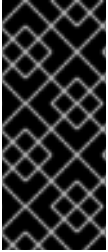
6.8.9. クラスターを内部に配置するようにのデフォルト Ingress コントローラーを設定する

削除や再作成を実行して、クラスターを内部に配置するように **default** Ingress コントローラーを設定できます。



警告

クラウドプロバイダーが Microsoft Azure の場合、ノードを参照するパブリックロードバランサーが少なくとも1つ必要です。これがない場合、すべてのノードがインターネットへの egress 接続を失います。



重要

IngressController オブジェクトの **スコープ** を変更する必要がある場合、**IngressController** オブジェクトを削除してから、これを再作成する必要があります。カスタムリソース (CR) の作成後に **.spec.endpointPublishingStrategy.loadBalancer.scope** パラメーターを変更することはできません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 削除や再作成を実行して、クラスターを内部に配置するように **default** Ingress コントローラーを設定します。

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

6.8.10. ルートの受付ポリシーの設定

管理者およびアプリケーション開発者は、同じドメイン名を持つ複数の namespace でアプリケーションを実行できます。これは、複数のチームが同じホスト名で公開されるマイクロサービスを開発する組織を対象としています。



警告

複数の namespace での要求の許可は、namespace 間の信頼のあるクラスターに対してのみ有効にする必要があります。有効にしないと、悪意のあるユーザーがホスト名を乗っ取る可能性があります。このため、デフォルトの受付ポリシーは複数の namespace 間でのホスト名の要求を許可しません。

前提条件

- クラスター管理者の権限。

手順

- 以下のコマンドを使用して、**ingresscontroller** リソース変数の **.spec.routeAdmission** フィールドを編集します。

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

イメージコントローラー設定例

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

ヒント

または、以下の YAML を適用してルートの受付ポリシーを設定できます。

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

6.8.11. ワイルドカードルートの使用

HAProxy Ingress コントローラーにはワイルドカードルートのサポートがあります。Ingress Operator は **wildcardPolicy** を使用して、Ingress コントローラーの **ROUTER_ALLOW_WILDCARD_ROUTES** 環境変数を設定します。

Ingress コントローラーのデフォルトの動作では、ワイルドカードポリシーの **None** (既存の **IngressController** リソースとの後方互換性がある) を持つルートを許可します。

手順

- ワイルドカードポリシーを設定します。
 - 以下のコマンドを使用して **IngressController** リソースを編集します。

```
$ oc edit IngressController
```

- spec** の下で、**wildcardPolicy** フィールドを **WildcardsDisallowed** または **WildcardsAllowed** に設定します。

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

6.8.12. X-Forwarded ヘッダーの使用

Forwarded および **X-Forwarded-For** を含む HTTP ヘッダーの処理方法についてのポリシーを指定するように HAProxy Ingress コントローラーを設定します。Ingress Operator は **HTTPHeaders** フィールドを使用して、Ingress コントローラーの **ROUTER_SET_FORWARDED_HEADERS** 環境変数を設定します。

手順

1. Ingress コントローラー用に **HTTPHeaders** フィールドを設定します。

- a. 以下のコマンドを使用して **IngressController** リソースを編集します。

```
$ oc edit IngressController
```

- b. **spec** の下で、**HTTPHeaders** ポリシーフィールドを **Append**、**Replace**、**IfNone**、または **Never** に設定します。

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

使用例

クラスター管理者として、以下を実行できます。

- Ingress コントローラーに転送する前に、**X-Forwarded-For** ヘッダーを各リクエストに挿入する外部プロキシを設定します。
ヘッダーを変更せずに渡すように Ingress コントローラーを設定するには、**never** ポリシーを指定します。これにより、Ingress コントローラーはヘッダーを設定しなくなり、アプリケーションは外部プロキシが提供するヘッダーのみを受信します。
- 外部プロキシが外部クラスター要求を設定する **X-Forwarded-For** ヘッダーを変更せずに渡すように Ingress コントローラーを設定します。
外部プロキシを通過しない内部クラスター要求に **X-Forwarded-For** ヘッダーを設定するように Ingress コントローラーを設定するには、**if-none** ポリシーを指定します。外部プロキシ経由で HTTP 要求にヘッダーがすでに設定されている場合、Ingress コントローラーはこれを保持します。要求がプロキシを通過していないためにヘッダーがない場合、Ingress コントローラーはヘッダーを追加します。

アプリケーション開発者として、以下を実行できます。

- **X-Forwarded-For** ヘッダーを挿入するアプリケーション固有の外部プロキシを設定します。他の Route のポリシーに影響を与えずに、アプリケーションの Route 用にヘッダーを変更せずに渡すように Ingress コントローラーを設定するには、アプリケーションの Route にアノテーション **haproxy.router.openshift.io/set-forwarded-headers: if-none** または **haproxy.router.openshift.io/set-forwarded-headers: never** を追加します。



注記

Ingress コントローラーのグローバルに設定された値とは別に、**haproxy.router.openshift.io/set-forwarded-headers** アノテーションをルートごとに設定できます。

6.8.13. HTTP/2 Ingress 接続の有効化

HAProxy で透過的なエンドツーエンド HTTP/2 接続を有効にすることができます。これにより、アプリケーションの所有者は、単一接続、ヘッダー圧縮、バイナリストリームなど、HTTP/2 プロトコル機能を使用できます。

個別の Ingress コントローラーまたはクラスター全体について、HTTP/2 接続を有効にすることができます。

クライアントから HAProxy への接続について HTTP/2 の使用を有効にするために、ルートはカスタム証明書を指定する必要があります。デフォルトの証明書を使用するルートは HTTP/2 を使用することができません。この制限は、クライアントが同じ証明書を使用する複数の異なるルートに接続を再使用するなどの、接続の結合 (coalescing) の問題を回避するために必要です。

HAProxy からアプリケーション Pod への接続は、re-encrypt ルートのみに HTTP/2 を使用でき、edge termination ルートまたは非セキュアなルートには使用しません。この制限は、HAProxy が TLS 拡張である Application-Level Protocol Negotiation (ALPN) を使用してバックエンドで HTTP/2 の使用をネゴシエートするためにあります。そのため、エンドツーエンドの HTTP/2 はパススルーおよび re-encrypt 使用できますが、非セキュアなルートまたは edge termination ルートでは使用できません。



警告

再暗号化ルートで WebSocket を使用し、Ingress Controller で HTTP/2 を有効にするには、HTTP/2 を介した WebSocket のサポートが必要です。HTTP/2 上の WebSockets は HAProxy 2.4 の機能であり、現時点では OpenShift Container Platform ではサポートされていません。

重要

パススルー以外のルートの場合、Ingress コントローラーはクライアントからの接続とは独立してアプリケーションへの接続をネゴシエートします。つまり、クライアントが Ingress コントローラーに接続して HTTP/1.1 をネゴシエートし、Ingress コントローラーは次にアプリケーションに接続して HTTP/2 をネゴシエートし、アプリケーションへの HTTP/2 接続を使用してクライアント HTTP/1.1 接続からの要求の転送を実行できます。Ingress コントローラーは WebSocket を HTTP/2 に転送できず、その HTTP/2 接続を WebSocket に対してアップグレードできないため、クライアントが後に HTTP/1.1 から WebSocket プロトコルに接続をアップグレードしようとする問題が発生します。そのため、WebSocket 接続を受け入れることが意図されたアプリケーションがある場合、これは HTTP/2 プロトコルのネゴシエートを許可できないようにする必要があります。そうしないと、クライアントは WebSocket プロトコルへのアップグレードに失敗します。

手順

単一 Ingress コントローラーで HTTP/2 を有効にします。

- Ingress コントローラーで HTTP/2 を有効にするには、**oc annotate** コマンドを入力します。

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

<ingresscontroller_name> をアノテーションを付ける Ingress コントローラーの名前に置き換えます。

クラスター全体で HTTP/2 を有効にします。

- クラスター全体で HTTP/2 を有効にするには、**oc annotate** コマンドを入力します。

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

ヒント

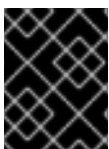
または、以下の YAML を適用してアノテーションを追加できます。

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

6.8.14. Ingress コントローラーの PROXY プロトコルの設定

クラスター管理者は、Ingress コントローラーが **HostNetwork** または **NodePortService** エンドポイントの公開ストラテジータイプのいずれかを使用する際に **PROXY プロトコル** を設定できます。PROXY プロトコルにより、ロードバランサーは Ingress コントローラーが受信する接続の元のクライアントアドレスを保持することができます。元のクライアントアドレスは、HTTP ヘッダーのロギング、フィルターリング、および挿入を実行する場合に便利です。デフォルト設定では、Ingress コントローラーが受信する接続には、ロードバランサーに関連付けられるソースアドレスのみが含まれます。

この機能は、クラウドデプロイメントではサポートされていません。この制限は、OpenShift Container Platform がクラウドプラットフォームで実行される場合、IngressController はサービ出力ドバランサーを使用するように指定し、Ingress Operator はロードバランサーサービスを設定し、ソースアドレスを保持するプラットフォーム要件に基づいて PROXY プロトコルを有効にするためにあります。



重要

PROXY プロトコルまたは TCP を使用するには、OpenShift Container Platform と外部ロードバランサーの両方を設定する必要があります。



警告

PROXY プロトコルは、Keepalived Ingress VIP を使用するクラウド以外のプラットフォーム上のインストーラーによってプロビジョニングされたクラスターを使用するデフォルトの Ingress コントローラーではサポートされていません。

前提条件

- Ingress コントローラーを作成している。

手順

1. Ingress コントローラーリソースを編集します。

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

2. PROXY 設定を設定します。

- Ingress コントローラーが `hostNetwork` エンドポイント公開ストラテジータイプを使用する場合は、`spec.endpointPublishingStrategy.nodePort.protocol` サブフィールドを **PROXY** に設定します。

PROXY への `hostNetwork` の設定例

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

- Ingress コントローラーが `NodePortService` エンドポイント公開ストラテジータイプを使用する場合は、`spec.endpointPublishingStrategy.nodePort.protocol` サブフィールドを **PROXY** に設定します。

PROXY へのサンプル `nodePort` 設定

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

6.8.15. `appsDomain` オプションを使用した代替クラスタードメインの指定

クラスター管理者は、**`appsDomain`** フィールドを設定して、ユーザーが作成したルートのデフォルトのクラスタードメインの代わりに指定できます。**`appsDomain`** フィールドは、**`domain`** フィールドで指定されているデフォルトの代わりに使用する OpenShift Container Platform のオプションのドメインです。代替ドメインを指定する場合、これは新規ルートのデフォルトホストを判別できるようにする目的でデフォルトのクラスタードメインを上書きします。

たとえば、所属企業の DNS ドメインを、クラスター上で実行されるアプリケーションのルートおよび `ingress` のデフォルトドメインとして使用できます。

前提条件

- OpenShift Container Platform クラスターをデプロイしていること。
- **`oc`** コマンドラインインターフェイスをインストールしている。

手順

1. ユーザーが作成するルートに代替のデフォルトドメインを指定して **`appsDomain`** フィールドを設定します。

- a. Ingress **cluster** リソースを編集します。

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

- b. YAML ファイルを編集します。

test.example.com への apps Domain の設定例

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

- 1 デフォルトドメインを指定します。インストール後にデフォルトドメインを変更することはできません。
- 2 オプション: アプリケーションルートに使用する OpenShift Container Platform インフラストラクチャーのドメイン。デフォルトの接頭辞である **apps** の代わりに、**test** のような別の接頭辞を使用できます。

2. ルートを公開し、ルートドメインの変更を確認して、既存のルートに、**appsDomain** フィールドで指定したドメイン名が含まれていることを確認します。



注記

ルートを公開する前に **openshift-apiserver** がローリング更新を終了するのを待機します。

- a. ルートを公開します。

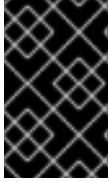
```
$ oc expose service hello-openshift
route.route.openshift.io/hello-openshift exposed
```

出力例:

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

6.8.16. HTTP ヘッダーケースの変換

HAProxy 2.2 では、デフォルトで HTTP ヘッダー名を小文字化します。たとえば、**Host: xyz.com** を **host: xyz.com** に変更します。レガシーアプリケーションが HTTP ヘッダー名の大文字を認識する場合、Ingress Controller の **spec.httpHeaders.headerNameCaseAdjustments** API フィールドを、修正されるまでレガシーアプリケーションに対応するソリューションに使用します。



重要

OpenShift Container Platform 4.9 には HAProxy 2.2 が含まれるため、アップグレードする前に **spec.httpHeaders.headerNameCaseAdjustments** を使用して必要な設定を追加するようにしてください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

クラスター管理者は、**oc patch** コマンドを入力するか、または Ingress コントローラー YAML ファイルの **HeaderNameCaseAdjustments** フィールドを設定して HTTP ヘッダーのケースを変換できます。

- **oc patch** コマンドを入力して、HTTP ヘッダーの大文字化を指定します。

1. **oc patch** コマンドを入力して、HTTP **host** ヘッダーを **Host** に変更します。

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

2. アプリケーションのルートにアノテーションを付けます。

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

次に、Ingress コントローラーは **host** 要求ヘッダーを指定どおりに調整します。

- Ingress コントローラーの YAML ファイルを設定し、**HeaderNameCaseAdjustments** フィールドを使用して調整を指定します。

1. 以下のサンプル Ingress コントローラー YAML は、適切にアノテーションが付けられたルートへの HTTP/1 要求について **host** ヘッダーを **Host** に調整します。

Ingress コントローラー YAML のサンプル

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

2. 以下のサンプルルートでは、**haproxy.router.openshift.io/h1-adjust-case** アノテーションを使用して HTTP 応答ヘッダー名のケース調整を有効にします。

ルート YAML のサンプル

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

```

annotations:
  haproxy.router.openshift.io/h1-adjust-case: true ❶
name: my-application
namespace: my-application
spec:
  to:
    kind: Service
    name: my-application

```

- ❶ **haproxy.router.openshift.io/h1-adjust-case** を true に設定します。

6.8.17. HAProxy エラーコードの応答ページのカスタマイズ

クラスター管理者は、503、404、またはその両方のエラーページにカスタムのエラーコード応答ページを指定できます。HAProxy ルーターは、アプリケーション Pod が実行していない場合や、要求された URL が存在しない場合に 404 エラーページを提供する 503 エラーページを提供します。たとえば、503 エラーコードの応答ページをカスタマイズする場合は、アプリケーション Pod が実行していないときにページが提供されます。また、デフォルトの 404 エラーコード HTTP 応答ページは、誤ったルートまたは存在しないルートについて HAProxy ルーターによって提供されます。

カスタムエラーコードの応答ページは ConfigMap に指定し、Ingress コントローラーにパッチを適用されます。ConfigMap キーには、**error-page-503.http** と **error-page-404.http** の 2 つの利用可能なファイル名があります。

カスタムの HTTP エラーコードの応答ページは、[HAProxy HTTP エラーページ設定のガイドライン](#) に従う必要があります。以下は、デフォルトの OpenShift Container Platform HAProxy ルーターの [http 503 エラーコード応答ページ](#) の例です。デフォルトのコンテンツを、独自のカスタムページを作成するためのテンプレートとして使用できます。

デフォルトで、HAProxy ルーターは、アプリケーションが実行していない場合や、ルートが正しくないまたは存在しない場合に 503 エラーページのみを提供します。このデフォルトの動作は、OpenShift Container Platform 4.8 以前の動作と同じです。HTTP エラーコード応答をカスタマイズするための ConfigMap が提供されておらず、カスタム HTTP エラーコード応答ページを使用している場合、ルーターはデフォルトの 404 または 503 エラーコード応答ページを提供します。



注記

カスタマイズ用のテンプレートとして OpenShift Container Platform のデフォルトの 503 エラーコードページを使用する場合、ファイルのヘッダーには CRLF 行の終了よりも多くのエディターが必要になります。

手順

1. **openshift-config** に **my-custom-error-code-pages** という名前の ConfigMap を作成します。

```

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

```



重要

カスタムエラーコードの応答ページに適した形式を指定しない場合は、ルーター Pod が停止します。この停止を解決するには、ConfigMap を削除するか、または修正し、影響を受けるルーター Pod を削除して、正しい情報で再作成できるようにします。

- Ingress コントローラーにパッチを適用し、名前を指定して **my-custom-error-code-pages** ConfigMap を参照します。

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

Ingress Operator は、**openshift-config** namespace から **openshift-ingress** namespace に **my-custom-error-code-pages** ConfigMap をコピーします。Operator は、**openshift-ingress** namespace のパターン **<your_ingresscontroller_name>-errorpages** に従って ConfigMap に名前を付けます。

- コピーを表示します。

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

出力例

NAME	DATA	AGE
default-errorpages	2	25s 1

- 1** **default** の Ingress Controller カスタムリソース (CR) にパッチが適用されているため、ConfigMap 名の例は **default-errorpages** です。

- カスタムエラー応答ページを含む ConfigMap がルーターボリュームにマウントされることを確認します。ConfigMap キーは、カスタム HTTP エラーコード応答を持つファイル名です。

- 503 カスタム HTTP カスタムエラーコード応答の場合:

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

- 404 カスタム HTTP カスタムエラーコード応答の場合:

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

検証

カスタムエラーコード HTTP 応答を確認します。

- テストプロジェクトおよびアプリケーションを作成します。

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

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

2. 503 カスタム http エラーコード応答の場合:

- a. アプリケーションのすべての Pod を停止します。
- b. 以下の curl コマンドを実行するか、ブラウザでルートのホスト名にアクセスします。

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

3. 404 カスタム http エラーコード応答の場合:

- a. 存在しないルートまたは正しくないルートにアクセスします。
- b. 以下の curl コマンドを実行するか、ブラウザでルートのホスト名にアクセスします。

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

4. **errorfile** 属性が **haproxy.config** ファイルで適切にあるかどうかを確認します。

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

6.9. 関連情報

- [カスタム PKI の設定](#)

第7章 エンドポイントへの接続の確認

Cluster Network Operator (CNO) は、クラスター内のリソース間の接続ヘルスチェックを実行するコントローラーである接続性チェックコントローラーを実行します。ヘルスチェックの結果を確認して、調査している問題が原因で生じる接続の問題を診断したり、ネットワーク接続を削除したりできます。

7.1. 実行する接続ヘルスチェック

クラスターリソースにアクセスできることを確認するには、以下のクラスター API サービスのそれぞれに対して TCP 接続が行われます。

- Kubernetes API サーバーサービス
- Kubernetes API サーバーエンドポイント
- OpenShift API サーバーサービス
- OpenShift API サーバーエンドポイント
- ロードバランサー

サービスおよびサービスエンドポイントがクラスター内のすべてのノードで到達可能であることを確認するには、以下の各ターゲットに対して TCP 接続が行われます。

- ヘルスチェックターゲットサービス
- ヘルスチェックターゲットエンドポイント

7.2. 接続ヘルスチェックの実装

接続チェックコントローラーは、クラスター内の接続検証チェックをオーケストレーションします。接続テストの結果は、**openshift-network-diagnostics** namespace の **PodNetworkConnectivity** オブジェクトに保存されます。接続テストは、1分ごとに並行して実行されます。

Cluster Network Operator (CNO) は、接続性ヘルスチェックを送受信するためにいくつかのリソースをクラスターにデプロイします。

ヘルスチェックのソース

このプログラムは、**Deployment** オブジェクトで管理される単一の Pod レプリカセットにデプロイします。このプログラムは **PodNetworkConnectivity** オブジェクトを消費し、各オブジェクトで指定される **spec.targetEndpoint** に接続されます。

ヘルスチェックのターゲット

クラスターのすべてのノードにデーモンセットの一部としてデプロイされた Pod。Pod はインバウンドのヘルスチェックをリスンします。すべてのノードにこの Pod が存在すると、各ノードへの接続をテストすることができます。

7.3. PODNETWORKCONNECTIVITYCHECK オブジェクトフィールド

PodNetworkConnectivityCheck オブジェクトフィールドについては、以下の表で説明されています。

表7.1 PodNetworkConnectivityCheck オブジェクトフィールド

フィールド	型	説明
metadata.name	string	以下の形式のオブジェクトの名前: <source>-to-<target><target> で記述される宛先には、以下のいずれかの文字列が含まれます。 <ul style="list-style-type: none"> ● load-balancer-api-external ● load-balancer-api-internal ● kubernetes-apiserver-endpoint ● kubernetes-apiserver-service-cluster ● network-check-target ● openshift-apiserver-endpoint ● openshift-apiserver-service-cluster
metadata.namespace	string	オブジェクトが関連付けられる namespace。この値は、常に openshift-network-diagnostics になります。
spec.sourcePod	string	接続チェックの起点となる Pod の名前 (例: network-check-source-596b4c6566-rgh92)。
spec.targetEndpoint	string	api.devcluster.example.com:6443 などの接続チェックのターゲット。
spec.tlsClientCert	object	使用する TLS 証明書の設定。
spec.tlsClientCert.name	string	使用される TLS 証明書の名前 (ある場合)。デフォルト値は空の文字列です。
status	object	接続テストの状態を表す、および最近の接続の成功および失敗についてのログ。
status.conditions	array	接続チェックと最新のステータスと以前のステータス。
status.failures	array	試行に失敗した接続テストのログ。
status.outages	array	停止が生じた期間が含まれる接続テストのログ。
status.successes	array	試行に成功した接続テストのログ。

以下の表は、**status.conditions** 配列内のオブジェクトのフィールドについて説明しています。

表7.2 status.conditions

フィールド	型	説明
lastTransitionTime	string	接続の条件がある状態から別の状態に移行した時間。
message	string	人が判読できる形式の最後の移行についての詳細。
reason	string	マシンの読み取り可能な形式での移行の最後のステータス。
status	string	状態のステータス。
type	string	状態のタイプ。

以下の表は、**status.conditions** 配列内のオブジェクトのフィールドについて説明しています。

表7.3 status.outages

フィールド	型	説明
end	string	接続の障害が解決された時点からのタイムスタンプ。
endLogs	array	接続ログエントリ (停止の正常な終了に関連するログエントリを含む)。
message	string	人が判読できる形式の停止について詳細情報の要約。
start	string	接続の障害が最初に検知された時点からのタイムスタンプ。
startLogs	array	元の障害を含む接続ログのエントリ。

接続ログフィールド

接続ログエントリのフィールドの説明は以下の表で説明されています。オブジェクトは以下のフィールドで使用されます。

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

表7.4 接続ログオブジェクト

フィールド	型	説明
latency	string	アクションの期間を記録します。
message	string	ステータスを人が判読できる形式で提供します。
reason	string	ステータスの理由をマシンが判読できる形式で提供します。値は TCPConnect 、 TCPConnectError 、 DNSResolve 、 DNSError のいずれかになります。
success	boolean	ログエントリが成功または失敗であることを示します。
time	string	接続チェックの開始時間。

7.4. エンドポイントのネットワーク接続の確認

クラスター管理者は、API サーバー、ロードバランサー、サービス、または Pod などのエンドポイントの接続を確認できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 現在の **PodNetworkConnectivityCheck** オブジェクトを一覧表示するには、以下のコマンドを入力します。

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

出力例

```

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

```

```

network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-master-0          75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-master-1          75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-master-2          75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-worker-c-n8mbf    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-
ln-x5sv9rb-f76d1-4rzrp-worker-d-4hnrz    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-
service-cluster                          75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2 74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-
service-cluster                          75m

```

2. 接続テストログを表示します。

- a. 直前のコマンドの出力から、接続ログを確認するエンドポイントを特定します。
- b. オブジェクトを表示するには、以下のコマンドを入力します。

```

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

```

ここで、**<name>** は **PodNetworkConnectivityCheck** オブジェクトの名前を指定します。

出力例

```

apiVersion: controlplane.operator.openshift.io/v1alpha1
kind: PodNetworkConnectivityCheck
metadata:
  name: network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-
apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0
  namespace: openshift-network-diagnostics
  ...
spec:
  sourcePod: network-check-source-7c88f6d9f-hmg2f
  targetEndpoint: 10.0.0.4:6443
  tlsClientCert:
    name: ""
status:
  conditions:
  - lastTransitionTime: "2021-01-13T20:11:34Z"
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnectSuccess
    status: "True"
    type: Reachable

```

```
failures:
- latency: 2.241775ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
    to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
    connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:10:34Z"
- latency: 2.582129ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
    to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
    connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:09:34Z"
- latency: 3.483578ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
    to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
    connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:08:34Z"
outages:
- end: "2021-01-13T20:11:34Z"
endLogs:
- latency: 2.032018ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
    tcp connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T20:11:34Z"
- latency: 2.241775ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
    failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
    connect: connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:10:34Z"
- latency: 2.582129ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
    failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
    connect: connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:09:34Z"
- latency: 3.483578ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
    failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
    connect: connection refused'
  reason: TCPConnectError
  success: false
  time: "2021-01-13T20:08:34Z"
message: Connectivity restored after 2m59.999789186s
start: "2021-01-13T20:08:34Z"
startLogs:
- latency: 3.483578ms
```

```
message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
  failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
  connect: connection refused'
reason: TCPConnectError
success: false
time: "2021-01-13T20:08:34Z"
successes:
- latency: 2.845865ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:14:34Z"
- latency: 2.926345ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:13:34Z"
- latency: 2.895796ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:12:34Z"
- latency: 2.696844ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:11:34Z"
- latency: 1.502064ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:10:34Z"
- latency: 1.388857ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:09:34Z"
- latency: 1.906383ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:08:34Z"
- latency: 2.089073ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:07:34Z"
- latency: 2.156994ms
```

```
message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
  connection to 10.0.0.4:6443 succeeded'
reason: TCPConnect
success: true
time: "2021-01-13T21:06:34Z"
- latency: 1.777043ms
message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
  connection to 10.0.0.4:6443 succeeded'
reason: TCPConnect
success: true
time: "2021-01-13T21:05:34Z"
```

第8章 ノードポートサービス範囲の設定

クラスター管理者は、利用可能なノードのポート範囲を拡張できます。クラスターで多数のノードポートが使用される場合、利用可能なポートの数を増やす必要がある場合があります。

デフォルトのポート範囲は **30000-32767** です。最初にデフォルト範囲を超えて拡張した場合でも、ポート範囲を縮小することはできません。

8.1. 前提条件

- クラスターインフラストラクチャーは、拡張された範囲内で指定するポートへのアクセスを許可する必要があります。たとえば、ノードのポート範囲を **30000-32900** に拡張する場合、ファイアウォールまたはパケットフィルターリングの設定によりこれに含まれるポート範囲 **32768-32900** を許可する必要があります。

8.2. ノードのポート範囲の拡張

クラスターのノードポート範囲を拡張できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする。

手順

1. ノードのポート範囲を拡張するには、以下のコマンドを入力します。<port> を、新規の範囲内で最大のポート番号に置き換えます。

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

ヒント

または、以下の YAML を適用してノードのポート範囲を更新することもできます。

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

出力例

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

2. 設定がアクティブであることを確認するには、以下のコマンドを入力します。更新が適用されるまでに数分の時間がかかることがあります。

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

出力例

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

8.3. 関連情報

- [NodePort を使用した ingress クラスタートラフィックの設定](#)
- [Network \[config.openshift.io/v1\]](#)
- [Service \[core/v1\]](#)

第9章 IP フェイルオーバーの設定

このトピックでは、OpenShift Container Platform クラスターの Pod およびサービスの IP フェイルオーバーの設定について説明します。

IP フェイルオーバーは、ノードセットの仮想 IP (VIP) アドレスのプールを管理します。セットのすべての VIP はセットから選択されるノードによって提供されます。VIP は単一ノードが利用可能である限り提供されます。ノード上で VIP を明示的に配布する方法がないため、VIP のないノードがある可能性も、多数の VIP を持つノードがある可能性もあります。ノードが1つのみ存在する場合は、すべての VIP がそのノードに配置されます。



注記

VIP はクラスター外からルーティングできる必要があります。

IP フェイルオーバーは各 VIP のポートをモニターし、ポートがノードで到達可能かどうかを判別します。ポートが到達不能な場合、VIP はノードに割り当てられません。ポートが **0** に設定されている場合、このチェックは抑制されます。check スクリプトは必要なテストを実行します。

IP フェイルオーバーは [Keepalived](#) を使用して、一連のホストでの外部からアクセスできる VIP アドレスのセットをホストします。各 VIP は1度に1つのホストによって提供されます。Keepalived は Virtual Router Redundancy Protocol (VRRP) を使用して、(一連のホストの) どのホストがどの VIP を提供するかを判別します。ホストが利用不可の場合や Keepalived が監視しているサービスが応答しない場合は、VIP は一連のホストの別のホストに切り換えられます。したがって、VIP はホストが利用可能である限り常に提供されます。

Keepalived を実行するノードが check スクリプトを渡す場合、ノードの VIP はプリエンプションストラテジーに応じて、その優先順位および現在のマスターの優先順位に基づいて **master** 状態になることができます。

クラスター管理者は **OPENSIFT_HA_NOTIFY_SCRIPT** 変数を介してスクリプトを提供できます。このスクリプトは、ノードの VIP の状態が変更されるたびに呼び出されます。Keepalived は VIP を提供する場合は **master** 状態を、別のノードが VIP を提供する場合は **backup** 状態を、または check スクリプトが失敗する場合は **fault** 状態を使用します。notify スクリプトは、状態が変更されるたびに新規の状態呼び出されます。

OpenShift Container Platform で IP フェイルオーバーのデプロイメント設定を作成できます。IP フェイルオーバーのデプロイメント設定は VIP アドレスのセットを指定し、それらの提供先となるノードのセットを指定します。クラスターには複数の IP フェイルオーバーのデプロイメント設定を持たせることができ、それぞれが固有な VIP アドレスの独自のセットを管理します。IP フェイルオーバー設定の各ノードは IP フェイルオーバー Pod として実行され、この Pod は Keepalived を実行します。

VIP を使用してホストネットワークを持つ Pod にアクセスする場合、アプリケーション Pod は IP フェイルオーバー Pod を実行しているすべてのノードで実行されます。これにより、いずれの IP フェイルオーバーノードもマスターになり、必要時に VIP を提供することができます。アプリケーション Pod が IP フェイルオーバーのすべてのノードで実行されていない場合、一部の IP フェイルオーバーノードが VIP を提供できないか、または一部のアプリケーション Pod がトラフィックを受信できなくなります。この不一致を防ぐために、IP フェイルオーバーとアプリケーション Pod の両方に同じセクターとレプリケーション数を使用します。

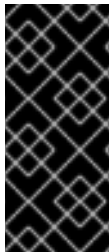
VIP を使用してサービスにアクセスしている間は、アプリケーション Pod が実行されている場所に関係なく、すべてのノードでサービスに到達できるため、任意のノードをノードの IP フェイルオーバーセットに含めることができます。いずれの IP フェイルオーバーノードも、いつでもマスターにすることができます。サービスは外部 IP およびサービスポートを使用するか、または **NodePort** を使用することができます。

サービス定義で外部 IP を使用する場合、VIP は外部 IP に設定され、IP フェイルオーバーのモニターリングポートはサービスポートに設定されます。ノードポートを使用する場合、ポートはクラスター内のすべてのノードで開かれ、サービスは、現在 VIP にサービスを提供しているあらゆるノードからのトラフィックの負荷を分散します。この場合、IP フェイルオーバーのモニターリングポートはサービス定義で **NodePort** に設定されます。



重要

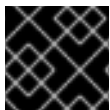
NodePort のセットアップは特権付きの操作で実行されます。



重要

サービス VIP の可用性が高い場合でも、パフォーマンスに影響が出る可能性があります。Keepalived は、各 VIP が設定内の一部のノードによってサービスされることを確認し、他のノードに VIP がない場合でも、複数の VIP が同じノードに配置される可能性があります。IP フェイルオーバーによって複数の VIP が同じノードに配置されると、VIP のセット全体で外部から負荷分散される戦略が妨げられる可能性があります。

ingressIP を使用する場合、IP フェイルオーバーを **ingressIP** 範囲と同じ VIP 範囲を持つように設定できます。また、モニターリングポートを無効にすることもできます。この場合、すべての VIP がクラスター内の同じノードに表示されます。すべてのユーザーが **ingressIP** でサービスをセットアップし、これを高い可用性のあるサービスにすることができます。



重要

クラスター内の VIP の最大数は 254 です。

9.1. IP フェイルオーバーの環境変数

以下の表は、IP フェイルオーバーの設定に使用される変数を示しています。

表9.1 IP フェイルオーバーの環境変数

変数名	デフォルト	説明
OPENSIFT_HA_MONITOR_PORT	80	IP フェイルオーバー Pod は、各仮想 IP (VIP) のこのポートに対して TCP 接続を開こうとします。接続が設定されると、サービスは実行中であると見なされます。このポートが 0 に設定される場合、テストは常にパスします。
OPENSIFT_HA_NETWORK_INTERFACE		IP フェイルオーバーが Virtual Router Redundancy Protocol (VRRP) トラフィックの送信に使用するインターフェイス名。デフォルト値は eth0 です。
OPENSIFT_HA_REPLICA_COUNT	2	作成するレプリカの数です。これは、IP フェイルオーバーデプロイメント設定の spec.replicas 値に一致する必要があります。
OPENSIFT_HA_VIRTUAL_IPS		複製する IP アドレス範囲の一覧です。これは指定する必要があります例: 1.2.3.4-6,1.2.3.9

変数名	デフォルト	説明
OPENSIFT_HA_VRRP_ID_OFFSET	0	仮想ルーター ID の設定に使用されるオフセット値。異なるオフセット値を使用すると、複数の IP フェイルオーバー設定が同じクラスター内に存在できるようになります。デフォルトのオフセットは 0 で、許可される範囲は 0 から 255 までです。
OPENSIFT_HA_VIP_GROUPS		VRRP に作成するグループの数です。これが設定されていない場合、グループは OPENSIFT_HA_VIP_GROUPS 変数で指定されている仮想 IP 範囲ごとに作成されます。
OPENSIFT_HA_IPTABLES_CHAIN	INPUT	iptables チェーンの名前であり、 iptables ルールを自動的に追加し、VRRP トラフィックをオンにすることを許可するために使用されます。この値が設定されていない場合、 iptables ルールは追加されません。チェーンが存在しない場合は作成されません。
OPENSIFT_HA_CHECK_SCRIPT		アプリケーションが動作していることを確認するために定期的に行われるスクリプトの Pod ファイルシステム内の完全パス名です。
OPENSIFT_HA_CHECK_INTERVAL	2	check スクリプトが実行される期間 (秒単位) です。
OPENSIFT_HA_NOTIFY_SCRIPT		状態が変更されるたびに実行されるスクリプトの Pod ファイルシステム内の完全パス名です。
OPENSIFT_HA_PREEMPTION	preempt_nodelay 300	新たな優先度の高いホストを処理するためのストラテジーです。 nopreempt ストラテジーでは、マスターを優先度の低いホストから優先度の高いホストに移動しません。

9.2. IP フェイルオーバーの設定

クラスター管理者は、クラスター全体に IP フェイルオーバーを設定することも、ラベルセレクターの定義に基づいてノードのサブセットに IP フェイルオーバーを設定することもできます。また、複数の IP フェイルオーバーのデプロイメント設定をクラスター内に設定することもでき、それぞれの設定をクラスター内で相互に切り離すことができます。

IP フェイルオーバーのデプロイメント設定により、フェイルオーバー Pod は、制約または使用されるラベルに一致する各ノードで確実に実行されます。

この Pod は Keepalived を実行します。これは、最初のノードがサービスまたはエンドポイントに到達できない場合に、エンドポイントを監視し、Virtual Router Redundancy Protocol (VRRP) を使用して仮想 IP (VIP) を別のノードにフェイルオーバーできます。

実稼働環境で使用する場合は、少なくとも 2 つのノードを選択し、選択したノードの数に相当する **replicas** を設定する **selector** を設定します。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていること。
- プルシークレットを作成している。

手順

1. IP フェイルオーバーのサービスアカウントを作成します。

```
$ oc create sa ipfailover
```

2. **hostNetwork** の SCC (Security Context Constraints) を更新します。

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

3. デプロイメント YAML ファイルを作成して IP フェイルオーバーを設定します。

IP フェイルオーバー設定のデプロイメント YAML の例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ipfailover-keepalived 1
  labels:
    ipfailover: hello-openshift
spec:
  strategy:
    type: Recreate
  replicas: 2
  selector:
    matchLabels:
      ipfailover: hello-openshift
  template:
    metadata:
      labels:
        ipfailover: hello-openshift
    spec:
      serviceAccountName: ipfailover
      privileged: true
      hostNetwork: true
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      containers:
        - name: openshift-ipfailover
          image: quay.io/openshift/origin-keepalived-ipfailover
          ports:
            - containerPort: 63000
              hostPort: 63000
          imagePullPolicy: IfNotPresent
          securityContext:
            privileged: true
          volumeMounts:
            - name: lib-modules
```

```

    mountPath: /lib/modules
    readOnly: true
  - name: host-slash
    mountPath: /host
    readOnly: true
    mountPropagation: HostToContainer
  - name: etc-sysconfig
    mountPath: /etc/sysconfig
    readOnly: true
  - name: config-volume
    mountPath: /etc/keepalive
env:
  - name: OPENSIFT_HA_CONFIG_NAME
    value: "ipfailover"
  - name: OPENSIFT_HA_VIRTUAL_IPS ❷
    value: "1.1.1.1-2"
  - name: OPENSIFT_HA_VIP_GROUPS ❸
    value: "10"
  - name: OPENSIFT_HA_NETWORK_INTERFACE ❹
    value: "ens3" #The host interface to assign the VIPs
  - name: OPENSIFT_HA_MONITOR_PORT ❺
    value: "30060"
  - name: OPENSIFT_HA_VRRP_ID_OFFSET ❻
    value: "0"
  - name: OPENSIFT_HA_REPLICA_COUNT ❼
    value: "2" #Must match the number of replicas in the deployment
  - name: OPENSIFT_HA_USE_UNICAST
    value: "false"
  #- name: OPENSIFT_HA_UNICAST_PEERS
  #value: "10.0.148.40,10.0.160.234,10.0.199.110"
  - name: OPENSIFT_HA_IPTABLES_CHAIN ❽
    value: "INPUT"
  #- name: OPENSIFT_HA_NOTIFY_SCRIPT ❾
  # value: /etc/keepalive/mynotifyscript.sh
  - name: OPENSIFT_HA_CHECK_SCRIPT ❿
    value: "/etc/keepalive/mycheckscript.sh"
  - name: OPENSIFT_HA_PREEMPTION ⓫
    value: "preempt_delay 300"
  - name: OPENSIFT_HA_CHECK_INTERVAL ⓬
    value: "2"
livenessProbe:
  initialDelaySeconds: 10
  exec:
    command:
      - pgrep
      - keepalived
volumes:
  - name: lib-modules
    hostPath:
      path: /lib/modules
  - name: host-slash
    hostPath:
      path: /
  - name: etc-sysconfig
    hostPath:

```

```

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

```

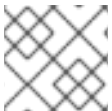
- 1 IP フェイルオーバーデプロイメントの名前。
- 2 複製する IP アドレス範囲の一覧です。これは指定する必要があります例: **1.2.3.4-6,1.2.3.9**
- 3 VRRP に作成するグループの数です。これが設定されていない場合、グループは **OPENSIFT_HA_VIP_GROUPS** 変数で指定されている仮想 IP 範囲ごとに作成されません。
- 4 IP フェイルオーバーが VRRP トラフィックの送信に使用するインターフェイス名。デフォルトで **eth0** が使用されます。
- 5 IP フェイルオーバー Pod は、各 VIP のこのポートに対して TCP 接続を開こうとします。接続が設定されると、サービスは実行中であると見なされます。このポートが **0** に設定される場合、テストは常にパスします。デフォルト値は **80** です。
- 6 仮想ルーター ID の設定に使用されるオフセット値。異なるオフセット値を使用すると、複数の IP フェイルオーバー設定が同じクラスター内に存在できるようになります。デフォルトのオフセットは **0** で、許可される範囲は **0** から **255** までです。
- 7 作成するレプリカの数です。これは、IP フェイルオーバーデプロイメント設定の **spec.replicas** 値に一致する必要があります。デフォルト値は **2** です。
- 8 **iptables** チェーンの名前であり、**iptables** ルールを自動的に追加し、VRRP トラフィックをオンにすることを許可するために使用されます。この値が設定されていない場合、**iptables** ルールは追加されません。チェーンが存在しない場合は作成されず、Keepalived はユニキャストモードで動作します。デフォルトは **INPUT** です。
- 9 状態が変更されるたびに実行されるスクリプトの Pod ファイルシステム内の完全パス名です。
- 10 アプリケーションが動作していることを確認するために定期的に行われるスクリプトの Pod ファイルシステム内の完全パス名です。
- 11 新たな優先度の高いホストを処理するための戦略です。デフォルト値は **preempt_delay 300** で、優先順位の低いマスターが VIP を保持する場合に、Keepalived インスタンスが VIP を 5 分後に引き継ぎます。
- 12 check スクリプトが実行される期間 (秒単位) です。デフォルト値は **2** です。
- 13 デプロイメントを作成する前にプルシークレットを作成します。作成しない場合には、デプロイメントの作成時にエラーが発生します。

9.3. 仮想 IP アドレスについて

Keepalived は一連の仮想 IP アドレス (VIP) を管理します。管理者はこれらすべてのアドレスについて以下の点を確認する必要があります。

- 仮想 IP アドレスは設定されたホストでクラスター外からアクセスできる。
- 仮想 IP アドレスはクラスター内でこれ以外の目的で使用されていない。

各ノードの Keepalived は、必要とされるサービスが実行中であるかどうかを判別します。実行中の場合、VIP がサポートされ、Keepalived はネゴシエーションに参加してどのノードが VIP を提供するかを決定します。これに参加するノードについては、このサービスが VIP の監視ポートでリッスンしている、またはチェックが無効にされている必要があります。



注記

セット内の各 VIP は最終的に別のノードによって提供される可能性があります。

9.4. CHECK スクリプトおよび NOTIFY スクリプトの設定

Keepalived は、オプションのユーザー指定の check スクリプトを定期的に行ってアプリケーションの正常性をモニターします。たとえば、このスクリプトは要求を発行し、応答を検証することで web サーバーをテストします。

チェックスクリプトが指定されない場合、TCP 接続をテストする単純なデフォルトスクリプトが実行されます。このデフォルトテストは、モニターポートが **0** の場合は抑制されます。

各 IP フェイルオーバー Pod は、Pod が実行されているノードで1つ以上の仮想 IP (VIP) を管理する Keepalived デーモンを管理します。Keepalived デーモンは、ノードの各 VIP の状態を維持します。特定のノード上の特定の VIP は、**master**、**backup**、または **fault** 状態にある可能性があります。

master 状態にあるノードでその VIP の check スクリプトが失敗すると、そのノードの VIP は **fault** 状態になり、再ネゴシエーションがトリガーされます。再ネゴシエーションの中に **fault** 状態にないノード上のすべての VIP は、どのノードが VIP を引き継ぐかを決定することに参加します。最終的に VIP は一部のノードで **master** の状態に入り、VIP は他のノードで **backup** 状態のままになります。

backup 状態の VIP を持つノードに障害が発生すると、そのノードの VIP は **fault** 状態になります。 **fault** 状態のノード上の VIP の check スクリプトが再度パスすると、そのノードの VIP は **fault** 状態を終了し、**master** 状態に入るためにネゴシエートします。次に、そのノードの VIP は、**master** 状態または **backup** 状態のいずれかになります。

クラスター管理者は、オプションの notify スクリプトを提供できます。このスクリプトは状態が変更されるたびに呼び出されます。Keepalived は以下の3つのパラメーターをこのスクリプトに渡します。

- **\$1** - **group** または **instance**
- **\$2**: **group** または **instance** の名前です。
- **\$3**: 新規の状態: **master**、**backup**、または **fault**

check および notify スクリプトは、IP フェイルオーバー Pod で実行され、ホストファイルシステムではなく Pod ファイルシステムを使用します。ただし、IP フェイルオーバー Pod はホストファイルシステムが **/hosts** マウントパスで利用可能にします。check または notify スクリプトを設定する場合は、スクリプトへの完全パスを指定する必要があります。スクリプトを提供する方法として、ConfigMap の使用が推奨されます。

check および notify スクリプトの完全パス名は、Keepalived 設定ファイル (`/etc/keepalived/keepalived.conf`) に追加されます。このファイルは、Keepalived が起動するたびにロードされます。スクリプトは、以下のように ConfigMap を使用して Pod に追加できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

1. 必要なスクリプトを作成し、これを保持する ConfigMap を作成します。スクリプトには入力引数は指定されず、**OK** の場合は **0** を、**fail** の場合は **1** を返す必要があります。

check スクリプト **mycheckscript.sh**:

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

2. ConfigMap を作成します。

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

3. スクリプトを Pod に追加します。マウントされた設定マップファイルの **defaultMode** は、**oc** コマンドを使用して、またはデプロイメント設定を編集して実行できる必要があります。通常は、**0755**、**493** (10 進数) の値が使用されます。

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

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



注記

oc set env コマンドは空白を区別します。= 記号の両側に空白を入れることはできません。

ヒント

または、**ipfailover-keepalived** デプロイメント設定を編集することもできます。

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

```
spec:
  containers:
  - env:
    - name: OPENSIFT_HA_CHECK_SCRIPT ❶
      value: /etc/keepalive/mycheckscript.sh
  ...
  volumeMounts: ❷
  - mountPath: /etc/keepalive
    name: config-volume
  dnsPolicy: ClusterFirst
  ...
  volumes: ❸
  - configMap:
    defaultMode: 0755 ❹
    name: customrouter
    name: config-volume
  ...
```

- ❶ **spec.container.env** フィールドで、マウントされたスクリプトファイルを参照する **OPENSIFT_HA_CHECK_SCRIPT** 環境変数を追加します。
- ❷ **spec.container.volumeMounts** フィールドを追加してマウントポイントを作成します。
- ❸ 新規の **spec.volumes** フィールドを追加して ConfigMap に言及します。
- ❹ これはファイルの実行パーミッションを設定します。読み取られる場合は 10 進数 (**493**) で表示されます。

変更を保存し、エディターを終了します。これにより **ipfailover-keepalived** が再起動されます。

9.5. VRRP プリエンプションの設定

ノードの仮想 IP (VIP) が check スクリプトを渡すことで **fault** 状態を終了すると、ノードの VIP は、現在 **master** 状態にあるノードの VIP よりも優先度が低い場合は **backup** 状態になります。ただし、**fault** 状態を終了するノードの VIP の優先度が高い場合は、プリエンプションストラテジーによってクラスター内でのそのロールが決定されます。

nopreempt ストラテジーは **master** をホスト上の優先度の低いホストからホスト上の優先度の高い VIP に移動しません。デフォルトの **preempt_delay 300** の場合、Keepalived は指定された 300 秒の間待機し、**master** をホスト上の優先度の高い VIP に移動します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

- プリエンプションを指定するには、**oc edit deploy ipfailover-keepalived** を入力し、ルーターのデプロイメント設定を編集します。


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

```
...
spec:
  containers:
  - env:
    - name: OPENSIFT_HA_PREEMPTION ❶
      value: preempt_delay 300
  ...
```

❶ **OPENSIFT_HA_PREEMPTION** の値を設定します。

- **preempt_delay 300**: Keepalived は、指定された 300 秒の間待機し、**master** をホスト上の優先度の高い VIP に移動します。これはデフォルト値です。
- **nopreempt: master** をホスト上の優先度の低い VIP からホスト上の優先度の高い VIP に移動しません。

9.6. VRRP ID オフセットについて

IP フェイルオーバーのデプロイメント設定で管理される各 IP フェイルオーバー Pod (ノード/レプリカあたり 1 Pod) は Keepalived デーモンを実行します。設定される IP フェイルオーバーのデプロイメント設定が多くなると、作成される Pod も多くなり、共通の Virtual Router Redundancy Protocol (VRRP) ネゴシエーションに参加するデーモンも多くなります。このネゴシエーションはすべての Keepalived デーモンによって実行され、これはどのノードがどの仮想 IP (VIP) を提供するかを決定します。

Keepalived は内部で固有の **vrrp-id** を各 VIP に割り当てます。ネゴシエーションはこの **vrrp-ids** セットを使用し、決定後には優先される **vrrp-id** に対応する VIP が優先されるノードで提供されます。

したがって、IP フェイルオーバーのデプロイメント設定で定義されるすべての VIP について、IP フェイルオーバー Pod は対応する **vrrp-id** を割り当てる必要があります。これは、**OPENSIFT_HA_VRRP_ID_OFFSET** から開始し、順序に従って **vrrp-ids** を VIP の一覧に割り当てることによって実行されます。**vrrp-ids** には範囲 **1..255** の値を設定できます。

複数の IP フェイルオーバーのデプロイメント設定がある場合は、**OPENSIFT_HA_VRRP_ID_OFFSET** を指定して、デプロイメント設定内の VIP 数を増やす余地があり、**vrrp-id** 範囲が重複しないようにする必要があります。

9.7. 254 を超えるアドレスについての IP フェイルオーバーの設定

IP フェイルオーバー管理は、仮想 IP (VIP) アドレスの 254 グループに制限されています。デフォルトでは、OpenShift Container Platform は各グループに 1 つの IP アドレスを割り当てます。**OPENSIFT_HA_VIP_GROUPS** 変数を使用してこれを変更し、複数の IP アドレスが各グループに含まれるようにして、IP フェイルオーバーを設定するときに各 Virtual Router Redundancy Protocol (VRRP) インスタンスで使用可能な VIP グループの数を定義できます。

VIP の作成により、VRRP フェイルオーバーの発生時の広範囲の VRRP の割り当てが作成され、これはクラスター内のすべてのホストがローカルにサービスにアクセスする場合に役立ちます。たとえば、サービスが **ExternalIP** で公開されている場合などがこれに含まれます。



注記

フェイルオーバーのルールとして、ルーターなどのサービスは特定の1つのホストに制限しません。代わりに、サービスは、IP フェイルオーバーの発生時にサービスが新規ホストに再作成されないように各ホストに複製可能な状態にする必要があります。



注記

OpenShift Container Platform のヘルスチェックを使用している場合、IP フェイルオーバーおよびグループの性質上、グループ内のすべてのインスタンスはチェックされません。そのため、[Kubernetes ヘルスチェック](#) を使ってサービスが有効であることを確認する必要があります。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていること。

手順

- 各グループに割り当てられた IP アドレスの数を変更するには、**OPENSIFT_HA_VIP_GROUPS** 変数の値を変更します。次に例を示します。

IP フェイルオーバー設定の Deployment YAML の例

```

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

```

- 1** たとえば、7つのVIPのある環境で **OPENSIFT_HA_VIP_GROUPS** が **3** に設定されている場合、これは3つのグループを作成し、3つのVIPを最初のグループに、2つのVIPを2つの残りのグループにそれぞれ割り当てます。



注記

OPENSIFT_HA_VIP_GROUPS で設定されたグループの数が、フェイルオーバーに設定されたIPアドレスの数より少ない場合、グループには複数のIPアドレスが含まれ、すべてのアドレスが1つのユニットとして移動します。

9.8. INGRESSIP の高可用性

クラウド以外のクラスターでは、IP フェイルオーバーおよびサービスへの **ingressIP** を組み合わせることができます。結果として、**ingressIP** を使用してサービスを作成するユーザーに高可用性サービスが提供されます。

この方法では、まず **ingressIPNetworkCIDR** 範囲を指定し、次に **ipfailover** 設定を作成する際に同じ範囲を使用します。

IP フェイルオーバーはクラスター全体に対して最大 255 のVIPをサポートできるため、**ingressIPNetworkCIDR** は **/24** 以下に設定する必要があります。

9.9. IP フェイルオーバーの削除

IP フェイルオーバーが最初に設定されている場合、クラスターのワーカーノードは、Keepalived 用に **224.0.0.18** のマルチキャストパケットを明示的に許可する **iptables** ルールを使用して変更されます。ノードが変更されるため、IP フェイルオーバーを削除するには、ジョブを実行して **iptables** ルールを削除し、Keepalived が使用する仮想 IP アドレスを削除する必要があります。

手順

1. オプション: ConfigMap として保存されるチェックおよび通知スクリプトを特定し、削除します。
 - a. IP フェイルオーバーの Pod が ConfigMap をボリュームとして使用するかどうかを決定します。

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

出力例

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

- b. 前述の手順でボリュームとして使用される ConfigMap の名前が提供されている場合は、ConfigMap を削除します。

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

2. IP フェイルオーバーの既存デプロイメントを特定します。

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

出力例

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

3. デプロイメントを削除します。

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

4. **ipfailover** サービスアカウントを削除します。

```
$ oc delete sa ipfailover
```

5. IP フェイルオーバーの設定時に追加された IP テーブルルールを削除するジョブを実行します。
 - a. 以下の例のような内容で **remove-ipfailover-job.yaml** などのファイルを作成します。

```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: remove-ipfailover-
  labels:
    app: remove-ipfailover
spec:
  template:
    metadata:
      name: remove-ipfailover
    spec:
      containers:
        - name: remove-ipfailover
          image: quay.io/openshift/origin-keepalived-ipfailover:4.9
          command: ["/var/lib/ipfailover/keepalived/remove-failover.sh"]
      nodeSelector:
        kubernetes.io/hostname: <host_name> <.>
      restartPolicy: Never
```

<.> IP フェイルオーバー用に設定されたクラスター内の各ノードのジョブを実行し、毎回ホスト名を置き換えます。

- b. ジョブを実行します。

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

出力例

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

検証

- ジョブが IP フェイルオーバーの初期設定を削除していることを確認します。

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

出力例

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

第10章 ベアメタルクラスターでの SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の使用

クラスター管理者は、クラスターで SCTP (Stream Control Transmission Protocol) を使用できます。

10.1. OPENSIFT CONTAINER PLATFORM での SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) のサポート

クラスター管理者は、クラスターのホストで SCTP を有効にできます。Red Hat Enterprise Linux CoreOS (RHCOS) で、SCTP モジュールはデフォルトで無効にされています。

SCTP は、IP ネットワークの上部で実行される信頼できるメッセージベースのプロトコルです。

これを有効にすると、SCTP を Pod、サービス、およびネットワークポリシーでプロトコルとして使用できます。**Service** オブジェクトは、**type** パラメーターを **ClusterIP** または **NodePort** のいずれかの値に設定して定義する必要があります。

10.1.1. SCTP プロトコルを使用した設定例

protocol パラメーターを Pod またはサービスリソース定義の **SCTP** 値に設定して、Pod またはサービスを SCTP を使用するように設定できます。

以下の例では、Pod は SCTP を使用するように設定されています。

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

以下の例では、サービスは SCTP を使用するように設定されています。

```
apiVersion: v1
kind: Service
metadata:
  namespace: project1
  name: sctpserver
spec:
  ...
  ports:
  - name: sctpserver
    protocol: SCTP
    port: 30100
    targetPort: 30100
  type: ClusterIP
```

以下の例では、**NetworkPolicy** オブジェクトは、特定のラベルの付いた Pod からポート **80** の SCTP ネットワークトラフィックに適用するように設定されます。

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

10.2. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) の有効化

クラスター管理者は、クラスターのワーカーノードでブラックリストに指定した SCTP カーネルモジュールを読み込み、有効にできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. 以下の YAML 定義が含まれる **load-sctp-module.yaml** という名前のファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: load-sctp-module
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: /etc/modprobe.d/sctp-blacklist.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,
        - path: /etc/modules-load.d/sctp-load.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:;,sctp
```

2. **MachineConfig** オブジェクトを作成するには、以下のコマンドを入力します。

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

3. オプション: MachineConfig Operator が設定変更を適用している間にノードのステータスを確認するには、以下のコマンドを入力します。ノードのステータスが **Ready** に移行すると、設定の更新が適用されます。

```
$ oc get nodes
```

10.3. SCTP (STREAM CONTROL TRANSMISSION PROTOCOL) が有効になっていることの確認

SCTP がクラスターで機能することを確認するには、SCTP トラフィックをリスンするアプリケーションで Pod を作成し、これをサービスに関連付け、公開されたサービスに接続します。

前提条件

- クラスターからインターネットにアクセスし、**nc** パッケージをインストールすること。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. SCTP リスナーを起動する Pod を作成します。
 - a. 以下の YAML で Pod を定義する **sctp-server.yaml** という名前のファイルを作成します。

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

- b. 以下のコマンドを入力して Pod を作成します。

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

2. SCTP リスナー Pod のサービスを作成します。

- a. 以下の YAML でサービスを定義する **sctp-service.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: sctpservice
  labels:
    app: sctpserver
spec:
  type: NodePort
  selector:
    app: sctpserver
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30102
      targetPort: 30102
```

- b. サービスを作成するには、以下のコマンドを入力します。

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

3. SCTP クライアントの Pod を作成します。

- a. 以下の YAML で **sctp-client.yaml** という名前のファイルを作成します。

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

- b. **Pod** オブジェクトを作成するには、以下のコマンドを入力します。

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

4. サーバーで SCTP リスナーを実行します。

- a. サーバー Pod に接続するには、以下のコマンドを入力します。

```
$ oc rsh sctpserver
```

- b. SCTP リスナーを起動するには、以下のコマンドを入力します。

```
$ nc -l 30102 --sctp
```


5. サーバーの SCTP リスナーに接続します。
 - a. ターミナルプログラムで新規のターミナルウィンドウまたはタブを開きます。
 - b. **sctp** サービスの IP アドレスを取得します。以下のコマンドを入力します。

```
$ oc get services sctp -o go-template='{{.spec.clusterIP}}{\n}'
```
 - c. クライアント Pod に接続するには、以下のコマンドを入力します。

```
$ oc rsh sctpclient
```
 - d. SCTP クライアントを起動するには、以下のコマンドを入力します。<cluster_IP> を **sctp** サービスのクラスター IP アドレスに置き換えます。

```
# nc <cluster_IP> 30102 --sctp
```

第11章 PTP ハードウェアの使用



重要

境界クロックとして設定した PTP (Precision Time Protocol) ハードウェアは、テクノロジープレビュー機能としてのみ提供されています。テクノロジープレビュー機能は、Red Hat の実稼働環境におけるサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

11.1. PTP ハードウェアについて

OpenShift Container Platform では、ノード上で PTP ハードウェアを使用できます。linuxptp サービスは、PTP 対応ハードウェアを搭載したノードで設定できます。



注記

PTP Operator は、ベアメタルインフラストラクチャーでのみプロビジョニングされるクラスタの PTP 対応デバイスと連携します。

PTP Operator をデプロイし、OpenShift Container Platform コンソールまたは **oc** を使用して PTP をインストールできます。PTP Operator は linuxptp サービスを作成し、管理し、以下の機能を提供します。

- クラスタ内の PTP 対応デバイスの検出。
- linuxptp サービスの設定の管理。
- PTP Operator **cloud-event-proxy** サイドカーによるアプリケーションのパフォーマンスおよび信頼性に悪影響を与える PTP クロックイベントの通知。

11.2. PTP について

Precision Time Protocol (PTP) は、ネットワーク内のクロックを同期するのに使用されます。ハードウェアサポートと併用する場合、PTP はマイクロ秒以下の正確性があり、Network Time Protocol (NTP) よりも正確になります。

linuxptp パッケージには、クロック同期用の **ptp4l** および **phc2sys** プログラムが含まれています。**ptp4l** は、PTP 境界クロックと通常のクロックを実装します。**ptp4l** は PTP ハードウェアクロックをハードウェアのタイムスタンプにソースクロックに同期し、システムクロックをソフトウェアタイムスタンプとクロックに同期します。**phc2sys** は、ネットワークインターフェイスコントローラー (NIC) 上の PTP ハードウェアクロックに同期するために、ハードウェアタイムスタンプに使用されます。

11.2.1. PTP ドメインの要素

PTP は、ネットワークに接続された複数のノードを各ノードのクロックと同期するために使用されます。以下のタイプのクロックを設定に追加できます。

グランドマスタークロック

グランドマスタークロックは、ネットワーク全体の他のクロックに標準時間情報を提供し、正確で安定した同期を保証します。グランドマスタークロックはタイムスタンプを書き込み、他のクロックからのタイムリクエストに応答します。

通常のクロック

通常のクロックには、ネットワーク内の位置に応じて、送信元クロックまたは宛先クロックのロールを果たすことができる単一のポート接続があります。通常のクロックは、タイムスタンプの読み取りおよび書き込みが可能です。

境界クロック

境界クロックには、2つ以上の通信パスにあるポートがあり、ソースと宛先の宛先を同時に他の宛先クロックに指定できます。境界クロックは、宛先クロックアップストリームとして機能します。宛先クロックはタイミングメッセージを受け取り、遅延に合わせて調整し、ネットワークを渡す新しいソースタイムシグナルを作成します。境界クロックは、ソースクロックと正しく同期され、ソースクロックに直接レポートする接続されたデバイスの数を減らすことができる新しいタイミングパケットを生成します。

11.2.2. NTP 上の PTP の利点

PTP が NTP を経由した主な利点の1つは、さまざまなネットワークインターフェイスコントローラー (NIC) およびネットワークスイッチにあるハードウェアサポートです。この特化されたハードウェアにより、PTP はメッセージ送信の遅れを説明でき、時間同期の精度を高められます。可能な限りの精度を実現するには、PTP クロック間の全ネットワークコンポーネントが PTP ハードウェアを有効にすることが推奨されます。

NIC は PTP パケットを送受信した瞬間にタイムスタンプを付けることができるため、ハードウェアベースの PTP は最適な精度を提供します。これをソフトウェアベースの PTP と比較します。これには、オペレーティングシステムによる PTP パケットの追加処理が必要になります。



重要

PTP を有効にする前に、必要なノードについて NTP が無効になっていることを確認します。**MachineConfig** カスタムリソースを使用して `chrony` タイムサービス (**chronyd**) を無効にすることができます。詳細は、[chrony タイムサービスの無効化](#) を参照してください。

11.3. CLI を使用した PTP OPERATOR のインストール

クラスター管理者は、CLI を使用して Operator をインストールできます。

前提条件

- PTP に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. PTP Operator の namespace を作成するには、以下のコマンドを入力します。

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

```

apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ntp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    name: openshift-ntp
    openshift.io/cluster-monitoring: "true"
EOF

```

2. Operator の Operator グループを作成するには、以下のコマンドを入力します。

```

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

```

3. PTP Operator にサブスクライブします。

- a. 以下のコマンドを実行して、OpenShift Container Platform のメジャーおよびマイナーバージョンを環境変数として設定します。これは次の手順で **channel** の値として使用されま

```

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

```

- b. PTP Operator のサブスクリプションを作成するには、以下のコマンドを入力します。

```

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

```

4. Operator がインストールされていることを確認するには、以下のコマンドを入力します。

```

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

```

出力例

Name	Phase
ptp-operator.4.4.0-202006160135	Succeeded

11.4. WEB コンソールを使用した PTP OPERATOR のインストール

クラスター管理者は、Web コンソールを使用して PTP Operator をインストールできます。



注記

先のセクションで説明されているように namespace および Operator グループを作成する必要があります。

手順

1. OpenShift Container Platform Web コンソールを使用して PTP Operator をインストールします。
 - a. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** をクリックします。
 - b. 利用可能な Operator の一覧から **PTP Operator** を選択してから **Install** をクリックします。
 - c. **Install Operator** ページの **A specific namespace on the cluster** の下で **openshift-ptp** を選択します。次に、**Install** をクリックします。
2. オプション: PTP Operator が正常にインストールされていることを確認します。
 - a. **Operators** → **Installed Operators** ページに切り替えます。
 - b. **PTP Operator** が **Status** が **InstallSucceeded** の状態で **openshift-ptp** プロジェクトに一覧表示されていることを確認します。



注記

インストール時に、Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

- **Operators** → **Installed Operators** ページに移動し、**Operator Subscriptions** および **Install Plans** タブで **Status** にエラーがあるかどうかを检查します。
- **Workloads** → **Pods** ページに移動し、**openshift-ptp** プロジェクトで Pod のログを確認します。

11.5. PTP ネットワークデバイスの自動検出

PTP Operator は **NodePtpDevice.ptp.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。

PTP Operator はクラスターで、各ノードの PTP 対応ネットワークデバイスを検索します。これは、互換性のある PTP デバイスを提供する各ノードの **NodePtpDevice** カスタムリソース (CR) オブジェクトを作成し、更新します。

1つの CR がノードごとに作成され、ノードと同じ名前を共有します。**.status.devices** 一覧は、ノード上の PTP デバイスについての情報を提供します。

以下は、PTP Operator によって作成される **NodePtpDevice** CR の例です。

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2019-11-15T08:57:11Z"
  generation: 1
  name: dev-worker-0 1
  namespace: openshift-ptp 2
  resourceVersion: "487462"
  selfLink: /apis/ptp.openshift.io/v1/namespaces/openshift-ptp/nodeptpdevices/dev-worker-0
  uid: 08d133f7-aae2-403f-84ad-1fe624e5ab3f
spec: {}
status:
  devices: 3
  - name: eno1
  - name: eno2
  - name: ens787f0
  - name: ens787f1
  - name: ens801f0
  - name: ens801f1
  - name: ens802f0
  - name: ens802f1
  - name: ens803
```

- 1** **name** パラメーターの値はノードの名前と同じです。
- 2** CR は PTP Operator によって **openshift-ptp** namespace に作成されます。
- 3** **devices** コレクションには、ノード上の Operator によって検出される PTP 対応デバイスの一覧が含まれます。

クラスター内の PTP 対応ネットワークデバイスの一覧を返すには、以下のコマンドを実行します。

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

11.6. LINUXPTP サービスを通常のクロックとして設定

PTP Operator は **PtpConfig.ptp.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。**PtpConfig** カスタムリソース (CR) オブジェクトを作成して、**linuxptp** サービス (**ptp4l**、**phc2sys**) を設定できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

- PTP Operator をインストールします。

手順

1. 以下の **PtpConfig** CR を作成してから、YAML を **ordinary-clock-ptp-config.yaml** ファイルに保存します。

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ordinary-clock-ptp-config ❶
  namespace: openshift-ptp
spec:
  profile: ❷
  - name: "profile1" ❸
    interface: "ens787f1" ❹
    ptp4lOpts: "-s -2" ❺
    phc2sysOpts: "-a -r" ❻
    ptp4lConf: "" ❼
    ptpSchedulingPolicy: SCHED_OTHER ❽
    ptpSchedulingPriority: 10 ❾
  recommend: ❿
  - profile: "profile1" ⓫
    priority: 10 ⓬
    match: ⓭
    - nodeLabel: "node-role.kubernetes.io/worker" ⓮
      nodeName: "compute-0.example.com" ⓯

```

- ❶ **PtpConfig** CR の名前。
- ❷ 1つ以上の **profile** オブジェクトの配列を指定します。
- ❸ プロファイルオブジェクトを一意に識別するプロファイルオブジェクトの名前を指定します。
- ❹ **ptp4l** サービスで使用するネットワークインターフェイス名を指定します (例: **ens787f1**)。
- ❺ **ptp4l** サービスのシステム設定オプションを指定します。たとえば、**-2** で IEEE 802.3 ネットワークトランスポートを選択します。ネットワークインターフェイス名とサービス設定ファイルが自動的に追加されるため、オプションには、ネットワークインターフェイス名 **-i <interface>** およびサービス設定ファイル **-f /etc/ptp4l.conf** を含めないでください。
- ❻ **phc2sys** サービスのシステム設定オプション (例: **-a -r**) を指定します。このフィールドが空の場合、PTP Operator は **phc2sys** サービスを開始しません。
- ❼ デフォルトの **/etc/ptp4l.conf** ファイルを置き換える設定が含まれる文字列を指定します。デフォルト設定を使用するには、フィールドを空のままにします。
- ❽ **ptp4l** と **phc2sys** プロセスのスケジューリングポリシー。デフォルト値は **SCHED_OTHER** です。FIFO スケジューリングをサポートするシステムでは、**SCHED_FIFO** を使用してください。
- ❾ **ptp SchedulingPolicy** が **SCHED_FIFO** に設定されている場合に、**ptp4l** および **phc2sys** プロセスの FIFO の優先度を設定するために使用される 1-65 の整数値 **ptpSchedulingPriority** フィールドは **ptpSchedulingPolicy** が **SCHED_OTHER** に

10. `priority` フィールドは、`priorityPolicy` の `SCHED_OTHER` に設定されている場合は使用されません。

- 10 **profile** がノードに適用される方法を定義する1つ以上の **recommend** オブジェクトの配列を指定します。
- 11 **profile** セクションに定義される **profile** オブジェクト名を指定します。
- 12 0 から 99 までの整数値で **priority** を指定します。数値が大きいくほど優先度が低くなるため、99 の優先度は 10 よりも低くなります。ノードが **match** フィールドで定義されるルールに基づいて複数のプロファイルに一致する場合、優先順位の高いプロファイルがそのノードに適用されます。
- 13 **match** ルールを、**nodeLabel** または **nodeName** で指定します。
- 14 `oc get nodes --show-labels` コマンドを使用して、ノードオブジェクトの `node.Labels` の `key` で `nodeLabel` を指定します。
- 15 `oc get nodes` コマンドを使用して、ノードオブジェクトの `node.Name` で `nodeName` を指定します。

2. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f ordinary-clock-ntp-config.yaml
```

検証手順

1. **PtpConfig** プロファイルがノードに適用されていることを確認します。
 - a. 以下のコマンドを実行して、**openshift-ntp** namespace の Pod の一覧を取得します。

```
$ oc get pods -n openshift-ntp -o wide
```

出力例

```
NAME                                READY STATUS RESTARTS AGE IP             NODE
linuxntp-daemon-4xkbb               1/1   Running  0      43m 10.1.196.24   compute-0.example.com
linuxntp-daemon-tdspf               1/1   Running  0      43m 10.1.196.25   compute-1.example.com
ntp-operator-657bbb64c8-2f8sj      1/1   Running  0      43m 10.129.0.61   control-plane-1.example.com
```

