OpenShift Container Platform 4.15

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

Last Updated: 2024-05-17

OpenShift Container Platform 4.15 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

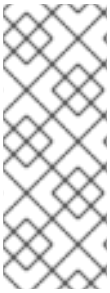
概要

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 ネットワークの概要

Red Hat OpenShift Networking は、1 つまたは複数のハイブリッドクラスターのネットワークトラフィックを管理するためにクラスターが必要とする高度なネットワーク関連機能で Kubernetes ネットワーキングを拡張する機能、プラグイン、高度なネットワーク機能のエコシステムです。このネットワーキング機能のエコシステムは、ingress、egress、ロードバランシング、高性能スループット、セキュリティー、クラスター間およびクラスター内のトラフィック管理を統合し、ロールベースの可観測性ツールを提供して複雑さを軽減します。



注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

以下のリストは、クラスターで利用可能な最も一般的に使用される Red Hat OpenShift Networking 機能の一部を強調しています。

- 次の Container Network Interface (CNI) プラグインのいずれかによって提供されるプライマリークラスターネットワーク:
  - OVN-Kubernetes ネットワークプラグイン (デフォルトのプラグイン)
  - OpenShift SDN ネットワークプラグイン
- 認定されたサードパーティーの代替プライマリーネットワークプラグイン
- ネットワークプラグイン管理用の Cluster Network Operator
- TLS 暗号化 Web トラフィックの Ingress Operator
- 名前割り当てのための DNS Operator
- ベアメタルクラスターでのトラフィック負荷分散用の MetalLB Operator
- 高可用性のための IP フェイルオーバーのサポート
- macvlan、ipvlan、SR-IOV ハードウェアネットワークなど、複数の CNI プラグインによる追加のハードウェアネットワークサポート
- IPv4、IPv6、およびデュアルスタックアドレッシング
- Windows ベースのワークロード用のハイブリッド Linux-Windows ホストクラスター
- サービスのディスカバリー、ロードバランシング、サービス間認証、障害リカバリー、メトリクス、およびモニター用の Red Hat OpenShift Service Mesh
- シングルノード OpenShift
- ネットワークのデバッグと洞察のための Network Observability Operator
- クラスター間ネットワーキングのための Submariner および Red Hat Application Interconnect テクノロジー

第2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ー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 とそのコンテナー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。

2.1. OPENSHIFT 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。

2.2. OPENSHIFT 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 Controller 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 Controller 内の設定は、Ingress Controller エンドポイントを公開する方法を提供します。

2.2.1. 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 Controller 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 Controller 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 Controller 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。

2.3. OPENSHIFT 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、インストール時にクラスター用に選択された Container Network Interface (CNI) ネットワークプラグインのデプロイが含まれます。

ConfigMap

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ーは、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ー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 Controller を実装します。

Ingress Controller

Ingress Operator は Ingress Controller を管理します。Ingress Controller の使用は、最も一般的な、OpenShift Container Platform クラスターへの外部アクセスを許可する方法です。

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

コンテナーが 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ーサービスであり、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ー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。

MetalLB Operator

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 つ以上のコンテナー。Pod は、定義、デプロイ、および管理される最小のコンピュート単位です。

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 Controller 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。

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

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ルされます。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 をスケジュールできます。

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3章 ゼロトラストネットワーク

ゼロトラストは、すべてのインタラクションが信頼できない状態から始まるという前提に基づきセキュリティーアーキテクチャーを設計するアプローチです。これは、通信がファイアウォールの内側で開始されるかどうかに基づき信頼性を決定する従来のアーキテクチャーとは対照的です。より具体的には、ゼロトラストは、暗黙的信頼モデルと 1 回限りの認証に依存するセキュリティーアーキテクチャーのギャップを埋めようとします。

OpenShift Container Platform は、コンテナーやコンテナー内で実行されているソフトウェアを変更することなく、プラットフォーム上で実行されているコンテナーにゼロトラストネットワーク機能を追加できます。Red Hat は、コンテナーのゼロトラストネットワーキング機能をさらに強化できる製品もさらにいくつか提供しています。コンテナー内で実行されているソフトウェアを変更できる場合は、Red Hat がサポートする他のプロジェクトを使用して、さらに機能を追加できます。

以下は、ゼロトラストネットワークの対象となる機能の詳細です。

3.1. ROOT OF TRUST

公開証明書と秘密鍵はゼロトラストネットワークにとって重要です。これらは、各コンポーネントを相互に識別し、認証し、トラフィックを保護するために使用されます。証明書は他の証明書によって署名され、ルート認証局 (CA) への信頼チェーンが存在します。ネットワークに参加するすべてが、信頼チェーンを検証できるように、最終的にルート CA の公開鍵を持っている必要があります。一般に公開されている場合、通常は世界的に知られているルート CA のセットであり、その鍵はオペレーティングシステムや Web ブラウザーなどとともに配布されます。ただし、プライベート CA の証明書がすべての関係者に配布されている場合、クラスターまたは企業向けにプライベート CA を実行できます。

活用方法:

- OpenShift Container Platform: OpenShift は、クラスターリソースの保護に使用される クラスター CA をインストール時に作成 します。ただし、OpenShift Container Platform は、クラスター内で サービス証明書 を作成して署名することもでき、そのクラスター CA バンドルを必要に応じて Pod に注入することもできます。OpenShift Container Platform によって作成および署名された サービス証明書 の有効期間 (TTL) は 26 カ月で、13 カ月ごとに自動的にローテーションされます。必要に応じて手動でローテーションすることも可能です。

- OpenShift cert-manager Operator: cert-manager を使用すると、外部の root of trust によって署名された鍵を要求できます。委譲された署名証明書を使用して実行する方法の他に、外部発行者と統合するために設定できる発行者が多数あります。cert-manager API は、ゼロトラストネットワーク内の他のソフトウェア (Red Hat OpenShift Service Mesh など) が必要な証明書を要求するために使用することも、お客様のソフトウェアが直接使用することも可能です。

3.2. トラフィック認証と暗号化

回線上のすべてのトラフィックが暗号化され、エンドポイントが識別可能になります。一例として、相互認証方式である Mutual TLS (mTLS) が挙げられます。

活用方法:

- OpenShift Container Platform: 透過的な Pod 間の IPsec を使用することで、トラフィックの送信元と宛先を IP アドレスで識別できます。Egress トラフィックを IPsec を使用して暗号化 する機能があります。Egress IP 機能を使用すると、トラフィックの送信元 IP アドレスを使用して、クラスター内のトラフィックの送信元を識別できます。

- Red Hat OpenShift Service Mesh : Pod から送信されるトラフィックを透過的に拡張して認証と暗号化を提供する強力な mTLS 機能 を提供します。

- OpenShift cert-manager Operator: カスタムリソース定義 (CRD) を使用して、プログラムが SSL/TLS プロトコルに使用するためにマウントできる証明書を要求します。

3.3. 識別と認証

CA を使用して証明書を作成できるようになると、それを使用して接続のもう一方の端 (ユーザーまたはクライアントマシン) のアイデンティティーを検証することで信頼関係を確立できるようになります。また、侵害された場合に使用を制限するために、証明書のライフサイクルを管理する必要もあります。

活用方法:

- OpenShift Container Platform: クライアントが信頼済みエンドポイントと通信していることを確認するためのクラスター署名付き サービス証明書。これには、サービスが SSL/TLS を使用し、クライアントが クラスター CA を使用することが求められます。クライアントアイデンティティーは他の手段で提供する必要があります。

- Red Hat Single Sign-On: エンタープライズユーザーディレクトリーまたはサードパーティーのアイデンティティープロバイダーとの要求認証統合を提供します。

- Red Hat OpenShift Service Mesh: mTLS への接続の 透過的なアップグレード、自動ローテーション、カスタム証明書の有効期限、JSON Web トークン (JWT) による要求認証。

- OpenShift cert-manager Operator: アプリケーションで使用する証明書の作成と管理。証明書は CRD により制御され、シークレットとしてマウントできます。また、cert-manager API と直接対話するようにアプリケーションを変更することもできます。

3.4. サービス間認可

リクエスト元のアイデンティティーに基づきサービスへのアクセスを制御できることが重要です。これはプラットフォームによって実行されるため、各アプリケーションで実装する必要はありません。これにより、ポリシーの監査と検査がより適切に行えるようになります。

活用方法:

- OpenShift Container Platform: Kubernetes **NetworkPolicy** および **AdminNetworkPolicy** オブジェクトを使用して、プラットフォームのネットワーク層で分離を強制できます。

- Red Hat OpenShift Service Mesh: 標準の Istio オブジェクトを使用し、mTLS を使用してトラフィックの送信元と宛先を識別し、その情報に基づきポリシーを適用する、高度な L4 および L7 の トラフィック制御。

3.5. トランザクションレベルの検証

接続の識別および認証機能に加えて、個々のトランザクションへのアクセスを制御するのにも役立ちます。これには、ソースによるレート制限、可観測性、トランザクションが適切に形成されていることのセマンティック検証などが含まれます。

活用方法:

- Red Hat OpenShift Service Mesh: リクエストの L7 検査、不正な HTTP リクエストの拒否、トランザクションレベルの 可観測性およびレポート を行います。Service Mesh は、JWT を使用した リクエストベースの認証 も提供できます。

3.6. リスク評価

クラスター内のセキュリティーポリシーの数が増えるにつれ、ポリシーが許可および拒否する内容の可視化がますます重要になります。これらのツールを使用すると、クラスターセキュリティーポリシーの作成、可視化、管理が容易になります。

活用方法:

- Red Hat OpenShift Service Mesh: OpenShift Web コンソール を使用して、Kubernetes の **NetworkPolicy** と **AdminNetworkPolicy**、および OpenShift Networking の **EgressFirewall** オブジェクトを作成および可視化します。
- Red Hat Advanced Cluster Security for Kubernetes : 高度な オブジェクトの可視化。

3.7. サイト全体のポリシーの施行と配布

クラスターにアプリケーションをデプロイした後、セキュリティールールを構成するすべてのオブジェクトを管理することが困難になります。サイト全体にポリシーを適用し、デプロイしたオブジェクトがポリシーに準拠しているかどうかを監査することが重要になります。これにより、定義された範囲内でユーザーとクラスター管理者に一部の権限を委譲できるようになり、必要に応じてポリシーの例外も許可されるようになります。

活用方法:

- Red Hat OpenShift Service Mesh: ポリシーオブジェクトと委譲を制御する RBAC。
- Red Hat Advanced Cluster Security for Kubernetes : ポリシー適用 エンジン。
- Red Hat Advanced Cluster Management (RHACM) for Kubernetes : 一元的なポリシー制御。

3.8. 継続的かつ遡及的な評価のための可観測性

クラスターを実行した後は、トラフィックを観察し、トラフィックが定義したルールに準拠していることを確認する必要があります。これは侵入検知やフォレンジックのために重要であり、運用負荷管理にも役立ちます。

活用方法:

- Network Observability Operator: クラスター内の Pod やノードへのネットワーク接続に関する検査、監視、アラートが可能になります。
- Red Hat Advanced Cluster Management (RHACM) for Kubernetes : プロセス実行、ネットワーク接続とフロー、権限昇格などのシステムレベルのイベントを監視、収集、評価します。クラスターのベースラインを決定し、異常なアクティビティーを検出して警告することができます。
- Red Hat OpenShift Service Mesh: Pod に出入りする トラフィックを監視 できます。
- Red Hat OpenShift 分散トレーシングプラットフォーム: 適切にインストルメント化されたアプリケーションの場合、特定のアクションがマイクロサービスへのサブリクエストに分割されるときに、そのアクションに関連付けられたすべてのトラフィックを確認できます。これにより、分散アプリケーション内のボトルネックを特定できます。

3.9. エンドポイントセキュリティー

クラスター内のサービスを実行しているソフトウェアが侵害されていないことを確信できることが重要です。たとえば、認定されたイメージが信頼済みハードウェア上で実行されるようにし、エンドポイン

トの特性に基づいてエンドポイントとの間の接続のみを許可するポリシーを設定する必要がある場合があります。

活用方法:

- OpenShift Container Platform: Secureboot を使用すると、クラスター内のノードが信頼済みのソフトウェアを実行していることを確認できるため、プラットフォーム自体 (コンテナーランタイムを含む) が改ざんされていないことを確認できます。OpenShift Container Platform を、特定の署名で署名された イメージのみを実行するように設定できます。

- Red Hat Trusted Artifact Signer : 信頼済みビルドチェーンで使用でき、署名付きコンテナーイメージを生成できます。

3.10. クラスター外への信頼の拡張

クラスターによるサブドメインの CA 作成を許可することで、クラスター外部に信頼を拡張する必要がある場合があります。あるいは、クラスター内のワークロードアイデンティティーをリモートエンドポイントに証明する必要があることもあります。

活用方法:

- OpenShift cert-manager Operator: cert-manager を使用して委譲された CA を管理し、クラスターをまたいで、もしくは組織全体で信頼を分散できます。

- Red Hat OpenShift Service Mesh : SPIFFE を使用して、リモートまたはローカルクラスターで実行されているエンドポイントにワークロードのリモートアテステーションを提供できます。

第4章 ホストへのアクセス

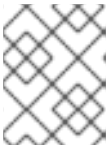
OpenShift Container Platform インスタンスにアクセスして、セキュアシェル (SSH) アクセスでコントロールプレーンノードにアクセスするために bastion ホストを作成する方法を学びます。

4.1. 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ークラスターでの 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ー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ため、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じ SSH キーを使用するようにします。

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

第5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。

5.1. 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、インストール時にクラスター用に選択された Container Network Interface (CNI) ネットワークプラグインのデプロイが含まれます。詳細は、OpenShift Container Platform における Cluster Network Operator を参照してください。

5.2. 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。

5.3. 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。

5.4. 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。詳細は、Understanding the External DNS Operator を参照してください。

5.5. INGRESS NODE FIREWALL OPERATOR

Ingress Node Firewall Operator は、拡張された Berkley Packet Filter (eBPF) と eXpress Data Path (XDP) プラグインを使用して、ノードファイアウォールルールを処理し、統計を更新し、ドロップされたトラフィックのイベントを生成します。Operator は、Ingress ノードファイアウォールリソースを管理し、ファイアウォール設定を検証し、クラスターアクセスを妨げる可能性がある誤設定されたルールを許可せず、ルールのオブジェクトで選択されたインターフェイスに Ingress ノードファイアウォール XDP プログラムをロードします。詳細は、Ingress Node Firewall Operator についてを参照してください。

5.6. NETWORK OBSERVABILITY OPERATOR

Network Observability Operator は、クラスター管理者が OpenShift Container Platform クラスターのネットワークトラフィックを観察するために使用できるオプションの Operator です。Network Observability Operator は、eBPF テクノロジーを使用してネットワークフローを作成します。その後、ネットワークフローは OpenShift Container Platform 情報で強化され、Loki に保存されます。保存されたネットワークフロー情報を OpenShift Container Platform コンソールで表示および分析して、さらなる洞察とトラブルシューティングを行うことができます。詳細は、Network Observability Operator について を参照してください。

第6章 ネットワークダッシュボード

ネットワークメトリクスは、OpenShift Container Platform Web コンソール内のダッシュボードから、**Observe → Dashboards** で表示できます。

6.1. NETWORK OBSERVABILITY OPERATOR

Network Observability Operator がインストールされている場合は、**Dashboards** ドロップダウンリストから **Netobserv** ダッシュボードを選択すると、ネットワークトラフィックメトリクスダッシュボードが表示されます。この **ダッシュボード** で利用できるメトリクスの詳細は、ネットワーク可観測性メトリクスのダッシュボード を参照してください。

6.2. ネットワーキングと OVN-KUBERNETES ダッシュボード

このダッシュボードからは、一般的なネットワークメトリクスと OVN-Kubernetes メトリクスの両方を表示できます。

一般的なネットワークメトリクスを表示するには、**Dashboards** ドロップダウンリストから **Networking/Linux Subsystem Stats** を選択します。ダッシュボードから表示できるネットワークメトリクスは、**Network Utilisation**、**Network Saturation**、**Network Errors** です。

OVN-Kubernetes メトリクスを表示するには、**Dashboards** ドロップダウンリストから **Networking/Infrastructure** を選択します。表示できる OVN-Kuberenetes メトリクスは、**Networking Configuration**、**TCP Latency Probes**、**Control Plane Resources**、**Worker Resources** です。

6.3. INGRESS OPERATOR ダッシュボード

Ingress Operator によって処理されるネットワークメトリクスをダッシュボードから表示できます。これには、次のようなメトリクスが含まれます。

- 受信および送信の帯域幅
- HTTP エラーレート
- HTTP サーバーの応答遅延

これらの Ingress メトリクスを表示するには、**Dashboards** ドロップダウンリストから **Networking/Ingress** を選択します。**Top 10 Per Route**、**Top 10 Per Namespace**、**Top 10 Per Shard** のカテゴリーの Ingress メトリクスを表示できます。

第7章 OPENSHIFT 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) ネットワークプラグインを含む、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。

7.1. 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、クラスターのインストール中に選択した OVN-Kubernetes ネットワークプラグインまたは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。

7.2. 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: 1
  Cluster Network:
    Cidr:         10.128.0.0/14
    Host Prefix:  23
  Network Type:   OpenShiftSDN
  Service Network:
    172.30.0.0/16
Status: 2
  Cluster Network:
    Cidr:               10.128.0.0/14
    Host Prefix:        23
  Cluster Network MTU:  8951
  Network Type:         OpenShiftSDN
  Service Network:
    172.30.0.0/16
Events:  <none>
```

1 **Spec** 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。

2 **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。

7.3. 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
```

7.4. 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
```

7.5. 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ます。

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

クラスターネットワークプラグイン。**OVNKubernetes** は、インストール時にサポートされる唯一のプラグインです。

注記

クラスターをインストールした後は、**clusterNetwork** IP アドレス範囲のみ変更できます。デフォルトのネットワークタイプは、移行時に OpenShift SDN から OVN-Kubernetes にのみ変更できます。

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ラグイン設定を指定できます。

7.5.1. 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7.1 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**	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リストです。以下に例を示します。 spec: clusterNetwork: - cidr: 10.128.0.0/19 hostPrefix: 23 - cidr: 10.128.32.0/19 hostPrefix: 23

フィールド	型	説明
spec.serviceNetwork	**array**	サービスの IP アドレスのブロック。OpenShift SDN および OVN-Kubernetes ネットワークプラグインは、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。 spec: serviceNetwork: - 172.30.0.0/14 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。
spec.defaultNetwork	**object**	クラスターネットワークのネットワークプラグインを設定します。
spec.kubeProxyConfig	**object**	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ラグインを使用している場合、kube-proxy 設定は機能しません。

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7.2 defaultNetwork オブジェクト

フィールド	型	説明
type	**string**	**OVNKubernetes**。Red Hat OpenShift Networking ネットワークプラグインは、インストール中に選択されます。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。 **注記** OpenShift Container Platform は、デフォルトで OVN-Kubernetes ネットワークプラグインを使用します。OpenShift SDN は、新しいクラスターのインストールの選択肢として利用できなくなりました。
ovnKubernetesConfig	**object**	このオブジェクトは、OVN-Kubernetes ネットワークプラグインに対してのみ有効です。

OpenShift SDN ネットワークプラグインの設定
以下の表では、OpenShift SDN ネットワークプラグインの設定フィールドについて説明します。

表7.3 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。

OpenShift SDN 設定の例

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

OVN-Kubernetes ネットワークプラグインの設定

次の表では、OVN-Kubernetes ネットワークプラグインの設定フィールドについて説明します。

表7.4 ovnKubernetesConfig オブジェクト

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**	クラスターの IPsec モードを示すオブジェクト。
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。

フィールド	型	説明
gatewayConfig	**object**	オプション: egress トラフィックのノードゲートウェイへの送信方法をカスタマイズするための設定オブジェクトを指定します。  **注記** egress トラフィックの移行中は、Cluster Network Operator (CNO) が変更を正常にロールアウトするまで、ワークロードとサービストラフィックに多少の中断が発生することが予想されます。
v4InternalSubnet	既存のネットワークインフラストラクチャーが **100.64.0.0/16** IPv4 サブネットと重複している場合は、OVN-Kubernetes による内部使用のために別の IP アドレス範囲を指定できます。IP アドレス範囲が、OpenShift Container Platform インストールで使用される他のサブネットと重複しないようにする必要があります。IP アドレス範囲は、クラスターに追加できるノードの最大数より大きくする必要があります。たとえば、**clusterNetwork.cidr** 値が **10.128.0.0/14** で、**clusterNetwork.hostPrefix** 値が **/23** の場合、ノードの最大数は **2^(23-14)=512** です。 このフィールドは、インストール後に変更できません。	デフォルト値は **100.64.0.0/16** です。

フィールド	型	説明
v6InternalSubnet	既存のネットワークインフラストラクチャーが **fd98::/48** IPv6 サブネットと重複する場合は、OVN-Kubernetes による内部使用のために別の IP アドレス範囲を指定できます。IP アドレス範囲が、OpenShift Container Platform インストールで使用される他のサブネットと重複しないようにする必要があります。IP アドレス範囲は、クラスターに追加できるノードの最大数より大きくする必要があります。 このフィールドは、インストール後に変更できません。	デフォルト値は **fd98::/48** です。

表7.5 policyAuditConfig オブジェクト

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** (50MB) です。
maxLogFiles	integer	保持されるログファイルの最大数。

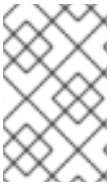
フィールド	型	説明
比較先	string	以下の追加の監査ログターゲットのいずれかになります。 **libc** ホスト上の journald プロセスの libc **syslog()** 関数。 **udp:<host>:<port>** syslog サーバー。**<host>:<port>** を syslog サーバーのホストおよびポートに置き換えます。 **unix:<file>** **<file>** で指定された Unix ドメインソケットファイル。 **null** 監査ログを追加のターゲットに送信しないでください。
syslogFacility	string	RFC5424 で定義される **kern** などの syslog ファシリティー。デフォルト値は **local0** です。

表7.6 gatewayConfig オブジェクト

フィールド	型	説明
routingViaHost	**boolean**	Pod からホストネットワークスタックへの egress トラフィックを送信するには、このフィールドを **true** に設定します。インストールおよびアプリケーションがカーネルルーティングテーブルに手動設定されたルートに依存するなど非常に特化されている場合には、egress トラフィックをホストネットワークスタックにルーティングすることを推奨します。デフォルトでは、egress トラフィックは OVN で処理され、クラスターを終了するために処理され、トラフィックはカーネルルーティングテーブルの特殊なルートによる影響を受けません。デフォルト値は **false** です。 このフィールドで、Open vSwitch ハードウェアオフロード機能との対話が可能になりました。このフィールドを**true**に設定すると、egress トラフィックがホストネットワークスタックで処理されるため、パフォーマンス的に、オフロードによる利点は得られません。
ipForwarding	**object**	**Network** リソースの **ipForwarding** 仕様を使用して、OVN-Kubernetes マネージドインターフェイス上のすべてのトラフィックの IP フォワーディングを制御できます。Kubernetes 関連のトラフィックの IP フォワーディングのみを許可するには、**Restricted** を指定します。すべての IP トラフィックの転送を許可するには、**Global** を指定します。新規インストールの場合、デフォルトは **Restricted** です。OpenShift Container Platform 4.14 以降に更新する場合、デフォルトは **Global** です。

表7.7 ipsecConfig オブジェクト

フィールド	型	説明
mode	**string**	IPsec 実装の動作を指定します。次の値のいずれかである必要があります。 ● **Disabled**: クラスターノードで IPsec が有効になりません。 ● **External**: 外部ホストとのネットワークトラフィックに対して IPsec が有効になります。 ● **Full**: Pod トラフィックおよび外部ホストとのネットワークトラフィックに対して IPsec が有効になります。



注記

クラスターのインストール中にのみクラスターネットワークプラグインの設定を変更できます。ただし、インストール後のアクティビティーとして実行時に変更できる**gatewayConfig**フィールドは除きます。

IPsec が有効な OVN-Kubernetes 設定の例

```
defaultNetwork:
  type: OVNKubernetes
  ovnKubernetesConfig:
    mtu: 1400
    genevePort: 6081
      ipsecConfig:
        mode: Full
```

重要

OVNKubernetes を使用すると、IBM Power® でスタック枯渇の問題が発生する可能性があります。

kubeProxyConfig オブジェクト設定 (OpenShiftSDN コンテナーネットワークインターフェイスのみ)

kubeProxyConfig オブジェクトの値は以下の表で定義されます。

表7.8 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**	**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。 **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。
proxyArguments.iptables-min-sync-period	**array**	**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: kubeProxyConfig: proxyArguments: iptables-min-sync-period: - 0s

7.5.2. 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:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  serviceNetwork:
  - 172.30.0.0/16
  networkType: OVNKubernetes
     clusterNetworkMTU: 8900
```

7.6. 関連情報

- **operator.openshift.io** API グループの **Network** API
- **clusterNetwork** の IP アドレス範囲の変更
- OpenShift SDN ネットワークプラグインからの移行

第8章 OPENSHIFT 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。

8.1. 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 を使用するように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。

8.2. 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できます。

- 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"}}'
```

8.3. 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。セキュリティーポリシーにより、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、実行しないように設定できます。

前提条件

- **oc** CLI 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

- 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 1
```

1 テイントが **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。

8.4. 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 1
  Cluster IP:      172.30.0.10 2
...
```

1 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。

2 クラスター 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. クラスターのサービス 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}'
```

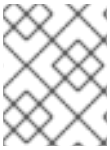
出力例

```
[172.30.0.0/16]
```

8.5. DNS 転送の使用

DNS 転送を使用して、次の方法で**/etc/resolv.conf**ファイルのデフォルトの転送設定を上書きできます。

- すべてのゾーンにネームサーバーを指定します。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。
- アップストリーム DNS サーバーのリストを指定します。
- デフォルトの転送ポリシーを変更します。



注記

デフォルトドメインの DNS 転送設定には、**/etc/resolv.conf** ファイルおよびアップストリーム DNS サーバーで指定されたデフォルトのサーバーの両方を設定できます。

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
```

以前のコマンドを実行した後に、Operator は **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 config map を作成して更新します。クエリーに一致するゾーンがサーバーにない場合には、名前解決はアップストリーム DNS サーバーにフォールバックします。

DNS 転送の設定

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

1. **rfc6335** サービス名の構文に準拠する必要があります。
2. **rfc1123** サービス名構文のサブドメイン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** フィールドの無効なサブドメインです。
3. アップストリームリゾルバーを選択するためのポリシーを定義します。デフォルト値は**Random**です。**RoundRobin** および **Sequential** の値を使用することもできます。
4. **forwardPlugin** ごとに最大 15 の **upstreams** が許可されます。
5. オプション: これを使用して、デフォルトポリシーを上書きし、デフォルトドメインで指定された DNS リゾルバー (アップストリームリゾルバー) に DNS 解決を転送できます。アップストリームリゾルバーを指定しない場合に、DNS 名のクエリーは **/etc/resolv.conf** のサーバーに送信されます。
6. クエリー用にアップストリームサーバーが選択される順序を決定します。**Random**、**Round Robin**、または**Sequential** のいずれかの値を指定できます。デフォルト値は**Sequential**です。
7. オプション: これを使用して、アップストリームリゾルバーを指定できます。
8. **SystemResolvConf**と**Network**の 2 種類の**アップストリーム**を指定できます。**SystemResolvConf** で、アップストリームが **/etc/resolv.conf** を使用するように設定して、**Network** で **Networkresolver** を定義します。1 つまたは両方を指定できます。
9. 指定したタイプが**Network**の場合には、IP アドレスを指定する必要があります。**address** フィールドは、有効な IPv4 または IPv6 アドレスである必要があります。
10. 指定したタイプが **Network** の場合、必要に応じてポートを指定できます。**port** フィールドの値は **1** ～ **65535** である必要があります。アップストリームのポートを指定しない場合、デフォルトでポート 853 が試行されます。

2. 任意: 高度に規制された環境で作業する場合は、要求をアップストリームリゾルバーに転送する際に DNS トラフィックのセキュリティーを確保して、追加の DNS トラフィックおよびデータのプライバシーを確保できるようにする必要がある場合があります。クラスター管理者は、転送された DNS クエリーに Transport Layer Security (TLS) を設定できるようになりました。

TLS を使用した DNS 転送の設定

```
apiVersion: operator.openshift.io/v1
kind: DNS
metadata:
  name: default
spec:
  servers:
  - name: example-server 1
    zones: 2
    - example.com
    forwardPlugin:
      transportConfig:
        transport: TLS 3
        tls:
          caBundle:
            name: mycacert
          serverName: dnstls.example.com 4
```

```
    policy: Random (5)
    upstreams: (6)
    - 1.1.1.1
    - 2.2.2.2:5353
  upstreamResolvers: (7)
   transportConfig:
     transport: TLS
     tls:
       caBundle:
         name: mycacert
       serverName: dnstls.example.com
   upstreams:
   - type: Network (8)
     address: 1.2.3.4 (9)
     port: 53 (10)
```

(1) **rfc6335** サービス名の構文に準拠する必要があります。

(2) **rfc1123** サービス名構文のサブドメイン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** フィールドの無効なサブドメインです。クラスタードメインの **cluster.local** は、 **zones** の無効な **subdomain** です。

(3) 転送された DNS クエリーの TLS を設定する場合、**transport** フィールドの値を **TLS** に設定します。デフォルトでは、CoreDNS は転送された接続を 10 秒間キャッシュします。要求が発行されない場合、CoreDNS はその 10 秒間、TCP 接続を開いたままにします。大規模なクラスターでは、ノードごとに接続を開始できるため、DNS サーバーが多くの新しい接続を開いたまま保持する可能性があることを認識しているか確認してください。パフォーマンスの問題を回避するために、それに応じて DNS 階層を設定します。

(4) 転送された DNS クエリー用に TLS を設定する場合、これは、アップストリーム TLS サーバー証明書を検証するための Server Name Indication (SNI) の一部として使用される必須のサーバー名です。

(5) アップストリームリゾルバーを選択するためのポリシーを定義します。デフォルト値は**Random**です。**RoundRobin** および **Sequential** の値を使用することもできます。

(6) 必須。これを使用して、アップストリームリゾルバーを指定できます。**forwardPlugin** エントリーごとに最大 15 の **upstreams** エントリーが許可されます。

(7) オプション: これを使用して、デフォルトポリシーを上書きし、デフォルトドメインで指定された DNS リゾルバー (アップストリームリゾルバー) に DNS 解決を転送できます。アップストリームリゾルバーを指定しない場合に、DNS 名のクエリーは **/etc/resolv.conf** のサーバーに送信されます。

(8) **Network** タイプは、このアップストリームリゾルバーが **/etc/resolv.conf** にリストされているアップストリームリゾルバーとは別に転送されたリクエストを処理する必要があることを示します。TLS を使用する場合、**Network** タイプのみが許可され、IP アドレスを指定する必要があります。

(9) **address** フィールドは、有効な IPv4 または IPv6 アドレスである必要があります。

(10) オプションでポートを指定できます。**port** の値は **1** ～ **65535** である必要があります。アップストリームのポートを指定しない場合、デフォルトでポート 853 が試行されます。

注記

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

検証

1. config map 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 1.2.3.4:53 {
            policy Random
        }
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  labels:
    dns.operator.openshift.io/owning-dns: default
  name: dns-default
  namespace: openshift-dns
```

1 **forwardPlugin** への変更により、CoreDNS デー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。

8.6. 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
```

8.7. 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
```

8.8. COREDNS ログレベルの設定

CoreDNS ログレベルを設定して、ログに記録されたエラーメッセージの情報量を決定できます。CoreDNS ログレベルの有効な値は、**Normal**、**Debug**、および **Trace** です。デフォルトの**log Level**は**Normal**です。

注記

エラープラグインは常に有効になっています。次の**logLevel**設定は、さまざまなエラー応答を報告します。

- **logLevel**: **Normal**は "errors" class: **log . { class error }** を有効にします。
- **logLevel**: **Debug**は "denial" class: **log . { class denial error }** を有効にします。
- **logLevel**: **Trace**は "all" class: **log . { class all }** を有効にします。

手順

- **logLevel**を**Debug**に設定するには、次のコマンドを入力します。

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

- **logLevel**を**Trace**に設定するには、次のコマンドを入力します。

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

検証

- 目的のログレベルが設定されていることを確認するには、ConfigMap を確認します。

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

8.9. COREDNS OPERATOR のログレベルの設定

クラスター管理者は、Operator ログレベルを設定して、OpenShift DNS の問題をより迅速に追跡できます。**operatorLogLevel**の有効な値は、 **Normal**、**Debug**、および**Trace**です。**Trace** には最も詳細にわたる情報が含まれます。デフォルトの**operatorlogLevel**は**Normal**です。問題のログレベルには、Trace、Debug、Info、Warning、Error、Fatal および Panic の 7 つがあります。ログレベルの設定後に、その重大度またはそれを超える重大度のログエントリーがログに記録されます。

- **operatorLogLevel: "Normal"** は **logrus.SetLogLevel("Info")** を設定します。
- **operatorLogLevel: "Debug"** は **logrus.SetLogLevel("Debug")** を設定します。
- **operatorLogLevel: "Trace"** は **logrus.SetLogLevel("Trace")** を設定します。

手順

- **operatorLogLevel**を**Debug**に設定するには、次のコマンドを入力します。

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

- **operatorLogLevel**を**Trace**に設定するには、次のコマンドを入力します。

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

8.10. COREDNS キャッシュのチューニング

CoreDNS によって実行される成功または失敗したキャッシュ (それぞれポジティブキャッシュまたはネガティブキャッシュとも呼ばれます) の最大期間を設定できます。DNS クエリー応答のキャッシュ期間を調整すると、上流の DNS リゾルバーの負荷を軽減できます。

手順

1. 次のコマンドを実行して、**default** という名前の DNS Operator オブジェクトを編集します。

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

2. Time-to-Live (TTL) キャッシュ値を変更します。

DNS キャッシングの設定

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

1 文字列値 **1h** は、CoreDNS によってそれぞれの秒数に変換されます。このフィールドを省略した場合、値は **0** と見なされ、クラスターはフォールバックとして内部デフォルト値の

2 文字列値は、**0.5h10m** などの単位の組み合わせにすることができ、CoreDNS によってそれぞれの秒数に変換されます。このフィールドを省略した場合、値は **0** と見なされ、クラスターはフォールバックとして内部デフォルト値の **30** を使用します。

警告

TTL フィールドを低い値に設定すると、クラスター、上流のリゾルバー、またはその両方の負荷が増加する可能性があります。

第9章 OPENSHIFT CONTAINER PLATFORM の INGRESS OPERATOR

9.1. OPENSHIFT 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 Controller 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 Controller 内の設定は、Ingress Controller エンドポイントを公開する方法を提供します。

9.2. 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.openshiftdemos.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 Controller 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。

9.3. INGRESS CONTROLLER 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	**domain** は Ingress Controller 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。 ● **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。 **domain** 値はすべての Ingress Controller の中でも固有の値であり、更新できません。 空の場合、デフォルト値は **ingress.config.openshift.io/cluster .spec.domain** です。
replicas	**replicas** は Ingress Controller 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。
endpointPublishingStrategy	**endpointPublishingStrategy** は Ingress Controller 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。 GCP、AWS、および Azure では、次の **endpointPublishingStrategy** フィールドを設定できます。 ● **loadBalancer.scope** ● **loadBalancer.allowedSourceRanges** 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。 ● Azure: **LoadBalancerService** (外部スコープあり) ● Google Cloud Platform (GCP): **LoadBalancerService** (外部スコープあり) ● Bare metal: **NodePortService** ● その他: **HostNetwork**

パラメーター	説明
	注記 **HostNetwork** には、オプションのバインディングポートのデフォルト値が **httpPort:80**、**httpsPort:443**、**statsPort:1936** の **hostNetwork** フィールドがあります。バインディングポートを使用すると、**HostNetwork** ストラテジーの同じノードに複数の Ingress Controller をデプロイできます。 **例** apiVersion: operator.openshift.io/v1 kind: IngressController metadata: name: internal namespace: openshift-ingress-operator spec: domain: example.com endpointPublishingStrategy: type: HostNetwork hostNetwork: httpPort: 80 httpsPort: 443 statsPort: 1936 注記 Red Hat OpenStack Platform (RHOSP) では、クラウドプロバイダーがヘルスモニターを作成するように設定されている場合にのみ、**LoadBalancerService** エンドポイントの公開ストラテジーがサポートされます。RHOSP 16.2 の場合、このストラテジーは Amphora Octavia プロバイダーを使用する場合にのみ可能です。 詳細については、RHOSP インストールドキュメントのクラウドプロバイダーオプションの設定セクションを参照してください。 ほとんどのプラットフォームでは、**endpointPublishingStrategy** 値は更新できます。GCP では、次の**endpointPublishingStrategy**フィールドを設定できます。 - **loadBalancer.scope** - **loadbalancer.providerParameters.gcp.clientAccess** - **hostNetwork.protocol** - **nodePort.protocol**

パラメーター	説明
defaultCertificate	**defaultCertificate** 値は、Ingress Controller 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。 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 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。 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。
namespaceSelector	**namespaceSelector** は、Ingress Controller 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。
routeSelector	**routeSelector** は、Ingress Controller 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。
nodePlacement	**nodePlacement** は、Ingress Controller のスケジュールに対する明示的な制御を有効にします。 設定されていない場合は、デフォルト値が使用されます。  **注記** **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。 `nodePlacement:` `  nodeSelector:` `  matchLabels:` `  kubernetes.io/os: linux` `  tolerations:` `  - effect: NoSchedule` `  operator: Exists`

パラメーター	説明
tlsSecurityProfile	**tlsSecurityProfile** は、Ingress Controller の TLS 接続の設定を指定します。 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。 **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 Controller に適用され、ロールアウトが生じる可能性があります。 Ingress Controller 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。  **注記** 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。  **重要** 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。
clientTLS	**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。 **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。 **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リスト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 Controller は証明書を拒否し、接続を拒否します。指定しないと、Ingress Controller は識別名に基づいて証明書を拒否しません。

パラメーター	説明
routeAdmission	**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。 **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。 ● **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。 **wildcardPolicy** は、ワイルドカードポリシーを使用するルートが Ingress Controller によって処理される方法を記述します。 ● **WildcardsAllowed**: ワイルドカードポリシーと共にルートが Ingress Controller 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 Controller 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 Controller 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	**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。 ● **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。 ○ **destination** はログメッセージの宛先を記述します。 ■ **type** はログの宛先のタイプです。 ● **Container** は、ログがサイドカーコンテナーに移動することを指定します。Ingress Operator は Ingress Controller Pod で **logs** という名前のコンテナーを設定し、Ingress Controller がログをコンテナーに書き込むように設定します。管理者がこのコンテナー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。コンテナーログを使用すると、ログの割合がコンテナー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、コンテナー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。 ● **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。 ● **maxLength**は、syslog メッセージの最大長です。サイズは **480** から **4096** バイトである必要があります。このフィールドが空の場合には、最大長はデフォルト値の**1024**バイトに設定されます。 ● **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
httpHeaders	**httpHeaders** は HTTP ヘッダーのポリシーを定義します。 **IngressControllerHTTPHeaders** の **forwardedHeaderPolicy** を設定することで、Ingress Controller 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。 デフォルトでは、ポリシーは **Append** に設定されます。 ● **Append** は、Ingress Controller 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 Controller 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 Controller がヘッダーを設定するように指定します。 ● **Never** は、Ingress Controller 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。 **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。 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。 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。 **actions** は、ヘッダーに対して特定のアクションを実行するためのオプションを指定します。TLS パススルー接続のヘッダーは設定または削除できません。**actions** フィールドには、**spec.httpHeader.actions.response** および **spec.httpHeader.actions.request** の追加のサブフィールドがあります。 ● **response** サブフィールドは、設定または削除する HTTP 応答ヘッダーのリストを指定します。 ● **request** サブフィールドは、設定または削除する HTTP 要求ヘッダーのリストを指定します。

パラメーター	説明
httpCompression	**http Compression**は、HTTP トラフィック圧縮のポリシーを定義します。 ● **mimeTypes** は、圧縮を適用する必要がある MIME タイプのリストを定義します。(例: **text/css; charset=utf-8**, **text/html**, **text/***, **image/svg+xml**, **application/octet-stream**, **X-custom/customsub**, using the format pattern,**type/subtype; [;attribute=value]**)**types**は、アプリケーション、イメージ、メッセージ、マルチパート、テキスト、ビデオ、または**X-**で始まるカスタムタイプ。例: MIME タイプとサブタイプの完全な表記を確認するには、 RFC1341を参照してください。
httpErrorCodePages	**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。
httpCaptureCookies	**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。 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。 ● **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。 以下に例を示します。 `httpCaptureCookies:` `- matchType: Exact` `  maxLength: 128` `  name: MYCOOKIE`

パラメーター	説明
httpCaptureHeaders	**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。 **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。 `httpCaptureHeaders:` `  request:` `  - maxLength: 256` `    name: Connection` `  - maxLength: 128` `    name: User-Agent` `  response:` `  - maxLength: 256` `    name: Content-Type` `  - maxLength: 256` `    name: Content-Length`
tuningOptions	**tuningOptions** は、Ingress Controller 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。 ● **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。デフォルトのタイムアウトは **1s** です。 ● **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。デフォルトのタイムアウトは **30s** です。 ● **headerBufferBytes** は、Ingress Controller 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 Controller 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推奨しません。**headerBufferBytes** 値が小さすぎると Ingress Controller が破損する可能性があり、**headerBufferBytes** 値が大きすぎると、Ingress Controller 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。 ● **headerBufferMaxRewriteBytes** は、HTTP ヘッダーの書き換えと Ingress Controller 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推奨しません。**headerBufferMaxRewriteBytes** 値が小さすぎると Ingress Controller が破損する可能性があり、**headerBufferMaxRewriteBytes** 値が大きすぎると、Ingress Controller 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。 ● **healthCheckInterval** は、ルーターがヘルスチェックの間隔として待機する時間を指定します。デフォルトは **5s** です。

パラメーター	説明
	● **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。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。デフォルトのタイムアウトは **30s** 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 Controller 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 Controller 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推奨しません。HAProxy スレッドの数を増やすと、Ingress Controller 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 Controller のパフォーマンスが低下する可能性があります。 ● **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。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。デフォルトのタイムアウトは **1h** です。 ● **maxConnections** は、HAProxy プロセスごとに確立できる同時接続の最大数を指定します。この値を増やすと、追加のシステムリソースで各 Ingress Controller Pod がより多くの接続を処理できるようになります。**0**、**-1**、**2000** から **2000000** の 範囲内の任意の値を使用でき、フィールドを空にすることも可能です。 o このフィールドが空のままであるか、値が **0** の場合、Ingress Controller はデフォルト値の **50000** を使用します。この値は、今後のリリースで変更される可能性があります。 o フィールド値が **-1** の場合、HAProxy は、実行中のコンテナーで使用可能な **ulimit** に基づき最大値を動的に計算します。このプロセスにより、計算値が大きくなり、現在のデフォルト値である **50000** と比較してかなり大きなメモリー使用量が発生します。 o フィールドの値が現在のオペレーティングシステムの制限よりも大きい場合、HAProxy プロセスは開始されません。 o 個別の値を選択し、ルーター Pod が新しいノードに移行された場合、新しいノードに同一の **ulimit** が設定されていない可能性があります。このような場合、Pod は起動に失敗します。 o 異なる **ulimit** を持つノードが設定されていて、離散値を選択する場合は、実行時に接続の最大数が計算されるように、このフィールドに **-1** の値を使用することを推奨します。

パラメーター	説明
logEmptyRequests	**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。 **LoggingPolicy** タイプは、以下のいずれかの値を受け入れます。 ● **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	**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。 **HTTPEmptyRequestsPolicy** タイプは、以下のいずれかの値を受け入れます。 ● **応答**: フィールドが **Respond** に設定されている場合、Ingress Controller 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、適切なメトリック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 Controller 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、メトリックを増分します。 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。

注記

すべてのパラメーターはオプションです。

9.3.1. Ingress Controller 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。

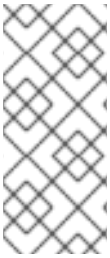
9.3.1.1. TLS セキュリティープロファイルについて

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9.1 TLS セキュリティープロファイル

プロファイル	説明
Old	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。 **Old** プロファイルには、最小 TLS バージョン 1.0 が必要です。  **注記** Ingress Controller 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。
Intermediate	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 Controller 、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。 **Intermediate** プロファイルには、最小 TLS バージョン 1.2 が必要です。
Modern	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。 **Modern** プロファイルには、最小 TLS バージョン 1.3 が必要です。
カスタム	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。  **警告** 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際には注意してください。



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。

9.3.1.2. Ingress Controller の TLS セキュリティープロファイルの設定

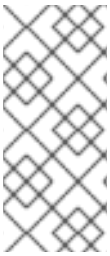
Ingress Controller の TLS セキュリティープロファイルを設定するには、**IngressController** カスタムリソース (CR) を編集して、事前定義済みまたはカスタムの TLS セキュリティープロファイルを指定します。TLS セキュリティープロファイルが設定されていない場合、デフォルト値は API サーバーに設定された TLS セキュリティープロファイルに基づいています。

Old TLS のセキュリティー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 Controller 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、TLS**1.3** 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 (1)
    custom: (2)
      ciphers: (3)
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
 ...
```

(1) TLS セキュリティー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。

(2) 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:**

(3) **custom** タイプには、TLS 暗号のリストと最小許容 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
...
```

9.3.1.3. 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 Controller 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 Controller 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リストも設定できます。

clientCA 値が X509v3 証明書失効リスト (CRL) ディストリビューションポイントを指定している場合、Ingress Operator は、提供された各証明書で指定されている HTTP URI X509v3 **CRL Distribution Point** に基づいて CRL config map をダウンロードおよび管理します。Ingress Controller は、mTLS/TLS ネゴシエーション中にこの config map を使用します。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。
- PEM でエンコードされた CA 証明書バンドルがある。
- CA バンドルが CRL ディストリビューションポイントを参照する場合は、エンドエンティティーまたはリーフ証明書もクライアント CA バンドルに含める必要があります。この証明書には、RFC 5280 で説明されているとおり、この証明書の **CRL Distribution Points** に HTTP URI が含まれている必要があります。以下に例を示します。

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

手順

1. **openshift-config** namespace で、CA バンドルから config map を作成します。

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

1 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$"
```

9.4. デフォルト INGRESS CONTROLLER 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 Controller 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 Controller を表示します。

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

9.5. 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
```

9.6. INGRESS CONTROLLER ログの表示

Ingress Controller ログを表示できます。

手順

- Ingress Controller ログを表示します。

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

9.7. INGRESS CONTROLLER ステータスの表示

特定の Ingress Controller のステータスを表示できます。

手順

- Ingress Controller のステータスを表示します。

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

9.8. INGRESS CONTROLLER の設定

9.8.1. 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 Controller がカスタム証明書を使用するように設定できます。

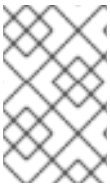
前提条件

- 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
```

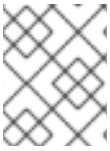


注記

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 証明書をリスト表示します。

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 Controller 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、以下のようになります。

<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 Controller のデプロイメントを更新します。

9.8.2. 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、以下のようになります。

<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
```

9.8.3. Ingress Controller の自動スケーリング

Ingress Controller 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動的に対応するために自動でスケーリングできます。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。

前提条件

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

3. Custom Metrics Autoscaler Operator がインストールされている。
4. **openshift-ingress-operator** プロジェクトの namespace に切り替えている。

手順

1. 以下のコマンドを実行して、Thanos で認証するためのサービスアカウントを作成します。

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

出力例

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

2. サービスアカウントのトークンを使用して、**openshift-ingress-operator** namespace 内で **TriggerAuthentication** オブジェクトを定義します。
   a. 以下のコマンドを実行して、シークレットを含む変数 **secret** を定義します。

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

   b. **TriggerAuthentication** オブジェクトを作成し、**secret** 変数の値を **TOKEN** パラメーターに渡します。

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

3. Thanos からメトリクスを読み取るためのロールを作成して適用します。

a. Pod およびノードからメトリクスを読み取る新規ロール **thanos-metrics-reader.yaml** を作成します。

thanos-metrics-reader.yaml

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

b. 以下のコマンドを実行して新規ロールを適用します。

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

4. 以下のコマンドを入力して、新しいロールをサービスアカウントに追加します。

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

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

注記

引数 **add-cluster-role-to-user** は、namespace 間のクエリーを使用する場合にのみ必要になります。以下の手順では、この引数を必要とする **kube-metrics** namespace からのクエリーを使用します。

5. デフォルトの Ingress Controller デプロイメントをターゲットにする新しい **ScaledObject** YAML ファイル **ingress-autoscaler.yaml** を作成します。

ScaledObject 定義の例

```
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: ingress-scaler
spec:
  scaleTargetRef: 1
    apiVersion: operator.openshift.io/v1
    kind: IngressController
    name: default
    envSourceContainerName: ingress-operator
  minReplicaCount: 1
  maxReplicaCount: 20 2
  cooldownPeriod: 1
  pollingInterval: 1
  triggers:
  - type: prometheus
    metricType: AverageValue
    metadata:
      serverAddress: https://thanos-querier.openshift-monitoring.svc.cluster.local:9091 3
      namespace: openshift-ingress-operator 4
      metricName: 'kube-node-role'
      threshold: '1'
      query: 'sum(kube_node_role{role="worker",service="kube-state-metrics"})' 5
      authModes: "bearer"
    authenticationRef:
      name: keda-trigger-auth-prometheus
```

1. 対象とするカスタムリソース。この場合、Ingress Controller。
2. オプション: レプリカの最大数。このフィールドを省略すると、デフォルトの最大値は 100 レプリカに設定されます。
3. **openshift-monitoring** namespace の Thanos サービスエンドポイント。
4. Ingress Operator namespace。
5. この式は、デプロイされたクラスターに存在するワーカーノードの数に対して評価されます。

重要

namespace 間クエリーを使用している場合は、**serverAddress** フィールドのポート 9092 ではなくポート 9091 をターゲットにする必要があります。また、このポートからメトリクスを読み取るには、昇格した権限が必要です。

6. 以下のコマンドを実行してカスタムリソース定義を適用します。

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

検証

- 以下のコマンドを実行して、デフォルトの Ingress Controller が **kube-state-metrics** クエリーによって返される値に一致するようにスケールアウトされていることを確認します。
  - **grep** コマンドを使用して、Ingress Controller の YAML ファイルでレプリカを検索します。

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

出力例

```
replicas: 3
```

  - **openshift-ingress** プロジェクトで Pod を取得します。

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

出力例

```
NAME                              READY   STATUS    RESTARTS   AGE
router-default-7b5df44ff-l9pmm   2/2     Running   0          17h
router-default-7b5df44ff-s5sl5   2/2     Running   0          3d22h
router-default-7b5df44ff-wwsth   2/2     Running   0          66s
```

関連情報

- ユーザー定義プロジェクトのモニタリングの有効化
- カスタムメトリクスオートスケーラーのインストール
- カスタムメトリクスオートスケーラートリガー認証について
- Configuring the custom metrics autoscaler to use OpenShift Container Platform monitoring
- カスタムメトリクスオートスケーラーの追加方法について

9.8.4. Ingress Controller のスケーリング

Ingress Controller 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 Controller を 3 つのレプリカにスケーリング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
```

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。

9.8.5. 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 Controller 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。

コンテナーロギングは、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 Controller 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のある環境では、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ー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、コンテナー **Container** の宛先に対してログ記録する Ingress Controller 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 Controller 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 Controller Pod 内に **logs** という名前のコンテナーを作成します。

```
$ 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ーを指定できます。以下の例は、**Syslog** 宛先に対してログを記録する Ingress Controller 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 Controller 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
```

サイドカーの使用時に Ingress Controller が HAProxy ログの長さを変更できるようにします。

- **spec.logging.access.destination.type: Syslog** を使用している場合は、**spec.logging.access.destination.syslog.maxLength** を使用します。

```
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
          maxLength: 4096
          port: 10514
```

- **spec.logging.access.destination.type: Container** を使用している場合は、**spec.logging.access.destination.container.maxLength** を使用します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: default
  namespace: openshift-ingress-operator
spec:
  replicas: 2
  logging:
    access:
      destination:
        type: Container
        container:
          maxLength: 8192
```

9.8.6. Ingress Controller 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 Controller 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 Controller がすでに作成されていることを前提とします。

手順

- Ingress Controller 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

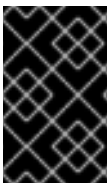
大量のリソースを実行できるノードがある場合、**spec.nodePlacement.nodeSelector** を、意図されているノードの容量に一致するラベルで設定し、**spec.tuningOptions.threadCount** を随時高い値に設定します。

9.8.7. 内部ロードバランサーを使用するための Ingress Controller の設定

クラウドプラットフォームで Ingress Controller を作成する場合、Ingress Controller 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 Controller を作成できます。