- b. プロファイルが正しいことを確認します。**PtpConfig** プロファイルで指定したノードに対応する **linuxntp** デーモンのログを検査します。以下のコマンドを実行します。

```
$ oc logs linuxntp-daemon-4xkbb -n openshift-ntp -c linuxntp-daemon-container
```

出力例

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] -----
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
```



```

I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -s -2
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r
I1115 09:41:17.117626 4143292 daemon.go:116] -----

```

関連情報

- PTP ハードウェアでの FIFO 優先度スケジューリングの詳細については、PTP ハードウェアの [FIFO 優先度スケジューリングの設定](#) を参照してください。

11.7. LINUXPTP サービスを境界クロックとして設定

PTP Operator は **PtpConfig.ptp.openshift.io** カスタムリソース定義 (CRD) を OpenShift Container Platform に追加します。**PtpConfig** カスタムリソース (CR) オブジェクトを作成して、**linuxptp** サービス (**ptp4l**、**phc2sys**) を設定できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- PTP Operator をインストールします。

手順

1. 以下の **PtpConfig** CR を作成してから、YAML を **boundary-clock-ptp-config.yaml** ファイルに保存します。

```

apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config ①
  namespace: openshift-ptp
spec:
  profile: ②
  - name: "profile1" ③
    interface: "" ④
    ptp4lOpts: "-2" ⑤
    ptp4lConf: | ⑥
      [ens1f0] ⑦
      masterOnly 0
      [ens1f3] ⑧
      masterOnly 1
      [global]
      #
      # Default Data Set
      #
      twoStepFlag          1
      #slaveOnly           1
      priority1            128
      priority2            128
      domainNumber        24

```

```
#utc_offset          37
clockClass           248
clockAccuracy        0xFE
offsetScaledLogVariance 0xFFFF
free_running         0
freq_est_interval    1
dscp_event           0
dscp_general         0
dataset_comparison   G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval      -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout  0
delayAsymmetry       0
fault_reset_interval 4
neighborPropDelayThresh 20000000
masterOnly            0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step      0
logging_level        6
path_trace_enabled   0
follow_up_info       0
hybrid_e2e          0
inhibit_multicast_service 0
net_sync_monitor     0
tc_spanning_tree     0
tx_timestamp_timeout 10
#was 1 (default !)
unicast_listen       0
unicast_master_table 0
unicast_req_duration 3600
use_syslog           1
verbose              0
summary_interval     -4
kernel_leap          1
check_fup_sync       0
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const     0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale     0.0
pi_integral_exponent  0.4
pi_integral_norm_max  0.3
```

```

step_threshold          2.0
first_step_threshold    0.00002
max_frequency           900000000
clock_servo             pi
sanity_freq_limit       200000000
ntpshm_segment          0
#
# Transport options
#
transportSpecific       0x0
ptp_dst_mac             01:1B:19:00:00:00
p2p_dst_mac             01:80:C2:00:00:0E
udp_ttl                 1
udp6_scope              0x0E
uds_address             /var/run/ptp4l
#
# Default interface options
#
clock_type              BC
network_transport       UDPv4
delay_mechanism         E2E
time_stamping           hardware
tsproc_mode            filter
delay_filter            moving_median
delay_filter_length     10
egressLatency           0
ingressLatency          0
boundary_clock_jbod 0 9
#
# Clock description
#
productDescription      ;;
revisionData            ;;
manufacturerIdentity   00:00:00
userDescription         ;
timeSource              0xA0
phc2sysOpts: "-a -r" 10
ptpSchedulingPolicy: SCHED_OTHER 11
ptpSchedulingPriority: 10 12
recommend: 13
- profile: "profile1" 14
priority: 10 15
match: 16
- nodeLabel: "node-role.kubernetes.io/worker" 17
  nodeName: "compute-0.example.com" 18

```

- ① **PtpConfig** CR の名前。
- ② 1つ以上の **profile** オブジェクトの配列を指定します。
- ③ プロファイルオブジェクトを一意に識別するプロファイルオブジェクトの名前を指定します。
- ④ このフィールドは、境界クロックの場合は空のままにする必要があります。

- 5 **ptp4l** サービスのシステム設定オプション (例: **-2**) を指定します。ネットワークインターフェイス名とサービス設定ファイルが自動的に追加されるため、オプションには、ネットワーク
- 6 **ptp4l** を境界クロックとして起動するために必要な設定を指定します。たとえば、**ens1f0** はグラントマスタークロックから同期し、**ens1f3** は接続されたデバイスを同期します。
- 7 同期元のインターフェイス名。
- 8 インターフェイスに接続されたデバイスを同期するインターフェイス。
- 9 Intel Columbiaville 800 Series NIC の場合、**boundary_clock_jbod** が **0** に設定されていることを確認します。Intel Fortville X710 シリーズ NIC の場合、**boundary_clock_jbod** が **1** に設定されていることを確認します。
- 10 **phc2sys** サービスのシステム設定オプション (例: **-a -r**) を指定します。このフィールドが空の場合、PTP Operator は **phc2sys** サービスを開始しません。
- 11 **ptp4l** と **phc2sys** プロセスのスケジューリングポリシー。デフォルト値は **SCHED_OTHER** です。FIFO スケジューリングをサポートするシステムでは、**SCHED_FIFO** を使用してください。
- 12 **ptp SchedulingPolicy** が **SCHED_FIFO** に設定されている場合に、**ptp4l** および **phc2sys** プロセスの FIFO の優先度を設定するために使用される 1-65 の整数値。**ptpSchedulingPriority** フィールドは、**ptpSchedulingPolicy** が **SCHED_OTHER** に設定されている場合は使用されません。
- 13 **profile** がノードに適用される方法を定義する 1 つ以上の **recommend** オブジェクトの配列を指定します。
- 14 **profile** セクションに定義される **profile** オブジェクト名を指定します。
- 15 **0** から **99** までの整数値で **priority** を指定します。数値が大きいほど優先度が低くなるため、**99** の優先度は **10** よりも低くなります。ノードが **match** フィールドで定義されるルールに基づいて複数のプロファイルに一致する場合、優先順位の高いプロファイルがそのノードに適用されます。
- 16 **match** ルールを、**nodeLabel** または **nodeName** で指定します。
- 17 **oc get nodes --show-labels** コマンドを使用して、ノードオブジェクトの **node.Labels** の **key** で **nodeLabel** を指定します。
- 18 **oc get nodes** コマンドを使用して、ノードオブジェクトの **node.Name** で **nodeName** を指定します。

2. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f boundary-clock-ptp-config.yaml
```

検証手順

1. **PtpConfig** プロファイルがノードに適用されていることを確認します。
 - a. 以下のコマンドを実行して、**openshift-ptp** namespace の Pod の一覧を取得します。

```
$ oc get pods -n openshift-ptp -o wide
```

出力例

```

NAME                READY STATUS  RESTARTS  AGE  IP           NODE
linuxptp-daemon-4xkbb 1/1   Running  0         43m  10.1.196.24 compute-
0.example.com
linuxptp-daemon-tdspf 1/1   Running  0         43m  10.1.196.25 compute-
1.example.com
ptp-operator-657bbb64c8-2f8sj 1/1   Running  0         43m  10.129.0.61 control-
plane-1.example.com

```

- b. プロファイルが正しいことを確認します。**PtpConfig** プロファイルで指定したノードに対応する **linuxptp** デモンのログを検査します。以下のコマンドを実行します。

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
```

出力例

```

I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] -----
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r
I1115 09:41:17.117626 4143292 daemon.go:116] -----

```

関連情報

- PTP ハードウェアでの FIFO 優先度スケジューリングの詳細については、PTP ハードウェアの [FIFO 優先度スケジューリングの設定](#) を参照してください。

11.8. PTP ハードウェアの FIFO 優先スケジューリングの設定

低遅延のパフォーマンスを確保する必要がある通信業者や他のデプロイメント設定では、PTP デモンスレッドは、制約された CPU フットプリントで、残りのインフラストラクチャーのコンポーネントと一緒に、実行されます。デフォルトでは、PTP スレッドは **SCHED_OTHER** ポリシーで実行されます。負荷が高いと、エラーなしで運用する必要のある、これらのスレッドのスケジューリングでレイテンシーが発生する可能性があります。

スケジューリングのレイテンシーでエラーが発生する可能性を軽減するために、**SCHED_FIFO** ポリシーでスレッドを実行できるように、PTP Operator の **linuxptp** サービスを設定できます。**Ptp ConfigCR** に **SCHED_FIFO** が設定されている場合には、**ptp4l** と **phc2sys** は、**Ptp ConfigCR** の **ptp Scheduling Priority** フィールドで設定された優先順位で、**chrt** の下の親コンテナで実行されます。



注記

ptp Scheduling Policy の設定はオプションで、レイテンシーエラーが発生している場合にのみ必要となります。

手順

1. **Ptp Config** CR プロファイルを編集します。

```
$ oc edit PtpConfig -n openshift-ntp
```

2. **ntp Scheduling Policy** と **ntp Scheduling Priority** フィールドを変更します。

```
apiVersion: ntp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ntp_config_name>
  namespace: openshift-ntp
...
spec:
  profile:
    - name: "profile1"
...
ntpSchedulingPolicy: SCHED_FIFO ①
ntpSchedulingPriority: 10 ②
```

- ① **ntp4l** と **phc2sys** プロセスのスケジューリングポリシー。FIFO スケジューリングをサポートするシステムでは、**SCHED_FIFO** を使用してください。
- ② 必須。 **ntp4l** および **phc2sys** プロセスの FIFO 優先度の設定に使用する 1~65 の整数値を設定します。

3. 保存して終了すると、**Ptp Config**CR に変更が適用されます。

検証

1. **Ptp Config**CR が適用された **linuxntp-daemon** Pod と対応するノードの名前を取得します。

```
$ oc get pods -n openshift-ntp -o wide
```

出力例

```
NAME                                READY STATUS RESTARTS AGE IP          NODE
linuxntp-daemon-gmv2n              3/3   Running  0      1d17h 10.1.196.24 compute-0.example.com
linuxntp-daemon-lgm55              3/3   Running  0      1d17h 10.1.196.25 compute-1.example.com
ntp-operator-3r4dcvf7f4-zndk7     1/1   Running  0      1d7h  10.129.0.61 control-plane-1.example.com
```

2. **ntp4l** プロセスが、更新された **chrt**FIFO 優先度で実行されていることを確認します。

```
$ oc -n openshift-ntp logs linuxntp-daemon-lgm55 -c linuxntp-daemon-container|grep chrt
```

出力例

```
l1216 19:24:57.091872 1600715 daemon.go:285] /bin/chrt -f 65 /usr/sbin/ntp4l -f /var/run/ntp4l.0.config -2 --summary_interval -4 -m
```

11.9. 一般的な PTP OPERATOR の問題のトラブルシューティング

以下の手順を実行して、PTP Operator で典型的な問題のトラブルシューティングを行います。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- PTP をサポートするホストを使用して、PTP Operator をベアメタルクラスターにインストールします。

手順

1. Operator およびオペランドが、設定されたノードについてクラスターに正常にデプロイされていることを確認します。

```
$ oc get pods -n openshift-ntp -o wide
```

出力例

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
linuxntp-daemon-lmvgn-0.example.com	3/3	Running	0	4d17h	10.1.196.24	compute-0.example.com
linuxntp-daemon-qhfg7-1.example.com	3/3	Running	0	4d17h	10.1.196.25	compute-1.example.com
ntp-operator-6b8dcbf7f4-zndk7-1.example.com	1/1	Running	0	5d7h	10.129.0.61	control-plane-1.example.com



注記

PTP 高速イベントバスが有効な場合には、準備できた **linuxntp-daemon** Pod の数は **3/3** になります。PTP 高速イベントバスが有効になっていない場合、**2/2** が表示されます。

2. サポートされているハードウェアがクラスターにあることを確認します。

```
$ oc -n openshift-ntp get nodeptpdevices.ptp.openshift.io
```

出力例

NAME	AGE
control-plane-0.example.com	10d
control-plane-1.example.com	10d
compute-0.example.com	10d
compute-1.example.com	10d
compute-2.example.com	10d

3. ノードで利用可能な PTP ネットワークインターフェイスを確認します。

```
$ oc -n openshift-ntp get nodeptpdevices.ptp.openshift.io <node_name> -o yaml
```

ここでは、以下のようになります。

<node_name>

問い合わせるノードを指定します (例: **compute-0.example.com**)。

出力例

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2021-09-14T16:52:33Z"
  generation: 1
  name: compute-0.example.com
  namespace: openshift-ptp
  resourceVersion: "177400"
  uid: 30413db0-4d8d-46da-9bef-737bacd548fd
spec: {}
status:
  devices:
    - name: eno1
    - name: eno2
    - name: eno3
    - name: eno4
    - name: enp5s0f0
    - name: enp5s0f1
```

4. 対応するノードの **linuxptp-daemon** Pod にアクセスし、PTP インターフェイスがプライマリクロックに正常に同期されていることを確認します。
 - a. 以下のコマンドを実行して、**linuxptp-daemon** Pod の名前と、トラブルシューティングに使用するノードを取得します。

```
$ oc get pods -n openshift-ptp -o wide
```

出力例

```
NAME                                READY STATUS RESTARTS AGE IP          NODE
linuxptp-daemon-lmvgn               3/3   Running 0       4d17h 10.1.196.24 compute-0.example.com
linuxptp-daemon-qhfg7               3/3   Running 0       4d17h 10.1.196.25 compute-1.example.com
ptp-operator-6b8dcbf7f4-zndk7       1/1   Running 0       5d7h  10.129.0.61 control-plane-1.example.com
```

- b. リモートシェルが必要な **linuxptp-daemon** コンテナへのリモートシェルです。

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container <linux_daemon_container>
```

ここでは、以下のようになります。

<linux_daemon_container>

診断するコンテナです (例: **linuxptp-daemon-lmvgn**)。

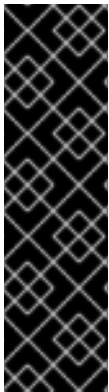
- c. **linuxptp-daemon** コンテナへのリモートシェル接続では、PTP 管理クライアント (**pmc**) ツールを使用して、ネットワークインターフェイスを診断します。以下の **pmc** コマンドを実行して、PTP デバイスの同期ステータスを確認します (例: **ptp4l**)。

```
# pmc -u -f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

ノードがプライマリークロックに正常に同期されたときの出力例

```
sending: GET PORT_DATA_SET
40a6b7.ffe.166ef0-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
portIdentity      40a6b7.ffe.166ef0-1
portState         SLAVE
logMinDelayReqInterval -4
peerMeanPathDelay 0
logAnnounceInterval -3
announceReceiptTimeout 3
logSyncInterval  -4
delayMechanism    1
logMinPdelayReqInterval -4
versionNumber     2
```

11.10. PTP ハードウェアの高速イベント通知フレームワーク



重要

通常のクロックを使用した PTP イベントは、テクノロジープレビューとしてのみ機能します。テクノロジープレビュー機能は、Red Hat の実稼働環境におけるサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

11.10.1. PTP およびクロック同期エラーイベントについて

仮想 RAN などのクラウドネイティブアプリケーションでは、ネットワーク全体の機能に重要なハードウェアタイミングイベントに関する通知へのアクセスが必要です。高速イベント通知は、差し迫ったおよび Real-time Precision Time Protocol (PTP) のクロック同期イベントに関する早期の警告シグナルです。PTP クロック同期エラーは、分散ユニット (DU) で実行している vRAN アプリケーションなど、低レイテンシーアプリケーションのパフォーマンスおよび信頼性に悪影響を及ぼす可能性があります。

PTP 同期の損失は、RAN ネットワークでは重大なエラーです。ノードで同期が失われると、無線がシャットダウンされ、ネットワークの OTA(Over the Air) トラフィックがワイヤレスネットワーク内の別のノードにシフトされる可能性があります。高速のイベント通知は、クラスターノードが DU で実行している vRAN アプリケーションに対して PTP クロック同期ステータスと通信できるようにすることで、ワークロードのエラーを軽減します。

イベント通知は、同じ DU ノードで実行している RAN アプリケーションで利用できます。パブリッシュ/サブスクライブ REST API は、イベント通知をメッセージングバスに渡します。パブリッシュ/サブスクライブメッセージング、または pub/sub メッセージングは、トピックに公開されたメッセージがトピックのすべてのサブスクライバーに即座に受信される、サービス通信アーキテクチャーへの非同期サービスです。

高速イベント通知は、すべての PTP 対応ネットワークインターフェイスについて OpenShift Container Platform の PTP Operator によって生成されます。イベントは、Advanced Message Queuing Protocol (AMQP) メッセージバスで **cloud-event-proxy** サイドカーコンテナを使用して利用可能になります。AMQP メッセージバスは AMQ Interconnect Operator によって提供されます。



注記

PTP 高速イベント通知は、PTP もしくは通常のクロックを使用するように設定されたネットワークインターフェイスでのみ利用できます。

11.10.2. PTP 高速イベント通知フレームワークについて

分散ユニット (DU) アプリケーションを、PTP Operator および **cloud-event-proxy** サイドカーコンテナを使用して、OpenShift Container Platform によって生成される Precision Time Protocol (PTP) 高速イベント通知にサブスクライブできます。**ptpOperatorConfig** カスタムリソース (CR) で **enableEventPublisher** フィールドを **true** に設定し、**transportHost** アドレスを指定することで、**cloud-event-proxy** サイドカーコンテナを有効にします。PTP 高速イベントは、AMQ Interconnect Operator によって提供される Advanced Message Queuing Protocol (AMQP) イベント通知バスを使用します。AMQ Interconnect は Red Hat AMQ のコンポーネントで、AMQP 対応エンドポイント間でメッセージを柔軟にルーティングするメッセージングルーターです。

cloud-event-proxy サイドカーコンテナは、プライマリーアプリケーションのリソースを使用せずに、プライマリー vRAN アプリケーションと同じリソースにアクセスでき、レイテンシーが大きくなくとも構いません。

高速イベント通知フレームワークは通信に REST API を使用し、O-RAN REST API 仕様に基づいています。フレームワークは、パブリッシャーとサブスクライバーアプリケーション間の通信を処理するパブリッシャー、サブスクライバー、および AMQ メッセージングバスで設定されます。**cloud-event-proxy** サイドカーは、DU ノードのメイン DU アプリケーションコンテナにゆるく結合された Pod で実行するユーティリティーコンテナです。これは、DU アプリケーションを公開された PTP イベントにサブスクライブできるようにするイベント公開フレームワークを提供します。

DU アプリケーションはサイドカーパターンで **cloud-event-proxy** コンテナを実行し、PTP イベントにサブスクライブします。以下のワークフローでは、DU アプリケーションが PTP 高速イベントを使用する方法について説明します。

1. **DU アプリケーションはサブスクリプションを要求:** DU は API リクエストを **cloud-event-proxy** サイドカーに送信し、PTP イベントサブスクリプションを作成します。**cloud-event-proxy** サイドカーは、サブスクリプションリソースを作成します。
2. **cloud-event-proxy サイドカーは、サブスクリプションを作成:** イベントリソースは **cloud-event-proxy** サイドカーによって永続化されます。**cloud-event-proxy** サイドカーコンテナは、ID と URL の場所で確認応答を送信し、保存されたサブスクリプションリソースにアクセスします。サイドカーは、サブスクリプションに指定されたリソースの AMQ メッセージングリスナープロトコルを作成します。
3. **DU アプリケーションは PTP イベント通知を受受け取る:** **cloud-event-proxy** サイドカーコンテナは、リソース修飾子で指定されたアドレスをリスンします。DU イベントのコンシューマーはメッセージを処理し、これをサブスクリプションで指定した返信 URL に渡します。
4. **cloud-event-proxy サイドカーは、PTP イベントを検証し、これを DU アプリケーションに送信:** **cloud-event-proxy** サイドカーはイベントを受信し、クラウドイベントオブジェクトをアンラップデータを取得し、イベントを返す URL を取得して DU コンシューマーアプリケーションに返します。

5. **DU アプリケーションは PTP イベントを使用**: DU アプリケーションイベントコンシューマーは PTP イベントを受信して処理します。

11.10.3. AMQ メッセージングバスのインストール

ノードのパブリッシャーとサブスクリバード間で PTP 高速イベント通知を渡すには、ノードでローカルに実行するように AMQ メッセージングバスをインストールおよび設定する必要があります。これは、クラスターで使用するために AMQ Interconnect Operator をインストールして行います。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

- AMQ Interconnect Operator を独自の **amq-interconnect** namespace にインストールします。 [Red Hat Integration - AMQ Interconnect Operator の追加](#) を参照してください。

検証

1. AMQ Interconnect Operator が利用可能で、必要な Pod が実行していることを確認します。

```
$ oc get pods -n amq-interconnect
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
amq-interconnect-645db76c76-k8ghs  1/1   Running 0      23h
interconnect-operator-5cb5fc7cc-4v7qm 1/1   Running 0      23h
```

2. 必要な **linuxptp-daemon** PTP イベントプロデューサー Pod が **openshift-ptp** namespace で実行していることを確認します。

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

出力例

```
NAME                READY STATUS RESTARTS AGE
linuxptp-daemon-2t78p 3/3   Running 0      12h
linuxptp-daemon-k8n88 3/3   Running 0      12h
```

11.10.4. PTP 高速イベント通知パブリッシャーの設定

クラスター内のネットワークインターフェイスの PTP 高速イベント通知の使用を開始するには、PTP Operator **PtpOperatorConfig** カスタムリソース (CR) で高速イベントパブリッシャーを有効にし、作成する **PtpConfig** CR に **ptpClockThreshold** 値を設定する必要があります。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールします。

- **cluster-admin** 権限を持つユーザーとしてログインしている。
- PTP Operator および AMQ Interconnect Operator をインストールします。

手順

1. **PtpOperatorConfig** リソースの **spec.ptpEventConfig** フィールドを変更し、以下のコマンドを実行して適切な値を設定します。

```
$ oc edit PtpOperatorConfig default -n openshift-ptp
```

```
...
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    enableEventPublisher: true ①
    transportHost: amqp://<instance_name>.<namespace>.svc.cluster.local ②
```

- ① **enableEventPublisher** を **true** に設定して、PTP 高速イベント通知を有効にします。
- ② **transportHost** を、設定した AMQ ルーターに設定します。 **<instance_name>** および **<namespace>** は AMQ Interconnect ルーターインスタンス名および namespace に対応します (例: **amqp://amq-interconnect.amq-interconnect.svc.cluster.local**)。

2. PTP 対応インターフェイスの **PtpConfig** カスタムリソースを作成し、**ptpClockThreshold** に必要な値を設定します。以下に例を示します。

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: example-ptpconfig
  namespace: openshift-ptp
spec:
  profile:
    - name: "profile1"
      interface: "enp5s0f0"
      ptp4lOpts: "-2 -s --summary_interval -4" ①
      phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" ②
      ptp4lConf: "" ③
  ptpClockThreshold: ④
    holdOverTimeout: 5
    maxOffsetThreshold: 100
    minOffsetThreshold: -100
```

- ① **--summary_interval -4**を追加して、PTP 高速イベントを使用します。
- ② **phc2sysOpts** の値が必要です。 **-m** はメッセージを **stdout** に出力します。 **linuxptp-daemon DaemonSet** はログを解析し、Prometheus メトリックを生成します。
- ③ デフォルトの **/etc/ptp4l.conf** ファイルを置き換える設定が含まれる文字列を指定します。デフォルト設定を使用するには、フィールドを空のままにします。

- 4 オプション:`ptpClockThreshold` スタンザが存在しない場合は、`ptpClockThreshold` フィールドにデフォルト値が使用されます。スタンザは、デフォルトの

11.10.5. DU アプリケーションを PTP イベントにサブスクライブする RESTAPI リファレンス

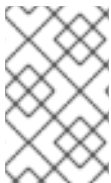
PTP イベント通知 REST API を使用して、分散ユニット (DU) アプリケーションを親ノードで生成される PTP イベントにサブスクライブします。

リソースアドレス/`cluster/node/<node_name>/ptp` を使用して、アプリケーションを PTP イベントにサブスクライブします。ここで、`<node_name>` は、DU アプリケーションを実行しているクラスターノードです。

`cloud-event-consumer` DU アプリケーションコンテナと `cloud-event-proxy` サイドカーコンテナを別々の DU アプリケーション Pod にデプロイします。`cloud-event-consumer` DU アプリケーションは、アプリケーション Pod の `cloud-event-proxy` コンテナにサブスクライブします。

次の API エンドポイントを使用して、`cloud-event-consumer` DU アプリケーションを、DU アプリケーション Pod の `http://localhost:8089/api/cloudNotifications/v1/` にある `cloud-event-proxy` コンテナによって投稿された PTP イベントにサブスクライブします。

- `/api/cloudNotifications/v1/subscriptions`
 - **POST**: 新しいサブスクリプションを作成します。
 - **GET**: サブスクリプションの一覧を取得します。
- `/api/cloudNotifications/v1/subscriptions/<subscription_id>`
 - **GET**: 指定されたサブスクリプション ID の詳細を返します。
- `api/cloudNotifications/v1/subscriptions/status/<subscription_id>`
 - **PUT**: 指定されたサブスクリプション ID に新しいステータス ping 要求を作成します。
- `/api/cloudNotifications/v1/health`
 - **GET**: `cloudNotifications API` の正常性ステータスを返します。



注記

9089 は、アプリケーション Pod にデプロイされた `cloud-event-consumer` コンテナのデフォルトポートです。必要に応じて、DU アプリケーションに別のポートを設定できます。

11.10.5.1. api/cloudNotifications/v1/subscriptions

11.10.5.1.1. HTTP メソッド

GET api/cloudNotifications/v1/subscriptions

11.10.5.1.1.1. 説明

サブスクリプションの一覧を返します。サブスクリプションが存在する場合は、サブスクリプションの一覧とともに **200 OK** のステータスコードが返されます。

API 応答の例

```
[
  {
    "id": "75b1ad8f-c807-4c23-acf5-56f4b7ee3826",
    "endpointUri": "http://localhost:9089/event",
    "uriLocation": "http://localhost:8089/api/cloudNotifications/v1/subscriptions/75b1ad8f-c807-4c23-acf5-56f4b7ee3826",
    "resource": "/cluster/node/compute-1.example.com/ptp"
  }
]
```

11.10.5.1.2. HTTP メソッド

POST api/cloudNotifications/v1/subscriptions

11.10.5.1.2.1. 説明

新しいサブスクリプションを作成します。サブスクリプションが正常に作成されるか、すでに存在する場合は、**201 Created** ステータスコードが返されます。

表11.1 クエリーパラメーター

パラメーター	型
subscription	data

ペイロードの例

```
{
  "uriLocation": "http://localhost:8089/api/cloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

11.10.5.2. api/cloudNotifications/v1/subscriptions/<subscription_id>

11.10.5.2.1. HTTP メソッド

GET api/cloudNotifications/v1/subscriptions/<subscription_id>

11.10.5.2.1.1. 説明

ID が **<subscription_id>** のサブスクリプションの詳細を返します。

表11.2 クエリーパラメーター

パラメーター	型
<subscription_id>	string

API 応答の例

```
{
  "id": "48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "endpointUri": "http://localhost:9089/event",
  "uriLocation": "http://localhost:8089/api/cloudNotifications/v1/subscriptions/48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

11.10.5.3. api/cloudNotifications/v1/subscriptions/status/<subscription_id>

11.10.5.3.1. HTTP メソッド

PUT api/cloudNotifications/v1/subscriptions/status/<subscription_id>

11.10.5.3.1.1. 説明

ID <subscription_id> のサブスクリプションの新規ステータス ping 要求を作成します。サブスクリプションが存在する場合は、ステータスリクエストに成功し、**202 Accepted** ステータスコードが返されます。

表11.3 クエリーパラメーター

パラメーター	型
<subscription_id>	string

API 応答の例

```
{"status": "ping sent"}
```

11.10.5.4. api/cloudNotifications/v1/health/

11.10.5.4.1. HTTP メソッド

GET api/cloudNotifications/v1/health/

11.10.5.4.1.1. 説明

cloudNotifications REST API の正常性ステータスを返します。

API 応答の例

```
OK
```

11.10.6. CLI を使用した PTP 高速イベントメトリクスの監視

oc CLI を使用して、**cloud-event-proxy** コンテナから直接高速イベントバスメトリクスをモニターできます。



注記

PTP 高速イベント通知メトリクスは OpenShift Container Platform Web コンソールでも利用できます。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- PTP Operator をインストールし、設定します。

手順

1. アクティブな **linuxptp-daemon** Pod の一覧を取得します。

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

出力例

```
NAME                READY STATUS  RESTARTS  AGE
linuxptp-daemon-2t78p 3/3   Running  0         8h
linuxptp-daemon-k8n88 3/3   Running  0         8h
```

2. 以下のコマンドを実行して、必要な **cloud-event-proxy** コンテナのメトリクスにアクセスします。

```
$ oc exec -it <linuxptp-daemon> -n openshift-ptp -c cloud-event-proxy -- curl 127.0.0.1:9091/metrics
```

ここでは、以下ようになります。

<linuxptp-daemon>

問い合わせる Pod を指定します (例: **linuxptp-daemon-2t78p**)。

出力例

```
# HELP cne_amqp_events_published Metric to get number of events published by the transport
# TYPE cne_amqp_events_published gauge
cne_amqp_events_published{address="/cluster/node/compute-1.example.com/ptp/status",status="success"} 1041
# HELP cne_amqp_events_received Metric to get number of events received by the transport
# TYPE cne_amqp_events_received gauge
cne_amqp_events_received{address="/cluster/node/compute-1.example.com/ptp",status="success"} 1019
# HELP cne_amqp_receiver Metric to get number of receiver created
# TYPE cne_amqp_receiver gauge
cne_amqp_receiver{address="/cluster/node/mock",status="active"} 1
cne_amqp_receiver{address="/cluster/node/compute-1.example.com/ptp",status="active"} 1
```



```
cne_amqp_receiver{address="/cluster/node/compute-  
1.example.com/redfish/event",status="active"}  
...
```

11.10.7. Web コンソールでの PTP 高速イベントメトリクスの監視

事前に設定された自己更新型の Prometheus モニターリングスタックを使用して、OpenShift Container Platform Web コンソールで PTP 高速イベントメトリクスをモニターリングできます。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下のコマンドを実行して、**cloud-event-proxy** サイドカーコンテナから利用可能な PTP メトリクスの一覧を返します。

```
$ oc exec -it <linuxptp_daemon_pod> -n openshift-ptp -c cloud-event-proxy -- curl  
127.0.0.1:9091/metrics
```

ここでは、以下のようになります。

<linuxptp_daemon_pod>

問い合わせる Pod を指定します (例: **linuxptp-daemon-2t78p**)。

2. 返されるメトリクスの一覧から問い合わせる PTP メトリクスの名前 (例: **cne_amqp_events_received**) をコピーします。
3. OpenShift Container Platform Web コンソールで、**Observe** → **Metrics** をクリックします。
4. PTP メトリクスを **Expression** フィールドに貼り付け、**Run queries** をクリックします。

関連情報

- [メトリックの管理](#)

第12章 ネットワークポリシー

12.1. ネットワークポリシーについて

クラスター管理者は、トラフィックをクラスター内の Pod に制限するネットワークポリシーを定義できます。

12.1.1. ネットワークポリシーについて

Kubernetes ネットワークポリシーをサポートする Kubernetes Container Network Interface (CNI) プラグインを使用するクラスターでは、ネットワークの分離は **NetworkPolicy** オブジェクトによって完全に制御されます。OpenShift Container Platform 4.9 では、OpenShift SDN はデフォルトのネットワーク分離モードでのネットワークポリシーの使用をサポートしています。

注記

OpenShift SDN クラスターネットワークプロバイダーを使用する場合、ネットワークポリシーについて、以下の制限が適用されます。

- **egress** フィールドで指定される egress ネットワークポリシーはサポートされていません。egress ファイアウォールは、OpenShift SDN では egress ネットワークポリシーとしても知られています。これはネットワークポリシーの egress とは異なります。
- IPBlock はネットワークポリシーでサポートされますが、**except** 句はサポートしません。**except** 句を含む IPBlock セクションのあるポリシーを作成する場合、SDN Pod は警告をログに記録し、そのポリシーの IPBlock セクション全体は無視されます。

警告



ネットワークポリシーは、ホストのネットワーク namespace には適用されません。ホストネットワークが有効にされている Pod はネットワークポリシールールによる影響を受けません。

デフォルトで、プロジェクトのすべての Pod は他の Pod およびネットワークのエンドポイントからアクセスできます。プロジェクトで1つ以上の Pod を分離するには、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。プロジェクト管理者は独自のプロジェクト内で **NetworkPolicy** オブジェクトの作成および削除を実行できます。

Pod が1つ以上の **NetworkPolicy** オブジェクトのセレクターで一致する場合、Pod はそれらの1つ以上の **NetworkPolicy** オブジェクトで許可される接続のみを受け入れます。**NetworkPolicy** オブジェクトによって選択されていない Pod は完全にアクセス可能です。

以下のサンプル **NetworkPolicy** オブジェクトは、複数の異なるシナリオをサポートすることを示しています。

- すべてのトラフィックを拒否します。

プロジェクトに deny by default (デフォルトで拒否) を実行させるには、すべての Pod に一致するが、トラフィックを一切許可しない **NetworkPolicy** オブジェクトを追加します。

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

- OpenShift Container Platform Ingress コントローラーからの接続のみを許可します。プロジェクトで OpenShift Container Platform Ingress コントローラーからの接続のみを許可するには、以下の **NetworkPolicy** オブジェクトを追加します。

```
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
```

- プロジェクト内の Pod からの接続のみを受け入れます。Pod が同じプロジェクト内の他の Pod からの接続を受け入れるが、他のプロジェクトの Pod からの接続を拒否するように設定するには、以下の **NetworkPolicy** オブジェクトを追加します。

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

- Pod ラベルに基づいて HTTP および HTTPS トラフィックのみを許可します。特定のラベル (以下の例の **role=frontend**) の付いた Pod への HTTP および HTTPS アクセスのみを有効にするには、以下と同様の **NetworkPolicy** オブジェクトを追加します。

```
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

```

- namespace および Pod セレクターの両方を使用して接続を受け入れます。namespace と Pod セレクターを組み合わせることでネットワークトラフィックのマッチングをするには、以下と同様の **NetworkPolicy** オブジェクトを使用できます。

```

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 オブジェクトは加算されるものです。つまり、複数の **NetworkPolicy** オブジェクトを組み合わせると複雑なネットワーク要件を満たすことができます。

たとえば、先の例で定義された **NetworkPolicy** オブジェクトの場合、同じプロジェクト内に **allow-same-namespace** と **allow-http-and-https** ポリシーの両方を定義することができます。これにより、ラベル **role=frontend** の付いた Pod は各ポリシーで許可されるすべての接続を受け入れます。つまり、同じ namespace の Pod からのすべてのポート、およびすべての namespace の Pod からのポート **80** および **443** での接続を受け入れます。

12.1.2. ネットワークポリシーの最適化

ネットワークポリシーを使用して、namespace 内でラベルで相互に区別される Pod を分離します。



注記

ネットワークポリシールールを効果的に使用するためのガイドラインは、OpenShift SDN クラスタネットワークプロバイダーのみに適用されます。

NetworkPolicy オブジェクトを単一 namespace 内の多数の個別 Pod に適用することは効率的ではありません。Pod ラベルは IP レベルには存在しないため、ネットワークポリシーは、**podSelector** で選択されるすべての Pod 間のすべてのリンクについての別個の Open vSwitch (OVS) フロールールを生成します。

たとえば、仕様の **podSelector** および **NetworkPolicy** オブジェクト内の ingress **podSelector** のそれぞれが 200 Pod に一致する場合、40,000 (200*200) OVS フロールールが生成されます。これにより、ノードの速度が低下する可能性があります。

ネットワークポリシーを設計する場合は、以下のガイドラインを参照してください。

- namespace を使用して分離する必要がある Pod のグループを組み込み、OVS フロールールの数を減らします。
namespace 全体を選択する **NetworkPolicy** オブジェクトは、**namespaceSelectors** または空の **podSelectors** を使用して、namespace の VXLAN 仮想ネットワーク ID に一致する単一の OVS フロールールのみを生成します。
- 分離する必要のない Pod は元の namespace に維持し、分離する必要がある Pod は1つ以上の異なる namespace に移します。
- 追加のターゲット設定された namespace 間のネットワークポリシーを作成し、分離された Pod から許可する必要がある特定のトラフィックを可能にします。

12.1.3. 次のステップ

- [ネットワークポリシーの作成](#)
- オプション: [デフォルトネットワークポリシーの定義](#)

12.1.4. 関連情報

- [プロジェクトおよび namespace](#)
- [マルチテナントネットワークポリシーの設定](#)
- [NetworkPolicy API](#)

12.2. ネットワークポリシーイベントのロギング

クラスター管理者は、クラスターのネットワークポリシー監査ロギングを設定し、1つ以上の namespace のロギングを有効にできます。



注記

ネットワークポリシーの監査ロギングは [OVN-Kubernetes クラスターネットワークプロバイダー](#) でのみ利用可能です。

12.2.1. ネットワークポリシー監査ロギング

OVN-Kubernetes クラスターネットワークプロバイダーは、Open Virtual Network (OVN) ACL を使用してネットワークポリシーを管理します。監査ロギングは ACL イベントの許可および拒否を公開します。

syslog サーバーや UNIX ドメインソケットなどのネットワークポリシー監査ログの宛先を設定できます。追加の設定に関係なく、監査ログは常にクラスター内の各 OVN-Kubernetes Pod の `/var/log/ovn/acl-audit-log.log` に保存されます。

以下の例のように、namespace に `k8s.ovn.org/acl-logging` キーでアノテーションを付けることにより、namespace ごとにネットワークポリシー監査ログを有効にします。

namespace アノテーションの例

```
kind: Namespace
apiVersion: v1
metadata:
  name: example1
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "info",
        "allow": "info"
      }
```

ロギング形式は RFC5424 によって定義される syslog と互換性があります。syslog ファシリティーは設定可能です。デフォルトは **local0** です。ログエントリーの例は、以下のようになります。

ACL 拒否ログエントリーの例

```
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-all",
verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,nw_dst=
10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

以下の表は、namespace アノテーションの値について説明しています。

表12.1 ネットワークポリシー監査ロギング namespace アノテーション

Annotation	値
k8s.ovn.org/acl-logging	<p>namespace のネットワークポリシー監査ロギングを有効にするには、allow、deny、または両方のうち、少なくとも1つを指定する必要があります。</p> <p>deny オプション: alert、warning、notice、info、または debug を指定します。</p> <p>allow オプション: alert、warning、notice、info、または debug を指定します。</p>

12.2.2. ネットワークポリシー監査の設定

監査ロギングの設定は、OVN-Kubernetes クラスターネットワークプロバイダー設定の一部として指定されます。以下の YAML は、ネットワークポリシーの監査ロギング機能のデフォルト値を示しています。

監査ロギング設定

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

```

defaultNetwork:
  ovnKubernetesConfig:
    policyAuditConfig:
      destination: "null"
      maxFileSize: 50
      rateLimit: 20
      syslogFacility: local0

```

以下の表は、ネットワークポリシー監査ロギングの設定フィールドについて説明しています。

表12.2 policyAuditConfig object

フィールド	型	説明
rateLimit	integer	ノードごとに毎秒生成されるメッセージの最大数。デフォルト値は、1秒あたり 20 メッセージです。
maxFileSize	integer	監査ログの最大サイズ (バイト単位)。デフォルト値は 50000000 または 50MB です。
destination	string	以下の追加の監査ログターゲットのいずれかになります。 libc ホスト上の journald プロセスの libc syslog() 関数。 udp:<host>:<port> syslog サーバー。<host>:<port> を syslog サーバーのホストおよびポートに置き換えます。 unix:<file> <file> で指定された Unix ドメインソケットファイル。 null 監査ログを追加のターゲットに送信しないでください。
syslogFacility	string	RFC5424 で定義される kern などの syslog ファシリティ。デフォルト値は local0 です。

12.2.3. クラスターのネットワークポリシー監査の設定

クラスター管理者は、クラスターのネットワークポリシー監査ロギングをカスタマイズできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする。

手順

- ネットワークポリシーの監査ロギングの設定をカスタマイズするには、以下のコマンドを入力します。

```
$ oc edit network.operator.openshift.io/cluster
```

ヒント

または、以下の YAML をカスタマイズして適用することで、監査ロギングを設定できます。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      policyAuditConfig:
        destination: "null"
        maxFileSize: 50
        rateLimit: 20
        syslogFacility: local0
```

検証

1. ネットワークポリシーを使用して namespace を作成するには、次の手順を実行します。
 - a. 検証用の namespace を作成します。

```
$ cat <<EOF | oc create -f -
kind: Namespace
apiVersion: v1
metadata:
  name: verify-audit-logging
annotations:
  k8s.ovn.org/acl-logging: '{ "deny": "alert", "allow": "alert" }'
EOF
```

出力例

```
namespace/verify-audit-logging created
```

- b. 監査ロギングを有効にします。

```
$ oc annotate namespace verify-audit-logging k8s.ovn.org/acl-logging='{ "deny": "alert",
"allow": "alert" }'
```

```
namespace/verify-audit-logging annotated
```

- c. namespace のネットワークポリシーを作成します。

```
$ cat <<EOF | oc create -n verify-audit-logging -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all
spec:
  podSelector:
    matchLabels:
```



```

policyTypes:
- Ingress
- Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-same-namespace
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector: {}
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          namespace: verify-audit-logging
EOF

```

出力例

```

networkpolicy.networking.k8s.io/deny-all created
networkpolicy.networking.k8s.io/allow-from-same-namespace created

```

2. ソーストラフィックの Pod を **default** namespace に作成します。

```

$ cat <<EOF | oc create -n default -f -
apiVersion: v1
kind: Pod
metadata:
  name: client
spec:
  containers:
  - name: client
    image: registry.access.redhat.com/rhel7/rhel-tools
    command: ["/bin/sh", "-c"]
    args:
      ["sleep inf"]
EOF

```

3. **verify-audit-logging** namespace に 2 つの Pod を作成します。

```

$ for name in client server; do
cat <<EOF | oc create -n verify-audit-logging -f -
apiVersion: v1
kind: Pod
metadata:
  name: ${name}
spec:
  containers:
  - name: ${name}

```

```

image: registry.access.redhat.com/rhel7/rhel-tools
command: ["/bin/sh", "-c"]
args:
  ["sleep inf"]
EOF
done

```

出力例

```

pod/client created
pod/server created

```

4. トラフィックを生成し、ネットワークポリシー監査ログエントリを作成するには、以下の手順を実行します。

- a. **verify-audit-logging** namespace で **server** という名前の Pod の IP アドレスを取得します。

```
$ POD_IP=$(oc get pods server -n verify-audit-logging -o jsonpath='{.status.podIP}')
```

- b. **default** の namespace の **client** という名前の Pod の直前のコマンドから IP アドレスに ping し、すべてのパケットがドロップされていることを確認します。

```
$ oc exec -it client -n default -- /bin/ping -c 2 $POD_IP
```

出力例

```

PING 10.128.2.55 (10.128.2.55) 56(84) bytes of data.

--- 10.128.2.55 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 2041ms

```

- c. **verify-audit-logging** namespace の **client** という名前の Pod から **POD_IP** シェル環境変数に保存されている IP アドレスに ping し、すべてのパケットが許可されていることを確認します。

```
$ oc exec -it client -n verify-audit-logging -- /bin/ping -c 2 $POD_IP
```

出力例

```

PING 10.128.0.86 (10.128.0.86) 56(84) bytes of data.
64 bytes from 10.128.0.86: icmp_seq=1 ttl=64 time=2.21 ms
64 bytes from 10.128.0.86: icmp_seq=2 ttl=64 time=0.440 ms

--- 10.128.0.86 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.440/1.329/2.219/0.890 ms

```

5. ネットワークポリシー監査ログの最新エントリを表示します。

```

$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-headers=true | awk '{ print $1 }'); do
  oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
done

```

done

出力例

```
Defaulting container name to ovn-controller.
Use 'oc describe pod/ovnkube-node-hdb8v -n openshift-ovn-kubernetes' to see all of the
containers in this pod.
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:33:12.614Z|00006|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:10.037Z|00007|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
2021-06-13T19:44:11.037Z|00008|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-
logging_allow-from-same-namespace_0", verdict=allow, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:3b,dl_dst=0a:58:0a:80:02:3a,nw_src=10.128.2.59,
nw_dst=10.128.2.58,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

12.2.4. namespace のネットワークポリシー監査ロギングの有効化

クラスター管理者は、namespace のネットワークポリシーの監査ロギングを有効化できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする。

手順

- オプション: namespace のネットワークポリシー監査ロギングを有効にするには、以下のコマンドを入力します。

```
$ oc annotate namespace <namespace> \
  k8s.ovn.org/acl-logging='{ "deny": "alert", "allow": "notice" }'
```

ここでは、以下のようになります。

<namespace>

namespace の名前を指定します。

ヒント

または、以下の YAML を適用して監査ロギングを有効化できます。

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/acl-logging: |-
      {
        "deny": "alert",
        "allow": "notice"
      }
```

出力例

```
namespace/verify-audit-logging annotated
```

検証

- ネットワークポリシー監査ログの最新エントリーを表示します。

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node --no-headers=true | awk '{ print $1 }'); do
  oc exec -it $pod -n openshift-ovn-kubernetes -- tail -4 /var/log/ovn/acl-audit-log.log
done
```

出力例

```
2021-06-13T19:33:11.590Z|00005|acl_log(ovn_pinctrl0)|INFO|name="verify-audit-logging_deny-all", verdict=drop, severity=alert:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:80:02:39,dl_dst=0a:58:0a:80:02:37,nw_src=10.128.2.57,
nw_dst=10.128.2.55,nw_tos=0,nw_ecn=0,nw_ttl=64,icmp_type=8,icmp_code=0
```

12.2.5. namespace のネットワークポリシー監査ロギングの無効化

クラスター管理者は、namespace のネットワークポリシー監査ロギングを無効化できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする。

手順

- namespace のネットワークポリシー監査ロギングを無効にするには、以下のコマンドを入力します。

```
$ oc annotate --overwrite namespace <namespace> k8s.ovn.org/acl-logging={}
```

ここでは、以下のようになります。

<namespace>

namespace の名前を指定します。

ヒント

または、以下の YAML を適用して監査ロギングを無効化できます。

```
kind: Namespace
apiVersion: v1
metadata:
  name: <namespace>
annotations:
  k8s.ovn.org/acl-logging: null
```

出力例

```
namespace/verify-audit-logging annotated
```

12.2.6. 関連情報

- [ネットワークポリシーについて](#)

12.3. ネットワークポリシーの作成

admin ロールを持つユーザーは、namespace のネットワークポリシーを作成できます。

12.3.1. ネットワークポリシーの作成

クラスターの namespace に許可される Ingress または egress ネットワークトラフィックを記述する詳細なルールを定義するには、ネットワークポリシーを作成できます。



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意の namespace でネットワークポリシーを作成できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用する (例: OVN-Kubernetes ネットワークプロバイダー、または **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが適用される namespace で作業している。

手順

1. ポリシールールを作成します。

- a. **<policy_name>.yaml** ファイルを作成します。

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

ここでは、以下のようになります。

<policy_name>

ネットワークポリシーファイル名を指定します。

- b. 作成したばかりのファイルで、以下の例のようなネットワークポリシーを定義します。

すべての namespace のすべての Pod から ingress を拒否します。

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

同じ namespace のすべての Pod から ingress を許可します。

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

2. ネットワークポリシーオブジェクトを作成するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>

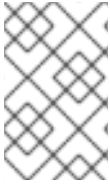
ネットワークポリシーファイル名を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

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



注記

コンソールで **cluster-admin** ロールが割り当てられたユーザーでログインした場合に、YAML ビューから直接、または Web コンソールのフォームから、クラスター内の任意の namespace にネットワークポリシーを作成できます。

12.3.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ❶
spec:
  podSelector: ❷
  matchLabels:
    app: mongodb
  ingress:
  - from:
    - podSelector: ❸
      matchLabels:
        app: app
  ports: ❹
  - protocol: TCP
    port: 27017
```

- ❶ NetworkPolicy オブジェクトの名前。
- ❷ ポリシーが適用される Pod を説明するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
- ❸ ポリシーオブジェクトが入力トラフィックを許可する Pod に一致するセレクター。セレクターは、NetworkPolicy と同じ namespace にある Pod を照合して検索します。
- ❹ トラフィックを受け入れる 1 つ以上の宛先ポートのリスト。

12.3.3. 関連情報

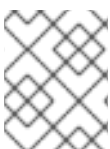
- [Web コンソールへのアクセス](#)

12.4. ネットワークポリシーの表示

admin ロールを持つユーザーは、namespace のネットワークポリシーを表示できます。

12.4.1. ネットワークポリシーの表示

namespace のネットワークポリシーを検査できます。



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意のネットワークポリシーを表示できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

手順

- namespace のネットワークポリシーを一覧表示します。
 - namespace で定義されたネットワークポリシーオブジェクトを表示するには、以下のコマンドを実行します。

```
$ oc get networkpolicy
```

- オプション: 特定のネットワークポリシーを検査するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>

検査するネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

以下に例を示します。

```
$ oc describe networkpolicy allow-same-namespace
```

oc describe コマンドの出力

```
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
```

12.4.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

■


```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ❶
spec:
  podSelector: ❷
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: ❸
      matchLabels:
        app: app
  ports: ❹
  - protocol: TCP
    port: 27017

```

- ❶ NetworkPolicy オブジェクトの名前。
- ❷ ポリシーが適用される Pod を説明するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
- ❸ ポリシーオブジェクトが入力トラフィックを許可する Pod に一致するセレクター。セレクターは、NetworkPolicy と同じ namespace にある Pod を照合して検索します。
- ❹ トラフィックを受け入れる 1 つ以上の宛先ポートのリスト。

12.5. ネットワークポリシーの編集

admin ロールを持つユーザーは、namespace の既存のネットワークポリシーを編集できます。

12.5.1. ネットワークポリシーの編集

namespace のネットワークポリシーを編集できます。



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意の namespace でネットワークポリシーを編集できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用する (例: OVN-Kubernetes ネットワークプロバイダー、または **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

手順

1. オプション: namespace のネットワークポリシーオブジェクトを一覧表示するには、以下のコマンドを入力します。

```
$ oc get networkpolicy
```

ここでは、以下のようになります。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

2. ネットワークポリシーオブジェクトを編集します。

- ネットワークポリシーの定義をファイルに保存した場合は、ファイルを編集して必要な変更を加えてから、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

<policy_file>

ネットワークポリシーを含むファイルの名前を指定します。

- ネットワークポリシーオブジェクトを直接更新する必要がある場合、以下のコマンドを入力できます。

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

ここでは、以下のようになります。

<policy_name>

ネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

3. ネットワークポリシーオブジェクトが更新されていることを確認します。

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

ここでは、以下のようになります。

<policy_name>

ネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

12.5.2. サンプル NetworkPolicy オブジェクト

以下は、サンプル NetworkPolicy オブジェクトにアノテーションを付けます。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-27107 ❶
spec:
  podSelector: ❷
  matchLabels:
    app: mongodb
  ingress:
  - from:
    - podSelector: ❸
      matchLabels:
        app: app
  ports: ❹
  - protocol: TCP
    port: 27017
```

- ❶ NetworkPolicy オブジェクトの名前。
- ❷ ポリシーが適用される Pod を説明するセレクター。ポリシーオブジェクトは NetworkPolicy オブジェクトが定義されるプロジェクトの Pod のみを選択できます。
- ❸ ポリシーオブジェクトが入力トラフィックを許可する Pod に一致するセレクター。セレクターは、NetworkPolicy と同じ namespace にある Pod を照合して検索します。
- ❹ トラフィックを受け入れる 1 つ以上の宛先ポートのリスト。

12.5.3. 関連情報

- [ネットワークポリシーの作成](#)

12.6. ネットワークポリシーの削除

admin ロールを持つユーザーは、namespace からネットワークポリシーを削除できます。

12.6.1. ネットワークポリシーの削除

namespace のネットワークポリシーを削除できます。



注記

cluster-admin ロールを持つユーザーでログインしている場合、クラスター内の任意のネットワークポリシーを削除できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用する (例: OVN-Kubernetes ネットワークプロバイダー、または **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは

OpenShiftSDN のデフォルトです。

- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークポリシーが存在する namespace で作業している。

手順

- ネットワークポリシーオブジェクトを削除するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>

ネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

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

12.7. プロジェクトのデフォルトネットワークポリシーの定義

クラスター管理者は、新規プロジェクトの作成時にネットワークポリシーを自動的に含めるように新規プロジェクトテンプレートを変更できます。新規プロジェクトのカスタマイズされたテンプレートがまだない場合には、まずテンプレートを作成する必要があります。

12.7.1. 新規プロジェクトのテンプレートの変更

クラスター管理者は、デフォルトのプロジェクトテンプレートを変更し、新規プロジェクトをカスタム要件に基づいて作成することができます。

独自のカスタムプロジェクトテンプレートを作成するには、以下を実行します。

手順

1. **cluster-admin** 権限を持つユーザーとしてログインしている。
2. デフォルトのプロジェクトテンプレートを生成します。

```
$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

3. オブジェクトを追加するか、または既存オブジェクトを変更することにより、テキストエディターで生成される **template.yaml** ファイルを変更します。
4. プロジェクトテンプレートは、**openshift-config** namespace に作成される必要があります。変更したテンプレートを読み込みます。

-

```
$ oc create -f template.yaml -n openshift-config
```

5. Web コンソールまたは CLI を使用し、プロジェクト設定リソースを編集します。

- Web コンソールの使用
 - i. **Administration** → **Cluster Settings** ページに移動します。
 - ii. **Configuration** をクリックし、すべての設定リソースを表示します。
 - iii. **Project** のエントリーを見つけ、**Edit YAML** をクリックします。
- CLI の使用
 - i. **project.config.openshift.io/cluster** リソースを編集します。

```
$ oc edit project.config.openshift.io/cluster
```

6. **spec** セクションを、**projectRequestTemplate** および **name** パラメーターを組み込むように更新し、アップロードされたプロジェクトテンプレートの名前を設定します。デフォルト名は **project-request** です。

カスタムプロジェクトテンプレートを含むプロジェクト設定リソース

```
apiVersion: config.openshift.io/v1
kind: Project
metadata:
  ...
spec:
  projectRequestTemplate:
    name: <template_name>
```

7. 変更を保存した後、変更が正常に適用されたことを確認するために、新しいプロジェクトを作成します。

12.7.2. 新規プロジェクトへのネットワークポリシーの追加

クラスター管理者は、ネットワークポリシーを新規プロジェクトのデフォルトテンプレートに追加できます。OpenShift Container Platform は、プロジェクトのテンプレートに指定されたすべての **NetworkPolicy** オブジェクトを自動的に作成します。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするデフォルトの CNI ネットワークプロバイダーを使用している (例: **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする。
- 新規プロジェクトのカスタムデフォルトプロジェクトテンプレートを作成している。

手順

1. 以下のコマンドを実行して、新規プロジェクトのデフォルトテンプレートを編集します。

```
$ oc edit template <project_template> -n openshift-config
```

<project_template> を、クラスターに設定したデフォルトテンプレートの名前に置き換えます。デフォルトのテンプレート名は **project-request** です。

2. テンプレートでは、各 **NetworkPolicy** オブジェクトを要素として **objects** パラメーターに追加します。**objects** パラメーターは、1つ以上のオブジェクトのコレクションを受け入れます。以下の例では、**objects** パラメーターのコレクションにいくつかの **NetworkPolicy** オブジェクトが含まれます。

```
objects:
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-same-namespace
  spec:
    podSelector: {}
    ingress:
      - from:
        - podSelector: {}
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-ingress
  spec:
    ingress:
      - from:
        - namespaceSelector:
            matchLabels:
              network.openshift.io/policy-group: ingress
    podSelector: {}
    policyTypes:
      - Ingress
...

```

3. オプション: 以下のコマンドを実行して、新規プロジェクトを作成し、ネットワークポリシーオブジェクトが正常に作成されることを確認します。

- a. 新規プロジェクトを作成します。

```
$ oc new-project <project> 1
```

1 **<project>** を、作成しているプロジェクトの名前に置き換えます。

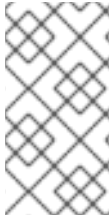
- b. 新規プロジェクトテンプレートのネットワークポリシーオブジェクトが新規プロジェクトに存在することを確認します。

```
$ oc get networkpolicy
NAME                                POD-SELECTOR  AGE
allow-from-openshift-ingress        <none>        7s
allow-from-same-namespace            <none>        7s

```

12.8. ネットワークポリシーを使用したマルチテナント分離の設定

クラスター管理者は、マルチテナントネットワークの分離を実行するようにネットワークポリシーを設定できます。



注記

OpenShift SDN クラスターネットワークプロバイダーを使用している場合、本セクションで説明されているようにネットワークポリシーを設定すると、マルチテナントモードと同様のネットワーク分離が行われますが、ネットワークポリシーモードが設定されません。

12.8.1. ネットワークポリシーを使用したマルチテナント分離の設定

他のプロジェクト namespace の Pod およびサービスから分離できるようにプロジェクトを設定できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用する (例: OVN-Kubernetes ネットワークプロバイダー、または **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

1. 以下の **NetworkPolicy** オブジェクトを作成します。
 - a. **allow-from-openshift-ingress** という名前のポリシー:

```
$ 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
```



注記

policy-group.network.openshift.io/ingress: ""は、OpenShift SDN の推奨の namespace セレクターラベルです。**network.openshift.io/policy-group: ingress** namespace セレクターラベルを使用できますが、これはレガシーラベルです。

- b. **allow-from-openshift-monitoring** という名前のポリシー。

```
$ 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. **allow-same-namespace** という名前のポリシー:

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

2. オプション: 以下のコマンドを実行し、ネットワークポリシーオブジェクトが現在のプロジェクトに存在することを確認します。

```
$ oc describe networkpolicy
```

出力例

```
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
```

12.8.2. 次のステップ

- [デフォルトのネットワークポリシーの定義](#)

12.8.3. 関連情報

- [OpenShift SDN ネットワーク分離モード](#)

第13章 複数ネットワーク

13.1. 複数ネットワークについて

Kubernetes では、コンテナネットワークは Container Network Interface (CNI) を実装するネットワークプラグインに委任されます。

OpenShift Container Platform は、Multus CNI プラグインを使用して CNI プラグインのチェーンを許可します。クラスターのインストール時に、**デフォルト** の Pod ネットワークを設定します。デフォルトのネットワークは、クラスターのすべての通常のネットワークトラフィックを処理します。利用可能な CNI プラグインに基づいて **additional network** を定義し、1つまたは複数のネットワークを Pod に割り当てることができます。必要に応じて、クラスターの複数のネットワークを追加で定義することができます。これは、スイッチまたはルーティングなどのネットワーク機能を提供する Pod を設定する場合に柔軟性を実現します。

13.1.1. 追加ネットワークの使用シナリオ

データプレーンとコントロールプレーンの分離など、ネットワークの分離が必要な状況で追加のネットワークを使用することができます。トラフィックの分離は、以下のようなパフォーマンスおよびセキュリティ関連の理由で必要になります。

パフォーマンス

各プレーンのトラフィック量を管理するために、2つの異なるプレーンにトラフィックを送信できます。

セキュリティ

機密トラフィックは、セキュリティ上の考慮に基づいて管理されているネットワークに送信でき、テナントまたはカスタマー間で共有できないプライベートを分離することができます。

クラスターのすべての Pod はクラスター全体のデフォルトネットワークを依然として使用し、クラスター全体での接続性を維持します。すべての Pod には、クラスター全体の Pod ネットワークに割り当てられる **eth0** インターフェイスがあります。Pod のインターフェイスは、**oc exec -it <pod_name> -- ip a** コマンドを使用して表示できます。Multus CNI を使用するネットワークを追加する場合、それらの名前は **net1**、**net2**、...、**netN** になります。

追加のネットワークを Pod に割り当てるには、インターフェイスの割り当て方法を定義する設定を作成する必要があります。それぞれのインターフェイスは、**NetworkAttachmentDefinition** カスタムリソース (CR) を使用して指定します。これらの CR のそれぞれにある CNI 設定は、インターフェイスの作成方法を定義します。

13.1.2. OpenShift Container Platform の追加ネットワーク

OpenShift Container Platform は、クラスターに追加のネットワークを作成するために使用する以下の CNI プラグインを提供します。

- **bridge**: [ブリッジベースの追加ネットワークを設定する](#) ことで、同じホストにある Pod が相互に、かつホストと通信できます。
- **host-device**: [ホストデバイスの追加ネットワークを設定する](#) ことで、Pod がホストシステム上の物理イーサネットネットワークデバイスにアクセスすることができます。
- **ipvlan**: [ipvlan ベースの追加ネットワークを設定する](#) ことで、macvlan ベースの追加ネットワークと同様に、ホスト上の Pod が他のホストやそれらのホストの Pod と通信できます。macvlan ベースの追加のネットワークとは異なり、各 Pod は親の物理ネットワークインターフェイスと同じ MAC アドレスを共有します。

- **macvlan:** [macvlan ベースの追加ネットワークを作成](#) することで、ホスト上の Pod が物理ネットワークインターフェイスを使用して他のホストやそれらのホストの Pod と通信できます。macvlan ベースの追加ネットワークに割り当てられる各 Pod には固有の MAC アドレスが割り当てられます。
- **SR-IOV:** [SR-IOV ベースの追加ネットワークを設定する](#) ことで、Pod を ホストシステム上の SR-IOV 対応ハードウェアの Virtual Function (VF) インターフェイスに割り当てることができません。

13.2. 追加のネットワークの設定

クラスター管理者は、クラスターの追加のネットワークを設定できます。以下のネットワークタイプに対応しています。

- [ブリッジ](#)
- [ホストデバイス](#)
- [IPVLAN](#)
- [MACVLAN](#)

13.2.1. 追加のネットワークを管理するためのアプローチ

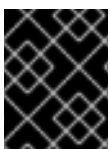
追加したネットワークのライフサイクルを管理するには、2つのアプローチがあります。各アプローチは同時に使用できず、追加のネットワークを管理する場合に1つのアプローチしか使用できません。いずれの方法でも、追加のネットワークは、お客様が設定した Container Network Interface (CNI) プラグインで管理します。

追加ネットワークの場合には、IP アドレスは、追加ネットワークの一部として設定する IPAM(IP Address Management)CNI プラグインでプロビジョニングされます。IPAM プラグインは、DHCP や静的割り当てなど、さまざまな IP アドレス割り当ての方法をサポートしています。

- **Cluster Network Operator (CNO) の設定を変更する:** CNO は自動的に **Network Attachment Definition** オブジェクトを作成し、管理します。CNO は、オブジェクトのライフサイクル管理に加えて、DHCP で割り当てられた IP アドレスを使用する追加のネットワークで確実に DHCP が利用できるようにします。
- **YAML マニフェストを適用する:** **Network Attachment Definition** オブジェクトを作成することで、追加のネットワークを直接管理できます。この方法では、CNI プラグインを連鎖させることができます。

13.2.2. ネットワーク追加割り当ての設定

追加のネットワークは、**k8s.cni.cncf.io**API グループの **Network Attachment Definition**API で設定されます。



重要

Network Attachment Definition オブジェクトには、プロジェクト管理ユーザーがアクセスできるので、機密情報やシークレットを保存しないでください。

API の設定については、以下の表で説明されています。

表13.1 NetworkAttachmentDefinition API フィールド

フィールド	型	説明
<code>metadata.name</code>	<code>string</code>	追加のネットワークの名前です。
<code>metadata.namespace</code>	<code>string</code>	オブジェクトが関連付けられる namespace。
<code>spec.config</code>	<code>string</code>	JSON 形式の CNI プラグイン設定。

13.2.2.1. Cluster Network Operator による追加ネットワークの設定

追加のネットワーク割り当ての設定は、Cluster Network Operator (CNO) の設定の一部として指定します。

以下の YAML は、CNO で追加のネットワークを管理するための設定パラメーターを記述しています。

Cluster Network Operator (CNO) の設定

```

apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
  additionalNetworks: ❶
  - name: <name> ❷
    namespace: <namespace> ❸
    rawCNIConfig: |- ❹
      {
        ...
      }
  type: Raw

```

- ❶ 1つまたは複数の追加ネットワーク設定の配列。
- ❷ 作成している追加ネットワーク割り当ての名前。名前は指定された **namespace** 内で一意である必要があります。
- ❸ ネットワークの割り当てを作成する namespace。値を指定しない場合、**default** の namespace が使用されます。
- ❹ JSON 形式の CNI プラグイン設定。

13.2.2.2. YAML マニフェストからの追加ネットワークの設定

追加ネットワークの設定は、以下の例のように YAML 設定ファイルから指定します。

```

apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: <name> ❶
spec:

```

```
config: |- ②
  {
    ...
  }
```

- ① 作成している追加ネットワーク割り当ての名前。
- ② JSON 形式の CNI プラグイン設定。

13.2.3. 追加のネットワークタイプの設定

次のセクションでは、追加のネットワークの具体的な設定フィールドについて説明します。

13.2.3.1. ブリッジネットワークの追加設定

以下のオブジェクトは、ブリッジ CNI プラグインの設定パラメーターについて説明しています。

表13.2 Bridge CNI プラグイン JSON 設定オブジェクト

フィールド	型	説明
cniVersion	string	CNI 仕様のバージョン。値 0.3.1 が必要です。
name	string	CNO 設定に以前に指定した name パラメーターの値。
type	string	
bridge	string	使用する仮想ブリッジの名前を指定します。ブリッジインターフェイスがホストに存在しない場合は、これが作成されます。デフォルト値は cni0 です。
ipam	object	IPAM CNI プラグインの設定オブジェクト。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。
ipMasq	boolean	仮想ネットワークから外すトラフィックについて IP マスカレードを有効にするには、 true に設定します。すべてのトラフィックのソース IP アドレスは、ブリッジの IP アドレスに書き換えられます。ブリッジに IP アドレスがない場合は、この設定は影響を与えません。デフォルト値は false です。
isGateway	boolean	IP アドレスをブリッジに割り当てるには true に設定します。デフォルト値は false です。
isDefaultGateway	boolean	ブリッジを仮想ネットワークのデフォルトゲートウェイとして設定するには、 true に設定します。デフォルト値は false です。 isDefaultGateway が true に設定される場合、 isGateway も自動的に true に設定されます。

フィールド	型	説明
forceAddress	boolean	仮想ブリッジの事前に割り当てられた IP アドレスの割り当てを許可するには、 true に設定します。 false に設定される場合、重複サブセットの IPv4 アドレスまたは IPv6 アドレスが仮想ブリッジに割り当てられるとエラーが発生します。デフォルト値は false です。
hairpinMode	boolean	仮想ブリッジが受信時に使用した仮想ポートでイーサネットフレームを送信できるようにするには、 true に設定します。このモードは、 Reflective Relay (リフレクティブリレー) としても知られています。デフォルト値は false です。
promiscMode	boolean	ブリッジで無作為検出モード (Promiscuous Mode) を有効にするには、 true に設定します。デフォルト値は false です。
vlan	string	仮想 LAN (VLAN) タグを整数値として指定します。デフォルトで、VLAN タグは割り当てません。
mtu	string	最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。

13.2.3.1.1. ブリッジ設定の例

以下の例では、**bridge-net** という名前の追加のネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
  }
}
```

13.2.3.2. ホストデバイスの追加ネットワークの設定



注記

device、**hwaddr**、**kernelpath**、または **pciBusID** のいずれかのパラメーターを設定してネットワークデバイスを指定します。

以下のオブジェクトは、ホストデバイス CNI プラグインの設定パラメーターについて説明しています。

表13.3 ホストデバイス CNI プラグイン JSON 設定オブジェクト

フィールド	型	説明
cniVersion	string	CNI 仕様のバージョン。値 0.3.1 が必要です。
name	string	CNO 設定に以前に指定した name パラメーターの値。
type	string	設定する CNI プラグインの名前: host-device
device	string	オプション: eth0 などのデバイスの名前。
hwaddr	string	オプション: デバイスハードウェアの MAC アドレス。
kernelpath	string	オプション: /sys/devices/pci0000:00/0000:00:1f.6 などの Linux カーネルデバイス。
pciBusID	string	オプション: 0000:00:1f.6 などのネットワークデバイスの PCI アドレスを指定します。

13.2.3.2.1. ホストデバイス設定例

以下の例では、**hostdev-net** という名前の追加のネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "host-device",
  "device": "eth1"
}
```

13.2.3.3. IPVLAN 追加ネットワークの設定

以下のオブジェクトは、IPVLAN CNI プラグインの設定パラメーターについて説明しています。

表13.4 IPVLAN CNI プラグイン JSON 設定オブジェクト

フィールド	型	説明
cniVersion	string	CNI 仕様のバージョン。値 0.3.1 が必要です。
name	string	CNO 設定に以前に指定した name パラメーターの値。
type	string	設定する CNI プラグインの名前: ipvlan 。
mode	string	仮想ネットワークの操作モード。この値は、 I2 、 I3 、または I3s である必要があります。デフォルト値は I2 です。
master	string	ネットワーク割り当てに関連付けるイーサネットインターフェイス。 master が指定されない場合、デフォルトのネットワークルートのインターフェイスが使用されます。

フィールド	型	説明
mtu	integer	最大転送単位 (MTU) を指定された値に設定します。デフォルト値はカーネルによって自動的に設定されます。
ipam	object	IPAM CNI プラグインの設定オブジェクト。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。 dhcp は指定しないでください。IPVLAN インターフェイスは MAC アドレスをホストインターフェイスと共有するため、IPVLAN の DHCP 設定はサポートされていません。

13.2.3.3.1. IPVLAN 設定例

以下の例では、**ipvlan-net** という名前の追加のネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "work-network",
  "type": "ipvlan",
  "master": "eth1",
  "mode": "l3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

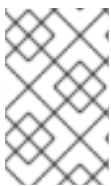
13.2.3.4. MACVLAN 追加ネットワークの設定

以下のオブジェクトは、macvlan CNI プラグインの設定パラメーターについて説明しています。

表13.5 MACVLAN CNI プラグイン JSON 設定オブジェクト

フィールド	型	説明
cniVersion	string	CNI 仕様のバージョン。値 0.3.1 が必要です。
name	string	CNO 設定に以前に指定した name パラメーターの値。
type	string	設定する CNI プラグインの名前: macvlan 。
mode	string	仮想ネットワークのトラフィックの可視性を設定します。 bridge 、 passthru 、 private 、または vepa のいずれかである必要があります。値が指定されない場合、デフォルト値は bridge になります。

フィールド	型	説明
master	string	新しく作成された macvlan インターフェイスに関連付けるホストネットワークインターフェイス。値が指定されていない場合は、デフォルトのルートインターフェイスが使用されます。
mtu	string	最大転送単位 (MTU) を指定された値。デフォルト値はカーネルによって自動的に設定されます。
ipam	object	IPAM CNI プラグインの設定オブジェクト。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。



注記

プラグイン設定の **master** キーを指定する場合は、競合の可能性を回避するために、プライマリーネットワークプラグインに関連付けられているものとは異なる物理ネットワークインターフェイスを使用してください。

13.2.3.4.1. macvlan 設定の例

以下の例では、**macvlan-net** という名前の追加のネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
  }
}
```