警告

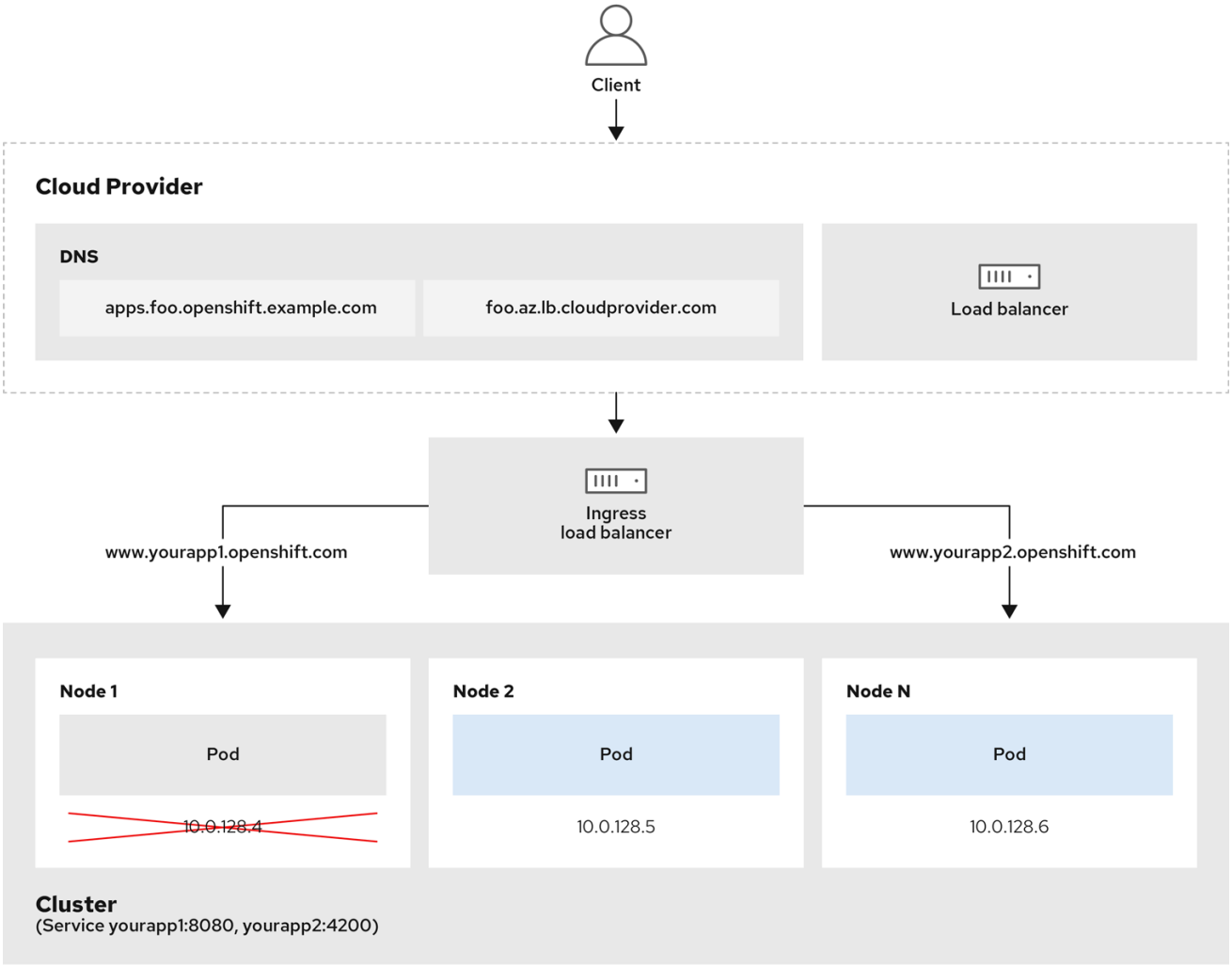
クラウドプロバイダーが Microsoft Azure の場合、ノードを参照するパブリックロードバランサーが少なくとも 1 つ必要です。これがない場合、すべてのノードがインターネットへの egress 接続を失います。



重要

IngressControllerの**scope**を変更する場合は、カスタムリソース (CR) の作成後に**.spec.endpoint Publishing Strategy.load Balancer.scope**パラメーターを変更できます。

図9.1 ロードバランサーの図



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> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: Internal 3
```

1. **<name>** を **IngressController** オブジェクトの名前に置き換えます。
2. 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。
3. 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 Controller を作成します。

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

1. **<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 Controller が作成されていることを確認します。

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

9.8.8. GCP での Ingress Controller 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 Controller 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ト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。

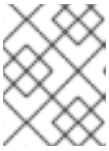
詳細情報は、GCP ドキュメントの グローバルアクセス について参照してください。

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 Controller 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 Controller リソースを設定します。



注記

Ingress Controller 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 Controller 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。

9.8.9. Ingress Controller のヘルスチェック間隔の設定

クラスター管理者は、ヘルスチェックの間隔を設定して、ルーターが連続する 2 回のヘルスチェックの間で待機する時間を定義できます。この値は、すべてのルートのデフォルトとしてグローバルに適用されます。デフォルト値は 5 秒です。

前提条件

- 以下では、Ingress Controller がすでに作成されていることを前提とします。

手順

- Ingress Controller を更新して、バックエンドヘルスチェックの間隔を変更します。

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

注記

単一ルートの **healthCheckInterval** をオーバーライドするには、ルートアノテーション **router.openshift.io/haproxy.health.check.interval** を使用します

9.8.10. クラスターを内部に配置するためのデフォルト Ingress Controller の設定

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 Controller 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の**scope**を変更する場合は、カスタムリソース (CR) の作成後に**.spec.endpoint Publishing Strategy.load Balancer.scope**パラメーターを変更できます。

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. 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 Controller 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
```

9.8.11. 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
```

9.8.12. ワイルドカードルートの使用

HAProxy Ingress Controller 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 Controller の **ROUTER_ALLOW_WILDCARD_ROUTES** 環境変数を設定します。

Ingress Controller 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

1. ワイルドカードポリシーを設定します。

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

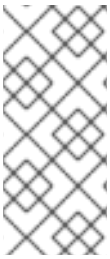
   ```
   $ oc edit IngressController
   ```

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

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

9.8.13. HTTP ヘッダーの設定

OpenShift Container Platform は、HTTP ヘッダーを操作するためのさまざまな方法を提供します。ヘッダーを設定または削除する場合、Ingress Controller の特定のフィールドまたは個々のルートを使用して、リクエストヘッダーと応答ヘッダーを変更できます。ルートアノテーションを使用して特定のヘッダーを設定することもできます。ヘッダーを設定するさまざまな方法は、連携時に課題となる可能性があります。



注記

IngressController または **Route** CR 内のヘッダーは設定または削除のみ可能で、追加はできません。HTTP ヘッダーに値が設定されている場合、その値は完全である必要があるため、今後追加する必要はありません。X-Forwarded-For ヘッダーなどのヘッダーを追加することが適切な状況では、**spec.httpHeaders.actions** の代わりに **spec.httpHeaders.forwardedHeaderPolicy** フィールドを使用します。

9.8.13.1. 優先順位

同じ HTTP ヘッダーを Ingress Controller とルートの両方で変更すると、HAProxy は、それがリクエストヘッダーであるか応答ヘッダーであるかに応じて、特定の方法でアクションの優先順位を付けます。

- HTTP 応答ヘッダーの場合、Ingress Controller で指定されたアクションは、ルートで指定されたアクションの後に実行されます。これは、Ingress Controller で指定されたアクションが優先されることを意味します。
- HTTP リクエストヘッダーの場合、ルートで指定されたアクションは、Ingress Controller で指定されたアクションの後に実行されます。これは、ルートで指定されたアクションが優先されることを意味します。

たとえば、クラスター管理者は、次の設定を使用して、Ingress Controller で X-Frame-Options 応答ヘッダーに値 **DENY** を設定します。

IngressController 仕様の例

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

```
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: DENY
```

ルート所有者は、クラスター管理者が Ingress Controller に設定したものと同じ応答ヘッダーを設定しますが、次の設定を使用して値 **SAMEORIGIN** を設定します。

Route 仕様の例

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

IngressController 仕様と **Route** 仕様の両方で X-Frame-Options ヘッダーを設定している場合、特定のルートでフレームが許可されている場合でも、Ingress Controller のグローバルレベルでこのヘッダーに設定された値が優先されます。

この優先順位付けは、**haproxy.config** ファイルが次のロジックを使用するために発生します。このロジックでは、Ingress Controller がフロントエンドとみなされ、個々のルートがバックエンドとみなされます。フロントエンド設定に適用されるヘッダー値 **DENY** は、バックエンドで設定されている値 **SAMEORIGIN** で同じヘッダーをオーバーライドします。

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

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

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

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

さらに、Ingress Controller またはルートのいずれかで定義されたアクションは、ルートアノテーションを使用して設定された値をオーバーライドします。

9.8.13.2. 特殊なケースのヘッダー

次のヘッダーは、設定または削除が完全に禁止されているか、特定の状況下で許可されています。

表9.2 特殊な場合のヘッダー設定オプション

ヘッダー名	IngressController 仕様を使用して設定可能かどうか	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか
proxy	いいえ	いいえ	**プロキシー** HTTP リクエストヘッダーを使用して、ヘッダー値を **HTTP_PROXY** 環境変数に挿入して、脆弱な CGI アプリケーションを悪用できます。**プロキシー** HTTP リクエストヘッダーも標準ではないため、設定中にエラーが発生しやすくなります。	いいえ
host	いいえ	はい	**IngressController** CR を使用して **ホスト** HTTP 要求ヘッダーが設定されている場合、HAProxy は正しいルートを検索するときに失敗する可能性があります。	いいえ
strict-transport-security	いいえ	いいえ	**strict-transport-security** HTTP 応答ヘッダーはルートアノテーションを使用してすでに処理されているため、別の実装は必要ありません。	はい: **haproxy.router.openshift.io/hsts_header** ルートアノテーション

ヘッダー名	IngressController 仕様を使用して設定可能かどうか	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか
cookie と **set-cookie**	いいえ	いいえ	HAProxy が設定する Cookie は、クライアント接続を特定のバックエンドサーバーにマップするセッション追跡に使用されます。これらのヘッダーの設定を許可すると、HAProxy のセッションアフィニティーが妨げられ、HAProxy の Cookie の所有権が制限される可能性があります。	はい: • **haproxy.router.openshift.io/disable_cookie** ルートアノテーション • **haproxy.router.openshift.io/cookie_name** ルートアノテーション

9.8.14. Ingress Controller での HTTP リクエストおよびレスポンスヘッダーの設定または削除

コンプライアンス目的またはその他の理由で、特定の HTTP 要求および応答ヘッダーを設定または削除できます。これらのヘッダーは、Ingress Controller によって提供されるすべてのルート、または特定のルートに対して設定または削除できます。

たとえば、相互 TLS を使用するようにクラスター上で実行されているアプリケーションを移行する場合があります。このような場合、お使いのアプリケーションで X-Forwarded-Client-Cert リクエストヘッダーをチェックする必要がありますが、OpenShift Container Platform のデフォルトの Ingress Controller は X-SSL-Client-Der リクエストヘッダーを提供します。

次の手順では、Ingress Controller を変更して X-Forwarded-Client-Cert リクエストヘッダーを設定し、X-SSL-Client-Der リクエストヘッダーを削除します。

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 OpenShift Container Platform クラスターにアクセスできる。

手順

1. Ingress Controller リソースを編集します。

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

2. X-SSL-Client-Der HTTP リクエストヘッダーは X-Forwarded-Client-Cert HTTP リクエストヘッダーに置き換えます。

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

1. HTTP ヘッダーに対して実行するアクションのリスト。
2. 変更するヘッダーのタイプ。この場合はリクエストヘッダーです。
3. 変更するヘッダーの名前。設定または削除できる使用可能なヘッダーのリストについては、**HTTP ヘッダーの設定** を参照してください。
4. ヘッダーに対して実行されるアクションのタイプ。このフィールドには、**Set** または **Delete** の値を指定できます。
5. HTTP ヘッダーの設定時は、**値** を指定する必要があります。値は、そのヘッダーで使用可能なディレクティブのリストからの文字列 (例: **DENY)** にすることも、HAProxy の動的値構文を使用して解釈される動的値にすることもできます。この場合、動的な値が追加されます。

注記

HTTP 応答の動的ヘッダー値を設定する場合、サンプルフェッチャーとして **res.hdr** および **ssl_c_der** を使用できます。HTTP リクエストの動的ヘッダー値を設定する場合、許可されるサンプルフェッチャーは **req.hdr** および **ssl_c_der** です。リクエストとレスポンスの両方の動的値で、**lower** コンバーターと **Base64** コンバーターを使用できます。

3. 変更を適用するためにファイルを保存します。

9.8.15. 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 Controller を設定します。Ingress Operator は **HTTPHeaders** フィールドを使用して、Ingress Controller の **ROUTER_SET_FORWARDED_HEADERS** 環境変数を設定します。

手順

1. Ingress Controller 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 Controller 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ーを設定します。
  ヘッダーを変更せずに渡すように Ingress Controller を設定するには、**never** ポリシーを指定します。これにより、Ingress Controller 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ーが提供するヘッダーのみを受信します。

- 外部プロキシー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 Controller を設定します。
  外部プロキシー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 Controller を設定するには、**if-none** ポリシーを指定します。外部プロキシー経由で HTTP 要求にヘッダーがすでに設定されている場合、Ingress Controller はこれを保持します。要求がプロキシーを通過していないためにヘッダーがない場合、Ingress Controller 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ーを設定します。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 Controller 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 Controller 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。

9.8.16. 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、バイナリーストリームなど、HTTP/2 プロトコル機能を使用できます。

個別の Ingress Controller 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。

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 Controller に接続して HTTP/1.1 をネゴシエートし、Ingress Controller 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 Controller 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と問題が発生します。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 Controller で HTTP/2 を有効にします。

- Ingress Controller 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 Controller 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"
```

9.8.17. Ingress Controller の PROXY プロトコルの設定

クラスター管理者は、Ingress Controller 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 Controller 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 Controller 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。この制限があるのは、OpenShift Container Platform がクラウドプラットフォームで実行され、IngressController がサービスロードバランサーを使用するように指定している場合に、Ingress Operator が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です。

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 Controller ではサポートされていません。

前提条件

- Ingress Controller を作成している。

手順

1. Ingress Controller 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 Controller 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 Controller 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
```

9.8.18. 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となるものを指定できます。**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

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
```

9.8.19. HTTP ヘッダーケースの変換

HAProxy 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** は **host: xyz.com** に変更されます。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 には HAProxy 2.6 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、Ingress Controller 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 Controller は **host** 要求ヘッダーを指定どおりに調整します。

- Ingress Controller 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 Controller 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 Controller 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。

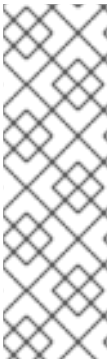
ルート YAML のサンプル

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

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

9.8.20. ルーター圧縮の使用

特定の MIME タイプに対してルーター圧縮をグローバルに指定するように HAProxy Ingress Controller を設定します。**mimeTypes**変数を使用して、圧縮が適用される MIME タイプの形式を定義できます。タイプは、アプリケーション、イメージ、メッセージ、マルチパート、テキスト、ビデオ、または X- で始まるカスタムタイプです。MIME タイプとサブタイプの完全な表記を確認するには、RFC1341を参照してください。



注記

圧縮用に割り当てられたメモリーは、最大接続数に影響を与える可能性があります。さらに、大きなバッファーを圧縮すると、正規表現による負荷が多い場合や正規表現のリストが長い場合など、レイテンシーが発生する可能性があります。

すべての MIME タイプが圧縮から利点を得るわけではありませんが、HAProxy は、指示された場合でもリソースを使用して圧縮を試みます。一般に、html、css、js などのテキスト形式は圧縮から利点を得ますが、イメージ、音声、ビデオなどのすでに圧縮済みの形式は、圧縮に時間とリソースが費やされるわりに利点はほぼありません。

手順

1. Ingress Controller の**httpCompression**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 -n openshift-ingress-operator ingresscontrollers/default
   ```

   b. **spec**で、 **httpCompression** ポリシーフィールドを**mimeTypes** に設定し、圧縮を適用する必要がある MIME タイプのリストを指定します。

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

9.8.21. ルーターメトリクスの公開

デフォルトで、HAProxy ルーターメトリクスをデフォルトの stats ポート (1936) に Prometheus 形式で公開できます。Prometheus などの外部メトリクス収集および集約システムは、HAProxy ルーターメメトリクスにアクセスできます。HAProxy ルーターメトリクスは、HTML およびコンマ区切り値 (CSV) 形式でブラウザーに表示できます。

前提条件

- ファイアウォールを、デフォルトの stats ポート (1936) にアクセスするように設定している。

手順

1. 次のコマンドを実行して、ルーター Pod 名を取得します。

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

   出力例

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

2. ルーター Pod が **/var/lib/haproxy/conf/metrics-auth/statsUsername** および **/var/lib/haproxy/conf/metrics-auth/statsPassword** ファイルに保存しているルーターのユーザー名およびパスワードを取得します。

   a. 次のコマンドを実行して、ユーザー名を取得します。

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

   b. 次のコマンドを実行して、パスワードを取得します。

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

3. 次のコマンドを実行して、ルーター IP およびメトリクス証明書を取得します。

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

4. つぎのコマンドを実行して、Prometheus 形式で未加工の統計情報を取得します。

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

5. 次のコマンドを実行して、安全にメトリクスにアクセスします。

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

6. 次のコマンドを実行して、デフォルトの stats ポート (1936) にアクセスします。

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

例9.1 出力例

```
...
# HELP haproxy_backend_connections_total Total number of connections.
# TYPE haproxy_backend_connections_total gauge
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route-alt"} 0
haproxy_backend_connections_total{backend="http",namespace="default",route="hello-route01"} 0
...
# HELP haproxy_exporter_server_threshold Number of servers tracked and the current threshold value.
# TYPE haproxy_exporter_server_threshold gauge
haproxy_exporter_server_threshold{type="current"} 11
haproxy_exporter_server_threshold{type="limit"} 500
...
# HELP haproxy_frontend_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_frontend_bytes_in_total gauge
haproxy_frontend_bytes_in_total{frontend="fe_no_sni"} 0
haproxy_frontend_bytes_in_total{frontend="fe_sni"} 0
```

```
haproxy_frontend_bytes_in_total{frontend="public"} 119070
...
# HELP haproxy_server_bytes_in_total Current total of incoming bytes.
# TYPE haproxy_server_bytes_in_total gauge
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_no_sni",service="
"} 0
haproxy_server_bytes_in_total{namespace="",pod="",route="",server="fe_sni",service=""}
0
haproxy_server_bytes_in_total{namespace="default",pod="docker-registry-5-
nk5fz",route="docker-registry",server="10.130.0.89:5000",service="docker-registry"} 0
haproxy_server_bytes_in_total{namespace="default",pod="hello-rc-vkjqx",route="hello-
route",server="10.130.0.90:8080",service="hello-svc-1"} 0
...
```

7. ブラウザーで以下の URL を入力して、stats ウィンドウを起動します。

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

8. オプション: ブラウザーに次の URL を入力して、CSV 形式で統計情報を取得します。

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

9.8.22. 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 Controller 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。

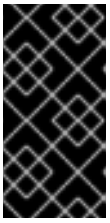
注記

OpenShift Container Platform のデフォルトの 503 エラーコードページをカスタマイズのテンプレートとして使用する場合、ファイル内のヘッダーで CRLF 改行コードを使用できる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、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。

2. Ingress Controller 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。

3. 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。

4. 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。

1. 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、ブラウザー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、ブラウザー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
```

9.8.23. Ingress Controller の最大接続数の設定

クラスター管理者は、OpenShift ルーターデプロイメントの同時接続の最大数を設定できます。既存の Ingress Controller にパッチを適用して、接続の最大数を増やすことができます。

前提条件

- 以下では、Ingress Controller が作成済みであることを前提とします。

手順

- Ingress Controller を更新して、HAProxy の最大接続数を変更します。

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

警告

spec.tuningOptions.maxConnections の値を現在のオペレーティングシステムの制限よりも大きく設定すると、HAProxy プロセスは開始しません。このパラメーターの詳細は、Ingress Controller 設定パラメーターセクションの表を参照してください。

9.9. 関連情報

- カスタム PKI の設定

第10章 OPENSHIFT CONTAINER PLATFORM での INGRESS シャーディング

OpenShift Container Platform では、Ingress Controller はすべてのルートを提供することも、ルートのサブセットを提供することもできます。デフォルトでは、Ingress Controller は、クラスター内の任意の namespace で作成されたすべてのルートを提供します。別の Ingress Controller をクラスターに追加して、選択した特性に基づくルートのサブセットである **シャード** を作成することにより、ルーティングを最適化できます。ルートをシャードのメンバーとしてマークするには、ルートまたは namespace の **メタデータ** フィールドでラベルを使用します。Ingress Controller は、**選択式** とも呼ばれる **セレクター** を使用して、ルートのプール全体からルートのサブセットを選択し、サービスを提供します。

Ingress シャーディングは、受信トラフィックを複数の Ingress Controller 間で負荷分散する場合に、トラフィックを分離して特定の Ingress Controller にルーティングする場合、または次のセクションで説明する他のさまざまな理由で役立ちます。

デフォルトでは、各ルートはクラスターのデフォルトドメインを使用します。ただし、代わりにルーターのドメインを使用するようにルートを設定できます。詳細は、Ingress Controller シャーディングのルートの作成 を参照してください。

10.1. INGRESS CONTROLLER のシャード化

Ingress シャーディング (ルーターシャーディングとも呼ばれます) を使用して、ルート、namespace、またはその両方にラベルを追加することで、一連のルートを複数のルーターに分散できます。Ingress Controller は、対応する一連のセレクターを使用して、指定されたラベルが含まれるルートのみを許可します。各 Ingress シャードは、特定の選択式を使用してフィルタリングされたルートで設定されます。

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 Controller 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 Controller 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 Controller 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。
- blue green デプロイ中に、アプリケーションの別のバージョンにトラフィックを転送します。

Ingress Controller がシャーディングされると、特定のルートがグループ内の 0 個以上の Ingress Controller に受け入れられます。ルートのステータスは、Ingress Controller がルートを受け入れたかどうかを示します。Ingress Controller は、ルートがそのシャードに固有である場合にのみルートを受け入れます。

Ingress Controller は、次の 3 つのシャーディング方法を使用できます。

- namespace セレクターとラベルが同じ namespace 内のすべてのルートが Ingress シャードに含まれるように、namespace セレクターのみを Ingress Controller に追加します。
- Ingress Controller にルートセレクターのみを追加して、ルートセレクターとラベルが同じ全ルートが Ingress シャードに含まれるようにします。

- namespace セレクターとラベルが同じ namespace 内のルートセレクターのラベルがルートと同じ場合に、Ingress シャード内に含まれるように、namespace セレクターとルートセレクターの両方を Ingress Controller に追加します。

シャーディングを使用すると、ルートのサブセットを複数の Ingress Controller に分散できます。これらのサブセットは、重複なし (**従来** のシャーディングとも呼ばれる) にすることも、重複 (**重複** シャーディングとも呼ばれる) にすることもできます。

10.1.1. 従来のシャーディングの例

Ingress Controller の **finops-router** は、ラベルセレクター **spec.namespaceSelector.matchLabels.name** を **finance** および **ops** に指定して設定されます。

finops-router の YAML 定義の例

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

2 番目の Ingress Controller **dev-router** は、ラベルセレクター **spec.namespaceSelector.matchLabels.name** を **dev** に指定して設定されます。

dev-router の YAML 定義の例

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

すべてのアプリケーションルートが個別の namespace にあり、それぞれに **name:finance**、**name:ops**、および **name:dev** というラベルが付けられている場合、この設定は 2 つの Ingress Controller 間でルートを効果的に分散します。コンソール、認証、およびその他の目的の OpenShift Container Platform ルートは処理しないでください。

上記のシナリオでは、シャード化は重複するセットを持たないパーティション設定の特別なケースとなります。ルートは複数のルーターシャード間で分割されます。

警告

デフォルト の Ingress Controller は、 **namespaceSelector** または **routeSelector** フィールドに除外対象のルートが含まれていない限り、引き続きすべてのルートを提供します。デフォルトの Ingress Controller からルートを除外する方法の詳細は、この Red Hat ナレッジベースのソリューション と「デフォルトの Ingress Controller のシャーディング」のセクションを参照してください。

10.1.2. 重複シャーディングの例

上記の例の **finops-router** と **dev-router** に加えて、ラベルセレクター **spec.namespaceSelector.matchLabels.name** を **dev** と **ops** に指定して設定された **devops-router** もあります。

Devops-router の YAML 定義の例

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

name:dev および **name:ops という** 名前の namespace のルートは、2 つの異なる Ingress Controller によって処理されるようになりました。この設定では、ルートのサブセットが重複しています。

重複するルートのサブセットを使用すると、より複雑なルーティングルールを作成できます。たとえば、優先度の低いトラフィックを **devops-router** に送信しながら、優先度の高いトラフィックを専用の **finops-router** に迂回させることができます。

10.1.3. デフォルトの Ingress Controller のシャーディング

新しい Ingress シャードを作成した後に、デフォルトの Ingress Controller と、新しい Ingress シャードの両方により許可されるルートが存在する場合があります。これは、デフォルトの Ingress Controller にセレクターがなく、デフォルトですべてのルートを許可するためです。

namespace セレクターまたはルートセレクターを使用して、Ingress Controller が特定のラベルが割り当てられたルートの処理を制限できます。次の手順では、namespace セレクターを使用して、デフォルトの Ingress Controller が新しく分割された **finance**、**ops**、および **dev** ルートにサービスを提供しないように制限します。これにより、Ingress シャードがさらに分離されます。

重要

OpenShift Container Platform のすべての管理ルートを同じ Ingress Controller で保持する必要があります。したがって、これらの重要なルートを除外するセレクターをデフォルトのIngress Controller に追加することは避けてください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- プロジェクト管理者としてログインしている。

手順

1. 次のコマンドを実行して、デフォルトのIngress Controller を変更します。

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

2. Ingress Controller を編集して、**finance**、**ops**、および **dev** ラベルのいずれかを持つルートを除外する **namespaceSelector** を含めます。

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

デフォルトの Ingress Controller では、**name:finance**、**name:ops**、および **name:dev** という名前の namespace が提供されなくなります。

10.1.4. Ingress シャーディングと DNS

クラスター管理者は、プロジェクト内のルーターごとに個別の DNS エントリーを作成します。ルーターは不明なルートを別のルーターに転送することはありません。

以下の例を考慮してください。

- Router A はホスト 192.168.0.5 にあり、***.foo.com** のルートを持つ。
- Router B はホスト 192.168.1.9 にあり、***.example.com** のルートを持つ。

個別の DNS エントリーは、***.foo.com** をルーター A をホストするノードに解決し、***.example.com** をルーター B をホストするノードに解決する必要があります。