13.2.4. 追加ネットワークの IP アドレス割り当ての設定

IPAM (IP アドレス管理) Container Network Interface (CNI) プラグインは、他の CNI プラグインの IP アドレスを提供します。

以下の IP アドレスの割り当てタイプを使用できます。

- 静的割り当て。
- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

13.2.4.1. 静的 IP アドレス割り当ての設定

以下の表は、静的 IP アドレスの割り当ての設定について説明しています。

表13.6 ipam 静的設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 static が必要です。
addresses	array	仮想インターフェイスに割り当てる IP アドレスを指定するオブジェクトの配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
routes	array	Pod 内で設定するルートを指定するオブジェクトの配列です。
dns	array	オプション: DNS の設定を指定するオブジェクトの配列です。

addressesの配列には、以下のフィールドのあるオブジェクトが必要です。

表13.7 ipam.addresses[] 配列

フィールド	型	説明
address	string	指定する IP アドレスおよびネットワーク接頭辞。たとえば、 10.10.21.10/24 を指定すると、追加のネットワークに IP アドレスの 10.10.21.10 が割り当てられ、ネットマスクは 255.255.255.0 になります。
gateway	string	egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。

表13.8 ipam.routes[] 配列

フィールド	型	説明
dst	string	CIDR 形式の IP アドレス範囲 (192.168.17.0/24 、またはデフォルトルートの 0.0.0.0/0)。
gw	string	ネットワークトラフィックがルーティングされるゲートウェイ。

表13.9 ipam.dns オブジェクト

フィールド	型	説明
nameservers	array	DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
domain	array	ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが example.com に設定されている場合、 example-host の DNS ルックアップクエリーは example-host.example.com として書き換えられます。

フィールド	型	説明
search	array	DNS ルックアップのクエリ時に非修飾ホスト名に追加されるドメイン名の配列 (例: example-host)。

静的 IP アドレス割り当ての設定例

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

13.2.4.2. 動的 IP アドレス (DHCP) 割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
      # ...
```

表13.10 ipam DHCP 設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 dhcp が必要です。

動的 IP アドレス (DHCP) 割り当ての設定例

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

13.2.4.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の表は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

表13.11 ipamwhereabouts 設定オブジェクト

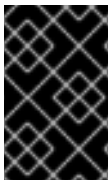
フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 whereabouts が必要です。
range	string	IP アドレスと範囲を CIDR 表記。IP アドレスは、この範囲内のアドレスから割り当てられます。
exclude	array	オプション: CIDR 表記の IP アドレスと範囲 (0 個以上) の一覧。除外されたアドレス範囲内の IP アドレスは割り当てられません。

Whereabouts を使用する動的 IP アドレス割り当ての設定例

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

13.2.5. Cluster Network Operator による追加ネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。



重要

Cluster Network Operator が管理する **NetworkAttachmentDefinition** オブジェクトは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. CNO 設定を編集するには、以下のコマンドを入力します。

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

2. 以下のサンプル CR のように、作成される追加ネットワークの設定を追加して、作成している CR を変更します。

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

```

kind: Network
metadata:
  name: cluster
spec:
  # ...
  additionalNetworks:
  - name: tertiary-net
    namespace: project2
    type: Raw
    rawCNIConfig: |-
      {
        "cniVersion": "0.3.1",
        "name": "tertiary-net",
        "type": "ipvlan",
        "master": "eth1",
        "mode": "l2",
        "ipam": {
          "type": "static",
          "addresses": [
            {
              "address": "192.168.1.23/24"
            }
          ]
        }
      }

```

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。

検証

- 以下のコマンドを実行して、CNO が NetworkAttachmentDefinition オブジェクトを作成していることを確認します。CNO がオブジェクトを作成するまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions -n <namespace>
```

ここでは、以下のようになります。

<namespace>

CNO の設定に追加したネットワーク割り当ての namespace を指定します。

出力例

```

NAME           AGE
test-network-1 14m

```

13.2.6. YAML マニフェストを適用した追加のネットワーク割り当ての作成

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の例のように、追加のネットワーク設定を含む YAML ファイルを作成します。

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: next-net
spec:
  config: |-
    {
      "cniVersion": "0.3.1",
      "name": "work-network",
      "type": "host-device",
      "device": "eth1",
      "ipam": {
        "type": "dhcp"
      }
    }
  }
```

2. 追加のネットワークを作成するには、次のコマンドを入力します。

```
$ oc apply -f <file>.yaml
```

ここでは、以下のようになります。

<file>

YAML マニフェストを含むファイルの名前を指定します。

13.3. 仮想ルーティングおよび転送について

13.3.1. 仮想ルーティングおよび転送について

VRF (Virtual Routing and Forwarding) デバイスは IP ルールとの組み合わせにより、仮想ルーティングと転送ドメインを作成する機能を提供します。VRF は、CNF で必要なパーミッションの数を減らし、セカンダリーネットワークのネットワークトポロジーの可視性を強化します。VRF はマルチテナンシー機能を提供するために使用されます。たとえば、この場合、各テナントには固有のルーティングテーブルがあり、異なるデフォルトゲートウェイが必要です。

プロセスは、ソケットを VRF デバイスにバインドできます。バインドされたソケット経由の packets は、VRF デバイスに関連付けられたルーティングテーブルを使用します。VRF の重要な機能として、これは OSI モデルレイヤー 3 以上にのみ影響を与えるため、LLDP などの L2 ツールは影響を受けません。これにより、ポリシーベースのルーティングなどの優先度の高い IP ルールが、特定のトラフィックを転送する VRF デバイスルールよりも優先されます。

13.3.1.1. Telecommunications Operator についての Pod のセカンダリーネットワークの利点

通信のユースケースでは、各 CNF が同じアドレス空間を共有する複数の異なるネットワークに接続される可能性があります。これらのセカンダリーネットワークは、クラスターのメインネットワーク CIDR と競合する可能性があります。CNI VRF プラグインを使用すると、ネットワーク機能は、同じ IP アドレスを使用して異なるユーザーのインフラストラクチャーに接続でき、複数の異なるお客様の分離された状態を維持します。IP アドレスは OpenShift Container Platform の IP スペースと重複します。CNI VRF プラグインは、CNF で必要なパーミッションの数も減らし、セカンダリーネットワークのネットワークトポロジーの可視性を高めます。

13.4. マルチネットワークポリシーの設定

クラスター管理者は、追加のネットワークのネットワークポリシーを設定できます。



注記

macvlan の追加ネットワークのみに対して、マルチネットワークポリシーを指定することができます。ipvlan などの他の追加のネットワークタイプはサポートされていません。

13.4.1. マルチネットワークポリシーとネットワークポリシーの違い

MultiNetworkPolicy API は、**NetworkPolicy** API を実装していますが、いくつかの重要な違いがあります。

- 以下の場合には、**MultiNetworkPolicy** API を使用する必要があります。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
```

- CLI を使用してマルチネットワークポリシーと対話する場合は、**multi-networkpolicy** リソース名を使用する必要があります。たとえば、**oc get multi-networkpolicy <name>** コマンドを使用してマルチネットワークポリシーオブジェクトを表示できます。ここで、**<name>** はマルチネットワークポリシーの名前になります。
- macvlan 追加ネットワークを定義するネットワーク接続定義の名前でアノテーションを指定する必要があります。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
```

ここでは、以下のようになります。

<network_name>

ネットワーク接続定義の名前を指定します。

13.4.2. クラスターのマルチネットワークポリシーの有効化

クラスター管理者は、クラスターでマルチネットワークポリシーのサポートを有効にすることができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインする。

手順

1. 以下の YAML で **multinetwork-enable-patch.yaml** ファイルを作成します。


```

apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  useMultiNetworkPolicy: true

```

- マルチネットワークポリシーを有効にするようにクラスターを設定します。

```

$ oc patch network.operator.openshift.io cluster --type=merge --patch-file=multinetwork-
enable-patch.yaml

```

出力例

```

network.operator.openshift.io/cluster patched

```

13.4.3. マルチネットワークポリシーの使用

クラスター管理者は、マルチネットワークポリシーを作成、編集、表示、および削除することができます。

13.4.3.1. 前提条件

- クラスターのマルチネットワークポリシーサポートを有効にしている。

13.4.3.2. マルチネットワークポリシーの作成

マルチネットワークポリシーを作成し、クラスターの namespace に許可される Ingress または egress ネットワークトラフィックを記述する詳細なルールを定義することができます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用する (例: OVN-Kubernetes ネットワークプロバイダー、または **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- cluster-admin** 権限を持つユーザーとしてクラスターにログインしていること。
- マルチネットワークポリシーが適用される namespace で作業していること。

手順

- ポリシールールを作成します。
 - <policy_name>.yaml** ファイルを作成します。

```

$ touch <policy_name>.yaml

```

ここでは、以下のようになります。

<policy_name>

マルチネットワークポリシーのファイル名を指定します。

- b. 作成したばかりのファイルで、以下の例のようなマルチネットワークポリシーを定義します。

すべての namespace のすべての Pod から ingress を拒否します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    ingress: []
```

ここでは、以下のようになります。

<network_name>

ネットワーク接続定義の名前を指定します。

同じ namespace のすべての Pod から ingress を許可します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    ingress:
      - from:
        - podSelector: {}
```

ここでは、以下のようになります。

<network_name>

ネットワーク接続定義の名前を指定します。

2. マルチネットワークポリシーオブジェクトを作成するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>

マルチネットワークポリシーのファイル名を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

```
multinetworkpolicy.k8s.cni.cncf.io/default-deny created
```



注記

コンソールで **cluster-admin** ロールが割り当てられたユーザーでログインした場合に、YAML ビューから直接、または Web コンソールのフォームから、クラスター内の任意の namespace にネットワークポリシーを作成できます。

13.4.3.3. マルチネットワークポリシーの編集

namespace のマルチネットワークポリシーを編集できます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするクラスターネットワークプロバイダーを使用する (例: OVN-Kubernetes ネットワークプロバイダー、または **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていること。
- マルチネットワークポリシーが存在する namespace で作業している。

手順

1. オプション: namespace のマルチネットワークポリシーオブジェクトを一覧表示するには、以下のコマンドを入力します。

```
$ oc get multi-networkpolicy
```

ここでは、以下ようになります。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

2. マルチネットワークポリシーオブジェクトを編集します。

- マルチネットワークポリシーの定義をファイルに保存した場合は、ファイルを編集して必要な変更を加えてから、以下のコマンドを入力します。

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

ここでは、以下ようになります。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

<policy_file>

ネットワークポリシーを含むファイルの名前を指定します。

- マルチネットワークポリシーオブジェクトを直接更新する必要がある場合、以下のコマンドを入力できます。

```
$ oc edit multi-networkpolicy <policy_name> -n <namespace>
```

ここでは、以下のようになります。

<policy_name>

ネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

3. マルチネットワークポリシーオブジェクトが更新されていることを確認します。

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

ここでは、以下のようになります。

<policy_name>

マルチネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

13.4.3.4. マルチネットワークポリシーの表示

namespace のマルチネットワークポリシーを検査できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていること。
- マルチネットワークポリシーが存在する namespace で作業している。

手順

- namespace のマルチネットワークポリシーを一覧表示します。
 - namespace で定義されたマルチネットワークポリシーオブジェクトを表示するには、以下のコマンドを実行します。

```
$ oc get multi-networkpolicy
```

- オプション: 特定のマルチネットワークポリシーを検査するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>

検査するマルチネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

13.4.3.5. マルチネットワークポリシーの削除

namespace のマルチネットワークポリシーを削除できます。

前提条件

- クラスタは、**NetworkPolicy** オブジェクトをサポートするクラスタネットワークプロバイダーを使用する (例: OVN-Kubernetes ネットワークプロバイダー、または **mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプロバイダー)。このモードは OpenShiftSDN のデフォルトです。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスタにログインしていること。
- マルチネットワークポリシーが存在する namespace で作業している。

手順

- マルチネットワークポリシーオブジェクトを削除するには、以下のコマンドを入力します。

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

ここでは、以下のようになります。

<policy_name>

マルチネットワークポリシーの名前を指定します。

<namespace>

オプション: オブジェクトが現在の namespace 以外の namespace に定義されている場合は namespace を指定します。

出力例

```
multinetworkpolicy.k8s.cni.cncf.io/default-deny deleted
```

13.4.4. 関連情報

- [ネットワークポリシーについて](#)

- [複数ネットワークについて](#)
- [macvlan ネットワークの設定](#)

13.5. POD の追加のネットワークへの割り当て

クラスターユーザーとして、Pod を追加のネットワークに割り当てることができます。

13.5.1. Pod の追加ネットワークへの追加

Pod を追加のネットワークに追加できます。Pod は、デフォルトネットワークで通常のクラスター関連のネットワークトラフィックを継続的に送信します。

Pod が作成されると、追加のネットワークが割り当てられます。ただし、Pod がすでに存在する場合は、追加のネットワークをこれに割り当ててはできません。

Pod が追加ネットワークと同じ namespace にあること。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターにログインする。

手順

1. アノテーションを **Pod** オブジェクトに追加します。以下のアノテーション形式のいずれかのみを使用できます。
 - a. カスタマイズせずに追加ネットワークを割り当てするには、以下の形式でアノテーションを追加します。**<network>** を、Pod に関連付ける追加ネットワークの名前に置き換えます。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

- 1** 複数の追加ネットワークを指定するには、各ネットワークをコンマで区切ります。コンマの間にはスペースを入れないでください。同じ追加ネットワークを複数回指定した場合、Pod は複数のネットワークインターフェイスをそのネットワークに割り当てます。

- b. カスタマイズして追加のネットワークを割り当てするには、以下の形式でアノテーションを追加します。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
    [
      {
        "name": "<network>", 1
        "namespace": "<namespace>", 2
```

```

    "default-route": ["<default-route>"] ❸
  }
]

```

- ❶ **NetworkAttachmentDefinition** オブジェクトによって定義される追加のネットワークの名前を指定します。
- ❷ **NetworkAttachmentDefinition** オブジェクトが定義される namespace を指定します。
- ❸ オプション: **192.168.17.1** などのデフォルトルートのオーバーライドを指定します。

2. Pod を作成するには、以下のコマンドを入力します。<name> を Pod の名前に置き換えます。

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

3. オプション: アノテーションが **Pod** CR に存在することを確認するには、<name> を Pod の名前に置き換えて、以下のコマンドを入力します。

```
$ oc get pod <name> -o yaml
```

以下の例では、**example-pod** Pod が追加ネットワークの **net1** に割り当てられています。

```

$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status: |- ❶
      [{
        "name": "openshift-sdn",
        "interface": "eth0",
        "ips": [
          "10.128.2.14"
        ],
        "default": true,
        "dns": {}
      },{
        "name": "macvlan-bridge",
        "interface": "net1",
        "ips": [
          "20.2.2.100"
        ],
        "mac": "22:2f:60:a5:f8:00",
        "dns": {}
      }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...

```

- 1 **k8s.v1.cni.cncf.io/networks-status** パラメーターは、オブジェクトの JSON 配列です。各オブジェクトは、Pod に割り当てられる追加のネットワークのステータスについて説明

13.5.1.1. Pod 固有のアドレスおよびルーティングオプションの指定

Pod を追加のネットワークに割り当てる場合、特定の Pod でそのネットワークに関するその他のプロパティを指定する必要がある場合があります。これにより、ルーティングの一部を変更することができ、静的 IP アドレスおよび MAC アドレスを指定できます。これを実行するには、JSON 形式のアノテーションを使用できます。

前提条件

- Pod が追加ネットワークと同じ namespace にあること。
- OpenShift CLI (**oc**) がインストールされている。
- クラスタにログインすること。

手順

アドレスおよび/またはルーティングオプションを指定する間に Pod を追加のネットワークに追加するには、以下の手順を実行します。

1. **Pod** リソース定義を編集します。既存の **Pod** リソースを編集する場合は、以下のコマンドを実行してデフォルトエディターでその定義を編集します。**<name>** を、編集する **Pod** リソースの名前に置き換えます。

```
$ oc edit pod <name>
```

2. **Pod** リソース定義で、**k8s.v1.cni.cncf.io/networks** パラメーターを Pod の **metadata** マッピングに追加します。**k8s.v1.cni.cncf.io/networks** は、追加のプロパティを指定するだけでなく、**NetworkAttachmentDefinition** カスタムリソース (CR) 名を参照するオブジェクト一覧の JSON 文字列を受け入れます。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: ' [<network>[,<network>,...] ' 1
```

- 1 **<network>** を、以下の例にあるように JSON オブジェクトに置き換えます。一重引用符が必要です。

3. 以下の例では、アノテーションで **default-route** パラメーターを使用して、デフォルトルートを持つネットワーク割り当てを指定します。

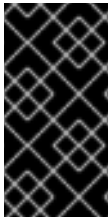
```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '
  {
    "name": "net1"
  },
```



```
{
  "name": "net2", ❶
  "default-route": ["192.0.2.1"] ❷
}'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

- ❶ **name** キーは、Pod に関連付ける追加ネットワークの名前です。
- ❷ **default-route** キーは、ルーティングテーブルに他のルーティングテーブルがない場合に、ルーティングされるトラフィックに使用されるゲートウェイ値を指定します。複数の **default-route** キーを指定すると、Pod がアクティブでなくなります。

デフォルトのルートにより、他のルートに指定されていないトラフィックがゲートウェイにルーティングされます。

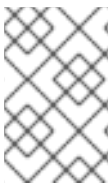


重要

OpenShift Container Platform のデフォルトのネットワークインターフェイス以外のインターフェイスへのデフォルトのルートを設定すると、Pod 間のトラフィックについて予想されるトラフィックが別のインターフェイスでルーティングされる可能性があります。

Pod のルーティングプロパティを確認する場合、**oc** コマンドを Pod 内で **ip** コマンドを実行するために使用できます。

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



注記

また、Pod の k8s.v1.cni.cncf.io/networks-status を参照して、JSON 形式の一覧のオブジェクトで **default-route** キーの有無を確認し、デフォルトルートが割り当てられている追加ネットワークを確認することができます。

Pod に静的 IP アドレスまたは MAC アドレスを設定するには、JSON 形式のアノテーションを使用できます。これには、この機能をとくに許可するネットワークを作成する必要があります。これは、CNO の `rawCNICConfig` で指定できます。

1. 以下のコマンドを実行して CNO CR を編集します。

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

以下の YAML は、CNO の設定パラメーターについて説明しています。

Cluster Network Operator YAML の設定

```
name: <name> ❶
namespace: <namespace> ❷
rawCNICConfig: '{ ❸
```

```
...
}'
type: Raw
```

- 1 作成している追加ネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
- 2 ネットワークの割り当てを作成する namespace を指定します。値を指定しない場合、**default** の namespace が使用されます。
- 3 以下のテンプレートに基づく CNI プラグイン設定を JSON 形式で指定します。

以下のオブジェクトは、macvlan CNI プラグインを使用して静的 MAC アドレスと IP アドレスを使用するための設定パラメーターについて説明しています。

静的 IP および MAC アドレスを使用した macvlan CNI プラグイン JSON 設定オブジェクト

```
{
  "cniVersion": "0.3.1",
  "name": "<name>", 1
  "plugins": [{ 2
    "type": "macvlan",
    "capabilities": { "ips": true }, 3
    "master": "eth0", 4
    "mode": "bridge",
    "ipam": {
      "type": "static"
    }
  }, {
    "capabilities": { "mac": true }, 5
    "type": "tuning"
  }
  ]
}
```

- 1 作成する追加のネットワーク割り当ての名前を指定します。名前は指定された **namespace** 内で一意である必要があります。
- 2 CNI プラグイン設定の配列を指定します。1つ目のオブジェクトは、macvlan プラグイン設定を指定し、2つ目のオブジェクトはチューニングプラグイン設定を指定します。
- 3 CNI プラグインのランタイム設定機能の静的 IP 機能を有効にするために要求が実行されるように指定します。
- 4 macvlan プラグインが使用するインターフェイスを指定します。
- 5 CNI プラグインの静的 MAC アドレス機能を有効にするために要求が実行されるように指定します。

上記のネットワーク割り当ては、特定の Pod に割り当てられる静的 IP アドレスと MAC アドレスを指定するキーと共に、JSON 形式のアノテーションで参照できます。

以下を使用して Pod を編集します。

```
$ oc edit pod <name>
```

静的 IP および MAC アドレスを使用した macvlan CNI プラグイン JSON 設定オブジェクト

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
  {
    "name": "<name>", ①
    "ips": [ "192.0.2.205/24" ], ②
    "mac": "CA:FE:C0:FF:EE:00" ③
  }
]'
```

- ① 上記の **rawCNICConfig** を作成する際に、指定されるように **<name>** を使用します。
- ② サブネットマスクを含む IP アドレスを指定します。
- ③ MAC アドレスを指定します。



注記

静的 IP アドレスおよび MAC アドレスを同時に使用することはできません。これらは個別に使用することも、一緒に使用することもできます。

追加のネットワークを持つ Pod の IP アドレスと MAC プロパティを検証するには、**oc** コマンドを使用して Pod 内で **ip** コマンドを実行します。

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

13.6. 追加ネットワークからの POD の削除

クラスターユーザーとして、追加のネットワークから Pod を削除できます。

13.6.1. 追加ネットワークからの Pod の削除

Pod を削除するだけで、追加のネットワークから Pod を削除できます。

前提条件

- 追加のネットワークが Pod に割り当てられている。
- OpenShift CLI (**oc**) がインストールされている。
- クラスターにログインする。

手順

- Pod を削除するには、以下のコマンドを入力します。

```
$ oc delete pod <name> -n <namespace>
```

- <name> は Pod の名前です。
- <namespace> は Pod が含まれる namespace です。

13.7. 追加ネットワークの編集

クラスター管理者は、既存の追加ネットワークの設定を変更することができます。

13.7.1. 追加ネットワーク割り当て定義の変更

クラスター管理者は、既存の追加ネットワークに変更を加えることができます。追加ネットワークに割り当てられる既存の Pod は更新されません。

前提条件

- クラスター用に追加のネットワークを設定している。
- OpenShift CLI (**oc**) がインストールされている。
- cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターの追加ネットワークを編集するには、以下の手順を実行します。

- 以下のコマンドを実行し、デフォルトのテキストエディターで Cluster Network Operator (CNO) CR を編集します。

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

- additionalNetworks** コレクションで、追加ネットワークを変更内容で更新します。
- 変更を保存し、テキストエディターを終了して、変更をコミットします。
- オプション: 以下のコマンドを実行して、CNO が **NetworkAttachmentDefinition** オブジェクトを更新していることを確認します。<network-name> を表示する追加ネットワークの名前に置き換えます。CNO が **NetworkAttachmentDefinition** オブジェクトを更新して変更内容が反映されるまでに遅延が生じる可能性があります。

```
$ oc get network-attachment-definitions <network-name> -o yaml
```

たとえば、以下のコンソールの出力は **net1** という名前の **NetworkAttachmentDefinition** オブジェクトを表示します。

```
$ oc get network-attachment-definitions net1 -o go-template='{{printf "%s\n" .spec.config}}'
{ "cniVersion": "0.3.1", "type": "macvlan",
  "master": "ens5",
  "mode": "bridge",
  "ipam":   {"type":"static","routes":[{"dst":"0.0.0.0/0","gw":"10.128.2.1"}],"addresses":
```

```
[{"address":"10.128.2.100/23","gateway":"10.128.2.1"}], "dns":{"nameservers":
["172.30.0.10"],"domain":"us-west-2.compute.internal","search":["us-west-
2.compute.internal"]}] }
```

13.8. 追加ネットワークの削除

クラスター管理者は、追加のネットワーク割り当てを削除できます。

13.8.1. 追加ネットワーク割り当て定義の削除

クラスター管理者は、追加ネットワークを OpenShift Container Platform クラスターから削除できます。追加ネットワークは、割り当てられている Pod から削除されません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

クラスターから追加ネットワークを削除するには、以下の手順を実行します。

1. 以下のコマンドを実行して、デフォルトのテキストエディターで Cluster Network Operator (CNO) を編集します。

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

2. 削除しているネットワーク割り当て定義の **additionalNetworks** コレクションから設定を削除し、CR を変更します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks: [] 1
```

- 1** **additionalNetworks** コレクションの追加ネットワーク割り当てのみの設定マッピングを削除する場合、空のコレクションを指定する必要があります。

3. 変更を保存し、テキストエディターを終了して、変更をコミットします。
4. オプション: 以下のコマンドを実行して、追加ネットワーク CR が削除されていることを確認します。

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

13.9. VRF へのセカンダリーネットワークの割り当て

13.9.1. VRF へのセカンダリーネットワークの割り当て

クラスター管理者は、CNI VRF プラグインを使用して、VRF ドメインの追加のネットワークを設定できます。このプラグインにより作成される仮想ネットワークは、指定する物理インターフェイスに関連付けられます。



注記

VRF を使用するアプリケーションを特定のデバイスにバインドする必要があります。一般的な使用方法として、ソケットに **SO_BINDTODEVICE** オプションを使用できます。**SO_BINDTODEVICE** は、渡されるインターフェイス名で指定されているデバイスにソケットをバインドします (例: **eth1**)。 **SO_BINDTODEVICE** を使用するには、アプリケーションに **CAP_NET_RAW** 機能がある必要があります。

ip vrf exec コマンドを使用した VRF の使用は、OpenShift Container Platform Pod ではサポートされません。VRF を使用するには、アプリケーションを VRF インターフェイスに直接バインドします。

13.9.1.1. CNI VRF プラグインを使用した追加のネットワーク割り当ての作成

Cluster Network Operator (CNO) は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、CNO は **NetworkAttachmentDefinition** カスタムリソース (CR) を自動的に作成します。



注記

Cluster Network Operator が管理する **NetworkAttachmentDefinition** CR は編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

CNI VRF プラグインで追加のネットワーク割り当てを作成するには、以下の手順を実行します。

前提条件

- OpenShift Container Platform CLI (oc) をインストールします。
- cluster-admin 権限を持つユーザーとして OpenShift クラスターにログインします。

手順

1. 以下のサンプル CR のように、追加のネットワーク割り当て用の **Network** カスタムリソース (CR) を作成し、追加ネットワークの **rawCNIConfig** 設定を挿入します。YAML を **additional-network-attachment.yaml** ファイルとして保存します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: test-network-1
    namespace: additional-network-1
    type: Raw
    rawCNIConfig: '{
      "cniVersion": "0.3.1",
      "name": "macvlan-vrf",
```

```
"plugins": [ ❶
{
  "type": "macvlan", ❷
  "master": "eth1",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "191.168.1.23/24"
      }
    ]
  }
},
{
  "type": "vrf",
  "vrfname": "example-vrf-name", ❸
  "table": 1001 ❹
}
}]
}'
```

- ❶ **plugins** は一覧である必要があります。一覧の最初の項目は、VRF ネットワークのベースとなるセカンダリーネットワークである必要があります。一覧の2つ目の項目は、VRF プラグイン設定です。
- ❷ **type** は **vrf** に設定する必要があります。
- ❸ **vrfname** は、インターフェイスが割り当てられた VRF の名前です。これが Pod に存在しない場合は作成されます。
- ❹ オプション **table** はルーティングテーブル ID です。デフォルトで、**tableid** パラメーターが使用されます。これが指定されていない場合、CNI は空のルーティングテーブル ID を VRF に割り当てます。



注記

VRF は、リソースが **netdevice** タイプの場合にのみ正常に機能します。

2. **Network** リソースを作成します。

```
$ oc create -f additional-network-attachment.yaml
```

3. 以下のコマンドを実行して、CNO が **NetworkAttachmentDefinition** CR を作成していることを確認します。<namespace> を、ネットワーク割り当ての設定時に指定した namespace に置き換えます (例: **additional-network-1**)。

```
$ oc get network-attachment-definitions -n <namespace>
```

出力例

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



注記

CNO が CR を作成するまでに遅延が生じる可能性があります。

追加の VRF ネットワーク割り当てが正常であることの確認

VRF CNI が正しく設定され、追加のネットワーク割り当てが接続されていることを確認するには、以下を実行します。

1. VRF CNI を使用するネットワークを作成します。
2. ネットワークを Pod に割り当てます。
3. Pod のネットワーク割り当てが VRF の追加ネットワークに接続されていることを確認します。Pod にリモートシェルを実行し、以下のコマンドを実行します。

```
$ ip vrf show
```

出力例

```
Name          Table
-----
red            10
```

4. VRF インターフェイスがセカンダリーインターフェイスのマスターであることを確認します。

```
$ ip link
```

出力例

```
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red
state UP mode
```


第14章 ハードウェアネットワーク

14.1. SINGLE ROOT I/O VIRTUALIZATION (SR-IOV) ハードウェアネットワークについて

Single Root I/O Virtualization (SR-IOV) 仕様は、単一デバイスを複数の Pod で共有できる PCI デバイス割り当てタイプの標準です。

SR-IOV を使用すると、準拠したネットワークデバイス (ホストノードで物理機能 (PF) として認識される) を複数の Virtual Function (VF) にセグメント化することができます。VF は他のネットワークデバイスと同様に使用されます。デバイスの SR-IOV ネットワークデバイスドライバは、VF がコンテナで公開される方法を判別します。

- **netdevice** ドライバー: コンテナの **netns** 内の通常のカーネルネットワークデバイス
- **vfio-pci** ドライバー: コンテナにマウントされるキャラクターデバイス

SR-IOV ネットワークデバイスは、ベアメタルまたは Red Hat Open Stack Platform (RHOSP) インフラ上にインストールされた OpenShift Container Platform クラスタにネットワークを追加して、高帯域または低遅延を確保する必要があるアプリケーションに使用できます。

次のコマンドを使用して、ノードで SR-IOV を有効にできます。

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

14.1.1. SR-IOV ネットワークデバイスを管理するコンポーネント

SR-IOV Network Operator は SR-IOV スタックのコンポーネントを作成し、管理します。以下の機能を実行します。

- SR-IOV ネットワークデバイスの検出および管理のオーケストレーション
- SR-IOV Container Network Interface (CNI) の **NetworkAttachmentDefinition** カスタムリソースの生成
- SR-IOV ネットワークデバイスプラグインの設定の作成および更新
- ノード固有の **SriovNetworkNodeState** カスタムリソースの作成
- 各 **SriovNetworkNodeState** カスタムリソースの **spec.interfaces** フィールドの更新

Operator は以下のコンポーネントをプロビジョニングします。

SR-IOV ネットワーク設定デーモン

SR-IOV Network Operator の起動時にワーカーノードにデプロイされるデーモンセット。デーモンは、クラスタで SR-IOV ネットワークデバイスを検出し、初期化します。

SR-IOV ネットワーク Operator Webhook

Operator カスタムリソースを検証し、未設定フィールドに適切なデフォルト値を設定する動的受付コントローラー Webhook。

SR-IOV Network Resources Injector

SR-IOV VF などのカスタムネットワークリソースの要求および制限のある Kubernetes Pod 仕様のパッチを適用するための機能を提供する動的受付コントローラー Webhook。SR-IOV ネットワークリソースインジェクターは、Pod 内の最初のコンテナのみに **resource** フィールドを自動的に追

加します。

SR-IOV ネットワークデバイスプラグイン

SR-IOV ネットワーク Virtual Function (VF) リソースの検出、公開、割り当てを実行するデバイスプラグイン。デバイスプラグインは、とりわけ物理デバイスでの制限されたリソースの使用を有効にするために Kubernetes で使用されます。デバイスプラグインは Kubernetes スケジューラーにリソースの可用性を認識させるため、スケジューラーはリソースが十分にあるノードで Pod をスケジューリングできます。

SR-IOV CNI プラグイン

SR-IOV ネットワークデバイスプラグインから割り当てられる VF インターフェイスを直接 Pod に割り当てる CNI プラグイン。

SR-IOV InfiniBand CNI プラグイン

SR-IOV ネットワークデバイスプラグインから割り当てられる InfiniBand (IB) VF インターフェイスを直接 Pod に割り当てる CNI プラグイン。



注記

SR-IOV Network Resources Injector および SR-IOV Network Operator Webhook は、デフォルトで有効にされ、**default** の **SriovOperatorConfig** CR を編集して無効にできます。SR-IOV Network Operator Admission Controller Webhook を無効にする場合は注意してください。トラブルシューティングなどの特定の状況下や、サポートされていないデバイスを使用する場合は、Webhook を無効にすることができます。

14.1.1.1. サポート対象のプラットフォーム

SR-IOV Network Operator は、以下のプラットフォームに対応しています。

- ベアメタル
- Red Hat OpenStack Platform (RHOSP)

14.1.1.2. サポートされるデバイス

以下のネットワークインターフェイスコントローラーは、OpenShift Container Platform でサポートされています。

表14.1 サポート対象のネットワークインターフェイスコントローラー

製造元	モデル	ベンダー ID	デバイス ID
Broadcom	BCM57414	14e4	16d7
Broadcom	BCM57508	14e4	1750
Intel	X710	8086	1572
Intel	XL710	8086	1583
Intel	XXV710	8086	158b
Intel	E810-CQDA2	8086	1592

製造元	モデル	ベンダー ID	デバイス ID
Intel	E810-2CQDA2	8086	1592
Intel	E810-XXVDA2	8086	159b
Intel	E810-XXVDA4	8086	1593
Mellanox	MT27700 Family [ConnectX-4]	15b3	1013
Mellanox	MT27710 Family [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 Family [ConnectX-6]	15b3	101b
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f



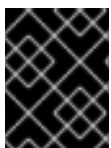
注記

サポートされているカードの最新リストおよび利用可能な互換性のある OpenShift Container Platform バージョンについては、[Openshift Single Root I/O Virtualization \(SR-IOV\) and PTP hardware networks Support Matrix](#) を参照してください。

14.1.1.3. SR-IOV ネットワークデバイスの自動検出

SR-IOV Network Operator は、クラスターでワーカーノード上の SR-IOV 対応ネットワークデバイスを検索します。Operator は、互換性のある SR-IOV ネットワークデバイスを提供する各ワーカーノードの `SriovNetworkNodeState` カスタムリソース (CR) を作成し、更新します。

CR にはワーカーノードと同じ名前が割り当てられます。`status.interfaces` 一覧は、ノード上のネットワークデバイスについての情報を提供します。



重要

`SriovNetworkNodeState` オブジェクトは変更しないでください。Operator はこれらのリソースを自動的に作成し、管理します。

14.1.1.3.1. SriovNetworkNodeState オブジェクトの例

以下の YAML は、SR-IOV Network Operator によって作成される `SriovNetworkNodeState` オブジェクトの例です。

SriovNetworkNodeState オブジェクト

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
```

```
name: node-25 ❶
namespace: openshift-sriov-network-operator
ownerReferences:
- apiVersion: sriovnetwork.openshift.io/v1
  blockOwnerDeletion: true
  controller: true
  kind: SriovNetworkNodePolicy
  name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: ❷
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
syncStatus: Succeeded
```

❶ **name** フィールドの値はワーカーノードの名前と同じです。

❷ **interfaces** スタンザには、ワーカーノード上の Operator によって検出されるすべての SR-IOV デバイスの一覧が含まれます。

14.1.1.4. Pod での Virtual Function (VF) の使用例

SR-IOV VF が割り当てられている Pod で、Remote Direct Memory Access (RDMA) または Data Plane Development Kit (DPDK) アプリケーションを実行できます。

以下の例では、RDMA モードで Virtual Function (VF) を使用する Pod を示しています。

RDMA モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-rdma-mlnx
spec:
  containers:
  - name: testpmd
    image: <RDMA_image>
    imagePullPolicy: IfNotPresent
    securityContext:
      runAsUser: 0
    capabilities:
      add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"]
    command: ["sleep", "infinity"]
```

以下の例は、DPDK モードの VF のある Pod を示しています。

DPDK モードを使用する Pod 仕様

```
apiVersion: v1
kind: Pod
metadata:
  name: dpdk-app
  annotations:
    k8s.v1.cni.cncf.io/networks: sriov-dpdk-net
spec:
  containers:
  - name: testpmd
    image: <DPDK_image>
    securityContext:
      runAsUser: 0
    capabilities:
      add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"]
  volumeMounts:
  - mountPath: /dev/hugepages
    name: hugepage
  resources:
    limits:
      memory: "1Gi"
      cpu: "2"
      hugepages-1Gi: "4Gi"
    requests:
      memory: "1Gi"
      cpu: "2"
```

```

hugepages-1Gi: "4Gi"
command: ["sleep", "infinity"]
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages

```

14.1.1.5. コンテナアプリケーションで使用する DPDK ライブラリー

オプションライブラリー の **app-netutil** は、その Pod 内で実行されるコンテナから Pod についてのネットワーク情報を収集するための複数の API メソッドを提供します。

このライブラリーは、DPDK (Data Plane Development Kit) モードの SR-IOV Virtual Function (VF) のコンテナへの統合を支援します。このライブラリーは Golang API と C API の両方を提供します。

現時点で 3 つの API メソッドが実装されています。

GetCPUInfo()

この機能は、コンテナで利用可能な CPU を判別し、一覧を返します。

GetHugepages()

この機能は、各コンテナの **Pod** 仕様で要求される huge page メモリーの量を判別し、値を返します。

GetInterfaces()

この機能は、コンテナのインターフェイスセットを判別し、インターフェイスタイプとタイプ固有のデータと共に一覧を返します。戻り値には、インターフェイスのタイプと、各インターフェイスのタイプ固有のデータが含まれます。

ライブラリーのリポジトリには、コンテナイメージ **dppdk-app-centos** をビルドするためのサンプル Dockerfile が含まれます。コンテナイメージは、Pod 仕様の環境変数に応じて、**l2fwd**、**l3fwd** または **testpmd** の DPDK サンプルアプリケーションのいずれかを実行できます。コンテナイメージは、**app-netutil** ライブラリーをコンテナイメージ自体に統合する例を提供します。ライブラリーを init コンテナに統合することもできます。init コンテナは必要なデータを収集し、データを既存の DPDK ワークロードに渡すことができます。

14.1.1.6. Downward API の Huge Page リソースの挿入

Pod 仕様に Huge Page のリソース要求または制限が含まれる場合、Network Resources Injector は Downward API フィールドを Pod 仕様に自動的に追加し、Huge Page 情報をコンテナに提供します。

Network Resources Injector は、**podnetinfo** という名前のボリュームを追加し、Pod の各コンテナ用に **/etc/podnetinfo** にマウントされます。ボリュームは Downward API を使用し、Huge Page の要求および制限についてのファイルを追加します。ファイルの命名規則は以下のとおりです。

- **/etc/podnetinfo/hugepages_1G_request_<container-name>**
- **/etc/podnetinfo/hugepages_1G_limit_<container-name>**
- **/etc/podnetinfo/hugepages_2M_request_<container-name>**
- **/etc/podnetinfo/hugepages_2M_limit_<container-name>**

直前の一覧で指定されているパスは、**app-netutil** ライブラリーと互換性があります。デフォルトで、ライブラリーは、**/etc/podnetinfo** ディレクトリーのリソース情報を検索するように設定されます。

Downward API パス項目を手動で指定する選択をする場合、**app-netutil** ライブラリーは前述の一覧のパスに加えて以下のパスを検索します。

- `/etc/podnetinfo/hugepages_request`
- `/etc/podnetinfo/hugepages_limit`
- `/etc/podnetinfo/hugepages_1G_request`
- `/etc/podnetinfo/hugepages_1G_limit`
- `/etc/podnetinfo/hugepages_2M_request`
- `/etc/podnetinfo/hugepages_2M_limit`

Network Resources Injector が作成できるパスと同様に、前述の一覧のパスの末尾にはオプションで `_<container-name>` 接尾辞を付けることができます。

14.1.2. 次のステップ

- [SR-IOV Network Operator のインストール](#)
- オプション: [SR-IOV Network Operator の設定](#)
- [SR-IOV ネットワークデバイスの設定](#)
- OpenShift Virtualization を使用する場合: [仮想マシンの SR-IOV ネットワークへの接続](#)
- [SR-IOV ネットワーク割り当ての設定](#)
- [Pod の SR-IOV の追加ネットワークへの追加](#)

14.2. SR-IOV NETWORK OPERATOR のインストール

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator をクラスターにインストールし、SR-IOV ネットワークデバイスとネットワークの割り当てを管理できます。

14.2.1. SR-IOV Network Operator のインストール

クラスター管理者は、OpenShift Container Platform CLI または Web コンソールを使用して SR-IOV Network Operator をインストールできます。

14.2.1.1. CLI: SR-IOV Network Operator のインストール

クラスター管理者は、CLI を使用して Operator をインストールできます。

前提条件

- SR-IOV に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つアカウント。

手順

1. **openshift-sriov-network-operator** namespace を作成するには、以下のコマンドを入力します。

```
$ cat << EOF | oc create -f -
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
  annotations:
    workload.openshift.io/allowed: management
EOF
```

2. OperatorGroup CR を作成するには、以下のコマンドを実行します。

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

3. SR-IOV Network Operator にサブスクライブします。

- a. 以下のコマンドを実行して OpenShift Container Platform のメジャーおよびマイナーバージョンを取得します。これは、次の手順の **channel** の値に必要です。

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

- b. SR-IOV Network Operator の Subscription CR を作成するには、以下のコマンドを入力します。

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

4. Operator がインストールされていることを確認するには、以下のコマンドを入力します。

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


出力例

Name	Phase
sriov-network-operator.4.9.0-202110121402	Succeeded

14.2.1.2. Web コンソール: SR-IOV Network Operator のインストール

クラスター管理者は、Web コンソールを使用して Operator をインストールできます。

前提条件

- SR-IOV に対応するハードウェアを持つノードでベアメタルハードウェアにインストールされたクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つアカウント。

手順

1. SR-IOV Network Operator をインストールします。
 - a. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** をクリックします。
 - b. 利用可能な Operator の一覧から **SR-IOV Network Operator** を選択してから **Install** をクリックします。
 - c. **Install Operator** ページの **Installed Namespace** で、**Operator recommend Namespace** を選択します。
 - d. **Install** をクリックします。
2. SR-IOV Network Operator が正常にインストールされていることを確認します。
 - a. **Operators** → **Installed Operators** ページに移動します。
 - b. **Status** が **InstallSucceeded** の状態で、**SR-IOV Network Operator** が **openshift-sriov-network-operator** プロジェクトに一覧表示されていることを確認します。



注記

インストール時に、Operator は **Failed** ステータスを表示する可能性があります。インストールが後に **InstallSucceeded** メッセージを出して正常に実行される場合は、**Failed** メッセージを無視できます。

Operator がインストール済みとして表示されない場合に、さらにトラブルシューティングを実行します。

- **Operator Subscriptions** および **Install Plans** タブで、**Status** の下の失敗またはエラーの有無を確認します。
- **Workloads** → **Pods** ページに移動し、**openshift-sriov-network-operator** プロジェクトで Pod のログを確認します。

- YAML ファイルの namespace を確認してください。アノテーションが抜けている場合は、次のコマンドを使用して、アノテーション **workload.openshift.io/allowed=management** を Operator namespace に追加できます。

```
$ oc annotate ns/openshift-sriov-network-operator
workload.openshift.io/allowed=management
```



注記

シングルノード OpenShift クラスターの場合は、namespace にアノテーション **workload.openshift.io/allowed=management** が必要です。

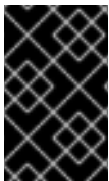
14.2.2. 次のステップ

- オプション: [SR-IOV Network Operator の設定](#)

14.3. SR-IOV NETWORK OPERATOR の設定

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator は、クラスターで SR-IOV ネットワークデバイスおよびネットワーク割り当てを管理します。

14.3.1. SR-IOV Network Operator の設定



重要

通常、SR-IOV Network Operator 設定を変更する必要はありません。デフォルト設定は、ほとんどのユースケースで推奨されます。Operator のデフォルト動作がユースケースと互換性がない場合にのみ、関連する設定を変更する手順を実行します。

SR-IOV Network Operator は **SriovOperatorConfig.sriovnetwork.openshift.io** CustomResourceDefinition リソースを追加します。Operator は、**openshift-sriov-network-operator** namespace に **default** という名前の SriovOperatorConfig カスタムリソース (CR) を自動的に作成します。



注記

default CR には、クラスターの SR-IOV Network Operator 設定が含まれます。Operator 設定を変更するには、この CR を変更する必要があります。

14.3.1.1. SR-IOV Network Operator config カスタムリソース

sriovoperatorconfig カスタムリソースのフィールドは、以下の表で説明されています。

表14.2 SR-IOV Network Operator config カスタムリソース

フィールド	型	説明
-------	---	----

フィールド	型	説明
<code>metadata.name</code>	<code>string</code>	SR-IOV Network Operator インスタンスの名前を指定します。デフォルト値は default です。別の値を設定しないでください。
<code>metadata.name space</code>	<code>string</code>	SR-IOV Network Operator インスタンスの namespace を指定します。デフォルト値は openshift-sriov-network-operator です。別の値を設定しないでください。
<code>spec.configDaemonNodeSelector</code>	<code>string</code>	選択されたノードで SR-IOV Network Config Daemon のスケジューリングを制御するノードの選択オプションを指定します。デフォルトでは、このフィールドは設定されておらず、Operator はワーカーノードに SR-IOV Network Config デモンセットを配置します。
<code>spec.disableDrain</code>	<code>boolean</code>	新しいポリシーを適用してノードに NIC を設定する時に、ノードドレインプロセスを無効にするか、有効にするかを指定します。このフィールドを true に設定すると、ソフトウェアの開発や OpenShift Container Platform の単一ノードへのインストールが容易になります。デフォルトでは、このフィールドは設定されていません。 シングルノードクラスターの場合は、Operator のインストール後にこのフィールドを true に設定します。このフィールドは必ず true に設定してください。
<code>spec.enableInjector</code>	<code>boolean</code>	Network Resources Injector デモンセットを有効にするか無効にするかを指定します。デフォルトでは、このフィールドは true に設定されています。
<code>spec.enableOperatorWebhook</code>	<code>boolean</code>	Operator Admission Controller の Webhook デモンセットを有効にするか無効にするかを指定します。デフォルトでは、このフィールドは true に設定されています。
<code>spec.logLevel</code>	<code>integer</code>	Operator のログの冗長度を指定します。 0 に設定すると、基本的なログのみを表示します。 2 に設定すると、利用可能なすべてのログが表示されます。デフォルトでは、このフィールドは 2 に設定されています。

14.3.1.2. Network Resources Injector について

Network Resources Injector は Kubernetes Dynamic Admission Controller アプリケーションです。これは、以下の機能を提供します。

- SR-IOV リソース名を SR-IOV ネットワーク割り当て定義アノテーションに従って追加するための、Pod 仕様でのリソース要求および制限の変更。
- Pod のアノテーション、ラベル、および Huge Page の要求および制限を公開するための Downward API ボリュームでの Pod 仕様の変更。Pod で実行されるコンテナは、公開される情報に `/etc/podnetinfo` パスでファイルとしてアクセスできます。

デフォルトで、Network Resources Injector は SR-IOV Network Operator によって有効にされ、すべてのコントロールプレーンノードでデーモンセットとして実行されます。以下は、3つのコントロールプレーンノードを持つクラスターで実行される Network Resources Injector Pod の例です。

```
$ oc get pods -n openshift-sriov-network-operator
```

出力例

NAME	READY	STATUS	RESTARTS	AGE
network-resources-injector-5cz5p	1/1	Running	0	10m
network-resources-injector-dwqpx	1/1	Running	0	10m
network-resources-injector-lktz5	1/1	Running	0	10m

14.3.1.3. SR-IOV Network Operator Admission Controller Webhook について

SR-IOV Network Operator Admission Controller Webhook は Kubernetes Dynamic Admission Controller アプリケーションです。これは、以下の機能を提供します。

- 作成時または更新時の **SriovNetworkNodePolicy** CR の検証
- CR の作成時または更新時の **priority** および **deviceType** フィールドのデフォルト値の設定による **SriovNetworkNodePolicy** CR の変更

デフォルトで、SR-IOV Network Operator Admission Controller Webhook は Operator によって有効にされ、すべてのコントロールプレーンノードでデーモンセットとして実行されます。



注記

SR-IOV Network Operator Admission Controller Webhook を無効にする場合は注意してください。トラブルシューティングなどの特定の状況下や、サポートされていないデバイスを使用する場合は、Webhook を無効にすることができます。

以下は、3つのコントロールプレーンノードを持つクラスターで実行される Operator Admission Controller Webhook Pod の例です。

```
$ oc get pods -n openshift-sriov-network-operator
```

出力例

NAME	READY	STATUS	RESTARTS	AGE
operator-webhook-9jkw6	1/1	Running	0	16m
operator-webhook-kbr5p	1/1	Running	0	16m
operator-webhook-rpfrl	1/1	Running	0	16m

14.3.1.4. カスタムノードセクターについて

SR-IOV Network Config デーモンは、クラスターノード上の SR-IOV ネットワークデバイスを検出し、設定します。デフォルトで、これはクラスター内のすべての **worker** ノードにデプロイされます。ノードラベルを使用して、SR-IOV Network Config デーモンが実行するノードを指定できます。

14.3.1.5. Network Resources Injector の無効化または有効化

デフォルトで有効にされている Network Resources Injector を無効にするか、または有効にするには、以下の手順を実行します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- SR-IOV Network Operator がインストールされていること。

手順

- **enableInjector** フィールドを設定します。<value> を **false** に置き換えて機能を無効にするか、または **true** に置き換えて機能を有効にします。

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

ヒント

または、以下の YAML を適用して Operator を更新することもできます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableInjector: <value>
```

14.3.1.6. SR-IOV Network Operator Admission Controller Webhook の無効化または有効化

デフォルトで有効にされている なっている受付コントローラー Webhook を無効にするか、または有効にするには、以下の手順を実行します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- SR-IOV Network Operator がインストールされていること。

手順

- **enableOperatorWebhook** フィールドを設定します。<value> を **false** に置き換えて機能を無効するか、**true** に置き換えて機能を有効にします。

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

ヒント

または、以下の YAML を適用して Operator を更新することもできます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableOperatorWebhook: <value>
```

14.3.1.7. SRIOV Network Config Daemon のカスタム NodeSelector の設定

SR-IOV Network Config デーモンは、クラスターノード上の SR-IOV ネットワークデバイスを検出し、設定します。デフォルトで、これはクラスター内のすべての **worker** ノードにデプロイされます。ノードラベルを使用して、SR-IOV Network Config デーモンが実行するノードを指定できます。

SR-IOV Network Config デーモンがデプロイされるノードを指定するには、以下の手順を実行します。



重要

configDaemonNodeSelector フィールドを更新する際に、SR-IOV Network Config デーモンがそれぞれの選択されたノードに再作成されます。デーモンが再作成されている間、クラスターのユーザーは新規の SR-IOV Network ノードポリシーを適用したり、新規の SR-IOV Pod を作成したりできません。

手順

- Operator のノードセレクターを更新するには、以下のコマンドを入力します。

```
$ oc patch sriovoperatorconfig default --type=json \
-n openshift-sriov-network-operator \
--patch '[{
  "op": "replace",
  "path": "/spec/configDaemonNodeSelector",
  "value": {<node_label>}
}]'
```

以下の例のように、<node_label> を適用するラベルに置き換えます: **"node-role.kubernetes.io/worker": ""**

ヒント

または、以下の YAML を適用して Operator を更新することもできます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  configDaemonNodeSelector:
    <node_label>
```

14.3.1.8. 単一ノードのインストール用の SR-IOV Network Operator の設定

デフォルトでは、SR-IOV Network Operator は、ポリシーを変更するたびに、ノードからワークロードをドレイン (解放) します。Operator は、このアクションを実行して、再設定する前に Virtual Function を使用しているワークロードがないことを確認します。

1つのノードにインストールする場合には、ワークロードを受信するノードは他にありません。そのため、Operator は、単一のノードからワークロードがドレインされないように設定する必要があります。



重要

以下の手順を実行してワークロードのドレインを無効にした後に、SR-IOV ネットワーク インターフェイスを使用しているワークロードを削除してから SR-IOV ネットワーク ノードのポリシーを変更する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- SR-IOV Network Operator がインストールされていること。

手順

- **disable Drain** フィールドを **true** に設定するには、次のコマンドを入力します。

```
$ oc patch sriovoperatorconfig default --type=merge \
  -n openshift-sriov-network-operator \
  --patch '{"spec": {"disableDrain": true }}'
```

ヒント

または、以下の YAML を適用して Operator を更新することもできます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  disableDrain: true
```

14.3.2. 次のステップ

- [SR-IOV ネットワークデバイスの設定](#)

14.4. SR-IOV ネットワークデバイスの設定

クラスターで Single Root I/O Virtualization (SR-IOV) デバイスを設定できます。

14.4.1. SR-IOV ネットワークノード設定オブジェクト

SR-IOV ネットワークノードポリシーを作成して、ノードの SR-IOV ネットワークデバイス設定を指定します。ポリシーの API オブジェクトは **sriovnetwork.openshift.io** API グループの一部です。

以下の YAML は SR-IOV ネットワークノードポリシーについて説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  needVhostNet: false 7
  numVfs: <num> 8
  nicSelector: 9
    vendor: "<vendor_code>" 10
    deviceID: "<device_id>" 11
    pfNames: ["<pf_name>", ...] 12
    rootDevices: ["<pci_bus_id>", ...] 13
    netFilter: "<filter_string>" 14
  deviceType: <device_type> 15
  isRdma: false 16
  linkType: <link_type> 17
```

- 1 カスタムリソースオブジェクトの名前。
- 2 SR-IOV Network Operator がインストールされている namespace を指定します。
- 3 SR-IOV ネットワークデバイスプラグインのリソース名。1つのリソース名に複数の SR-IOV ネットワークポリシーを作成できます。
- 4 ノードセレクターは設定するノードを指定します。選択したノード上の SR-IOV ネットワークデバイスのみが設定されます。SR-IOV Container Network Interface (CNI) プラグインおよびデバイスプラグインは、選択したノードにのみデプロイされます。
- 5 オプション: 優先度は **0** から **99** までの整数値で指定されます。値が小さいほど優先度が高くなります。たとえば、**10** の優先度は **99** よりも高くなります。デフォルト値は **99** です。
- 6 オプション: Virtual Function (VF) の最大転送単位 (MTU)。MTU の最大値は、複数の異なるネットワークインターフェイスコントローラー (NIC) に応じて異なります。
- 7 オプション: **/dev/vhost-net** デバイスを Pod にマウントするには、**needVhostNet** を **true** に設定します。Data Plane Development Kit (DPDK) と共にマウントされた **/dev/vhost-net** デバイスを使用して、トラフィックをカーネルネットワークスタックに転送します。
- 8 SR-IOV 物理ネットワークデバイス用に作成する Virtual Function (VF) の数。Intel ネットワークインターフェイスコントローラー (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **128** よりも大きくすることはできません。

- 9 NIC セレクターは、Operator が設定するデバイスを特定します。すべてのパラメーターの値を指定する必要はありません。意図せずにデバイスを選択しないように、ネットワークデバイスを極め
- rootDevices** を指定する場合、**vendor**、**deviceID**、または **pfName** の値も指定する必要があります。**pfNames** および **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスを参照していることを確認します。**netFilter** の値を指定する場合、ネットワーク ID は一意の ID であるためにその他のパラメーターを指定する必要はありません。
- 10 オプション: SR-IOV ネットワークデバイスのベンダーの 16 進数コード。許可される値は **8086** および **15b3** のみになります。
- 11 オプション: SR-IOV ネットワークデバイスのデバイスの 16 進数コード。たとえば、**101b** は Mellanox ConnectX-6 デバイスのデバイス ID です。
- 12 オプション: 1つ以上のデバイスの物理機能 (PF) 名の配列。
- 13 オプション: デバイスの PF 用の 1つ以上の PCI バスアドレスの配列。以下の形式でアドレスを指定します: **0000:02:00.1**
- 14 オプション: プラットフォーム固有のネットワークフィルター。サポートされるプラットフォームは Red Hat OpenStack Platform (RHOSP) のみです。許可される値は、**openstack/NetworkID:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** の形式を使用します。**xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx** を、**/var/config/openstack/latest/network_data.json** メタデータファイルの値に置き換えます。
- 15 オプション: Virtual Function (VF) のドライバータイプ。許可される値は **netdevice** および **vfiopci** のみです。デフォルト値は **netdevice** です。
- Mellanox NIC をベアメタルノードの DPDK モードで機能させるには、**netdevice** ドライバータイプを使用し、**isRdma** を **true** に設定します。
- 16 オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうかを設定します。デフォルト値は **false** です。
- isRdma** パラメーターが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。
- isRdma** を **true** に設定し、追加の **needVhostNet** を **true** に設定して、Fast Datapath DPDK アプリケーションで使用する Mellanox NIC を設定します。
- 17 オプション: VF のリンクタイプ。イーサネットのデフォルト値は **eth** です。InfiniBand の場合は、この値を **ib** に変更します。
- linkType** が **ib** に設定されている場合、SR-IOV Network Operator Webhook によって **isRdma** は **true** に自動的に設定されます。**linkType** が **ib** に設定されている場合、**deviceType** は **vfiopci** に設定できません。
- SriovNetworkNodePolicy** の **linkType** を **eth** に設定しないでください。デバイスプラグインによって報告される使用可能なデバイスの数が正しくなくなる可能性があります。

14.4.1.1. SR-IOV ネットワークノードの設定例

以下の例では、InfiniBand デバイスの設定について説明します。

InfiniBand デバイスの設定例

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-ib-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: ibnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
    rootDevices:
      - "0000:19:00.0"
  linkType: ib
  isRdma: true

```

以下の例では、RHOSP 仮想マシンの SR-IOV ネットワークデバイスの設定について説明します。

仮想マシンの SR-IOV デバイスの設定例

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-sriov-net-openstack-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnic1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 1 ❶
  nicSelector:
    vendor: "15b3"
    deviceID: "101b"
  netFilter: "openstack/NetworkID:ea24bd04-8674-4f69-b0ee-fa0b3bd20509" ❷

```

- ❶ 仮想マシンのノードネットワークポリシーを設定する際に、**numVfs** フィールドは常に **1** に設定されます。
- ❷ **netFilter** フィールドは、仮想マシンが RHOSP にデプロイされる際にネットワーク ID を参照する必要があります。**netFilter** の有効な値は、**SriovNetworkNodeState** オブジェクトから選択できません。

14.4.1.2. SR-IOV デバイスの Virtual Function (VF) パーティション設定

Virtual Function (VF) を同じ物理機能 (PF) から複数のリソースプールに分割する必要がある場合があります。たとえば、VF の一部をデフォルトドライバーで読み込み、残りの VF を **vfiopci** ドライバーで読み込む必要がある場合などです。このようなデプロイメントでは、**SriovNetworkNodePolicy** カスタムリソース (CR) の **pfNames** セレクターは、以下の形式を使用してプールの VF の範囲を指定するために使用できます: **<pfname>#<first_vf>-<last_vf>**

たとえば、以下の YAML は、VF が **2** から **7** までである **netpf0** という名前のインターフェイスのセレクターを示します。

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

- **netpf0** は PF インターフェイス名です。
- **2** は、範囲に含まれる最初の VF インデックス (0 ベース) です。
- **7** は、範囲に含まれる最後の VF インデックス (0 ベース) です。

以下の要件を満たす場合、異なるポリシー CR を使用して同じ PF から VF を選択できます。

- **numVfs** の値は、同じ PF を選択するポリシーで同一である必要があります。
- VF インデックスは、**0** から **<numVfs>-1** の範囲にある必要があります。たとえば、**numVfs** が **8** に設定されているポリシーがある場合、**<first_vf>** の値は **0** よりも小さくすることはできず、**<last_vf>** は **7** よりも大きくすることはできません。
- 異なるポリシーの VF の範囲は重複しないようにしてください。
- **<first_vf>** は **<last_vf>** よりも大きくすることはできません。

以下の例は、SR-IOV デバイスの NIC パーティション設定を示しています。

ポリシー **policy-net-1** は、デフォルトの VF ドライバーと共に PF **netpf0** の VF **0** が含まれるリソースプール **net-1** を定義します。ポリシー **policy-net-1-dpdk** は、**vfi**o VF ドライバーと共に PF **netpf0** の VF **8** から **15** ままで含まれるリソースプール **net-1-dpdk** を定義します。

ポリシー **policy-net-1**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
  nicSelector:
    pfNames: ["netpf0#0-0"]
  deviceType: netdevice
```

ポリシー **policy-net-1-dpdk**:

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-net-1-dpdk
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1dpdk
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 16
```

```
nicSelector:
  pfNames: ["netpf0#8-15"]
deviceType: vfio-pci
```

14.4.2. SR-IOV ネットワークデバイスの設定

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition を OpenShift Container Platform に追加します。SR-IOV ネットワークデバイスは、SriovNetworkNodePolicy カスタムリソース (CR) を作成して設定できます。



注記

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。

設定の変更が適用されるまでに数分かかる場合があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- SR-IOV Network Operator がインストールされている。
- ドレイン (解放) されたノードからエビクトされたワークロードを処理するために、クラスター内に利用可能な十分なノードがあること。
- SR-IOV ネットワークデバイス設定についてコントロールプレーンノードを選択していないこと。

手順

1. **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **<name>-sriov-node-network.yaml** ファイルに保存します。<name> をこの設定の名前に置き換えます。
2. オプション: SR-IOV 対応のクラスターノードにまだラベルが付いていない場合は、**SriovNetworkNodePolicy.Spec.NodeSelector** でラベルを付けます。ノードのラベル付けについて、詳しくはノードのラベルを更新する方法についてを参照してください。
3. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f <name>-sriov-node-network.yaml
```

ここで、<name> はこの設定の名前を指定します。

設定の更新が適用された後に、**sriov-network-operator** namespace のすべての Pod が **Running** ステータスに移行します。

4. SR-IOV ネットワークデバイスが設定されていることを確認するには、以下のコマンドを実行します。<node_name> を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

関連情報

- [ノードでラベルを更新する方法について](#)

14.4.3. SR-IOV 設定のトラブルシューティング

SR-IOV ネットワークデバイスの設定の手順を実行した後に、以下のセクションではエラー状態の一部に対応します。

ノードの状態を表示するには、以下のコマンドを実行します。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name>
```

ここで、<node_name> は SR-IOV ネットワークデバイスを持つノードの名前を指定します。

エラー出力: Cannot allocate memory

```
"lastSyncError": "write /sys/bus/pci/devices/0000:3b:00.1/sriov_numvfs: cannot allocate memory"
```

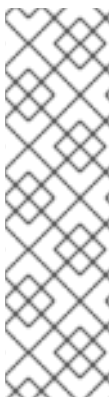
ノードがメモリーを割り当てることができないことを示す場合は、以下の項目を確認します。

- ノードの BIOS でグローバル SR-IOV 設定が有効になっていることを確認します。
- ノードの BIOS で VT-d が有効であることを確認します。

14.4.4. SR-IOV ネットワークの VRF への割り当て

クラスター管理者は、CNI VRF プラグインを使用して、SR-IOV ネットワークインターフェイスを VRF ドメインに割り当てることができます。

これを実行するには、VRF 設定を **SriovNetwork** リソースのオプションの **metaPlugins** パラメーターに追加します。



注記

VRF を使用するアプリケーションを特定のデバイスにバインドする必要があります。一般的な使用方法として、ソケットに **SO_BINDTODEVICE** オプションを使用できます。**SO_BINDTODEVICE** は、渡されるインターフェイス名で指定されているデバイスにソケットをバインドします (例: **eth1**)。 **SO_BINDTODEVICE** を使用するには、アプリケーションに **CAP_NET_RAW** 機能がある必要があります。

ip vrf exec コマンドを使用した VRF の使用は、OpenShift Container Platform Pod ではサポートされません。VRF を使用するには、アプリケーションを VRF インターフェイスに直接バインドします。

14.4.4.1. CNI VRF プラグインを使用した追加 SR-IOV ネットワーク割り当ての作成

SR-IOV Network Operator は追加ネットワークの定義を管理します。作成する追加ネットワークを指定する場合、SR-IOV Network Operator は **NetworkAttachmentDefinition** カスタムリソース (CR) を自動的に作成します。



注記

SR-IOV Network Operator が管理する **NetworkAttachmentDefinition** カスタムリソースは編集しないでください。これを実行すると、追加ネットワークのネットワークトラフィックが中断する可能性があります。

CNI VRF プラグインで追加の SR-IOV ネットワーク割り当てを作成するには、以下の手順を実行します。

前提条件

- OpenShift Container Platform CLI (oc) をインストールします。
- cluster-admin 権限を持つユーザーとして OpenShift Container Platform クラスターにログインします。

手順

1. 追加の SR-IOV ネットワーク割り当て用の **SriovNetwork** カスタムリソース (CR) を作成し、以下のサンプル CR のように **metaPlugins** 設定を挿入します。YAML を **sriov-network-attachment.yaml** ファイルとして保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: example-network
  namespace: additional-sriov-network-1
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "10.56.217.1"
    }
  vlan: 0
  resourceName: intelnic
  metaPlugins : |
    {
      "type": "vrf", ①
      "vrfname": "example-vrf-name" ②
    }
```

① **type** は **vrf** に設定する必要があります。

② **vrfname** は、インターフェイスが割り当てられた VRF の名前です。これが Pod に存在しない場合は作成されます。

2. **SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network-attachment.yaml
```

NetworkAttachmentDefinition CR が正常に作成されることの確認

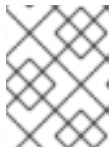
- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。

```
$ oc get network-attachment-definitions -n <namespace> ❶
```

- ❶ **<namespace>** を、ネットワーク割り当ての設定時に指定した namespace に置き換えます (例: **additional-sriov-network-1**)。

出力例

```
NAME                AGE
additional-sriov-network-1  14m
```



注記

SR-IOV Network Operator が CR を作成するまでに遅延が生じる可能性があります。

追加の SR-IOV ネットワーク割り当てが正常であることの確認

VRF CNI が正しく設定され、追加の SR-IOV ネットワーク割り当てが接続されていることを確認するには、以下を実行します。

- VRF CNI を使用する SR-IOV ネットワークを作成します。
- ネットワークを Pod に割り当てます。
- Pod のネットワーク割り当てが SR-IOV の追加ネットワークに接続されていることを確認します。Pod にリモートシェルを実行し、以下のコマンドを実行します。

```
$ ip vrf show
```

出力例

```
Name          Table
-----
red            10
```

- VRF インターフェイスがセカンダリーインターフェイスのマスターであることを確認します。

```
$ ip link
```

出力例

```
...
5: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master red
state UP mode
```




14.4.5. 次のステップ

- [SR-IOV ネットワーク割り当ての設定](#)

14.5. SR-IOV イーサネットネットワーク割り当ての設定

クラスター内の Single Root I/O Virtualization (SR-IOV) デバイスのイーサネットネットワーク割り当てを設定できます。

14.5.1. イーサネットデバイス設定オブジェクト

イーサネットネットワークデバイスは、**SriovNetwork** オブジェクトを定義して設定できます。

以下の YAML は **SriovNetwork** オブジェクトについて説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  vlan: <vlan> 5
  spoofChk: "<spoof_check>" 6
  ipam: |- 7
    {}
  linkState: <link_state> 8
  maxTxRate: <max_tx_rate> 9
  minTxRate: <min_tx_rate> 10
  vlanQoS: <vlan_qos> 11
  trust: "<trust_vf>" 12
  capabilities: <capabilities> 13
```

- 1 オブジェクトの名前。SR-IOV Network Operator は、同じ名前を持つ **NetworkAttachmentDefinition** オブジェクトを作成します。
- 2 SR-IOV Network Operator がインストールされている namespace を指定します。
- 3 この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーターの値。
- 4 **SriovNetwork** オブジェクトのターゲット namespace。ターゲット namespace の Pod のみを追加ネットワークに割り当てることができます。
- 5 オプション: 追加ネットワークの仮想 LAN (VLAN) ID。整数値は **0** から **4095** である必要があります。デフォルト値は **0** です。
- 6 オプション: VF の spoof チェックモード。許可される値は、文字列の **"on"** および **"off"** です。

**重要**

指定する値を引用符で囲む必要があります。そうしないと、オブジェクトは SR-IOV ネットワーク Operator によって拒否されます。

- 7 YAML ブロックスケーラーとしての IPAM CNI プラグインの設定オブジェクトプラグインは、割り当て定義についての IP アドレスの割り当てを管理します。
- 8 オプション: Virtual Function (VF) のリンク状態。許可される値は、**enable**、**disable**、および **auto** です。
- 9 オプション: VF の最大伝送レート (Mbps)。
- 10 オプション: VF の最小伝送レート (Mbps)。この値は、最大伝送レート以下である必要があります。

**注記**

Intel NIC は **minTxRate** パラメーターをサポートしません。詳細は、[BZ#1772847](#) を参照してください。

- 11 オプション: VF の IEEE 802.1p 優先度レベル。デフォルト値は **0** です。
- 12 オプション: VF の信頼モード。許可される値は、文字列の **"on"** および **"off"** です。

**重要**

指定する値を引用符で囲む必要があります。囲まないと、SR-IOV Network Operator はオブジェクトを拒否します。

- 13 オプション: この追加ネットワークに設定する機能。IP アドレスのサポートを有効にするには、**"{ "ips": true }"** を指定できます。または、MAC アドレスのサポートを有効にするには **"{ "mac": true }"** を指定します。

14.5.1.1. 追加ネットワークの IP アドレス割り当ての設定

IPAM (IP アドレス管理) Container Network Interface (CNI) プラグインは、他の CNI プラグインの IP アドレスを提供します。

以下の IP アドレスの割り当てタイプを使用できます。

- 静的割り当て。
- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

14.5.1.1.1. 静的 IP アドレス割り当ての設定

以下の表は、静的 IP アドレスの割り当ての設定について説明しています。

表14.3 ipam 静的設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 static が必要です。
addresses	array	仮想インターフェイスに割り当てる IP アドレスを指定するオブジェクトの配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
routes	array	Pod 内で設定するルート指定するオブジェクトの配列です。
dns	array	オプション: DNS の設定を指定するオブジェクトの配列です。

addressesの配列には、以下のフィールドのあるオブジェクトが必要です。

表14.4 ipam.addresses[] 配列

フィールド	型	説明
address	string	指定する IP アドレスおよびネットワーク接頭辞。たとえば、 10.10.21.10/24 を指定すると、追加のネットワークに IP アドレスの 10.10.21.10 が割り当てられ、ネットマスクは 255.255.255.0 になります。
gateway	string	egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。

表14.5 ipam.routes[] 配列

フィールド	型	説明
dst	string	CIDR 形式の IP アドレス範囲 (192.168.17.0/24 、またはデフォルトルートの 0.0.0.0/0)。
gw	string	ネットワークトラフィックがルーティングされるゲートウェイ。

表14.6 ipam.dns オブジェクト

フィールド	型	説明
nameservers	array	DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
domain	array	ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが example.com に設定されている場合、 example-host の DNS ルックアップクエリーは example-host.example.com として書き換えられます。

フィールド	型	説明
search	array	DNS ルックアップのクエリ時に非修飾ホスト名に追加されるドメイン名の配列 (例: example-host)。

静的 IP アドレス割り当ての設定例

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

14.5.1.1.2. 動的 IP アドレス (DHCP) 割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

SR-IOV ネットワーク Operator は DHCP サーバーデプロイメントを作成しません。Cluster Network Operator は最小限の DHCP サーバーデプロイメントを作成します。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
# ...
```

表14.7 ipam DHCP 設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 dhcp が必要です。

動的 IP アドレス (DHCP) 割り当ての設定例

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

14.5.1.1.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の表は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

表14.8 ipamwhereabouts 設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 whereabouts が必要です。
range	string	IP アドレスと範囲を CIDR 表記。IP アドレスは、この範囲内のアドレスから割り当てられます。
exclude	array	オプション: CIDR 表記の IP アドレスと範囲 (0 個以上) の一覧。除外されたアドレス範囲内の IP アドレスは割り当てられません。

Whereabouts を使用する動的 IP アドレス割り当ての設定例

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

14.5.2. SR-IOV の追加ネットワークの設定

SriovNetwork オブジェクトを作成して、SR-IOV ハードウェアを使用する追加のネットワークを設定できます。**SriovNetwork** オブジェクトの作成時に、SR-IOV Network Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。



注記

SriovNetwork オブジェクトが **running** 状態の Pod に割り当てられている場合、これを変更したり、削除したりしないでください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **SriovNetwork** オブジェクトを作成してから、YAML を **<name>.yaml** ファイルに保存します。**<name>** はこの追加ネットワークの名前になります。オブジェクト仕様は以下の例のようになります。

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

```

kind: SrioNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }

```

- オブジェクトを作成するには、以下のコマンドを入力します。

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

ここで、**<name>** は追加ネットワークの名前を指定します。

- オプション: 以下のコマンドを実行して、直前の手順で作成した **SrioNetwork** オブジェクトに関連付けられた **NetworkAttachmentDefinition** オブジェクトが存在することを確認するには、以下のコマンドを入力します。**<namespace>** を **SrioNetwork** オブジェクトで指定した `networkNamespace` に置き換えます。

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

14.5.3. 次のステップ

- [Pod の SR-IOV の追加ネットワークへの追加](#)

14.5.4. 関連情報

- [SR-IOV ネットワークデバイスの設定](#)

14.6. SR-IOV INFINIBAND ネットワーク割り当ての設定

クラスター内の Single Root I/O Virtualization (SR-IOV) デバイスの InfiniBand (IB) ネットワーク割り当てを設定できます。

14.6.1. InfiniBand デバイス設定オブジェクト

SrioIBNetwork オブジェクトを定義することで、InfiniBand (IB) ネットワークデバイスを設定できます。

以下の YAML は、**SrioIBNetwork** オブジェクトについて説明しています。

```

apiVersion: srioNetwork.openshift.io/v1
kind: SrioIBNetwork
metadata:
  name: <name> 1

```

```

namespace: openshift-sriov-network-operator ❷
spec:
  resourceName: <sriov_resource_name> ❸
  networkNamespace: <target_namespace> ❹
  ipam: |- ❺
    {}
  linkState: <link_state> ❻
  capabilities: <capabilities> ❼

```

- ❶ オブジェクトの名前。SR-IOV Network Operator は、同じ名前を持つ **NetworkAttachmentDefinition** オブジェクトを作成します。
- ❷ SR-IOV Operator がインストールされている namespace。
- ❸ この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** パラメーターの値。
- ❹ **SriovIBNetwork** オブジェクトのターゲット namespace。ターゲット namespace の Pod のみをネットワークデバイスに割り当てることができます。
- ❺ オプション: YAML ブロックスケーラーとしての IPAM CNI プラグインの設定オブジェクト。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。
- ❻ オプション: Virtual Function (VF) のリンク状態。許可される値は、**enable**、**disable**、および **auto** です。
- ❼ オプション: このネットワークに設定する機能。"**{ "ips": true }**" を指定して IP アドレスのサポートを有効にするか、"**{ "infinibandGUID": true }**" を指定して IB Global Unique Identifier (GUID) サポートを有効にします。

14.6.1.1. 追加ネットワークの IP アドレス割り当ての設定

IPAM (IP アドレス管理) Container Network Interface (CNI) プラグインは、他の CNI プラグインの IP アドレスを提供します。

以下の IP アドレスの割り当てタイプを使用できます。

- 静的割り当て。
- DHCP サーバーを使用した動的割り当て。指定する DHCP サーバーは、追加のネットワークから到達可能である必要があります。
- Whereabouts IPAM CNI プラグインを使用した動的割り当て。

14.6.1.1.1. 静的 IP アドレス割り当ての設定

以下の表は、静的 IP アドレスの割り当ての設定について説明しています。

表14.9 ipam 静的設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 static が必要です。

フィールド	型	説明
addresses	array	仮想インターフェイスに割り当てる IP アドレスを指定するオブジェクトの配列。IPv4 と IPv6 の IP アドレスの両方がサポートされます。
routes	array	Pod 内で設定するルートを指定するオブジェクトの配列です。
dns	array	オプション: DNS の設定を指定するオブジェクトの配列です。

addressesの配列には、以下のフィールドのあるオブジェクトが必要です。

表14.10 ipam.addresses[] 配列

フィールド	型	説明
address	string	指定する IP アドレスおよびネットワーク接頭辞。たとえば、 10.10.21.10/24 を指定すると、追加のネットワークに IP アドレスの 10.10.21.10 が割り当てられ、ネットマスクは 255.255.255.0 になります。
gateway	string	egress ネットワークトラフィックをルーティングするデフォルトのゲートウェイ。

表14.11 ipam.routes[] 配列

フィールド	型	説明
dst	string	CIDR 形式の IP アドレス範囲 (192.168.17.0/24 、またはデフォルトルートの 0.0.0.0/0)。
gw	string	ネットワークトラフィックがルーティングされるゲートウェイ。

表14.12 ipam.dns オブジェクト

フィールド	型	説明
nameservers	array	DNS クエリーの送信先となる 1 つ以上の IP アドレスの配列。
domain	array	ホスト名に追加するデフォルトのドメイン。たとえば、ドメインが example.com に設定されている場合、 example-host の DNS ルックアップクエリーは example-host.example.com として書き換えられます。

フィールド	型	説明
search	array	DNS ルックアップのクエリ時に非修飾ホスト名に追加されるドメイン名の配列 (例: example-host)。

静的 IP アドレス割り当ての設定例

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

14.6.1.1.2. 動的 IP アドレス (DHCP) 割り当ての設定

以下の JSON は、DHCP を使用した動的 IP アドレスの割り当ての設定について説明しています。

DHCP リースの更新

Pod は、作成時に元の DHCP リースを取得します。リースは、クラスターで実行している最小限の DHCP サーバーデプロイメントで定期的に更新する必要があります。

DHCP サーバーのデプロイメントをトリガーするには、以下の例にあるように Cluster Network Operator 設定を編集して shim ネットワーク割り当てを作成する必要があります。

shim ネットワーク割り当ての定義例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: dhcp-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
        "name": "dhcp-shim",
        "cniVersion": "0.3.1",
        "type": "bridge",
        "ipam": {
          "type": "dhcp"
        }
      }
# ...
```

表14.13 ipam DHCP 設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 dhcp が必要です。

動的 IP アドレス (DHCP) 割り当ての設定例

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

14.6.1.1.3. Whereabouts を使用した動的 IP アドレス割り当ての設定

Whereabouts CNI プラグインにより、DHCP サーバーを使用せずに IP アドレスを追加のネットワークに動的に割り当てることができます。

以下の表は、Whereabouts を使用した動的 IP アドレス割り当ての設定について説明しています。

表14.14 ipamwhereabouts 設定オブジェクト

フィールド	型	説明
type	string	IPAM のアドレスタイプ。値 whereabouts が必要です。
range	string	IP アドレスと範囲を CIDR 表記。IP アドレスは、この範囲内のアドレスから割り当てられます。
exclude	array	オプション: CIDR 表記の IP アドレスと範囲 (0 個以上) の一覧。除外されたアドレス範囲内の IP アドレスは割り当てられません。

Whereabouts を使用する動的 IP アドレス割り当ての設定例

```
{
  "ipam": {
    "type": "whereabouts",
    "range": "192.0.2.192/27",
    "exclude": [
      "192.0.2.192/30",
      "192.0.2.196/32"
    ]
  }
}
```

14.6.2. SR-IOV の追加ネットワークの設定

SriovIBNetwork オブジェクトを作成して、SR-IOV ハードウェアを使用する追加のネットワークを設定できます。**SriovIBNetwork** オブジェクトの作成時に、SR-IOV Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。



注記

SriovIBNetwork オブジェクトが、**running** 状態の Pod に割り当てられている場合、これを変更したり、削除したりしないでください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **SriovIBNetwork** CR を作成してから、YAML を **<name>.yaml** ファイルに保存します。**<name>** は、この追加ネットワークの名前になります。オブジェクト仕様は以下の例のようになります。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: attach1
  namespace: openshift-sriov-network-operator
```

```
spec:
  resourceName: net1
  networkNamespace: project2
  ipam: |-
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "gateway": "10.56.217.1"
    }
  }
```

- オブジェクトを作成するには、以下のコマンドを入力します。

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

ここで、**<name>** は追加ネットワークの名前を指定します。

- オプション: 以下のコマンドを実行して、直前の手順で作成した **SriovIBNetwork** オブジェクトに関連付けられた **NetworkAttachmentDefinition** オブジェクトが存在することを確認します。**<namespace>** を **SriovIBNetwork** オブジェクトで指定した `networkNamespace` に置き換えます。

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

14.6.3. 次のステップ

- Pod の SR-IOV の追加ネットワークへの追加

14.6.4. 関連情報

- SR-IOV ネットワークデバイスの設定

14.7. POD の SR-IOV の追加ネットワークへの追加

Pod を既存の Single Root I/O Virtualization (SR-IOV) ネットワークに追加できます。

14.7.1. ネットワーク割り当てのランタイム設定

Pod を追加のネットワークに割り当てる場合、ランタイム設定を指定して Pod の特定のカスタマイズを行うことができます。たとえば、特定の MAC ハードウェアアドレスを要求できます。

Pod 仕様にアノテーションを設定して、ランタイム設定を指定します。アノテーションキーは **k8s.v1.cni.cncf.io/networks** で、ランタイム設定を記述する JSON オブジェクトを受け入れます。

14.7.1.1. イーサネットベースの SR-IOV 割り当てのランタイム設定

以下の JSON は、イーサネットベースの SR-IOV ネットワーク割り当て用のランタイム設定オプションを説明しています。

```
[
  {
    "name": "<name>", 1
```

```
"mac": "<mac_address>", ②
"ips": ["<cidr_range>"] ③
}
]
```

- ① SR-IOV ネットワーク割り当て定義 CR の名前。
- ② オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの MAC アドレス。この機能を使用するには、**SriovNetwork** オブジェクトで { "mac": true } も指定する必要があります。
- ③ オプション: SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレス。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovNetwork** オブジェクトで { "ips": true } も指定する必要があります。

ランタイム設定の例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "net1",
          "mac": "20:04:0f:f1:88:01",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

14.7.1.2. InfiniBand ベースの SR-IOV 割り当てのランタイム設定

以下の JSON は、InfiniBand ベースの SR-IOV ネットワーク割り当て用のランタイム設定オプションを説明しています。

```
[
  {
    "name": "<network_attachment>", ①
    "infiniband-guid": "<guid>", ②
    "ips": ["<cidr_range>"] ③
  }
]
```

- ① SR-IOV ネットワーク割り当て定義 CR の名前。
- ② SR-IOV デバイスの InfiniBand GUID この機能を使用するには、**SriovIBNetwork** オブジェクトで { "infinibandGUID": true } も指定する必要があります。

- 3 SR-IOV ネットワーク割り当て定義 CR で定義されるリソースタイプから割り当てられる SR-IOV デバイスの IP アドレス。IPv4 と IPv6 アドレスの両方がサポートされます。この機能を使用するには、**SriovNetwork** オブジェクトで `{ "ips": true }` も指定する必要があります。

ランタイム設定の例

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

14.7.2. Pod の追加ネットワークへの追加

Pod を追加のネットワークに追加できます。Pod は、デフォルトネットワークで通常のクラスター関連のネットワークトラフィックを継続的に送信します。

Pod が作成されると、追加のネットワークが割り当てられます。ただし、Pod がすでに存在する場合は、追加のネットワークをこれに割り当ててはできません。

Pod が追加ネットワークと同じ namespace にあること。

注記

SR-IOV Network Resource Injector は、Pod の最初のコンテナに **resource** フィールドを自動的に追加します。

データプレーン開発キット (DPDK) モードでインテル製のネットワークインターフェイスコントローラー (NIC) を使用している場合には、Pod 内の最初のコンテナのみが NIC にアクセスできるように設定されています。SR-IOV 追加ネットワークは、**Sriov Network Node Policy** オブジェクトで **device Type** が **vfiopci** に設定されてる場合は DPDK モードに設定されます。

この問題は、NIC にアクセスする必要のあるコンテナが **Pod** オブジェクトで定義された最初のコンテナであることを確認するか、Network Resource Injector を無効にすることで回避できます。詳細は、[BZ#1990953](#) を参照してください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスタにログインする。
- SR-IOV Operator のインストール。
- Pod を割り当てる **SriovNetwork** オブジェクトまたは **SriovIBNetwork** オブジェクトのいずれかを作成する。

手順

1. アノテーションを **Pod** オブジェクトに追加します。以下のアノテーション形式のいずれかのみを使用できます。

- a. カスタマイズせずに追加ネットワークを割り当てるには、以下の形式でアノテーションを追加します。**<network>** を、Pod に関連付ける追加ネットワークの名前に置き換えます。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: <network>[,<network>,...] 1
```

- 1 複数の追加ネットワークを指定するには、各ネットワークをコンマで区切ります。コンマの間にはスペースを入れないでください。同じ追加ネットワークを複数回指定した場合、Pod は複数のネットワークインターフェイスをそのネットワークに割り当てます。

- b. カスタマイズして追加のネットワークを割り当てるには、以下の形式でアノテーションを追加します。

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "<network>", 1
          "namespace": "<namespace>", 2
          "default-route": ["<default-route>"] 3
        }
      ]
```

- 1 **NetworkAttachmentDefinition** オブジェクトによって定義される追加のネットワークの名前を指定します。

- 2 **NetworkAttachmentDefinition** オブジェクトが定義される namespace を指定します。

- 3 オプション: **192.168.17.1** などのデフォルトルートのオーバーライドを指定します。

2. Pod を作成するには、以下のコマンドを入力します。**<name>** を Pod の名前に置き換えます。

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

- オプション: アノテーションが **Pod** CR に存在することを確認するには、**<name>** を Pod の名前に置き換えて、以下のコマンドを入力します。

```
$ oc get pod <name> -o yaml
```

以下の例では、**example-pod** Pod が追加ネットワークの **net1** に割り当てられています。

```
$ oc get pod example-pod -o yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-bridge
    k8s.v1.cni.cncf.io/networks-status: |- 1
    [
      {
        "name": "openshift-sdn",
        "interface": "eth0",
        "ips": [
          "10.128.2.14"
        ],
        "default": true,
        "dns": {}
      },{
        "name": "macvlan-bridge",
        "interface": "net1",
        "ips": [
          "20.2.2.100"
        ],
        "mac": "22:2f:60:a5:f8:00",
        "dns": {}
      }
    ]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

- k8s.v1.cni.cncf.io/networks-status** パラメーターは、オブジェクトの JSON 配列です。各オブジェクトは、Pod に割り当てられる追加のネットワークのステータスについて説明します。アノテーションの値はプレーンテキストの値として保存されます。

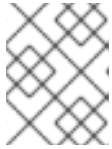
14.7.3. Non-Uniform Memory Access (NUMA) で配置された SR-IOV Pod の作成

NUMA で配置された SR-IOV Pod は、**restricted** または **single-numa-node** Topology Manager ポリシーで同じ NUMA ノードから割り当てられる SR-IOV および CPU リソースを制限することによって作成できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- CPU マネージャーのポリシーを **static** に設定している。CPU マネージャーの詳細は、関連情報セクションを参照してください。

- Topology Manager ポリシーを **single-numa-node** に設定している。



注記

single-numa-node が要求を満たさない場合は、Topology Manager ポリシーを **restricted** にするように設定できます。

手順

1. 以下の SR-IOV Pod 仕様を作成してから、YAML を **<name>-sriov-pod.yaml** ファイルに保存します。<name> をこの Pod の名前に置き換えます。
以下の例は、SR-IOV Pod 仕様を示しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: <name> ❶
spec:
  containers:
  - name: sample-container
    image: <image> ❷
    command: ["sleep", "infinity"]
    resources:
      limits:
        memory: "1Gi" ❸
        cpu: "2" ❹
      requests:
        memory: "1Gi"
        cpu: "2"
```

- ❶ <name> を、SR-IOV ネットワーク割り当て定義 CR の名前に置き換えます。
- ❷ <image> を **sample-pod** イメージの名前に置き換えます。
- ❸ Guaranteed QoS を指定して SR-IOV Pod を作成するには、**メモリー要求** に等しい **メモリー制限** を設定します。
- ❹ Guaranteed QoS を指定して SR-IOV Pod を作成するには、**cpu 要求** に等しい **cpu 制限** を設定します。

2. 以下のコマンドを実行して SR-IOV Pod のサンプルを作成します。

```
$ oc create -f <filename> ❶
```

- ❶ <filename> を、先の手順で作成したファイルの名前に置き換えます。

3. **sample-pod** が Guaranteed QoS を指定して設定されていることを確認します。

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

4. **sample-pod** が排他的 CPU を指定して割り当てられていることを確認します。

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

5. **sample-pod** に割り当てられる SR-IOV デバイスと CPU が同じ NUMA ノード上にあることを確認します。

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

14.7.4. 関連情報

- [SR-IOV イーサネットネットワーク割り当ての設定](#)
- [SR-IOV InfiniBand ネットワーク割り当ての設定](#)
- [CPU マネージャーの使用](#)

14.8. 高パフォーマンスのマルチキャストの使用

Single Root I/O Virtualization (SR-IOV) ハードウェアネットワーク上でマルチキャストを使用できません。

14.8.1. 高パフォーマンスのマルチキャスト

OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーは、デフォルトネットワーク上の Pod 間のマルチキャストをサポートします。これは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のアプリケーションには適していません。インターネットプロトコルテレビ (IPTV) やマルチポイントビデオ会議など、ストリーミングメディアなどのアプリケーションでは、Single Root I/O Virtualization (SR-IOV) ハードウェアを使用してネイティブに近いパフォーマンスを提供できます。

マルチキャストに追加の SR-IOV インターフェイスを使用する場合:

- マルチキャストパッケージは、追加の SR-IOV インターフェイス経由で Pod によって送受信される必要があります。
- SR-IOV インターフェイスに接続する物理ネットワークは、OpenShift Container Platform で制御されないマルチキャストルーティングとトポロジーを判別します。

14.8.2. マルチキャストでの SR-IOV インターフェイスの設定

以下の手順では、サンプルのマルチキャスト用の SR-IOV インターフェイスを作成します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

1. **SriovNetworkNodePolicy** オブジェクトを作成します。

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-example
  namespace: openshift-sriov-network-operator
spec:
  resourceName: example
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 4
  nicSelector:
    vendor: "8086"
    pfNames: ["ens803f0"]
    rootDevices: ["0000:86:00.0"]

```

2. SriovNetwork オブジェクトを作成します。

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: net-example
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: default
  ipam: | 1
    {
      "type": "host-local", 2
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [
        {"dst": "224.0.0.0/5"},
        {"dst": "232.0.0.0/5"}
      ],
      "gateway": "10.56.217.1"
    }
  resourceName: example

```

- 1** **2** DHCP を IPAM として設定する選択をした場合は、DHCP サーバー経由でデフォルトルート (**224.0.0.0/5** および **232.0.0.0/5**) をプロビジョニングするようにしてください。これにより、デフォルトのネットワークプロバイダーによって設定された静的なマルチキャストルートが上書きされます。

3. マルチキャストアプリケーションで Pod を作成します。

```

apiVersion: v1
kind: Pod
metadata:
  name: testpmd
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: nic1
spec:
  containers:

```

```
- name: example
  image: rhel7:latest
  securityContext:
    capabilities:
      add: ["NET_ADMIN"] ❶
  command: ["sleep", "infinity"]
```

- ❶ **NET_ADMIN** 機能は、アプリケーションがマルチキャスト IP アドレスを SR-IOV インターフェイスに割り当てる必要がある場合にのみ必要です。それ以外の場合は省略できます。

14.9. DPDK および RDMA の使用

コンテナ化された Data Plane Development Kit (DPDK) アプリケーションは OpenShift Container Platform でサポートされています。Single Root I/O Virtualization (SR-IOV) ネットワークハードウェアは、Data Plane Development Kit (DPDK) および Remote Direct Memory Access (RDMA) で利用できません。

対応しているデバイスの詳細は、[Supported devices](#) を参照してください。

14.9.1. NIC を使用した DPDK モードでの Virtual Function の使用

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **intel-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: intel-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intelnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "8086"
    deviceID: "158b"
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: vfio-pci ❶
```

- 1 Virtual Function (VF) のドライバータイプを **vfio-pci** に指定します。



注記

SriovNetworkNodePolicy の各オプションに関する詳細は、**Configuring SR-IOV network devices** セクションを参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分であることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

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

3. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **intel-dpdk-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: intel-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |-
    # ... 1
  vlan: <vlan>
  resourceName: intelnic
```

- 1 IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。



注記

SriovNetwork の各オプションに関する詳細は、SR-IOV の追加ネットワークの設定セクションを参照してください。

オプションのライブラリー **app-netutil** は、コンテナの親 Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

4. 以下のコマンドを実行して、**SriovNetwork** オブジェクトを作成します。

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

5. 以下の **Pod** 仕様を作成してから、YAML を **intel-dpdk-pod.yaml** ファイルに保存します。



```

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

```

- ❶ **SriovNetwork** オブジェクトの **intel-dpdk-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を Pod 仕様および **SriovNetwork** オブジェクトの両方で変更します。
- ❷ アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。
- ❸ hugepage の割り当て、システムリソースの割り当て、およびネットワークインターフェイスアクセス用のコンテナ内のアプリケーションに必要な追加機能を指定します。
- ❹ hugepage ボリュームを、**/dev/hugepages** の下にある DPDK Pod にマウントします。hugepage ボリュームは、メディアが **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。
- ❺ オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースインジェクターによって自動的に追加されます。SR-IOV ネットワークリソースインジェクターは、SR-IOV Operator によって管理される受付コントローラーコンポーネントです。これはデフォルトで有効にされており、デフォルト **SriovOperatorConfig** CR で **enableInjector** オプションを **false** に設定して無効にすることができます。

- 6 CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed QoS** を持つ Pod を作成して実行されます。
- 7 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。たとえば、カーネル引数 **default_hugepagesz=1GB**、**hugepagesz=1G** および **hugepages=16** を追加すると、**16*1Gi** hugepage がシステムの起動時に割り当てられます。

6. 以下のコマンドを実行して DPDK Pod を作成します。

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

14.9.2. Mellanox NIC を使用した DPDK モードでの Virtual Function の使用

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **mlx-dpdk-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-dpdk-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnic
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" 1
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice 2
  isRdma: true 3
```

- 1 SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。Mellanox カードに許可される値は **1015**、**1017** です。
- 2 Virtual Function (VF) のドライバータイプを **netdevice** に指定します。Mellanox SR-IOV

- RDMA モードを有効にします。これは、DPDK モードで機能するために Mellanox カードで必要とされます。



注記

SriovNetworkNodePolicy の各オプションに関する詳細は、**Configuring SR-IOV network devices** セクションを参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分であることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

- 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

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

- 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **mlx-dpdk-network.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-dpdk-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- ❶
  # ...
  vlan: <vlan>
  resourceName: mlxnic
```

- IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。



注記

SriovNetwork の各オプションに関する詳細は、SR-IOV の追加ネットワークの設定セクションを参照してください。

オプションのライブラリー `app-netutil` は、コンテナの親 Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

- 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

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

- 以下の **Pod** 仕様を作成してから、YAML を **mlx-dpdk-pod.yaml** ファイルに保存します。


```

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

```

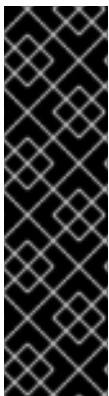
- ❶ **SriovNetwork** オブジェクトの **mlx-dpdk-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を Pod 仕様および **SriovNetwork** オブジェクトの両方で変更します。
- ❷ アプリケーションとアプリケーションが使用する DPDK ライブラリーが含まれる DPDK イメージを指定します。
- ❸ hugepage の割り当て、システムリソースの割り当て、およびネットワークインターフェイスアクセス用のコンテナ内のアプリケーションに必要な追加機能を指定します。
- ❹ hugepage ボリュームを、**/dev/hugepages** の下にある DPDK Pod にマウントします。hugepage ボリュームは、メディアが **Hugepages** に指定されている emptyDir ボリュームタイプでサポートされます。
- ❺ オプション: DPDK Pod に割り当てられる DPDK デバイスの数を指定します。このリソース要求および制限は、明示的に指定されていない場合、SR-IOV ネットワークリソースインジェクターによって自動的に追加されます。SR-IOV ネットワークリソースインジェクターは、SR-IOV Operator によって管理される受付コントローラーコンポーネントです。これはデフォルトで有効にされており、デフォルト **SriovOperatorConfig** CR で **enableInjector** オプションを **false** に設定して無効にすることができます。

- 6 CPU の数を指定します。DPDK Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed QoS**
- 7 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、DPDK Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。

6. 以下のコマンドを実行して DPDK Pod を作成します。

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

14.9.3. Mellanox NIC を使用した RDMA モードでの Virtual Function の使用



重要

RoCE (RDMA over Converged Ethernet) はテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

RoCE (RDMA over Converged Ethernet) は、OpenShift Container Platform で RDMA を使用する場合に唯一サポートされているモードです。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator をインストールします。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **mlx-rdma-node-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: mlx-rdma-node-policy
  namespace: openshift-sriov-network-operator
spec:
  resourceName: mlxnic
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
```

```

vendor: "15b3"
deviceID: "1015" ❶
pfNames: ["<pf_name>", ...]
rootDevices: ["<pci_bus_id>", "..."]
deviceType: netdevice ❷
isRdma: true ❸

```

- ❶ SR-IOV ネットワークデバイスのデバイス ID を指定します。Mellanox カードに許可される値は **1015**、**1017** です。
- ❷ Virtual Function (VF) のドライバータイプを **netdevice** に指定します。
- ❸ RDMA モードを有効にします。



注記

SriovNetworkNodePolicy の各オプションに関する詳細は、**Configuring SR-IOV network devices** セクションを参照してください。

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。設定の変更が適用されるまでに数分の時間がかかる場合があります。エビクトされたワークロードを処理するために、クラスター内に利用可能なノードが十分であることを前もって確認します。

設定の更新が適用された後に、**openshift-sriov-network-operator** namespace のすべての Pod が **Running** ステータスに変更されます。

2. 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

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

3. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **mlx-rdma-network.yaml** ファイルに保存します。

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: mlx-rdma-network
  namespace: openshift-sriov-network-operator
spec:
  networkNamespace: <target_namespace>
  ipam: |- ❶
# ...
  vlan: <vlan>
  resourceName: mlxnic

```

- ❶ IPAM CNI プラグインの設定オブジェクトを YAML ブロックスケーラーとして指定します。プラグインは、割り当て定義についての IP アドレスの割り当てを管理します。



注記

SriovNetwork の各オプションに関する詳細は、SR-IOV の追加ネットワークの設定セクションを参照してください。

オプションのライブラリー `app-netutil` は、コンテナの親 Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

- 以下のコマンドを実行して **SriovNetworkNodePolicy** オブジェクトを作成します。

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

- 以下の **Pod** 仕様を作成してから、YAML を **mlx-rdma-pod.yaml** ファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: rdma-app
  namespace: <target_namespace> ❶
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> ❷
    securityContext:
      runAsUser: 0
    capabilities:
      add: ["IPC_LOCK", "SYS_RESOURCE", "NET_RAW"] ❸
    volumeMounts:
    - mountPath: /dev/hugepages ❹
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "4" ❺
        hugepages-1Gi: "4Gi" ❻
      requests:
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
      command: ["sleep", "infinity"]
    volumes:
    - name: hugepage
      emptyDir:
        medium: HugePages
```

❶ **SriovNetwork** オブジェクトの **mlx-rdma-network** が作成される同じ **target_namespace** を指定します。Pod を異なる namespace に作成する場合、**target_namespace** を Pod 仕様および **SriovNetwork** オブジェクトの両方で変更します。

❷ アプリケーションとアプリケーションが使用する RDMA ライブラリーが含まれる RDMA イメージを指定します。

- 3 hugepage の割り当て、システムリソースの割り当て、およびネットワークインターフェイスアクセス用のコンテナ内のアプリケーションに必要な追加機能を指定します。
- 4 hugepage ボリュームを、`/dev/hugepages` の下にある RDMA Pod にマウントします。hugepage ボリュームは、メディアが **Hugepages** に指定されている `emptyDir` ボリュームタイプでサポートされます。
- 5 CPU の数を指定します。RDMA Pod には通常、kubelet から排他的 CPU を割り当てる必要があります。これは、CPU マネージャーポリシーを **static** に設定し、**Guaranteed** QoS を持つ Pod を作成して実行されます。
- 6 hugepage サイズ **hugepages-1Gi** または **hugepages-2Mi** を指定し、RDMA Pod に割り当てられる hugepage の量を指定します。**2Mi** および **1Gi** hugepage を別々に設定します。**1Gi** hugepage を設定するには、カーネル引数をノードに追加する必要があります。

6. 以下のコマンドを実行して RDMA Pod を作成します。

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

14.9.4. 関連情報

- [SR-IOV イーサネットネットワーク割り当ての設定](#)
- `app-netutil` ライブラリーは、コンテナの親 Pod に関するネットワーク情報を収集するための複数の API メソッドを提供します。

14.10. SR-IOV NETWORK OPERATOR のインストール

SR-IOV Network Operator をアンインストールするには、実行中の SR-IOV ワークロードをすべて削除し、Operator をアンインストールして、Operator が使用した Webhook を削除する必要があります。

14.10.1. SR-IOV Network Operator のインストール

クラスター管理者は、SR-IOV Network Operator をアンインストールできます。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- SR-IOV Network Operator がインストールされている。

手順

1. すべての SR-IOV カスタムリソース (CR) を削除します。

```
$ oc delete sriovnetwork -n openshift-sriov-network-operator --all
```

```
$ oc delete sriovnetworknodepolicy -n openshift-sriov-network-operator --all
```

```
$ oc delete sriovibnetwork -n openshift-sriov-network-operator --all
```

2. クラスターからの Operator の削除セクションに記載された手順に従い、クラスターから SR-IOV Network Operator を削除します。
3. SR-IOV Network Operator のアンインストール後にクラスターに残っている SR-IOV カスタムリソース定義を削除します。

```
$ oc delete crd sriovnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodepolicies.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodestates.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworkpoolconfigs.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovoperatorconfigs.sriovnetwork.openshift.io
```

4. SR-IOV Webhook を削除します。

```
$ oc delete mutatingwebhookconfigurations network-resources-injector-config
```

```
$ oc delete MutatingWebhookConfiguration sriov-operator-webhook-config
```

```
$ oc delete ValidatingWebhookConfiguration sriov-operator-webhook-config
```

5. SR-IOV Network Operator の namespace を削除します。

```
$ oc delete namespace openshift-sriov-network-operator
```

関連情報

- [クラスターからの Operator の削除](#)

第15章 OPENSIFT SDN デフォルト CNI ネットワークプロバイダー

15.1. OPENSIFT SDN デフォルト CNI ネットワークプロバイダーについて

OpenShift Container Platform は、Software Defined Networking (SDN) アプローチを使用して、クラスターのネットワークを統合し、OpenShift Container Platform クラスターの Pod 間の通信を可能にします。OpenShift SDN により、このような Pod ネットワークが確立され、メンテナンスされます。OpenShift SDN は Open vSwitch (OVS) を使用してオーバーレイネットワークを設定します。

15.1.1. OpenShift SDN ネットワーク分離モード

OpenShift SDN では以下のように、Pod ネットワークを設定するための SDN モードを 3 つ提供します。

- **ネットワークポリシーモード**は、プロジェクト管理者が **NetworkPolicy** オブジェクトを使用して独自の分離ポリシーを設定することを可能にします。ネットワークポリシーは、OpenShift Container Platform 4.9 のデフォルトモードです。
- **マルチテナントモード**は、Pod およびサービスのプロジェクトレベルの分離を可能にします。異なるプロジェクトの Pod は、別のプロジェクトの Pod およびサービスとパケットの送受信をすることができなくなります。プロジェクトの分離を無効にし、クラスター全体のすべての Pod およびサービスにネットワークトラフィックを送信したり、それらの Pod およびサービスからネットワークトラフィックを受信したりすることができます。
- **サブネットモード**は、すべての Pod が他のすべての Pod およびサービスと通信できる Pod ネットワークを提供します。ネットワークポリシーモードは、サブネットモードと同じ機能を提供します。

15.1.2. サポートされるデフォルトの CNI ネットワークプロバイダー機能マトリクス

OpenShift Container Platform は、OpenShift SDN と OVN-Kubernetes の 2 つのサポート対象のオプションをデフォルトの Container Network Interface (CNI) ネットワークプロバイダーに提供します。以下の表は、両方のネットワークプロバイダーの現在の機能サポートをまとめたものです。

表15.1 デフォルトの CNI ネットワークプロバイダー機能の比較

機能	OpenShift SDN	OVN-Kubernetes
Egress IP	サポート対象	サポート対象
Egress ファイアウォール [1]	サポート対象	サポート対象
Egress ルーター	サポート対象	サポート対象 [2]
IPsec 暗号化	サポート対象外	サポート対象
IPv6	サポート対象外	サポート対象 [3]
Kubernetes ネットワークポリシー	一部サポート対象 [4]	サポート対象

機能	OpenShift SDN	OVN-Kubernetes
Kubernetes ネットワークポリシーログ	サポート対象外	サポート対象
マルチキャスト	サポート対象	サポート対象

1. egress ファイアウォールは、OpenShift SDN では egress ネットワークポリシーとしても知られています。これはネットワークポリシーの egress とは異なります。
2. OVN-Kubernetes の egress ルーターはリダイレクトモードのみをサポートします。
3. IPv6 はベアメタルクラスターでのみサポートされます。
4. OpenShift SDN のネットワークポリシーは、egress ルールおよび一部の **ipBlock** ルールをサポートしません。

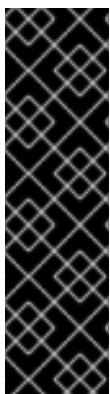
15.2. プロジェクトの EGRESS IP の設定

クラスター管理者は、OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーが1つ以上の egress IP アドレスをプロジェクトに割り当てるように設定できます。

15.2.1. プロジェクトの egress トラフィックについての egress IP アドレスの割り当て

プロジェクトの egress IP アドレスを設定することにより、指定されたプロジェクトからのすべての外部送信接続が同じ固定ソース IP アドレスを共有します。外部リソースは、egress IP アドレスに基づいて特定のプロジェクトからのトラフィックを認識できます。プロジェクトに割り当てられる egress IP アドレスは、トラフィックを特定の宛先に送信するために使用される egress ルーターとは異なります。

egress IP アドレスは、ノードのプライマリーネットワークインターフェイスの追加 IP アドレスとして実装され、ノードのプライマリー IP アドレスと同じサブネットにある必要があります。



重要

egress IP アドレスは、**ifcfg-eth0** などのように Linux ネットワーク設定ファイルで設定することはできません。

Amazon Web Services (AWS)、Google Cloud Platform (GCP)、および Azure の Egress IP は、OpenShift Container Platform バージョン 4.10 以降でのみサポートされます。

一部のクラウドまたは仮想マシンソリューションを使用する場合に、プライマリーネットワークインターフェイスで追加の IP アドレスを許可するには追加の設定が必要になる場合があります。

egress IP アドレスは、**NetNamespace** オブジェクトの **egressIPs** パラメーターを設定して namespace に割り当てることができます。egress IP がプロジェクトに関連付けられた後に、OpenShift SDN は 2 つの方法で Egress IP をホストに割り当てることができるようにします。

- **自動的に割り当てる** 方法では、egress IP アドレス範囲はノードに割り当てられます。
- **手動で割り当てる** 方法では、1つ以上の egress IP アドレスの一覧がノードに割り当てられます。

egress IP アドレスを要求する namespace は、それらの egress IP アドレスをホストできるノードに一致し、egress IP アドレスはそれらのノードに割り当てられます。**egressIPs** パラメーターが **NetNamespace** オブジェクトに設定されるものの、ノードがその egress IP アドレスをホストしない場合、namespace からの egress トラフィックはドロップされます。

ノードの高可用性は自動的に実行されます。egress IP アドレスをホストするノードが到達不可能であり、egress IP アドレスをホストできるノードがある場合、egress IP アドレスは新規ノードに移行します。到達不可能なノードが再びオンラインに戻ると、ノード間で egress IP アドレスのバランスを図るために egress IP アドレスは自動的に移行します。

重要

OpenShift SDN クラスターネットワークプロバイダーで egress IP アドレスを使用する場合、以下の制限が適用されます。

- 手動で割り当てられた egress IP アドレスと、自動的に割り当てられた egress IP アドレスは同じノードで使用することができません。
- IP アドレス範囲から egress IP アドレスを手動で割り当てる場合、その範囲を自動の IP 割り当てで利用可能にすることはできません。
- OpenShift SDN egress IP アドレス実装を使用して、複数の namespace で egress IP アドレスを共有することはできません。複数の namespace 間で IP アドレスを共有する必要がある場合は、OVN-Kubernetes クラスターネットワークプロバイダーの egress IP アドレスの実装により、複数の namespace で IP アドレスを共有できます。

注記

OpenShift SDN をマルチテナントモードで使用する場合、それらに関連付けられたプロジェクトによって別の namespace に参加している namespace と共に egress IP アドレスを使用することはできません。たとえば、**project1** および **project2** に **oc adm pod-network join-projects --to=project1 project2** コマンドを実行して参加している場合、どちらもプロジェクトも egress IP アドレスを使用できません。詳細は、[BZ#1645577](#) を参照してください。

15.2.1.1. 自動的に割り当てられた egress IP アドレスを使用する場合の考慮事項

egress IP アドレスの自動割り当て方法を使用する場合、以下の考慮事項が適用されます。

- 各ノードの **HostSubnet** リソースの **egressCIDRs** パラメーターを設定して、ノードでホストできる egress IP アドレスの範囲を指定します。OpenShift Container Platform は、指定する IP アドレス範囲に基づいて **HostSubnet** リソースの **egressIPs** パラメーターを設定します。

namespace の egress IP アドレスをホストするノードに到達できない場合、OpenShift Container Platform は互換性のある egress IP アドレス範囲を持つ別のノードに egress IP アドレスを再割り当てします。自動割り当て方法は、追加の IP アドレスをノードに関連付ける柔軟性のある環境にインストールされたクラスターに最も適しています。

15.2.1.2. 手動で割り当てられた egress IP アドレスを使用する場合の考慮事項

このアプローチは、パブリッククラウド環境など、追加の IP アドレスをノードに関連付ける際に制限がある可能性があるクラスターに使用されます。

egress IP アドレスに手動割り当て方法を使用する場合、以下の考慮事項が適用されます。

- 各ノードの **HostSubnet** リソースの **egressIPs** パラメーターを設定して、ノードでホストできる IP アドレスを指定します。
- namespace ごとに複数の egress IP アドレスがサポートされます。

namespace に複数の egress IP アドレスがあり、それらのアドレスが複数のノードでホストされる場合、以下の追加の考慮事項が適用されます。

- Pod が egress IP アドレスをホストするノード上にある場合、その Pod はノード上の egress IP アドレスを常に使用します。
- Pod が egress IP アドレスをホストするノードにない場合、その Pod はランダムで egress IP アドレスを使用します。

15.2.2. namespace の自動的に割り当てられた egress IP アドレスの有効化

OpenShift Container Platform では、1つ以上のノードで特定の namespace の egress IP アドレスの自動的な割り当てを有効にできます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 以下の JSON を使用して、**NetNamespace** オブジェクトを egress IP アドレスで更新します。

```
$ oc patch netnamespace <project_name> --type=merge -p \
{
  "egressIPs": [
    "<ip_address>"
  ]
}
```

ここでは、以下のようになります。

<project_name>

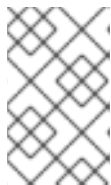
プロジェクトの名前を指定します。

<ip_address>

egressIPs 配列の1つ以上の egress IP アドレスを指定します。

たとえば、**project1** を IP アドレスの 192.168.1.100 に、**project2** を IP アドレスの 192.168.1.101 に割り当てるには、以下を実行します。

```
$ oc patch netnamespace project1 --type=merge -p \
{"egressIPs": ["192.168.1.100"]}
$ oc patch netnamespace project2 --type=merge -p \
{"egressIPs": ["192.168.1.101"]}
```



注記

OpenShift SDN は **NetNamespace** オブジェクトを管理するため、既存の **NetNamespace** オブジェクトを変更することによってのみ変更を加えることができます。新規 **NetNamespace** オブジェクトは作成しません。

- 以下の JSON を使用して、各ホストの **egressCIDRs** パラメーターを設定して egress IP アドレスをホストできるノードを示します。

```
$ oc patch hostsubnet <node_name> --type=merge -p \
{
  "egressCIDRs": [
    "<ip_address_range>", "<ip_address_range>"
  ]
}
```

ここでは、以下ようになります。

<node_name>

ノード名を指定します。

<ip_address_range>

CIDR 形式の IP アドレス範囲を指定します。 **egressCIDRs** 配列に複数のアドレス範囲を指定できます。

たとえば、 **node1** および **node2** を、192.168.1.0 から 192.168.1.255 の範囲で egress IP アドレスをホストするように設定するには、以下を実行します。

```
$ oc patch hostsubnet node1 --type=merge -p \
{"egressCIDRs": ["192.168.1.0/24"]}
$ oc patch hostsubnet node2 --type=merge -p \
{"egressCIDRs": ["192.168.1.0/24"]}
```

OpenShift Container Platform はバランスを取りながら特定の egress IP アドレスを利用可能なノードに自動的に割り当てます。この場合、egress IP アドレス 192.168.1.100 を **node1** に、egress IP アドレス 192.168.1.101 を **node2** に割り当て、その逆も行います。

15.2.3. namespace の手動で割り当てられた egress IP アドレスの設定

OpenShift Container Platform で、1つ以上の egress IP アドレスを namespace に関連付けることができます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

- 以下の JSON オブジェクトを必要な IP アドレスで指定して、 **NetNamespace** オブジェクトを更新します。

```
$ oc patch netnamespace <project_name> --type=merge -p \
```

```
{
  "egressIPs": [
    "<ip_address>"
  ]
}
```

ここでは、以下のようになります。

<project_name>

プロジェクトの名前を指定します。

<ip_address>

egressIPs 配列の1つ以上の egress IP アドレスを指定します。

たとえば、**project1** プロジェクトを IP アドレス **192.168.1.100** および **192.168.1.101** に割り当てるには、以下を実行します。

```
$ oc patch netnamespace project1 --type=merge \
  -p '{"egressIPs": ["192.168.1.100","192.168.1.101"]}'
```

高可用性を提供するには、**egressIPs** の値を異なるノードの2つ以上の IP アドレスに設定します。複数の egress IP アドレスが設定されている場合、Pod はすべての egress IP アドレスをほぼ均等に使用します。



注記

OpenShift SDN は **NetNamespace** オブジェクトを管理するため、既存の **NetNamespace** オブジェクトを変更することによってのみ変更を加えることができます。新規 **NetNamespace** オブジェクトは作成しません。

2. egress IP をノードホストに手動で割り当てます。**egressIPs** パラメーターを、ノードホストの **HostSubnet** オブジェクトに設定します。以下の JSON を使用して、そのノードホストに割り当てる必要のある任意の数の IP アドレスを含めることができます。

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

ここでは、以下のようになります。

<node_name>

ノード名を指定します。

<ip_address>

IP アドレスを指定します。**egressIPs** 配列に複数の IP アドレスを指定できます。

たとえば、**node1** に Egress IP **192.168.1.100**、**192.168.1.101**、および **192.168.1.102** が設定されるように指定するには、以下を実行します。

```
$ oc patch hostsubnet node1 --type=merge -p \
  '{"egressIPs": ["192.168.1.100", "192.168.1.101", "192.168.1.102"]}'
```

直前の例では、**project1** のすべての egress トラフィックは、指定された egress IP をホストするノードにルーティングされてから、その IP アドレスに Network Address Translation (NAT) を使用して接続されます。

15.3. プロジェクトの EGRESS ファイアウォールの設定

クラスター管理者は、OpenShift Container Platform クラスター外に出るプロジェクトのプロジェクトについて、egress トラフィックを制限する egress ファイアウォールを作成できます。

15.3.1. egress ファイアウォールのプロジェクトでの機能

クラスター管理者は、**egress ファイアウォール** を使用して、一部またはすべての Pod がクラスター内からアクセスできる外部ホストを制限できます。egress ファイアウォールポリシーは以下のシナリオをサポートします。

- Pod の接続を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
- Pod の接続をパブリックインターネットに制限し、OpenShift Container Platform クラスター外にある内部ホストへの接続を開始できないようにする。
- Pod は OpenShift Container Platform クラスター外の指定された内部サブネットまたはホストにアクセスできません。
- Pod は特定の外部ホストにのみ接続することができます。

たとえば、指定された IP 範囲へのあるプロジェクトへのアクセスを許可する一方で、別のプロジェクトへの同じアクセスを拒否することができます。または、アプリケーション開発者の (Python) pip mirror からの更新を制限したり、更新を承認されたソースからの更新のみに強制的に制限したりすることができます。



注記

Egress ファイアウォールは、ホストネットワークの namespace には適用されません。ホストネットワークが有効になっている Pod は、Egress ファイアウォールルールの影響を受けません。

EgressNetworkPolicy カスタムリソース (CR) オブジェクトを作成して egress ファイアウォールポリシーを設定します。egress ファイアウォールは、以下のいずれかの基準を満たすネットワークトラフィックと一致します。

- CIDR 形式の IP アドレス範囲。
- IP アドレスに解決する DNS 名

重要

egress ファイアウォールに **0.0.0.0/0** の拒否ルールが含まれる場合、OpenShift Container Platform API サーバーへのアクセスはブロックされます。Pod が OpenShift Container Platform API サーバーへのアクセスを継続できるようにするには、以下の例にあるように API サーバーが egress ファイアウォールルールでリッスンする IP アドレス範囲を含める必要があります。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
  namespace: <namespace> ❶
spec:
  egress:
    - to:
      cidrSelector: <api_server_address_range> ❷
      type: Allow
  # ...
  - to:
    cidrSelector: 0.0.0.0/0 ❸
    type: Deny
```

- ❶ egress ファイアウォールの namespace。
- ❷ OpenShift Container Platform API サーバーを含む IP アドレス範囲。
- ❸ グローバル拒否ルールにより、OpenShift Container Platform API サーバーへのアクセスが阻止されます。

API サーバーの IP アドレスを見つけるには、**oc get ep kubernetes -n default** を実行します。

詳細は、[BZ#1988324](#) を参照してください。

重要

egress ファイアウォールを設定するには、ネットワークポリシーまたはマルチテナントモードのいずれかを使用するように OpenShift SDN を設定する必要があります。

ネットワークポリシーモードを使用している場合、egress ファイアウォールは namespace ごとに1つのポリシーとのみ互換性を持ち、グローバルプロジェクトなどのネットワークを共有するプロジェクトでは機能しません。



警告

egress ファイアウォールルールは、ルーターを通過するトラフィックには適用されません。ルート CR オブジェクトを作成するパーミッションを持つユーザーは、禁止されている宛先を参照するルートを作成することにより、egress ファイアウォールポリシールールをバイパスできます。

15.3.1.1. egress ファイアウォールの制限

egress ファイアウォールには以下の制限があります。

- いずれのプロジェクトも複数の EgressNetworkPolicy オブジェクトを持つことができません。
- 最大 1,000 のルールを持つ最大 1 つの EgressNetworkPolicy オブジェクトはプロジェクトごとに定義できます。
- **default** プロジェクトは egress ファイアウォールを使用できません。
- マルチテナントモードで OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーを使用する場合、以下の制限が適用されます。
 - グローバルプロジェクトは egress ファイアウォールを使用できません。 **oc adm pod-network make-projects-global** コマンドを使用して、プロジェクトをグローバルにすることができます。
 - **oc adm pod-network join-projects** コマンドを使用してマージされるプロジェクトでは、結合されたプロジェクトのいずれでも egress ファイアウォールを使用することはできません。

上記の制限のいずれかに違反すると、プロジェクトの egress ファイアウォールに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

egress ファイアウォールリソースは、**kube-node-lease**、**kube-public**、**kube-system**、**openshift**、**openshift-** プロジェクトで作成できます。

15.3.1.2. egress ポリシールールのマッチング順序

egress ファイアウォールポリシールールは、最初から最後へと定義された順序で評価されます。Pod からの egress 接続に一致する最初のルールが適用されます。この接続では、後続のルールは無視されます。

15.3.1.3. DNS (Domain Name Server) 解決の仕組み

egress ファイアウォールポリシールールのいずれかで DNS 名を使用する場合、ドメイン名の適切な解決には、以下の制限が適用されます。

- ドメイン名の更新は、TTL (Time-to-live) 期間に基づいてポーリングされます。デフォルトの期間は 30 秒です。egress ファイアウォールコントローラーがローカルネームサーバーでドメイン名をクエリーする場合に、応答に 30 秒未満の TTL が含まれる場合は、コントローラーはその期間を返される値に設定します。応答の TTL が 30 分を超える場合、コントローラーは期間を 30 分に設定します。TTL が 30 秒から 30 分の間に設定される場合、コントローラーは値を無視し、期間を 30 秒に設定します。
- Pod は、必要に応じて同じローカルネームサーバーからドメインを解決する必要があります。そうしない場合、egress ファイアウォールコントローラーと Pod によって認識されるドメインの IP アドレスが異なる可能性があります。ホスト名の IP アドレスが異なる場合、egress ファイアウォールは一貫して実行されないことがあります。
- egress ファイアウォールコントローラーおよび Pod は同じローカルネームサーバーを非同期にポーリングするため、Pod は egress コントローラーが実行する前に更新された IP アドレスを取得する可能性があります。これにより、競合状態が生じます。この現時点の制限により、EgressNetworkPolicy オブジェクトのドメイン名の使用は、IP アドレスの変更が頻繁に生じないドメインの場合にのみ推奨されます。



注記

egress ファイアウォールは、DNS 解決用に Pod が置かれるノードの外部インターフェイスに Pod が常にアクセスできるようにします。

ドメイン名を egress ファイアウォールで使用し、DNS 解決がローカルノード上の DNS サーバーによって処理されない場合は、Pod でドメイン名を使用している場合には DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールを追加する必要があります。

15.3.2. EgressNetworkPolicy カスタムリソース (CR) オブジェクト

egress ファイアウォールのルールを 1 つ以上定義できます。ルールは、ルールが適用されるトラフィックを指定して **Allow** ルールまたは **Deny** ルールのいずれかになります。

以下の YAML は EgressNetworkPolicy CR オブジェクトについて説明しています。

EgressNetworkPolicy オブジェクト

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: <name> ❶
spec:
  egress: ❷
  ...
```

- ❶ egress ファイアウォールポリシーの名前。
- ❷ 以下のセクションで説明されているように、egress ネットワークポリシールールのコレクション。

15.3.2.1. EgressNetworkPolicy ルール

以下の YAML は egress ファイアウォールルールオブジェクトについて説明しています。**egress** スタンザは、単一または複数のオブジェクトの配列を予想します。

Egress ポリシールールのスタンザ

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

- ❶ ルールのタイプ。値には **Allow** または **Deny** のいずれかを指定する必要があります。
- ❷ egress トラフィックのマッチングルールを記述するスタンザ。ルールの **cidrSelector** フィールドまたは **dnsName** フィールドのいずれかの値。同じルールで両方のフィールドを使用することはできません。
- ❸ CIDR 形式の IP アドレス範囲。

4 ドメイン名。

15.3.2.2. EgressNetworkPolicy CR オブジェクトの例

以下の例では、複数の egress ファイアウォールポリシールールを定義します。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
spec:
  egress: 1
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Allow
    to:
      dnsName: www.example.com
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 egress ファイアウォールポリシールールオブジェクトのコレクション。

15.3.3. egress ファイアウォールポリシーオブジェクトの作成

クラスター管理者は、プロジェクトの egress ファイアウォールポリシーオブジェクトを作成できません。



重要

プロジェクトに EgressNetworkPolicy オブジェクトがすでに定義されている場合、既存のポリシーを編集して egress ファイアウォールルールを変更する必要があります。

前提条件

- OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者としてクラスターにログインする必要があります。

手順

1. ポリシールールを作成します。
 - a. **<policy_name>.yaml** ファイルを作成します。この場合、**<policy_name>** は egress ポリシールールを記述します。
 - b. 作成したファイルで、egress ポリシーオブジェクトを定義します。

- 以下のコマンドを入力してポリシーオブジェクトを作成します。<policy_name> をポリシーの名前に、<project> をルールが適用されるプロジェクトに置き換えます。

```
$ oc create -f <policy_name>.yaml -n <project>
```

以下の例では、新規の EgressNetworkPolicy オブジェクトが **project1** という名前のプロジェクトに作成されます。

```
$ oc create -f default.yaml -n project1
```

出力例

```
egressnetworkpolicy.network.openshift.io/v1 created
```

- オプション: 後に変更できるように <policy_name>.yaml ファイルを保存します。

15.4. プロジェクトの EGRESS ファイアウォールの編集

クラスター管理者は、既存の egress ファイアウォールのネットワークトラフィックルールを変更できます。

15.4.1. EgressNetworkPolicy オブジェクトの表示

クラスターで EgressNetworkPolicy オブジェクトを表示できます。

前提条件

- OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- oc** として知られる OpenShift コマンドラインインターフェイス (CLI) のインストール。
- クラスターにログインすること。

手順

- オプション: クラスターで定義された EgressNetworkPolicy オブジェクトの名前を表示するには、以下のコマンドを入力します。

```
$ oc get egressnetworkpolicy --all-namespaces
```

- ポリシーを検査するには、以下のコマンドを入力します。<policy_name> を検査するポリシーの名前に置き換えます。

```
$ oc describe egressnetworkpolicy <policy_name>
```

出力例

```
Name: default
Namespace: project1
Created: 20 minutes ago
Labels: <none>
```

```
Annotations: <none>
Rule: Allow to 1.2.3.0/24
Rule: Allow to www.example.com
Rule: Deny to 0.0.0.0/0
```

15.5. プロジェクトの EGRESS ファイアウォールの編集

クラスター管理者は、既存の egress ファイアウォールのネットワークトラフィックルールを変更できます。

15.5.1. EgressNetworkPolicy オブジェクトの編集

クラスター管理者は、プロジェクトの egress ファイアウォールを更新できます。

前提条件

- OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者としてクラスターにログインする必要があります。

手順

1. プロジェクトの EgressNetworkPolicy オブジェクトの名前を検索します。<project> をプロジェクトの名前に置き換えます。

```
$ oc get -n <project> egressnetworkpolicy
```

2. オプション: egress ネットワークファイアウォールの作成時に EgressNetworkPolicy オブジェクトのコピーを保存しなかった場合には、以下のコマンドを入力してコピーを作成します。

```
$ oc get -n <project> egressnetworkpolicy <name> -o yaml > <filename>.yaml
```

<project> をプロジェクトの名前に置き換えます。<name> をオブジェクトの名前に置き換えます。<filename> をファイルの名前に置き換え、YAML を保存します。

3. ポリシールールに変更を加えたら、以下のコマンドを実行して EgressNetworkPolicy オブジェクトを置き換えます。<filename> を、更新された EgressNetworkPolicy オブジェクトを含むファイルの名前に置き換えます。

```
$ oc replace -f <filename>.yaml
```

15.6. プロジェクトからの EGRESS ファイアウォールの削除

クラスター管理者は、プロジェクトから egress ファイアウォールを削除して、OpenShift Container Platform クラスター外に出るプロジェクトからネットワークトラフィックについてのすべての制限を削除できます。

15.6.1. EgressNetworkPolicy オブジェクトの削除

クラスター管理者は、プロジェクトから Egress ファイアウォールを削除できます。

前提条件

- OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者としてクラスターにログインする必要があります。

手順

1. プロジェクトの EgressNetworkPolicy オブジェクトの名前を検索します。<project> をプロジェクトの名前に置き換えます。

```
$ oc get -n <project> egressnetworkpolicy
```

2. 以下のコマンドを入力し、EgressNetworkPolicy オブジェクトを削除します。<project> をプロジェクトの名前に、<name> をオブジェクトの名前に置き換えます。

```
$ oc delete -n <project> egressnetworkpolicy <name>
```

15.7. EGRESS ルーター POD の使用についての考慮事項

15.7.1. egress ルーター Pod について

OpenShift Container Platform egress ルーター Pod は、他の用途で使用されていないプライベートソース IP アドレスから指定されたリモートサーバーにトラフィックをリダイレクトします。Egress ルーター Pod により、特定の IP アドレスからのアクセスのみを許可するように設定されたサーバーにネットワークトラフィックを送信できます。



注記

egress ルーター Pod はすべての発信接続のために使用されることが意図されていません。多数の egress ルーター Pod を作成することで、ネットワークハードウェアの制限を引き上げられる可能性があります。たとえば、すべてのプロジェクトまたはアプリケーションに egress ルーター Pod を作成すると、ソフトウェアの MAC アドレスのフィルタリングに戻る前にネットワークインターフェイスが処理できるローカル MAC アドレス数の上限を超えてしまう可能性があります。



重要

egress ルーターイメージには Amazon AWS, Azure Cloud またはレイヤー 2 操作をサポートしない他のクラウドプラットフォームとの互換性がありません。それらに macvlan トラフィックとの互換性がないためです。

15.7.1.1. Egress ルーターモード

リダイレクトモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから1つ以上の宛先 IP アドレスにリダイレクトするために **iptables** ルールをセットアップします。予約されたソース IP アドレスを使用する必要があるクライアント Pod は、宛先 IP に直接接続するのではなく、egress ルーターに接続するように変更される必要があります。

HTTP プロキシモードでは、egress ルーター Pod はポート **8080** で HTTP プロキシとして実行され

ます。このモードは、HTTP または HTTPS ベースのサービスと通信するクライアントの場合にのみ機能しますが、通常それらを機能させるのにクライアント Pod への多くの変更は不要です。環境変数を設定することで、数多くのプログラムは HTTP プロキシを使用するように指示されます。

DNS プロキシモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから1つ以上の宛先 IP アドレスに送信する TCP ベースのサービスの DNS プロキシとして実行されます。予約されたソース IP アドレスを使用するには、クライアント Pod は、宛先 IP アドレスに直接接続するのではなく、egress ルーター Pod に接続するように変更される必要があります。この修正により、外部の宛先でトラフィックが既知のソースから送信されているかのように処理されます。

リダイレクトモードは、HTTP および HTTPS 以外のすべてのサービスで機能します。HTTP および HTTPS サービスの場合は、HTTP プロキシモードを使用します。IP アドレスまたはドメイン名を持つ TCP ベースのサービスの場合は、DNS プロキシモードを使用します。

15.7.1.2. egress ルーター Pod の実装

egress ルーター Pod の設定は、初期化コンテナで実行されます。このコンテナは特権付きコンテナキストで実行され、macvlan インターフェイスを設定して **iptables** ルールを設定できます。初期化コンテナが **iptables** ルールの設定を終了すると、終了します。次に、egress ルーター Pod はコンテナを実行して egress ルーターのトラフィックを処理します。使用されるイメージは、egress ルーターモードによって異なります。

環境変数は、egress-router イメージが使用するアドレスを判別します。イメージは macvlan インターフェイスを、**EGRESS_SOURCE** をその IP アドレスとして使用し、**EGRESS_GATEWAY** をゲートウェイの IP アドレスとして使用するよう設定します。

ネットワークアドレス変換 (NAT) ルールは、TCP ポートまたは UDP ポート上の Pod のクラスター IP アドレスへの接続が **EGRESS_DESTINATION** 変数で指定される IP アドレスの同じポートにリダイレクトされるよう設定されます。

クラスター内の一部のノードのみが指定されたソース IP アドレスを要求でき、指定されたゲートウェイを使用できる場合、受け入れ可能なノードを示す **nodeName** または **nodeSelector** を指定することができます。

15.7.1.3. デプロイメントに関する考慮事項

egress ルーター Pod は追加の IP アドレスおよび MAC アドレスをノードのプライマリーネットワークインターフェイスに追加します。その結果、ハイパーバイザーまたはクラウドプロバイダーを、追加のアドレスを許可するよう設定する必要がある場合があります。

Red Hat OpenStack Platform (RHOSP)

OpenShift Container Platform を RHOSP にデプロイする場合、OpenStack 環境の egress ルーター Pod の IP および MAC アドレスからのトラフィックを許可する必要があります。トラフィックを許可しないと、**通信は失敗** します。

```
$ openstack port set --allowed-address \
  ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

Red Hat Virtualization (RHV)

RHV を使用している場合は、仮想インターフェイスカード (vNIC) に **No Network Filter** を選択する必要があります。

VMware vSphere

VMware vSphere を使用している場合は、[vSphere 標準スイッチのセキュリティー保護についての VMware ドキュメント](#) を参照してください。vSphere Web クライアントからホストの仮想スイッチを選択して、VMware vSphere デフォルト設定を表示し、変更します。

とくに、以下が有効にされていることを確認します。

- [MAC アドレスの変更](#)
- [偽装転送 \(Forged Transit\)](#)
- [無作為別モード \(Promiscuous Mode\) 操作](#)

15.7.1.4. フェイルオーバー設定

ダウンタイムを回避するには、以下の例のように **Deployment** リソースで egress ルーター Pod をデプロイできます。サンプルのデプロイメント用に新規 **Service** オブジェクトを作成するには、**oc expose deployment/egress-demo-controller** コマンドを使用します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: egress-demo-controller
spec:
  replicas: 1 1
  selector:
    matchLabels:
      name: egress-router
  template:
    metadata:
      name: egress-router
    labels:
      name: egress-router
    annotations:
      pod.network.openshift.io/assign-macvlan: "true"
  spec: 2
    initContainers:
      ...
    containers:
      ...
```

1 1つの Pod のみが指定される egress ソース IP アドレスを使用できるため、レプリカが **1** に設定されていることを確認します。これは、単一コピーのルーターのみがノード実行されることを意味します。

2 egress ルーター Pod の **Pod** オブジェクトテンプレートを指定します。

15.7.2. 関連情報

- [リダイレクトモードでの egress ルーターのデプロイ](#)
- [HTTP プロキシモードでの egress ルーターのデプロイ](#)
- [DNS プロキシモードでの egress ルーターのデプロイ](#)

15.8. リダイレクトモードでの EGRESS ルーター POD のデプロイ

クラスター管理者は、トラフィックを指定された宛先 IP アドレスにリダイレクトするように設定された egress ルーター Pod をデプロイできます。

15.8.1. リダイレクトモードの egress ルーター Pod 仕様

Pod オブジェクトで egress ルーター Pod の設定を定義します。以下の YAML は、リダイレクトモードでの egress ルーター Pod の設定のフィールドについて説明しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
  env:
  - name: EGRESS_SOURCE ❷
    value: <egress_router>
  - name: EGRESS_GATEWAY ❸
    value: <egress_gateway>
  - name: EGRESS_DESTINATION ❹
    value: <egress_destination>
  - name: EGRESS_ROUTER_MODE
    value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift4/ose-pod
```

❶ このアノテーションは、OpenShift Container Platform に対して、プライマリーネットワークインターフェイスコントローラー (NIC) に macvlan ネットワークインターフェイスを作成し、その macvlan インターフェイスを Pod ネットワークの namespace に移動するよう指示します。引用符を **"true"** 値の周囲に含める必要があります。OpenShift Container Platform が別の NIC インターフェイスに macvlan インターフェイスを作成するには、アノテーションの値をそのインターフェイスの名前に設定します。たとえば、**eth1** を使用します。

❷ ノードが置かれている物理ネットワークの IP アドレスは egress ルーター Pod で使用するために予約されます。オプション: サブネットの長さ /24 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

❸ ノードで使用されるデフォルトゲートウェイと同じ値です。

❹ トラフィックの送信先となる外部サーバー。この例では、Pod の接続は **203.0.113.25** にリダイレクトされます。ソース IP アドレスは **192.168.12.99** です。

egress ルーター Pod 仕様の例

```

apiVersion: v1
kind: Pod
metadata:
  name: egress-multi
  labels:
    name: egress-multi
  annotations:
    pod.network.openshift.io/assign-macvlan: "true"
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
  env:
  - name: EGRESS_SOURCE
    value: 192.168.12.99/24
  - name: EGRESS_GATEWAY
    value: 192.168.12.1
  - name: EGRESS_DESTINATION
    value: |
      80 tcp 203.0.113.25
      8080 tcp 203.0.113.26 80
      8443 tcp 203.0.113.26 443
      203.0.113.27
  - name: EGRESS_ROUTER_MODE
    value: init
  containers:
  - name: egress-router-wait
    image: registry.redhat.io/openshift4/ose-pod

```

15.8.2. egress 宛先設定形式

egress ルーター Pod がリダイレクトモードでデプロイされる場合、以下のいずれかの形式を使用してリダイレクトルールを指定できます。

- **<port> <protocol> <ip_address>**: 指定される **<port>** への着信接続が指定される **<ip_address>** の同じポートにリダイレクトされます。 **<protocol>** は **tcp** または **udp** のいずれかになります。
- **<port> <protocol> <ip_address> <remote_port>**: 接続が **<ip_address>** の別の **<remote_port>** にリダイレクトされるのを除き、上記と同じになります。
- **<ip_address>**: 最後の行が単一 IP アドレスである場合、それ以外のポートの接続はその IP アドレスの対応するポートにリダイレクトされます。フォールバック IP アドレスがない場合、他のポートでの接続は拒否されます。

以下の例では、複数のルールが定義されます。

- 最初の行はローカルポート **80** から **203.0.113.25** のポート **80** にトラフィックをリダイレクトします。

- 2番目と3番目の行では、ローカルポート **8080** および **8443** を、**203.0.113.26** のリモートポート **80** および **443** にリダイレクトします。
- 最後の行は、先のルールで指定されていないポートのトラフィックに一致します。

設定例

```
80 tcp 203.0.113.25
8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443
203.0.113.27
```

15.8.3. リダイレクトモードでの egress ルーター Pod のデプロイ

リダイレクトモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから1つ以上の宛先 IP アドレスにリダイレクトするために iptables ルールをセットアップします。予約されたソース IP アドレスを使用する必要があるクライアント Pod は、宛先 IP に直接接続するのではなく、egress ルーターに接続するように変更される必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. egress ルーター Pod の作成
2. 他の Pod が egress ルーター Pod の IP アドレスを見つられるようにするには、以下の例のように、egress ルーター Pod を参照するサービスを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
    - name: http
      port: 80
    - name: https
      port: 443
  type: ClusterIP
  selector:
    name: egress-1
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにリダイレクトされます。

15.8.4. 関連情報

- [ConfigMap を使用した egress ルーターの宛先マッピングの設定](#)

15.9. HTTP プロキシモードでの EGRESS ルーター POD のデプロイ

クラスター管理者は、トラフィックを指定された HTTP および HTTPS ベースのサービスにプロキシするように設定された egress ルーター Pod をデプロイできます。

15.9.1. HTTP モードの egress ルーター Pod 仕様

Pod オブジェクトで egress ルーター Pod の設定を定義します。以下の YAML は、HTTP モードでの egress ルーター Pod の設定のフィールドについて説明しています。

```
apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
  - name: EGRESS_SOURCE ❷
    value: <egress-router>
  - name: EGRESS_GATEWAY ❸
    value: <egress-gateway>
  - name: EGRESS_ROUTER_MODE
    value: http-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-http-proxy
    env:
  - name: EGRESS_HTTP_PROXY_DESTINATION ❹
    value: |-
      ...
      ...
```

❶ このアノテーションは、OpenShift Container Platform に対して、プライマリーネットワークインターフェイスコントローラー (NIC) に macvlan ネットワークインターフェイスを作成し、その macvlan インターフェイスを Pod ネットワークの namespace に移動するよう指示します。引用符を **"true"** 値の周囲に含める必要があります。OpenShift Container Platform が別の NIC インターフェイスに macvlan インターフェイスを作成するには、アノテーションの値をそのインターフェイスの名前に設定します。たとえば、**eth1** を使用します。

❷ ノードが置かれている物理ネットワークの IP アドレスは egress ルーター Pod で使用するために予約されます。オプション: サブネットの長さ /24 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。

❸ ノードで使用されるデフォルトゲートウェイと同じ値です。

❹ プロキシの設定方法を指定する文字列または YAML の複数行文字列です。これは、init コンテナの他の環境変数ではなく、HTTP プロキシコンテナの環境変数として指定されることに注意してください。

思しくください。

15.9.2. egress 宛先設定形式

egress ルーター Pod が HTTP プロキシモードでデプロイされる場合、以下の形式のいずれかを使用してリダイレクトルールを指定できます。これはすべてのリモート宛先への接続を許可することを意味します。設定の各行には、許可または拒否する接続の1つのグループを指定します。

- IP アドレス (例: **192.168.1.1**) は該当する IP アドレスへの接続を許可します。
- CIDR 範囲 (例: **192.168.1.0/24**) は CIDR 範囲への接続を許可します。
- ホスト名 (例: **www.example.com**) は該当ホストへのプロキシを許可します。
- * が前に付けられているドメイン名 (例: ***.example.com**) は該当ドメインおよびそのサブドメインのすべてへのプロキシを許可します。
- 先的一致 (match) 式のいずれかの後に来る ! は接続を拒否します。
- 最後の行が * の場合、明示的に拒否されていないすべてのものが許可されます。それ以外の場合、許可されないすべてのものが拒否されます。

* を使用してすべてのリモート宛先への接続を許可することもできます。

設定例

```
!*example.com
!192.168.1.0/24
192.168.2.1
*
```

15.9.3. HTTP プロキシモードでの egress ルーター Pod のデプロイ

HTTP プロキシモードでは、egress ルーター Pod はポート **8080** で HTTP プロキシとして実行されます。このモードは、HTTP または HTTPS ベースのサービスと通信するクライアントの場合にのみ機能しますが、通常それらを機能させるのにクライアント Pod への多くの変更は不要です。環境変数を設定することで、数多くのプログラムは HTTP プロキシを使用するように指示されます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. egress ルーター Pod の作成
2. 他の Pod が egress ルーター Pod の IP アドレスを見つられるようにするには、以下の例のように、egress ルーター Pod を参照するサービスを作成します。

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

```

name: egress-1
spec:
  ports:
  - name: http-proxy
    port: 8080 ❶
    type: ClusterIP
  selector:
    name: egress-1

```

- ❶ **http** ポートが常に **8080** に設定されていることを確認します。

3. **http_proxy** または **https_proxy** 変数を設定して、クライアント Pod (egress プロキシ Pod ではない) を HTTP プロキシを使用するように設定します。

```

apiVersion: v1
kind: Pod
metadata:
  name: app-1
  labels:
    name: app-1
spec:
  containers:
  env:
  - name: http_proxy
    value: http://egress-1:8080/ ❶
  - name: https_proxy
    value: http://egress-1:8080/
  ...