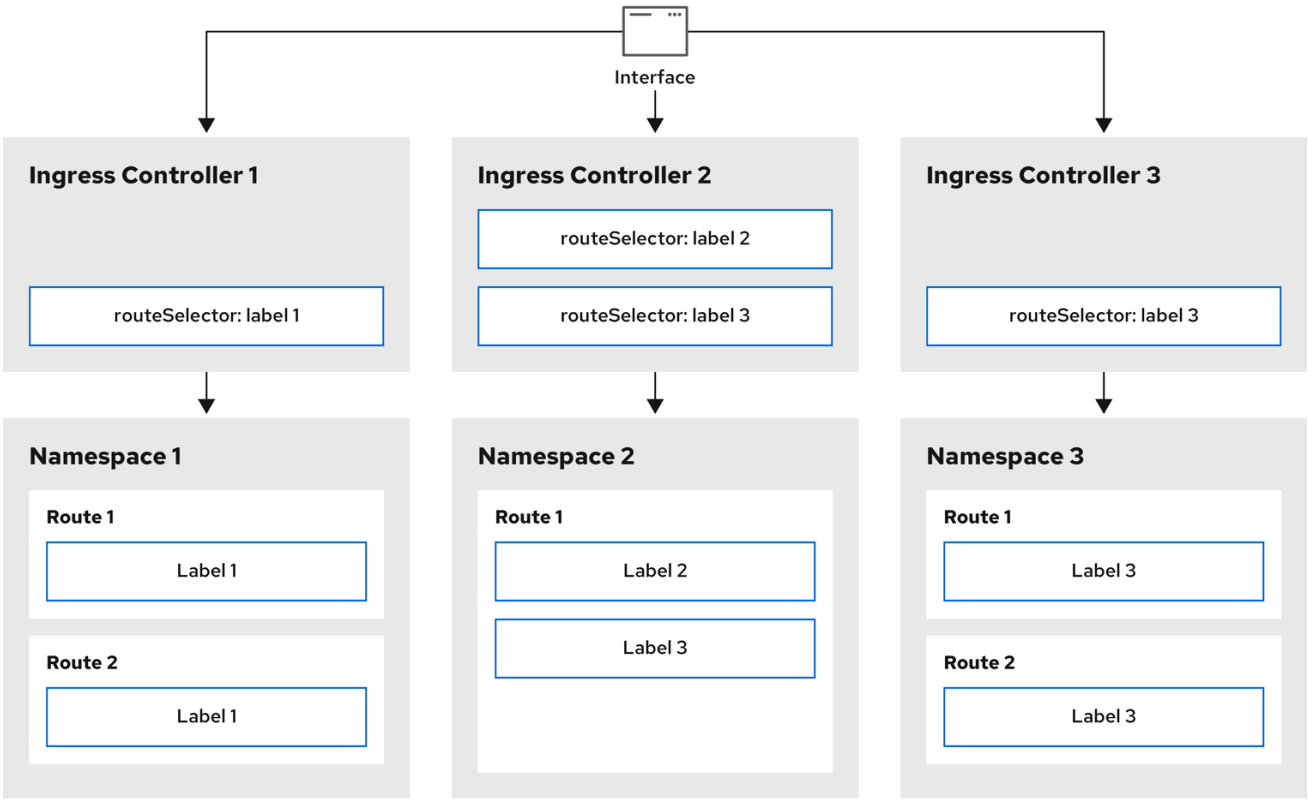
- ***.foo.com A IN 192.168.0.5**

- ***.example.com A IN 192.168.1.9**

10.1.5. ルートラベルを使用した Ingress Controller のシャード化の設定

ルートラベルを使用した Ingress Controller のシャード化とは、Ingress Controller 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。

図10.1 ルートラベルを使用した Ingress シャーディング



Ingress Controller のシャード化は、一連の Ingress Controller 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 Controller 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 Controller に指定し、Company B を別の Ingress Controller 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: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> (1)
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
```

```
  routeSelector:
    matchLabels:
      type: sharded
```

1 Ingress Controller が使用するドメインを指定します。このドメインは、デフォルトの Ingress Controller ドメインとは異なる必要があります。

2. Ingress Controller の **router-internal.yaml** ファイルを適用します。

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

Ingress Controller は、**type: sharded** というラベルのある namespace のルートを選択します。

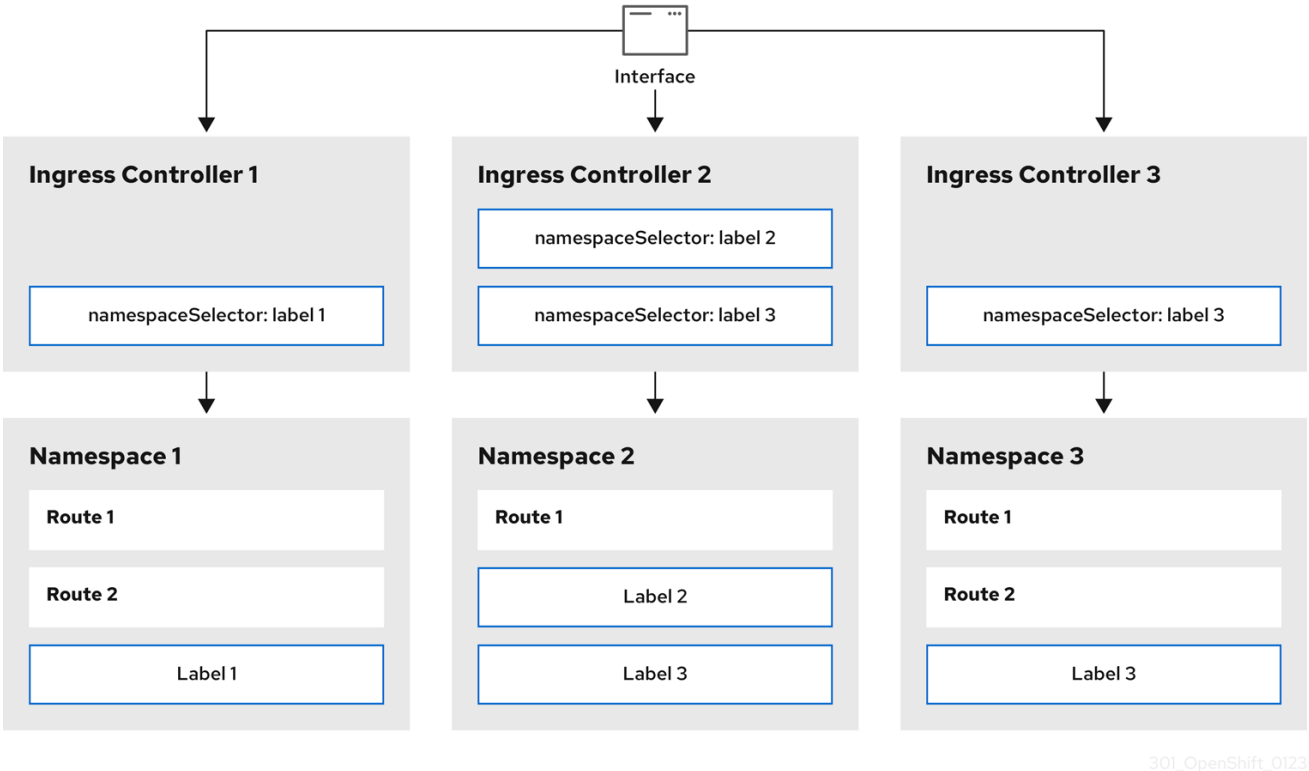
3. **router-internal.yaml** で設定されたドメインを使用して新しいルートを作成します。

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

10.1.6. namespace ラベルを使用した Ingress Controller のシャード化の設定

namespace ラベルを使用した Ingress Controller のシャード化とは、Ingress Controller 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。

図10.2 namespace ラベルを使用した Ingress シャーディング



Ingress Controller のシャード化は、一連の Ingress Controller 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 Controller 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 Controller に指定し、Company B を別の Ingress Controller 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
```

出力例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
```

1 Ingress Controller が使用するドメインを指定します。このドメインは、デフォルトの Ingress Controller ドメインとは異なる必要があります。

2. Ingress Controller の **router-internal.yaml** ファイルを適用します。

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

Ingress Controller 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。

3. **router-internal.yaml** で設定されたドメインを使用して新しいルートを作成します。

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

10.2. INGRESS CONTROLLER シャーディングのルート作成

ルートを使用すると、URL でアプリケーションをホストできます。この場合、ホスト名は設定されず、ルートは代わりにサブドメインを使用します。サブドメインを指定すると、ルートを公開する Ingress Controller のドメインが自動的に使用されます。ルートが複数の Ingress Controller によって公開されている状況では、ルートは複数の URL でホストされます。

以下の手順では、例として **hello-openshift** アプリケーションを使用して、Ingress Controller シャーディングのルートを作成する方法について説明します。

Ingress Controller のシャード化は、一連の Ingress Controller 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 Controller 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 Controller に指定し、Company B を別の Ingress Controller に指定できます。

前提条件

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

- プロジェクト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 HTTP または TCP エンドポイントがある。
- シャーディング用に Ingress Controller を設定している。

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. **hello-openshift-route.yaml** というルート定義を作成します。

シャーディング用に作成されたルートの YAML 定義:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded (1)
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift (2)
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

(1) ラベルキーとそれに対応するラベル値の両方が、Ingress Controller で指定されたものと一致する必要があります。この例では、Ingress Controller にはラベルキーと値 **type: sharded** があります。

(2) ルートは、**subdomain** フィールドの値を使用して公開されます。**subdomain** フィールドを指定するときは、ホスト名を未設定のままにしておく必要があります。**host** フィールドと **subdomain** フィールドの両方を指定すると、ルートは **host** フィールドの値を使用し、**subdomain** フィールドを無視します。

5. 次のコマンドを実行し、**hello-openshift-route.yaml** を使用して **hello-openshift** アプリケーションへのルートを作成します。

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

検証

- 次のコマンドを使用して、ルートのステータスを取得します。

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

結果の **Route** リソースは次のようになります。

出力例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1. Ingress Controller またはルーターがルートを公開するために使用するホスト名。**host** フィールドの値は、Ingress Controller によって自動的に決定され、そのドメインを使用します。この例では、Ingress Controller のドメインは **<apps-sharded.basedomain.example.net>** です。
2. Ingress Controller のホスト名。
3. Ingress Controller の名前。この例では、Ingress Controller の名前は **sharded** です。

関連情報

- ベースライン Ingress Controller (ルーター) のパフォーマンス

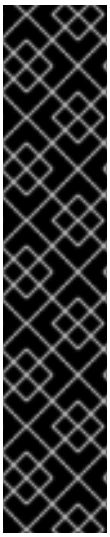
第11章 OPENSHIFT CONTAINER PLATFORM の INGRESS NODE FIREWALL OPERATOR

Ingress Node Firewall Operator を使用すると、管理者はノードレベルでファイアウォール設定を管理できます。

11.1. INGRESS NODE FIREWALL OPERATOR

Ingress Node Firewall Operator は、ファイアウォール設定で指定および管理するノードにデーモンセットをデプロイすることにより、ノードレベルで ingress ファイアウォールルールを提供します。デーモンセットをデプロイするには、**IngressNodeFirewallConfig** カスタムリソース (CR) を作成します。Operator は **IngressNodeFirewallConfig** CR を適用して、**nodeSelector** に一致するすべてのノードで実行されるingress ノードファイアウォールデーモンセット (**daemon**) を作成します。

IngressNodeFirewall CR の **rule** を設定し、**nodeSelector** を使用して値を true に設定してクラスターに適用します。



重要

Ingress Node Firewall Operator は、ステートレスファイアウォールルールのみをサポートします。

ネイティブ XDP ドライバーをサポートしないネットワークインターフェイスコントローラー (NIC) は、より低いパフォーマンスで実行されます。

OpenShift Container Platform 4.14 以降の場合は、RHEL 9.0 以降で Ingress Node Firewall Operator を実行する必要があります。

Ingress Node Firewall Operator は、デフォルトの OpenShift インストールを備えた Amazon Web Services (AWS) または Red Hat OpenShift Service on AWS (ROSA) ではサポートされていません。Red Hat OpenShift Service on AWS のサポートと Ingress の詳細は、Red Hat OpenShift Service on AWS の Ingress Operator を参照してください。

11.2. INGRESS NODE FIREWALL 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 Ingress Node Firewall Operator をインストールできます。

11.2.1. CLI を使用した Ingress Node Firewall 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

- OpenShift CLI (**oc**) がインストールされている。
- 管理者権限を持つアカウントを持っています。

手順

1. **openshift-ingress-node-firewall** namespaceを作成するには、次のコマンドを入力します。

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

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/enforce-version: v1.24
  name: openshift-ingress-node-firewall
EOF
```

2. **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: ingress-node-firewall-operators
  namespace: openshift-ingress-node-firewall
EOF
```

3. Ingress Node Firewall Operator にサブスクライブします。

   a. Ingress Node Firewall Operator の **Subscription** CR を作成するには、次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: ingress-node-firewall-sub
  namespace: openshift-ingress-node-firewall
spec:
  name: ingress-node-firewall
  channel: stable
  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 ip -n openshift-ingress-node-firewall
```

出力例

```
NAME            CSV                                        APPROVAL    APPROVED
install-5cvnz   ingress-node-firewall.4.15.0-202211122336   Automatic   true
```

5. Operator のバージョンを確認するには、次のコマンドを入力します。

```
$ oc get csv -n openshift-ingress-node-firewall
```

出力例

```
NAME                                        DISPLAY                          VERSION          REPLACES
```

```
PHASE
ingress-node-firewall.4.15.0-202211122336   Ingress Node Firewall Operator   4.15.0-
202211122336   ingress-node-firewall.4.15.0-202211102047   Succeeded
```

11.2.2. Web コンソールを使用した Ingress Node Firewall Operator のインストール

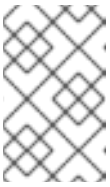
クラスター管理者は、Web コンソールを使用して Operator をインストールできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 管理者権限を持つアカウントを持っています。

手順

1. Ingress Node Firewall 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 のリストから **Ingress Node Firewall Operator** を選択し、**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. Ingress Node Firewall 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. **Ingress Node Firewall Operator** が **openshift-ingress-node-firewall** プロジェクトにリストされ、**Status** が **InstallSucceeded** であ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 の **Status** が **InstallSucceeded** でない場合は、次の手順を使用してトラブルシューティングを行います。

- **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-ingress-node-firewall** プロジェクトの 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-ingress-node-firewall
workload.openshift.io/allowed=management
```

注記

単一ノードの OpenShift クラスターの場合、**openshift-ingress-node-firewall** namespaceには **workload.openshift.io/allowed=management** アノテーションが必要です。

11.3. INGRESS NODE FIREWALL OPERATOR のデプロイ

前提条件

- Ingress Node Firewall Operator がインストールされます。

手順

Ingress Node Firewall Operator をデプロイするには、Operator のデーモンセットをデプロイする **IngressNodeFirewallConfig** カスタムリソースを作成します。ファイアウォールルールを適用することで、1 つまたは複数の **IngressNodeFirewall** CRD をノードにデプロイできます。

1. **ingressnodefirewallconfig** という名前の **openshift-ingress-node-firewall** namespace内に **IngressNodeFirewallConfig** を作成します。
2. 次のコマンドを実行して、Ingress Node Firewall Operator ルールをデプロイします。

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

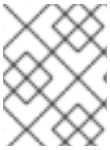
11.3.1. ingress ノードファイアウォール設定オブジェクト

Ingress Node Firewall 設定オブジェクトのフィールドについて、次の表で説明します。

表11.1 ingress ノードファイアウォール設定オブジェクト

フィールド	型	説明
metadata.name	**string**	CR オブジェクトの名前。ファイアウォールルールオブジェクトの名前は **ingressnodefirewallconfig** である必要があります。
metadata.namespace	**string**	Ingress Firewall Operator CR オブジェクトの namespace。**IngressNodeFirewallConfig** CR は、**openshift-ingress-node-firewall** namespace内に作成する必要があります。

フィールド	型	説明
spec.nodeSelector	**string**	指定されたノードラベルを介してノードをターゲットにするために使用されるノード選択制約。以下に例を示します。 spec: nodeSelector: node-role.kubernetes.io/worker: ""  **注記** デーモンセットを開始するには、**nodeSelector** で使用される 1 つのラベルがノードのラベルと一致する必要があります。たとえば、ノードラベル **node-role.kubernetes.io/worker** および **node-type.kubernetes.io/vm** がノードに適用される場合、デーモンセットを開始するには、**nodeSelector** を使用して少なくとも 1 つのラベルを設定する必要があります。



注記

Operator は CR を使用し、**nodeSelector** に一致するすべてのノード上に Ingress ノードファイアウォールデーモンセットを作成します。

Ingress Node Firewall Operator の設定例

次の例では、完全な Ingress ノードファイアウォール設定が指定されています。

ingress ノードファイアウォール設定オブジェクトの例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewallConfig
metadata:
  name: ingressnodefirewallconfig
  namespace: openshift-ingress-node-firewall
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

注記

Operator は CR を使用し、**nodeSelector** に一致するすべてのノード上に Ingress ノードファイアウォールデーモンセットを作成します。

11.3.2. ingress ノードファイアウォールルールオブジェクト

ingress ノードファイアウォールルールオブジェクトのフィールドについて、次の表で説明します。

表11.2 ingress ノードファイアウォールルールオブジェクト

フィールド	型	説明
metadata.name	**string**	CR オブジェクトの名前。
interfaces	**array**	このオブジェクトのフィールドは、ファイアウォールルールを適用するインターフェイスを指定します。たとえば、**-en0** と **-en1** です。
nodeSelector	**array**	**nodeSelector** を使用して、ファイアウォールルールを適用するノードを選択できます。名前付き **nodeselector** ラベルの値を **true** に設定して、ルールを適用します。
ingress	**object**	**ingress** を使用すると、クラスター上のサービスへの外部アクセスを許可するルールを設定できます。

Ingress オブジェクトの設定

Ingress オブジェクトの値は、次の表で定義されています。

表11.3 ingress オブジェクト

フィールド	型	説明
sourceCIDRs	**array**	CIDR ブロックを設定できます。異なるアドレスファミリーから複数の CIDR を設定できます。 **注記** 異なる CIDR を使用すると、同じ順序ルールを使用できます。CIDR が重複する同じノードおよびインターフェイスに対して複数の **IngressNodeFirewall** オブジェクトがある場合、**order** フィールドは最初に適用されるルールを指定します。ルールは昇順で適用されます。

フィールド	型	説明
rules	**array**	Ingress ファイアウォール **rules.order** オブジェクトは、**source.CIDR** ごとに **1** から順に並べられ、CIDR ごとに最大 100 のルールがあります。低次ルールが最初に実行されます。 **rules.protocolConfig.protocol** は次のプロトコルをサポートします: TCP、UDP、SCTP、ICMP、および ICMPv6。ICMP および ICMPv6 ルールは、ICMP および ICMPv6 のタイプまたはコードと照合できます。TCP、UDP、および SCTP ルールは、**<start : end-1>** 形式を使用して、単一の宛先ポートまたはポートの範囲に対して照合できます。 **rules.action** を設定してルールの適用を **allow** するか、**deny** してルールを禁止します。  **注記** ingress ファイアウォールルールは、無効な設定をブロックする検証 Webhook を使用して検証されます。検証 Webhook は、API サーバーや SSH などの重要なクラスターサービスをブロックすることを防ぎます。

ingress ノードファイアウォールルールオブジェクトの例
次の例では、完全な Ingress ノードファイアウォール設定が指定されています。

Ingress ノードファイアウォールの設定例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
  name: ingressnodefirewall
spec:
  interfaces:
  - eth0
  nodeSelector:
    matchLabels:
      <ingress_firewall_label_name>: <label_value> 1
  ingress:
  - sourceCIDRs:
       - 172.16.0.0/12
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMP
        icmp:
          icmpType: 8 #ICMP Echo request
      action: Deny
    - order: 20
```

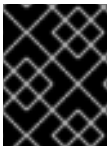
```
    protocolConfig:
      protocol: TCP
      tcp:
        ports: "8000-9000"
    action: Deny
  - sourceCIDRs:
      - fc00:f853:ccd:e793::0/64
    rules:
    - order: 10
      protocolConfig:
        protocol: ICMPv6
        icmpv6:
          icmpType: 128 #ICMPV6 Echo request
      action: Deny
```

1 <label_name> と <label_value> はノード上に存在する必要があり、**ingressfirewallconfig** CR を実行するノードに適用される **nodeselector** ラベルと値に一致する必要があります。<label_value> は、**true** または **false** です。**nodeSelector** ラベルを使用すると、ノードのグループを個別にターゲットにして、**ingressfirewallconfig** CR の使用に異なるルールを適用できます。

ゼロトラスト Ingress ノードファイアウォールルールオブジェクトの例

ゼロトラストの Ingress ノードファイアウォールルールは、マルチインターフェイスクラスターに追加のセキュリティーを提供できます。たとえば、ゼロトラストの Ingress ノードファイアウォールルールを使用して、SSH を除く特定のインターフェイス上のすべてのトラフィックをドロップできます。

次の例では、ゼロトラスト Ingress ノードファイアウォールルールセットの完全な設定が指定されています。



重要

次の場合、ユーザーはアプリケーションが使用するすべてのポートを許可リストに追加して、適切な機能を確保する必要があります。

ゼロトラストの Ingress ノードファイアウォールルールの例

```
apiVersion: ingressnodefirewall.openshift.io/v1alpha1
kind: IngressNodeFirewall
metadata:
 name: ingressnodefirewall-zero-trust
spec:
 interfaces:
 - eth1 1
 nodeSelector:
   matchLabels:
     <ingress_firewall_label_name>: <label_value> 2
 ingress:
 - sourceCIDRs:
     - 0.0.0.0/0 3
   rules:
   - order: 10
     protocolConfig:
       protocol: TCP
       tcp:
         ports: 22
```

```
  action: Allow
- order: 20
  action: Deny 4
```

1. ネットワークインターフェイスクラスター
2. <label_name> と <label_value> は、**ingressfirewallconfig** CR を適用する特定のノードに適用される **nodeSelector** ラベルと値に一致する必要があります。
3. **0.0.0.0/0** は、任意の CIDR に一致するように設定されています
4. **Deny** に設定された **action**

11.4. INGRESS NODE FIREWALL OPERATOR ルールの表示

手順

1. 次のコマンドを実行して、現在のルールをすべて表示します。

```
$ oc get ingressnodefirewall
```

2. 返された **<resource>** 名のいずれかを選択し、次のコマンドを実行してルールまたは設定を表示します。

```
$ oc get <resource> <name> -o yaml
```

11.5. INGRESS NODE FIREWALL OPERATOR のトラブルシューティング

- 次のコマンドを実行して、インストールされている Ingress ノードファイアウォールのカスタムリソース定義 (CRD) を一覧表示します。

```
$ oc get crds | grep ingressnodefirewall
```

出力例

```
NAME              READY   UP-TO-DATE   AVAILABLE   AGE
ingressnodefirewallconfigs.ingressnodefirewall.openshift.io      2022-08-25T10:03:01Z
ingressnodefirewallnodestates.ingressnodefirewall.openshift.io   2022-08-25T10:03:00Z
ingressnodefirewalls.ingressnodefirewall.openshift.io            2022-08-25T10:03:00Z
```

- 次のコマンドを実行して、Ingress Node Firewall Operator の状態を表示します。

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

出力例

```
NAME                                  READY  STATUS        RESTARTS  AGE
ingress-node-firewall-controller-manager  2/2    Running       0         5d21h
ingress-node-firewall-daemon-pqx56        3/3    Running       0         5d21h
```

次のフィールドは、Operator のステータスに関する情報を提供します: **READY**、**STATUS**、**AGE**、および **RESTARTS**。Ingress Node Firewall Operator が割り当てられたノードに設定されたデーモンをデプロイしている場合、**STATUS** フィールドは **Running** になります。

- 次のコマンドを実行して、すべての ingress ファイアウォールノード Pod のログを収集します。

```
$ oc adm must-gather – gather_ingress_node_firewall
```

ログは、**/sos_commands/ebpf** にある eBPF **bpftool** 出力を含む sos ノードのレポートで利用できます。これらのレポートには、Ingress ファイアウォール XDP がパケット処理を処理し、統計を更新し、イベントを発行するときに使用または更新されたルックアップテーブルが含まれます。

第12章 手動 DNS 管理のための INGRESS CONTROLLER の設定

クラスター管理者として Ingress Controller を作成すると、Operator は DNS レコードを自動的に管理します。必要な DNS ゾーンがクラスター DNS ゾーンと異なる場合、または DNS ゾーンがクラウドプロバイダーの外部でホストされている場合、これにはいくつかの制限があります。

クラスター管理者は、Ingress Controller を設定して、自動 DNS 管理を停止し、手動 DNS 管理を開始することができます。**dnsManagementPolicy** を設定して、いつ自動または手動で管理するかを指定します。

Ingress Controller を **Managed** から **Unmanaged** DNS 管理ポリシーに変更すると、Operator はクラウドでプロビジョニングされた以前のワイルドカード DNS レコードをクリーンアップしません。Ingress Controller を **Unmanaged** から **Managed** DNS 管理ポリシーに変更すると、Operator は、クラウドプロバイダーに DNS レコードが存在しない場合は作成を試み、DNS レコードがすでに存在する場合は更新を試みます。

重要

dnsManagementPolicy を **unmanaged** に設定すると、クラウドプロバイダーでワイルドカード DNS レコードのライフサイクルを手動で管理する必要があります。

12.1. MANAGED DNS 管理ポリシー

Ingress Controller の **Managed** DNS 管理ポリシーにより、クラウドプロバイダーのワイルドカード DNS レコードのライフサイクルが Operator によって自動的に管理されるようになります。

12.2. UNMANAGED DNS 管理ポリシー

Ingress Controller の **Unmanaged** DNS 管理ポリシーにより、クラウドプロバイダーのワイルドカード DNS レコードのライフサイクルが自動的に管理されず、代わりにクラスター管理者の責任になります。

注記

AWS クラウドプラットフォームでは、Ingress Controller のドメインが **dnsConfig.Spec.BaseDomain** と一致しない場合、DNS 管理ポリシーは自動的に **Unmanaged** に設定されます。

12.3. UNMANAGED DNS 管理ポリシーを使用したカスタム INGRESS CONTROLLER の作成

クラスター管理者は、**Unmanaged** DNS 管理ポリシーを使用して、新しいカスタム Ingress Controller を作成できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **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. 以下を含む **sample-ingress.yaml** という名前のカスタムリソース (CR) ファイルを作成します。

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  namespace: openshift-ingress-operator
  name: <name> 1
spec:
  domain: <domain> 2
  endpointPublishingStrategy:
    type: LoadBalancerService
    loadBalancer:
      scope: External 3
      dnsManagementPolicy: Unmanaged 4
```

1. **IngressController** オブジェクトの名前で **<name>** を指定します。
2. 前提として作成した DNS レコードをもとに **domain** を指定します。
3. **scope** を **External** として指定して、ロードバランサーを外部に公開します。
4. **dnsManagementPolicy** は、Ingress Controller がロードバランサーに関連付けられたワイルドカード DNS レコードのライフサイクルを管理しているかどうかを示します。有効な値は **Managed** および **Unmanaged** です。デフォルト値は **Managed** です。

2. 変更を適用するためにファイルを保存します。

```
oc apply -f <name>.yaml 1
```

12.4. 既存の INGRESS CONTROLLER の変更

クラスター管理者は、既存の Ingress Controller を変更して、DNS レコードのライフサイクルを手動で管理できます。

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. 選択した **IngressController** を変更して **dnsManagementPolicy** を設定します。