```

- ❶ 先の手順で作成したサービス。



注記

すべてのセットアップに **http_proxy** および **https_proxy** 環境変数が必要になる訳ではありません。上記を実行しても作業用セットアップが作成されない場合は、Pod で実行しているツールまたはソフトウェアについてのドキュメントを参照してください。

15.9.4. 関連情報

- [ConfigMap を使用した egress ルーターの宛先マッピングの設定](#)

15.10. DNS プロキシモードでの EGRESS ルーター POD のデプロイ

クラスター管理者は、トラフィックを指定された DNS 名および IP アドレスにプロキシするように設定された egress ルーター Pod をデプロイできます。

15.10.1. DNS モードの egress ルーター Pod 仕様

Pod オブジェクトで egress ルーター Pod の設定を定義します。以下の YAML は、DNS モードでの egress ルーター Pod の設定のフィールドについて説明しています。

```

apiVersion: v1
kind: Pod
metadata:
  name: egress-1
  labels:
    name: egress-1
  annotations:
    pod.network.openshift.io/assign-macvlan: "true" ❶
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
  - name: EGRESS_SOURCE ❷
    value: <egress-router>
  - name: EGRESS_GATEWAY ❸
    value: <egress-gateway>
  - name: EGRESS_ROUTER_MODE
    value: dns-proxy
  containers:
  - name: egress-router-pod
    image: registry.redhat.io/openshift4/ose-egress-dns-proxy
    securityContext:
      privileged: true
    env:
  - name: EGRESS_DNS_PROXY_DESTINATION ❹
    value: |-
      ...
  - name: EGRESS_DNS_PROXY_DEBUG ❺
    value: "1"
  ...

```

- ❶ このアノテーションは、OpenShift Container Platform に対して、プライマリーネットワークインターフェイスコントローラー (NIC) に macvlan ネットワークインターフェイスを作成し、その macvlan インターフェイスを Pod ネットワークの namespace に移動するよう指示します。引用符を **"true"** 値の周囲に含める必要があります。OpenShift Container Platform が別の NIC インターフェイスに macvlan インターフェイスを作成するには、アノテーションの値をそのインターフェイスの名前に設定します。たとえば、**eth1** を使用します。
- ❷ ノードが置かれている物理ネットワークの IP アドレスは egress ルーター Pod で使用するために予約されます。オプション: サブネットの長さ /24 接尾辞を組み込み、ローカルサブネットへの適切なルートがセットアップされるようにできます。サブネットの長さを指定しない場合、egress ルーターは **EGRESS_GATEWAY** 変数で指定されたホストにのみアクセスでき、サブネットの他のホストにはアクセスできません。
- ❸ ノードで使用されるデフォルトゲートウェイと同じ値です。
- ❹ 1つ以上のプロキシ宛先の一覧を指定します。
- ❺ オプション: DNS プロキシログ出力を **stdout** に出力するために指定します。

15.10.2. egress 宛先設定形式

ルーターが DNS プロキシモードでデプロイされる場合、ポートおよび宛先マッピングの一覧を指定します。宛先には、IP アドレスまたは DNS 名のいずれかを使用できます。

egress ルーター Pod は、ポートおよび宛先マッピングを指定するために以下の形式をサポートします。

ポートおよびリモートアドレス

送信元ポートおよび宛先ホストは、2つのフィールド形式 (`<port> <remote_address>`) を使用して指定できます。

ホストには、IP アドレスまたは DNS 名を指定できます。DNS 名を指定すると、DNS 解決が起動時に行われます。特定のホストについては、プロキシは、宛先ホスト IP アドレスへの接続時に、宛先ホストの指定された送信元ポートに接続されます。

ポートとリモートアドレスペアの例

```
80 172.16.12.11
100 example.com
```

ポート、リモートアドレス、およびリモートポート

送信元ポート、宛先ホスト、および宛先ポートは、`<port> <remote_address> <remote_port>` の3つのフィールドからなる形式を使用して指定できます。

3つのフィールド形式は、2つのフィールドバージョンと同じように動作しますが、宛先ポートが送信元ポートとは異なる場合があります。

ポート、リモートアドレス、およびリモートポートの例

```
8080 192.168.60.252 80
8443 web.example.com 443
```

15.10.3. DNS プロキシモードでの egress ルーター Pod のデプロイ

DNS プロキシモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから1つ以上の宛先 IP アドレスに送信する TCP ベースのサービスの DNS プロキシとして機能します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. egress ルーター Pod の作成
2. egress ルーター Pod のサービスを作成します。
 - a. 以下の YAML 定義が含まれる **egress-router-service.yaml** という名前のファイルを作成します。**spec.ports** を、**EGRESS_DNS_PROXY_DESTINATION** 環境変数に先に定義したポートの一覧に設定します。

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

```

metadata:
  name: egress-dns-svc
spec:
  ports:
    ...
  type: ClusterIP
  selector:
    name: egress-dns-proxy

```

以下に例を示します。

```

apiVersion: v1
kind: Service
metadata:
  name: egress-dns-svc
spec:
  ports:
    - name: con1
      protocol: TCP
      port: 80
      targetPort: 80
    - name: con2
      protocol: TCP
      port: 100
      targetPort: 100
  type: ClusterIP
  selector:
    name: egress-dns-proxy

```

- b. サービスを作成するには、以下のコマンドを入力します。

```
$ oc create -f egress-router-service.yaml
```

Pod がこのサービスに接続できるようになります。これらの接続は、予約された egress IP アドレスを使用して外部サーバーの対応するポートにプロキシ送信されます。

15.10.4. 関連情報

- [ConfigMap を使用した egress ルーターの宛先マッピングの設定](#)

15.11. CONFIGMAP からの EGRESS ルーター POD 宛先一覧の設定

クラスター管理者は、egress ルーター Pod の宛先マッピングを指定する **ConfigMap** オブジェクトを定義できます。設定の特定の形式は、egress ルーター Pod のタイプによって異なります。形式についての詳細は、特定の egress ルーター Pod のドキュメントを参照してください。

15.11.1. ConfigMap を使用した egress ルーター宛先マッピングの設定

宛先マッピングのセットのサイズが大きいか、またはこれが頻繁に変更される場合、ConfigMap を使用して一覧を外部で維持できます。この方法の利点は、設定マップを編集するパーミッションを **cluster-admin** 権限を持たないユーザーに委任できることです。egress ルーター Pod には特権付きコンテナを必要とするため、**cluster-admin** 権限を持たないユーザーは Pod 定義を直接編集することはできません。



注記

egress ルーター Pod は、ConfigMap が変更されても自動的に更新されません。更新を取得するには、egress ルーター Pod を再起動する必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の例のように、egress ルーター Pod のマッピングデータが含まれるファイルを作成します。

```
# Egress routes for Project "Test", version 3

80 tcp 203.0.113.25

8080 tcp 203.0.113.26 80
8443 tcp 203.0.113.26 443

# Fallback
203.0.113.27
```

空の行とコメントをこのファイルに追加できます。

2. このファイルから **ConfigMap** オブジェクトを作成します。

```
$ oc delete configmap egress-routes --ignore-not-found
```

```
$ oc create configmap egress-routes \
  --from-file=destination=my-egress-destination.txt
```

直前のコマンドで、**egress-routes** 値は、作成する **ConfigMap** オブジェクトの名前で、**my-egress-destination.txt** はデータの読み取り元のファイルの名前です。

ヒント

または、以下の YAML を適用して ConfigMap を作成できます。

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: egress-routes
data:
  destination: |
    # Egress routes for Project "Test", version 3

    80 tcp 203.0.113.25

    8080 tcp 203.0.113.26 80
    8443 tcp 203.0.113.26 443

    # Fallback
    203.0.113.27

```

3. egress ルーター Pod 定義を作成し、environment スタンザの **EGRESS_DESTINATION** フィールドに **configMapKeyRef** スタンザを指定します。

```

...
env:
- name: EGRESS_DESTINATION
  valueFrom:
    configMapKeyRef:
      name: egress-routes
      key: destination
...

```

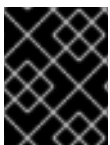
15.11.2. 関連情報

- [リダイレクトモード](#)
- [HTTP_PROXY](#)
- [DNS プロキシモード](#)

15.12. プロジェクトのマルチキャストの有効化

15.12.1. マルチキャストについて

IP マルチキャストを使用すると、データが多数の IP アドレスに同時に配信されます。



重要

現時点で、マルチキャストは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のソリューションとしては適していません。

OpenShift Container Platform の Pod 間のマルチキャストトラフィックはデフォルトで無効にされます。OpenShift SDN デフォルト Container Network Interface (CNI) ネットワークプロバイダーを使用している場合は、プロジェクトごとにマルチキャストを有効にできます。

networkpolicy 分離モードで OpenShift SDN ネットワークプラグインを使用する場合は、以下を行います。

- Pod によって送信されるマルチキャストパケットは、**NetworkPolicy** オブジェクトに関係なく、プロジェクトの他のすべての Pod に送信されます。Pod はユニキャストで通信できない場合でもマルチキャストで通信できます。
- 1つのプロジェクトの Pod によって送信されるマルチキャストパケットは、**NetworkPolicy** オブジェクトがプロジェクト間の通信を許可する場合であっても、それ以外のプロジェクトの Pod に送信されることはありません。

multitenant 分離モードで OpenShift SDN ネットワークプラグインを使用する場合は、以下を行います。

- Pod で送信されるマルチキャストパケットはプロジェクトにあるその他の全 Pod に送信されません。
- あるプロジェクトの Pod によって送信されるマルチキャストパケットは、各プロジェクトが結合し、マルチキャストが結合した各プロジェクトで有効にされている場合にのみ、他のプロジェクトの Pod に送信されます。

15.12.2. Pod 間のマルチキャストの有効化

プロジェクトの Pod でマルチキャストを有効にすることができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行し、プロジェクトのマルチキャストを有効にします。<namespace> を、マルチキャストを有効にする必要のある namespace に置き換えます。

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

検証

マルチキャストがプロジェクトについて有効にされていることを確認するには、以下の手順を実行します。

1. 現在のプロジェクトを、マルチキャストを有効にしたプロジェクトに切り替えます。<project> をプロジェクト名に置き換えます。

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

2. マルチキャストレシーバーとして機能する Pod を作成します。

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
  - name: mlistener
    image: registry.access.redhat.com/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat hostname && sleep inf"]
    ports:
    - containerPort: 30102
      name: mlistener
      protocol: UDP
EOF
```

3. マルチキャストセNDERとして機能する Pod を作成します。

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
  - name: msender
    image: registry.access.redhat.com/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat && sleep inf"]
EOF
```

4. 新しいターミナルウィンドウまたはタブで、マルチキャストリスナーを起動します。
 - a. Pod の IP アドレスを取得します。

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

- b. 次のコマンドを入力して、マルチキャストリスナーを起動します。

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

5. マルチキャストトランスミッターを開始します。
 - a. Pod ネットワーク IP アドレス範囲を取得します。

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

- b. マルチキャストメッセージを送信するには、以下のコマンドを入力します。

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

マルチキャストが機能している場合、直前のコマンドは以下の出力を返します。

```
mlistener
```

15.13. プロジェクトのマルチキャストの無効化

15.13.1. Pod 間のマルチキャストの無効化

プロジェクトの Pod でマルチキャストを無効にすることができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行して、マルチキャストを無効にします。

```
$ oc annotate netnamespace <namespace> \ 1
  netnamespace.network.openshift.io/multicast-enabled-
```

- 1** マルチキャストを無効にする必要のあるプロジェクトの **namespace**。

15.14. OPENSIFT SDN を使用したネットワーク分離の設定

クラスターが OpenShift SDN CNI プラグインのマルチテナント分離モードを使用するように設定されている場合、各プロジェクトはデフォルトで分離されます。ネットワークトラフィックは、マルチテナント分離モードでは、異なるプロジェクトの Pod およびサービス間で許可されません。

プロジェクトのマルチテナント分離の動作を 2 つの方法で変更することができます。

- 1 つ以上のプロジェクトを結合し、複数の異なるプロジェクトの Pod とサービス間のネットワークトラフィックを可能にします。
- プロジェクトのネットワーク分離を無効にできます。これはグローバルにアクセスできるようになり、他のすべてのプロジェクトの Pod およびサービスからのネットワークトラフィックを受け入れます。グローバルにアクセス可能なプロジェクトは、他のすべてのプロジェクトの Pod およびサービスにアクセスできます。

15.14.1. 前提条件

- クラスタは、マルチテナント分離ノードで OpenShift SDN Container Network Interface (CNI) プラグインを使用するように設定されている必要があります。

15.14.2. プロジェクトの結合

2つ以上のプロジェクトを結合し、複数の異なるプロジェクトの Pod とサービス間のネットワークトラフィックを可能にします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスタにログインする必要があります。

手順

1. 以下のコマンドを使用して、プロジェクトを既存のプロジェクトネットワークに参加させます。

```
$ oc adm pod-network join-projects --to=<project1> <project2> <project3>
```

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

2. オプション: 以下のコマンドを実行し、結合した Pod ネットワークを表示します。

```
$ oc get netnamespaces
```

同じ Pod ネットワークのプロジェクトには、**NETID** 列に同じネットワーク ID があります。

15.14.3. プロジェクトの分離

他のプロジェクトの Pod およびサービスがその Pod およびサービスにアクセスできないようにするためにプロジェクトを分離することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスタにログインする必要があります。

手順

- クラスタのプロジェクトを分離するには、以下のコマンドを実行します。

```
$ oc adm pod-network isolate-projects <project1> <project2>
```

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

15.14.4. プロジェクトのネットワーク分離の無効化

プロジェクトのネットワーク分離を無効にできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- プロジェクトの以下のコマンドを実行します。

```
$ oc adm pod-network make-projects-global <project1> <project2>
```

または、特定のプロジェクト名を指定する代わりに **--selector=<project_selector>** オプションを使用し、関連付けられたラベルに基づいてプロジェクトを指定できます。

15.15. KUBE-PROXY の設定

Kubernetes ネットワークプロキシ (kube-proxy) は各ノードで実行され、Cluster Network Operator (CNO) で管理されます。kube-proxy は、サービスに関連付けられたエンドポイントの接続を転送するためのネットワークルールを維持します。

15.15.1. iptables ルールの同期について

同期の期間は、Kubernetes ネットワークプロキシ (kube-proxy) がノードで iptables ルールを同期する頻度を定めます。

同期は、以下のイベントのいずれかが生じる場合に開始します。

- サービスまたはエンドポイントのクラスターへの追加、またはクラスターからの削除などのイベントが発生する。
- 最後の同期以後の時間が kube-proxy に定義される同期期間を超過している。

15.15.2. kube-proxy 設定パラメーター

以下の **kubeProxyConfig** パラメーターを変更することができます。



注記

OpenShift Container Platform 4.3 以降で強化されたパフォーマンスの向上により、**iptablesSyncPeriod** パラメーターを調整する必要はなくなりました。

表15.2 パラメーター

パラメーター	説明	値	デフォルト
iptablesSyncPeriod	iptables ルールの更新期間。	30s または 2m などの期間。 有効な接尾辞には、 s 、 m 、および h などが含まれ、これらについては、 Go Package time ドキュメントで説明されています。	30s

パラメーター	説明	値	デフォルト
proxyArguments.iptables-min-sync-period	iptables ルールを更新する前の最小期間。このパラメーターにより、更新の頻度が高くなり過ぎないようにできます。デフォルトでは、 iptables ルールに影響する変更が生じるとすぐに、更新が開始されます。	30s または 2m などの期間。有効な接尾辞には、 s 、 m 、および h が含まれ、これらについては、 Go Package time で説明されています。	0s

15.15.3. kube-proxy 設定の変化

クラスターの Kubernetes ネットワークプロキシ設定を変更することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールで実行中のクラスターにログインします。

手順

1. 以下のコマンドを実行して、**Network.operator.openshift.io** カスタムリソース (CR) を編集します。

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

2. 以下のサンプル CR のように、kube-proxy 設定への変更内容で、CR の **kubeProxyConfig** パラメーターを変更します。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  kubeProxyConfig:
    iptablesSyncPeriod: 30s
    proxyArguments:
      iptables-min-sync-period: ["30s"]
```

3. ファイルを保存し、テキストエディターを編集します。
構文は、ファイルを保存し、エディターを終了する際に **oc** コマンドによって検証されます。変更内容に構文エラーが含まれる場合、エディターはファイルを開き、エラーメッセージを表示します。
4. 以下のコマンドを実行して、設定の更新を確認します。

```
$ oc get networks.operator.openshift.io -o yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: operator.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    clusterNetwork:
      - cidr: 10.128.0.0/14
        hostPrefix: 23
    defaultNetwork:
      type: OpenShiftSDN
    kubeProxyConfig:
      iptablesSyncPeriod: 30s
      proxyArguments:
        iptables-min-sync-period:
          - 30s
    serviceNetwork:
      - 172.30.0.0/16
  status: {}
kind: List
```

5. オプション: 以下のコマンドを実行し、Cluster Network Operator が設定変更を受け入れていることを確認します。

```
$ oc get clusteroperator network
```

出力例

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

設定の更新が正常に適用されると、**AVAILABLE** フィールドが **True** になります。

第16章 OVN-KUBERNETES デフォルト CNI ネットワークプロバイダー

16.1. OVN-KUBERNETES デフォルト CONTAINER NETWORK INTERFACE (CNI) ネットワークプロバイダーについて

OpenShift Container Platform クラスタは、Pod およびサービスネットワークに仮想化ネットワークを使用します。OVN-Kubernetes Container Network Interface (CNI) プラグインは、デフォルトのクラスタネットワークのネットワークプロバイダーです。OVN-Kubernetes は Open Virtual Network (OVN) をベースとしており、オーバーレイベースのネットワーク実装を提供します。OVN-Kubernetes ネットワークプロバイダーを使用するクラスタは、各ノードで Open vSwitch (OVS) も実行します。OVN は、宣言ネットワーク設定を実装するように各ノードで OVS を設定します。

16.1.1. OVN-Kubernetes の機能

OVN-Kubernetes Container Network Interface (CNI) クラスタネットワークプロバイダーは、以下の機能を実装します。

- Open Virtual Network (OVN) を使用してネットワークトラフィックフローを管理します。OVN はコミュニティで開発され、ベンダーに依存しないネットワーク仮想化ソリューションです。
- ingress および egress ルールを含む Kubernetes ネットワークポリシーのサポートを実装します。
- ノード間にオーバーレイネットワークを作成するには、VXLAN ではなく GENEVE (Generic Network Virtualization Encapsulation) プロトコルを使用します。

16.1.2. サポートされるデフォルトの CNI ネットワークプロバイダー機能マトリクス

OpenShift Container Platform は、OpenShift SDN と OVN-Kubernetes の2つのサポート対象のオプションをデフォルトの Container Network Interface (CNI) ネットワークプロバイダーに提供します。以下の表は、両方のネットワークプロバイダーの現在の機能サポートをまとめたものです。

表16.1 デフォルトの CNI ネットワークプロバイダー機能の比較

機能	OVN-Kubernetes	OpenShift SDN
Egress IP	サポート対象	サポート対象
Egress ファイアウォール ^[1]	サポート対象	サポート対象
Egress ルーター	サポート対象 ^[2]	サポート対象
IPsec 暗号化	サポート対象	サポート対象外
IPv6	サポート対象 ^[3]	サポート対象外
Kubernetes ネットワークポリシー	サポート対象	一部サポート対象 ^[4]

機能	OVN-Kubernetes	OpenShift SDN
Kubernetes ネットワークポリシーログ	サポート対象	サポート対象外
マルチキャスト	サポート対象	サポート対象

1. egress ファイアウォールは、OpenShift SDN では egress ネットワークポリシーとしても知られています。これはネットワークポリシーの egress とは異なります。
2. OVN-Kubernetes の egress ルーターはリダイレクトモードのみをサポートします。
3. IPv6 はベアメタルクラスターでのみサポートされます。
4. OpenShift SDN のネットワークポリシーは、egress ルールおよび一部の **ipBlock** ルールをサポートしません。

16.1.3. OVN-Kubernetes の制限

OVN-Kubernetes Container Network Interface (CNI) クラスターネットワークプロバイダーには以下の制限があります。

- OVN-Kubernetes には、Kubernetes サービスの外部トラフィックポリシーまたは内部トラフィックポリシーを **local** に設定するサポートはありません。デフォルト値は **cluster** で、両方のパラメーターでサポートされます。この制限は、**LoadBalancer** タイプ、**NodePort** タイプのサービスを追加するか、外部 IP でサービスを追加する際に影響を受ける可能性があります。
- **sessionAffinityConfig.clientIP.timeoutSeconds** サービスは、OpenShift OVN 環境では効果がありませんが、OpenShiftSDN 環境では効果があります。この非互換性により、OpenShiftSDN から OVN への移行が困難になる可能性があります。
- デュアルスタックネットワークに設定されたクラスターでは、IPv4 と IPv6 の両方のトラフィックがデフォルトゲートウェイとして同じネットワークインターフェイスを使用する必要があります。この要件が満たされない場合には、**ovnkube-node** デモンセットのホストにある Pod は、**CrashLoopBackOff** 状態になります。**oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml** のようなコマンドで Pod を表示すると、以下の出力のように、**status** フィールドにデフォルトゲートウェイに関する複数のメッセージが表示されます。

```
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
```

唯一の解決策は、両方の IP ファミリーがデフォルトゲートウェイに同じネットワークインターフェイスを使用するように、ホストネットワークを再設定することです。

- デュアルスタックネットワーク用に設定されたクラスターの場合、IPv4 と IPv6 の両方のルーティングテーブルにデフォルトゲートウェイが含まれている必要があります。この要件が満たされない場合には、**ovnkube-node** デモンセットのホストにある Pod は、**CrashLoopBackOff** 状態になります。**oc get pod -n openshift-ovn-kubernetes -l app=ovnkube-node -o yaml** のようなコマンドで Pod を表示すると、以下の出力のように、**status** フィールドにデフォルトゲートウェイに関する複数のメッセージが表示されます。

```
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
```

唯一の解決策として、両方の IP ファミリーにデフォルトゲートウェイが含まれるようにホストネットワークを再設定できます。

関連情報

- [プロジェクトの egress ファイアウォールの設定](#)
- [ネットワークポリシーについて](#)
- [ネットワークポリシーイベントのロギング](#)
- [プロジェクトのマルチキャストの有効化](#)
- [IPsec 暗号化の設定](#)
- [Network \[operator.openshift.io/v1\]](#)

16.2. OPENSIFT SDN クラスターネットワークプロバイダーからの移行

クラスター管理者は、OpenShift SDN CNI クラスターネットワークプロバイダーから OVN-Kubernetes Container Network Interface(CNI) クラスターネットワークプロバイダーに移行できます。

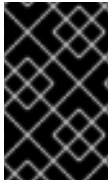
OVN-Kubernetes についての詳細は、[OVN-Kubernetes ネットワークプロバイダーについて](#) を参照してください。

16.2.1. OVN-Kubernetes ネットワークプロバイダーへの移行

OVN-Kubernetes Container Network Interface (CNI) クラスターネットワークプロバイダーへの移行は、クラスターに到達できなくなるダウンタイムも含まれる手動プロセスです。ロールバック手順が提供されますが、移行は一方方向プロセスとなることが意図されています。

OVN-Kubernetes クラスターネットワークプロバイダーへの移行は、以下のプラットフォームでサポートされます。

- ベアメタルハードウェア
- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure
- Red Hat OpenStack Platform (RHOSP)
- Red Hat Virtualization (RHV)
- VMware vSphere



重要

OVN-Kubernetes ネットワークプラグインとの間の移行は、OpenShift Dedicated や Red Hat OpenShift Service on AWS (ROSA) などのマネージド OpenShift クラウドサービスではサポートされていません。

16.2.1.1. OVN-Kubernetes ネットワークプロバイダーへの移行についての考慮点

OpenShift Container Platform クラスタに 150 を超えるノードがある場合は、OVN-Kubernetes ネットワークプラグインへの移行について相談するサポートケースを開きます。

ノードに割り当てられたサブネット、および個々の Pod に割り当てられた IP アドレスは、移行時に保持されません。

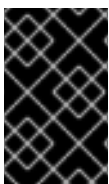
OVN-Kubernetes ネットワークプロバイダーは OpenShift SDN ネットワークプロバイダーに存在する多くの機能を実装しますが、設定は同じではありません。

- クラスタが以下の OpenShift SDN 機能のいずれかを使用する場合、OVN-Kubernetes で同じ機能を手動で設定する必要があります。
 - namespace の分離
 - Egress IP アドレス
 - Egress ネットワークポリシー
 - Egress ルーター Pod
 - マルチキャスト
- クラスタが **100.64.0.0/16** IP アドレス範囲の一部を使用する場合、この IP アドレス範囲は内部で使用されるため、OVN-Kubernetes に移行することはできません。

以下のセクションでは、OVN-Kubernetes と OpenShift SDN の上記の機能間の設定の違いについて説明します。

namespace の分離

OVN-Kubernetes はネットワークポリシーの分離モードのみをサポートします。



重要

クラスタがマルチテナントまたはサブネットの分離モードのいずれかで設定された OpenShift SDN を使用する場合、OVN-Kubernetes ネットワークプロバイダーに移行することはできません。

Egress IP アドレス

OVN-Kubernetes と OpenShift SDN との間に egress IP アドレスを設定する際の相違点は、以下の表で説明されています。

表16.2 egress IP アドレス設定の違い

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● EgressIPs オブジェクトを作成します。 ● アノテーションを Node オブジェクトに追加します。 	<ul style="list-style-type: none"> ● NetNamespace オブジェクトにパッチを適用します。 ● HostSubnet オブジェクトにパッチを適用します。

OVN-Kubernetes で egress IP アドレスを使用する方法についての詳細は、egress IP アドレスの設定について参照してください。

Egress ネットワークポリシー

OVN-Kubernetes と OpenShift SDN との間に egress ファイアウォールとしても知られる egress ネットワークポリシーの設定についての相違点は、以下の表に記載されています。

表16.3 egress ネットワークポリシー設定の相違点

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● EgressFirewall オブジェクトを namespace に作成します。 	<ul style="list-style-type: none"> ● EgressNetworkPolicy オブジェクトを namespace に作成します。

OVN-Kubernetes で egress ファイアウォールを使用する方法についての詳細は、プロジェクトの egress ファイアウォールの設定について参照してください。

Egress ルーター Pod

OVN-Kubernetes は、リダイレクトモードで Egress ルーター Pod をサポートします。OVN-Kubernetes は、HTTP プロキシモードまたは DNS プロキシモードでは Egress ルーター Pod をサポートしません。

Cluster Network Operator で Egress ルーターをデプロイする場合、ノードセレクターを指定して、Egress ルーター Pod のホストに使用するノードを制御することはできません。

マルチキャスト

OVN-Kubernetes と OpenShift SDN でマルチキャストトラフィックを有効にする方法についての相違点は、以下の表で説明されています。

表16.4 マルチキャスト設定の相違点

OVN-Kubernetes	OpenShift SDN
<ul style="list-style-type: none"> ● アノテーションを Namespace オブジェクトに追加します。 	<ul style="list-style-type: none"> ● アノテーションを NetNamespace オブジェクトに追加します。

OVN-Kubernetes でのマルチキャストの使用についての詳細は、プロジェクトのマルチキャストの有効化を参照してください。

ネットワークポリシー

OVN-Kubernetes は、**networking.k8s.io/v1** API グループで Kubernetes **NetworkPolicy** API を完全にサポートします。OpenShift SDN から移行する際に、ネットワークポリシーで変更を加える必要はありません。

16.2.1.2. 移行プロセスの仕組み

以下の表は、プロセスのユーザーが開始する手順と、移行が応答として実行するアクション間を区分して移行プロセスを要約しています。

表16.5 OpenShift SDN から OVN-Kubernetes への移行

ユーザー起動の手順	移行アクティビティー
<p>cluster という名前の Network.operator.openshift.io カスタムリソース (CR) の migration フィールドを OVNKubernetes に設定します。 migration フィールドを値に設定する前に null であることを確認します。</p>	<p>Cluster Network Operator (CNO)</p> <p>cluster という名前の Network.config.openshift.io CR のステータスを更新します。</p> <p>Machine Config Operator (MCO)</p> <p>OVN-Kubernetes に必要な systemd 設定への更新をロールアウトします。MCO はデフォルトで1度にプールごとに単一のマシンを更新します。これにより、移行にかかる合計時間がクラスターのサイズと共に増加します。</p>
<p>Network.config.openshift.io CR の networkType フィールドを更新します。</p>	<p>CNO</p> <p>以下のアクションを実行します。</p> <ul style="list-style-type: none"> ● OpenShift SDN コントロールプレーン Pod を破棄します。 ● OVN-Kubernetes コントロールプレーン Pod をデプロイします。 ● 新しいクラスターネットワークプロバイダーを反映するように Multus オブジェクトを更新します。
<p>クラスターの各ノードを再起動します。</p>	<p>Cluster</p> <p>ノードの再起動時に、クラスターは OVN-Kubernetes クラスターネットワークの Pod に IP アドレスを割り当てます。</p>

OpenShift SDN へのロールバックが必要な場合、以下の表がプロセスについて説明します。

表16.6 OpenShift SDN へのロールバックの実行

ユーザー起動の手順	移行アクティビティー
<p>MCO を一時停止し、移行が中断されないようにします。</p>	<p>MCO が停止します。</p>

ユーザー起動の手順	移行アクティビティー
<p>cluster という名前の Network.operator.openshift.io カスタムリソース (CR) の migration フィールドを OpenShiftSDN に設定します。 migration フィールドを値に設定する前に null であることを確認します。</p>	<p>CNO</p> <p>cluster という名前の Network.config.openshift.io CR のステータスを更新します。</p>
<p>networkType フィールドを更新します。</p>	<p>CNO</p> <p>以下のアクションを実行します。</p> <ul style="list-style-type: none"> ● OVN-Kubernetes コントロールプレーン Pod を破棄します。 ● OpenShift SDN コントロールプレーン Pod をデプロイします。 ● 新しいクラスターネットワークプロバイダーを反映するように Multus オブジェクトを更新します。
<p>クラスターの各ノードを再起動します。</p>	<p>Cluster</p> <p>ノードがリブートすると、クラスターは OpenShift-SDN ネットワーク上の Pod に IP アドレスを割り当てます。</p>
<p>クラスターのすべてのノードが再起動した後に MCO を有効にします。</p>	<p>MCO</p> <p>OpenShift SDN に必要な systemd 設定への更新をロールアウトします。MCO はデフォルトで1度にプールごとに単一のマシンを更新します。これにより、移行にかかる合計時間がクラスターのサイズと共に増加します。</p>

16.2.2. OVN-Kubernetes デフォルト CNI ネットワークプロバイダーへの移行

クラスター管理者は、クラスターのデフォルトの Container Network Interface (CNI) ネットワークプロバイダーを OVN-Kubernetes に変更できます。移行時に、クラスター内のすべてのノードを再起動する必要があります。



重要

移行の実行中はクラスターを利用できず、ワークロードが中断される可能性があります。サービスの中断が許容可能な場合にのみ移行を実行します。

前提条件

- ネットワークポリシーの分離モードで OpenShift SDN CNI クラスターネットワークプロバイダーで設定されたクラスター。

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- etcd データベースの最新のバックアップが利用可能である。
- 再起動は、ノードごとに手動でトリガーできます。
- クラスターは既知の正常な状態にあり、エラーがないこと。

手順

1. クラスターネットワークの設定のバックアップを作成するには、以下のコマンドを入力します。

```
$ oc get Network.config.openshift.io cluster -o yaml > cluster-openshift-sdn.yaml
```

2. 移行のすべてのノードを準備するには、以下のコマンドを入力して Cluster Network Operator 設定オブジェクトに **migration** フィールドを設定します。

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



注記

この手順では、OVN-Kubernetes はすぐにデプロイしません。その代わりに、**migration** フィールドを指定すると、新規マシン設定が OVN-Kubernetes デプロイメントの準備に向けてクラスター内のすべてのノードに適用されるように Machine Config Operator (MCO) がトリガーされます。

3. オプション: ネットワークインフラストラクチャーの要件を満たすように OVN-Kubernetes の以下の設定をカスタマイズできます。

- Maximum transmission unit (MTU)
- Geneve (Generic Network Virtualization Encapsulation) オーバーレイネットワークポート

以前の設定のいずれかをカスタマイズするには、以下のコマンドを入力してカスタマイズします。デフォルト値を変更する必要がない場合は、パッチのキーを省略します。

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

mtu

Geneve オーバーレイネットワークの MTU。この値は通常は自動的に設定されますが、クラスターにあるノードすべてが同じ MTU を使用しない場合、これを最小のノード MTU 値よりも **100** 小さく設定する必要があります。

port

Geneve オーバーレイネットワークの UDP ポート。値が指定されない場合、デフォルトは **6081** になります。ポートは、OpenShift SDN で使用される VXLAN ポートと同じにすることはできません。VXLAN ポートのデフォルト値は **4789** です。

mtu フィールドを更新するパッチコマンドの例

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

4. MCO がそれぞれのマシン設定プールのマシンを更新すると、各ノードが1つずつ再起動します。すべてのノードが更新されるまで待機する必要があります。以下のコマンドを実行してマシン設定プールのステータスを確認します。

```
$ oc get mcp
```

正常に更新されたノードには、**UPDATED=true**、**UPDATING=false**、**DEGRADED=false** のステータスがあります。



注記

デフォルトで、MCO はプールごとに一度に1つのマシンを更新するため、移行にかかる合計時間がクラスタのサイズと共に増加します。

5. ホスト上の新規マシン設定のステータスを確認します。
 - a. マシン設定の状態と適用されたマシン設定の名前を一覧表示するには、以下のコマンドを入力します。

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

出力例

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

以下のステートメントが true であることを確認します。

- **machineconfiguration.openshift.io/state** フィールドの値は **Done** です。
- **machineconfiguration.openshift.io/currentConfig** フィールドの値は、**machineconfiguration.openshift.io/desiredConfig** フィールドの値と等しくなります。

- b. マシン設定が正しいことを確認するには、以下のコマンドを入力します。

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

ここで、**<config_name>** は、 **machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前になります。

マシン設定には、systemd 設定に以下の更新を含める必要があります。

```
ExecStart=/usr/local/bin/configure-ovs.sh OVNKubernetes
```

- c. ノードが **NotReady** 状態のままになっている場合、マシン設定デーモン Pod のログを調べ、エラーを解決します。

- i. Pod を一覧表示するには、以下のコマンドを入力します。

```
$ oc get pod -n openshift-machine-config-operator
```

出力例

NAME	READY	STATUS	RESTARTS	AGE
machine-config-controller-75f756f89d-sjp8b	1/1	Running	0	37m
machine-config-daemon-5cf4b	2/2	Running	0	43h
machine-config-daemon-7wzcd	2/2	Running	0	43h
machine-config-daemon-fc946	2/2	Running	0	43h
machine-config-daemon-g2v28	2/2	Running	0	43h
machine-config-daemon-gcl4f	2/2	Running	0	43h
machine-config-daemon-l5tnv	2/2	Running	0	43h
machine-config-operator-79d9c55d5-hth92	1/1	Running	0	37m
machine-config-server-bsc8h	1/1	Running	0	43h
machine-config-server-hklrm	1/1	Running	0	43h
machine-config-server-k9rtx	1/1	Running	0	43h

設定デーモン Pod の名前は以下の形式になります。 **machine-config-daemon-
<seq><seq>** 値は、ランダムな 5 文字の英数字シーケンスになります。

- ii. 以下のコマンドを入力して、直前の出力に表示される最初のマシン設定デーモン Pod の Pod ログを表示します。

```
$ oc logs <pod> -n openshift-machine-config-operator
```

ここで、**pod** はマシン設定デーモン Pod の名前になります。

- iii. 直前のコマンドの出力で示されるログ内のエラーを解決します。

6. 移行を開始するには、以下のコマンドのいずれかを使用して、OVN-Kubernetes クラスタネットワークプロバイダーを設定します。

- クラスタネットワークの IP アドレスブロックを変更せずにネットワークプロバイダーを指定するには、以下のコマンドを入力します。

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

- 別のクラスタネットワーク IP アドレスブロックを指定するには、以下のコマンドを入力します。

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

ここで、**cidr** は CIDR ブロックであり、**prefix** はクラスター内の各ノードに割り当てられる CIDR ブロックのスライスです。OVN-Kubernetes ネットワークプロバイダーはこのブロックを内部で使用するため、**100.64.0.0/16** CIDR ブロックと重複する CIDR ブロックは使用できません。



重要

移行時に、サービスネットワークのアドレスブロックを変更することはできません。

7. Multus デモンセットのロールアウトが完了したことを確認してから、後続の手順を続行します。

```
$ oc -n openshift-multus rollout status daemonset/multus
```

Multus Pod の名前の形式は **multus-<xxxxxx>** です。ここで、**<xxxxxx>** は文字のランダムなシーケンスになります。Pod が再起動するまでにしばらく時間がかかる可能性があります。

出力例

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

8. 移行を完了するには、クラスター内の各ノードを再起動します。たとえば、以下のような bash スクリプトを使用できます。このスクリプトは、**ssh** を使用して各ホストに接続でき、**sudo** がパスワードを要求しないように設定されていることを前提としています。

```
#!/bin/bash

for ip in $(oc get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}')
do
  echo "reboot node $ip"
  ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

ssh アクセスが使用できない場合、インフラストラクチャプロバイダーの管理ポータルから各ノードを再起動できる場合があります。

9. 移行が正常に完了したことを確認します。

- a. CNI ネットワークプロバイダーが OVN-Kubernetes であることを確認するには、以下のコマンドを入力します。 **status.networkType** の値は **OVNKubernetes** である必要があります。

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

- b. クラスターノードが **Ready** 状態にあることを確認するには、以下のコマンドを実行します。

```
$ oc get nodes
```

- c. Pod がエラー状態ではないことを確認するには、以下のコマンドを入力します。

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

ノードの Pod がエラー状態にある場合は、そのノードを再起動します。

- d. すべてのクラスター Operator が異常な状態にないことを確認するには、以下のコマンドを入力します。

```
$ oc get co
```

すべてのクラスター Operator のステータスは、

AVAILABLE="True"、**PROGRESSING="False"**、**DEGRADED="False"** になります。クラスター Operator が利用できないか、またはそのパフォーマンスが低下する場合には、クラスター Operator のログで詳細を確認します。

10. 以下の手順は、移行に成功し、クラスターの状態が正常である場合にのみ実行します。

- a. CNO 設定オブジェクトから移行設定を削除するには、以下のコマンドを入力します。

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

- b. OpenShift SDN ネットワークプロバイダーのカスタム設定を削除するには、以下のコマンドを入力します。

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

- c. OpenShift SDN ネットワークプロバイダー namespace を削除するには、以下のコマンドを入力します。

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

16.2.3. 関連情報

- [OVN-Kubernetes デフォルト CNI ネットワークプロバイダーの設定パラメーター](#)
- [etcd のバックアップ](#)
- [ネットワークポリシーについて](#)

- OVN-Kubernetes の機能
 - egress IP アドレスの設定
 - プロジェクトの egress ファイアウォールの設定
 - プロジェクトのマルチキャストの有効化
- OpenShift SDN の機能
 - プロジェクトの egress IP の設定
 - プロジェクトの egress ファイアウォールの設定
 - プロジェクトのマルチキャストの有効化
- [Network \[operator.openshift.io/v1\]](https://operator.openshift.io/v1)

16.3. OPENSIFT SDN ネットワークプロバイダーへのロールバック

クラスター管理者は、OVN-Kubernetes CNI クラスターのネットワークプロバイダーから OpenShift SDN クラスターの Container Network Interface (CNI) クラスターネットワークプロバイダーにロールバックできます (OVN-Kubernetes への移行に失敗した場合)。

16.3.1. デフォルトの CNI ネットワークプロバイダーの OpenShift SDN へのロールバック

クラスター管理者は、クラスターを OpenShift SDN Container Network Interface (CNI) クラスターネットワークプロバイダーにロールバックできます。ロールバック時に、クラスター内のすべてのノードを再起動する必要があります。



重要

OVN-Kubernetes への移行に失敗した場合にのみ OpenShift SDN にロールバックします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OVN-Kubernetes CNI クラスターネットワークプロバイダーで設定されたインフラストラクチャーにクラスターがインストールされている。

手順

1. Machine Config Operator (MCO) によって管理されるすべてのマシン設定プールを停止します。
 - マスター設定プールを停止します。

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

- ワーカーマシン設定プールを停止します。

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

2. 移行を開始するには、以下のコマンドを入力してクラスターネットワークプロバイダーを OpenShift SDN に戻します。

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

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

3. オプション: ネットワークインフラストラクチャーの要件を満たすように OpenShift SDN の以下の設定をカスタマイズできます。

- Maximum transmission unit (MTU)
- VXLAN ポート

以前の設定のいずれかを両方をカスタマイズするには、カスタマイズし、以下のコマンドを入力します。デフォルト値を変更する必要がない場合は、パッチのキーを省略します。

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

mtu

VXLAN オーバーレイネットワークの MTU。この値は通常は自動的に設定されますが、クラスターにあるノードすべてが同じ MTU を使用しない場合、これを最小のノード MTU 値よりも **50** 小さく設定する必要があります。

port

VXLAN オーバーレイネットワークの UDP ポート。値が指定されない場合は、デフォルトは **4789** になります。ポートは OVN-Kubernetes で使用される Geneve ポートと同じにすることはできません。Geneve ポートのデフォルト値は **6081** です。

patch コマンドの例

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

4. Multus デモンセットのロールアウトが完了するまで待機します。

```
$ oc -n openshift-multus rollout status daemonset/multus
```

Multus Pod の名前の形式は **multus-`<xxxxxx>`** です。ここで、`<xxxxxx>` は文字のランダムなシーケンスになります。Pod が再起動するまでにしばらく時間がかかる可能性があります。

出力例

```
Waiting for daemon set "multus" rollout to finish: 1 out of 6 new pods have been updated...
...
Waiting for daemon set "multus" rollout to finish: 5 of 6 updated pods are available...
daemon set "multus" successfully rolled out
```

5. ロールバックを完了するには、クラスター内の各ノードを再起動します。たとえば、以下のような bash スクリプトを使用できます。このスクリプトは、**ssh** を使用して各ホストに接続でき、**sudo** がパスワードを要求しないように設定されていることを前提としています。

```
#!/bin/bash

for ip in $(oc get nodes -o jsonpath='{.items[*].status.addresses[?(@.type=="InternalIP")].address}');
do
    echo "reboot node $ip"
    ssh -o StrictHostKeyChecking=no core@$ip sudo shutdown -r -t 3
done
```

ssh アクセスが使用できない場合、インフラストラクチャプロバイダーの管理ポータルから各ノードを再起動できる場合があります。

6. クラスターのノードが再起動したら、すべてのマシン設定プールを起動します。
 - マスター設定プールを開始します。

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

- ワーカー設定プールを開始します。

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

MCO が各設定プールのマシンを更新すると、各ノードを再起動します。

デフォルトで、MCO は一度にプールごとに単一のマシンを更新するため、移行が完了するまでに必要な時間がクラスターのサイズと共に増加します。

7. ホスト上の新規マシン設定のステータスを確認します。
 - a. マシン設定の状態と適用されたマシン設定の名前を一覧表示するには、以下のコマンドを入力します。

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

出力例

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

以下のステートメントが true であることを確認します。

- **machineconfiguration.openshift.io/state** フィールドの値は **Done** です。
- **machineconfiguration.openshift.io/currentConfig** フィールドの値は、**machineconfiguration.openshift.io/desiredConfig** フィールドの値と等しくなります。

b. マシン設定が正しいことを確認するには、以下のコマンドを入力します。

```
$ oc get machineconfig <config_name> -o yaml
```

ここで、**<config_name>** は、**machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前になります。

8. 移行が正常に完了したことを確認します。

a. デフォルトの CNI ネットワークプロバイダーが OVN-Kubernetes であることを確認するには、以下のコマンドを入力します。**status.networkType** の値は **OpenShiftSDN** である必要があります。

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

b. クラスターノードが **Ready** 状態にあることを確認するには、以下のコマンドを実行します。

```
$ oc get nodes
```

c. ノードが **NotReady** 状態のままになっている場合、マシン設定デーモン Pod のログを調べ、エラーを解決します。

i. Pod を一覧表示するには、以下のコマンドを入力します。

```
$ oc get pod -n openshift-machine-config-operator
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
machine-config-controller-75f756f89d-sjp8b 1/1 Running 0 37m
machine-config-daemon-5cf4b             2/2 Running 0 43h
machine-config-daemon-7wzcd             2/2 Running 0 43h
machine-config-daemon-fc946             2/2 Running 0 43h
machine-config-daemon-g2v28             2/2 Running 0 43h
machine-config-daemon-gcl4f             2/2 Running 0 43h
machine-config-daemon-l5tnv             2/2 Running 0 43h
machine-config-operator-79d9c55d5-hth92 1/1 Running 0 37m
```



```

machine-config-server-bsc8h      1/1  Running  0    43h
machine-config-server-hklrm      1/1  Running  0    43h
machine-config-server-k9rtx      1/1  Running  0    43h

```

設定デーモン Pod の名前は以下の形式になります。 **machine-config-daemon-
<seq><seq>** 値は、ランダムな 5 文字の英数字シーケンスになります。

- ii. 直前の出力に表示されるそれぞれのマシン設定デーモン Pod の Pod ログを表示するには、以下のコマンドを入力します。

```
$ oc logs <pod> -n openshift-machine-config-operator
```

ここで、 **pod** はマシン設定デーモン Pod の名前になります。

- iii. 直前のコマンドの出力で示されるログ内のエラーを解決します。

- d. Pod がエラー状態ではないことを確認するには、以下のコマンドを入力します。

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

ノードの Pod がエラー状態にある場合は、そのノードを再起動します。

- 9. 以下の手順は、移行に成功し、クラスターの状態が正常である場合にのみ実行します。

- a. Cluster Network Operator 設定オブジェクトから移行設定を削除するには、以下のコマンドを入力します。

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

- b. OVN-Kubernetes 設定を削除するには、以下のコマンドを入力します。

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

- c. OVN-Kubernetes ネットワークプロバイダー namespace を削除するには、以下のコマンドを入力します。

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

16.4. IPV4/IPV6 デュアルスタックネットワークへの変換

クラスター管理者は、IPv4 および IPv6 アドレスファミリーをサポートするデュアルネットワーククラスターネットワークに、IPv4 の単一スタッククラスターを変換できます。デュアルスタックに変換した後、新規に作成された Pod はすべてデュアルスタック対応になります。



注記

デュアルスタックネットワークは、ベアメタル、IBM Power インフラストラクチャー、および単一ノードの OpenShift クラスターでプロビジョニングされたクラスターでサポートされます。

16.4.1. デュアルスタッククラスターネットワークへの変換

クラスター管理者は、単一スタッククラスターネットワークをデュアルスタッククラスターネットワークに変換できます。



注記

デュアルスタックネットワークへの変換後に、新規に作成された Pod のみに IPv6 アドレスが割り当てられます。変換前に作成された Pod は、IPv6 アドレスを受信するように再作成される必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていること。
- クラスターで OVN-Kubernetes CNI クラスターネットワークプロバイダーを使用している。
- クラスターノードに IPv6 アドレスがある。

手順

1. クラスターおよびサービスネットワークの IPv6 アドレスブロックを指定するには、以下の YAML を含むファイルを作成します。

```
- op: add
  path: /spec/clusterNetwork/-
  value: ①
    cidr: fd01::/48
    hostPrefix: 64
- op: add
  path: /spec/serviceNetwork/-
  value: fd02::/112 ②
```

- ① **cidr** および **hostPrefix** フィールドでオブジェクトを指定します。ホストの接頭辞は **64** 以上である必要があります。IPv6 CIDR 接頭辞は、指定されたホスト接頭辞に対応する十分な大きさである必要があります。
- ② 接頭辞が **112** である IPv6 CIDR を指定します。Kubernetes は最低レベルの 16 ビットのみを使用します。接頭辞が **112** の場合、IP アドレスは **112** から **128** ビットに割り当てられます。

2. クラスターネットワーク設定にパッチを適用するには、以下のコマンドを入力します。

```
$ oc patch network.config.openshift.io cluster \
  --type='json' --patch-file <file>.yaml
```

ここでは、以下のようになります。

file

先の手順で作成したファイルの名前を指定します。

出力例

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

検証

以下の手順を実施して、クラスターネットワークが直前の手順で指定した IPv6 アドレスブロックを認識していることを確認します。

1. ネットワーク設定を表示します。

```
$ oc describe network
```

出力例

```
Status:
Cluster Network:
  Cidr:      10.128.0.0/14
  Host Prefix: 23
  Cidr:      fd01::/48
  Host Prefix: 64
Cluster Network MTU: 1400
Network Type:      OVNKubernetes
Service Network:
  172.30.0.0/16
  fd02::/112
```

16.5. IPSEC 暗号化の設定

IPsec を有効にすると、OVN-Kubernetes Container Network Interface (CNI) クラスターネットワーク上のノード間のすべてのネットワークトラフィックは暗号化されたトンネルを通過します。

IPsec はデフォルトで無効にされています。



注記

IPsec 暗号化はクラスターのインストール時にのみ有効にでき、有効にした後は無効にすることはできません。インストールのドキュメントについては、[クラスターインストール方法の選択およびその使用に向けた準備](#) について参照してください。

16.5.1. IPsec で暗号化したネットワークトラフィックフローのタイプ

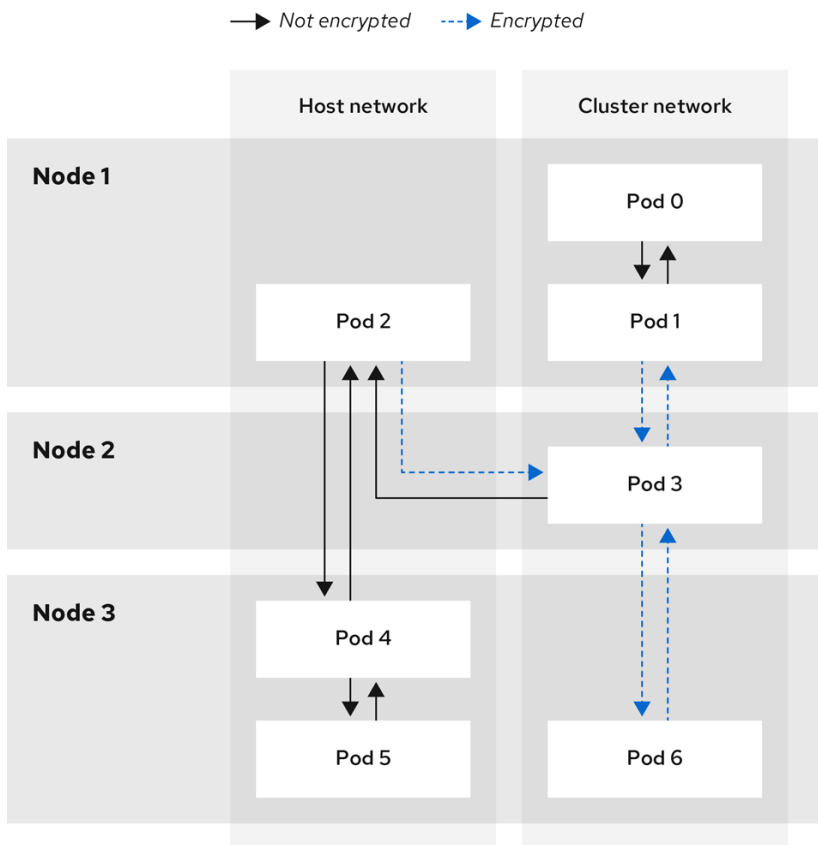
IPsec を有効にすると、Pod 間の以下のネットワークトラフィックフローのみが暗号化されます。

- クラスターネットワーク上の複数の異なるノードの Pod 間のトラフィック
- ホストネットワークの Pod からクラスターネットワーク上の Pod へのトラフィック

以下のトラフィックフローは暗号化されません。

- クラスターネットワーク上の同じノードの Pod 間のトラフィック
- ホストネットワーク上の Pod 間のトラフィック
- クラスターネットワークの Pod からホストネットワークの Pod へのトラフィック

暗号化されていないフローと暗号化されていないフローを以下の図に示します。



138_OpenShift_0421

16.5.1.1. IPsec が有効になっている場合のネットワーク接続要件

OpenShift Container Platform クラスターのコンポーネントが通信できるように、マシン間のネットワーク接続を設定する必要があります。すべてのマシンではクラスターの他のすべてのマシンのホスト名を解決する必要があります。

表16.7 すべてのマシンからすべてのマシンへの通信に使用されるポート

プロトコル	ポート	説明
UDP	500	IPsec IKE パケット
	4500	IPsec NAT-T パケット
ESP	該当なし	IPsec Encapsulating Security Payload (ESP)

16.5.2. 暗号化プロトコルおよび IPsec モード

使用する暗号化は **AES-GCM-16-256** です。整合性チェック値 (ICV) は **16** バイトです。鍵の長さは **256** ビットです。

使用される IPsec モードは **トランスポートモード** です。これは、元のパケットの IP ヘッダーに Encapsulated Security Payload (ESP) ヘッダーを追加してパケットデータを暗号化することで、エンドツーエンドの通信を暗号化するモードです。OpenShift Container Platform は現在、Pod 間通信に IPsec **Tunnel モード** を使用したり、サポートしたりしません。

16.5.3. セキュリティー証明書の生成およびローテーション

Cluster Network Operator (CNO) は、暗号化用に IPsec によって使用される自己署名の X.509 認証局 (CA) を生成します。各ノードの証明書署名要求 (CSR) は、CNO によって自動的に満たされます。

この CA は 10 年間有効です。個別のノード証明書は 5 年間有効で、4 年半が経過すると自動的にローテーションされます。

16.6. プロジェクトの EGRESS ファイアウォールの設定

クラスター管理者は、OpenShift Container Platform クラスター外に出るプロジェクトのプロジェクトについて、egress トラフィックを制限する egress ファイアウォールを作成できます。

16.6.1. egress ファイアウォールのプロジェクトでの機能

クラスター管理者は、**egress ファイアウォール** を使用して、一部またはすべての Pod がクラスター内からアクセスできる外部ホストを制限できます。egress ファイアウォールポリシーは以下のシナリオをサポートします。

- Pod の接続を内部ホストに制限し、パブリックインターネットへの接続を開始できないようにする。
- Pod の接続をパブリックインターネットに制限し、OpenShift Container Platform クラスター外にある内部ホストへの接続を開始できないようにする。
- Pod は OpenShift Container Platform クラスター外の指定された内部サブネットまたはホストにアクセスできません。
- Pod は特定の外部ホストにのみ接続することができます。

たとえば、指定された IP 範囲へのあるプロジェクトへのアクセスを許可する一方で、別のプロジェクトへの同じアクセスを拒否することができます。または、アプリケーション開発者の (Python) pip mirror からの更新を制限したり、更新を承認されたソースからの更新のみに強制的に制限したりすることができます。



注記

Egress ファイアウォールは、ホストネットワークの namespace には適用されません。ホストネットワークが有効になっている Pod は、Egress ファイアウォールルールの影響を受けません。

EgressFirewall カスタムリソース (CR) オブジェクトを作成して egress ファイアウォールポリシーを設定します。egress ファイアウォールは、以下のいずれかの基準を満たすネットワークトラフィックと一致します。

- CIDR 形式の IP アドレス範囲。
- IP アドレスに解決する DNS 名
- ポート番号
- プロトコル。TCP、UDP、および SCTP のいずれかになります。

重要

egress ファイアウォールに **0.0.0.0/0** の拒否ルールが含まれる場合、OpenShift Container Platform API サーバーへのアクセスはブロックされます。Pod が OpenShift Container Platform API サーバーへのアクセスを継続できるようにするには、以下の例にあるように API サーバーが egress ファイアウォールルールでリッスンする IP アドレス範囲を含める必要があります。

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
  namespace: <namespace> ❶
spec:
  egress:
  - to:
    cidrSelector: <api_server_address_range> ❷
    type: Allow
  # ...
  - to:
    cidrSelector: 0.0.0.0/0 ❸
    type: Deny
```

- ❶ egress ファイアウォールの namespace。
- ❷ OpenShift Container Platform API サーバーを含む IP アドレス範囲。
- ❸ グローバル拒否ルールにより、OpenShift Container Platform API サーバーへのアクセスが阻止されます。

API サーバーの IP アドレスを見つけるには、**oc get ep kubernetes -n default** を実行します。

詳細は、[BZ#1988324](#) を参照してください。



警告

egress ファイアウォールルールは、ルーターを通過するトラフィックには適用されません。ルート CR オブジェクトを作成するパーミッションを持つユーザーは、禁止されている宛先を参照するルートを作成することにより、egress ファイアウォールポリシールールをバイパスできます。

16.6.1.1. egress ファイアウォールの制限

egress ファイアウォールには以下の制限があります。

- 複数の EgressFirewall オブジェクトを持つプロジェクトはありません。
- 最大 8,000 のルールを持つ最大 1 つの EgressFirewall オブジェクトはプロジェクトごとに定義できます。
- Red Hat OpenShift Networking の共有ゲートウェイモードで OVN-Kubernetes ネットワークプ

ラグインを使用している場合に、リターン Ingress 応答は Egress ファイアウォールルールの影響を受けます。送信ファイアウォールルールが受信応答宛先 IP をドロップすると、トラフィックはドロップされます。

上記の制限のいずれかに違反すると、プロジェクトの egress ファイアウォールに障害が発生し、すべての外部ネットワークトラフィックがドロップされる可能性があります。

egress ファイアウォールリソースは、**kube-node-lease**、**kube-public**、**kube-system**、**openshift**、**openshift-** プロジェクトで作成できます。

16.6.1.2. egress ポリシールールのマッチング順序

egress ファイアウォールポリシールールは、最初から最後へと定義された順序で評価されます。Pod からの egress 接続に一致する最初のルールが適用されます。この接続では、後続のルールは無視されません。

16.6.1.3. DNS (Domain Name Server) 解決の仕組み

egress ファイアウォールポリシールールのいずれかで DNS 名を使用する場合、ドメイン名の適切な解決には、以下の制限が適用されます。

- ドメイン名の更新は、TTL (Time-to-live) 期間に基づいてポーリングされます。デフォルトで、期間は 30 分です。egress ファイアウォールコントローラーがローカルネームサーバーでドメイン名をクエリーする場合に、応答に 30 分未満の TTL が含まれる場合、コントローラーは DNS 名の期間を返される値に設定します。それぞれの DNS 名は、DNS レコードの TTL の期限が切れた後にクエリーされます。
- Pod は、必要に応じて同じローカルネームサーバーからドメインを解決する必要があります。そうしない場合、egress ファイアウォールコントローラーと Pod によって認識されるドメインの IP アドレスが異なる可能性があります。ホスト名の IP アドレスが異なる場合、egress ファイアウォールは一貫して実行されないことがあります。
- egress ファイアウォールコントローラーおよび Pod は同じローカルネームサーバーを非同期にポーリングするため、Pod は egress コントローラーが実行する前に更新された IP アドレスを取得する可能性があります。これにより、競合状態が生じます。この現時点の制限により、EgressFirewall オブジェクトのドメイン名の使用は、IP アドレスの変更が頻繁に生じないドメインの場合にのみ推奨されます。

注記

egress ファイアウォールは、DNS 解決用に Pod が置かれるノードの外部インターフェイスに Pod が常にアクセスできるようにします。

ドメイン名を egress ファイアウォールで使用し、DNS 解決がローカルノード上の DNS サーバーによって処理されない場合は、Pod でドメイン名を使用している場合には DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールを追加する必要があります。

16.6.2. EgressFirewall カスタムリソース (CR) オブジェクト

egress ファイアウォールのルールを 1 つ以上定義できます。ルールは、ルールが適用されるトラフィックを指定して **Allow** ルールまたは **Deny** ルールのいずれかになります。

以下の YAML は EgressFirewall CR オブジェクトについて説明しています。

EgressFirewall オブジェクト

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: <name> ❶
spec:
  egress: ❷
  ...
```

- ❶ オブジェクトの名前は **default** である必要があります。
- ❷ 以下のセクションで説明されているように、egress ネットワークポリシーールのコレクション。

16.6.2.1. EgressFirewall ルール

以下の YAML は egress ファイアウォールルールオブジェクトについて説明しています。**egress** スタンザは、単一または複数のオブジェクトの配列を予想します。

Egress ポリシーールのスタンザ

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

- ❶ ルールのタイプ。値には **Allow** または **Deny** のいずれかを指定する必要があります。
- ❷ **cidrSelector** フィールドまたは **dnsName** フィールドを指定する egress トラフィックのマッチングルールを記述するスタンザ。同じルールで両方のフィールドを使用することはできません。
- ❸ CIDR 形式の IP アドレス範囲。
- ❹ DNS ドメイン名。
- ❺ オプション: ルールのネットワークポートおよびプロトコルのコレクションを記述するスタンザ。

ポートスタンザ

```
ports:
- port: <port> ❶
  protocol: <protocol> ❷
```

- ❶ **80** や **443** などのネットワークポート。このフィールドの値を指定する場合は、**protocol** の値も指定する必要があります。
- ❷ ネットワークプロトコル。値は **TCP**、**UDP**、または **SCTP** のいずれかである必要があります。

16.6.2.2. EgressFirewall CR オブジェクトの例

以下の例では、複数の egress ファイアウォールポリシールールを定義します。

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress: 1
  - type: Allow
    to:
      cidrSelector: 1.2.3.0/24
  - type: Deny
    to:
      cidrSelector: 0.0.0.0/0
```

1 egress ファイアウォールポリシールールオブジェクトのコレクション。

以下の例では、トラフィックが TCP プロトコルおよび宛先ポート **80** または任意のプロトコルと宛先ポート **443** のいずれかを使用している場合に、IP アドレス **172.16.1.1** でホストへのトラフィックを拒否するポリシールールを定義します。

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
  egress:
  - type: Deny
    to:
      cidrSelector: 172.16.1.1
    ports:
      - port: 80
      protocol: TCP
      - port: 443
```

16.6.3. egress ファイアウォールポリシーオブジェクトの作成

クラスター管理者は、プロジェクトの egress ファイアウォールポリシーオブジェクトを作成できます。



重要

プロジェクトに EgressFirewall オブジェクトがすでに定義されている場合、既存のポリシーを編集して egress ファイアウォールルールを変更する必要があります。

前提条件

- OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) がインストールされている。

- クラスター管理者としてクラスターにログインする必要があります。

手順

1. ポリシールールを作成します。
 - a. **<policy_name>.yaml** ファイルを作成します。この場合、**<policy_name>** は egress ポリシールールを記述します。
 - b. 作成したファイルで、egress ポリシーオブジェクトを定義します。
2. 以下のコマンドを入力してポリシーオブジェクトを作成します。**<policy_name>** をポリシーの名前に、**<project>** をルールが適用されるプロジェクトに置き換えます。

```
$ oc create -f <policy_name>.yaml -n <project>
```

以下の例では、新規の EgressFirewall オブジェクトが **project1** という名前のプロジェクトに作成されます。

```
$ oc create -f default.yaml -n project1
```

出力例

```
egressfirewall.k8s.ovn.org/v1 created
```

3. オプション: 後に変更できるように **<policy_name>.yaml** ファイルを保存します。

16.7. プロジェクトの EGRESS ファイアウォールの表示

クラスター管理者は、既存の egress ファイアウォールの名前を一覧表示し、特定の egress ファイアウォールのトラフィックルールを表示できます。

16.7.1. EgressFirewall オブジェクトの表示

クラスターで EgressFirewall オブジェクトを表示できます。

前提条件

- OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- **oc** として知られる OpenShift コマンドラインインターフェイス (CLI) のインストール。
- クラスターにログインすること。

手順

1. オプション: クラスターで定義された EgressFirewall オブジェクトの名前を表示するには、以下のコマンドを入力します。

```
$ oc get egressfirewall --all-namespaces
```

2. ポリシーを検査するには、以下のコマンドを入力します。**<policy_name>** を検査するポリシーの名前に置き換えます。

```
$ oc describe egressfirewall <policy_name>
```

出力例

```
Name: default
Namespace: project1
Created: 20 minutes ago
Labels: <none>
Annotations: <none>
Rule: Allow to 1.2.3.0/24
Rule: Allow to www.example.com
Rule: Deny to 0.0.0.0/0
```

16.8. プロジェクトの EGRESS ファイアウォールの編集

クラスター管理者は、既存の egress ファイアウォールのネットワークトラフィックルールを変更できます。

16.8.1. EgressFirewall オブジェクトの編集

クラスター管理者は、プロジェクトの egress ファイアウォールを更新できます。

前提条件

- OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者としてクラスターにログインする必要があります。

手順

1. プロジェクトの EgressFirewall オブジェクトの名前を検索します。**<project>** をプロジェクトの名前に置き換えます。

```
$ oc get -n <project> egressfirewall
```

2. オプション: egress ネットワークファイアウォールの作成時に EgressFirewall オブジェクトのコピーを保存しなかった場合には、以下のコマンドを入力してコピーを作成します。

```
$ oc get -n <project> egressfirewall <name> -o yaml > <filename>.yaml
```

<project> をプロジェクトの名前に置き換えます。**<name>** をオブジェクトの名前に置き換えます。**<filename>** をファイルの名前に置き換え、YAML を保存します。

3. ポリシールールに変更を加えたら、以下のコマンドを実行して EgressFirewall オブジェクトを置き換えます。**<filename>** を、更新された EgressFirewall オブジェクトを含むファイルの名前に置き換えます。

```
$ oc replace -f <filename>.yaml
```

16.9. プロジェクトからの EGRESS ファイアウォールの削除

クラスター管理者は、プロジェクトから egress ファイアウォールを削除して、OpenShift Container Platform クラスター外に出るプロジェクトからネットワークトラフィックについてのすべての制限を削除できます。

16.9.1. EgressFirewall オブジェクトの削除

クラスター管理者は、プロジェクトから Egress ファイアウォールを削除できます。

前提条件

- OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダープラグインを使用するクラスター。
- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者としてクラスターにログインする必要があります。

手順

1. プロジェクトの EgressFirewall オブジェクトの名前を検索します。<project> をプロジェクトの名前に置き換えます。

```
$ oc get -n <project> egressfirewall
```

2. 以下のコマンドを入力し、EgressFirewall オブジェクトを削除します。<project> をプロジェクトの名前に、<name> をオブジェクトの名前に置き換えます。

```
$ oc delete -n <project> egressfirewall <name>
```

16.10. EGRESS IP アドレスの設定

クラスター管理者は、1つ以上の egress IP アドレスを namespace に、または namespace 内の特定の pod に割り当てるように、OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダーを設定することができます。

16.10.1. Egress IP アドレスアーキテクチャーの設計および実装

OpenShift Container Platform の egress IP アドレス機能を使用すると、1つ以上の namespace の1つ以上の Pod からのトラフィックに、クラスターネットワーク外のサービスに対する一貫したソース IP アドレスを持たせることができます。

たとえば、クラスター外のサーバーでホストされるデータベースを定期的にクエリーする Pod がある場合があります。サーバーにアクセス要件を適用するために、パケットフィルターリングデバイスは、特定の IP アドレスからのトラフィックのみを許可するよう設定されます。この特定の Pod のみからサーバーに確実にアクセスできるようにするには、サーバーに要求を行う Pod に特定の egress IP アドレスを設定できます。

egress IP アドレスは、ノードのプライマリーネットワークインターフェイスの追加 IP アドレスとして実装され、ノードのプライマリー IP アドレスと同じサブネットにある必要があります。追加の IP アドレスは、クラスター内の他のノードには割り当てないでください。

一部のクラスター設定では、アプリケーション Pod と Ingress ルーター Pod が同じノードで実行されます。このシナリオでアプリケーションプロジェクトの Egress IP アドレスを設定する場合、アプリケーションプロジェクトからルートに要求を送信するときに IP アドレスは使用されません。

16.10.1.1. プラットフォームサポート

各種のプラットフォームでの egress IP アドレス機能のサポートについては、以下の表で説明されています。



重要

egress IP アドレスの実装は、Amazon Web Services (AWS)、Azure Cloud、または egress IP 機能に必要な自動レイヤー 2 ネットワーク操作と互換性のない他のパブリッククラウドプラットフォームと互換性がありません。

プラットフォーム	サポート対象
ベアメタル	はい
vSphere	はい
Red Hat OpenStack Platform (RHOSP)	いいえ
パブリッククラウド	いいえ

16.10.1.2. egress IP の Pod への割り当て

1つ以上の egress IP を namespace に、または namespace の特定の Pod に割り当てるには、以下の条件を満たす必要があります。

- クラスター内の1つ以上のノードに **k8s.ovn.org/egress-assignable: ""** ラベルがなければなりません。
- **EgressIP** オブジェクトが存在し、これは namespace の Pod からクラスターを離脱するトラフィックのソース IP アドレスとして使用する1つ以上の egress IP アドレスを定義します。



重要

egress IP の割り当て用にクラスター内のノードにラベルを付ける前に **EgressIP** オブジェクトを作成する場合、OpenShift Container Platform は **k8s.ovn.org/egress-assignable: ""** ラベルですべての egress IP アドレスを最初のノードに割り当てる可能性があります。

egress IP アドレスがクラスター内のノード全体に広く分散されるようにするには、**EgressIP** オブジェクトを作成する前に、egress IP アドレスをホストする予定のノードにラベルを常に適用します。

16.10.1.3. egress IP のノードへの割り当て

EgressIP オブジェクトを作成する場合、`k8s.ovn.org/egress-assignable: ""` ラベルのラベルが付いたノードに以下の条件が適用されます。

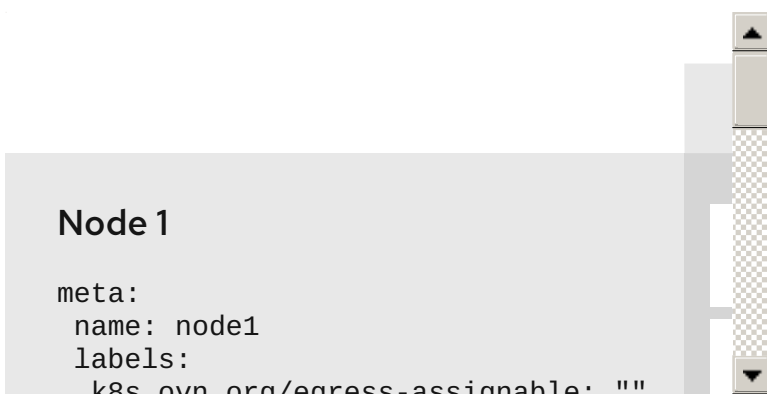
- egress IP アドレスは一度に複数のノードに割り当てられることはありません。
- egress IP アドレスは、egress IP アドレスをホストできる利用可能なノード間で均等に分散されます。
- **EgressIP** オブジェクトの `spec.EgressIPs` 配列が複数の IP アドレスを指定する場合は、以下の条件が適用されます。
 - 指定された IP アドレスを複数ホストするノードはありません。
 - トラフィックは、指定された namespace の指定された IP アドレス間でほぼ均等に分散されます。
- ノードが利用不可の場合、そのノードに割り当てられる egress IP アドレスは自動的に再割り当てされます (前述の条件が適用されます)。

Pod が複数の **EgressIP** オブジェクトのセレクターに一致する場合、**EgressIP** オブジェクトに指定される egress IP アドレスのどれが Pod の egress IP アドレスとして割り当てられるのかという保証はありません。

さらに、**EgressIP** オブジェクトが複数の送信 IP アドレスを指定する場合、どの送信 IP アドレスが使用されるかは保証されません。たとえば、Pod が `10.10.20.1` と `10.10.20.2` の 2 つの egress IP アドレスを持つ **EgressIP** オブジェクトのセレクターと一致する場合、各 TCP 接続または UDP 会話にいずれかが使用される可能性があります。

16.10.1.4. egress IP アドレス設定のアーキテクチャー図

以下の図は、egress IP アドレス設定を示しています。この図では、クラスターの 3 つのノードで実行される 2 つの異なる namespace の 4 つの Pod について説明します。ノードには、ホストネットワークの `192.168.126.0/18` CIDR ブロックから IP アドレスが割り当てられます。



ノード 1 とノード 3 の両方に `k8s.ovn.org/egress-assignable: ""` というラベルが付けられるため、egress IP アドレスの割り当てに利用できます。

図の破線は、pod1、pod2、および pod3 からのトラフィックフローが Pod ネットワークを通過し、クラスターがノード 1 およびノード 3 から出る様子を示しています。外部サービスが、**EgressIP** オブジェクトの例で選択した Pod からトラフィックを受信する場合、ソース IP アドレスは `192.168.126.10` または `192.168.126.102` のいずれかになります。トラフィックはこれらの 2 つのノード間でほぼ均等に分散されます。

図にある次のリソースの詳細を以下に示します。

Namespace オブジェクト

namespace は以下のマニフェストで定義されます。

namespace オブジェクト

```

apiVersion: v1
kind: Namespace
metadata:
  name: namespace1
  labels:
    env: prod
---
apiVersion: v1
kind: Namespace
metadata:
  name: namespace2
  labels:
    env: prod

```

EgressIP オブジェクト

以下の **EgressIP** オブジェクトは、**env** ラベルが **prod** に設定される namespace のすべての Pod を選択する設定を説明しています。選択された Pod の egress IP アドレスは **192.168.126.10** および **192.168.126.102** です。

EgressIP オブジェクト

```

apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egressips-prod
spec:
  egressIPs:
    - 192.168.126.10
    - 192.168.126.102
  namespaceSelector:
    matchLabels:
      env: prod
status:
  items:
    - node: node1
      egressIP: 192.168.126.10
    - node: node3
      egressIP: 192.168.126.102

```

直前の例の設定の場合、OpenShift Container Platform は両方の egress IP アドレスを利用可能なノードに割り当てます。**status** フィールドは、egress IP アドレスの割り当ての有無および割り当てられる場所を反映します。

16.10.2. EgressIP オブジェクト

以下の YAML は、**EgressIP** オブジェクトの API について説明しています。オブジェクトの範囲はクラスター全体です。これは namespace では作成されません。

-

```

apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: <name> ❶
spec:
  egressIPs: ❷
  - <ip_address>
  namespaceSelector: ❸
  ...
  podSelector: ❹
  ...

```

- ❶ **EgressIPs** オブジェクトの名前。
- ❷ 1つ以上の IP アドレスの配列。
- ❸ egress IP アドレスを関連付ける namespace の1つ以上のセレクター。
- ❹ オプション: egress IP アドレスを関連付けるための指定された namespace の Pod の1つ以上のセレクター。これらのセレクターを適用すると、namespace 内の Pod のサブセットを選択できます。

以下の YAML は namespace セレクターのスタンザについて説明しています。

namespace セレクタースタンザ

```

namespaceSelector: ❶
matchLabels:
  <label_name>: <label_value>

```

- ❶ namespace の1つ以上のマッチングルール。複数のマッチングルールを指定すると、一致するすべての namespace が選択されます。

以下の YAML は Pod セレクターのオプションのスタンザについて説明しています。

Pod セレクタースタンザ

```

podSelector: ❶
matchLabels:
  <label_name>: <label_value>

```

- ❶ オプション: 指定された **namespaceSelector** ルールに一致する、namespace の Pod の1つ以上のマッチングルール。これが指定されている場合、一致する Pod のみが選択されます。namespace の他の Pod は選択されていません。

以下の例では、**EgressIP** オブジェクトは **192.168.126.11** および **192.168.126.102** egress IP アドレスを、**app** ラベルが **web** に設定されており、**env** ラベルが **prod** に設定されている namespace にある Pod に関連付けます。

EgressIP オブジェクトの例

```

apiVersion: k8s.ovn.org/v1

```



```

kind: EgressIP
metadata:
  name: egress-group1
spec:
  egressIPs:
  - 192.168.126.11
  - 192.168.126.102
  podSelector:
    matchLabels:
      app: web
  namespaceSelector:
    matchLabels:
      env: prod

```

以下の例では、**EgressIP** オブジェクトは、**192.168.127.30** および **192.168.127.40** egress IP アドレスを、**environment** ラベルが **development** に設定されていない Pod に関連付けます。

EgressIP オブジェクトの例

```

apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-group2
spec:
  egressIPs:
  - 192.168.127.30
  - 192.168.127.40
  namespaceSelector:
    matchExpressions:
    - key: environment
      operator: NotIn
      values:
      - development

```

16.10.3. egress IP アドレスをホストするノードのラベル付け

OpenShift Container Platform が1つ以上の egress IP アドレスをノードに割り当てることができるように、**k8s.ovn.org/egress-assignable=""** ラベルをクラスター内のノードに適用することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者としてクラスターにログインします。

手順

- 1つ以上の egress IP アドレスをホストできるようにノードにラベルを付けるには、以下のコマンドを入力します。

```
$ oc label nodes <node_name> k8s.ovn.org/egress-assignable="" 1
```

- 1 ラベルを付けるノードの名前。

ヒント

または、以下の YAML を適用してラベルをノードに追加できます。

```
apiVersion: v1
kind: Node
metadata:
  labels:
    k8s.ovn.org/egress-assignable: ""
name: <node_name>
```

16.10.4. 次のステップ

- [egress IP の割り当て](#)

16.10.5. 関連情報

- [LabelSelector meta/v1](#)
- [LabelSelectorRequirement meta/v1](#)

16.11. EGRESS IP アドレスの割り当て

クラスター管理者は、namespace または namespace の特定の Pod からクラスターを出るトラフィックに egress IP アドレスを割り当てることができます。

16.11.1. egress IP アドレスの namespace への割り当て

1つ以上の egress IP アドレスを namespace または namespace の特定の Pod に割り当てることができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者としてクラスターにログインします。
- egress IP アドレスをホストするように1つ以上のノードを設定します。

手順

1. **EgressIP** オブジェクトを作成します。
 - a. **<egressips_name>.yaml** ファイルを作成します。<egressips_name> はオブジェクトの名前になります。
 - b. 作成したファイルで、以下の例のように **EgressIPs** オブジェクトを定義します。

```
apiVersion: k8s.ovn.org/v1
kind: EgressIP
metadata:
  name: egress-project1
spec:
  egressIPs:
```

```
- 192.168.127.10
- 192.168.127.11
namespaceSelector:
matchLabels:
  env: qa
```

- オブジェクトを作成するには、以下のコマンドを入力します。

```
$ oc apply -f <egressips_name>.yaml ❶
```

- ❶** `<egressips_name>` をオブジェクトの名前に置き換えます。

出力例

```
egressips.k8s.ovn.org/<egressips_name> created
```

- オプション: 後に変更できるように `<egressips_name>.yaml` ファイルを保存します。
- egress IP アドレスを必要とする namespace にラベルを追加します。手順1で定義した **Egress IP** オブジェクトの namespace にラベルを追加するには、以下のコマンドを実行します。

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

- ❶** `<namespace>` は、egress IP アドレスを必要とする namespace に置き換えてください。

16.11.2. 関連情報

- [egress IP アドレスの設定](#)

16.12. EGRESS ルーター POD の使用についての考慮事項

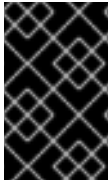
16.12.1. egress ルーター Pod について

OpenShift Container Platform egress ルーター Pod は、他の用途で使用されていないプライベートソース IP アドレスから指定されたリモートサーバーにトラフィックをリダイレクトします。Egress ルーター Pod により、特定の IP アドレスからのアクセスのみを許可するように設定されたサーバーにネットワークトラフィックを送信できます。



注記

egress ルーター Pod はすべての発信接続のために使用されることが意図されていません。多数の egress ルーター Pod を作成することで、ネットワークハードウェアの制限を引き上げられる可能性があります。たとえば、すべてのプロジェクトまたはアプリケーションに egress ルーター Pod を作成すると、ソフトウェアの MAC アドレスのフィルタに戻る前にネットワークインターフェイスが処理できるローカル MAC アドレス数の上限を超えてしまう可能性があります。



重要

egress ルーターイメージには Amazon AWS, Azure Cloud またはレイヤー 2 操作をサポートしないその他のクラウドプラットフォームとの互換性がありません。それらに macvlan トラフィックとの互換性がないためです。

16.12.1.1. Egress ルーターモード

リダイレクトモードでは、egress ルーター Pod は、トラフィックを独自の IP アドレスから 1 つ以上の宛先 IP アドレスにリダイレクトするために **iptables** ルールをセットアップします。予約されたソース IP アドレスを使用する必要があるクライアント Pod は、宛先 IP に直接接続するのではなく、egress ルーターに接続するように変更される必要があります。



注記

egress ルーター CNI プラグインはリダイレクトモードのみをサポートします。これは、OpenShift SDN でデプロイできる egress ルーター実装の相違点です。OpenShift SDN の Egress ルーターとは異なり、Egress ルーター CNI プラグインは HTTP プロキシモードまたは DNS プロキシモードをサポートしません。

16.12.1.2. egress ルーター Pod の実装

egress ルーターの実装では、egress ルーターの Container Network Interface (CNI) プラグインを使用します。プラグインはセカンダリーネットワークインターフェイスを Pod に追加します。

egress ルーターは、2 つのネットワークインターフェイスを持つ Pod です。たとえば、Pod には、**eth0** および **net1** ネットワークインターフェイスを使用できます。**eth0** インターフェイスはクラスターネットワークにあり、Pod は通常のクラスター関連のネットワークトラフィックにこのインターフェイスを引き続き使用します。**net1** インターフェイスはセカンダリーネットワークにあり、そのネットワークの IP アドレスとゲートウェイを持ちます。OpenShift Container Platform クラスターの他の Pod は egress ルーターサービスにアクセスでき、サービスにより Pod が外部サービスにアクセスできるようになります。egress ルーターは、Pod と外部システム間のブリッジとして機能します。

egress ルーターから出るトラフィックはノードで終了しますが、パケットには egress ルーター Pod からの **net1** インターフェイスの MAC アドレスがあります。

Egress ルーターのカスタムリソースを追加すると、Cluster Network Operator は以下のオブジェクトを作成します。

- Pod の **net1** セカンダリーネットワークインターフェイス用のネットワーク接続定義。
- Egress ルーターのデプロイメント。

Egress ルーターカスタムリソースを削除する場合、Operator は Egress ルーターに関連付けられた直前の一覧の 2 つのオブジェクトを削除します。

16.12.1.3. デプロイメントに関する考慮事項

egress ルーター Pod は追加の IP アドレスおよび MAC アドレスをノードのプライマリーネットワークインターフェイスに追加します。その結果、ハイパーバイザーまたはクラウドプロバイダーを、追加のアドレスを許可するように設定する必要がある場合があります。

Red Hat OpenStack Platform (RHOSP)

OpenShift Container Platform を RHOSP にデプロイする場合、OpenStack 環境の egress ルーター Pod の IP および MAC アドレスからのトラフィックを許可する必要があります。トラフィックを許可しないと、**通信は失敗** します。

```
$ openstack port set --allowed-address \  
ip_address=<ip_address>,mac_address=<mac_address> <neutron_port_uuid>
```

Red Hat Virtualization (RHV)

RHV を使用している場合は、仮想インターフェイスカード (vNIC) に **No Network Filter** を選択する必要があります。

VMware vSphere

VMware vSphere を使用している場合は、[vSphere 標準スイッチのセキュリティー保護についての VMware ドキュメント](#) を参照してください。vSphere Web クライアントからホストの仮想スイッチを選択して、VMware vSphere デフォルト設定を表示し、変更します。

とくに、以下が有効にされていることを確認します。

- [MAC アドレスの変更](#)
- [偽装転送 \(Forged Transit\)](#)
- [無作為別モード \(Promiscuous Mode\) 操作](#)

16.12.1.4. フェイルオーバー設定

ダウンタイムを回避するために、Cluster Network Operator は Egress ルーター Pod をデプロイメントリソースとしてデプロイします。デプロイメント名は **egress-router-cni-deployment** です。デプロイメントに対応する Pod には **app=egress-router-cni** のラベルがあります。

デプロイメントの新規サービスを作成するには、**oc expose deployment/egress-router-cni-deployment --port <port_number>** コマンドを使用するか、以下のようにファイルを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: app-egress
spec:
  ports:
    - name: tcp-8080
      protocol: TCP
      port: 8080
    - name: tcp-8443
      protocol: TCP
      port: 8443
    - name: udp-80
      protocol: UDP
      port: 80
  type: ClusterIP
  selector:
    app: egress-router-cni
```

16.12.2. 関連情報

- [リダイレクトモードでの egress ルーターのデプロイ](#)

16.13. リダイレクトモードでの EGRESS ルーター POD のデプロイ

クラスター管理者は、トラフィックを予約されたソース IP アドレスから指定された宛先 IP アドレスにリダイレクトするように egress ルーター Pod をデプロイできます。

egress ルーターの実装では、egress ルーターの Container Network Interface (CNI) プラグインを使用します。

16.13.1. Egress ルーターのカスタムリソース

Egress ルーターのカスタムリソースで Egress ルーター Pod の設定を定義します。以下の YAML は、リダイレクトモードでの Egress ルーターの設定のフィールドについて説明しています。

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: <egress_router_name>
  namespace: <namespace> <.>
spec:
  addresses: [ <.>
    {
      ip: "<egress_router>", <.>
      gateway: "<egress_gateway>" <.>
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [ <.>
      {
        destinationIP: "<egress_destination>",
        port: <egress_router_port>,
        targetPort: <target_port>, <.>
        protocol: <network_protocol> <.>
      },
      ...
    ],
    fallbackIP: "<egress_destination>" <.>
  }
}
```

<.> オプション: **namespace** フィールドは、Egress ルーターを作成するための namespace を指定します。ファイルまたはコマンドラインで値を指定しない場合には、**default** namespace が使用されます。

<.> **addresses** フィールドは、セカンダリーネットワークインターフェイスに設定する IP アドレスを指定します。

<.> **ip** フィールドは、ノードが Egress ルーター Pod と使用する物理ネットワークからの予約済みソース IP アドレスとネットマスクを指定します。CIDR 表記を使用して IP アドレスとネットマスクを指定します。

<.> **gateway** フィールドは、ネットワークゲートウェイの IP アドレスを指定します。

<.> オプション: **redirectRules** フィールドは、Egress 宛先 IP アドレス、Egress ルーターポート、およびプロトコルの組み合わせを指定します。指定されたポートとプロトコルでの Egress ルーターへの着信接続は、宛先 IP アドレスにルーティングされます。

<> オプション: **targetPort** フィールドは、宛先 IP アドレスのネットワークポートを指定します。このフィールドが指定されていない場合、トラフィックは到達したネットワークポートと同じネットワークポートにルーティングされます。

<> **protocol** フィールドは TCP、UDP、または SCTP をサポートします。

<> オプション: **fallbackIP** フィールドは、宛先 IP アドレスを指定します。リダイレクトルールを指定しない場合、Egress ルーターはすべてのトラフィックをこのフォールバック IP アドレスに送信します。リダイレクトルールを指定する場合、ルールに定義されていないネットワークポートへの接続は、Egress ルーターによってこのフォールバック IP アドレスに送信されます。このフィールドを指定しない場合、Egress ルーターはルールで定義されていないネットワークポートへの接続を拒否します。

egress ルーター仕様の例

```
apiVersion: network.operator.openshift.io/v1
kind: EgressRouter
metadata:
  name: egress-router-redirect
spec:
  networkInterface: {
    macvlan: {
      mode: "Bridge"
    }
  }
  addresses: [
    {
      ip: "192.168.12.99/24",
      gateway: "192.168.12.1"
    }
  ]
  mode: Redirect
  redirect: {
    redirectRules: [
      {
        destinationIP: "10.0.0.99",
        port: 80,
        protocol: UDP
      },
      {
        destinationIP: "203.0.113.26",
        port: 8080,
        targetPort: 80,
        protocol: TCP
      },
      {
        destinationIP: "203.0.113.27",
        port: 8443,
        targetPort: 443,
        protocol: TCP
      }
    ]
  }
}
```

16.13.2. リダイレクトモードでの Egress ルーターのデプロイ

egress ルーターをデプロイして、独自の予約済みソース IP アドレスから1つ以上の宛先 IP アドレスにトラフィックをリダイレクトできます。

egress ルーターを追加した後に、予約済みソース IP アドレスを使用する必要があるクライアント Pod は、宛先 IP に直接接続するのではなく、egress ルーターに接続するように変更される必要があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. egress ルーター定義の作成
2. 他の Pod が egress ルーター Pod の IP アドレスを見つられるようにするには、以下の例のように、egress ルーターを使用するサービスを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: egress-1
spec:
  ports:
  - name: web-app
    protocol: TCP
    port: 8080
  type: ClusterIP
  selector:
    app: egress-router-cni <.>
```

<.> egress ルーターのラベルを指定します。表示されている値は Cluster Network Operator によって追加され、設定不可能です。

サービスの作成後に、Pod はサービスに接続できます。egress ルーター Pod は、トラフィックを宛先 IP アドレスの対応するポートにリダイレクトします。接続は、予約されたソース IP アドレスを起点とします。

検証

Cluster Network Operator が egress ルーターを起動したことを確認するには、以下の手順を実行します。

1. Operator が egress ルーター用に作成したネットワーク接続定義を表示します。

```
$ oc get network-attachment-definition egress-router-cni-nad
```

ネットワーク接続定義の名前は設定できません。

出力例

```
NAME          AGE
egress-router-cni-nad 18m
```


2. egress ルーター Pod のデプロイメントを表示します。

```
$ oc get deployment egress-router-cni-deployment
```

デプロイメントの名前は設定できません。

出力例

```
NAME                                READY UP-TO-DATE AVAILABLE AGE
egress-router-cni-deployment 1/1    1          1      18m
```

3. egress ルーター Pod のステータスを表示します。

```
$ oc get pods -l app=egress-router-cni
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
egress-router-cni-deployment-575465c75c-qkq6m 1/1    Running 0      18m
```

4. egress ルーター Pod のログとルーティングテーブルを表示します。

- a. egress ルーター Pod のノード名を取得します。

```
$ POD_NODENAME=$(oc get pod -l app=egress-router-cni -o jsonpath="{.items[0].spec.nodeName}")
```

- b. ターゲットノードのデバッグセッションに入ります。この手順は、**<node_name>-debug** というデバッグ Pod をインスタンス化します。

```
$ oc debug node/$POD_NODENAME
```

- c. **/host** をデバッグシェル内の root ディレクトリーとして設定します。デバッグ Pod は、Pod 内の **/host** にホストのルートファイルシステムをマウントします。ルートディレクトリーを **/host** に変更すると、ホストの実行可能パスに含まれるバイナリーを実行できます。

```
# chroot /host
```

- d. **chroot** 環境コンソール内から、egress ルーターログを表示します。

```
# cat /tmp/egress-router-log
```

出力例

```
2021-04-26T12:27:20Z [debug] Called CNI ADD
2021-04-26T12:27:20Z [debug] Gateway: 192.168.12.1
2021-04-26T12:27:20Z [debug] IP Source Addresses: [192.168.12.99/24]
2021-04-26T12:27:20Z [debug] IP Destinations: [80 UDP 10.0.0.99/30 8080 TCP
203.0.113.26/30 80 8443 TCP 203.0.113.27/30 443]
2021-04-26T12:27:20Z [debug] Created macvlan interface
2021-04-26T12:27:20Z [debug] Renamed macvlan to "net1"
2021-04-26T12:27:20Z [debug] Adding route to gateway 192.168.12.1 on macvlan interface
```

```

2021-04-26T12:27:20Z [debug] deleted default route {lindex: 3 Dst: <nil> Src: <nil> Gw:
10.128.10.1 Flags: [] Table: 254}
2021-04-26T12:27:20Z [debug] Added new default route with gateway 192.168.12.1
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p
UDP --dport 80 -j DNAT --to-destination 10.0.0.99
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p
TCP --dport 8080 -j DNAT --to-destination 203.0.113.26:80
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p
TCP --dport 8443 -j DNAT --to-destination 203.0.113.27:443
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat -o net1 -j SNAT --to-
source 192.168.12.99

```

この手順で説明されているように、**EgressRouter** オブジェクトを作成して egress ルーターを起動する場合、ロギングファイルの場所とロギングレベルは設定できません。

- e. **chroot** 環境コンソール内で、コンテナ ID を取得します。

```
# crictl ps --name egress-router-cni-pod | awk '{print $1}'
```

出力例

```
CONTAINER
bac9fae69ddb6
```

- f. コンテナのプロセス ID を判別します。この例では、コンテナ ID は **bac9fae69ddb6** です。

```
# crictl inspect -o yaml bac9fae69ddb6 | grep 'pid:' | awk '{print $2}'
```

出力例

```
68857
```

- g. コンテナのネットワーク namespace を入力します。

```
# nsenter -n -t 68857
```

- h. ルーティングテーブルを表示します。

```
# ip route
```

以下の出力例では、**net1** ネットワークインターフェイスはデフォルトのルートです。クラスターネットワークのトラフィックは **eth0** ネットワークインターフェイスを使用します。**192.168.12.0/24** ネットワークのトラフィックは、**net1** ネットワークインターフェイスを使用し、予約されたソース IP アドレス **192.168.12.99** を起点とします。Pod は他のすべてのトラフィックを IP アドレス **192.168.12.1** のゲートウェイにルーティングします。サービスネットワークのルーティングは表示されません。

出力例

```
default via 192.168.12.1 dev net1
10.128.10.0/23 dev eth0 proto kernel scope link src 10.128.10.18
192.168.12.0/24 dev net1 proto kernel scope link src 192.168.12.99
192.168.12.1 dev net1
```

16.14. プロジェクトのマルチキャストの有効化

16.14.1. マルチキャストについて

IP マルチキャストを使用すると、データが多数の IP アドレスに同時に配信されます。



重要

現時点で、マルチキャストは低帯域幅の調整またはサービスの検出での使用に最も適しており、高帯域幅のソリューションとしては適していません。

OpenShift Container Platform の Pod 間のマルチキャストトラフィックはデフォルトで無効にされます。OVN-Kubernetes デフォルト Container Network Interface (CNI) ネットワークプロバイダーを使用している場合には、プロジェクトごとにマルチキャストを有効にすることができます。

16.14.2. Pod 間のマルチキャストの有効化

プロジェクトの Pod でマルチキャストを有効にすることができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行し、プロジェクトのマルチキャストを有効にします。<namespace> を、マルチキャストを有効にする必要のある namespace に置き換えます。

```
$ oc annotate namespace <namespace> \
  k8s.ovn.org/multicast-enabled=true
```

ヒント

または、以下の YAML を適用してアノテーションを追加できます。

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
  annotations:
    k8s.ovn.org/multicast-enabled: "true"
```

検証

マルチキャストがプロジェクトについて有効にされていることを確認するには、以下の手順を実行します。

1. 現在のプロジェクトを、マルチキャストを有効にしたプロジェクトに切り替えます。<project> をプロジェクト名に置き換えます。

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

2. マルチキャストレシーバーとして機能する Pod を作成します。

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: mlistener
  labels:
    app: multicast-verify
spec:
  containers:
  - name: mlistener
    image: registry.access.redhat.com/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat hostname && sleep inf"]
    ports:
    - containerPort: 30102
      name: mlistener
      protocol: UDP
EOF
```

3. マルチキャストセNDERとして機能する Pod を作成します。

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
  - name: msender
    image: registry.access.redhat.com/ubi8
    command: ["/bin/sh", "-c"]
    args:
      ["dnf -y install socat && sleep inf"]
EOF
```

4. 新しいターミナルウィンドウまたはタブで、マルチキャストリスナーを起動します。

- a. Pod の IP アドレスを取得します。

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

- b. 次のコマンドを入力して、マルチキャストリスナーを起動します。

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

5. マルチキャストトランスミッターを開始します。

a. Pod ネットワーク IP アドレス範囲を取得します。

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

b. マルチキャストメッセージを送信するには、以下のコマンドを入力します。

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

マルチキャストが機能している場合、直前のコマンドは以下の出力を返します。

```
mlistener
```

16.15. プロジェクトのマルチキャストの無効化

16.15.1. Pod 間のマルチキャストの無効化

プロジェクトの Pod でマルチキャストを無効にすることができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインする必要があります。

手順

- 以下のコマンドを実行して、マルチキャストを無効にします。

```
$ oc annotate namespace <namespace> \ 1
k8s.ovn.org/multicast-enabled-
```

- 1** マルチキャストを無効にする必要のあるプロジェクトの **namespace**。

ヒント

または、以下の YAML を適用してアノテーションを削除できます。

```
apiVersion: v1
kind: Namespace
metadata:
  name: <namespace>
annotations:
  k8s.ovn.org/multicast-enabled: null
```

16.16. ネットワークフローの追跡

クラスター管理者は、以下の領域をサポートする、クラスターからの Pod ネットワークフローについての情報を収集できます。

- Pod ネットワークで ingress および egress トラフィックをモニターします。
- パフォーマンスに関する問題のトラブルシューティング
- 容量計画およびセキュリティ監査に関するデータを収集します。

ネットワークフローのコレクションを有効にすると、トラフィックに関するメタデータのみが収集されます。たとえば、パケットデータは収集されませんが、プロトコル、ソースアドレス、宛先アドレス、ポート番号、バイト数、その他のパケットレベルの情報を収集します。

データは、以下の1つ以上のレコード形式で収集されます。

- NetFlow
- sFlow
- IPFIX

1つ以上のコレクター IP アドレスおよびポート番号を使用して Cluster Network Operator (CNO) を設定する場合、Operator は各ノードで Open vSwitch (OVS) を設定し、ネットワークフローレコードを各コレクターに送信します。

Operator を、複数のネットワークフローコレクターにレコードを送信するように設定できます。たとえば、レコードを NetFlow コレクターに送信し、レコードを sFlow コレクターに送信することもできます。

OVS がデータをコレクターに送信すると、それぞれのタイプのコレクターは同一レコードを受け取ります。たとえば、2つの NetFlow コレクターを設定すると、ノード上の OVS は同じレコードを2つのコレクターに送信します。また、2つの sFlow コレクターを設定した場合には、2つの sFlow コレクターが同じレコードを受け取ります。ただし、各コレクタータイプには固有のレコード形式があります。

ネットワークフローデータを収集し、レコードをコレクターに送信すると、パフォーマンスに影響があります。ノードは低速でパケットを処理します。パフォーマンスへの影響が大きすぎる場合は、コレクターの宛先を削除し、ネットワークフローデータの収集を無効にしてパフォーマンスを回復できます。



注記

ネットワークフローコレクターを有効にすると、クラスターネットワークの全体的なパフォーマンスに影響を与える可能性があります。

16.16.1. ネットワークフローを追跡するためのネットワークオブジェクト設定

Cluster Network Operator (CNO) でネットワークフローコレクターを設定するフィールドを以下の表に示します。

表16.8 ネットワークフローの設定

フィールド	型	説明
<code>metadata.name</code>	<code>string</code>	CNO オブジェクトの名前。この名前は常に cluster です。

フィールド	型	説明
<code>spec.exportNetworkFlows</code>	<code>object</code>	1つ以上の <code>netFlow</code> 、 <code>sFlow</code> 、または <code>ipfix</code> 。
<code>spec.exportNetworkFlows.netFlow.collectors</code>	<code>array</code>	最大 10 コレクターの IP アドレスとネットワークポートのペアの一覧。
<code>spec.exportNetworkFlows.sFlow.collectors</code>	<code>array</code>	最大 10 コレクターの IP アドレスとネットワークポートのペアの一覧。
<code>spec.exportNetworkFlows.ipfix.collectors</code>	<code>array</code>	最大 10 コレクターの IP アドレスとネットワークポートのペアの一覧。

以下のマニフェストを CNO に適用した後に、Operator は、**192.168.1.99:2056** でリッスンする NetFlow コレクターにネットワークフローレコードを送信するようにクラスター内の各ノードで Open vSwitch (OVS) を設定します。

ネットワークフローを追跡するための設定例

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

16.16.2. ネットワークフローコレクターの宛先の追加

クラスター管理者として、Cluster Network Operator (CNO) を設定して、Pod ネットワークについてのネットワークフローメタデータのネットワークフローコレクターへの送信を停止することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。
- ネットワークフローコレクターがあり、リッスンする IP アドレスとポートを把握している。

手順

1. ネットワークフローコレクターのタイプおよびコレクターの IP アドレスとポート情報を指定するパッチファイルを作成します。

```
spec:
  exportNetworkFlows:
    netFlow:
      collectors:
        - 192.168.1.99:2056
```

2. ネットワークフローコレクターで CNO を設定します。

```
$ oc patch network.operator cluster --type merge -p "$(cat <file_name>.yaml)"
```

出力例

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

検証

検証は通常必須ではありません。以下のコマンドを実行して、各ノードの Open vSwitch (OVS) がネットワークフローレコードを1つ以上のコレクターに送信するように設定されていることを確認できます。

1. Operator 設定を表示して、**exportNetworkFlows** フィールドが設定されていることを確認します。

```
$ oc get network.operator cluster -o jsonpath="{.spec.exportNetworkFlows}"
```

出力例

```
{"netFlow":{"collectors":["192.168.1.99:2056"]}}
```

2. 各ノードから OVS のネットワークフロー設定を表示します。

```
$ for pod in $(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node -o
jsonpath='{range@.items[*]}{.metadata.name}{ "\n"}{end}');
do ;
echo;
echo $pod;
oc -n openshift-ovn-kubernetes exec -c ovnkube-node $pod \
-- bash -c 'for type in ipfix sflow netflow ; do ovs-vsctl find $type ; done';
done
```

出力例

```
ovnkube-node-xrn4p
__uuid          : a4d2aaca-5023-4f3d-9400-7275f92611f9
active_timeout  : 60
add_id_to_interface : false
engine_id       : []
engine_type     : []
external_ids    : {}
targets        : ["192.168.1.99:2056"]

ovnkube-node-z4vq9
__uuid          : 61d02fdb-9228-4993-8ff5-b27f01a29bd6
```



```

active_timeout    : 60
add_id_to_interface : false
engine_id        : []
engine_type      : []
external_ids     : {}
targets          : ["192.168.1.99:2056"]-
...

```

16.16.3. ネットワークフローコレクターのすべての宛先の削除

クラスター管理者として、Cluster Network Operator (CNO) を設定して、ネットワークフローメタデータのネットワークフローコレクターへの送信を停止することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。

手順

1. すべてのネットワークフローコレクターを削除します。

```

$ oc patch network.operator cluster --type='json' \
  -p=[{"op": "remove", "path": "/spec/exportNetworkFlows"}]

```

出力例

```

network.operator.openshift.io/cluster patched

```

16.16.4. 関連情報

- [Network \[operator.openshift.io/v1\]](https://network.operator.openshift.io/v1)

16.17. ハイブリッドネットワークの設定

クラスター管理者は、OVN-Kubernetes Container Network Interface (CNI) クラスターネットワークプロバイダーを、Linux および Windows ノードがそれぞれ Linux および Windows ワークロードをできるように設定できます。

16.17.1. OVN-Kubernetes を使用したハイブリッドネットワークの設定

OVN-Kubernetes でハイブリッドネットワークを使用するようにクラスターを設定できます。これにより、異なるノードのネットワーク設定をサポートするハイブリッドクラスターが可能になります。たとえば、これはクラスター内の Linux ノードと Windows ノードの両方を実行するために必要です。



重要

クラスターのインストール時に、OVN-Kubernetes を使用してハイブリッドネットワークを設定する必要があります。インストールプロセス後に、ハイブリッドネットワークに切り替えることはできません。

前提条件

- **install-config.yaml** ファイルで **networking.networkType** パラメーターの **OVNKubernetes** を定義していること。詳細は、選択したクラウドプロバイダーでの OpenShift Container Platform ネットワークのカスタマイズの設定についてのインストールドキュメントを参照してください。

手順

1. インストールプログラムが含まれるディレクトリーに切り替え、マニフェストを作成します。

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

ここでは、以下ようになります。

<installation_directory>

クラスターの **install-config.yaml** ファイルが含まれるディレクトリーの名前を指定します。

2. **cluster-network-03-config.yml** という名前の、高度なネットワーク設定用のスタブマニフェストファイルを **<installation_directory>/manifests/** ディレクトリーに作成します。

```
$ cat <<EOF > <installation_directory>/manifests/cluster-network-03-config.yml
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  EOF
```

ここでは、以下ようになります。

<installation_directory>

クラスターの **manifests/** ディレクトリーが含まれるディレクトリー名を指定します。

3. **cluster-network-03-config.yml** ファイルをエディターで開き、以下の例のようにハイブリッドネットワークで OVN-Kubernetes を設定します。

ハイブリッドネットワーク設定の指定

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  defaultNetwork:
    ovnKubernetesConfig:
      hybridOverlayConfig:
        hybridClusterNetwork: ①
        - cidr: 10.132.0.0/14
          hostPrefix: 23
        hybridOverlayVXLANPort: 9898 ②
```

- 1 追加のオーバーレイネットワーク上のノードに使用される CIDR 設定を指定します。 **hybridClusterNetwork** CIDR は **clusterNetwork** CIDR と重複できません。
- 2 追加のオーバーレイネットワークのカスタム VXLAN ポートを指定します。これは、vSphere にインストールされたクラスターで Windows ノードを実行するために必要であり、その他のクラウドプロバイダー用に設定することはできません。カスタムポートには、デフォルトの **4789** ポートを除くいずれかのオープンポートを使用できます。この要件についての詳細は、Microsoft ドキュメントの [Pod-to-pod connectivity between hosts is broken](#) を参照してください。



注記

Windows Server Long-Term Servicing Channel (LTSC): Windows Server 2019 は、カスタムの VXLAN ポートの選択をサポートしないため、カスタムの **hybridOverlayVXLANPort** 値を持つクラスターではサポートされません。

4. **cluster-network-03-config.yml** ファイルを保存し、テキストエディターを終了します。
5. オプション: **manifests/cluster-network-03-config.yml** ファイルをバックアップします。インストールプログラムは、クラスターの作成時に **manifests/** ディレクトリーを削除します。

追加のインストール設定を完了してから、クラスターを作成します。インストールプロセスが終了すると、ハイブリッドネットワークが有効になります。

16.17.2. 関連情報

- [Windows コンテナワークロードについて](#)
- [Windows コンテナワークロードの有効化](#)
- [ネットワークのカスタマイズによる AWS へのクラスターのインストール](#)
- [ネットワークのカスタマイズによる Azure へのクラスターのインストール](#)

第17章 ルートの作成

17.1. ルート設定

17.1.1. HTTP ベースのルートの作成

ルートを使用すると、公開された URL でアプリケーションをホストできます。これは、アプリケーションのネットワークセキュリティ設定に応じて、セキュリティ保護または保護なしを指定できます。HTTP ベースのルートとは、セキュアではないルートで、基本的な HTTP ルーティングプロトコルを使用してセキュリティ保護されていないアプリケーションポートでサービスを公開します。

以下の手順では、**hello-openshift** アプリケーションを例に、Web アプリケーションへのシンプルな HTTP ベースのルートを作成する方法を説明します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 管理者としてログインしている。
- あるポートを公開する Web アプリケーションと、そのポートでトラフィックをリッスンする TCP エンドポイントがあります。

手順

1. 次のコマンドを実行して、**hello-openshift** というプロジェクトを作成します。

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

2. 以下のコマンドを実行してプロジェクトに Pod を作成します。

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

3. 以下のコマンドを実行して、**hello-openshift** というサービスを作成します。

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

4. 次のコマンドを実行して、**hello-openshift** アプリケーションに対して、セキュアではないルートを作成します。

```
$ oc expose svc hello-openshift
```

結果として生成される **Route** リソースを検査すると、以下のようになります。

上記で作成されたセキュアでないルートの YAML 定義

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: hello-openshift
spec:
```

```
host: hello-openshift-hello-openshift.<Ingress_Domain> ❶
port:
  targetPort: 8080 ❷
to:
  kind: Service
  name: hello-openshift
```

- ❶ **<Ingress_Domain>** はデフォルトの Ingress ドメイン名です。 **ingresses.config/cluster** オブジェクトはインストール中に作成され、変更できません。別のドメインを指定する場合は、 **appsDomain** オプションを使用して別のクラスタードメインを指定できます。
- ❷ **targetPort** は、このルートが指すサービスによって選択される Pod のターゲットポートです。



注記

デフォルトの ingress ドメインを表示するには、以下のコマンドを実行します。

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

17.1.2. ルートのタイムアウトの設定

Service Level Availability (SLA) で必要とされる、低タイムアウトが必要なサービスや、バックエンドでの処理速度が遅いケースで高タイムアウトが必要なサービスがある場合は、既存のルートに対してデフォルトのタイムアウトを設定することができます。

前提条件

- 実行中のクラスターでデプロイ済みの Ingress コントローラーが必要になります。

手順

1. **oc annotate** コマンドを使用して、ルートにタイムアウトを追加します。

```
$ oc annotate route <route_name> \
  --overwrite haproxy.router.openshift.io/timeout=<timeout><time_unit> ❶
```

- ❶ サポートされる時間単位は、マイクロ秒 (us)、ミリ秒 (ms)、秒 (s)、分 (m)、時間 (h)、または日 (d) です。

以下の例では、2 秒のタイムアウトを **myroute** という名前のルートに設定します。

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

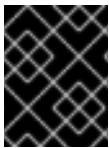
17.1.3. HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS) ポリシーは、HTTPS トラフィックのみがルートホストで許可されるブラウザクライアントに通知するセキュリティの拡張機能です。また、HSTS は、HTTP リダイレクトを使用せずに HTTPS トラフィックにシグナルを送ることで Web トラフィックを最適化します。HSTS は Web サイトとの対話を迅速化するのに便利です。

HSTS ポリシーが適用されると、HSTS はサイトから Strict Transport Security ヘッダーを HTTP および HTTPS 応答に追加します。HTTP を HTTPS にリダイレクトするルートで **insecureEdgeTerminationPolicy** 値を使用できます。HSTS を強制している場合は、要求の送信前にクライアントがすべての要求を HTTP URL から HTTPS に変更するため、リダイレクトの必要がなくなります。

クラスター管理者は、以下を実行するために HSTS を設定できます。

- ルートごとに HSTS を有効にします。
- ルートごとに HSTS を無効にします。
- ドメインごとに HSTS を適用するか、ドメインと組み合わせた namespace ラベルを使用します。



重要

HSTS はセキュアなルート (edge-termination または re-encrypt) でのみ機能します。この設定は、HTTP またはパススルールートには適していません。

17.1.3.1. ルートごとの HTTP Strict Transport Security の有効化

HTTP 厳密なトランスポートセキュリティ (HSTS) は HAProxy テンプレートに実装され、**haproxy.router.openshift.io/hsts_header** アノテーションを持つ edge および re-encrypt ルートに適用されます。

前提条件

- プロジェクトの管理者権限があるユーザーで、クラスターにログインしている。
- **oc** CLI をインストールしていること。

手順

- ルートで HSTS を有効にするには、**haproxy.router.openshift.io/hsts_header** 値を edge-terminated または re-encrypt ルートに追加します。これを実行するには、**oc annotate** ツールを使用してこれを実行できます。

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

- 1** この例では、最長期間は **31536000** ミリ秒 (約 8 時間および半分) に設定されます。



注記

この例では、等号 (=) が引用符で囲まれています。これは、annotate コマンドを正しく実行するために必要です。

アノテーションで設定されたルートの例

```
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 必須。**max-age** は、HSTS ポリシーが有効な期間 (秒単位) を測定します。0 に設定すると、これはポリシーを無効にします。
- 2 オプション:**includeSubDomains** は、クライアントに対し、ホストのすべてのサブドメインにホストと同じ HSTS ポリシーを持つ必要があることを指示します。
- 3 オプション:**max-age** が 0 より大きい場合、**preload** を **haproxy.router.openshift.io/hsts_header** に追加し、外部サービスがこのサイトをそれぞれの HSTS プリロード一覧に含めることができます。たとえば、Google などのサイトは **preload** が設定されているサイトの一覧を作成します。ブラウザはこれらの一覧を使用し、サイトと対話する前でも HTTPS 経由で通信できるサイトを判別できます。**preload** を設定していない場合、ブラウザはヘッダーを取得するために、HTTPS を介してサイトと少なくとも 1 回対話している必要があります。

17.1.3.2. ルートごとの HTTP Strict Transport Security の無効化

ルートごとに HSTS (HTTP Strict Transport Security) を無効にするには、ルートアノテーションの **max-age** の値を 0 に設定します。

前提条件

- プロジェクトの管理者権限があるユーザーで、クラスターにログインしている。
- **oc** CLI をインストールしていること。

手順

- HSTS を無効にするには、以下のコマンドを入力してルートアノテーションの **max-age** の値を 0 に設定します。

```

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

```

ヒント

または、以下の YAML を適用して ConfigMap を作成できます。

ルートごとに HSTS を無効にする例

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

- namespace のすべてのルートで HSTS を無効にするには、following コマンドを入力します。

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

検証

- すべてのルートのアノテーションをクエリーするには、以下のコマンドを入力します。

```
$ 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}}\n{{end}}\n{{end}}'
```

出力例

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

17.1.3.3. ドメインごとに HTTP Strict Transport Security の強制

安全なルートのドメインごとに HTTP Strict Transport Security (HSTS) を適用するには、**requiredHSTSPolicies** レコードを Ingress 仕様に追加して、HSTS ポリシーの設定を取得します。

requiredHSTSPolicy を設定して HSTS を適用する場合は、新規に作成されたルートは準拠された HSTS ポリシーアノテーションで設定する必要があります。



注記

準拠しない HSTS ルートを持つアップグレードされたクラスターを処理するには、ソースでマニフェストを更新し、更新を適用できます。



注記

oc expose route コマンドまたは **oc create route** コマンドを使用して、HSTS を強制するドメインにルートを追加することはできません。このコマンドの API はアノテーションを受け入れないためです。



重要

HSTS がすべてのルートに対してグローバルに要求されている場合でも、セキュアではないルートや非 TLS ルートに適用することはできません。

前提条件

- プロジェクトの管理者権限があるユーザーで、クラスターにログインしている。
- **oc** CLI をインストールしていること。

手順

1. Ingress 設定ファイルを編集します。

```
$ oc edit ingresses.config.openshift.io/cluster
```

HSTS ポリシーの例

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  domain: 'hello-openshift-default.apps.username.devcluster.openshift.com'
  requiredHSTSPolicies: 1
  - domainPatterns: 2
    - '*hello-openshift-default.apps.username.devcluster.openshift.com'
    - '*hello-openshift-default2.apps.username.devcluster.openshift.com'
  namespaceSelector: 3
    matchLabels:
      myPolicy: strict
  maxAge: 4
    smallestMaxAge: 1
    largestMaxAge: 31536000
  preloadPolicy: RequirePreload 5
  includeSubDomainsPolicy: RequireIncludeSubDomains 6
  - domainPatterns: 7
    - 'abc.example.com'
    - '*xyz.example.com'
  namespaceSelector:
    matchLabels: {}
  maxAge: {}
  preloadPolicy: NoOpinion
  includeSubDomainsPolicy: RequireNoIncludeSubDomains
```

- 1** 必須。**requiredHSTSPolicies** は順番に検証され、最初に一致する **domainPatterns** が適用されます。
- 2** **7** 必須。1つ以上の **domainPatterns** ホスト名を指定する必要があります。任意の数のドメインを一覧表示できます。さまざまな **domainPatterns** について、Enforcing オプションの複数のセクションを含めることができます。
- 3** オプション:**namespaceSelector** を含める場合、ルートを配置するプロジェクトのラベルと一致する必要があります。これにより、ルートに設定された HSTS ポリシーを適用する必要があります。**domainPatterns** ではなく **namespaceSelector** のみに一致するルートは検証されません。
- 4** 必須。**max-age** は、HSTS ポリシーが有効な期間 (秒単位) を測定します。このポリシー

- **largestMaxAge** の値は **0** から **2147483647** の範囲内で指定する必要があります。これを指定しないと、上限が強制されないことを意味します。
- **smallestMaxAge** の値は **0** から **2147483647** の範囲内で指定する必要があります。トラブルシューティングのために HSTS を無効にするには、**0** を入力します。HSTS を無効にする必要がない場合は **1** を入力します。これを指定しないと、下限が強制されません。

5 オプション:**haproxy.router.openshift.io/hsts_header** に **preload** を含めることで、外部サービスがこのサイトをそれぞれの HSTS プリロード一覧に含めることができます。ブラウザはこれらの一覧を使用し、サイトと対話する前でも HTTPS 経由で通信できるサイトを判別できます。**preload** 設定がない場合、ブラウザは少なくともサイトと通信してヘッダーを取得する必要があります。**preload** は、以下のいずれかで設定できます。

- **RequirePreload**: **preload** は **RequiredHSTSPolicy** で必要になります。
- **RequireNoPreload**: **preload** は **RequiredHSTSPolicy** によって禁止されます。
- **NoOpinion**: **preload** は **RequiredHSTSPolicy** に重要ではありません。

6 オプション:**includeSubDomainsPolicy** は、以下のいずれかで設定できます。

- **RequireIncludeSubDomains**: **includeSubDomains** は **RequiredHSTSPolicy** が必要です。
- **RequireNoIncludeSubDomains**: **includeSubDomains** は **RequiredHSTSPolicy** によって禁止されています。
- **NoOpinion**: **includeSubDomains** は **RequiredHSTSPolicy** に重要ではありません。

2. **oc annotate command** を入力して、HSTS をクラスターのすべてのルートまたは特定の namespace に適用することができます。

- HSTS をクラスターのすべてのルートに適用するには、**oc annotate command** を実行します。以下に例を示します。

```
$ oc annotate route --all --all-namespaces --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

- 特定の namespace のすべてのルートに HSTS を適用するには、**oc annotate command** を実行します。以下に例を示します。

```
$ oc annotate route --all -n my-namespace --overwrite=true
"haproxy.router.openshift.io/hsts_header"="max-age=31536000"
```

検証

設定した HSTS ポリシーを確認できます。以下に例を示します。

- 必要な HSTS ポリシーの **maxAge** セットを確認するには、以下のコマンドを入力します。

```
$ oc get clusteroperator/ingress -n openshift-ingress-operator -o jsonpath='{range .spec.requiredHSTSPolicies[*]}{.spec.requiredHSTSPolicies.maxAgePolicy.largestMaxAge}{"\n"}{end}'
```

- すべてのルートで HSTS アノテーションを確認するには、以下のコマンドを入力します。

```
$ 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 }}'
```

出力例

```
Name: <_routename_> HSTS: max-age=31536000;preload;includeSubDomains
```

17.1.4. スループット関連の問題のトラブルシューティング

OpenShift Container Platform でデプロイされるアプリケーションでは、特定のサービス間で非常に長い待ち時間が発生するなど、ネットワークのスループットの問題が生じることがあります。

Pod のログが問題の原因を指摘しない場合は、以下の方法を使用してパフォーマンスの問題を分析します。

- ping または `tcpdump` などのパケットアナライザーを使用して Pod とそのノード間のトラフィックを分析します。
たとえば、問題を生じさせる動作を再現している間に各 Pod で `tcpdump` ツールを実行します。両サイトでキャプチャーしたデータを確認し、送信および受信タイムスタンプを比較して Pod への/からのトラフィックの待ち時間を分析します。待ち時間は、ノードのインターフェイスが他の Pod やストレージデバイス、またはデータプレーンからのトラフィックでオーバーロードする場合に OpenShift Container Platform で発生する可能性があります。

```
$ tcpdump -s 0 -i any -w /tmp/dump.pcap host <podip 1> && host <podip 2> 1
```

- 1** `podip` は Pod の IP アドレスです。 `oc get pod <pod_name> -o wide` コマンドを実行して Pod の IP アドレスを取得します。

`tcpdump` は 2 つの Pod 間のすべてのトラフィックが含まれる `/tmp/dump.pcap` のファイルを生成します。理想的には、ファイルサイズを最小限に抑えるために問題を再現するすぐ前と問題を再現したすぐ後にアナライザーを実行することが良いでしょう。以下のようにノード間でパケットアナライザーを実行することもできます (式から SDN を排除する)。

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

- ストリーミングのスループットおよび UDP スループットを測定するために `iperf` などの帯域幅測定ツールを使用します。ボトルネックの特定を試行するには、最初に Pod から、次にノードからツールを実行します。
 - `iperf` のインストールおよび使用についての詳細は、こちらの [Red Hat ソリューション](#) を参照してください。

17.1.5. Cookie に使用によるルートのステートフル性の維持

OpenShift Container Platform は、すべてのトラフィックを同じエンドポイントにヒットさせることによりステートフルなアプリケーションのトラフィックを可能にするスティッキーセッションを提供します。ただし、エンドポイント Pod が再起動、スケールアップ、または設定の変更などによって終了する場合、このステートフル性はなくなります。

OpenShift Container Platform は Cookie を使用してセッションの永続化を設定できます。Ingress コントローラーはユーザー要求を処理するエンドポイントを選択し、そのセッションの Cookie を作成します。Cookie は要求の応答として戻され、ユーザーは Cookie をセッションの次の要求と共に送り返します。Cookie は Ingress コントローラーに対し、セッションを処理しているエンドポイントを示し、クライアント要求が Cookie を使用して同じ Pod にルーティングされるようにします。



注記

cookie は、HTTP トラフィックを表示できないので、パススルールートで設定できません。代わりに、ソース IP アドレスをベースに数が計算され、バックエンドを判断します。

バックエンドが変わると、トラフィックが間違ったサーバーに転送されてしまい、スティッキーではなくなります。ソース IP を非表示にするロードバランサーを使用している場合は、すべての接続に同じ番号が設定され、トラフィックは同じ Pod に送られます。

17.1.5.1. Cookie を使用したルートのアノテーション

ルート用に自動生成されるデフォルト名を上書きするために Cookie 名を設定できます。これにより、ルートトラフィックを受信するアプリケーションが Cookie 名を認識できるようになります。Cookie を削除すると、次の要求でエンドポイントの再選択が強制的に実行される可能性があります。そのためサーバーがオーバーロードしている場合には、クライアントからの要求を取り除き、それらの再分配を試行します。

手順

1. 指定される cookie 名でルートにアノテーションを付けます。

```
$ oc annotate route <route_name> router.openshift.io/cookie_name="<cookie_name>"
```

ここでは、以下ようになります。

<route_name>

Pod の名前を指定します。

<cookie_name>

cookie の名前を指定します。

たとえば、ルート **my_route** に cookie 名 **my_cookie** でアノテーションを付けるには、以下を実行します。

```
$ oc annotate route my_route router.openshift.io/cookie_name="my_cookie"
```

2. 変数でルートのホスト名を取得します。

```
$ ROUTE_NAME=$(oc get route <route_name> -o jsonpath='{.spec.host}')
```

ここでは、以下ようになります。

<route_name>

Pod の名前を指定します。

3. cookie を保存してからルートにアクセスします。

```
$ curl $ROUTE_NAME -k -c /tmp/cookie_jar
```

ルートに接続する際に、直前のコマンドによって保存される cookie を使用します。

```
$ curl $ROUTE_NAME -k -b /tmp/cookie_jar
```

17.1.6. パスベースのルート

パスベースのルートは、URL に対して比較できるパスコンポーネントを指定します。この場合、ルートのトラフィックは HTTP ベースである必要があります。そのため、それぞれが異なるパスを持つ同じホスト名を使用して複数のルートを提供できます。ルーターは、最も具体的なパスの順に基づいてルートと一致する必要があります。ただし、これはルーターの実装によって異なります。

以下の表は、ルートのサンプルおよびそれらのアクセシビリティを示しています。

表17.1 ルートの可用性

ルート	比較対象	アクセス可能
www.example.com/test	www.example.com/test	はい
	www.example.com	いいえ
www.example.com/test および www.example.com	www.example.com/test	はい
	www.example.com	はい
www.example.com	www.example.com/test	Yes (ルートではなく、ホストで一致)
	www.example.com	はい

パスが1つでセキュリティ保護されていないルート

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

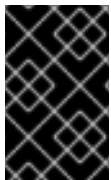
❶ パスは、パスベースのルートに唯一追加される属性です。

**注記**

ルーターは TLS を終了させず、要求のコンテンツを読み込みできないので、パースベースのルーティングは、パススルー TLS を使用する場合には利用できません。

17.1.7. ルート固有のアノテーション

Ingress コントローラーは、公開するすべてのルートのデフォルトオプションを設定できます。個別のルートは、アノテーションに個別の設定を指定して、デフォルトの一部を上書きできます。Red Hat では、ルートアノテーションの Operator 管理ルートへの追加はサポートしません。

**重要**

複数のソース IP またはサブネットのホワイトリストを作成するには、スペースで区切られたリストを使用します。他の区切りタイプを使用すると、一覧が警告やエラーメッセージなしに無視されます。

表17.2 ルートアノテーション

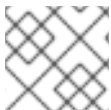
変数	説明	デフォルトで使用される環境変数
<code>haproxy.router.openshift.io/balancer</code>	ロードバランシングアルゴリズムを設定します。使用できるオプションは、 random 、 source 、 roundrobin 、および leastconn です。デフォルト値は random です。	パススルールートの場合、 ROUTER_TCP_BALANCE_SCHEME です。それ以外の場合は ROUTER_LOAD_BALANCE_ALGORITHM を使用します。
<code>haproxy.router.openshift.io/disable_cookies</code>	関連の接続を追跡する cookie の使用を無効にします。'true' または 'TRUE' に設定する場合は、分散アルゴリズムを使用して、受信する HTTP 要求ごとに、どのバックエンドが接続を提供するかを選択します。	
<code>router.openshift.io/cookie_name</code>	このルートに使用するオプションの cookie を指定します。名前は、大文字、小文字、数字、"_" または "-" を任意に組み合わせて指定する必要があります。デフォルトは、ルートのハッシュ化された内部キー名です。	

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/pod-concurrent-connections	<p>ルーターからバックアップされる Pod に対して許容される接続最大数を設定します。</p> <p>注意: Pod が複数ある場合には、それぞれに対応する接続数を設定できます。複数のルーターがある場合は、それらのルーター間で調整は行われず、それぞれがこれに複数回接続する可能性があります。設定されていない場合または 0 に設定されている場合には制限はありません。</p>	
haproxy.router.openshift.io/rate-limit-connections	<p>'true' または 'TRUE' を設定すると、ルートごとに特定のバックエンドの stick-tables で実装されるレート制限機能が有効になります。</p> <p>注記: このアノテーションを使用すると、DDoS (Distributed Denial-of-service) 攻撃に対する基本的な保護機能が提供されます。</p>	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	<p>同じソース IP アドレスで行われる同時 TCP 接続の数を制限します。数値を受け入れます。</p> <p>注記: このアノテーションを使用すると、DDoS (Distributed Denial-of-service) 攻撃に対する基本的な保護機能が提供されます。</p>	
haproxy.router.openshift.io/rate-limit-connections.rate-http	<p>同じソース IP アドレスを持つクライアントが HTTP 要求を実行できるレートを制限します。数値を受け入れます。</p> <p>注記: このアノテーションを使用すると、DDoS (Distributed Denial-of-service) 攻撃に対する基本的な保護機能が提供されます。</p>	

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	<p>同じソース IP アドレスを持つクライアントが TCP 接続を確立するレートを制限します。数値を受け入れます。</p> <p>注記: このアノテーションを使用すると、DDoS (Distributed Denial-of-service) 攻撃に対する基本的な保護機能が提供されません。</p>	
haproxy.router.openshift.io/ti meout	ルートのサーバー側のタイムアウトを設定します。(TimeUnits)	ROUTER_DEFAULT_SERVER_TIMEOUT
haproxy.router.openshift.io/ti meout-tunnel	<p>このタイムアウトは、クリアテキスト、エッジ、再暗号化、またはパススルーのルートを介した Web Socket などトンネル接続に適用されます。cleartext、edge、または reencrypt のルートタイプでは、このアノテーションは、タイムアウト値がすでに存在するタイムアウトトンネルとして適用されます。パススルーのルートタイプでは、アノテーションは既存のタイムアウト値の設定よりも優先されます。</p>	ROUTER_DEFAULT_TUNNEL_TIMEOUT
ingresses.config/cluster ingress.operator.openshift.io/hard-stop-after	<p>設定できるのは、Ingress Controller または ingress config です。このアノテーションでは、ルーターを再デプロイし、HA プロキシが haproxyhard-stop-after グローバルオプションを実行するように設定します。このオプションは、クリーンなソフトウェア実行で最大許容される時間を定義します。</p>	ROUTER_HARD_STOP_AFTER
router.openshift.io/haproxy.health.check.interval	バックエンドのヘルスチェックの間隔を設定します。(TimeUnits)	ROUTER_BACKEND_CHECK_INTERVAL

変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/iptables_whitelist	<p>ルートのホワイトリストを設定します。ホワイトリストは、承認したソースアドレスの IP アドレスおよび CIDR 範囲の一覧をスペース区切りにします。ホワイトリストに含まれていない IP アドレスからの要求は破棄されます。</p> <p>ホワイトリストの許可される IP アドレスおよび CIDR 範囲の最大数は 61 です。</p>	
haproxy.router.openshift.io/https_header	<p>edge terminated または re-encrypt ルートの Strick-Transport-Security ヘッダーを設定します。</p>	
haproxy.router.openshift.io/log-send-hostname	<p>Syslog ヘッダーの hostname フィールドを設定します。システムのホスト名を使用します。サイドカーや Syslog ファシリティーなどの Ingress API ロギングメソッドがルーターに対して有効になっている場合、log-send-hostname はデフォルトで有効になります。</p>	
haproxy.router.openshift.io/rewrite-target	<p>バックエンドの要求の書き換えパスを設定します。</p>	
router.openshift.io/cookie-same-site	<p>Cookie を制限するために値を設定します。値は以下のようになります。</p> <p>Lax: Cookie はアクセスしたサイトとサードパーティーのサイト間で転送されます。</p> <p>Strict: Cookie はアクセスしたサイトに制限されます。</p> <p>None: Cookie はアクセスしたサイトに制限されます。</p> <p>この値は、re-encrypt および edge ルートにのみ適用されません。詳細は、SameSite cookie のドキュメント を参照してください。</p>	

変数	説明	デフォルトで 사용되는環境変数
haproxy.router.openshift.io/set-forwarded-headers	<p>ルートごとに Forwarded および X-Forwarded-For HTTP ヘッダーを処理するポリシーを設定します。値は以下ようになります。</p> <p>append: ヘッダーを追加し、既存のヘッダーを保持します。これはデフォルト値です。</p> <p>Replace: ヘッダーを設定し、既存のヘッダーを削除します。</p> <p>never: ヘッダーを設定しませんが、既存のヘッダーを保持します。</p> <p>if-none: ヘッダーがまだ設定されていない場合にこれを設定します。</p>	ROUTER_SET_FORWARDER_HEADERS



注記

環境変数を編集することはできません。

ルータータイムアウト変数

TimeUnits は数字、その後に単位を指定して表現します。 **us** *(マイクロ秒)、 **ms** (ミリ秒、デフォルト)、 **s** (秒)、 **m** (分)、 **h** *(時間)、 **d** (日)

正規表現: `[1-9][0-9]*(us|ms|s|m|h|d)`

変数	デフォルト	説明
ROUTER_BACKEND_CHECK_INTERVAL	5000ms	バックエンドでの後続の liveness チェックの時間の長さ。
ROUTER_CLIENT_FIN_TIMEOUT	1s	クライアントがルートに接続する場合の TCP FIN タイムアウトの期間を制御します。接続切断のために送信された FIN が指定の時間内に応答されない場合は、HAProxy が接続を切断します。小さい値を設定し、ルーターでリソースをあまり使用していない場合には、リスクはありません。
ROUTER_DEFAULT_CLIENT_TIMEOUT	30s	クライアントがデータを確認するか、送信するための時間の長さ。

変数	デフォルト	説明
ROUTER_DEFAULT_CONNECT_TIMEOUT	5s	最大接続時間。
ROUTER_DEFAULT_SERVER_FIN_TIMEOUT	1s	ルーターからルートを保ッキングする Pod の TCP FIN タイムアウトを制御します。
ROUTER_DEFAULT_SERVER_TIMEOUT	30s	サーバーがデータを確認するか、送信するための時間の長さ。
ROUTER_DEFAULT_TUNNEL_TIMEOUT	1h	TCP または WebSocket 接続が開放された状態で保つ時間数。このタイムアウト期間は、HAProxy が再読み込みされるたびにリセットされます。
ROUTER_SLOWLORIS_HTTP_KEEPAKALIVE	300s	<p>新しい HTTP 要求が表示されるまで待機する最大時間を設定します。この値が低すぎる場合には、ブラウザおよびアプリケーションの keepalive 値が低くなりすぎて、問題が発生する可能性があります。</p> <p>有効なタイムアウト値には、想定した個別のタイムアウトではなく、特定の変数を合計した値に指定することができます。たとえば、ROUTER_SLOWLORIS_HTTP_KEEPAKALIVE は、timeout http-keep-alive を調整します。HAProxy はデフォルトで 300s に設定されていますが、HAProxy は tcp-request inspect-delay も待機します。これは 5s に設定されています。この場合、全体的なタイムアウトは 300s に 5s を加えたこととなります。</p>
ROUTER_SLOWLORIS_TIMEOUT	10s	HTTP 要求の伝送にかかる時間。
RELOAD_INTERVAL	5s	ルーターがリロードし、新規の変更を受け入れる最小の頻度を許可します。
ROUTER_METRICS_HAPROXY_TIMEOUT	5s	HAProxy メトリクスの収集タイムアウト。

ルート設定のカスタムタイムアウト

```

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

```

```

annotations:
  haproxy.router.openshift.io/timeout: 5500ms ❶
...

```

- ❶ HAProxy 対応の単位 (**us**、**ms**、**s**、**m**、**h**、**d**) で新規のタイムアウトを指定します。単位が指定されていない場合は、**ms** がデフォルトになります。



注記

パススルールート of the server side of the timeout value is set too low, WebSocket connections may frequently timeout on that route.