```
SCOPE=$(oc -n openshift-ingress-operator get ingresscontroller <name> -o=jsonpath="
{.status.endpointPublishingStrategy.loadBalancer.scope}")

oc -n openshift-ingress-operator patch ingresscontrollers/<name> --type=merge --
patch='{"spec":{"endpointPublishingStrategy":{"type":"LoadBalancerService","loadBalancer":
{"dnsManagementPolicy":"Unmanaged", "scope":"${SCOPE}"}}}}'
```

2. オプション: クラウドプロバイダーで関連付けられている DNS レコードを削除できます。

12.5. 関連情報

- Ingress Controller 設定パラメーター

第13章 INGRESS CONTROLLER エンドポイント公開戦略の設定

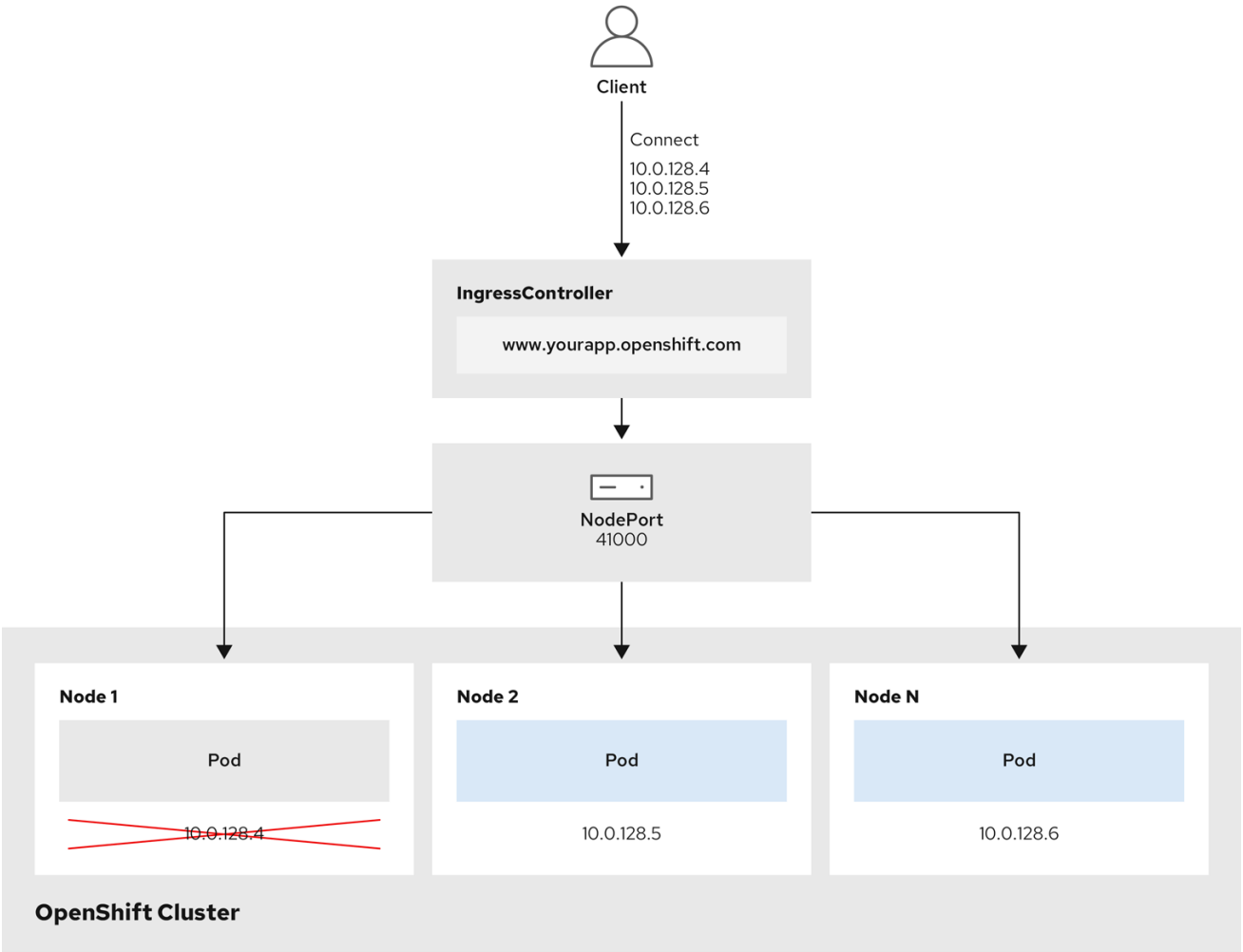
13.1. INGRESS CONTROLLER エンドポイントの公開ストラテジー

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 Controller を公開します。

この設定では、Ingress Controller のデプロイメントはコンテナー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。

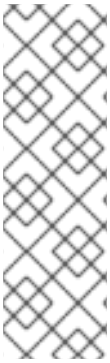
図13.1 NodePortService の図



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。

- クラスター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。クラスター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が、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、マネージド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 Controller がデプロイされるノードポートで Ingress Controller 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 Controller 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。

13.1.1. Ingress Controller エンドポイント公開スコープの内部への設定

クラスター管理者がクラスターをプライベートに指定せずに新しいクラスターをインストールすると、**scope**が**External**に設定されたデフォルトの Ingress Controller が作成されます。クラスター管理者は、**External** スコープの Ingress Controller を **Internal**に変更できます。

前提条件

- **oc** CLI をインストールした。

手順

- **External**スコープの Ingress Controller を **Internal**に変更するには、次のコマンドを入力します。

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

- Ingress Controller のステータスを確認するには、次のコマンドを入力します。

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

- ステータス状態が **Progressing** の場合は、さらにアクションを実行する必要があるかどうかを示します。たとえば、ステータスの状態によっては、次のコマンドを入力して、サービスを削除する必要があることを示している可能性があります。

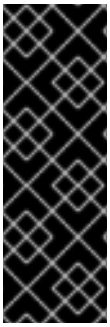
```
$ oc -n openshift-ingress delete services/router-default
```

サービスを削除すると、Ingress Operator はサービスを**Internal**として再作成します。

13.1.2. Ingress Controller エンドポイント公開スコープの外部への設定

クラスター管理者がクラスターをプライベートに指定せずに新しいクラスターをインストールすると、**scope**が**External**に設定されたデフォルトの Ingress Controller が作成されます。

Ingress Controller のスコープは、インストール中またはインストール後に**Internal**になるように設定でき、クラスター管理者は**Internal** の Ingress Controller を **External**に変更できます。



重要

一部のプラットフォームでは、サービスを削除して再作成する必要があります。

スコープを変更すると、場合によっては数分間、Ingress トラフィックが中断される可能性があります。これが該当するのは、サービスを削除して再作成する必要があるプラットフォームです。理由は、この手順により、OpenShift Container Platform が既存のサービスロードバランサーのプロビジョニングを解除して新しいサービスロードバランサーをプロビジョニングし、DNS を更新する可能性があるためです。

前提条件

- **oc** CLI をインストールした。

手順

- **Internal**スコープの入力コントローラーを**External**に変更するには、次のコマンドを入力します。

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

- Ingress Controller のステータスを確認するには、次のコマンドを入力します。

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

- ステータス状態が **Progressing** の場合は、さらにアクションを実行する必要があるかどうかを示します。たとえば、ステータスの状態によっては、次のコマンドを入力して、サービスを削除する必要があることを示している可能性があります。

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

サービスを削除すると、Ingress Operator はサービスを**External**として再作成します。

13.2. 関連情報

- 詳細は、Ingress Controller configuration parameters を参照してください。

第14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。

14.1. 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

14.2. 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ッスンします。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。

14.3. 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14.1 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。 • **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14.2 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14.3 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**	接続ログエントリー (停止の正常な終了に関連するログエントリーを含む)。
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ー。

接続ログフィールド

接続ログエントリー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14.4 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**	ログエントリーが成功または失敗であるかを示します。
time	**string**	接続チェックの開始時間。

14.4. 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** オブジェクトをリスト表示するには、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15章 クラスターネットワークの MTU 変更

クラスター管理者は、クラスターのインストール後にクラスターネットワークの MTU を変更できます。MTU 変更の適用には、クラスターノードを再起動する必要があるため、変更により致命的な問題が発生する可能性があります。OVN-Kubernetes または OpenShift SDN ネットワークプラグインを使用して、クラスターの MTU のみを変更できます。

15.1. クラスター MTU について

インストール中に、クラスターネットワークの最大伝送ユニット (MTU) は、クラスター内の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 MTU をもとに、自動的に検出されます。通常、検出された MTU をオーバーライドする必要はありません。

以下のような理由でクラスターネットワークの MTU を変更する場合があります。

- クラスターのインストール中に検出された MTU が使用中のインフラストラクチャーに適していない
- クラスターインフラストラクチャーに異なる MTU が必要となった (例: パフォーマンスの最適化にさまざまな MTU を必要とするノードが追加された)。

OVN-Kubernetes クラスターネットワークプラグインのみが MTU 値の変更をサポートしています。

15.1.1. サービス中断に関する考慮事項

クラスターで MTU の変更を開始すると、次の動作が原因でサービスの可用性に影響を与える可能性があります。

- 新しい MTU への移行を完了するには、少なくとも 2 回のローリングリブートが必要です。この間、一部のノードは再起動するため使用できません。
- 特定のアプリケーションに、絶対 TCP タイムアウト間隔よりもタイムアウトの間隔が短いクラスターにデプロイされた場合など、MTU の変更中に中断が発生する可能性があります。

15.1.2. MTU 値の選択

MTU の移行を計画するときは、関連しているが異なる MTU 値を 2 つ考慮する必要があります。

- **ハードウェア MTU**: この MTU 値は、ネットワークインフラストラクチャーの詳細に基づいて設定されます。
- **クラスターネットワーク MTU**: この MTU 値は、クラスターネットワークオーバーレイのオーバーヘッドを考慮して、常にハードウェア MTU よりも小さくなります。具体的なオーバーヘッドは、ネットワークプラグインによって決まります。OVN-Kubernetes の場合、オーバーヘッドは **100** バイトです。

クラスターがノードごとに異なる MTU 値を必要とする場合は、クラスター内の任意のノードで使用される最小の MTU 値から、ネットワークプラグインのオーバーヘッド値を差し引く必要があります。たとえば、クラスター内の一部のノードでは MTU が **9001** であり、MTU が **1500** のクラスターもある場合には、この値を **1400** に設定する必要があります。

重要

ノードが受け入れられない MTU 値の選択を回避するには、**ip -d link** コマンドを使用して、ネットワークインターフェイスが受け入れる最大 MTU 値 (**maxmtu**) を確認します。

15.1.3. 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。

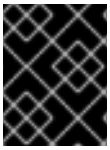
表15.1 クラスター MTU のライブマイグレーション

ユーザーが開始する手順	OpenShift Container Platform アクティビティー
Cluster Network Operator 設定で次の値を指定します。 ● **spec.migration.mtu.machine.to** ● **spec.migration.mtu.network.from** ● **spec.migration.mtu.network.to**	**Cluster Network Operator (CNO)**: 各フィールドが有効な値に設定されていることを確認します。 ● **mtu.machine.to**は、新しいハードウェア MTU、またはハードウェアの MTU が変更されていない場合は、現在のハードウェア MTU のいずれかに設定する必要があります。この値は一時的なものであり、移行プロセスの一部として使用されます。これとは別に、既存のハードウェア MTU 値とは異なるハードウェア MTU を指定する場合は、マシン設定、DHCP 設定、Linux カーネルコマンドラインなどの他の方法で永続化するように MTU を手動で設定する必要があります。 ● **mtu.network.from**フィールドは、クラスターネットワークの現在の MTU である**network.status.cluster Network MTU**フィールドと同じである必要があります。 ● **mtu.network.to**フィールドは、ターゲットクラスターネットワーク MTU に設定する必要があり、ネットワークプラグインのオーバーレイオーバーヘッドを考慮して、ハードウェア MTU よりも低くする必要があります。OVN-Kubernetes の場合、オーバーヘッドは **100** バイトです。 指定の値が有効な場合に、CNO は、クラスターネットワークの MTU が**mtu.network.to**フィールドの値に設定された新しい一時設定を書き出します。 **Machine Config Operator (MCO)**: クラスター内の各ノードのローリングリブートを実行します。

ユーザーが開始する手順	OpenShift Container Platform アクティビティー
クラスター上の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 MTU を再設定します。これを実現するには、次のようなさまざまな方法を使用できます。 - MTU を変更した新しい Network Manager 接続プロファイルのデプロイ - DHCP サーバー設定による MTU の変更 - ブートパラメーターによる MTU の変更	該当なし
ネットワークプラグインの CNO 設定で **mtu** 値を設定し、**spec.migration** を **null** に設定します。	**Machine Config Operator (MCO)** 新しい MTU 設定を使用して、クラスター内の各ノードのローリングリブートを実行します。

15.2. クラスターネットワーク MTU の変更

クラスター管理者は、クラスターの最大伝送単位 (MTU) を増減できます。



重要

移行には中断が伴うため、MTU 更新が有効になると、クラスター内のノードが一時的に使用できなくなる可能性があります。

次の手順では、マシン設定、Dynamic Host Configuration Protocol (DHCP)、または ISO イメージのいずれかを使用してクラスターネットワーク MTU を変更する方法について説明します。DHCP または ISO アプローチを使用する場合は、クラスターのインストール後に保持した設定アーティファクトを参照して、手順を完了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** 権限を持つアカウントを使用してクラスターにアクセスできる。
- クラスターのターゲット MTU を特定している。OVN-Kubernetes ネットワークプラグインの MTU は、クラスター内の最小のハードウェア MTU 値から **100** を引いた値に設定する必要があります。

手順

1. クラスターネットワークの現在の MTU を取得するには、次のコマンドを入力します。

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

出力例

```
...
```

```
Status:
  Cluster Network:
    Cidr:               10.217.0.0/22
    Host Prefix:        23
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    10.217.4.0/23
...
```

2. ハードウェア MTU の設定を準備します。

- ハードウェア MTU が DHCP で指定されている場合は、次の dnsmasq 設定などで DHCP 設定を更新します。

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

ここでは、以下のようになります。

<mtu>
DHCP サーバーがアドバタイズするハードウェア MTU を指定します。

- ハードウェア MTU が PXE を使用したカーネルコマンドラインで指定されている場合は、それに応じてその設定を更新します。

- ハードウェア MTU が Network Manager 接続設定で指定されている場合は、以下のステップを実行します。OpenShift Container Platform では、これは、DHCP、カーネルコマンドラインなどの方法でネットワーク設定を明示的に指定していない場合のデフォルトのアプローチです。変更なしで次の手順を機能させるには、全クラスターノードで、同じ基盤となるネットワーク設定を使用する必要があります。

i. プライマリーネットワークインターフェイスを見つけます。

  - OpenShift SDN ネットワークプラグインを使用している場合は、次のコマンドを入力します。

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

ここでは、以下のようになります。

<node_name>
クラスター内のノードの名前を指定します。

  - OVN-Kubernetes ネットワークプラグインを使用している場合は、次のコマンドを入力します。

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

ここでは、以下のようになります。

<node_name>
クラスター内のノードの名前を指定します。

ii. **<interface>-mtu.conf** ファイルに次の NetworkManager 設定を作成します。

Network Manager 接続設定の例

```
[connection-<interface>-mtu]
match-device=interface-name:<interface>
ethernet.mtu=<mtu>
```

ここでは、以下のようになります。

<mtu>
新しいハードウェア MTU 値を指定します。

<interface>
プライマリーネットワークインターフェイス名を指定します。

iii. 1 つはコントロールプレーンノード用、もう 1 つはクラスター内のワーカーノード用に、2 つの**MachineConfig**オブジェクトを作成します。

A. **control-plane-interface.bu** ファイルに次の Butane 設定を作成します。

```
variant: openshift
version: 4.15.0
metadata:
  name: 01-control-plane-interface
  labels:
    machineconfiguration.openshift.io/role: master
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf (1)
      contents:
        local: <interface>-mtu.conf (2)
      mode: 0600
```

(1) (1) プライマリーネットワークインターフェイスの NetworkManager 接続名を指定します。

(2) 前の手順で更新された NetworkManager 設定ファイルのローカルファイル名を指定します。

B. **worker-interface.bu** ファイルに次の Butane 設定を作成します。

```
variant: openshift
version: 4.15.0
metadata:
  name: 01-worker-interface
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf (1)
      contents:
        local: <interface>-mtu.conf (2)
      mode: 0600
```

1. プライマリーネットワークインターフェイスの NetworkManager 接続名を指定します。
2. 前の手順で更新された NetworkManager 設定ファイルのローカルファイル名を指定します。

C. 次のコマンドを実行して、Butane 設定から **MachineConfig** オブジェクトを作成します。

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

警告

これらのマシン設定は、後で明示的に指示されるまで適用しないでください。これらのマシン設定を適用すると、クラスターの安定性が失われます。

3. MTU 移行を開始するには、次のコマンドを入力して移行設定を指定します。Machine Config Operator は、MTU の変更に備えて、クラスター内のノードをローリングリブートします。

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

ここでは、以下のようになります。

<overlay_from>

現在のクラスターネットワークの MTU 値を指定します。

<overlay_to>

クラスターネットワークのターゲット MTU を指定します。この値は、**<machine_to>** の値を基準にして設定します。OVN-Kubernetes の場合、この値は **<machine_to>** の値から **100** を引いた値である必要があります。

<machine_to>

基盤となるホストネットワークのプライマリーネットワークインターフェイスの MTU を指定します。

クラスター MTU を増やす例

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

4. Machine Config Operator (MCO) は、各マシン設定プール内のマシンを更新するときに、各ノードを 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 machineconfigpools
```

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

Machine Config Operator は、デフォルトでプールごとに 1 つずつマシンを更新するため、クラスターのサイズに応じて移行にかかる合計時間が増加します。

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リスト表示するには、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
   ```

   b. 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。

   c. 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
   ```

   <config_name> は **machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前です。

   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/mtu-migration.sh
   ```

6. 基盤となるネットワークインターフェイスの MTU 値を更新します。

   - Network Manager 接続設定で新しい MTU を指定する場合は、次のコマンドを入力します。Machine Config Operator は、クラスター内のノードのローリングリブートを自動的に実行します。

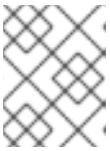
```
$ for manifest in control-plane-interface worker-interface; do
    oc create -f $manifest.yaml
  done
```

- DHCP サーバーオプションまたはカーネルコマンドラインと PXE を使用して新しい MTU を指定する場合は、インフラストラクチャーに必要な変更を加えます。

7. Machine Config Operator (MCO) は、各マシン設定プール内のマシンを更新するときに、各ノードを 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 machineconfigpools
```

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

Machine Config Operator は、デフォルトでプールごとに 1 つずつマシンを更新するため、クラスターのサイズに応じて移行にかかる合計時間が増加します。

8. 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リスト表示するには、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 path:
```

<config_name> は **machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前です。

マシン設定が正常にデプロイされた場合には、前の出力に **/etc/NetworkManager/system-connections/<connection_name>** のファイルパスが含まれます。

マシン設定には、**ExecStart=/usr/local/bin/mtu-migration.sh** 行を含めることはできません。

9. MTU の移行を完了するために、OVN-Kubernetes ネットワークプラグインに対して次のコマンドを入力します。

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

ここでは、以下のようになります。

<mtu>

<overlay_to> で指定した新しいクラスターネットワーク MTU を指定します。

10. MTU の移行が完了すると、各マシン設定プールノードが 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 machineconfigpools
```

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。

検証

1. クラスターネットワークの現在の MTU を取得するには、次のコマンドを入力します。

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

2. 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現在の MTU を取得します。

   a. クラスター内のノードをリスト表示するには、次のコマンドを入力します。

```
$ oc get nodes
```

   b. 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現在の MTU 設定を取得するには、次のコマンドを入力します。

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

ここでは、以下のようになります。

<node>

前のステップの出力をもとに、ノードを指定します。

<interface>

ノードのプライマリーネットワークインターフェイス名を指定します。

出力例

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

15.3. 関連情報

- PXE および ISO インストールの高度なネットワークオプションの使用
- 鍵ファイル形式で NetworkManager プロファイルの手動による作成
- nmcli で動的イーサネット接続の設定

第16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。

16.1. 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。

16.2. 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"]
```

16.3. 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17章 クラスターネットワーク範囲の設定

クラスター管理者は、クラスターのインストール後にクラスターネットワークの範囲を拡張できます。追加ノード用にさらに多くの IP アドレスが必要な場合は、クラスターネットワーク範囲を拡張することを推奨します。

たとえば、クラスターをデプロイし、クラスターネットワーク範囲として **10.128.0.0/19** を指定し、ホスト接頭辞 **23** を指定した場合、16 ノードに制限されます。クラスターの CIDR マスクを **/14** に変更することで、これを 510 ノードに拡張できます。

クラスターのネットワークアドレス範囲を拡張する場合、クラスターは OVN-Kubernetes ネットワークプラグイン を使用する必要があります。他のネットワークプラグインはサポートされていません。

クラスターネットワークの IP アドレス範囲を変更する場合、次の制限が適用されます。

- 指定する CIDR マスクサイズは、現在設定されている CIDR マスクサイズより常に小さくする必要があります。これは、インストールされたクラスターにノードを追加することによってのみ IP スペースを増やすことができるためです。
- ホスト 接頭辞は変更できません
- オーバーライドされたデフォルトゲートウェイで設定された Pod は、クラスターネットワークの拡張後に再作成する必要があります。

17.1. クラスターネットワークの IP アドレス範囲の拡張

クラスターネットワークの IP アドレス範囲を拡張できます。この変更には新しい Operator 設定をクラスター全体にロールアウトする必要があるため、有効になるまでに最大 30 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** 権限を持つユーザーとしてクラスターにログインする。
- クラスターが OVN-Kubernetes ネットワークプラグインを使用していることを確認します。

手順

1. クラスターのクラスターネットワーク範囲とホスト接頭辞を取得するには、次のコマンドを入力します。

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.clusterNetwork}"
```

出力例

```
[{"cidr":"10.217.0.0/22","hostPrefix":23}]
```

2. クラスターネットワークの IP アドレス範囲を拡張するには、次のコマンドを入力します。前のコマンドの出力から返された CIDR IP アドレス範囲とホスト接頭辞を使用します。

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch \
  '{
```

```
    "spec":{
      "clusterNetwork": [ {"cidr":"<network>/<cidr>","hostPrefix":<prefix>} ],
      "networkType": "OVNKubernetes"
    }
  }'
```

ここでは、以下のようになります。

<network>

前の手順で取得した **cidr** フィールドのネットワーク部分を指定します。この値は変更できません。

<cidr>

ネットワーク接頭辞長を指定します。たとえば、**14** です。この値を前の手順の出力の値よりも小さい数値に変更して、クラスターネットワークの範囲を拡大します。

<prefix>

クラスターの現在のホスト接頭辞を指定します。この値は、前の手順で取得した **hostPrefix** フィールドの値と同じである必要があります。

コマンドの例

```
$ oc patch Network.config.openshift.io cluster --type='merge' --patch \
  '{
    "spec":{
      "clusterNetwork": [ {"cidr":"10.217.0.0/14","hostPrefix": 23} ],
      "networkType": "OVNKubernetes"
    }
  }'
```

出力例

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

3. 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。この変更が有効になるまで、最大 30 分かかる場合があります。

```
$ oc get network.operator.openshift.io \
  -o jsonpath="{.items[0].spec.clusterNetwork}"
```

出力例

```
[{"cidr":"10.217.0.0/14","hostPrefix":23}]
```

17.2. 関連情報

- Red Hat OpenShift Network Calculator
- OVN-Kubernetes ネットワークプラグインについて

第18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 Keepalived を使用して、一連のホストで外部からアクセスできる仮想 IP (VIP) アドレスのセットをホストします。各仮想 IP アドレスは、一度に 1 つのホストによってのみサービスされます。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。

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。単一のノードが使用可能な場合は、仮想 IP が提供されます。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。

管理者は、すべての仮想 IP アドレスが次の要件を満たしていることを確認する必要があります。

- 設定されたホストでクラスター外からアクセスできる。
- 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

セット内の各仮想 IP は、異なるノードによってサービスされる可能性があります。

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。

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 **OPENSHIFT_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で呼び出されます。

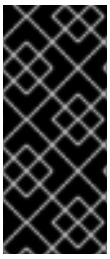
OpenShift Container Platform で IP フェイルオーバーのデプロイメント設定を作成できます。IP フェイルオーバーのデプロイメント設定は VIP アドレスのセットを指定し、それらの提供先となるノードのセットを指定します。クラスターには複数の IP フェイルオーバーのデプロイメント設定を持たせることができ、それぞれが固有な VIP アドレスの独自のセットを管理します。IP フェイルオーバー設定の各ノードは IP フェイルオーバー Pod として実行され、この Pod は Keepalived を実行します。

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、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 の両方に同じセレ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、**NodePort** を使用することができます。**NodePort** のセットアップは特権付きの操作で実行されます。

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リングポートはサービス定義で **NodePort** に設定されます。



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。

ExternalIP を使用する場合は、**ExternalIP** 範囲と同じ仮想 IP 範囲を持つように IP フェイルオーバーを設定できます。また、モニタ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。どのユーザーでも、**ExternalIP** を使用してサービスをセットアップし、高可用性を実現できます。

重要

クラスター内の VIP の最大数は 254 です。

18.1. 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18.1 IP フェイルオーバーの環境変数

変数名	デフォルト	説明
OPENSHIFT_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。
OPENSHIFT_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。

変数名	デフォルト	説明
OPENSHIFT_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。
OPENSHIFT_HA_VIRTUAL_IPS		レプリケートする IP アドレス範囲のリストです。これは指定する必要があります。例: **1.2.3.4-6,1.2.3.9**
OPENSHIFT_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。
OPENSHIFT_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 **OPENSHIFT_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。
OPENSHIFT_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。
OPENSHIFT_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。
OPENSHIFT_HA_CHECK_INTERVAL	**2**	check スクリプトが実行される期間 (秒単位) です。
OPENSHIFT_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。
OPENSHIFT_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。

18.2. クラスター内の 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。
- 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: OPENSHIFT_HA_CONFIG_NAME
        value: "ipfailover"
      - name: OPENSHIFT_HA_VIRTUAL_IPS 2
        value: "1.1.1.1-2"
      - name: OPENSHIFT_HA_VIP_GROUPS 3
        value: "10"
      - name: OPENSHIFT_HA_NETWORK_INTERFACE 4
        value: "ens3" #The host interface to assign the VIPs
      - name: OPENSHIFT_HA_MONITOR_PORT 5
        value: "30060"
      - name: OPENSHIFT_HA_VRRP_ID_OFFSET 6
        value: "0"
      - name: OPENSHIFT_HA_REPLICA_COUNT 7
        value: "2" #Must match the number of replicas in the deployment
      - name: OPENSHIFT_HA_USE_UNICAST
        value: "false"
      #- name: OPENSHIFT_HA_UNICAST_PEERS
        #value: "10.0.148.40,10.0.160.234,10.0.199.110"
      - name: OPENSHIFT_HA_IPTABLES_CHAIN 8
        value: "INPUT"
      #- name: OPENSHIFT_HA_NOTIFY_SCRIPT 9
      #  value: /etc/keepalive/mynotifyscript.sh
      - name: OPENSHIFT_HA_CHECK_SCRIPT 10
        value: "/etc/keepalive/mycheckscript.sh"
      - name: OPENSHIFT_HA_PREEMPTION 11
        value: "preempt_delay 300"
      - name: OPENSHIFT_HA_CHECK_INTERVAL 12
        value: "2"
      livenessProbe:
        initialDelaySeconds: 10
        exec:
```

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

1. IP フェイルオーバーデプロイメントの名前。

2. レプリケートする IP アドレス範囲のリストです。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 **OPENSHIFT_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。

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。

11 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。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

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。

18.3. 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チェックスクリプトを定期的に実行し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。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。

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** オブジェクト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。次の方法で説明するように、**ConfigMap** オブジェクトを使用してスクリプトを Pod に追加できます。

Check script

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) アドレスを管理する 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。

チェックスクリプトがゼロ以外を返す場合、ノードは **backup** 状態になり、保持されている仮想 IP は再割り当てされます。

Notify script

Keepalived は、次の 3 つのパラメーターを通知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**