特定の IP アドレスを 1 つだけ許可するルート

```

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

```

複数の IP アドレスを許可するルート

```

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

```

IP アドレスの CIDR ネットワークを許可するルート

```

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

```

IP アドレスと IP アドレスの CIDR ネットワークの両方を許可するルート

```

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

```

書き換えターゲットを指定するルート

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  annotations:
    haproxy.router.openshift.io/rewrite-target: / ❶
...

```

- ❶ バックエンドの要求の書き換えパスとして / を設定します。

ルートに **haproxy.router.openshift.io/rewrite-target** アノテーションを設定すると、要求をバックエンドアプリケーションに転送する前に Ingress コントローラーがこのルートを使用して HTTP 要求のパスを書き換える必要があることを指定します。 **spec.path** で指定されたパスに一致する要求パスの一部

は、アノテーションで指定された書き換えターゲットに置き換えられます。

以下の表は、**spec.path**、要求パス、および書き換えターゲットの各種の組み合わせについてのパスの書き換え動作の例を示しています。

表17.3 rewrite-target の例:

Route.spec.path	要求パス	書き換えターゲット	転送された要求パス
/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	/	該当なし (要求パスがルートパスに一致しない)
/foo/	/foo/	/	/
/foo/	/foo/bar	/	/bar

17.1.8. ルートの受付ポリシーの設定

管理者およびアプリケーション開発者は、同じドメイン名を持つ複数の namespace でアプリケーションを実行できます。これは、複数のチームが同じホスト名で公開されるマイクロサービスを開発する組織を対象としています。



警告

複数の namespace での要求の許可は、namespace 間の信頼のあるクラスターに対してのみ有効にする必要があります。有効にしないと、悪意のあるユーザーがホスト名を乗っ取る可能性があります。このため、デフォルトの受付ポリシーは複数の namespace 間でのホスト名の要求を許可しません。

前提条件

- クラスター管理者の権限。

手順

- 以下のコマンドを使用して、**ingresscontroller** リソース変数の **.spec.routeAdmission** フィールドを編集します。

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

イメージコントローラー設定例

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

ヒント

または、以下の YAML を適用してルートの受付ポリシーを設定できます。

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

17.1.9. Ingress オブジェクトを使用したルートの作成

一部のエコシステムコンポーネントには Ingress リソースとの統合機能がありますが、ルートリソースとは統合しません。これに対応するために、OpenShift Container Platform は Ingress オブジェクトの作成時に管理されるルートオブジェクトを自動的に作成します。これらのルートオブジェクトは、対応する Ingress オブジェクトが削除されると削除されます。

手順

1. OpenShift Container Platform コンソールで Ingress オブジェクトを定義するか、または **oc create** コマンドを実行します。

Ingress の YAML 定義

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: frontend
  annotations:
    route.openshift.io/termination: "reencrypt" 1
spec:
```

```

rules:
- host: www.example.com
  http:
    paths:
    - backend:
        service:
          name: frontend
          port:
            number: 443
        path: /
        pathType: Prefix
tls:
- hosts:
  - www.example.com
  secretName: example-com-tls-certificate

```

1 **route.openshift.io/termination** アノテーションは、**Route** の **spec.tls.termination** フィールドを設定するために使用できます。**Ingress** にはこのフィールドがありません。許可される値は **edge**、**passthrough**、および **reencrypt** です。その他のすべての値は警告なしに無視されます。アノテーション値が設定されていない場合は、**edge** がデフォルトルートになります。デフォルトのエッジルートを実装するには、TLS 証明書の詳細をテンプレートファイルで定義する必要があります。

- a. **route.openshift.io/termination** アノテーションで **passthrough** の値を指定する場合は、仕様で **path** を `"` に設定し、**pathType** を **ImplementationSpecific** に設定します。

```

spec:
  rules:
  - host: www.example.com
    http:
      paths:
      - path: ""
        pathType: ImplementationSpecific
      backend:
        service:
          name: frontend
          port:
            number: 443

```

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

2. ルートを一覧表示します。

```
$ oc get routes
```

結果には、**frontend-** で始まる名前の自動生成ルートが含まれます。

NAME	HOST/PORT	PATH	SERVICES	PORT	TERMINATION
WILDCARD					
frontend-gnztq	www.example.com		frontend	443	reencrypt/Redirect
					None

このルートを検査すると、以下のようになります。

自動生成されるルートの YAML 定義

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: frontend-gnztq
  ownerReferences:
  - apiVersion: networking.k8s.io/v1
    controller: true
    kind: Ingress
    name: frontend
    uid: 4e6c59cc-704d-4f44-b390-617d879033b6
spec:
  host: www.example.com
  path: /
  port:
    targetPort: https
  tls:
    certificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
    insecureEdgeTerminationPolicy: Redirect
    key: |
      -----BEGIN RSA PRIVATE KEY-----
      [...]
      -----END RSA PRIVATE KEY-----
    termination: reencrypt
  to:
    kind: Service
    name: frontend

```

17.1.10. Ingress オブジェクトを介してデフォルトの証明書を使用してルートを作成する

TLS 設定を指定せずに Ingress オブジェクトを作成すると、OpenShift Container Platform は安全でないルートを生成します。デフォルトの Ingress 証明書を使用してセキュアなエッジ終端ルートを生成する Ingress オブジェクトを作成するには、次のように空の TLS 設定を指定できます。

前提条件

- 公開したいサービスがあります。
- OpenShift CLI (**oc**) にアクセスできる。

手順

1. Ingress オブジェクトの YAML ファイルを作成します。この例では、ファイルの名前は **example-ingress.yaml** です。

Ingress オブジェクトの YAML 定義

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:

```



```

name: frontend
...
spec:
  rules:
    ...
  tls:
    - {} ❶

```

- ❶ この正確な構文を使用して、カスタム証明書を指定せずに TLS を指定します。

2. 次のコマンドを実行して、Ingress オブジェクトを作成します。

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

検証

- 以下のコマンドを実行して、OpenShift Container Platform が Ingress オブジェクトの予期されるルートを作成したことを確認します。

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

出力例

```

apiVersion: v1
items:
- apiVersion: route.openshift.io/v1
  kind: Route
  metadata:
    name: frontend-j9sdd ❶
    ...
  spec:
    ...
    tls: ❷
      insecureEdgeTerminationPolicy: Redirect
      termination: edge ❸
    ...

```

- ❶ ルートの名前には、Ingress オブジェクトの名前とそれに続くランダムな接尾辞が含まれます。
- ❷ デフォルトの証明書を使用するには、ルートで **spec.certificate** を指定しないでください。
- ❸ ルートは、**edge** の終了ポリシーを指定する必要があります。

17.1.11. デュアルスタックネットワーク用の OpenShift Container Platform Ingress コントローラーの設定

OpenShift Container Platform クラスターが IPv4 および IPv6 デュアルスタックネットワーク用に設定されている場合、クラスターは OpenShift Container Platform ルートによって外部からアクセス可能です。

Ingress コントローラーは、IPv4 エンドポイントと IPv6 エンドポイントの両方を持つサービスを自動的に提供しますが、シングルスタックまたはデュアルスタックサービス用に Ingress コントローラーを設定できます。

前提条件

- ベアメタルに OpenShift Container Platform クラスタをデプロイしていること。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. Ingress コントローラーが、IPv4 / IPv6 を介してトラフィックをワークロードに提供するようにするには、**ipFamilies** フィールドおよび **ipFamilyPolicy** フィールドを設定して、サービス YAML ファイルを作成するか、既存のサービス YAML ファイルを変更します。以下に例を示します。

サービス YAML ファイルの例

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: yyyy-mm-ddT00:00:00Z
  labels:
    name: <service_name>
    manager: kubectl-create
    operation: Update
    time: yyyy-mm-ddT00:00:00Z
  name: <service_name>
  namespace: <namespace_name>
  resourceVersion: "<resource_version_number>"
  selfLink: "/api/v1/namespaces/<namespace_name>/services/<service_name>"
  uid: <uid_number>
spec:
  clusterIP: 172.30.0.0/16
  clusterIPs: ①
  - 172.30.0.0/16
  - <second_IP_address>
  ipFamilies: ②
  - IPv4
  - IPv6
  ipFamilyPolicy: RequireDualStack ③
  ports:
    - port: 8080
      protocol: TCP
      targetport: 8080
  selector:
    name: <namespace_name>
  sessionAffinity: None
  type: ClusterIP
status:
  loadbalancer: {}
```

- ① デュアルスタックインスタンスでは、2つの異なる **clusterIPs** が提供されます。

- 2 シングルスタックインスタンスの場合は、**IPv4** または **IPv6** と入力します。デュアルスタックインスタンスの場合は、**IPv4** と **IPv6** の両方を入力します。
- 3 シングルスタックインスタンスの場合は、**SingleStack** と入力します。デュアルスタックインスタンスの場合は、**RequireDualStack** と入力します。

これらのリソースは、対応する **endpoints** を生成します。Ingress コントローラーは、**endpointslices** を監視するようになりました。

2. **endpoints** を表示するには、以下のコマンドを入力します。

```
$ oc get endpoints
```

3. **endpointslices** を表示するには、以下のコマンドを入力します。

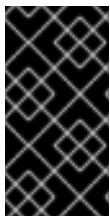
```
$ oc get endpointslices
```

関連情報

- [appsDomain オプションを使用した代替クラスタードメインの指定](#)

17.2. セキュリティー保護されたルート

セキュアなルートは、複数の TLS 終端タイプを使用してクライアントに証明書を提供できます。以下のセクションでは、カスタム証明書を使用して re-encrypt、edge、および passthrough ルートを作成する方法を説明します。



重要

パブリックエンドポイントを使用して Microsoft Azure にルートを作成する場合、リソース名は制限されます。特定の用語を使用するリソースを作成することはできません。Azure が制限する語の一覧は、Azure ドキュメントの [Resolve reserved resource name errors](#) を参照してください。

17.2.1. カスタム証明書を使用した re-encrypt ルートの作成

oc create route コマンドを使用し、カスタム証明書と共に reencrypt TLS termination を使用してセキュアなルートを設定できます。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアが必要です。ここで、証明書はルートホストに対して有効となっています。
- 証明書チェーンを完了する PEM エンコードされたファイルの別の CA 証明書が必要です。
- PEM エンコードされたファイルの別の宛先 CA 証明書が必要です。
- 公開する必要があるサービスが必要です。



注記

パスワードで保護されるキーファイルはサポートされません。キーファイルからパスワードを削除するには、以下のコマンドを使用します。

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

手順

この手順では、カスタム証明書および reencrypt TLS termination を使用して **Route** リソースを作成します。以下では、証明書/キーのペアが現在の作業ディレクトリーの **tls.crt** および **tls.key** ファイルにあることを前提としています。また、Ingress コントローラーがサービスの証明書を信頼できるように宛先 CA 証明書を指定する必要もあります。必要な場合には、証明書チェーンを完了するために CA 証明書を指定することもできます。**tls.crt**、**tls.key**、**cacert.crt**、および (オプションで) **ca.crt** を実際のパス名に置き換えます。**frontend** を、公開する必要がある **Service** リソースに置き換えます。**www.example.com** を適切な名前に置き換えます。

- reencrypt TLS 終端およびカスタム証明書を使用してセキュアな **Route** リソースを作成します。

```
$ 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
```

結果として生成される **Route** リソースを検査すると、以下のようになります。

セキュアなルートの YAML 定義

```
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-----
```

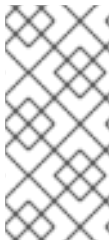
他のオプションについては、`oc create route reencrypt --help` を参照してください。

17.2.2. カスタム証明書を使用した edge ルートの作成

`oc create route` コマンドを使用し、edge TLS termination とカスタム証明書を使用してセキュアなルートを設定できます。edge ルートの場合、Ingress コントローラーは、トラフィックを宛先 Pod に転送する前に TLS 暗号を終了します。ルートは、Ingress コントローラーがルートに使用する TLS 証明書およびキーを指定します。

前提条件

- PEM エンコードされたファイルに証明書/キーのペアが必要です。ここで、証明書はルートホストに対して有効となっています。
- 証明書チェーンを完了する PEM エンコードされたファイルの別の CA 証明書が必要です。
- 公開する必要があるサービスが必要です。



注記

パスワードで保護されるキーファイルはサポートされません。キーファイルからパスワードを削除するには、以下のコマンドを使用します。

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

手順

この手順では、カスタム証明書および edge TLS termination を使用して **Route** リソースを作成します。以下では、証明書/キーのペアが現在の作業ディレクトリーの `tls.crt` および `tls.key` ファイルにあることを前提としています。必要な場合には、証明書チェーンを完了するために CA 証明書を指定することもできます。`tls.crt`、`tls.key`、および (オプションで) `ca.crt` を実際のパス名に置き換えます。`frontend` を、公開する必要があるサービスの名前に置き換えます。`www.example.com` を適切な名前に置き換えます。

- edge TLS termination およびカスタム証明書を使用して、セキュアな **Route** リソースを作成します。

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

結果として生成される **Route** リソースを検査すると、以下のようになります。

セキュアなルートの YAML 定義

```
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-----

```

他のオプションについては、**oc create route edge --help** を参照してください。

17.2.3. passthrough ルートの作成

oc create route コマンドを使用し、passthrough termination を使用してセキュアなルートを設定できます。passthrough termination では、暗号化されたトラフィックが TLS 終端を提供するルーターなしに宛先に直接送信されます。そのため、ルートでキーや証明書は必要ありません。

前提条件

- 公開する必要があるサービスが必要です。

手順

- Route** リソースを作成します。

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

結果として生成される **Route** リソースを検査すると、以下のようになります。

passthrough termination を使用したセキュリティー保護されたルート

```

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 オブジェクトの名前で、63 文字に制限されます。

2 **termination** フィールドを **passthrough** に設定します。これは、必要な唯一の **tls** フィールドです。

ルトです。

- 3 オプションの **insecureEdgeTerminationPolicy**。唯一有効な値は **None**、**Redirect**、または空の値です (無効にする場合)。

宛先 Pod は、エンドポイントでトラフィックに証明書を提供します。これは、必須となるクライアント証明書をサポートするための唯一の方法です (相互認証とも呼ばれる)。

第18章 INGRESS クラスタートラフィックの設定

18.1. INGRESS クラスタートラフィックの設定の概要

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする以下の方法を提供します。

以下の方法が推奨されます。以下は、これらの方法の優先される順です。

- HTTP/HTTPS を使用する場合は Ingress コントローラーを使用する。
- HTTPS 以外の TLS で暗号化されたプロトコルを使用する場合、たとえば、SNI ヘッダーを使用する TLS の場合は、Ingress コントローラーを使用します。
- それ以外の場合は、ロードバランサー、外部 IP、または **NodePort** を使用します。

方法	目的
Ingress コントローラーの使用	HTTP/HTTPS トラフィックおよび HTTPS 以外の TLS で暗号化されたプロトコル (TLS と SNI ヘッダーの使用など) へのアクセスを許可します。
ロードバランサーサービスを使用した外部 IP の自動割り当て	プールから割り当てられた IP アドレスを使った非標準ポートへのトラフィックを許可します。ほとんどのクラウドプラットフォームは、ロードバランサーの IP アドレスでサービスを開始する方法を提供します。
MetalLB および MetalLB Operator について	マシンネットワーク上のプールから特定の IP アドレスまたはアドレスへのトラフィックを許可します。ベアメタルインストールまたはベアメタルのようなプラットフォームの場合、MetalLB は、ロードバランサーの IP アドレスを使用してサービスを開始する方法を提供します。
外部 IP のサービスへの手動割り当て	特定の IP アドレスを使った非標準ポートへのトラフィックを許可します。
NodePort の設定	クラスターのすべてのノードでサービスを公開します。

18.1.1. 比較: 外部 IP アドレスへのフォールトトレランスアクセス

外部 IP アドレスへのアクセスを提供する通信メソッドの場合、IP アドレスへのフォールトトレランスアクセスは別の考慮事項となります。以下の機能は、外部 IP アドレスへのフォールトトレランスアクセスを提供します。

IP フェイルオーバー

IP フェイルオーバーはノードセットの仮想 IP アドレスのプールを管理します。これは、Keepalived および Virtual Router Redundancy Protocol (VRRP) で実装されます。IP フェイルオーバーはレイヤー 2 のメカニズムのみで、マルチキャストに依存します。マルチキャストには、一部のネット

ワークに欠点がある場合があります。

MetalLB

MetalLB にはレイヤー 2 モードがありますが、マルチキャストは使用されません。レイヤー 2 モードには、1つのノードで外部 IP アドレスのトラフィックをすべて転送する欠点があります。

外部 IP アドレスの手動割り当て

クラスターを、外部 IP アドレスをサービスに割り当てるために使用される IP アドレスブロックで設定できます。デフォルトでは、この機能は無効にされています。この機能は柔軟性がありますが、クラスターまたはネットワーク管理者に最大の負担をかけます。クラスターは、外部 IP 宛てのトラフィックを受信する準備ができていますが、各顧客は、トラフィックをノードにルーティングする方法を決定する必要があります。

18.2. サービスの EXTERNALIP の設定

クラスター管理者は、トラフィックをクラスター内のサービスに送信できるクラスター外の IP アドレスブロックを指定できます。

この機能は通常、ベアメタルハードウェアにインストールされているクラスターに最も役立ちます。

18.2.1. 前提条件

- ネットワークインフラストラクチャーは、外部 IP アドレスのトラフィックをクラスターにルーティングする必要があります。

18.2.2. ExternalIP について

クラウド以外の環境では、OpenShift Container Platform は **ExternalIP** 機能を使用して外部 IP アドレスの **Service** オブジェクトの **spec.externalIPs[]** フィールドへの割り当てをサポートします。このフィールドを設定すると、OpenShift Container Platform は追加の仮想 IP アドレスをサービスに割り当てます。IP アドレスは、クラスターに定義されたサービスネットワーク外に指定できます。 **type=NodePort** が設定されたサービスと同様に ExternalIP 機能で設定されたサービスにより、トラフィックを負荷分散のためにローカルノードに転送することができます。

ネットワークインフラストラクチャーを設定し、定義する外部 IP アドレスブロックがクラスターにルーティングされるようにする必要があります。

OpenShift Container Platform は以下の機能を追加して Kubernetes の ExternalIP 機能を拡張します。

- 設定可能なポリシーでの、ユーザーによる外部 IP アドレスの使用の制限
- 要求時の外部 IP アドレスのサービスへの自動割り当て



警告

ExternalIP 機能の使用はデフォルトで無効にされます。これは、外部 IP アドレスへのクラスター内のトラフィックがそのサービスにダイレクトされるため、セキュリティ上のリスクを生じさせる可能性があります。これにより、クラスターユーザーは外部リソースについての機密性の高いトラフィックをインターセプトできるようになります。



重要

この機能は、クラウド以外のデプロイメントでのみサポートされます。クラウドデプロイメントの場合、クラウドの自動デプロイメントのためにロードバランサーサービスを使用し、サービスのエンドポイントをターゲットに設定します。

以下の方法で外部 IP アドレスを割り当てることができます。

外部 IP の自動割り当て

OpenShift Container Platform は、**spec.type=LoadBalancer** を設定して **Service** オブジェクトを作成する際に、IP アドレスを **autoAssignCIDRs** CIDR ブロックから **spec.externalIPs[]** 配列に自動的に割り当てます。この場合、OpenShift Container Platform はロードバランサーサービスタイプのクラウド以外のバージョンを実装し、IP アドレスをサービスに割り当てます。自動割り当てはデフォルトで無効にされており、以下のセクションで説明されているように、これはクラスター管理者が設定する必要があります。

外部 IP の手動割り当て

OpenShift Container Platform は **Service** オブジェクトの作成時に **spec.externalIPs[]** 配列に割り当てられた IP アドレスを使用します。別のサービスによってすでに使用されている IP アドレスを指定することはできません。

18.2.2.1. ExternalIP の設定

OpenShift Container Platform での外部 IP アドレスの使用は、**cluster** という名前の **Network.config.openshift.io** CR の以下のフィールドで管理されます。

- **spec.externalIP.autoAssignCIDRs** は、サービスの外部 IP アドレスを選択する際にロードバランサーによって使用される IP アドレスブロックを定義します。OpenShift Container Platform は、自動割り当て用の単一 IP アドレスブロックのみをサポートします。これは、ExternalIP をサービスに手動で割り当てる際に、制限された数の共有 IP アドレスのポート領域を管理しなくてはならない場合よりも単純になります。自動割り当てが有効な場合には、**spec.type=LoadBalancer** が設定された **Service** オブジェクトには外部 IP アドレスが割り当てられます。
- **spec.externalIP.policy** は、IP アドレスを手動で指定する際に許容される IP アドレスブロックを定義します。OpenShift Container Platform は、**spec.externalIP.autoAssignCIDRs** で定義される IP アドレスブロックにポリシールールを適用しません。

ルーティングが正しく行われると、設定された外部 IP アドレスブロックからの外部トラフィックは、サービスが公開する TCP ポートまたは UDP ポートを介してサービスのエンドポイントに到達できます。



重要

クラスター管理者は、OpenShiftSDN ネットワークタイプと OVN-Kubernetes ネットワークタイプの両方で externalIP へのルーティングを設定する必要があります。割り当てる IP アドレスブロックがクラスター内の 1 つ以上のノードで終了することを確認する必要があります。詳細は、[Kubernetes External IPs](#) を参照してください。

OpenShift Container Platform は IP アドレスの自動および手動割り当ての両方をサポートしており、それぞれのアドレスは 1 つのサービスの最大数に割り当てられることが保証されます。これにより、各サービスは、ポートが他のサービスで公開されているかによらず、自らの選択したポートを公開できます。



注記

OpenShift Container Platform の **autoAssignCIDRs** で定義された IP アドレスブロックを使用するには、ホストのネットワークに必要な IP アドレスの割り当ておよびルーティングを設定する必要があります。

以下の YAML は、外部 IP アドレスが設定されたサービスについて説明しています。

spec.externalIPs[] が設定された Service オブジェクトの例

```
apiVersion: v1
kind: Service
metadata:
  name: http-service
spec:
  clusterIP: 172.30.163.110
  externalIPs:
  - 192.168.132.253
  externalTrafficPolicy: Cluster
  ports:
  - name: httpport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
```

18.2.2.2. 外部 IP アドレスの割り当ての制限

クラスター管理者は、IP アドレスブロックを指定して許可および拒否できます。

制限は、**cluster-admin** 権限を持たないユーザーにのみ適用されます。クラスター管理者は、サービスの **spec.externalIPs[]** フィールドを任意の IP アドレスに常に設定できます。

spec.ExternalIP.policy フィールドを指定して、**policy** オブジェクトが定義された IP アドレスポリシーを設定します。ポリシーオブジェクトには以下の形があります。

```
{
  "policy": {
    "allowedCIDRs": [],
    "rejectedCIDRs": []
  }
}
```

ポリシーの制限を設定する際に、以下のルールが適用されます。

- **policy={}** が設定される場合、**spec.ExternalIPs[]** が設定されている **Service** オブジェクトの作成は失敗します。これは OpenShift Container Platform のデフォルトです。**policy=null** が設定される動作は同一です。
- **policy** が設定され、**policy.allowedCIDRs[]** または **policy.rejectedCIDRs[]** のいずれかが設定される場合、以下のルールが適用されます。
 - **allowedCIDRs[]** と **rejectedCIDRs[]** の両方が設定される場合、**rejectedCIDRs[]** が **allowedCIDRs[]** よりも優先されます。
 - **allowedCIDRs[]** が設定される場合、**spec.ExternalIPs[]** が設定されている **Service** オブジェクトの作成は、指定された IP アドレスが許可される場合にのみ正常に実行されます。
 - **rejectedCIDRs[]** が設定される場合、**spec.ExternalIPs[]** が設定されている **Service** オブジェクトの作成は、指定された IP アドレスが拒否されていない場合にのみ正常に実行されます。

18.2.2.3. ポリシーオブジェクトの例

以下に続く例では、複数のポリシー設定の例を示します。

- 以下の例では、ポリシーは OpenShift Container Platform が外部 IP アドレスが指定されたサービスを作成するのを防ぎます。

Service オブジェクトの **spec.externalIPs[]** に指定された値を拒否するポリシーの例

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

- 以下の例では、**allowedCIDRs** および **rejectedCIDRs** フィールドの両方が設定されます。

許可される、および拒否される CIDR ブロックの両方を含むポリシーの例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy:
      allowedCIDRs:
        - 172.16.66.10/23
      rejectedCIDRs:
        - 172.16.66.10/24
  ...
```

- 以下の例では、**policy** は **null** に設定されます。**null** に設定されている場合、**oc get networks.config.openshift.io -o yaml** を入力して設定オブジェクトを検査する際に、**policy** フィールドは出力に表示されません。

Service オブジェクトの `spec.externalIPs[]` に指定された値を許可するポリシーの例

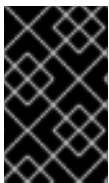
```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    policy: null
  ...

```

18.2.3. ExternalIP アドレスブロックの設定

ExternalIP アドレスブロックの設定は、**cluster** という名前の Network カスタムリソース (CR) で定義されます。ネットワーク CR は **config.openshift.io** API グループに含まれます。



重要

クラスタのインストール時に、Cluster Version Operator (CVO) は **cluster** という名前のネットワーク CR を自動的に作成します。このタイプのその他の CR オブジェクトの作成はサポートされていません。

以下の YAML は ExternalIP 設定について説明しています。

cluster という名前の `network.config.openshift.io` CR

```

apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  externalIP:
    autoAssignCIDRs: [] 1
    policy: 2
  ...

```

- 1** 外部 IP アドレスのサービスへの自動割り当てに使用できる CIDR 形式で IP アドレスブロックを定義します。1つの IP アドレス範囲のみが許可されます。
- 2** IP アドレスのサービスへの手動割り当ての制限を定義します。制限が定義されていない場合は、**Service** オブジェクトに **spec.externalIP** フィールドを指定しても許可されません。デフォルトで、制限は定義されません。

以下の YAML は、**policy** スタンザのフィールドについて説明しています。

Network.config.openshift.io `policy` スタンザ

```

policy:
  allowedCIDRs: [] 1
  rejectedCIDRs: [] 2

```

- 1** CIDR 形式の許可される IP アドレス範囲の一覧。

2 CIDR 形式の拒否される IP アドレス範囲の一覧。

外部 IP 設定の例

外部 IP アドレスプールの予想される複数の設定が以下の例で表示されています。

- 以下の YAML は、自動的に割り当てられた外部 IP アドレスを有効にする設定について説明しています。

spec.externallIP.autoAssignCIDRs が設定された設定例

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

- 以下の YAML は、許可された、および拒否された CIDR 範囲のポリシールールを設定します。

spec.externallIP.policy が設定された設定例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externallIP:
    policy:
      allowedCIDRs:
        - 192.168.132.0/29
        - 192.168.132.8/29
      rejectedCIDRs:
        - 192.168.132.7/32
```

18.2.4. クラスターの外部 IP アドレスブロックの設定

クラスター管理者は、以下の ExternallIP を設定できます。

- **Service** オブジェクトの **spec.clusterIP** フィールドを自動的に設定するために OpenShift Container Platform によって使用される ExternallIP アドレスブロック。
- IP アドレスを制限するポリシーオブジェクトは **Service** オブジェクトの **spec.clusterIP** 配列に手動で割り当てられます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. オプション: 現在の外部 IP 設定を表示するには、以下のコマンドを入力します。

```
$ oc describe networks.config cluster
```

2. 設定を編集するには、以下のコマンドを入力します。

```
$ oc edit networks.config cluster
```

3. 以下の例のように ExternallIP 設定を変更します。

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externallIP: ①
  ...
```

- ① **externallIP** スタンザの設定を指定します。

4. 更新された ExternallIP 設定を確認するには、以下のコマンドを入力します。

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

18.2.5. 次のステップ

- [サービスの外部 IP を使用した ingress クラスタートラフィックの設定](#)

18.3. INGRESS コントローラーを使用した INGRESS クラスターの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法は Ingress コントローラーを使用します。

18.3.1. Ingress コントローラーおよびルートの使用

Ingress Operator は Ingress コントローラーおよびワイルドカード DNS を管理します。

Ingress コントローラーの使用は、OpenShift Container Platform クラスターへの外部アクセスを許可するための最も一般的な方法です。

Ingress コントローラーは外部要求を許可し、設定されたルートに基づいてそれらをプロキシ送信するように設定されます。これは、HTTP、SNI を使用する HTTPS、SNI を使用する TLS に限定されており、SNI を使用する TLS で機能する Web アプリケーションやサービスには十分な設定です。

管理者と連携して Ingress コントローラーを設定します。外部要求を許可し、設定されたルートに基づいてそれらをプロキシ送信するように Ingress コントローラーを設定します。

管理者はワイルドカード DNS エントリーを作成してから Ingress コントローラーを設定できます。その後は管理者に問い合わせることなく edge Ingress コントローラーと連携できます。

デフォルトで、クラスター内のすべての Ingress コントローラーはクラスター内の任意のプロジェクトで作成されたすべてのルートを許可します。

Ingress コントローラー:

- デフォルトでは2つのレプリカがあるので、これは2つのワーカーノードで実行する必要があります。
- 追加のノードにレプリカを組み込むためにスケールアップすることができます。



注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

18.3.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが1名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- OpenShift Container Platform クラスターを、1つ以上のマスターと1つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

18.3.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. **oc new-project** コマンドを実行して、サービス用の新しいプロジェクトを作成します。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。

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


3. サービスが作成されたことを確認するには、以下のコマンドを実行します。

```
$ oc get svc -n myproject
```

出力例

```
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
nodejs-ex ClusterIP  172.30.197.157 <none>      8080/TCP  70s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

18.3.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project myproject
```

3. **oc expose service** コマンドを実行して、ルートを公開します。

```
$ oc expose service nodejs-ex
```

出力例

```
route.route.openshift.io/nodejs-ex exposed
```

4. サービスが公開されていることを確認するには、cURL などのツールを使って、クラスター外からサービスにアクセスできることを確認します。
 - a. ルートのホスト名を調べるには、**oc get route** コマンドを使用します。

```
$ oc get route
```

出力例

```
NAME      HOST/PORT                                PATH  SERVICES  PORT  TERMINATION
WILDCARD
nodejs-ex nodejs-ex-myproject.example.com        nodejs-ex  8080-tcp  None
```

- b. cURL を使用して、ホストが GET 要求に応答することを確認します。

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

出力例

```
HTTP/1.1 200 OK
```

```
...
```

18.3.5. ルートラベルを使用した Ingress コントローラーのシャード化の設定

ルートラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーがルートセクターによって選択される任意 namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。

手順

1. **router-internal.yaml** ファイルを編集します。

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

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

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

Ingress コントローラーは、**type: sharded** というラベルのある namespace のルートを選択します。

18.3.6. namespace ラベルを使用した Ingress コントローラーのシャード化の設定

namespace ラベルを使用した Ingress コントローラーのシャード化とは、Ingress コントローラーが namespace セクターによって選択される任意の namespace の任意のルートを提供することを意味します。

Ingress コントローラーのシャード化は、一連の Ingress コントローラー間で着信トラフィックの負荷を

分散し、トラフィックを特定の Ingress コントローラーに分離する際に役立ちます。たとえば、Company A のトラフィックをある Ingress コントローラーに指定し、Company B を別の Ingress コントローラーに指定できます。



警告

Keepalived Ingress VIP をデプロイする場合は、**endpoint Publishing Strategy** パラメーターに **Host Network** の値が割り当てられた、デフォルト以外の Ingress Controller をデプロイしないでください。デプロイしてしまうと、問題が発生する可能性があります。**endpoint Publishing Strategy** に **Host Network** ではなく、**Node Port** という値を使用してください。

手順

1. **router-internal.yaml** ファイルを編集します。

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

出力例

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

2. Ingress コントローラーの **router-internal.yaml** ファイルを適用します。

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

Ingress コントローラーは、**type: sharded** というラベルのある namespace セレクターによって選択される namespace のルートを選択します。

18.3.7. 関連情報

- Ingress Operator はワイルドカード DNS を管理します。詳細は、[OpenShift Container Platform の Ingress Operator](#)、[クラスターのベアメタルへのインストール](#)、および [クラスターの vSphere へのインストール](#) を参照してください。

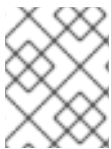
18.4. ロードバランサーを使用した INGRESS クラスターの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法では、ロードバランサーを使用します。

18.4.1. ロードバランサーを使用したトラフィックのクラスターへの送信

特定の外部 IP アドレスを必要としない場合、ロードバランサーサービスを OpenShift Container Platform クラスターへの外部アクセスを許可するよう設定することができます。

ロードバランサーサービスは固有の IP を割り当てます。ロードバランサーには単一の edge ルーター IP があります (これは仮想 IP (VIP) の場合もありますが、初期の負荷分散では単一マシンになります)。



注記

プールが設定される場合、これはクラスター管理者によってではなく、インフラストラクチャーレベルで実行されます。



注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

18.4.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが1名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin username
```

- OpenShift Container Platform クラスターを、1つ以上のマスターと1つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

18.4.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスタ管理者として **oc** CLI をインストールし、ログインします。

手順

1. **oc new-project** コマンドを実行して、サービス用の新しいプロジェクトを作成します。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。

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

3. サービスが作成されたことを確認するには、以下のコマンドを実行します。

```
$ oc get svc -n myproject
```

出力例

```
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
nodejs-ex ClusterIP  172.30.197.157 <none>      8080/TCP  70s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

18.4.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project myproject
```

3. **oc expose service** コマンドを実行して、ルートを公開します。

```
$ oc expose service nodejs-ex
```

出力例

```
route.route.openshift.io/nodejs-ex exposed
```

4. サービスが公開されていることを確認するには、cURL などのツールを使って、クラスタ外からサービスにアクセスできることを確認します。
 - a. ルートのホスト名を調べるには、**oc get route** コマンドを使用します。

```
$ oc get route
```

出力例

```
NAME          HOST/PORT          PATH SERVICES  PORT      TERMINATION
WILDCARD
nodejs-ex    nodejs-ex-myproject.example.com    nodejs-ex  8080-tcp  None
```

- b. cURL を使用して、ホストが GET 要求に応答することを確認します。

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

出力例

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

18.4.5. ロードバランサーサービスの作成

以下の手順を使用して、ロードバランサーサービスを作成します。

前提条件

- 公開するプロジェクトとサービスがあること。

手順

ロードバランサーサービスを作成するには、以下を実行します。

- OpenShift Container Platform にログインします。
- 公開するサービスが置かれているプロジェクトを読み込みます。

```
$ oc project project1
```

- コントロールプレーンノードでテキストファイルを開き、以下のテキストを貼り付け、必要に応じてファイルを編集します。

ロードバランサー設定ファイルのサンプル

```
apiVersion: v1
kind: Service
metadata:
  name: egress-2 1
spec:
  ports:
    - name: db
      port: 3306 2
  loadBalancerIP:
  loadBalancerSourceRanges: 3
    - 10.0.0.0/8
    - 192.168.0.0/16
  type: LoadBalancer 4
  selector:
    name: mysql 5
```

- 1 ロードバランサーサービスの説明となる名前を入力します。
- 2 公開するサービスがリッスンしている同じポートを入力します。
- 3 特定の IP アドレスの一覧を入力して、ロードバランサー経由でトラフィックを制限します。クラウドプロバイダーがこの機能に対応していない場合、このフィールドは無視されます。
- 4 タイプに **loadbalancer** を入力します。
- 5 サービスの名前を入力します。



注記

ロードバランサーを介して特定の IP アドレスへのトラフィックを制限するには、**loadBalancerSourceRanges** フィールドを設定するのではなく、**service.beta.kubernetes.io/load-balancer-source-ranges** アノテーションを使用することが推奨されます。アノテーションを使用すると、OpenShift API により簡単に移行でき、今後のリリースで実装されます。

4. ファイルを保存し、終了します。
5. 以下のコマンドを実行してサービスを作成します。

```
$ oc create -f <file-name>
```

以下に例を示します。

```
$ oc create -f mysql-lb.yaml
```

6. 以下のコマンドを実行して新規サービスを表示します。

```
$ oc get svc
```

出力例

```
NAME      TYPE          CLUSTER-IP    EXTERNAL-IP          PORT(S)
AGE
egress-2  LoadBalancer 172.30.22.226  ad42f5d8b303045-487804948.example.com
3306:30357/TCP 15m
```

有効にされたクラウドプロバイダーがある場合、サービスには外部 IP アドレスが自動的に割り当てられます。

7. マスターで cURL などのツールを使用し、パブリック IP アドレスを使用してサービスに到達できることを確認します。

```
$ curl <public-ip>:<port>
```

以下に例を示します。

```
$ curl 172.29.121.74:3306
```

このセクションの例では、クライアントアプリケーションを必要とする MySQL サービスを使用しています。**Got packets out of order** のメッセージと共に文字ストリングを取得する場合は、このサービスに接続していることとなります。

MySQL クライアントがある場合は、標準 CLI コマンドでログインします。

```
$ mysql -h 172.30.131.89 -u admin -p
```

出力例

```
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.

MySQL [(none)]>
```

18.5. ネットワークロードバランサーを使用した AWS での INGRESS クラスタートラフィックの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法では、クライアントの IP アドレスをノードに転送する Network Load Balancer (NLB) を使用します。NLB を新規または既存の AWS クラスタに設定することができます。

18.5.1. Ingress Controller Classic Load Balancer の Network Load Balancer への置き換え

Classic Load Balancer (CLB) を使用している Ingress Controller は、AWS の Network Load Balancer (NLB) を使用している Ingress Controller に置き換えることができます。



警告

この手順を実行すると、新しい DNS レコードの伝搬、新しいロードバランサーのプロビジョニングなどの要因により、数分間にわたる障害が発生することが予想されます。この手順を適用すると、Ingress Controller ロードバランサーの IP アドレスや正規名が変更になる場合があります。

手順

1. 新しいデフォルトの Ingress Controller を含むファイルを作成します。以下の例では、デフォルトの Ingress Controller の範囲が **External** で、その他のカスタマイズをしていないことを想定しています。

ingresscontroller.yml ファイルの例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
```



```

name: default
namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
      aws:
        type: NLB
      type: LoadBalancerService

```

デフォルトの Ingress Controller が他にカスタマイズされている場合には、それに応じてファイルを修正してください。

2. Ingress Controller の YAML ファイルを強制的に置き換えます。

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

Ingress Controller の置き換えが完了するまでお待ちください。数分ほど、サービスが停止します。

18.5.2. 既存 AWS クラスタでの Ingress コントローラーネットワークロードバランサーの設定

AWS Network Load Balancer (NLB) がサポートする Ingress コントローラーを既存のクラスタに作成できます。

前提条件

- AWS クラスタがインストールされている。
- インフラストラクチャーリソースの **PlatformStatus** は AWS である必要があります。
 - **PlatformStatus** が AWS であることを確認するには、以下を実行します。

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

手順

既存のクラスタの AWS NLB がサポートする Ingress コントローラーを作成します。

1. Ingress コントローラーのマニフェストを作成します。

```
$ cat ingresscontroller-aws-nlb.yaml
```

出力例

```

apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: $my_ingress_controller 1
  namespace: openshift-ingress-operator

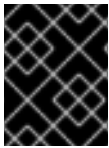
```

```
spec:
  domain: $my_unique_ingress_domain ❷
  endpointPublishingStrategy:
    type: LoadBalancerService
  loadBalancer:
    scope: External ❸
  providerParameters:
    type: AWS
    aws:
      type: NLB
```

- ❶ **\$my_ingress_controller** を Ingress コントローラーの一意の名前に置き換えます。
- ❷ **\$my_unique_ingress_domain** を、クラスター内のすべての Ingress コントローラー間で一意のドメイン名に置き換えます。この変数は、DNS 名 **<clustername>.<domain>** のサブドメインである必要があります。
- ❸ **External** を内部 NLB を使用するために **Internal** に置き換えることができます。

2. クラスターにリソースを作成します。

```
$ oc create -f ingresscontroller-aws-nlb.yaml
```



重要

新規 AWS クラスターで Ingress コントローラー NLB を設定する前に、[インストール設定ファイルの作成](#) 手順を実行する必要があります。

18.5.3. 新規 AWS クラスターでの Ingress コントローラーネットワークロードバランサーの設定

新規クラスターに AWS Network Load Balancer (NLB) がサポートする Ingress コントローラーを作成できます。

前提条件

- **install-config.yaml** ファイルを作成し、これに対する変更を完了します。

手順

新規クラスターの AWS NLB がサポートする Ingress コントローラーを作成します。

1. インストールプログラムが含まれるディレクトリーに切り替え、マニフェストを作成します。

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

- ❶ **<installation_directory>** については、クラスターの **install-config.yaml** ファイルが含まれるディレクトリーの名前を指定します。

2. **cluster-ingress-default-ingresscontroller.yaml** という名前のファイルを **<installation_directory>/manifests/** ディレクトリーに作成します。

```
$ touch <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml 1
```

- 1 **<installation_directory>** については、クラスターの **manifests/** ディレクトリーが含まれるディレクトリー名を指定します。

ファイルの作成後は、以下のようにいくつかのネットワーク設定ファイルが **manifests/** ディレクトリーに置かれます。

```
$ ls <installation_directory>/manifests/cluster-ingress-default-ingresscontroller.yaml
```

出力例

```
cluster-ingress-default-ingresscontroller.yaml
```

3. エディターで **cluster-ingress-default-ingresscontroller.yaml** ファイルを開き、必要な Operator 設定を記述するカスタムリソース (CR) を入力します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  creationTimestamp: null
  name: default
  namespace: openshift-ingress-operator
spec:
  endpointPublishingStrategy:
    loadBalancer:
      scope: External
      providerParameters:
        type: AWS
      aws:
        type: NLB
    type: LoadBalancerService
```

4. **cluster-ingress-default-ingresscontroller.yaml** ファイルを保存し、テキストエディターを終了します。
5. オプション: **manifests/cluster-ingress-default-ingresscontroller.yaml** ファイルをバックアップします。インストールプログラムは、クラスターの作成時に **manifests/** ディレクトリーを削除します。

18.5.4. 関連情報

- [ネットワークのカスタマイズによる AWS へのクラスターのインストール](#)
- 詳細は、[Network Load Balancer support on AWS](#) を参照してください。

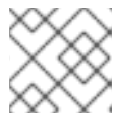
18.6. サービスの外部 IP を使用した INGRESS クラスタートラフィックの設定

外部 IP アドレスをサービスに割り当てることで、これをクラスター外のトラフィックで使用できるようにします。通常、これはベアメタルハードウェアにインストールされているクラスターの場合にのみ役立ちます。外部ネットワークインフラストラクチャーは、トラフィックをサービスにルーティングす

るように正しく設定される必要があります。

18.6.1. 前提条件

- クラスタは ExternalIP が有効にされた状態で設定されます。詳細は、[サービスの ExternalIP の設定](#) について参照してください。



注記

egress IP に同じ ExternalIP を使用しないでください。

18.6.2. ExternalIP のサービスへの割り当て

ExternalIP をサービスに割り当てることができます。クラスタが ExternalIP を自動的に割り当てするように設定されている場合、ExternalIP をサービスに手動で割り当てる必要がない場合があります。

手順

1. オプション: ExternalIP で使用するために設定される IP アドレス範囲を確認するには、以下のコマンドを入力します。

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

autoAssignCIDRs が設定されている場合、**spec.externalIPs** フィールドが指定されていない場合、OpenShift Container Platform は ExternalIP を新規 **Service** オブジェクトに自動的に割り当てます。

2. ExternalIP をサービスに割り当てます。
 - a. 新規サービスを作成する場合は、**spec.externalIPs** フィールドを指定し、1つ以上の有効な IP アドレスの配列を指定します。以下に例を示します。

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

- b. ExternalIP を既存のサービスに割り当てる場合は、以下のコマンドを入力します。**<name>** をサービス名に置き換えます。**<ip_address>** を有効な ExternalIP アドレスに置き換えます。コンマで区切られた複数の IP アドレスを指定できます。

```
$ oc patch svc <name> -p \
{
  "spec": {
    "externalIPs": [ "<ip_address>" ]
  }
}
```

以下に例を示します。

```
$ oc patch svc mysql-55-rhel7 -p '{"spec":{"externalIPs":["192.174.120.10"]}}'
```

出力例

```
"mysql-55-rhel7" patched
```

3. ExternalIP アドレスがサービスに割り当てられていることを確認するには、以下のコマンドを入力します。新規サービスに ExternalIP を指定した場合、まずサービスを作成する必要があります。

```
$ oc get svc
```

出力例

NAME	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
mysql-55-rhel7	172.30.131.89	192.174.120.10	3306/TCP	13m

18.6.3. 関連情報

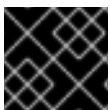
- [サービスの ExternalIP の設定](#)

18.7. NODEPORT を使用した INGRESS クラスタートラフィックの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使ってクラスター外からの通信を可能にする方法を提供します。この方法は **NodePort** を使用します。

18.7.1. NodePort を使用したトラフィックのクラスターへの送信

NodePort-type **Service** リソースを使用して、クラスター内のすべてのノードの特定のポートでサービスを公開します。ポートは **Service** リソースの **.spec.ports[*].nodePort** フィールドで指定されます。



重要

ノードポートを使用するには、追加のポートリソースが必要です。

NodePort は、ノードの IP アドレスの静的ポートでサービスを公開します。**NodePort** はデフォルトで **30000** から **32767** の範囲に置かれます。つまり、**NodePort** はサービスの意図されるポートに一致しないことが予想されます。たとえば、ポート **8080** はノードのポート **31020** として公開できます。

管理者は、外部 IP アドレスがノードにルーティングされることを確認する必要があります。

NodePort および外部 IP は独立しており、両方を同時に使用できます。



注記

このセクションの手順では、クラスターの管理者が事前に行っておく必要のある前提条件があります。

18.7.2. 前提条件

以下の手順を開始する前に、管理者は以下の条件を満たしていることを確認する必要があります。

- 要求がクラスターに到達できるように、クラスターネットワーク環境に対して外部ポートをセットアップします。
- クラスター管理者ロールを持つユーザーが1名以上いることを確認します。このロールをユーザーに追加するには、以下のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user cluster-admin <user_name>
```

- OpenShift Container Platform クラスターを、1つ以上のマスターと1つ以上のノード、およびクラスターへのネットワークアクセスのあるクラスター外のシステムと共に用意します。この手順では、外部システムがクラスターと同じサブセットにあることを前提とします。別のサブセットの外部システムに必要な追加のネットワーク設定については、このトピックでは扱いません。

18.7.3. プロジェクトおよびサービスの作成

公開するプロジェクトおよびサービスが存在しない場合、最初にプロジェクトを作成し、次にサービスを作成します。

プロジェクトおよびサービスがすでに存在する場合は、サービスを公開してルートを作成する手順に進みます。

前提条件

- クラスター管理者として **oc** CLI をインストールし、ログインします。

手順

1. **oc new-project** コマンドを実行して、サービス用の新しいプロジェクトを作成します。

```
$ oc new-project myproject
```

2. **oc new-app** コマンドを使用してサービスを作成します。

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

3. サービスが作成されたことを確認するには、以下のコマンドを実行します。

```
$ oc get svc -n myproject
```

出力例

```
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
nodejs-ex ClusterIP  172.30.197.157 <none>       8080/TCP 70s
```

デフォルトで、新規サービスには外部 IP アドレスがありません。

18.7.4. ルートの作成によるサービスの公開

oc expose コマンドを使用して、サービスをルートとして公開することができます。

手順

サービスを公開するには、以下を実行します。

1. OpenShift Container Platform にログインします。
2. 公開するサービスが置かれているプロジェクトにログインします。

```
$ oc project myproject
```

3. アプリケーションのノードポートを公開するには、以下のコマンドを入力します。OpenShift Container Platform は **30000-32767** 範囲の利用可能なポートを自動的に選択します。

```
$ oc expose service nodejs-ex --type=NodePort --name=nodejs-ex-nodeport --generator="service/v2"
```

出力例

```
service/nodejs-ex-nodeport exposed
```

4. オプション: サービスが公開されるノードポートで利用可能なことを確認するには、以下のコマンドを入力します。

```
$ oc get svc -n myproject
```

出力例

```
NAME             TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
nodejs-ex        ClusterIP     172.30.217.127 <none>       3306/TCP         9m44s
nodejs-ex-ingress NodePort      172.30.107.72  <none>       3306:31345/TCP  39s
```

5. オプション: **oc new-app** コマンドによって自動的に作成されたサービスを削除するには、以下のコマンドを入力します。

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

18.7.5. 関連情報

- [ノードポートサービス範囲の設定](#)

第19章 KUBERNETES NMSTATE

19.1. KUBERNETES NMSTATE OPERATOR について

Kubernetes NMState Operator は、NMState の OpenShift Container Platform クラスターのノード間でステートドリブンのネットワーク設定を実行するための Kubernetes API を提供します。Kubernetes NMState Operator は、ユーザーに対して、クラスターノードの各種のネットワークインターフェイスタイプ、DNS、およびルーティングを設定する機能を提供します。さらに、クラスターノードのデーモンは、各ノードの API サーバーへのネットワークインターフェイスの状態の定期的な報告を行います。

重要

Kubernetes NMState Operator はテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供することで、お客様による開発段階での機能テストの実施とフィードバックのご提供を可能にすることを目的としています。Open Shift Container Platform と Red Hat Virtualization (RHV) の両方で Kubernetes NMState Operator を同時に使用しないでください。このような設定はサポートされていません。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、[テクノロジープレビュー機能のサポート範囲](#) を参照してください。

OpenShift Container Platform で NMState を使用する前に、Kubernetes NMState Operator をインストールする必要があります。



警告

OVN-Kubernetes を使用する場合は、デフォルトゲートウェイインターフェイスの変更がサポートされていません。

19.1.1. Kubernetes NMState Operator のインストール

管理者権限でログインし、Web コンソールから Kubernetes NMState Operator をインストールする必要があります。インストールが完了すると、Operator はすべてのクラスターノードに NMState State Controller をデーモンセットとしてデプロイできます。

手順

1. **Operators** → **OperatorHub** を選択します。
2. **All Items** の下の検索フィールドに、**nmstate** と入力し、**Enter** をクリックして Kubernetes NMState Operator を検索します。
3. Kubernetes NMState Operator の検索結果をクリックします。
4. **Install** をクリックして、**Install Operator** ウィンドウを開きます。

5. **Install** をクリックして Operator をインストールします。
6. Operator のインストールが完了したら、**View Operator** をクリックします。
7. **Provided APIs** で **Create Instance** をクリックし、**kubernetes-nmstate** のインスタンスを作成するダイアログボックスを開きます。
8. ダイアログボックスの **Name** フィールドで、インスタンスの名前が **nmstate** であることを確認します。



注記

名前の制限は既知の問題です。インスタンスはクラスター全体のシングルトンです。

9. デフォルト設定を受け入れ、**Create** をクリックしてインスタンスを作成します。

概要

完了後に、Operator はすべてのクラスターノードに NMState State Controller をデーモンセットとしてデプロイしています。

19.2. ノードのネットワーク状態の確認

ノードのネットワーク状態は、クラスター内のすべてのノードのネットワーク設定です。

19.2.1. nmstate について

OpenShift Container Platform は **nmstate** を使用して、ノードネットワークの状態を報告し、これを設定します。これにより、単一の設定マニフェストをクラスターに適用して、すべてのノードに Linux ブリッジを作成するなどして、ネットワークポリシーの設定を変更することができます。

ノードのネットワークは、以下のオブジェクトによって監視され更新されます。

NodeNetworkState

そのノード上のネットワークの状態を報告します。

NodeNetworkConfigurationPolicy

ノードで要求されるネットワーク設定について説明します。**NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用して、インターフェイスの追加および削除など、ノードネットワーク設定を更新します。

NodeNetworkConfigurationEnactment

各ノードに制定されたネットワークポリシーを報告します。

OpenShift Container Platform は、以下の nmstate インターフェイスタイプの使用をサポートします。

- Linux Bridge
- VLAN
- Bond
- イーサネット



注記

OpenShift Container Platform クラスターが OVN-Kubernetes をデフォルトの Container Network Interface (CNI) プロバイダーとして使用する場合、OVN-Kubernetes のホストネットワークポロジの変更により、Linux ブリッジまたはボンディングをホストのデフォルトインターフェイスに割り当てることはできません。回避策として、ホストに接続されたセカンダリーネットワークインターフェイスを使用するか、OpenShift SDN デフォルト CNI プロバイダーに切り替えることができます。

19.2.2. ノードのネットワーク状態の表示

NodeNetworkState オブジェクトはクラスター内のすべてのノードにあります。このオブジェクトは定期的に更新され、ノードのネットワークの状態を取得します。

手順

1. クラスターのすべての **NodeNetworkState** オブジェクトを一覧表示します。

```
$ oc get nns
```

2. **NodeNetworkState** オブジェクトを検査して、そのノードにネットワークを表示します。この例の出力は、明確にするために編集されています。

```
$ oc get nns node01 -o yaml
```

出力例

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkState
metadata:
  name: node01 1
status:
  currentState: 2
  dns-resolver:
  ...
  interfaces:
  ...
  route-rules:
  ...
  routes:
  ...
lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" 3
```

- 1** **NodeNetworkState** オブジェクトの名前はノードから取られています。
- 2** **currentState** には、DNS、インターフェイス、およびルートを含む、ノードの完全なネットワーク設定が含まれます。
- 3** 最後に成功した更新のタイムスタンプ。これは、ノードが到達可能であり、レポートの鮮度の評価に使用できる限り定期的に更新されます。

19.3. ノードのネットワーク設定の更新

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用して、ノードからのインターフェイスの追加または削除など、ノードネットワーク設定を更新できます。



警告

OVN-Kubernetes を使用する場合は、デフォルトゲートウェイインターフェイスの変更がサポートされていません。

19.3.1. nmstate について

OpenShift Container Platform は **nmstate** を使用して、ノードネットワークの状態を報告し、これを設定します。これにより、単一の設定マニフェストをクラスターに適用して、すべてのノードに Linux ブリッジを作成するなどして、ネットワークポリシーの設定を変更することができます。

ノードのネットワークは、以下のオブジェクトによって監視され更新されます。

NodeNetworkState

そのノード上のネットワークの状態を報告します。

NodeNetworkConfigurationPolicy

ノードで要求されるネットワーク設定について説明します。**NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用して、インターフェイスの追加および削除など、ノードネットワーク設定を更新します。

NodeNetworkConfigurationEnactment

各ノードに制定されたネットワークポリシーを報告します。

OpenShift Container Platform は、以下の nmstate インターフェイスタイプの使用をサポートします。

- Linux Bridge
- VLAN
- Bond
- イーサネット



注記

OpenShift Container Platform クラスターが OVN-Kubernetes をデフォルトの Container Network Interface (CNI) プロバイダーとして使用する場合、OVN-Kubernetes のホストネットワークポロジの変更により、Linux ブリッジまたはボンディングをホストのデフォルトインターフェイスに割り当てることはできません。回避策として、ホストに接続されたセカンダリーネットワークインターフェイスを使用するか、OpenShift SDN デフォルト CNI プロバイダーに切り替えることができます。

19.3.2. ノード上でのインターフェイスの作成

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上にインターフェイスを作成します。マニフェストには、インターフェイスの要求された設定の詳細が含まれます。

デフォルトでは、マニフェストはクラスター内のすべてのノードに適用されます。インターフェイスを特定ノードに追加するには、ノードセクターの **spec: nodeSelector** パラメーターおよび適切な **<key>:<value>** を追加します。

手順

1. **NodeNetworkConfigurationPolicy** マニフェストを作成します。以下の例は、すべてのワーカーノードで Linux ブリッジを設定します。

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> ❶
spec:
  nodeSelector: ❷
    node-role.kubernetes.io/worker: "" ❸
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port ❹
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
        port:
          - name: eth1
```

- ❶ ポリシーの名前。
- ❷ オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- ❸ この例では **node-role.kubernetes.io/worker: ""** ノードセクターを使用し、クラスター内のすべてのワーカーノードを選択します。
- ❹ オプション: インターフェイスの人間が判読できる説明。

2. ノードのネットワークポリシーを作成します。

```
$ oc apply -f <br1-eth1-policy.yaml> ❶
```

- ❶ ノードネットワーク設定ポリシーマニフェストのファイル名。

関連情報

- [同じポリシーで複数のインターフェイスを作成する例](#)
- [ポリシーの各種 IP の管理方法の例](#)

19.3.3. ノード上でのノードネットワークポリシー更新の確認

NodeNetworkConfigurationPolicy マニフェストは、クラスターのノードについて要求されるネットワーク設定を記述します。ノードネットワークポリシーには、要求されたネットワーク設定と、クラスター全体でのポリシーの実行ステータスが含まれます。

ノードネットワークポリシーを適用する際に、**NodeNetworkConfigurationEnactment** オブジェクトがクラスター内のすべてのノードについて作成されます。ノードネットワーク設定の enactment (実行) は、そのノードでのポリシーの実行ステータスを表す読み取り専用オブジェクトです。ポリシーがノードに適用されない場合、そのノードの enactment (実行) にはトラブルシューティングのためのトレースバックが含まれます。

手順

1. ポリシーがクラスターに適用されていることを確認するには、ポリシーとそのステータスを一覧表示します。

```
$ oc get nncp
```

2. オプション: ポリシーの設定に想定されている以上の時間がかかる場合は、特定のポリシーの要求される状態とステータスの状態を検査できます。

```
$ oc get nncp <policy> -o yaml
```

3. オプション: ポリシーのすべてのノード上での設定に想定されている以上の時間がかかる場合は、クラスターの enactment (実行) のステータスを一覧表示できます。

```
$ oc get nnce
```

4. オプション: 特定の enactment (実行) の設定 (失敗した設定のエラーレポートを含む) を表示するには、以下を実行します。

```
$ oc get nnce <node>.<policy> -o yaml
```

19.3.4. ノードからインターフェイスの削除

NodeNetworkConfigurationPolicy オブジェクトを編集し、インターフェイスの **state** を **absent** に設定して、クラスターの1つ以上のノードからインターフェイスを削除できます。

ノードからインターフェイスを削除しても、ノードのネットワーク設定は以前の状態に自動的に復元されません。以前の状態に復元する場合、そのノードのネットワーク設定をポリシーで定義する必要があります。

ブリッジまたはボンディングインターフェイスを削除すると、そのブリッジまたはボンディングインターフェイスに以前に接続されたか、またはそれらの下位にあるノード NIC は **down** の状態になり、到達できなくなります。接続が失われないようにするには、同じポリシーでノード NIC を設定し、ステータスを **up** にし、DHCP または静的 IP アドレスのいずれかになるようにします。



注記

インターフェイスを追加したポリシーを削除しても、ノード上のポリシーの設定は変更されません。**NodeNetworkConfigurationPolicy** はクラスターのオブジェクトですが、これは要求される設定のみを表します。同様に、インターフェイスを削除してもポリシーは削除されません。

手順

1. インターフェイスの作成に使用する **NodeNetworkConfigurationPolicy** マニフェストを更新します。以下の例は Linux ブリッジを削除し、接続が失われないように DHCP で **eth1** NIC を設定します。

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> ❶
spec:
  nodeSelector: ❷
    node-role.kubernetes.io/worker: "" ❸
  desiredState:
    interfaces:
      - name: br1
        type: linux-bridge
        state: absent ❹
      - name: eth1 ❺
        type: ethernet ❻
        state: up ❼
        ipv4:
          dhcp: true ❽
          enabled: true ❾
```

- ❶ ポリシーの名前。
 - ❷ オプション: **nodeSelector** パラメータを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
 - ❸ この例では **node-role.kubernetes.io/worker: ""** ノードセレクターを使用し、クラスター内のすべてのワーカーノードを選択します。
 - ❹ 状態を **absent** に変更すると、インターフェイスが削除されます。
 - ❺ ブリッジインターフェイスから接続が解除されるインターフェイスの名前。
 - ❻ インターフェイスのタイプ。この例では、イーサネットネットワークインターフェイスを作成します。
 - ❼ インターフェイスの要求された状態。
 - ❽ オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
 - ❾ この例では **ipv4** を有効にします。
2. ノード上でポリシーを更新し、インターフェイスを削除します。

```
$ oc apply -f <br1-eth1-policy.yaml> ❶
```

- ❶ ポリシーマニフェストのファイル名。

19.3.5. 異なるインターフェイスのポリシー設定の例

19.3.5.1. 例: Linux ブリッジインターフェイスノードネットワーク設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上に Linux ブリッジインターフェイスを作成します。

以下の YAML ファイルは、Linux ブリッジインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy ❶
spec:
  nodeSelector: ❷
    kubernetes.io/hostname: <node01> ❸
  desiredState:
    interfaces:
      - name: br1 ❹
        description: Linux bridge with eth1 as a port ❺
        type: linux-bridge ❻
        state: up ❼
        ipv4:
          dhcp: true ❽
          enabled: true ❾
        bridge:
          options:
            stp:
              enabled: false ❿
        port:
          - name: eth1 ⓫
```

- ❶ ポリシーの名前。
- ❷ オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- ❸ この例では、**hostname** ノードセレクターを使用します。
- ❹ インターフェイスの名前。
- ❺ オプション: 人間が判読できるインターフェイスの説明。
- ❻ インターフェイスのタイプ。この例では、ブリッジを作成します。
- ❼ 作成後のインターフェイスの要求された状態。
- ❽ オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
- ❾ この例では **ipv4** を有効にします。
- ❿ この例では **stp** を無効にします。

- 11 ブリッジが接続されるノードの NIC。

19.3.5.2. 例: VLAN インターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上に VLAN インターフェイスを作成します。

以下の YAML ファイルは、VLAN インターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan-eth1-policy 1
spec:
  nodeSelector: 2
  kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: eth1.102 4
      description: VLAN using eth1 5
      type: vlan 6
      state: up 7
      vlan:
        base-iface: eth1 8
        id: 102 9
```

- 1 ポリシーの名前。
- 2 オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- 3 この例では、**hostname** ノードセレクターを使用します。
- 4 インターフェイスの名前。
- 5 オプション: 人間が判読できるインターフェイスの説明。
- 6 インターフェイスのタイプ。以下の例では VLAN を作成します。
- 7 作成後のインターフェイスの要求された状態。
- 8 VLAN が接続されているノードの NIC。
- 9 VLAN タグ。

19.3.5.3. 例: ボンドインターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してノード上にボンドインターフェイスを作成します。



注記

OpenShift Container Platform は以下の bond モードのみをサポートします。

- mode=1 active-backup
- mode=2 balance-xor
- mode=4 802.3ad
- mode=5 balance-tlb
- mode=6 balance-alb

以下の YAML ファイルは、ボンドインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```

apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy ❶
spec:
  nodeSelector: ❷
    kubernetes.io/hostname: <node01> ❸
  desiredState:
    interfaces:
    - name: bond0 ❹
      description: Bond with ports eth1 and eth2 ❺
      type: bond ❻
      state: up ❼
      ipv4:
        dhcp: true ❽
        enabled: true ❾
      link-aggregation:
        mode: active-backup ❿
      options:
        miimon: '140' ㉑
      port: ㉒
      - eth1
      - eth2
      mtu: 1450 ㉓

```

- ❶ ポリシーの名前。
- ❷ オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- ❸ この例では、**hostname** ノードセレクターを使用します。
- ❹ インターフェイスの名前。
- ❺ オプション: 人間が判読できるインターフェイスの説明。
- ❻ インターフェイスのタイプ。この例では、ボンドを作成します。

- 7 作成後のインターフェイスの要求された状態。
- 8 オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
- 9 この例では **ipv4** を有効にします。
- 10 ボンドのドライバーモード。この例では、アクティブなバックアップモードを使用します。
- 11 オプション: この例では、miimon を使用して 140ms ごとにボンドリンクを検査します。
- 12 ボンド内の下位ノードの NIC。
- 13 オプション: ボンドの Maximum transmission unit (MTU) 指定がない場合、この値はデフォルトで **1500** に設定されます。

19.3.5.4. 例: イーサネットインターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノードにイーサネットインターフェイスを作成します。

以下の YAML ファイルは、イーサネットインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: eth1 4
        description: Configuring eth1 on node01 5
        type: ethernet 6
        state: up 7
        ipv4:
          dhcp: true 8
          enabled: true 9
```

- 1 ポリシーの名前。
- 2 オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- 3 この例では、**hostname** ノードセレクターを使用します。
- 4 インターフェイスの名前。
- 5 オプション: 人間が判読できるインターフェイスの説明。
- 6 インターフェイスのタイプ。この例では、イーサネットネットワークインターフェイスを作成します。

- 7 作成後のインターフェースの要求された状態。
- 8 オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェースを出ることができます。
- 9 この例では **ipv4** を有効にします。

19.3.5.5. 例: 同じノードネットワーク設定ポリシーでの複数のインターフェース

同じノードネットワーク設定ポリシーで複数のインターフェースを作成できます。これらのインターフェースは相互に参照でき、単一のポリシーマニフェストを使用してネットワーク設定をビルドし、デプロイできます。

以下のスニペット例では、2つの NIC 間に **bond10** という名前のボンドと、ボンドに接続する **br1** という名前の Linux ブリッジを作成します。

```
#...
  interfaces:
  - name: bond10
    description: Bonding eth2 and eth3 for Linux bridge
    type: bond
    state: up
    link-aggregation:
      port:
      - eth2
      - eth3
  - name: br1
    description: Linux bridge on bond
    type: linux-bridge
    state: up
    bridge:
      port:
      - name: bond10
#...
```

19.3.6. 例: IP 管理

以下の設定スニペットの例は、さまざまな IP 管理方法を示しています。

これらの例では、**ethernet** インターフェイスタイプを使用して、ポリシー設定に関連するコンテキストを表示しつつ、サンプルを単純化します。これらの IP 管理のサンプルは、他のインターフェイスタイプでも使用できます。

19.3.6.1. 静的

以下のスニペットは、イーサネットインターフェースで IP アドレスを静的に設定します。

```
...
  interfaces:
  - name: eth1
    description: static IP on eth1
    type: ethernet
    state: up
    ipv4:
```

```
dhcp: false
address:
- ip: 192.168.122.250 1
  prefix-length: 24
enabled: true
```

...

- 1** この値を、インターフェイスの静的 IP アドレスに置き換えます。

19.3.6.2. IP アドレスなし

以下のスニペットでは、インターフェイスに IP アドレスがないことを確認できます。

```
...
interfaces:
- name: eth1
  description: No IP on eth1
  type: ethernet
  state: up
  ipv4:
    enabled: false
```

...

19.3.6.3. 動的ホストの設定

以下のスニペットは、動的 IP アドレス、ゲートウェイアドレス、および DNS を使用するイーサネットインターフェイスを設定します。

```
...
interfaces:
- name: eth1
  description: DHCP on eth1
  type: ethernet
  state: up
  ipv4:
    dhcp: true
    enabled: true
```

...

以下のスニペットは、動的 IP アドレスを使用しますが、動的ゲートウェイアドレスまたは DNS を使用しないイーサネットインターフェイスを設定します。

```
...
interfaces:
- name: eth1
  description: DHCP without gateway or DNS on eth1
  type: ethernet
  state: up
  ipv4:
    dhcp: true
    auto-gateway: false
```

```

auto-dns: false
enabled: true

```

```
...
```

19.3.6.4. DNS

以下のスニペットは、ホストに DNS 設定を設定します。

```

...
interfaces:
  ...
  dns-resolver:
    config:
      search:
        - example.com
        - example.org
      server:
        - 8.8.8.8
  ...

```

19.3.6.5. 静的ルーティング

以下のスニペットは、インターフェイス **eth1** に静的ルートおよび静的 IP を設定します。

```

...
interfaces:
  - name: eth1
    description: Static routing on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: false
      address:
        - ip: 192.0.2.251 ①
          prefix-length: 24
      enabled: true
    routes:
      config:
        - destination: 198.51.100.0/24
          metric: 150
          next-hop-address: 192.0.2.1 ②
          next-hop-interface: eth1
          table-id: 254
  ...

```

① イーサネットインターフェイスの静的 IP アドレス。

② ノードトラフィックのネクストホップアドレス。これは、イーサネットインターフェイスに設定される IP アドレスと同じサブネットにある必要があります。

19.4. ノードのネットワーク設定のトラブルシューティング

ノードのネットワーク設定で問題が発生した場合には、ポリシーが自動的にロールバックされ、enactment (実行) レポートは失敗します。これには、以下のような問題が含まれます。

- ホストで設定を適用できません。
- ホストはデフォルトゲートウェイへの接続を失います。
- ホストは API サーバーへの接続を失います。

19.4.1. 正確でないノードネットワーク設定のポリシー設定のトラブルシューティング

ノードネットワーク設定ポリシーを適用し、クラスター全体でノードのネットワーク設定への変更を適用することができます。正確でない設定を適用する場合、以下の例を使用して、失敗したノードネットワークポリシーのトラブルシューティングと修正を行うことができます。

この例では、Linux ブリッジポリシーは、3つのコントロールプレーンノード (マスター) と3つのコンピュート (ワーカー) ノードを持つクラスターのサンプルに適用されます。ポリシーは正しくないインターフェイスを参照するために、適用することができません。エラーを確認するには、利用可能な NMState リソースを調べます。その後、正しい設定でポリシーを更新できます。

手順

1. ポリシーを作成し、これをクラスターに適用します。以下の例では、**ens01** インターフェイスに単純なブリッジを作成します。

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with the wrong port
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
        port:
          - name: ens01
```

```
$ oc apply -f ens01-bridge-testfail.yaml
```

出力例

```
nodenetworkconfigurationpolicy.nmstate.io/ens01-bridge-testfail created
```

2. 以下のコマンドを実行してポリシーのステータスを確認します。

```
$ oc get nncp
```

この出力は、ポリシーが失敗したことを示しています。

出力例

```
NAME          STATUS
ens01-bridge-testfail FailedToConfigure
```

ただし、ポリシーのステータスのみでは、すべてのノードで失敗したか、またはノードのサブセットで失敗したかを確認することはできません。

3. ノードのネットワーク設定の enactment (実行) を一覧表示し、ポリシーがいずれかのノードで成功したかどうかを確認します。このポリシーがノードのサブセットに対してのみ失敗した場合は、問題が特定のノード設定にあることが示唆されます。このポリシーがすべてのノードで失敗した場合には、問題はポリシーに関連するものであることが示唆されます。

```
$ oc get nnce
```

この出力は、ポリシーがすべてのノードで失敗したことを示しています。

出力例

```
NAME          STATUS
control-plane-1.ens01-bridge-testfail FailedToConfigure
control-plane-2.ens01-bridge-testfail FailedToConfigure
control-plane-3.ens01-bridge-testfail FailedToConfigure
compute-1.ens01-bridge-testfail FailedToConfigure
compute-2.ens01-bridge-testfail FailedToConfigure
compute-3.ens01-bridge-testfail FailedToConfigure
```

4. 失敗した enactment (実行) のいずれかを表示し、トレースバックを確認します。以下のコマンドは、出力ツール **jsonpath** を使用して出力をフィルターします。

```
$ oc get nnce compute-1.ens01-bridge-testfail -o jsonpath='{.status.conditions[?(@.type=="Failing")].message}'
```

このコマンドは、簡潔にするために編集されている大きなトレースバックを返します。

出力例

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to execute
nmstatectl set --no-commit --timeout 480: 'exit status 1' "
...
libnmstate.error.NmstateVerificationError:
desired
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
```

```
mac-ageing-time: 300
multicast-snooping: true
stp:
  enabled: false
  forward-delay: 15
  hello-time: 2
  max-age: 20
  priority: 32768
port:
- name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

current
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
      priority: 32768
  port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

difference
=====
```



```

--- desired
+++ current
@@ -13,8 +13,7 @@
    hello-time: 2
    max-age: 20
    priority: 32768
- port:
- - name: ens01
+ port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
line 651, in _assert_interfaces_equal\n
current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:

```

NmstateVerificationError は、**desired** ポリシー設定、ノード上のポリシーの **current** 設定、および一致しないパラメーターを強調表示する **difference** を一覧表示します。この例では、**port** は **difference** に組み込まれ、これは問題がポリシーのポート設定に関連するものであることを示唆します。

5. ポリシーが適切に設定されていることを確認するには、**NodeNetworkState** オブジェクトを要求して、1つまたはすべてのノードのネットワーク設定を表示します。以下のコマンドは、**control-plane-1** ノードのネットワーク設定を返します。

```
$ oc get nns control-plane-1 -o yaml
```

出力は、ノード上のインターフェイス名は **ens1** であるものの、失敗したポリシーが **ens01** を誤って使用していることを示します。

出力例

```

- ipv4:
...
  name: ens1
  state: up
  type: ethernet

```

6. 既存のポリシーを編集してエラーを修正します。

```
$ oc edit nncp ens01-bridge-testfail
```

```

...
  port:
    - name: ens1

```

ポリシーを保存して修正を適用します。

7. ポリシーのステータスをチェックして、更新が正常に行われたことを確認します。

```
$ oc get nncp
```

出力例

NAME	STATUS
ens01-bridge-testfail	SuccessfullyConfigured

更新されたポリシーは、クラスターのすべてのノードで正常に設定されました。

第20章 クラスター全体のプロキシの設定

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシを使用することができます。既存クラスターのプロキシオブジェクトを変更するか、または新規クラスターの `install-config.yaml` ファイルでプロキシ設定を行うことにより、OpenShift Container Platform をプロキシを使用するように設定できます。

20.1. 前提条件

- [クラスターがアクセスする必要のあるサイト](#)を確認し、プロキシをバイパスする必要があるかどうかを判断します。デフォルトで、すべてのクラスターシステムの egress トラフィック (クラスターをホストするクラウドのクラウドプロバイダー API に対する呼び出しを含む) はプロキシされます。システム全体のプロキシは、ユーザーのワークロードではなく、システムコンポーネントにのみ影響を与えます。プロキシオブジェクトの `spec.noProxy` フィールドにサイトを追加し、必要に応じてプロキシをバイパスします。



注記

Proxy オブジェクトの `status.noProxy` フィールドには、インストール設定の `networking.machineNetwork[].cidr`、`networking.clusterNetwork[].cidr`、および `networking.serviceNetwork[]` フィールドの値が設定されます。

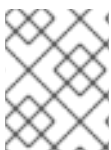
Amazon Web Services (AWS)、Google Cloud Platform (GCP)、Microsoft Azure、および Red Hat OpenStack Platform (RHOSP) へのインストールの場合、Proxy オブジェクトの `status.noProxy` フィールドには、インスタンスメタデータのエンドポイント (`169.254.169.254`) も設定されます。

20.2. クラスター全体のプロキシの有効化

Proxy オブジェクトは、クラスター全体の egress プロキシを管理するために使用されます。プロキシを設定せずにクラスターがインストールまたはアップグレードされると、Proxy オブジェクトは引き続き生成されますが、`spec` は設定されません。以下に例を示します。

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

クラスター管理者は、この `cluster Proxy` オブジェクトを変更して OpenShift Container Platform のプロキシを設定できます。



注記

`cluster` という名前の Proxy オブジェクトのみがサポートされ、追加のプロキシを作成することはできません。

前提条件

- クラスター管理者のパーミッション。

- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. HTTPS 接続のプロキシに必要な追加の CA 証明書が含まれる config map を作成します。



注記

プロキシのアイデンティティ証明書が RHCOS 信頼バンドルからの認証局によって署名される場合は、これを省略できます。

- a. 以下の内容で **user-ca-bundle.yaml** というファイルを作成して、PEM でエンコードされた証明書の値を指定します。

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

- 1** このデータキーは **ca-bundle.crt** という名前にする必要があります。
- 2** プロキシのアイデンティティ証明書に署名するために使用される1つ以上の PEM でエンコードされた X.509 証明書。
- 3** **Proxy** オブジェクトから参照される config map 名。
- 4** config map は **openshift-config** namespace になければなりません。

- b. このファイルから ConfigMap を作成します。

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

2. **oc edit** コマンドを使用して **Proxy** オブジェクトを変更します。

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

3. プロキシに必要なフィールドを設定します。

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

```
- https://www.google.com
```

```
trustedCA:
```

```
name: user-ca-bundle 5
```

- 1** クラスター外の HTTP 接続を作成するために使用するプロキシ URL。URL スキームは **http** である必要があります。
- 2** クラスター外で HTTPS 接続を作成するために使用するプロキシ URL。URL スキームは **http** または **https** である必要があります。URL スキームをサポートするプロキシの URL を指定します。たとえば、ほとんどのプロキシは、**https** を使用するように設定されていても、**http** しかサポートしていない場合、エラーを報告します。このエラーメッセージはログに反映されず、代わりにネットワーク接続エラーのように見える場合があります。クラスターからの **https** 接続をリッスンするプロキシを使用している場合は、プロキシが使用する CA と証明書を受け入れるようにクラスターを設定する必要がある場合があります。
- 3** プロキシを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のコンマ区切りの一覧。

サブドメインのみと一致するように、ドメインの前に **.** を付けます。たとえば、**.y.com** は **x.y.com** に一致しますが、**y.com** には一致しません。***** を使用し、すべての宛先のプロキシをバイパスします。インストール設定で **networking.machineNetwork[].cidr** フィールドで定義されるネットワークに含まれていないワーカーをスケールアップする場合、それらをこの一覧に追加し、接続の問題を防ぐ必要があります。

httpProxy または **httpsProxy** フィールドのいずれも設定されていない場合に、このフィールドは無視されます。

- 4** **httpProxy** および **httpsProxy** の値をステータスに書き込む前の readiness チェックに使用するクラスター外の 1 つ以上の URL。
- 5** HTTPS 接続のプロキシに必要な追加の CA 証明書が含まれる、**openshift-config** namespace の config map の参照。ここで参照する前に config map が存在する必要があります。このフィールドは、プロキシのアイデンティティ証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

4. 変更を適用するためにファイルを保存します。

20.3. クラスター全体のプロキシの削除

cluster プロキシオブジェクトは削除できません。クラスターからプロキシを削除するには、プロキシオブジェクトからすべての **spec** フィールドを削除します。

前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform **oc** CLI ツールがインストールされている。

手順

1. **oc edit** コマンドを使用してプロキシを変更します。

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

2. プロキシオブジェクトからすべての **spec** フィールドを削除します。以下に例を示します。

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

3. 変更を適用するためにファイルを保存します。

関連情報

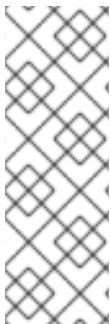
- [CA バンドル証明書の置き換え](#)
- [プロキシ証明書のカスタマイズ](#)

第21章 カスタム PKI の設定

Web コンソールなどの一部のプラットフォームコンポーネントは、通信にルートを使用し、それらと対話するために他のコンポーネントの証明書を信頼する必要があります。カスタムのパブリックキーインフラストラクチャー (PKI) を使用している場合は、プライベートに署名された CA 証明書がクラスター全体で認識されるようにこれを設定する必要があります。

プロキシ API を使用して、クラスター全体で信頼される CA 証明書を追加できます。インストール時またはランタイム時にこれを実行する必要があります。

- **インストール** 時に、**クラスター全体のプロキシを設定します**。プライベートに署名された CA 証明書は、**install-config.yaml** ファイルの **additionalTrustBundle** 設定で定義する必要があります。インストールプログラムは、定義した追加の CA 証明書が含まれる **user-ca-bundle** という名前の ConfigMap を生成します。次に Cluster Network Operator は、これらの CA 証明書を Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージする **trusted-ca-bundle** ConfigMap を作成し、この ConfigMap はプロキシオブジェクトの **trustedCA** フィールドで参照されます。
- **ランタイム** 時に、**デフォルトのプロキシオブジェクトを変更して、プライベートに署名された CA 証明書を追加** します (これは、クラスターのプロキシ有効化のワークフローの一部です)。これには、クラスターで信頼される必要があるプライベートに署名された CA 証明書が含まれる ConfigMap を作成し、次にプライベートに署名された証明書の ConfigMap を参照する **trustedCA** でプロキシリソースを変更することが関係します。



注記

インストーラー設定の **additionalTrustBundle** フィールドおよびプロキシリソースの **trustedCA** フィールドは、クラスター全体の信頼バンドルを管理するために使用されます。**additionalTrustBundle** はインストール時に使用され、プロキシの **trustedCA** がランタイム時に使用されます。

trustedCA フィールドは、クラスターコンポーネントによって使用されるカスタム証明書とキーのペアを含む **ConfigMap** の参照です。

21.1. インストール時のクラスター全体のプロキシの設定

実稼働環境では、インターネットへの直接アクセスを拒否し、代わりに HTTP または HTTPS プロキシを使用することができます。プロキシ設定を **install-config.yaml** ファイルで行うことにより、新規の OpenShift Container Platform クラスターをプロキシを使用するように設定できます。

前提条件

- 既存の **install-config.yaml** ファイルがある。
- クラスターがアクセスする必要のあるサイトを確認済みで、それらのいずれかがプロキシをバイパスする必要があるかどうかを判別している。デフォルトで、すべてのクラスター egress トラフィック (クラスターをホストするクラウドについてのクラウドプロバイダー API に対する呼び出しを含む) はプロキシされます。プロキシを必要に応じてバイパスするために、サイトを **Proxy** オブジェクトの **spec.noProxy** フィールドに追加している。



注記

Proxy オブジェクトの **status.noProxy** フィールドには、インストール設定の **networking.machineNetwork[].cidr**、**networking.clusterNetwork[].cidr**、および **networking.serviceNetwork[]** フィールドの値が設定されます。

Amazon Web Services (AWS)、Google Cloud Platform (GCP)、Microsoft Azure、および Red Hat OpenStack Platform (RHOSP) へのインストールの場合、**Proxy** オブジェクトの **status.noProxy** フィールドには、インスタンスメタデータのエンドポイント (**169.254.169.254**) も設定されます。

手順

1. **install-config.yaml** ファイルを編集し、プロキシ設定を追加します。以下に例を示します。

```
apiVersion: v1
baseDomain: my.domain.com
proxy:
  httpProxy: http://<username>:<pswd>@<ip>:<port> 1
  httpsProxy: https://<username>:<pswd>@<ip>:<port> 2
  noProxy: example.com 3
additionalTrustBundle: | 4
  -----BEGIN CERTIFICATE-----
  <MY_TRUSTED_CA_CERT>
  -----END CERTIFICATE-----
...
```

- 1 クラスタ外の HTTP 接続を作成するために使用するプロキシ URL。URL スキームは **http** である必要があります。
- 2 クラスタ外で HTTPS 接続を作成するために使用するプロキシ URL。
- 3 プロキシから除外するための宛先ドメイン名、IP アドレス、または他のネットワーク CIDR のコンマ区切りの一覧。サブドメインのみと一致するように、ドメインの前に **.** を付けます。たとえば、**.y.com** は **x.y.com** に一致しますが、**y.com** には一致しません。***** を使用し、すべての宛先のプロキシをバイパスします。
- 4 指定されている場合には、インストールプログラムは、**openshift-config** namespace に **user-ca-bundle** という名前の設定魔府を生成して、追加の CA 証明書を保存します。**additionalTrustBundle** と少なくとも1つのプロキシ設定を指定した場合には、**Proxy** オブジェクトは **trusted CA** フィールドで **user-ca-bundle** 設定マップを参照するように設定されます。その後、Cluster Network Operator は、**trustedCA** パラメーターに指定されたコンテンツを RHCOS トラストバンドルにマージする **trusted-ca-bundle** 設定マップを作成します。**additionalTrustBundle** フィールドは、プロキシのアイデンティティ証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

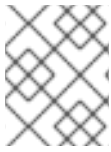


注記

インストールプログラムは、プロキシの **readinessEndpoints** フィールドをサポートしません。

2. ファイルを保存し、OpenShift Container Platform のインストール時にこれを参照します。

インストールプログラムは、指定の `install-config.yaml` ファイルのプロキシ設定を使用する `cluster` という名前のクラスター全体のプロキシを作成します。プロキシ設定が指定されていない場合、`cluster Proxy` オブジェクトが依然として作成されますが、これには `spec` がありません。



注記

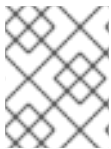
`cluster` という名前の `Proxy` オブジェクトのみがサポートされ、追加のプロキシを作成することはできません。

21.2. クラスター全体のプロキシの有効化

`Proxy` オブジェクトは、クラスター全体の egress プロキシを管理するために使用されます。プロキシを設定せずにクラスターがインストールまたはアップグレードされると、`Proxy` オブジェクトは引き続き生成されますが、`spec` は設定されません。以下に例を示します。

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

クラスター管理者は、この `cluster Proxy` オブジェクトを変更して OpenShift Container Platform のプロキシを設定できます。



注記

`cluster` という名前の `Proxy` オブジェクトのみがサポートされ、追加のプロキシを作成することはできません。

前提条件

- クラスター管理者のパーミッション。
- OpenShift Container Platform `oc` CLI ツールがインストールされている。

手順

1. HTTPS 接続のプロキシに必要な追加の CA 証明書が含まれる config map を作成します。



注記

プロキシのアイデンティティ証明書が RHCOS 信頼バンドルからの認証局によって署名される場合は、これを省略できます。

- a. 以下の内容で `user-ca-bundle.yaml` というファイルを作成して、PEM でエンコードされた証明書の値を指定します。

```
apiVersion: v1
data:
  ca-bundle.crt: | 1
```

```
<MY_PEM_ENCODED_CERTS> 2
kind: ConfigMap
metadata:
  name: user-ca-bundle 3
  namespace: openshift-config 4
```

- 1 このデータキーは **ca-bundle.crt** という名前にする必要があります。
- 2 プロキシのアイデンティティ証明書に署名するために使用される1つ以上の PEM でエンコードされた X.509 証明書。
- 3 **Proxy** オブジェクトから参照される config map 名。
- 4 config map は **openshift-config** namespace になければなりません。

b. このファイルから ConfigMap を作成します。

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

2. **oc edit** コマンドを使用して **Proxy** オブジェクトを変更します。

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

3. プロキシに必要なフィールドを設定します。

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

- 1 クラスタ外の HTTP 接続を作成するために使用するプロキシ URL。URL スキームは **http** である必要があります。
- 2 クラスタ外で HTTPS 接続を作成するために使用するプロキシ URL。URL スキームは **http** または **https** である必要があります。URL スキームをサポートするプロキシの URL を指定します。たとえば、ほとんどのプロキシは、**https** を使用するように設定されていても、**http** しかサポートしていない場合、エラーを報告します。このエラーメッセージはログに反映されず、代わりにネットワーク接続エラーのように見える場合があります。クラスタからの **https** 接続をリッスンするプロキシを使用している場合は、プロキシが使用する CA と証明書を受け入れるようにクラスタを設定する必要がある場合があります。
- 3 プロキシを除外するための宛先ドメイン名、ドメイン、IP アドレス、または他のネットワーク CIDR のコンマ区切りの一覧。

サブドメインのみと一致するように、ドメインの前に `.` を付けます。たとえば、`.y.com` は `x.y.com` に一致しますが、`y.com` には一致しません。`*` を使用し、すべての宛先のプロキシをバイパスします。インストール設定で `networking.machineNetwork[].cidr` フィールドで定義されるネットワークに含まれていないワーカーをスケールアップする場合、それらをこの一覧に追加し、接続の問題を防ぐ必要があります。

`httpProxy` または `httpsProxy` フィールドのいずれも設定されていない場合に、このフィールドは無視されます。

- 4 `httpProxy` および `httpsProxy` の値をステータスに書き込む前の readiness チェックに使用するクラスター外の1つ以上の URL。
- 5 HTTPS 接続のプロキシに必要な追加の CA 証明書が含まれる、`openshift-config` namespace の config map の参照。ここで参照する前に config map が存在する必要があります。このフィールドは、プロキシのアイデンティティ証明書が RHCOS 信頼バンドルからの認証局によって署名されない限り必要になります。

4. 変更を適用するためにファイルを保存します。

21.3. OPERATOR を使用した証明書の挿入

カスタム CA 証明書が ConfigMap 経由でクラスターに追加されると、Cluster Network Operator はユーザーによってプロビジョニングされる CA 証明書およびシステム CA 証明書を単一バンドルにマージし、信頼バンドルの挿入を要求する Operator にマージされたバンドルを挿入します。

Operator は、以下のラベルの付いた空の ConfigMap を作成してこの挿入を要求します。

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

空の ConfigMap の例:

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

- ① 空の ConfigMap 名を指定します。

Operator は、この ConfigMap をコンテナのローカル信頼ストアにマウントします。



注記

信頼された CA 証明書の追加は、証明書が Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルに含まれない場合にのみ必要になります。

証明書の挿入は Operator に制限されません。Cluster Network Operator は、空の ConfigMap が `config.openshift.io/inject-trusted-cabundle=true` ラベルを使用して作成される場合に、すべての namespace で証明書を挿入できます。

ConfigMap はすべての namespace に置くことができますが、ConfigMap はカスタム CA を必要とする Pod 内の各コンテナに対してボリュームとしてマウントされる必要があります。以下に例を示します。

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

① **ca-bundle.crt** は ConfigMap キーとして必要になります。

② **tls-ca-bundle.pem** は ConfigMap パスとして必要になります。

第22章 RHOSP での負荷分散

22.1. KURYR SDN を使用した OCTAVIA OVN ロードバランサープロバイダードライバーの使用

OpenShift Container Platform クラスターが Kuryr を使用し、これが後に RHOSP 16 にアップグレードされた Red Hat OpenStack Platform (RHOSP) 13 クラウドにインストールされている場合、これを Octavia OVN プロバイダードライバーを使用するように設定できます。



重要

Kuryr はプロバイダードライバーの変更後に既存のロードバランサーを置き換えます。このプロセスにより、ダウンタイムが生じます。

前提条件

- RHOSP CLI の **openstack** をインストールします。
- OpenShift Container Platform CLI の **oc** をインストールします。
- RHOSP の Octavia OVN ドライバーが有効になっていることを確認している。

ヒント

利用可能な Octavia ドライバーの一覧を表示するには、コマンドラインで **openstack loadbalancer provider list** を入力します。

ovn ドライバーはコマンドの出力に表示されます。

手順

Octavia Amphora プロバイダードライバーから Octavia OVN に変更するには、以下を実行します。

1. **kuryr-config** ConfigMap を開きます。コマンドラインで、以下を実行します。

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

2. ConfigMap で、**kuryr-octavia-provider: default** が含まれる行を削除します。以下に例を示します。

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: default 1
...
```

- 1** この行を削除します。クラスターは、**ovn** を値としてこれを再生成します。

Cluster Network Operator が変更を検出し、**kuryr-controller** および **kuryr-cni** Pod を再デプロイするのを待機します。このプロセスには数分の時間がかかる可能性があります。

3. **kuryr-config** ConfigMap アノテーションで **ovn** をその値として表示されていることを確認します。コマンドラインで、以下を実行します。

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

ovn プロバイダーの値は出力に表示されます。

```
...
kind: ConfigMap
metadata:
  annotations:
    networkoperator.openshift.io/kuryr-octavia-provider: ovn
...
```

4. RHOSP がそのロードバランサーを再作成していることを確認します。

- a. コマンドラインで、以下を実行します。

```
$ openstack loadbalancer list | grep amphora
```

単一の Amphora ロードバランサーが表示されます。以下に例を示します。

```
a4db683b-2b7b-4988-a582-c39daaad7981 | ostest-7mbj6-kuryr-api-loadbalancer |
84c99c906edd475ba19478a9a6690efd | 172.30.0.1 | ACTIVE | amphora
```

- b. 以下を入力して **ovn** ロードバランサーを検索します。

```
$ openstack loadbalancer list | grep ovn
```

ovn タイプの残りのロードバランサーが表示されます。以下に例を示します。

```
2dffe783-98ae-4048-98d0-32aa684664cc | openshift-apiserver-operator/metrics |
84c99c906edd475ba19478a9a6690efd | 172.30.167.119 | ACTIVE | ovn
0b1b2193-251f-4243-af39-2f99b29d18c5 | openshift-etcd/etcd |
84c99c906edd475ba19478a9a6690efd | 172.30.143.226 | ACTIVE | ovn
f05b07fc-01b7-4673-bd4d-adaa4391458e | openshift-dns-operator/metrics |
84c99c906edd475ba19478a9a6690efd | 172.30.152.27 | ACTIVE | ovn
```

22.2. OCTAVIA を使用したアプリケーショントラフィック用のクラスターのスケールリング

Red Hat OpenStack Platform (RHOSP) で実行される OpenShift Container Platform クラスターでは、Octavia 負荷分散サービスを使用して、複数の仮想マシン (VM) または Floating IP アドレスにトラフィックを分散することができます。この機能は、単一マシンまたはアドレスが生じさせるボトルネックを軽減します。

クラスターで Kuryr を使用する場合、Cluster Network Operator はデプロイメント時に内部 Octavia ロードバランサーを作成していました。アプリケーションネットワークのスケールリングには、このロードバランサーを使用できます。

クラスターで Kuryr を使用しない場合、アプリケーションのネットワークのスケールリングに使用する独自の Octavia ロードバランサーを作成する必要があります。

22.2.1. Octavia を使用したクラスターのスケールリング

複数の API ロードバランサーを使用する場合や、クラスターが Kuryr を使用しない場合、Octavia ロードバランサーを作成してから、クラスターをこれを使用するように設定します。

前提条件

- Octavia は Red Hat OpenStack Platform (RHOSP) デプロイメントで利用できます。

手順

1. コマンドラインから、Amphora ドライバーを使用する Octavia ロードバランサーを作成します。

```
$ openstack loadbalancer create --name API_OCP_CLUSTER --vip-subnet-id
<id_of_worker_vms_subnet>
```

API_OCP_CLUSTER の代わりに、任意の名前を使用することができます。

2. ロードバランサーがアクティブになったら、リスナーを作成します。

```
$ openstack loadbalancer listener create --name API_OCP_CLUSTER_6443 --protocol
HTTPS--protocol-port 6443 API_OCP_CLUSTER
```



注記

ロードバランサーのステータスを表示するには、**openstack loadbalancer list** と入力します。

3. ラウンドロビンアルゴリズムを使用し、セッションの永続性が有効にされているプールを作成します。

```
$ openstack loadbalancer pool create --name API_OCP_CLUSTER_pool_6443 --lb-
algorithm ROUND_ROBIN --session-persistence type=<source_IP_address> --listener
API_OCP_CLUSTER_6443 --protocol HTTPS
```

4. コントロールプレーンマシンが利用可能であることを確認するには、ヘルスマニターを作成します。

```
$ openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --timeout 10 --type
TCP API_OCP_CLUSTER_pool_6443
```

5. コントロールプレーンマシンをロードバランサープールのメンバーとして追加します。

```
$ for SERVER in $(MASTER-0-IP MASTER-1-IP MASTER-2-IP)
do
  openstack loadbalancer member create --address $SERVER --protocol-port 6443
  API_OCP_CLUSTER_pool_6443
done
```

6. オプション: クラスター API の Floating IP アドレスを再利用するには、設定を解除します。

```
$ openstack floating ip unset $API_FIP
```

- 設定を解除された **API_FIP**、または新規アドレスを、作成されたロードバランサー VIP に追加します。

```
$ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value API_OCP_CLUSTER) $API_FIP
```

クラスターは、負荷分散に Octavia を使用するようになりました。



注記

Kuryr が Octavia Amphora ドライバーを使用する場合、すべてのトラフィックは単一の Amphora 仮想マシン (VM) 経由でルーティングされます。

この手順を繰り返して追加のロードバランサーを作成します。これにより、ボトルネックを軽減することができます。

22.2.2. Octavia の使用による Kuryr を使用するクラスターのスケールリング

クラスターで Kuryr を使用する場合、クラスターの API Floating IP アドレスを既存の Octavia ロードバランサーに関連付けます。

前提条件

- OpenShift Container Platform クラスターは Kuryr を使用します。
- Octavia は Red Hat OpenStack Platform (RHOSP) デプロイメントで利用できます。

手順

- オプション: コマンドラインからクラスター API の Floating IP アドレスを再利用するには、この設定を解除します。

```
$ openstack floating ip unset $API_FIP
```

- 設定を解除された **API_FIP**、または新規アドレスを、作成されたロードバランサー VIP に追加します。

```
$ openstack floating ip set --port $(openstack loadbalancer show -c <vip_port_id> -f value ${OCP_CLUSTER}-kuryr-api-loadbalancer) $API_FIP
```

クラスターは、負荷分散に Octavia を使用するようになりました。



注記

Kuryr が Octavia Amphora ドライバーを使用する場合、すべてのトラフィックは単一の Amphora 仮想マシン (VM) 経由でルーティングされます。

この手順を繰り返して追加のロードバランサーを作成します。これにより、ボトルネックを軽減することができます。

22.3. RHOSP OCTAVIA を使用した INGRESS トラフィックのスケールリング

Octavia ロードバランサーを使用して、Kuryr を使用するクラスターで Ingress コントローラーをスケールリングできます。

前提条件

- OpenShift Container Platform クラスターは Kuryr を使用します。
- Octavia は RHOSP デプロイメントで利用できます。

手順

1. 現在の内部ルーターサービスをコピーするには、コマンドラインで以下を入力します。

```
$ oc -n openshift-ingress get svc router-internal-default -o yaml > external_router.yaml
```

2. **external_router.yaml** ファイルで、**metadata.name** および **spec.type** の値を **LoadBalancer** に変更します。

ルーターファイルの例

```
apiVersion: v1
kind: Service
metadata:
  labels:
    ingresscontroller.operator.openshift.io/owning-ingresscontroller: default
  name: router-external-default 1
  namespace: openshift-ingress
spec:
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    - name: https
      port: 443
      protocol: TCP
      targetPort: https
    - name: metrics
      port: 1936
      protocol: TCP
      targetPort: 1936
  selector:
    ingresscontroller.operator.openshift.io/deployment-ingresscontroller: default
  sessionAffinity: None
  type: LoadBalancer 2
```

1 この値は **router-external-default** のように記述的であることを確認します。

2 この値は **LoadBalancer** であることを確認します。



注記

ロードバランシングと関連性のないタイムスタンプやその他の情報を削除できます。

1. コマンドラインで、**external_router.yaml** ファイルからサービスを作成します。

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

2. サービスの外部 IP アドレスがロードバランサーに関連付けられているものと同じであることを確認します。

- a. コマンドラインで、サービスの外部 IP アドレスを取得します。

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

出力例

```
NAME                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)
AGE
router-external-default LoadBalancer  172.30.235.33  10.46.22.161
80:30112/TCP,443:32359/TCP,1936:30317/TCP 3m38s
router-internal-default ClusterIP     172.30.115.123 <none>
80/TCP,443/TCP,1936/TCP                22h
```

- b. ロードバランサーの IP アドレスを取得します。

```
$ openstack loadbalancer list | grep router-external
```

出力例

```
| 21bf6afe-b498-4a16-a958-3229e83c002c | openshift-ingress/router-external-default |
66f3816acf1b431691b8d132cc9d793c | 172.30.235.33 | ACTIVE | octavia |
```

- c. 直前のステップで取得したアドレスが、Floating IP の一覧で相互に関連付けられていることを確認します。

```
$ openstack floating ip list | grep 172.30.235.33
```

出力例

```
| e2f80e97-8266-4b69-8636-e58bacf1879e | 10.46.22.161 | 172.30.235.33 | 655e7122-
806a-4e0a-a104-220c6e17bda6 | a565e55a-99e7-4d15-b4df-f9d7ee8c9deb |
66f3816acf1b431691b8d132cc9d793c |
```

EXTERNAL-IP の値を新規 Ingress アドレスとして使用できるようになりました。



注記

Kuryr が Octavia Amphora ドライバーを使用する場合、すべてのトラフィックは単一の Amphora 仮想マシン (VM) 経由でルーティングされます。

この手順を繰り返して追加のロードバランサーを作成します。これにより、ボトルネックを軽減することができます。

22.4. 外部ロードバランサーの設定

Red Hat OpenStack Platform (RHOSP) の OpenShift Container Platform クラスターを、デフォルトのロードバランサーの代わりに外部ロードバランサーを使用するように設定できます。

前提条件

- ロードバランサーでは、システムの任意のユーザーが TCP をポート 6443、443、および 80 が利用できる必要があります。
- それぞれのコントロールプレーンノード間で API ポート 6443 を負荷分散します。
- すべてのコンピュートノード間でアプリケーションポート 443 と 80 を負荷分散します。
- ロードバランサーでは、Ignition 起動設定をノードに提供するために使用されるポート 22623 はクラスター外に公開されません。
- ロードバランサーはクラスター内のすべてのマシンにアクセスできる必要があります。このアクセスを許可する方法には、以下が含まれます。
 - ロードバランサーをクラスターのマシンのサブネットに割り当てます。
 - ロードバランサーを使用するマシンに Floating IP アドレスを割り当てます。



重要

外部の負荷分散サービスとコントロールプレーンノードは同じ L2 ネットワークで実行する必要があります。また、VLAN を使用して負荷分散サービスとコントロールプレーンノード間のトラフィックをルーティングする際に同じ VLAN で実行する必要があります。

手順

1. ポート 6443、443、および 80 でロードバランサーからクラスターへのアクセスを有効にします。たとえば、以下の HAProxy 設定に留意してください。

サンプル HAProxy 設定のセクション

```
...
listen my-cluster-api-6443
  bind 0.0.0.0:6443
  mode tcp
  balance roundrobin
  server my-cluster-master-2 192.0.2.2:6443 check
  server my-cluster-master-0 192.0.2.3:6443 check
  server my-cluster-master-1 192.0.2.1:6443 check
listen my-cluster-apps-443
  bind 0.0.0.0:443
  mode tcp
  balance roundrobin
  server my-cluster-worker-0 192.0.2.6:443 check
  server my-cluster-worker-1 192.0.2.5:443 check
  server my-cluster-worker-2 192.0.2.4:443 check
listen my-cluster-apps-80
  bind 0.0.0.0:80
  mode tcp
  balance roundrobin
```

```
server my-cluster-worker-0 192.0.2.7:80 check
server my-cluster-worker-1 192.0.2.9:80 check
server my-cluster-worker-2 192.0.2.8:80 check
```

- ロードバランサーでクラスター API およびアプリケーションの DNS サーバーにレコードを追加します。以下に例を示します。

```
<load_balancer_ip_address> api.<cluster_name>.<base_domain>
<load_balancer_ip_address> apps.<cluster_name>.<base_domain>
```

- コマンドラインで **curl** を使用して、外部ロードバランサーおよび DNS 設定が機能することを確認します。

- クラスター API がアクセス可能であることを確認します。

```
$ curl https://<loadbalancer_ip_address>:6443/version --insecure
```

設定が正しい場合は、応答として JSON オブジェクトを受信します。

```
{
  "major": "1",
  "minor": "11+",
  "gitVersion": "v1.11.0+ad103ed",
  "gitCommit": "ad103ed",
  "gitTreeState": "clean",
  "buildDate": "2019-01-09T06:44:10Z",
  "goVersion": "go1.10.3",
  "compiler": "gc",
  "platform": "linux/amd64"
}
```

- クラスターアプリケーションがアクセス可能であることを確認します。



注記

Web ブラウザーで OpenShift Container Platform コンソールを開き、アプリケーションのアクセスを確認することもできます。

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --insecure
```

設定が正しい場合は、HTTP 応答を受信します。

```
HTTP/1.1 302 Found
content-length: 0
location: https://console-openshift-console.apps.<cluster-name>.<base domain>/
cache-control: no-cacheHTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=39HoZgztDnzjJkq/JuLJMeoKNXIfiVv2YgZc09c3TBOBU4NI6kDXaJH1LdicNhN1UsQ
Wzon4Dor9GWGfopaTEQ==; Path=/; Secure
x-content-type-options: nosniff
x-dns-prefetch-control: off
```

```
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Tue, 17 Nov 2020 08:42:10 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=9b714eb87e93cf34853e87a92d6894be; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

第23章 METALLB を使用した負荷分散

23.1. METALLB および METALLB OPERATOR について

クラスター管理者は、MetalLB Operator をクラスターに追加し、タイプ **LoadBalancer** のサービスがクラスターに追加されると、MetalLB はサービスのフォールトトレラントな外部 IP アドレスを追加できます。外部 IP アドレスがクラスターのホストネットワークに追加されます。

23.1.1. MetalLB を使用するタイミング

MetalLB の使用は、ベアメタルクラスター、またはベアメタルのようなインフラストラクチャーがある場合や、外部 IP アドレスを使用したアプリケーションへのフォールトトレラントがあるアクセスが必要な場合に役立ちます。

ネットワークインフラストラクチャーを設定し、外部 IP アドレスのネットワークトラフィックがクライアントからクラスターのホストネットワークにルーティングされるようにする必要があります。

MetalLB Operator を使用して MetalLB をデプロイした後、タイプ **LoadBalancer** のサービスを追加すると、MetalLB はプラットフォームネイティブなロードバランサーを提供します。

23.1.2. MetalLB Operator カスタムリソース

MetalLB Operator は、2つのカスタムリソースについて独自の namespace を監視します。

MetalLB

MetalLB カスタムリソースをクラスターに追加する際に、MetalLB Operator は MetalLB をクラスターにデプロイします。Operator はカスタムリソースの単一インスタンスのみをサポートします。インスタンスが削除されると、Operator はクラスターから MetalLB を削除します。

AddressPool

MetalLB には、タイプ **LoadBalancer** のサービスを追加する際にサービスに割り当てることができる IP アドレスの1つ以上のプールが必要です。 **AddressPool** カスタムリソースをクラスターに追加する際に、MetalLB Operator はプールから IP アドレスを割り当てることができるように MetalLB を設定します。アドレスプールには IP アドレスの一覧が含まれます。リストは、1.1.1.1-1.1.1.1 などの範囲を使用して設定された単一の IP アドレス、CIDR 表記で指定された範囲、ハイフンで区切られた開始アドレスと終了アドレスとして指定された範囲、またはこの3つの組み合わせにすることができます。アドレスプールには名前が必要です。ドキュメントは、**doc-example**、**doc-example-reserved**、**doc-example-ipv6** などの名前を使用します。アドレスプールは、Bare MetalLB がプールから IP アドレスを自動的に割り当てるか、名前でプールを明示的に指定するサービス用に IP アドレスを自動的に予約するかを指定します。

MetalLB カスタムリソースをクラスターに追加し、Operator は MetalLB、Bare MetalLB ソフトウェアコンポーネント、**controller** および **講演者** をデプロイした後に、実行を開始します。

23.1.3. MetalLB ソフトウェアコンポーネント

MetalLB Operator のインストール時に、**metallb-operator-controller-manager** デプロイメントは Pod を起動します。Pod は Operator の実装です。Pod は、**MetalLB** カスタムリソースおよび **AddressPool** カスタムリソースへの変更の有無を監視します。

Operator が MetalLB のインスタンスを起動すると、**controller** デプロイメントと **speaker** のデーモンセットが開始します。

controller

Operator はデプロイメントおよび単一の Pod を起動します。**LoadBalancer** タイプのサービスを追加する場合、Kubernetes は **controller** を使用してアドレスプールから IP アドレスを割り当てます。サービスに障害が発生した場合は、**controller** Pod のログに次のエントリーがあることを確認します。

出力例

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

speaker

Operator は、クラスター内の各ノードに対して1つの **speaker** Pod を持つデーモンセットを起動します。**controller** がサービスに IP アドレスを割り当てても、サービスがまだ利用できない場合は、**speaker** Pod のログを確認してください。**スピーカー** Pod が使用できない場合は、**oc describe pod -n** コマンドを実行します。

レイヤー 2 モードの場合、**controller** がサービスに IP アドレスを割り当てると、各 **speaker** Pod は、それがサービスのエンドポイントと同じノードにあるかどうかを判別します。ノード名とサービス名をハッシュ化するアルゴリズムは、ロードバランサー IP アドレスを通知するために単一の **speaker** Pod を選択するために使用されます。**speaker** は、Address Resolution Protocol (ARP) を使用して IPv4 アドレスと Neighbor Discovery Protocol (NDP) を公開して、IPv6 アドレスにアナウンスします。

ロードバランサーの IP アドレスの要求は、IP アドレスを通知する **speaker** でノードにルーティングされます。ノードがパケットを受信した後に、サービスプロキシはパケットをサービスのエンドポイントにルーティングします。エンドポイントは、最適なケースでは同じノードに配置することも、別のノードに配置することもできます。サービスプロキシは、接続が確立されるたびにエンドポイントを選択します。

23.1.4. レイヤー 2 モードの MetalLB の概念

レイヤー 2 モードでは、1つのノードの **speaker** Pod が、サービスの外部 IP アドレスをホストネットワークに公開します。ネットワークの観点からは、ノードで複数の IP アドレスがネットワークインターフェイスに割り当てられるように見えます。



注記

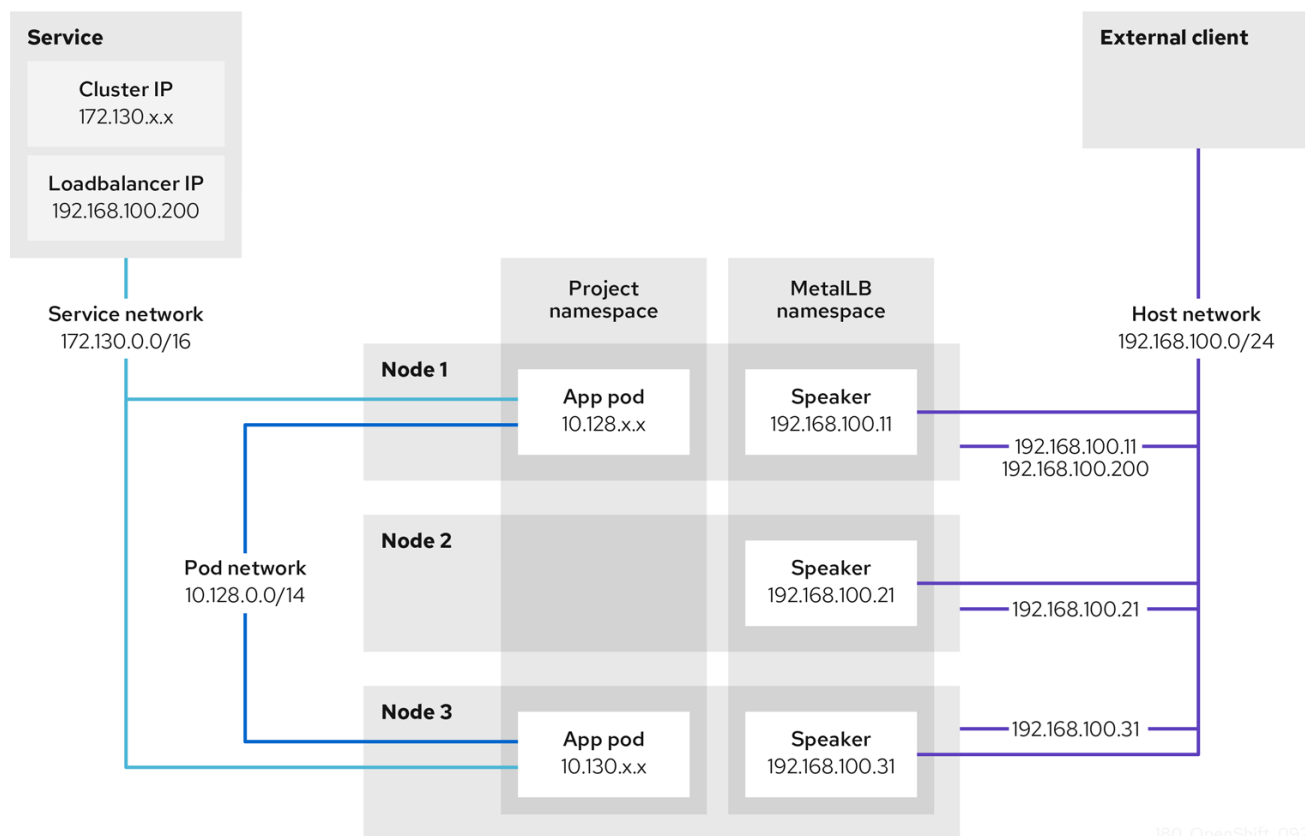
レイヤー 2 モードは ARP と NDP に依存しているため、MetalLB が機能するには、クライアントがサービスをアナウンスするノードの同じサブネット上にある必要があります。さらに、サービスに割り当てられた IP アドレスは、クライアントがサービスに到達するために使用するネットワークの同じサブネット上にある必要があります。

speaker Pod は、IPv4 サービスの ARP 要求と IPv6 の NDP 要求に応答します。

レイヤー 2 モードでは、サービス IP アドレスのすべてのトラフィックは1つのノードを介してルーティングされます。トラフィックがノードに入ると、CNI ネットワークプロバイダーのサービスプロキシはトラフィックをサービスのすべての Pod に配信します。

サービスのすべてのトラフィックがレイヤー 2 モードで単一のノードを通過するので、より厳密な意味で、MetalLB はレイヤー 2 のロードバランサーを実装しません。むしろ、MetalLB はレイヤー 2 のフェイルオーバーメカニズムを実装しているため、**speaker** Pod が利用できなくなったときに、別のノードの **speaker** Pod がサービス IP アドレスをアナウンスできます。

ノードが使用できなくなると、フェイルオーバーが自動的に行われます。他のノードの **speaker** Pod は、ノードが使用できないことを検出し、障害が発生したノードから、新しい **speaker** Pod とノードがサービス IP アドレスの所有権を取得します。



180_OpenShift_0921

前述のグラフは、MetalLB に関する以下の概念を示しています。

- アプリケーションは、**172.130.0.0/16** サブネットのクラスター IP を持つサービスで利用できます。その IP アドレスはクラスター内からアクセスできます。サービスには、MetalLB がサービス **192.168.100.200** に割り当てられている外部 IP アドレスもあります。
- ノード1および3には、アプリケーションの Pod があります。
- **speaker** デモンセットは、各ノードで Pod を実行します。MetalLB Operator はこれらの Pod を起動します。
- 各 **speaker** Pod はホストネットワーク化された Pod です。Pod の IP アドレスは、ホストネットワーク上のノードの IP アドレスと同じです。
- ノード1の **speaker** Pod は ARP を使用して、サービスの外部 IP アドレスに **192.168.100.200** を認識します。外部 IP アドレスをアナウンスする **speaker** Pod は、サービスのエンドポイントと同じノード上にあり、エンドポイントは **Ready** 状態である必要があります。
- クライアントトラフィックはホストネットワークにルーティングされ、**192.168.100.200** の IP アドレスに接続します。トラフィックがノードに入ると、サービスプロキシは、サービスに設定した外部トラフィックポリシーに従って、同じノードまたは別のノードのアプリケーション Pod にトラフィックを送信します。
- ノード1が利用できない場合、外部 IP アドレスは別のノードにフェイルオーバーします。アプリケーション Pod およびサービスエンドポイントのインスタンスを持つ別のノードでは、**speaker** Pod は外部 IP アドレス **192.168.100.200** になり、新規ノードがクライアントトラフィックを受信します。図では、唯一の候補はノード3です。

23.1.4.1. レイヤー 2 および外部トラフィックポリシー

レイヤー 2 モードでは、クラスター内のノードはサービス IP アドレスのすべてのトラフィックを受信します。クラスターがノードに入った後にトラフィックを処理する方法は、外部トラフィックポリシーの影響を受けます。

cluster

これは `spec.externalTrafficPolicy` のデフォルト値です。

cluster トラフィックポリシーでは、ノードがトラフィックを受信した後に、サービスプロキシはトラフィックをサービスのすべての Pod に分散します。このポリシーは、Pod 全体に均一なトラフィック分散を提供しますが、クライアントの IP アドレスを覆い隠し、トラフィックがクライアントではなくノードから発信されているように Pod 内のアプリケーションに表示される可能性があります。

local

local トラフィックポリシーでは、ノードがトラフィックを受信した後に、サービスプロキシはトラフィックを同じノードの Pod にのみ送信します。たとえば、ノード A の **speaker** Pod が外部サービス IP をアナウンスすると、すべてのトラフィックがノード A に送信されます。トラフィックがノード A に入った後、サービスプロキシはノード A にあるサービスの Pod にのみトラフィックを送信します。追加のノードにあるサービスの Pod は、ノード A からトラフィックを受信しません。追加のノードにあるサービスの Pod は、フェイルオーバーが必要な場合にレプリカとして機能します。

このポリシーは、クライアントの IP アドレスには影響しません。アプリケーション Pod は、受信接続からクライアント IP アドレスを判別できます。

23.1.5. 制限および制限

23.1.5.1. レイヤー 2 のみのサポート

MetalLB Operator を使用して OpenShift Container Platform 4.9 に MetalLB をインストールして設定する場合、サポートはレイヤー 2 モードのみに制限されます。反対に、オープンソースの MetalLB プロジェクトは、レイヤー 2 モードに負荷分散と、BGP(border gateway protocol) を使用するレイヤー 3 の負荷分散を提供します。

23.1.5.2. シングルスタックネットワークのサポート

同じアドレスプールに IPv4 アドレスと IPv6 アドレスを指定することができますが、MetalLB はロードバランサーに IP アドレスを 1 つだけ割り当てます。

MetalLB がデュアルスタックネットワーク用に設定されたクラスターにデプロイされると、Bare MetalLB は、サービスのクラスター IP の IP アドレスファミリーに応じて、ロードバランサーの IPv4 または IPv6 アドレスを 1 つ割り当てます。たとえば、サービスのクラスター IP が IPv4 の場合、MetalLB はロードバランサーに IPv4 アドレスを割り当てます。MetalLB は、IPv4 アドレスと IPv6 アドレスを同時に割り当てません。

IPv6 は、OVN-Kubernetes ネットワークプロバイダーを使用するクラスターでのみサポートされません。

23.1.5.3. MetalLB のインフラストラクチャーについての考慮事項

MetalLB は、ネイティブのロードバランサー機能が含まれていないため、主にオンプレミスのベアメタルインストールに役立ちます。ベアメタルのインストールに加え、一部のインフラストラクチャーに OpenShift Container Platform をインストールする場合は、ネイティブのロードバランサー機能が含ま

れていない場合があります。たとえば、以下のインフラストラクチャーは MetalLB Operator を追加するのに役立ちます。

- ベアメタル
- VMware vSphere

MetalLB Operator および MetalLB は、OpenShift SDN および OVN-Kubernetes ネットワークプロバイダーでサポートされます。

23.1.5.4. レイヤー 2 モードの制限

23.1.5.4.1. 単一ノードのボトルネック

MetalLB は、1つのノードを介してサービス内のすべてのトラフィックをルーティングします。この際、ノードはボトルネックとなり、パフォーマンスを制限する可能性があります。

レイヤー 2 モードは、サービスの Ingress 帯域幅を単一ノードの帯域幅に制限します。これは、ARP および NDP を使用してトラフィックを転送するための基本的な制限です。

23.1.5.4.2. フェイルオーバーのパフォーマンスの低下

ノード間のフェイルオーバーは、クライアントからの操作によって異なります。フェイルオーバーが発生すると、MetalLB は Gratuitous ARP パケットを送信して、サービス IP に関連付けられた MAC アドレスが変更されたことをクライアントに通知します。

ほとんどのクライアントオペレーティングシステムは、Gratuitous ARP パケットを正しく処理し、隣接キャッシュを迅速に更新します。クライアントがキャッシュを迅速に更新すると、フェイルオーバーは数秒以内に完了します。通常、クライアントは新しいノードに 10 秒以内にフェイルオーバーします。しかし、一部のクライアントオペレーティングシステムは Gratuitous ARP パケットをまったく処理しないか、キャッシュの更新を遅延させる古い実装があります。

Windows、macOS、Linux などの一般的なオペレーティングシステムの新しいバージョンは、レイヤー 2 フェイルオーバーを正しく実装します。フェイルオーバーが遅いという問題は、古くてあまり一般的ではないクライアントオペレーティングシステムを除いて、予期されていません。

古いクライアントで予定されているフェイルオーバーの影響を最小限にするには、リーダーシップをフラップした後に、古いノードを数分にわたって実行したままにします。古いノードは、キャッシュが更新されるまで、古いクライアントのトラフィックを転送することができます。

予定外のフェイルオーバー時に、古いクライアントがキャッシュエントリを更新するまでサービス IP に到達できません。

23.1.5.5. IP フェイルオーバーの非互換性

MetalLB には IP フェイルオーバー機能との互換性がありません。MetalLB Operator をインストールする前に、IP フェイルオーバーを削除します。

23.1.6. 関連情報

- [Comparison: Fault tolerant access to external IP addresses](#)
- [IP フェイルオーバーの削除](#)

23.2. METALLB OPERATOR のインストール

クラスター管理者は、Operator がクラスター上の MetalLB インスタンスのライフサイクルを管理できるようにする MetallB Operator を追加できます。

インストール手順では、**metallb-system** namespace を使用します。Operator をインストールし、カスタムリソースを別の namespace に設定できます。Operator は、Operator がインストールされている同じ namespace で MetalLB を起動します。

MetalLB および IP フェイルオーバーは互換性がありません。クラスターの IP フェイルオーバーを設定している場合、Operator をインストールする前に **IP フェイルオーバーを削除する** 手順を実行します。

23.2.1. Web コンソールを使用した OperatorHub からのインストール

OpenShift Container Platform Web コンソールを使用して OperatorHub から Operator をインストールし、これをサブスクライブできます。

手順

1. Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. スクロールするか、またはキーワードを **Filter by keyword** ボックスに入力し、必要な Operator を見つけます。たとえば、**metallb** と入力して MetalLB Operator を見つけます。また、**インフラストラクチャー機能** でオプションをフィルターすることもできます。たとえば、非接続環境 (ネットワークが制限された環境としても知られる) で機能する Operator を表示するには、**Disconnected** を選択します。
3. Operator を選択して、追加情報を表示します。



注記

コミュニティ Operator を選択すると、Red Hat がコミュニティ Operator を認定していないことを警告します。続行する前に警告を確認する必要があります。

4. Operator についての情報を確認してから、**Install** をクリックします。
5. **Install Operator** ページで以下を行います。
 - a. **Update Channel** を選択します (複数を選択できる場合)。
 - b. 前述のように、**自動 (Automatic)** または **手動 (Manual)** の承認ストラテジーを選択します。
6. **Install** をクリックし、Operator をこの OpenShift Container Platform クラスターの選択した namespace で利用可能にします。
 - a. **手動** の承認ストラテジーを選択している場合、サブスクリプションのアップグレードステータスは、そのインストール計画を確認し、承認するまで **Upgrading** のままになります。
Install Plan ページでの承認後に、サブスクリプションのアップグレードステータスは **Up to date** に移行します。
 - b. **自動** の承認ストラテジーを選択している場合、アップグレードステータスは、介入なしに **Up to date** に解決するはずです。

- サブスクリプションのアップグレードステータスが **Up to date** になった後に、**Operators → Installed Operators** を選択し、インストールされた Operator のクラスターサービスバージョン (CSV) が表示されることを確認します。その **Status** は最終的に関連する namespace で **InstallSucceeded** に解決するはずですが。



注記

All namespaces... インストールモードの場合、ステータスは **openshift-operators** namespace で **InstallSucceeded** になりますが、他の namespace でチェックする場合、ステータスは **Copied** になります。

上記通りにならない場合、以下を実行します。

- さらにトラブルシューティングを行うために問題を報告している **Workloads → Pods** ページで、**openshift-operators** プロジェクト (または **A specific namespace...** インストールモードが選択されている場合は他の関連の namespace) の Pod のログを確認します。

23.2.2. CLI を使用した OperatorHub からのインストール

OpenShift Container Platform Web コンソールを使用する代わりに、CLI を使用して OperatorHub から Operator をインストールできます。**oc** コマンドを使用して、**Subscription** オブジェクトを作成または更新します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- cluster-admin** 権限を持つユーザーとしてログインしている。

手順

- MetalLB Operator が利用可能であることを確認します。

```
$ oc get packagemanifests -n openshift-marketplace metallb-operator
```

出力例

```
NAME          CATALOG          AGE
metallb-operator Red Hat Operators 9h
```

- metallb-system** namespace を作成します。

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: metallb-system
EOF
```

- namespace に Operator グループのカスタムリソースを作成します。

```
$ cat << EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
```

```
kind: OperatorGroup
metadata:
  name: metallb-operator
  namespace: metallb-system
spec:
  targetNamespaces:
  - metallb-system
EOF
```

4. Operator グループが namespace にインストールされていることを確認します。

```
$ oc get operatorgroup -n metallb-system
```

出力例

```
NAME          AGE
metallb-operator 14m
```

5. MetalLB Operator にサブスクライブします。

- a. 以下のコマンドを実行して OpenShift Container Platform のメジャーおよびマイナーバージョンを取得します。値を使用して、次の手順で **channel** 値を設定します。

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

- b. Operator のサブスクリプションカスタムリソースを作成するには、以下のコマンドを入力します。

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

6. インストール計画が namespace にあることを確認します。

```
$ oc get installplan -n metallb-system
```

出力例

```
NAME          CSV                                APPROVAL APPROVED
install-wzg94 metallb-operator.4.9.0-nnnnnnnnnnnn Automatic true
```

7. Operator がインストールされていることを確認するには、以下のコマンドを入力します。

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

出力例

```
Name                      Phase
metallb-operator.4.9.0-nnnnnnnnnnnn Succeeded
```

23.2.3. クラスタでの MetalLB の起動

Operator のインストール後に、MetalLB カスタムリソースの単一のインスタンスを設定する必要があります。カスタムリソースの設定後、Operator はクラスタで MetalLB を起動します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- MetalLB Operator をインストールしている。

手順

1. MetalLB カスタムリソースの単一のインスタンスを作成します。

```
$ cat << EOF | oc apply -f -
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
EOF
```

検証

MetalLB コントローラーのデプロイメントと、BareLB スピーカーのデーモンセットが実行していることを確認します。

1. コントローラーのデプロイメントが稼働していることを確認します。

```
$ oc get deployment -n metallb-system controller
```

出力例

```
NAME      READY  UP-TO-DATE  AVAILABLE  AGE
controller 1/1    1           1          11m
```

2. スピーカーに設定されているデーモンが実行していることを確認します。

```
$ oc get daemonset -n metallb-system speaker
```

出力例

-

```

NAME      DESIRED CURRENT READY UP-TO-DATE AVAILABLE NODE
SELECTOR  AGE
speaker 6      6      6      6      6      kubernetes.io/os=linux 18m

```

この出力例は、6つの speaker Pod を示しています。クラスターの speaker Pod の数は出力例とは異なる場合があります。出力で各ノードの1つの Pod が表示されることを確認します。

23.2.4. 次のステップ

- [MetalLB アドレスプールの設定](#)

23.3. METALLB アドレスプールの設定

クラスター管理者は、アドレスプールを追加、変更、および削除できます。MetalLB Operator は、アドレスプールカスタムリソースを使用して、MetalLB がサービスに割り当てることのできる IP アドレスを設定します。

23.3.1. アドレスプールのカスタムリソースについて

アドレスプールカスタムリソースのフィールドは、以下の表で説明されています。

表23.1 MetalLB アドレスプールのカスタムリソース

フィールド	型	説明
<code>metadata.name</code>	<code>string</code>	アドレスプールの名前を指定します。サービスを追加する場合は、 <code>metallb.universe.tf/address-pool</code> アノテーションにこのプール名を指定して、特定のプールから IP アドレスを選択できます。ドキュメント全体で、 <code>doc-example</code> 、 <code>silver</code> 、および <code>gold</code> の名前が使用されます。
<code>metadata.name space</code>	<code>string</code>	アドレスプールの namespace を指定します。MetalLB Operator が使用するものと同じ namespace を指定します。
<code>spec.protocol</code>	<code>string</code>	ロードバランサー IP アドレスをピアノードに通知するためのプロトコルを指定します。サポートされている値は <code>layer2</code> のみです。
<code>spec.autoAssign</code>	<code>boolean</code>	オプション: MetalLB がこのプールから IP アドレスを自動的に割り当てるかどうかを指定します。 <code>metallb.universe.tf/address-pool</code> アノテーションを使用してこのプールから IP アドレスを明示的に要求する場合は、 <code>false</code> を指定します。デフォルト値は <code>true</code> です。
<code>spec.addresses</code>	<code>array</code>	サービスに割り当てる MetalLB の IP アドレスの一覧を指定します。1つのプールに複数の範囲を指定できます。CIDR 表記で各範囲を指定するか、開始および終了の IP アドレスをハイフンで区切って指定します。

23.3.2. アドレスプールの設定

クラスター管理者は、クラスターにアドレスプールを追加して、MetaLLB がロードバランサーサービスに割り当てることができる IP アドレスを制御できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の例のような内容で、**addresspool.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1alpha1
kind: AddressPool
metadata:
  namespace: metallb-system
  name: doc-example
spec:
  protocol: layer2
  addresses:
  - 203.0.113.1-203.0.113.10
  - 203.0.113.65-203.0.113.75
```

2. アドレスプールの設定を適用します。

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

検証

- アドレスプールを表示します。

```
$ oc describe -n metallb-system addresspool doc-example
```

出力例

```
Name:      doc-example
Namespace: metallb-system
Labels:    <none>
Annotations: <none>
API Version: metallb.io/v1alpha1
Kind:      AddressPool
Metadata:
  ...
Spec:
  Addresses:
    203.0.113.1-203.0.113.10
    203.0.113.65-203.0.113.75
  Auto Assign: true
  Protocol:    layer2
  Events:     <none>
```

doc-example などのアドレスプール名と IP アドレス範囲が出力に表示されることを確認します。

23.3.3. アドレスプールの設定例

23.3.3.1. 例: IPv4 および CIDR 範囲

CIDR 表記で IP アドレスの範囲を指定できます。CIDR 表記と、ハイフンを使用する表記を組み合わせると下層と上限を分けることができます。

```
apiVersion: metallb.io/v1beta1
kind: AddressPool
metadata:
  name: doc-example-cidr
  namespace: metallb-system
spec:
  protocol: layer2
  addresses:
  - 192.168.100.0/24
  - 192.168.200.0/24
  - 192.168.255.1-192.168.255.5
```

23.3.3.2. 例: IP アドレスの予約

MetalLB がプールから IP アドレスを自動的に割り当てないように **autoAssign** フィールドを **false** に設定できます。サービスを追加する場合は、プールから特定の IP アドレスを要求するか、またはそのプールから任意の IP アドレスを要求するためにアノテーションでプール名を指定できます。

```
apiVersion: metallb.io/v1beta1
kind: AddressPool
metadata:
  name: doc-example-reserved
  namespace: metallb-system
spec:
  protocol: layer2
  addresses:
  - 10.0.100.0/28
  autoAssign: false
```

23.3.3.3. 例: IPv6 アドレスプール

IPv6 を使用するアドレスプールを追加できます。以下の例は、1つの IPv6 範囲を示しています。ただし、複数の IPv4 の例と同様に、**addresses** 一覧で複数の範囲を指定できます。

```
apiVersion: metallb.io/v1beta1
kind: AddressPool
metadata:
  name: doc-example-ipv6
  namespace: metallb-system
spec:
  protocol: layer2
  addresses:
  - 2002:2:2::1-2002:2:2::100
```

23.3.4. 次のステップ

- [MetalLB を使用するためのサービスの設定](#)

23.4. METALLB を使用するためのサービスの設定

クラスター管理者は、タイプ **LoadBalancer** のサービスを追加するときに、MetalLB が IP アドレスを割り当てる方法を制御できます。

23.4.1. 特定の IP アドレスの要求

他のロードバランサーの実装と同様に、MetalLB はサービス仕様の **spec.loadBalancerIP** フィールドを受け入れます。

要求された IP アドレスが任意のアドレスプールの範囲内にある場合、MetalLB は要求された IP アドレスを割り当てます。要求された IP アドレスが範囲外の場合、MetalLB は警告を報告します。

特定の IP アドレスのサービス YAML の例

```
apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  loadBalancerIP: <ip_address>
```

MetalLB が要求された IP アドレスを割り当てることができない場合、サービスの **EXTERNAL-IP** が **<pending>** を報告し、**oc describe service <service_name>** の実行には、以下の例のようなイベントが含まれます。

MetalLB が要求された IP アドレスを割り当てることができない場合のイベントの例

```
...
Events:
  Type    Reason          Age   From          Message
  ----    -
  Warning AllocationFailed 3m16s metallb-controller Failed to allocate IP for "default/invalid-request": "4.3.2.1" is not allowed in config
```

23.4.2. 特定のプールからの IP アドレスの要求

特定の範囲から IP アドレスを割り当てても、特定の IP アドレスを気にしない場合は、**metallb.universe.tf/address-pool** アノテーションを使用して、指定したアドレスプールから IP アドレスを要求できます。

特定プールからの IP アドレスのサービス YAML の例

■

```

apiVersion: v1
kind: Service
metadata:
  name: <service_name>
  annotations:
    metallb.universe.tf/address-pool: <address_pool_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer

```

<address_pool_name> に指定するアドレスプールが存在しない場合、MetalLB は、自動割り当てを許可する任意のプールから IP アドレスを割り当てようとします。

23.4.3. 任意の IP アドレスを許可します。

デフォルトでは、アドレスプールは自動割り当てを許可するように設定されます。MetalLB は、これらのアドレスプールから IP アドレスを割り当てます。

自動割り当て用に設定されたプールから IP アドレスを受け入れるには、特別なアノテーションや設定は必要ありません。

任意の IP アドレスを受け入れるサービス YAML の例

```

apiVersion: v1
kind: Service
metadata:
  name: <service_name>
spec:
  selector:
    <label_key>: <label_value>
  ports:
    - port: 8080
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer

```

23.4.4. 特定の IP アドレスを共有

デフォルトでは、サービスは IP アドレスを共有しません。ただし、単一の IP アドレスにサービスを配置する必要がある場合は、**metallb.universe.tf/allow-shared-ip** アノテーションをサービスに追加することで、選択的な IP 共有を有効にできます。

```

apiVersion: v1
kind: Service
metadata:
  name: service-http
  annotations:
    metallb.universe.tf/address-pool: doc-example
    metallb.universe.tf/allow-shared-ip: "web-server-svc" 1

```

```

spec:
  ports:
    - name: http
      port: 80 ②
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value> ③
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7 ④
---
apiVersion: v1
kind: Service
metadata:
  name: service-https
  annotations:
    metallb.universe.tf/address-pool: doc-example
    metallb.universe.tf/allow-shared-ip: "web-server-svc" ⑤
spec:
  ports:
    - name: https
      port: 443 ⑥
      protocol: TCP
      targetPort: 8080
  selector:
    <label_key>: <label_value> ⑦
  type: LoadBalancer
  loadBalancerIP: 172.31.249.7 ⑧

```

- ① ⑤ **metallb.universe.tf/allow-shared-ip** アノテーションに同じ値を指定します。この値は共有キーと呼ばれます。
- ② ⑥ サービスに異なるポート番号を指定します。
- ③ ⑦ **externalTrafficPolicy: local** を指定し、サービスが同じ Pod のセットにトラフィックを送信できるようにするために、同じ Pod セレクターを指定します。**cluster** の外部トラフィックポリシーを使用する場合、Pod セレクターは同じである必要はありません。
- ④ ⑧ オプション: 上記の 3 つの項目を指定すると、MetalLB は同じ IP アドレスにサービスを配置する場合があります。サービスが IP アドレスを共有することを確認するには、共有する IP アドレスを指定します。

デフォルトで、Kubernetes はマルチプロトコルロードバランサーサービスを許可しません。この制限は通常、TCP と UDP の両方をリッスンする必要がある DNS などのサービスを実行できなくなります。MetalLB を使用して Kubernetes のこの制限を回避するには、2 つのサービスを作成します。

- 1 つのサービスには TCP を指定し、2 番目のサービスには UDP を指定します。
- 両方のサービスで、同じ Pod セレクターを指定します。
- 同じ共有キーと **spec.loadBalancerIP** 値を指定して、TCP サービスと UDP サービスを同じ IP アドレスに配置します。

23.4.5. MetalLB を使用したサービスの設定

アドレスプールから外部 IP アドレスを使用するように、負荷分散サービスを設定することができます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- MetalLB Operator をインストールして、MetalLB を起動します。
- 1つ以上のアドレスプールを設定します。
- トラフィックをクライアントからクラスターのホストネットワークにルーティングするようにネットワークを設定します。

手順

1. **<service_name>.yaml** ファイルを作成します。このファイルで、**spec.type** フィールドが **LoadBalancer** に設定されていることを確認します。
MetalLB がサービスに割り当てる外部 IP アドレスを要求する方法については、例を参照してください。
2. サービスを作成します。

```
$ oc apply -f <service_name>.yaml
```

出力例

```
service/<service_name> created
```

検証

- サービスを記述します。

```
$ oc describe service <service_name>
```

出力例

```
Name:          <service_name>
Namespace:     default
Labels:        <none>
Annotations:   metallb.universe.tf/address-pool: doc-example <.>
Selector:      app=service_name
Type:          LoadBalancer <.>
IP Family Policy:  SingleStack
IP Families:    IPv4
IP:             10.105.237.254
IPs:            10.105.237.254
LoadBalancer Ingress:  192.168.100.5 <.>
Port:          <unset> 80/TCP
TargetPort:    8080/TCP
NodePort:      <unset> 30550/TCP
Endpoints:     10.244.0.50:8080
Session Affinity:  None
External Traffic Policy: Cluster
```

```
Events: <.>
  Type Reason      Age          From          Message
  ---- -
Normal nodeAssigned 32m (x2 over 32m) metallb-speaker announcing from node "
<node_name>"
```

<.> 特定のプールから IP アドレスを要求すると、アノテーションが表示されます。<.> サービスタイプは **LoadBalancer** を示す必要があります。<.> サービスが正しく割り当てられている場合、ロードバランサーの Ingress フィールドは外部 IP アドレスを示します。<.> events フィールドは、外部 IP アドレスを通知するために割り当てられたノード名を示します。エラーが発生した場合、events フィールドはエラーの理由を示します。

第24章 セカンダリーインターフェイスメトリクスのネットワーク割り当てへの関連付け

24.1. モニタリングのためのセカンダリーネットワークメトリックの拡張

セカンダリーデバイス(インターフェイス)は、各種の用途に合わせて使用されます。セカンダリーデバイスのメトリクスを同じ分類で集計するために、それらを分類する方法を確保する必要があります。

公開されるメトリクスにはインターフェイスが含まれますが、インターフェイスの出所は指定されません。これは、追加のインターフェイスがない場合に実行できます。ただし、セカンダリーインターフェイスが追加された場合、インターフェイス名だけを使用してインターフェイスを識別するのは難しいため、メトリックの使用が困難になる可能性があります。

セカンダリーインターフェイスを追加する場合、その名前は追加された順序によって異なります。また、異なるセカンダリーインターフェイスが異なるネットワークに属し、これらを異なる目的に使用できます。

`pod_network_name_info` を使用すると、現在のメトリクスをインターフェイスタイプを識別する追加情報を使用して拡張できます。このようにして、メトリクスを集約し、特定のインターフェイスタイプに特定のアラームを追加できます。

ネットワークタイプは、関連する **NetworkAttachmentDefinition** の名前を使用して生成されます。この名前は、セカンダリーネットワークの異なるクラスを区別するために使用されます。たとえば、異なるネットワークに属するインターフェイスや、異なる CNI を使用するインターフェイスは、異なるネットワーク割り当て定義名を使用します。

24.1.1. Network Metrics Daemon

Network Metrics Daemon は、ネットワーク関連のメトリクスを収集し、公開するデーモンコンポーネントです。

kubelet はすでに確認できるネットワーク関連のメトリクスを公開しています。以下は、これらのメトリクスになります。

- `container_network_receive_bytes_total`
- `container_network_receive_errors_total`
- `container_network_receive_packets_total`
- `container_network_receive_packets_dropped_total`
- `container_network_transmit_bytes_total`
- `container_network_transmit_errors_total`
- `container_network_transmit_packets_total`
- `container_network_transmit_packets_dropped_total`

これらのメトリクスのラベルには、とくに以下が含まれます。

- Pod の名前
- Pod の namespace

- インターフェイス名 (例: `eth0`)

これらのメトリクスは、たとえば [Multus](#) を使用して、新規インターフェイスが Pod に追加されるまで正常に機能します。

インターフェイスのラベルはインターフェイス名を参照しますが、そのインターフェイスの用途は明確ではありません。多くの異なるインターフェイスがある場合、監視しているメトリクスが参照するネットワークを把握することはできません。

これには、以降のセクションで説明する新規の `pod_network_name_info` を導入して対応できます。

24.1.2. ネットワーク名を持つメトリクス

この daemonset は、固定の値が `0` の `pod_network_name_info` 測定メトリクスを公開します。

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

ネットワーク名ラベルは、Multus によって追加されるアノテーションを使用して生成されます。これは、ネットワークの割り当て定義が属する namespace の連結と、ネットワーク割り当て定義の名前です。

新しいメトリクスのみでは十分な値が提供されませんが、ネットワーク関連の `container_network_*` メトリクスと組み合わせて、セカンダリーネットワークの監視に対するサポートを強化します。

以下のような `promql` クエリーを使用すると、`k8s.v1.cni.cncf.io/networks-status` アノテーションから取得した値とネットワーク名を含む新規のメトリクスを取得できます。

```
(container_network_receive_bytes_total) + on(namespace,pod,interface) group_left(network_name) (
pod_network_name_info )
(container_network_receive_errors_total) + on(namespace,pod,interface) group_left(network_name) (
pod_network_name_info )
(container_network_receive_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_receive_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_bytes_total) + on(namespace,pod,interface) group_left(network_name)
( pod_network_name_info )
(container_network_transmit_errors_total) + on(namespace,pod,interface) group_left(network_name)
( pod_network_name_info )
(container_network_transmit_packets_total) + on(namespace,pod,interface)
group_left(network_name) ( pod_network_name_info )
(container_network_transmit_packets_dropped_total) + on(namespace,pod,interface)
group_left(network_name)
```