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、それを保持する **ConfigMap** オブジェクトを作成します。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** オブジェクトを作成します。

```
$ 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 **ConfigMap** オブジェクトファイルの **defaultMode** は、**oc** コマンドを使用するか、デプロイメント設定を編集することで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 \
    OPENSHIFT_HA_CHECK_SCRIPT=/etc/keepalive/mycheckscript.sh
```

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

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: OPENSHIFT_HA_CHECK_SCRIPT (1)
          value: /etc/keepalive/mycheckscript.sh
...
        volumeMounts: (2)
        - mountPath: /etc/keepalive
          name: config-volume
      dnsPolicy: ClusterFirst
...
      volumes: (3)
      - configMap:
          defaultMode: 0755 (4)
          name: customrouter
        name: config-volume
...
```

1. **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 **OPENSHIFT_HA_CHECK_SCRIPT** 環境変数を追加します。
2. **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。
3. 新規の **spec.volumes** フィールドを追加して ConfigMap に言及します。
4. 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。

18.4. 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。**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。

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: OPENSHIFT_HA_PREEMPTION (1)
          value: preempt_delay 300
...
```

(1) **OPENSHIFT_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。

18.5. 複数の IP フェイルオーバーインスタンスのデプロイ

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。

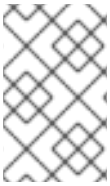
したがって、IP フェイルオーバーのデプロイメント設定で定義されるすべての VIP について、IP フェイルオーバー Pod は対応する **vrrp-id** を割り当てる必要があります。これは、**OPENSHIFT_HA_VRRP_ID_OFFSET** から開始し、順序に従って **vrrp-ids** を VIP のリストに割り当てることによって実行されます。**vrrp-ids** には範囲 **1..255** の値を設定できます。

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、**OPENSHIFT_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。

18.6. 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

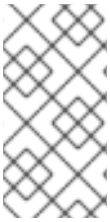
IP フェイルオーバー管理は、仮想 IP (VIP) アドレスの 254 グループに制限されています。デフォルトでは、OpenShift Container Platform は各グループに 1 つの IP アドレスを割り当てます。**OPENSHIFT_HA_VIP_GROUPS** 変数を使用してこれを変更し、複数の IP アドレスが各グループに含まれるようにして、IP フェイルオーバーを設定するときに各 Virtual Router Redundancy Protocol (VRRP) インスタンスで使用可能な VIP グループの数を定義できます。

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 を使用し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。

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、**OPENSHIFT_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: OPENSHIFT_HA_VIP_GROUPS (1)
      value: "3"
...
```

(1) たとえば、7 つの VIP のある環境で **OPENSHIFT_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

OPENSHIFT_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。

18.7. EXTERNALIP の高可用性

非クラウドクラスターでは、IP フェイルオーバーとサービスへの **ExternalIP** を組み合わせることができます。結果として、**ExternalIP** を使用してサービスを作成するユーザーに高可用サービスが提供されます。

このアプローチでは、クラスターネットワーク設定の **spec.ExternalIP.autoAssignCIDRs** 範囲を指定し、同じ範囲を使用して IP フェイルオーバー設定を作成します。

IP フェイルオーバーはクラスター全体で最大 255 個の仮想 IP をサポートできるため、**spec.ExternalIP.autoAssignCIDRs** は **/24** 以下にする必要があります。

18.8. 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:4.15
        command: ["/var/lib/ipfailover/keepalived/remove-failover.sh"]
      nodeSelector: 1
        kubernetes.io/hostname: <host_name> 2
      restartPolicy: Never
```

1 **nodeSelector** は、古い IP フェイルオーバーデプロイメントで使用されていたセレクターと同じである可能性があります。

2 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。

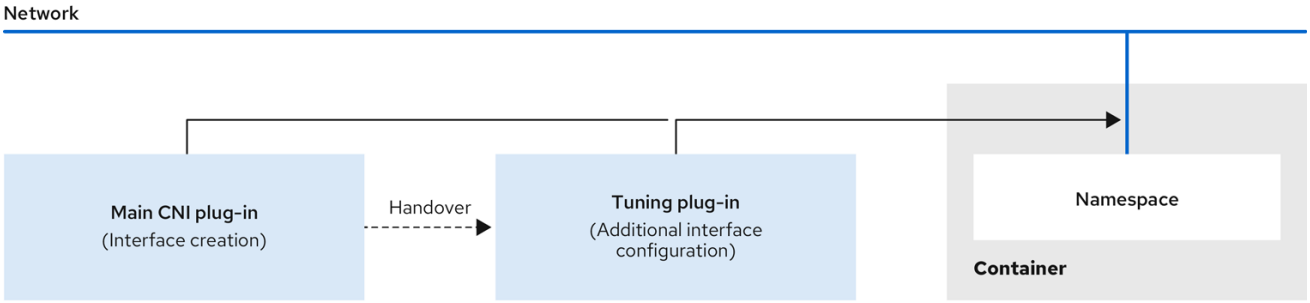
```
$ oc logs job/remove-ipfailover-2h8dm
```

出力例

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

第19章 チューニングプラグインを使用してシステムコントロールとインターフェイス属性を設定する

Linux では、管理者は sysctl を使用してランタイム時にカーネルパラメーターを変更できます。チューニング Container Network Interface (CNI) メタプラグインを使用して、インターフェイスレベルのネットワーク sysctl を変更できます。チューニング CNI メタプラグインは、図に示すように、メインの CNI プラグインとチェーンで動作します。



メインの CNI プラグインはインターフェイスを割り当て、それをランタイム時にチューニング CNI メタプラグインに渡します。チューニング CNI メタプラグインを使用して、ネットワーク namespace の一部の sysctl といくつかのインターフェイス属性 (プロミスキャスモード、オールマルチキャストモード、MTU、および MAC アドレス) を変更できます。

19.1. チューニング CNI を使用してシステム制御を設定する

次の手順では、チューニング CNI を設定し、インターフェイスレベルのネットワーク **net.ipv4.conf.IFNAME.accept_redirects** sysctl を変更します。この例では、ICMP リダイレクトパケットの受け入れと送信を有効にします。チューニング CNI メタプラグイン設定では、インターフェイス名は **IFNAME** トークンで表され、ランタイム時にインターフェイスの実際の名前に置き換えられます。

手順

1. 次の内容で、**tuning-example.yaml** などのネットワーク接続定義を作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <name> 1
  namespace: default 2
spec:
  config: '{
    "cniVersion": "0.4.0", 3
    "name": "<name>", 4
    "plugins": [{
      "type": "<main_CNI_plugin>" 5
    },
    {
     "type": "tuning", 6
     "sysctl": {
          "net.ipv4.conf.IFNAME.accept_redirects": "1" 7
     }
```

```
    }
   ]
}
```

1. 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. オブジェクトが関連付けられている namespace を指定します。
3. CNI 仕様のバージョンを指定します。
4. 設定の名前を指定します。設定名をネットワーク接続定義の name の値に一致させることが推奨されます。
5. 設定するメイン CNI プラグインの名前を指定します。
6. CNI メタプラグインの名前を指定します。
7. 設定する sysctl を指定します。インターフェイス名は **IFNAME** トークンで表され、ランタイム時にインターフェイスの実際の名前に置き換えられます。

YAMLl ファイルの例を次に示します。

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

2. 以下のコマンドを実行して YAML を適用します。

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

出力例

```
networkattachmentdefinition.k8.cni.cncf.io/tuningnad created
```

3. 次のようなネットワーク接続定義を使用して、**examplepod.yaml** などの Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: tuningnad (1)
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000 (2)
      runAsGroup: 3000 (3)
      allowPrivilegeEscalation: false (4)
      capabilities: (5)
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true (6)
    seccompProfile: (7)
      type: RuntimeDefault
```

(1) 設定済みの **NetworkAttachmentDefinition** の名前を指定します。

(2) **runAsUser** は、コンテナーが実行されるユーザー ID を制御します。

(3) **runAsGroup** は、コンテナーが実行されるプライマリーグループ ID を制御します。

(4) **allowPrivilegeEscalation** は、Pod が特権の昇格を許可するように要求できるかどうかを決定します。指定しない場合、デフォルトで true に設定されます。このブール値は、**no_new_privs** フラグがコンテナープロセスに設定されるかどうかを直接制御します。

(5) **capabilities** は、完全なルートアクセスを許可せずに権限操作を許可します。このポリシーにより、すべての機能が Pod から削除されます。

(6) **runAsNonRoot: true** は、コンテナーが 0 以外の任意の UID を持つユーザーで実行されることを要求します。

(7) **RuntimeDefault** は、Pod またはコンテナーワークロードのデフォルトの seccomp プロファイルを有効にします。

4. 以下のコマンドを実行して yaml を適用します。

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

5. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod
```

出力例

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

6. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh tunepod
```

7. 設定された sysctl フラグの値を確認します。たとえば、次のコマンドを実行して、値 **net.ipv4.conf.net1.accept_redirects** を見つけます。

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

予想される出力

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

19.2. チューニング CNI を使用してオールマルチキャストモードを有効にする

チューニング Container Network Interface (CNI) メタプラグインを使用して、オールマルチキャストモードを有効にできます。

次の手順では、チューニング CNI を設定してオールマルチキャストモードを有効にする方法について説明します。

手順

1. 次の内容で、**tuning-example.yaml** などのネットワーク接続定義を作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <name> 1
  namespace: default 2
spec:
  config: '{
    "cniVersion": "0.4.0", 3
    "name": "<name>", 4
    "plugins": [{
      "type": "<main_CNI_plugin>" 5
    },
    {
      "type": "tuning", 6
      "allmulti": true 7
    }
   ]
}
```

1 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 オブジェクトが関連付けられている namespace を指定します。

3 CNI 仕様のバージョンを指定します。

4 設定の名前を指定します。設定名は、ネットワーク接続定義の名前の値と一致させます。

5 設定するメイン CNI プラグインの名前を指定します。

6 CNI メタプラグインの名前を指定します。

7 インターフェイスのオールマルチキャストモードを変更します。有効にすると、ネットワーク上のすべてのマルチキャストパケットがそのインターフェイスで受信されます。

YAMLl ファイルの例を次に示します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: setallmulti
  namespace: default
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "setallmulti",
    "plugins": [
      {
        "type": "bridge"
      },
      {
        "type": "tuning",
        "allmulti": true
      }
    ]
  }'
```

2. 次のコマンドを実行して、YAML ファイルで指定された設定を適用します。

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

出力例

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

3. 次の **examplepod.yaml** サンプルファイルで指定されているものと同様のネットワーク接続定義を持つ Pod を作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: allmultipod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/networks: setallmulti 1
spec:
```

```
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000 2
      runAsGroup: 3000 3
      allowPrivilegeEscalation: false 4
      capabilities: 5
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true 6
    seccompProfile: 7
      type: RuntimeDefault
```

1. 設定された **NetworkAttachmentDefinition** の名前を指定します。
2. コンテナーの実行に使用するユーザー ID を指定します。
3. コンテナーの実行に使用するプライマリーグループ ID を指定します。
4. Pod が権限昇格を要求できるか指定します。指定しない場合、デフォルトで **true** に設定されます。このブール値は、**no_new_privs** フラグがコンテナープロセスに設定されるかどうかを直接制御します。
5. コンテナーのケイパビリティーを指定します。**drop: ["ALL"]** ステートメントは、すべての Linux ケイパビリティーが Pod からドロップされ、より制限的なセキュリティープロファイルが提供されていることを示します。
6. UID が 0 以外のユーザーでコンテナーが実行されるように指定します。
7. コンテナーの seccomp プロファイルを指定します。この場合、タイプは **RuntimeDefault** に設定されます。Seccomp は、プロセスで使用できるシステムコールを制限し、攻撃対象領域を最小化してセキュリティーを強化する Linux カーネル機能です。

4. 次のコマンドを実行して、YAML ファイルで指定された設定を適用します。

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

5. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod
```

出力例

```
NAME         READY   STATUS    RESTARTS   AGE
allmultipod  1/1     Running   0          23s
```

6. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh allmultipod
```

7. 次のコマンドを実行して、Pod に関連付けられているすべてのインターフェイスをリスト表示します。

```
sh-4.4# ip link
```

出力例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8901 qdisc noqueue state
UP mode DEFAULT group default
    link/ether 0a:58:0a:83:00:10 brd ff:ff:ff:ff:ff:ff link-netnsid 0 (1)
3: net1@if24: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether ee:9b:66:a4:ec:1d brd ff:ff:ff:ff:ff:ff link-netnsid 0 (2)
```

(1) **eth0@if22** は、プライマリーインターフェイスです。

(2) **net1@if24** は、オールマルチキャストモード (ALLMULTI フラグ) をサポートするネットワーク接続定義で設定されたセカンダリーインターフェイスです。

19.3. 関連情報

- コンテナーでの sysctl の使用
- SR-IOV ネットワークノード設定オブジェクト
- SR-IOV ネットワークのインターフェイスレベルのネットワーク sysctl 設定とオールマルチキャストモードを設定する

第20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。

20.1. OPENSHIFT 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。

20.1.1. 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
```

20.2. 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
```

20.3. 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 トラフィックをリッ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/ubi9/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/ubi9/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. **sctpservice** 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 sctpservice -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 **sctpservice** 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21章 PTP ハードウェアの使用

21.1. OPENSHIFT CONTAINER PLATFORM クラスターノードの 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。

OpenShift Container Platform クラスターノードで**linuxptp** サービスを設定し、PTP 対応ハードウェアを使用できます。

OpenShift Container Platform Web コンソールまたは OpenShift CLI (**oc**) を使用して、PTP Operator をデプロイして PTP をインストールします。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。

- クラスター内の 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。

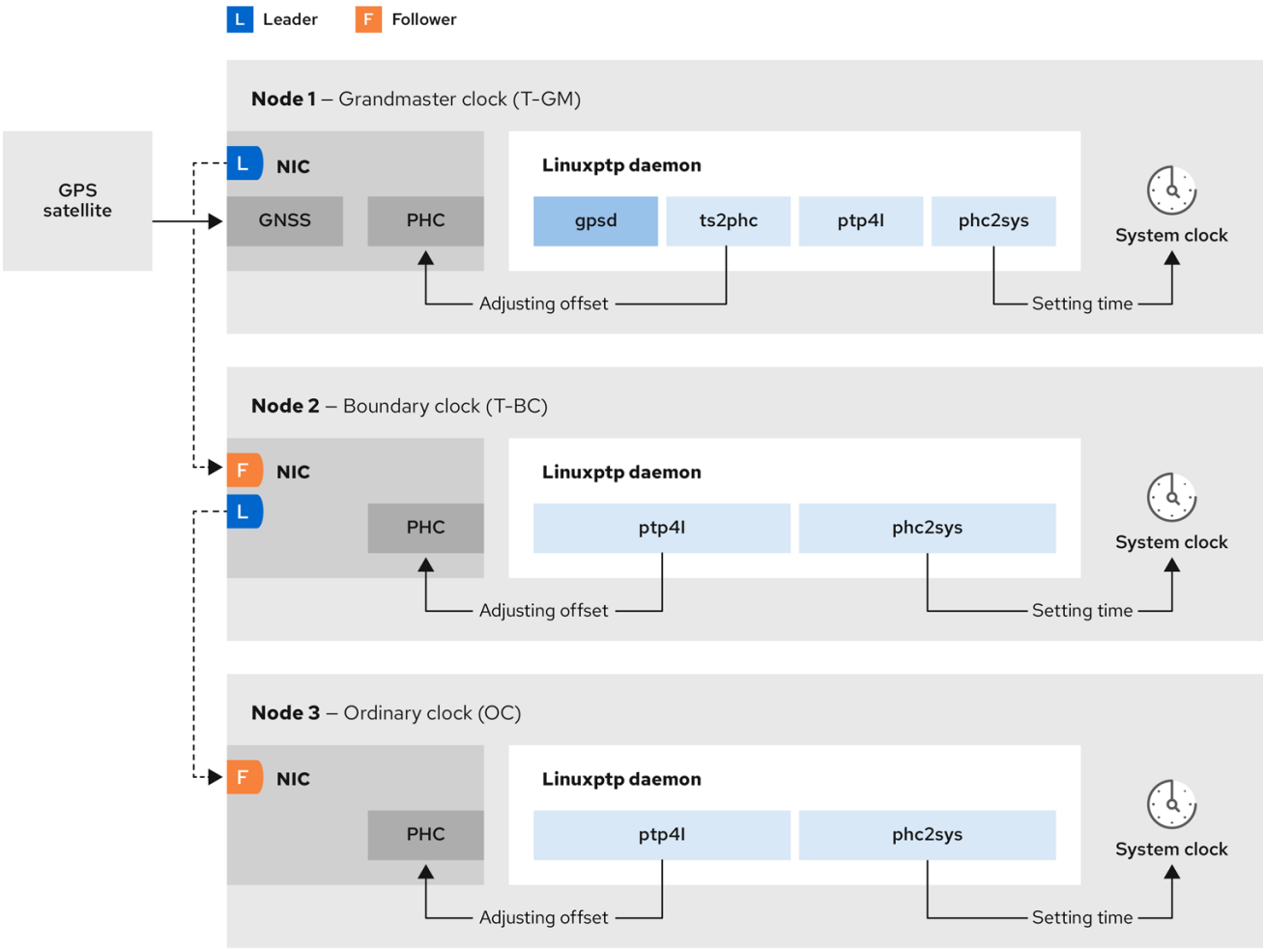
注記

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ーの PTP 対応デバイスと連携します。

21.1.1. 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。PTP によって同期されるクロックは、リーダーとフォロワーの階層で構成されています。この階層は、1 クロックごとに実行される best master clock (BMC) アルゴリズムによって自動的に作成および更新されます。フォロワークロックはリーダークロックと同期しており、フォロワークロック自体が他のダウンストリームクロックのソースになることができます。

図21.1 ネットワーク内の PTP ノード



PTP クロックの 3 つの主要なタイプについて以下に説明します。

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。タイムスタンプを書き込み、他のクロックからの時間の要求に応答します。グランドマスタークロックは、Global Navigation Satellite System (GNSS) のタイムソースと同期します。グランドマスタークロックは、ネットワーク内の時刻の信頼できるソースとして、他のすべてのデバイスに時刻同期を提供します。

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。

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。

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。

21.1.2. PTP を使用するデュアル Intel E810 NIC ハードウェアの使用

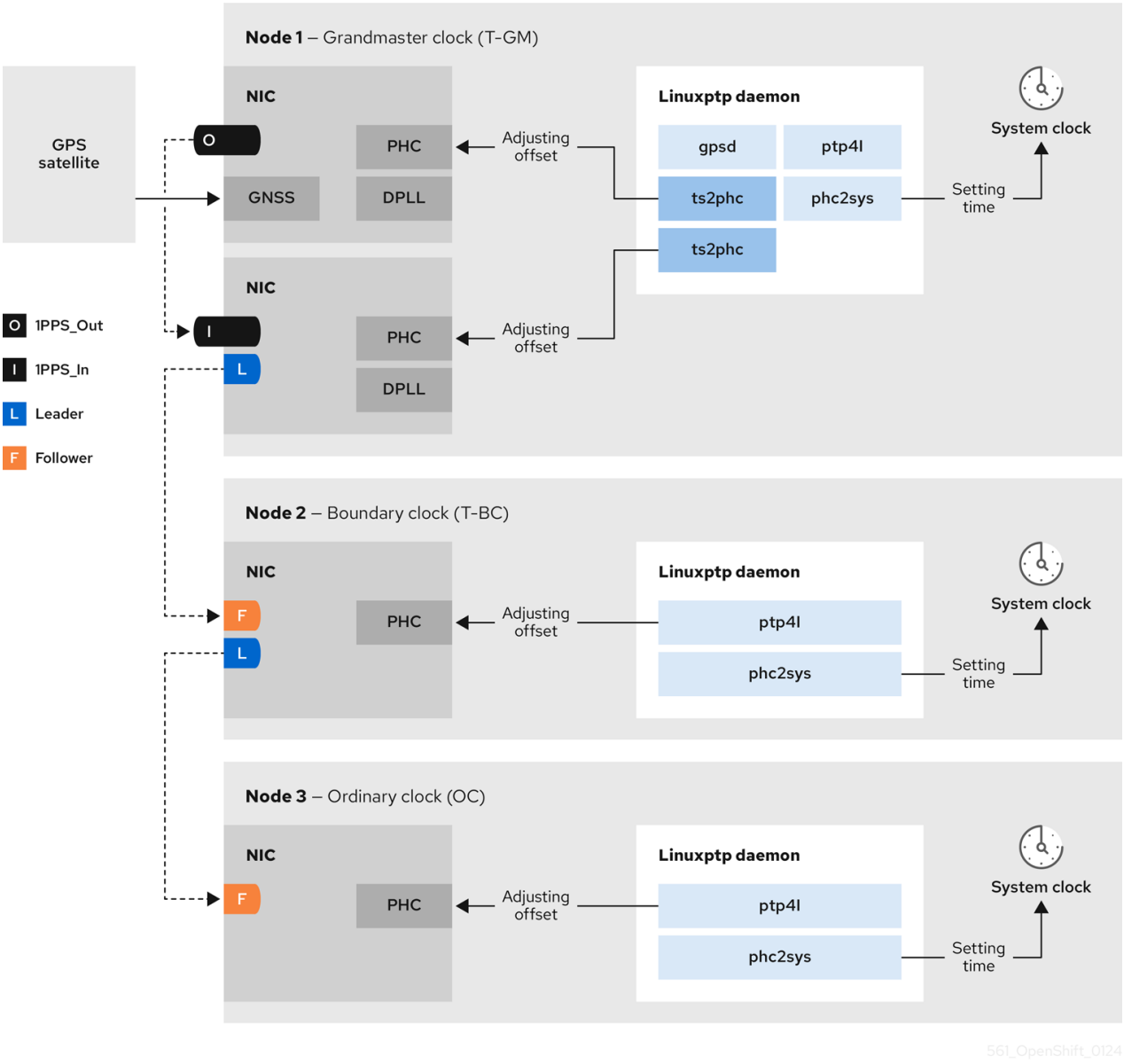
OpenShift Container Platform は、シングルおよびデュアル NIC Intel E810 ハードウェアをサポートし、グランドマスタークロック (T-GM) および境界クロック (T-BC) の高精度の PTP タイミングを実現します。

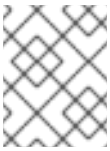
デュアル NIC グランドマスタークロック

デュアル NIC ハードウェアを備えたクラスターホストを PTP グランドマスタークロックとして使用できます。1 つ目の NIC は、Global Navigation Satellite System (GNSS) からタイミング情報を受信します。2 つ目の NIC は、E810 NIC フェイスプレート上の SMA1 Tx/Rx 接続を使用して、1 つ目の NIC からタイミング情報を受信します。クラスターホストのシステムクロックは、GNSS 衛星に接続されている NIC から同期されます。

デュアル NIC グランドマスタークロックは、Remote Radio Unit (RRU) と Baseband Unit (BBU) が同じ無線セルサイトに配置されている分散型 RAN (D-RAN) 構成の機能です。D-RAN は、コアネットワークにリンクするバックホール接続により、複数のサイトに無線機能を分散します。

図21.2 デュアル NIC グランドマスタークロック





注記

デュアル NIC T-GM 構成では、単一の **ts2phc** プロセスがシステム内の 2 つの **ts2phc** インスタンスとして報告されます。

デュアル NIC 境界クロック

ミッドバンドスペクトルカバレッジを提供する 5G 電話会社ネットワークの場合、各仮想分散ユニット (vDU) には 6 つの無線ユニット (RU) への接続が必要です。これらの接続を確立するには、各 vDU ホストに境界クロックとして設定された 2 つの NIC が必要です。

デュアル NIC ハードウェアを使用すると、各 NIC を同じアップストリームリーダークロックに接続し、NIC ごとに個別の **ptp4l** インスタンスをダウンストリームクロックに供給することができます。

21.1.3. OpenShift Container Platform ノードの linuxptp および gpsd の概要

OpenShift Container Platform は、高精度のネットワーク同期のために、PTP Operator とともに

linuxptp および **gpsd** パッケージを使用します。**linuxptp** パッケージは、ネットワーク内の PTP タイミング用のツールとデーモンを提供します。Global Navigation Satellite System (GNSS) 対応の NIC を備えたクラスターホストは、GNSS クロックソースとのインターフェイスに **gpsd** を使用します。

linuxptp パッケージには、システムクロック同期用の **ts2phc**、**pmc**、**ptp4l**、および **phc2sys** プログラムが含まれています。

ts2phc

ts2phc は、PTP デバイス間で PTP ハードウェアクロック (PHC) を高精度で同期します。 **ts2phc** はグランドマスタークロック設定で使用されます。Global Navigation Satellite System (GNSS) などの高精度クロックソースから正確なタイミング信号を受信します。GNSS は、大規模な分散ネットワークで使用するための、正確で信頼性の高い同期時刻ソースを提供します。GNSS クロックは通常、数ナノ秒の精度で時刻情報を提供します。

ts2phc システムデーモンは、グランドマスタークロックから時刻情報を読み取り、PHC 形式に変換することにより、グランドマスタークロックからのタイミング情報をネットワーク内の他の PTP デバイスに送信します。PHC 時間は、ネットワーク内の他のデバイスがクロックをグランドマスタークロックと同期させるために使用されます。

pmc

pmc は、IEEE 標準 1588.1588 に従って PTP 管理クライアント (**pmc**) を実装します。**pmc** は、**ptp4l** システムデーモンの基本的な管理アクセスを提供します。**pmc** は、標準入力から読み取り、選択されたトランスポート経由で出力を送信し、受信した応答を出力します。

ptp4l

ptp4l は、PTP 境界クロックと通常のクロックを実装し、システムデーモンとして実行されます。**ptp4l** は、以下を行います。

- ハードウェアタイムスタンプを使用して PHC をソースクロックに同期します。
- ソフトウェアタイムスタンプを使用してシステムクロックをソースクロックに同期します。

phc2sys

phc2sys は、システムクロックをネットワークインターフェイスコントローラー (NIC) 上の PHC に同期します。**phc2sys** システムデーモンは、PHC のタイミング情報を継続的に監視します。PHC はタイミングエラーを検出すると、システムクロックを修正します。

gpsd パッケージには、ホストクロックと GNSS クロックを同期するためのプログラム **ubxtool**、**gspipe**、**gpsd** が含まれています。

ubxtool

ubxtool CLI を使用すると、u-blox GPS システムと通信できます。 **ubxtool** CLI は、u-blox バイナリープロトコルを使用して GPS と通信します。

gpspipe

gpspipe は **gpsd** 出力に接続し、それを **stdout** にパイプします。

gpsd

gpsd は、ホストに接続されている 1 つ以上の GPS または AIS 受信機を監視するサービスデーモンです。

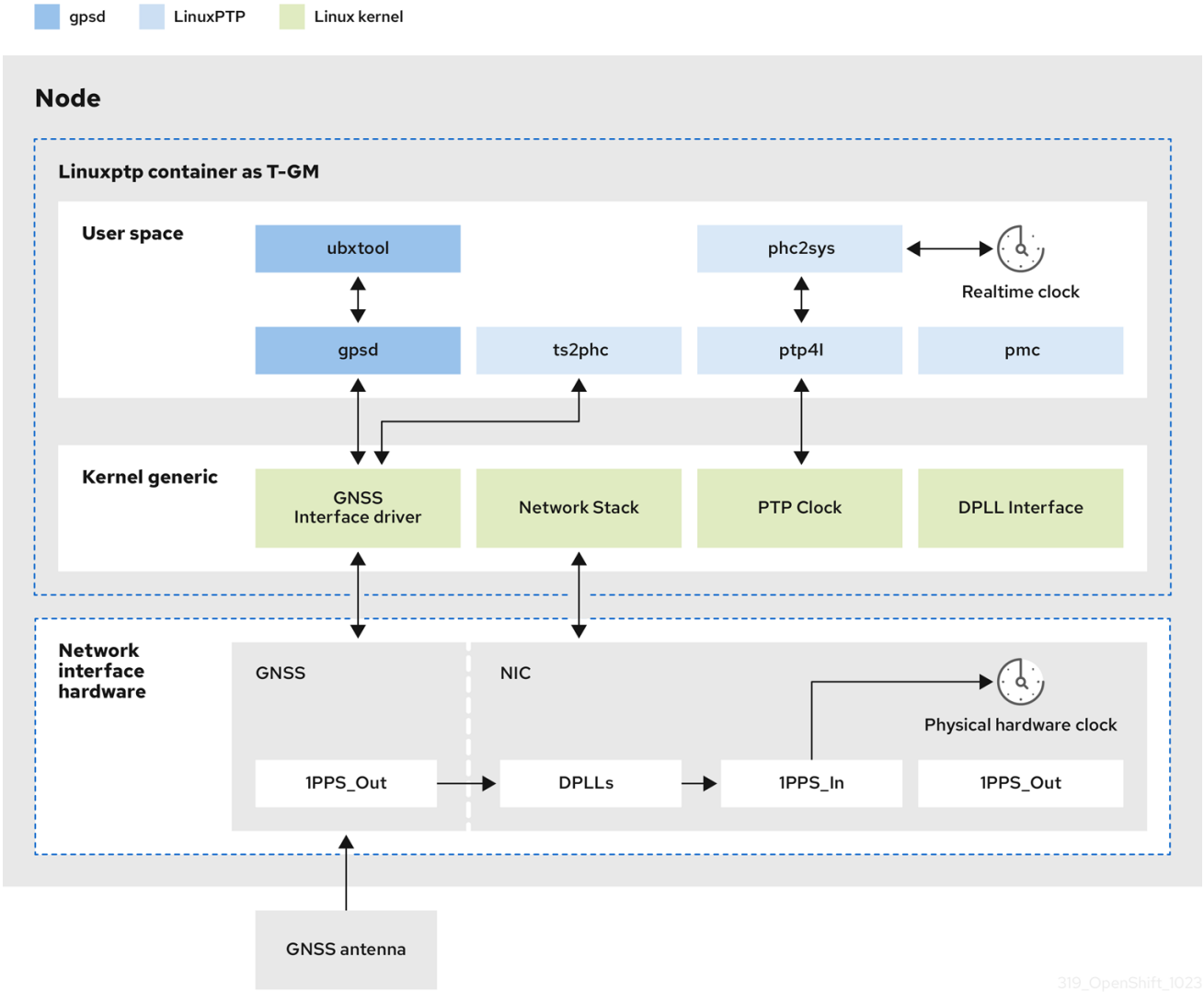
21.1.4. PTP グランドマスタークロックの GNSS タイミングの概要

OpenShift Container Platform は、クラスター内の Global Navigation Satellite System (GNSS) ソースおよびグランドマスタークロック (T-GM) からの高精度 PTP タイミングの受信をサポートします。

重要

OpenShift Container Platform は、Intel E810 Westport Channel NIC を使用した GNSS ソースからの PTP タイミングのみをサポートします。

図21.3 GNSS および T-GM との同期の概要



Global Navigation Satellite System (GNSS)

GNSS は、測位情報、ナビゲーション情報、タイミング情報を世界中の受信機に提供するために使用される衛星ベースのシステムです。PTP では、高精度で安定した基準クロックソースとして GNSS 受信機がよく使用されます。これらの受信機は、複数の GNSS 衛星から信号を受信し、正確な時刻情報を計算できます。GNSS から取得したタイミング情報は、PTP グランドマスタークロックの基準として使用されます。

GNSS を基準として使用することにより、PTP ネットワークのグランドマスタークロックは、他のデバイスに高精度のタイムスタンプを提供し、ネットワーク全体での正確な同期を可能にします。

Digital Phase-Locked Loop (DPLL)

DPLL はネットワーク内の各 PTP ノード間のクロック同期を提供します。DPLL は、ローカルシステムクロック信号の位相を、受信同期信号 (PTP グランドマスタークロックからの PTP メッセージなど) の位相と比較します。DPLL は、ローカルクロックの周波数と位相を継続的に調整して、ローカルクロックと基準クロック間の位相差を最小限に抑えます。

21.2. PTP デバイスの設定

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。

インストールが完了すると、PTP Operator はクラスターを検索して各ノードで PTP 対応の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 対応のネットワーク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。

21.2.1. 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 を作成します。
   a. 次の YAML を**ptp-namespace.yaml**ファイルに保存します。

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

   b. **namespace** CR を作成します。

```
$ oc create -f ptp-namespace.yaml
```

2. PTP Operator の Operator グループを作成します。
   a. 次の YAML を**ptp-operatorgroup.yaml**ファイルに保存します。

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

b. **OperatorGroup** CR を作成します。

```
$ oc create -f ptp-operatorgroup.yaml
```

3. PTP Operator にサブスクライブします。

a. 次の YAML を**ptp-sub.yaml**ファイルに保存します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. **Subscription** CR を作成します。

```
$ oc create -f ptp-sub.yaml
```

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-ptp -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

出力例

```
Name                         Phase
4.15.0-202301261535          Succeeded
```

21.2.2. 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 のリストから **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** プロジェクトにリスト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検査します。
- **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。

21.2.3. クラスター内の PTP 対応ネットワークデバイスの検出

- 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
```

出力例

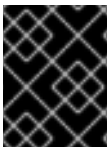
```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    creationTimestamp: "2022-01-27T15:16:28Z"
    generation: 1
    name: dev-worker-0 (1)
    namespace: openshift-ptp
    resourceVersion: "6538103"
    uid: d42fc9ad-bcbf-4590-b6d8-b676c642781a
  spec: {}
  status:
    devices: (2)
    - name: eno1
    - name: eno2
    - name: eno3
    - name: eno4
```

```
    - name: enp5s0f0
    - name: enp5s0f1
...
```

1. **name** パラメーターの値は、親ノードの名前と同じです。
2. **デバイス**コレクションには、PTP Operator がノードに対して検出した PTP 対応デバイスのリストが含まれています。

21.2.4. PTP Operator でハードウェア固有の NIC 機能を使用する

PTP 機能が組み込まれた NIC ハードウェアでは、デバイス固有の設定が必要な場合があります。**PtpConfig** カスタムリソース (CR) でプラグインを設定することで、PTP Operator でサポートされているハードウェア固有の NIC 機能を使用できます。**linuxptp-daemon** サービスが、**plugin** スタンザ内の名前付きパラメーターを使用して、特定のハードウェア設定に基づいて **linuxptp** プロセス (**ptp4l** および **phc2sys**) を開始します。



重要

OpenShift Container Platform 4.15 では、Intel E810 NIC が **PtpConfig** プラグインでサポートされています。

21.2.5. linuxptp サービスをグランドマスタークロックとして設定する

ホスト NIC を設定する **PtpConfig** カスタムリソース (CR) を作成することで、**linuxptp** サービス (**ptp4l**、**phc2sys**、**ts2phc**) をグランドマスタークロック (T-GM) として設定できます。

ts2phc ユーティリティーを使用すると、システムクロックを PTP グランドマスタークロックと同期できるため、ノードは高精度クロック信号をダウンストリームの PTP 通常クロックおよび境界クロックにストリーミングできます。

注記

次の **PtpConfig** CR の例をベースとして使用して、**linuxptp** サービスを Intel Westport Channel E810-XXVDA4T ネットワークインターフェイスの T-GM として設定してください。

PTP 高速イベントを設定するには、**ptp4lOpts**、**ptp4lConf**、**ptpClockThreshold** に適切な値を設定します。**ptpClockThreshold** は、イベントが有効になっている場合にのみ使用されます。詳細は、「PTP 高速イベント通知パブリッシャーの設定」を参照してください。

前提条件

- 実稼働環境の T-GM クロックの場合は、ベアメタルクラスターホストに Intel E810 Westport Channel NIC がインストールされている。
- 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。
- PTP Operator をインストールします。

手順

1. **PtpConfig** CR を作成します。以下に例を示します。

   a. 要件に応じて、デプロイメントに次の T-GM 設定のいずれかを使用します。YAML を **grandmaster-clock-ptp-config.yaml** ファイルに保存します。

例21.1 PTP グランドマスタークロック設定の例

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    plugins:
      e810:
        enableDefaultConfig: false
        settings:
          LocalMaxHoldoverOffSet: 1500
          LocalHoldoverTimeout: 14400
          MaxInSpecOffset: 100
        pins: $e810_pins
        #  "$iface_master":
        #    "U.FL2": "0 2"
        #    "U.FL1": "0 1"
        #    "SMA2": "0 2"
        #    "SMA1": "0 1"
        ublxCmds:
          - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
              - "-P"
              - "29.20"
              - "-z"
              - "CFG-HW-ANT_CFG_VOLTCTRL,1"
            reportOutput: false
          - args: #ubxtool -P 29.20 -e GPS
              - "-P"
              - "29.20"
              - "-e"
              - "GPS"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d Galileo
              - "-P"
              - "29.20"
              - "-d"
              - "Galileo"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d GLONASS
              - "-P"
              - "29.20"
```

```
          - "-d"
          - "GLONASS"
        reportOutput: false
      - args: #ubxtool -P 29.20 -d BeiDou
          - "-P"
          - "29.20"
          - "-d"
          - "BeiDou"
        reportOutput: false
      - args: #ubxtool -P 29.20 -d SBAS
          - "-P"
          - "29.20"
          - "-d"
          - "SBAS"
        reportOutput: false
      - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
          - "-P"
          - "29.20"
          - "-t"
          - "-w"
          - "5"
          - "-v"
          - "1"
          - "-e"
          - "SURVEYIN,600,50000"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p MON-HW
          - "-P"
          - "29.20"
          - "-p"
          - "MON-HW"
        reportOutput: true
  ts2phcOpts: " "
  ts2phcConf: |
    [nmea]
    ts2phc.master 1
    [global]
    use_syslog  0
    verbose 1
    logging_level 7
    ts2phc.pulsewidth 100000000
    ts2phc.nmea_serialport $gnss_serialport
    leapfile  /usr/share/zoneinfo/leap-seconds.list
    [$iface_master]
    ts2phc.extts_polarity rising
    ts2phc.extts_correction 0
  ptp4lConf: |
    [$iface_master]
    masterOnly 1
    [$iface_master_1]
    masterOnly 1
    [$iface_master_2]
    masterOnly 1
    [$iface_master_3]
    masterOnly 1
    [global]
```

```
#
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
```

```
    pi_integral_const 0.0
    pi_proportional_scale 0.0
    pi_proportional_exponent -0.3
    pi_proportional_norm_max 0.7
    pi_integral_scale 0.0
    pi_integral_exponent 0.4
    pi_integral_norm_max 0.3
    step_threshold 2.0
    first_step_threshold 0.00002
    clock_servo pi
    sanity_freq_limit  200000000
    ntpshm_segment 0
    #
    # Transport options
    #
    transportSpecific 0x0
    ptp_dst_mac 01:1B:19:00:00:00
    p2p_dst_mac 01:80:C2:00:00:0E
    udp_ttl 1
    udp6_scope 0x0E
    uds_address /var/run/ptp4l
    #
    # Default interface options
    #
    clock_type BC
    network_transport L2
    delay_mechanism E2E
    time_stamping hardware
    tsproc_mode filter
    delay_filter moving_median
    delay_filter_length 10
    egressLatency 0
    ingressLatency 0
    boundary_clock_jbod 0
    #
    # Clock description
    #
    productDescription ;;
    revisionData ;;
    manufacturerIdentity 00:00:00
    userDescription ;
    timeSource 0x20
recommend:
- profile: "grandmaster"
  priority: 4
  match:
  - nodeLabel: "node-role.kubernetes.io/$mcp"
```

注記

PTP グランドマスタークロック設定の例は、テストのみを目的としており、実稼働環境向けのものではありません。

例21.2 E810 NIC の PTP グランドマスタークロック設定

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_master -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    plugins:
      e810:
        enableDefaultConfig: false
        settings:
          LocalMaxHoldoverOffSet: 1500
          LocalHoldoverTimeout: 14400
          MaxInSpecOffset: 100
        pins: $e810_pins
        #  "$iface_master":
        #    "U.FL2": "0 2"
        #    "U.FL1": "0 1"
        #    "SMA2": "0 2"
        #    "SMA1": "0 1"
        ublxCmds:
          - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
              - "-P"
              - "29.20"
              - "-z"
              - "CFG-HW-ANT_CFG_VOLTCTRL,1"
            reportOutput: false
          - args: #ubxtool -P 29.20 -e GPS
              - "-P"
              - "29.20"
              - "-e"
              - "GPS"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d Galileo
              - "-P"
              - "29.20"
              - "-d"
              - "Galileo"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d GLONASS
              - "-P"
              - "29.20"
              - "-d"
              - "GLONASS"
            reportOutput: false
          - args: #ubxtool -P 29.20 -d BeiDou
              - "-P"
```

```
            - "29.20"
            - "-d"
            - "BeiDou"
          reportOutput: false
        - args: #ubxtool -P 29.20 -d SBAS
            - "-P"
            - "29.20"
            - "-d"
            - "SBAS"
          reportOutput: false
        - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
            - "-P"
            - "29.20"
            - "-t"
            - "-w"
            - "5"
            - "-v"
            - "1"
            - "-e"
            - "SURVEYIN,600,50000"
          reportOutput: true
        - args: #ubxtool -P 29.20 -p MON-HW
            - "-P"
            - "29.20"
            - "-p"
            - "MON-HW"
          reportOutput: true
    ts2phcOpts: " "
    ts2phcConf: |
      [nmea]
      ts2phc.master 1
      [global]
      use_syslog  0
      verbose 1
      logging_level 7
      ts2phc.pulsewidth 100000000
      ts2phc.nmea_serialport $gnss_serialport
      leapfile  /usr/share/zoneinfo/leap-seconds.list
      [$iface_master]
      ts2phc.extts_polarity rising
      ts2phc.extts_correction 0
    ptp4lConf: |
      [$iface_master]
      masterOnly 1
      [$iface_master_1]
      masterOnly 1
      [$iface_master_2]
      masterOnly 1
      [$iface_master_3]
      masterOnly 1
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      priority1 128
```

```
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
```

```
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      clock_servo pi
      sanity_freq_limit  200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type BC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0x20
  recommend:
  - profile: "grandmaster"
    priority: 4
    match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"
```

注記

E810 Westport Channel NIC の場合は、**ts2phc.nmea_serialport** の値を **/dev/gnss0** に設定します。

b. 以下のコマンドを実行して CR を作成します。

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

検証

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-74m2g           3/3     Running   3          4d15h   10.16.230.7  compute-
   1.example.com
   ptp-operator-5f4f48d7c-x7zkf    1/1     Running   1          4d15h   10.128.1.145  compute-
   1.example.com
   ```

   b. 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 linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
   ```

   出力例

   ```
   ts2phc[94980.334]: [ts2phc.0.config] nmea delay: 98690975 ns
   ts2phc[94980.334]: [ts2phc.0.config] ens3f0 extts index 0 at 1676577329.999999999 corr
   0 src 1676577330.901342528 diff -1
   ts2phc[94980.334]: [ts2phc.0.config] ens3f0 master offset         -1 s2 freq      -1
   ts2phc[94980.441]: [ts2phc.0.config] nmea sentence:
   GNRMC,195453.00,A,4233.24427,N,07126.64420,W,0.008,,160223,,,A,V
   phc2sys[94980.450]: [ptp4l.0.config] CLOCK_REALTIME phc offset       943 s2 freq  -
   89604 delay    504
   phc2sys[94980.512]: [ptp4l.0.config] CLOCK_REALTIME phc offset      1000 s2 freq  -
   89264 delay    474
   ```

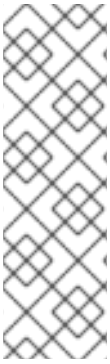
21.2.6. linuxptp サービスをデュアル E810 Westport Channel NIC のグランドマスタークロックとして設定する

ホスト NIC を設定する **PtpConfig** カスタムリソース (CR) を作成することで、**linuxptp** サービス (**ptp4l**、**phc2sys**、**ts2phc**) をデュアル E810 Westport Channel NIC のグランドマスタークロック (T-GM) として設定できます。

分散型 RAN (D-RAN) のユースケースでは、次の方法でデュアル NIC の PTP を設定できます。

- NIC 1 を、Global Navigation Satellite System (GNSS) のタイムソースに同期します。
- NIC 2 を、NIC 1 によって提供される 1PPS タイミング出力に同期します。この設定は、**PtpConfig** CR の PTP ハードウェアプラグインによって提供します。

デュアル NIC PTP T-GM 構成では、1 つの **ptp4l** インスタンスと、各 NIC に 1 つずつ、2 つの **ts2phc** インスタンスを報告する 1 つの **ts2phc** プロセスを使用します。ホストのシステムクロックは、GNSS タイムソースに接続されている NIC から同期されます。



注記

次の **PtpConfig** CR の例をベースとして使用して、**linuxptp** サービスをデュアル Intel Westport Channel E810-XXVDA4T ネットワークインターフェイスの T-GM として設定してください。

PTP 高速イベントを設定するには、**ptp4lOpts**、**ptp4lConf**、**ptpClockThreshold** に適切な値を設定します。**ptpClockThreshold** は、イベントが有効になっている場合にのみ使用されます。詳細は、「PTP 高速イベント通知パブリッシャーの設定」を参照してください。

前提条件

- 実稼働環境の T-GM クロックの場合は、ベアメタルクラスターホストに 2 つの Intel E810 Westport Channel NIC がインストールされている。
- 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。
- PTP Operator をインストールします。

手順

1. **PtpConfig** CR を作成します。以下に例を示します。
   a. 次の YAML を **grandmaster-clock-ptp-config-dual-nics.yaml** ファイルに保存します。

例21.3 デュアル E810 NIC の PTP グランドマスタークロック設定

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -O -37 -N 8 -R 16 -s $iface_nic1 -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    plugins:
      e810:
        enableDefaultConfig: false
        settings:
          LocalMaxHoldoverOffSet: 1500
          LocalHoldoverTimeout: 14400
          MaxInSpecOffset: 100
        pins:
         "$iface_nic1":
           "U.FL2": "0 2"
           "U.FL1": "0 1"
```

```
    "SMA2": "0 2"
    "SMA1": "2 1"
  "$iface_nic2":
    "U.FL2": "0 2"
    "U.FL1": "0 1"
    "SMA2": "0 2"
    "SMA1": "1 1"
ublxCmds:
  - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
      - "-P"
      - "29.20"
      - "-z"
      - "CFG-HW-ANT_CFG_VOLTCTRL,1"
    reportOutput: false
  - args: #ubxtool -P 29.20 -e GPS
      - "-P"
      - "29.20"
      - "-e"
      - "GPS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d Galileo
      - "-P"
      - "29.20"
      - "-d"
      - "Galileo"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d GLONASS
      - "-P"
      - "29.20"
      - "-d"
      - "GLONASS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d BeiDou
      - "-P"
      - "29.20"
      - "-d"
      - "BeiDou"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d SBAS
      - "-P"
      - "29.20"
      - "-d"
      - "SBAS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
      - "-P"
      - "29.20"
      - "-t"
      - "-w"
      - "5"
      - "-v"
      - "1"
      - "-e"
      - "SURVEYIN,600,50000"
    reportOutput: true
  - args: #ubxtool -P 29.20 -p MON-HW
```

```
          - "-P"
          - "29.20"
          - "-p"
          - "MON-HW"
        reportOutput: true
    ts2phcOpts: " "
    ts2phcConf: |
      [nmea]
      ts2phc.master 1
      [global]
      use_syslog  0
      verbose 1
      logging_level 7
      ts2phc.pulsewidth 100000000
      #cat /dev/GNSS to find available serial port
      #example value of gnss_serialport is /dev/ttyGNSS_1700_0
      ts2phc.nmea_serialport $gnss_serialport
      leapfile  /usr/share/zoneinfo/leap-seconds.list
      [$iface_nic1]
      ts2phc.extts_polarity rising
      ts2phc.extts_correction 0
      [$iface_nic2]
      ts2phc.master 0
      ts2phc.extts_polarity rising
      #this is a measured value in nanoseconds to compensate for SMA cable delay
      ts2phc.extts_correction -10
    ptp4lConf: |
      [$iface_nic1]
      masterOnly 1
      [$iface_nic1_1]
      masterOnly 1
      [$iface_nic1_2]
      masterOnly 1
      [$iface_nic1_3]
      masterOnly 1
      [$iface_nic2]
      masterOnly 1
      [$iface_nic2_1]
      masterOnly 1
      [$iface_nic2_2]
      masterOnly 1
      [$iface_nic2_3]
      masterOnly 1
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      priority1 128
      priority2 128
      domainNumber 24
      #utc_offset 37
      clockClass 6
      clockAccuracy 0x27
      offsetScaledLogVariance 0xFFFF
      free_running 0
```

```
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit  200000000
ntpshm_segment 0
```

```
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type BC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 1
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0x20
  recommend:
  - profile: "grandmaster"
    priority: 4
    match:
    - nodeLabel: "node-role.kubernetes.io/$mcp"
```

注記

E810 Westport Channel NIC の場合は、**ts2phc.nmea_serialport** の値を **/dev/gnss0** に設定します。

b. 以下のコマンドを実行して CR を作成します。

```
$ oc create -f grandmaster-clock-ptp-config-dual-nics.yaml
```

検証

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-74m2g         3/3     Running   3          4d15h   10.16.230.7  compute-
1.example.com
ptp-operator-5f4f48d7c-x7zkf  1/1     Running   1          4d15h   10.128.1.145  compute-
1.example.com
```

b. 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 linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
```

出力例

```
ts2phc[509863.660]: [ts2phc.0.config] nmea delay: 347527248 ns
ts2phc[509863.660]: [ts2phc.0.config] ens2f0 extts index 0 at 1705516553.000000000
corr 0 src 1705516553.652499081 diff 0
ts2phc[509863.660]: [ts2phc.0.config] ens2f0 master offset          0 s2 freq      -0
I0117 18:35:16.000146 1633226 stats.go:57] state updated for ts2phc =s2
I0117 18:35:16.000163 1633226 event.go:417] dpll State s2, gnss State s2, tsphc state
s2, gm state s2,
ts2phc[1705516516]:[ts2phc.0.config] ens2f0 nmea_status 1 offset 0 s2
GM[1705516516]:[ts2phc.0.config] ens2f0 T-GM-STATUS s2
ts2phc[509863.677]: [ts2phc.0.config] ens7f0 extts index 0 at 1705516553.000000010
corr -10 src 1705516553.652499081 diff 0
ts2phc[509863.677]: [ts2phc.0.config] ens7f0 master offset          0 s2 freq      -0
I0117 18:35:16.016597 1633226 stats.go:57] state updated for ts2phc =s2
phc2sys[509863.719]: [ptp4l.0.config] CLOCK_REALTIME phc offset        -6 s2 freq
+15441 delay    510
phc2sys[509863.782]: [ptp4l.0.config] CLOCK_REALTIME phc offset        -7 s2 freq
+15438 delay    502
```

関連情報

- PTP 高速イベント通知パブリッシャーの設定

21.2.6.1. グランドマスタークロックの PtpConfig 設定リファレンス

このリファレンスでは、**linuxptp** サービス (**ptp4l**、**phc2sys**、**ts2phc**) をグランドマスタークロックとして設定する **PtpConfig** カスタムリソース (CR) の設定オプションについて説明します。

表21.1 PTP グランドマスタークロックの PtpConfig 設定オプション

PtpConfig CR フィールド	説明

PtpConfig CR フィールド	説明
plugins	grandmaster クロック動作用に NIC を設定する **.exec.cmdline** オプションの配列を指定します。grandmaster クロック設定では、特定の PTP ピンを無効にする必要があります。 プラグインメカニズムにより、PTP Operatro は自動ハードウェア設定を行うことができます。Intel Westport Channel NIC の場合、**enableDefaultConfig** が true の場合、PTP Operator はハードコーディングされたスクリプトを実行して、NIC に必要な設定を実行します。
ptp4lOpts	**ptp4l** サービスのシステム設定オプションを指定します。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。
ptp4lConf	**ptp4l** をグランドマスター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、**ens2f1** インターフェイスは、ダウンストリームに接続されたデバイスを同期します。グランドマスタークロックの場合、**ClockClass** を **6** に設定し、**ClockAccuracy** を **0x27** に設定します。Global Navigation Satellite System (GNSS) からタイミング信号を受信する場合は、**timeSource** を **0x20** に設定します。
tx_timestamp_timeout	データを破棄する前に、送信者からの送信 (TX) タイムスタンプを待機する最大時間を指定します。
boundary_clock_jbod	JBOD 境界クロック時間遅延値を指定します。この値は、ネットワーク時間デバイス間で受け渡される時間値を修正するために使用されます。
phc2sysOpts	**phc2sys** サービスのシステム設定オプションを指定します。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。  **注記** ここにリストされているネットワークインターフェイスがグランドマスターとして設定されており、**ts2phcConf** および **ptp4lConf** フィールドで必要に応じて参照されていることを確認してください。
ptpSchedulingPolicy	**ptp4l** および **phc2sys** プロセスのスケジューリングポリシーを設定します。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。
ptpSchedulingPriority	**ptpSchedulingPolicy** 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 優先順位を設定するには、1~65 の整数値を設定します。**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。

PtpConfig CR フィールド	説明
ptpClockThreshold	オプション: **ptpClockThreshold**スタンザが存在しない場合には、**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 **ptpClockThreshold**値を示します。**ptpClockThreshold** 値は、PTP マスタークロックが切断されてから PTP イベントが発生するまでの時間を設定します。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4l**) の値と比較するナノ秒単位のオフセット値を設定します。**ptp4l** または **phc2sys** のオフセット値がこの範囲外の場合、PTP クロックの状態が **FREERUN** に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が **LOCKED** に設定されます。
ts2phcConf	**ts2phc** コマンドの設定を設定します。 **Leapfile** は、PTP Operator コンテナーイメージ内の現在のうるう秒定義ファイルへのデフォルトパスです。 **ts2phc.nmea_serialport** は、NMEA GPS クロックソースに接続されているシリアルポートデバイスです。設定すると、**/dev/gnss<id>** で GNSS 受信機にアクセスできるようになります。ホストに複数の GNSS 受信機がある場合は、次のいずれかのデバイスを列挙することで正しいデバイスを見つけることができます。 ● **/sys/class/net/<eth_port>/device/gnss/** ● **/sys/class/gnss/gnss<id>/device/**
ts2phcOpts	**ts2phc** コマンドのオプションを設定します。
recommend	**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。
.recommend.profile	**profile** セクションで定義されている **.recommend.profile** オブジェクト名を指定します。
.recommend.priority	**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。
.recommend.match	**.recommend.match** ルールを **nodeLabel** または **nodeName** の値に指定します。
.recommend.match.nodeLabel	**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** フィールドの **key** で **nodeLabel** を設定します。例: **node-role.kubernetes.io/worker**。

PtpConfig CR フィールド	説明
.recommend.match.nodeName	**oc get nodes** コマンドを使用して、**nodeName** をノードオブジェクトの **node.Name** フィールドの値に設定します。**compute-1.example.com** はその例です。

21.2.6.2. グランドマスタークロッククラスの同期状態のリファレンス

次の表では、PTP グランドマスタークロック (T-GM) の **gm.ClockClass** の状態について説明します。クロッククラスの状態では、Primary Reference Time Clock (PRTC) またはその他のタイミングソースに関連する精度と安定性に基づいて T-GM クロックが分類されます。

ホールドオーバー仕様とは、PTP クロックがプライマリータイムソースから更新を受信せずに同期を維持できる時間です。

表21.2 T-GM クロッククラスの状態

クロッククラスの状態	説明
gm.ClockClass 6	T-GM クロックは **LOCKED** モードで PRTC に接続しています。たとえば、PRTC は GNSS タイムソースまで追跡できます。
gm.ClockClass 7	T-GM クロックは **HOLDOVER** モードであり、ホールドオーバー仕様の範囲内にあります。クロックソースはカテゴリー 1 の周波数ソースまで追跡できない場合があります。
gm.ClockClass 140	T-GM クロックは **HOLDOVER** モードであり、ホールドオーバー仕様の範囲外にありますが、カテゴリー 1 の周波数ソースまで追跡可能です。
gm.ClockClass 248	T-GM クロックは **FREERUN** モードです。

詳細は、"Phase/time traceability information", ITU-T G.8275.1/Y.1369.1 Recommendations を参照してください。

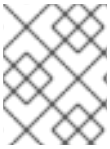
21.2.6.3. Intel Westport Channel E810 ハードウェア設定リファレンス

ここでは、Intel E810-XXVDA4T ハードウェアプラグイン を使用して E810 ネットワークインターフェイスを PTP グランドマスタークロックとして設定する方法を説明します。ハードウェアピンの設定により、ネットワークインターフェイスがシステム内の他のコンポーネントやデバイスとどのようにやり取りするかが決まります。E810-XXVDA4T NIC には、外部 1PPS 信号用の 4 つのコネクター (**SMA1**、**SMA2**、**U.FL1**、および **U.FL2)** があります。

表21.3 Intel E810 NIC ハードウェアコネクターの設定

ハードウェアピン	推奨設定	説明
U.FL1	**0 1**	**U.FL1** コネクター入力を無効にします。**U.FL1** コネクターは出力専用です。

ハードウェアピン	推奨設定	説明
U.FL2	**0 2**	**U.FL2** コネクター出力を無効にします。**U.FL2** コネクターは入力専用です。
SMA1	**0 1**	**SMA1** コネクター入力を無効にします。**SMA1** コネクターは双方向です。
SMA2	**0 2**	**SMA2** コネクター出力を無効にします。**SMA2** コネクターは双方向です。



注記

SMA1 コネクターと **U.FL1** コネクターはチャネル 1 を共有します。**SMA2** コネクターと **U.FL2** コネクターはチャネル 2 を共有します。

PtpConfig カスタムリソース (CR) で GNSS クロックを設定するには、**spec.profile.plugins.e810.ublxCmds** パラメーターを設定します。これらの **ublxCmds** スタンザはそれぞれ、**ubxtool** コマンドを使用してホスト NIC に適用する設定と対応しています。以下に例を示します。

```
ublxCmds:
  - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
    - "-P"
    - "29.20"
    - "-z"
    - "CFG-HW-ANT_CFG_VOLTCTRL,1"
   reportOutput: false
```

次の表では、同等の **ubxtool** コマンドについて説明します。

表21.4 Intel E810 ublxCmds の設定

ubxtool コマンド	説明
ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1	アンテナ電圧制御を有効にします。アンテナのステータスを **UBX-MON-RF** および **UBX-INF-NOTICE** ログメッセージで報告できるようにします。
ubxtool -P 29.20 -e GPS	アンテナが GPS 信号を受信できるようにします。
ubxtool -P 29.20 -d Galileo	Galileo GPS 衛星から信号を受信するようにアンテナを設定します。
ubxtool -P 29.20 -d GLONASS	アンテナが GLONASS GPS 衛星から信号を受信できないようにします。

ubxtool コマンド	説明
ubxtool -P 29.20 -d BeiDou	アンテナが BeiDou GPS 衛星から信号を受信できないようにします。
ubxtool -P 29.20 -d SBAS	アンテナが SBAS GPS 衛星から信号を受信できないようにします。
ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000	GNSS 受信機のサーベイインプロセスを設定して、初期位置の推定を改善します。最適な結果が得られるまでに最大 24 時間かかる場合があります。
ubxtool -P 29.20 -p MON-HW	ハードウェアの自動スキャンを 1 回実行し、NIC の状態と構成設定を報告します。

E810 プラグインは次のインターフェイスを実装します。

表21.5 E810 プラグインインターフェイス

インターフェイス	説明
OnPTPConfigChangeE810	**PtpConfig** CR を更新するたびに実行されます。この機能は、プラグインオプションを解析し、設定データに基づいて必要な設定をネットワークデバイスピンに適用します。
AfterRunPTPCommandE810	PTP プロセスを起動して **gpspipe** PTP コマンドを実行した後に実行されます。この機能は、プラグインオプションを処理して **ubxtool** コマンドを実行し、出力をプラグイン固有のデータに保存します。
PopulateHwConfigE810	**PtpConfig** CR 内のハードウェア固有のデータに基づいて **NodePtpDevice** CR を設定します。

E810 プラグインには次の構造体と変数があります。

表21.6 E810 プラグインの構造体と変数

構造体	説明
E810Opts	E810 プラグインのオプション (ブール値フラグやネットワークデバイスピンのマップなど) を表します。
E810UblxCmds	ブール値フラグとコマンド引数の文字列のスライスを使用して、**ubxtool** コマンドの設定を表します。
E810PluginData	プラグインの実行中に使用されるプラグイン固有のデータを保持します。

21.2.6.4. デュアル E810 Westport Channel NIC 設定リファレンス

ここでは、Intel E810-XXVDA4T ハードウェアプラグイン を使用して、E810 ネットワークインターフェイスのペアを PTP グランドマスタークロック (T-GM) として設定する方法を説明します。

デュアル NIC クラスターホストを設定する前に、1PPS フェイスプレート接続を使用して 2 つの NIC を SMA1 ケーブルで接続する必要があります。

デュアル NIC T-GM を設定する際は、SMA1 接続ポートを使用して NIC を接続する場合に発生する 1PPS 信号遅延を補正する必要があります。ケーブルの長さ、周囲温度、コンポーネントと製作公差などのさまざまな要因が信号遅延に影響を与える可能性があります。遅延を補正するには、信号遅延のオフセットに使用する特定の値を計算する必要があります。

表21.7 E810 デュアル NIC T-GM PtpConfig CR リファレンス

PtpConfig フィールド	説明
spec.profile.plugins.e810.pins	PTP Operator E810 ハードウェアプラグインを使用して E810 ハードウェアピンを設定します。 • ピン **2 1** は、NIC 1 上の**SMA1** の **1PPS OUT** 接続を有効にします。 • ピン **1 1** は、NIC 2 上の**SMA1** の **1PPS IN** 接続を有効にします。
spec.profile.ts2phcConf	**ts2phcConf** フィールドは、NIC 1 と NIC 2 のパラメーターを設定するために使用します。NIC 2 には **ts2phc.master 0** を設定します。これにより、GNSS ではなく 1PPS 入力から NIC 2 のタイミングソースが設定されます。使用する特定の SMA ケーブルおよびケーブルの長さにより発生する遅延を補正するには、NIC 2 の **ts2phc.extts_correction** 値を設定します。設定する値は、具体的な測定値と SMA1 ケーブルの長さによって異なります。
spec.profile.ptp4lConf	複数の NIC のサポートを有効にするには、**boundary_lock_jbod** の値を 1 に設定します。

21.2.7. linuxptp サービスを境界クロックとして設定

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。

注記

次の例の **PtpConfig** CR を、特定のハードウェアおよび環境の境界クロックとして **linuxptp** サービスを設定する基礎として使用します。この例の CR は PTP 高速イベントを設定しません。PTP 高速イベントを設定するには、**ptp4lOpts**、**ptp4lConf**、**ptpClockThreshold** に適切な値を設定します。**ptpClockThreshold** は、イベントが有効になっている場合にのみ使用されます。詳細は、「PTP 高速イベント通知パブリッシャーの設定」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。

- 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。

   PTP 境界クロックの設定例

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: boundary-clock
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4lConf: |
        # The interface name is hardware-specific
        [$iface_slave]
        masterOnly 0
        [$iface_master_1]
        masterOnly 1
        [$iface_master_2]
        masterOnly 1
        [$iface_master_3]
        masterOnly 1
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 248
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison G.8275.x
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
```

```
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval -4
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
```

```
        udp6_scope 0x0E
        uds_address /var/run/ptp4l
        #
        # Default interface options
        #
        clock_type BC
        network_transport L2
        delay_mechanism E2E
        time_stamping hardware
        tsproc_mode filter
        delay_filter moving_median
        delay_filter_length 10
        egressLatency 0
        ingressLatency 0
        boundary_clock_jbod 0
        #
        # Clock description
        #
        productDescription ;;
        revisionData ;;
        manufacturerIdentity 00:00:00
        userDescription ;
        timeSource 0xA0
    recommend:
      - profile: boundary-clock
        priority: 4
        match:
          - nodeLabel: "node-role.kubernetes.io/$mcp"
```

表21.8 PTP 境界クロックの CR 設定オプション

CR フィールド	説明
name	**PtpConfig** CR の名前。
profile	1 つ以上の **profile** オブジェクトの配列を指定します。
name	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。
ptp4lOpts	**ptp4l** サービスのシステム設定オプションを指定します。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。
ptp4lConf	**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** はグランド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。
<interface_1>	同期クロックを受信するインターフェイス。
<interface_2>	Synchronization クロックを送信するインターフェイス。

CR フィールド	説明
tx_timestamp_timeout	Intel Columbiaville 800 Series NIC の場合、**tx_timestamp_timeout** を **50** に設定します。
boundary_clock_jbod	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。
phc2sysOpts	**phc2sys** サービスのシステム設定オプションを指定します。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。
ptpSchedulingPolicy	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。
ptpSchedulingPriority	**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。
ptpClockThreshold	オプション: **ptpClockThreshold** が存在しない場合、**ptpClockThreshold** フィールドにはデフォルト値が使用されます。**ptpClockThreshold** は、PTP マスタークロックが切断されてから PTP イベントが発生するまでの時間を設定します。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4l**) の値と比較するナノ秒単位のオフセット値を設定します。**ptp4l** または **phc2sys** のオフセット値がこの範囲外の場合、PTP クロックの状態が **FREERUN** に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が **LOCKED** に設定されます。
recommend	**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。
.recommend.profile	**profile** セクションで定義される **.recommend.profile** オブジェクト名を指定します。
.recommend.priority	**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。
.recommend.match	**.recommend.match** ルールを **nodeLabel** または **nodeName** の値に指定します。

CR フィールド	説明
.recommend.match.nodeLabel	**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** フィールドの **key** で **nodeLabel** を設定します。例: **node-role.kubernetes.io/worker**。
.recommend.match.nodeName	**oc get nodes** コマンドを使用して、**nodeName** をノードオブジェクトの **node.Name** フィールドの値に設定します。**compute-1.example.com** はその例です。

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

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 linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
```

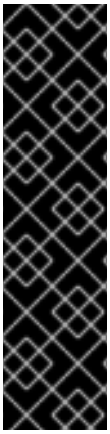
出力例

```
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 -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
```

関連情報

- PTP ハードウェアの FIFO 優先スケジューリングの設定
- PTP 高速イベント通知パブリッシャーの設定

21.2.7.1. linuxptp サービスをデュアル NIC ハードウェアの境界クロックとして設定



重要

デュアル NIC で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 製品サポート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場合があります。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。

NIC ごとに **PtpConfig** カスタムリソース (CR) オブジェクトを作成することにより、**linuxptp** サービス (**ptp4l**、**phc2sys**) をデュアル NIC ハードウェアの境界クロックとして設定できます。

デュアル NIC ハードウェアを使用すると、各 NIC を同じアップストリームリーダークロックに接続し、NIC ごとに個別の **ptp4l** インスタンスをダウンストリームクロックに供給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ログインしている。
- PTP Operator をインストールします。

手順

1. linuxptp サービスを境界クロックとして設定の参照 CR を各 CR の基礎として使用して、NIC ごとに 1 つずつ、2 つの個別の **PtpConfigCR** を作成します。以下に例を示します。

   a. **phc2sysOpts** の値を指定して、**boundary-clock-ptp-config-nic1.yaml** を作成します。

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic1
  namespace: openshift-ptp
spec:
  profile:
  - name: "profile1"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: | (1)
      [ens5f1]
      masterOnly 1
      [ens5f0]
```

```
    masterOnly 0
  ...
  phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
```

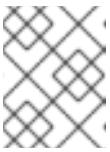
1 **ptp4l** を境界クロックとして開始するために必要なインターフェイスを指定します。たとえば、**ens5f0** はグランドマスタークロックから同期し、**ens5f1** は接続された機器から同期します。

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。

b. **boundary-clock-ptp-config-nic2.yaml** を作成し、**phc2sysOpts** フィールドを完全に削除して、2 番目の NIC の **phc2sys** サービスを無効にします。

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
  - name: "profile2"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: | 1
      [ens7f1]
      masterOnly 1
      [ens7f0]
      masterOnly 0
...
```

1 2 番目の NIC の境界クロックとして **ptp4l** を開始するために必要なインターフェイスを指定します。



注記

2 番目の NIC で **phc2sys** サービスを無効にするには、2 番目の **PtpConfig** CR から **phc2sysOpts** フィールドを完全に削除する必要があります。

2. 次のコマンドを実行して、デュアル NIC **PtpConfigCR** を作成します。

a. 1 番目の NIC の PTP を設定する CR を作成します。

```
$ oc create -f boundary-clock-ptp-config-nic1.yaml
```

b. 2 番目の NIC の PTP を設定する CR を作成します。

```
$ oc create -f boundary-clock-ptp-config-nic2.yaml
```

検証

- PTP Operator が両方の NIC に **PtpConfigCR** を適用していることを確認してください。デュアル NIC ハードウェアがインストールされているノードに対応する **linuxptp** デーモンのログを調べます。たとえば、以下のコマンドを実行します。

```
$ oc logs linuxptp-daemon-cvgr6 -n openshift-ptp -c linuxptp-daemon-container
```

出力例

```
ptp4l[80828.335]: [ptp4l.1.config] master offset          5 s2 freq   -5727 path delay       519
ptp4l[80828.343]: [ptp4l.0.config] master offset         -5 s2 freq  -10607 path delay       533
phc2sys[80828.390]: [ptp4l.0.config] CLOCK_REALTIME phc offset         1 s2 freq  -87239
delay    539
```

21.2.8. linuxptp サービスを通常のクロックとして設定

Ptp Config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**) を通常のクロックとして設定できます。

注記

次の例の **PtpConfig** CR を、特定のハードウェアおよび環境の通常クロックとして **linuxptp** サービスを設定する基礎として使用します。この例の CR は PTP 高速イベントを設定しません。PTP 高速イベントを設定するには、**ptp4lOpts**、**ptp4lConf**、**ptpClockThreshold** に適切な値を設定します。**ptpClockThreshold** は、イベントが有効な場合にのみ必要です。詳細は、「PTP 高速イベント通知パブリッシャーの設定」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。
- 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。

PTP 通常クロックの設定例

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ordinary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: ordinary-clock
      # The interface name is hardware-specific
      interface: $interface
      ptp4lOpts: "-2 -s"
```

```
    phc2sysOpts: "-a -r -n 24"
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    ptp4lConf: |
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      slaveOnly 1
      priority1 128
      priority2 128
      domainNumber 24
      #utc_offset 37
      clockClass 255
      clockAccuracy 0xFE
      offsetScaledLogVariance 0xFFFF
      free_running 0
      freq_est_interval 1
      dscp_event 0
      dscp_general 0
      dataset_comparison G.8275.x
      G.8275.defaultDS.localPriority 128
      #
      # Port Data Set
      #
      logAnnounceInterval -3
      logSyncInterval -4
      logMinDelayReqInterval -4
      logMinPdelayReqInterval -4
      announceReceiptTimeout 3
      syncReceiptTimeout 0
      delayAsymmetry 0
      fault_reset_interval -4
      neighborPropDelayThresh 20000000
      masterOnly 0
      G.8275.portDS.localPriority 128
      #
      # Run time options
      #
      assume_two_step 0
      logging_level 6
      path_trace_enabled 0
      follow_up_info 0
      hybrid_e2e 0
      inhibit_multicast_service 0
      net_sync_monitor 0
      tc_spanning_tree 0
      tx_timestamp_timeout 50
      unicast_listen 0
      unicast_master_table 0
      unicast_req_duration 3600
      use_syslog 1
      verbose 0
```

```
      summary_interval 0
      kernel_leap 1
      check_fup_sync 0
      clock_class_threshold 7
      #
      # Servo Options
      #
      pi_proportional_const 0.0
      pi_integral_const 0.0
      pi_proportional_scale 0.0
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      max_frequency 900000000
      clock_servo pi
      sanity_freq_limit 200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type OC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
recommend:
  - profile: ordinary-clock
    priority: 4
    match:
      - nodeLabel: "node-role.kubernetes.io/$mcp"
```

表21.9 PTP 通常クロック CR 設定のオプション

CR フィールド	説明
name	**PtpConfig** CR の名前。
profile	1 つ以上の **profile** オブジェクトの配列を指定します。各プロファイルの名前は一意である必要があります。
interface	**ptp4l** サービスで使用するネットワークインターフェイスを指定します (例: **ens787f1**)。
ptp4lOpts	**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。このインターフェイスで PTP 高速イベントを使用するには、**--summary_interval -4** を追加します。
phc2sysOpts	**phc2sys** サービスのシステム設定オプションを指定します。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。Intel Columbiaville 800 Series NIC の場合、**phc2sysOpts** オプションを **-a -r -m -n 24 -N 8 -R 16** に設定します。**-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。
ptp4lConf	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。
tx_timestamp_timeout	Intel Columbiaville 800 Series NIC の場合、**tx_timestamp_timeout** を **50** に設定します。
boundary_clock_jbod	Intel Columbiaville 800 Series NIC の場合、**boundary_clock_jbod** を **0** に設定します。
ptpSchedulingPolicy	**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。
ptpSchedulingPriority	**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。

CR フィールド	説明
ptpClockThreshold	オプション: **ptpClockThreshold** が存在しない場合、**ptpClockThreshold** フィールドにはデフォルト値が使用されます。**ptpClockThreshold** は、PTP マスタークロックが切断されてから PTP イベントが発生するまでの時間を設定します。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4l**) の値と比較するナノ秒単位のオフセット値を設定します。**ptp4l** または **phc2sys** のオフセット値がこの範囲外の場合、PTP クロックの状態が **FREERUN** に設定されます。オフセット値がこの範囲内にある場合、PTP クロックの状態が **LOCKED** に設定されます。
recommend	**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。
.recommend.profile	**profile** セクションで定義される **.recommend.profile** オブジェクト名を指定します。
.recommend.priority	通常クロックの **.recommend.priority** を **0** に設定します。
.recommend.match	**.recommend.match** ルールを **nodeLabel** または **nodeName** の値に指定します。
.recommend.match.nodeLabel	**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** フィールドの **key** で **nodeLabel** を設定します。例: **node-role.kubernetes.io/worker**。
.recommend.match.nodeName	**oc get nodes** コマンドを使用して、**nodeName** をノードオブジェクトの **node.Name** フィールドの値に設定します。**compute-1.example.com** はその例です。

2. 次のコマンドを実行して、 **PtpConfig** CR を作成します。

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

検証

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 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: ens787f1
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2 -s
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
```

関連情報

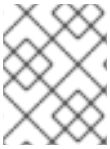
- PTP ハードウェアの FIFO 優先スケジューリングの設定
- PTP 高速イベント通知パブリッシャーの設定

21.2.8.1. PTP の通常クロックリファレンスとしての Intel Columbiaville E800 シリーズ NIC

次の表では、Intel Columbiaville E800 シリーズ NIC を通常のクロックとして使用するために、PTP リファレンス設定に加える必要がある変更について説明します。クラスターに適用する **PtpConfig** カスタムリソース (CR) に変更を加えます。

表21.10 Intel Columbiaville NIC の推奨 PTP 設定

PTP 設定	推奨設定
phc2sysOpts	**-a -r -m -n 24 -N 8 -R 16**
tx_timestamp_timeout	**50**
boundary_clock_jbod	**0**



注記

phc2sysOpts の場合、**-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。

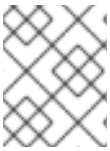
関連情報

- PTP 高速イベントを使用して **linuxptp** サービスを通常のクロックとして設定する CR の詳細な例は、linuxptp サービスを通常のクロックとして設定する を参照してください。

21.2.9. 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のある通信業者などのデプロイメントタイプでは、PTP デーモンスレッドが、残りのインフラストラクチャーコンポーネントとともに、限られた CPU リソースで実行されます。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 Config**CR に **SCHED_FIFO** が設定されている場合には、**ptp4l** と **phc2sys** は、**Ptp Config**CR の **ptp Scheduling Priority** フィールドで設定された優先順位で、**chrt** の下の親コンテナーで実行されます。



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-ptp
```

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

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
  - name: "profile1"
...
    ptpSchedulingPolicy: SCHED_FIFO (1)
    ptpSchedulingPriority: 10 (2)
```

(1) **ptp4l** 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。**ptp4l** 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 **linuxptp-daemon** Pod と対応するノードの名前を取得します。

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

出力例

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

2. **ptp4l** 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-ptp logs linuxptp-daemon-lgm55 -c linuxptp-daemon-container|grep chrt
```

出力例

```
I1216 19:24:57.091872 1600715 daemon.go:285] /bin/chrt -f 65 /usr/sbin/ptp4l -f
/var/run/ptp4l.0.config -2  --summary_interval -4 -m
```

21.2.10. linuxptp サービスのログフィルタリングの設定

linuxptp デーモンは、デバッグに使用できるログを生成します。ストレージ容量が制限されている通信業者などのデプロイメントタイプでは、これらのログによりストレージ需要が増大する可能性があります。

ログメッセージの数を減らすために、**PtpConfig** カスタムリソース (CR) を設定して、 **master offset** 値をレポートするログメッセージを除外できます。**master offset** ログメッセージは、現在のノードのクロックとマスタークロックの違いをナノ秒単位でレポートします。

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。
- PTP Operator をインストールします。

手順

1. **PtpConfig** CR を編集します。

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

2. **spec.profile** で、**ptpSettings.logReduce** 仕様を追加し、値を **true** に設定します。

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

```
spec:
  profile:
  - name: "profile1"
...
    ptpSettings:
      logReduce: "true"
```

注記

デバッグの目的で、この仕様を **False** に戻すと、マスターオフセットメッセージを含めることができます。

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 **linuxptp-daemon** Pod と対応するノードの名前を取得します。

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

出力例

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

2. 次のコマンドを実行して、マスターオフセットメッセージがログから除外されていることを確認します。

```
$ oc -n openshift-ptp logs <linux_daemon_container> -c linuxptp-daemon-container | grep
"master offset" 1
```

1 <linux_daemon_container> は **linuxptp-daemon** Pod の名前です (例: **linuxptp-daemon-gmv2n**)。

logReduce 仕様を設定する場合、このコマンドは **linuxptp** デーモンのログに **master offset** のインスタンスを報告しません。

21.2.11. 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-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
```

注記

PTP 高速イベントバスが有効な場合には、準備できた **linuxptp-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-ptp 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-ptp 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ー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** コンテナー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>
   診断するコンテナーです (例: **linuxptp-daemon-lmvgn**)。

   c. **linuxptp-daemon** コンテナー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.fffe.166ef0-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
    portIdentity            40a6b7.fffe.166ef0-1
    portState               SLAVE
    logMinDelayReqInterval  -4
    peerMeanPathDelay       0
    logAnnounceInterval     -3
    announceReceiptTimeout  3
    logSyncInterval         -4
    delayMechanism          1
    logMinPdelayReqInterval -4
    versionNumber           2
```

5. GNSS をソースとするグランドマスタークロックの場合は、次のコマンドを実行して、ツリー内 NIC ice ドライバーが正しいことを確認します。

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-74m2g ethtool -i ens7f0
```

出力例

```
driver: ice
version: 5.14.0-356.bz2232515.el9.x86_64
firmware-version: 4.20 0x8001778b 1.3346.0
```

6. GNSS をソースとするグランドマスタークロックの場合は、**linuxptp-daemon** コンテナーが GNSS アンテナから信号を受信していることを確認します。コンテナーが GNSS 信号を受信していない場合、**/dev/gnss0** ファイルにデータが入力されません。検証するには、次のコマンドを実行します。

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-jnz6r cat /dev/gnss0
```

出力例

```
$GNRMC,125223.00,A,4233.24463,N,07126.64561,W,0.000,,300823,,,A,V*0A
$GNVTG,,T,,M,0.000,N,0.000,K,A*3D
$GNGGA,125223.00,4233.24463,N,07126.64561,W,1,12,99.99,98.6,M,-33.1,M,,*7E
$GNGSA,A,3,25,17,19,11,12,06,05,04,09,20,,,99.99,99.99,99.99,1*37
$GPGSV,3,1,10,04,12,039,41,05,31,222,46,06,50,064,48,09,28,064,42,1*62
```

21.2.12. PTP Operator データの収集

oc adm must-gather コマンドを使用すると、PTP Operator に関連する機能やオブジェクトなど、クラスターに関する情報を収集できます。

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。
- PTP Operator がインストールされている。

手順

- **must-gather** を使用して PTP Operator データを収集するには、PTP Operator **must-gather** イメージを指定する必要があります。

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/ptp-must-gather-rhel8:v4.15
```

21.3. PTP ハードウェア高速イベント通知フレームワークの使用

仮想 RAN (vRAN) 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。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。

21.3.1. PTP およびクロック同期エラーイベントについて

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 ノード上で実行されている vRAN 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 (pub-sub メッセージング) は、非同期のサービス間通信アーキテクチャーです。このアーキテクチャーでは、トピックにパブリッシュされたメッセージが、そのトピックのすべてのサブスクライバーによって即座に受信されます。

PTP Operator は、すべての PTP 対応ネットワークインターフェイスの高速イベント通知を生成します。イベントには、HTTP またはアドバンストメッセージキュープロトコル (AMQP) メッセージバス経由で **cloud-event-proxy** サイドカーコンテナーを使用してアクセスできます。

注記

PTP 高速イベント通知は、PTP 通常クロック、PTP グランドマスタークロック、または PTP 境界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使用できます。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

21.3.2. PTP 高速イベント通知フレームワークについて

Precision Time Protocol (PTP) 高速イベント通知フレームワークを使用して、ベアメタルクラスターノードが生成する PTP イベントにクラスターアプリケーションをサブスクライブします。

注記

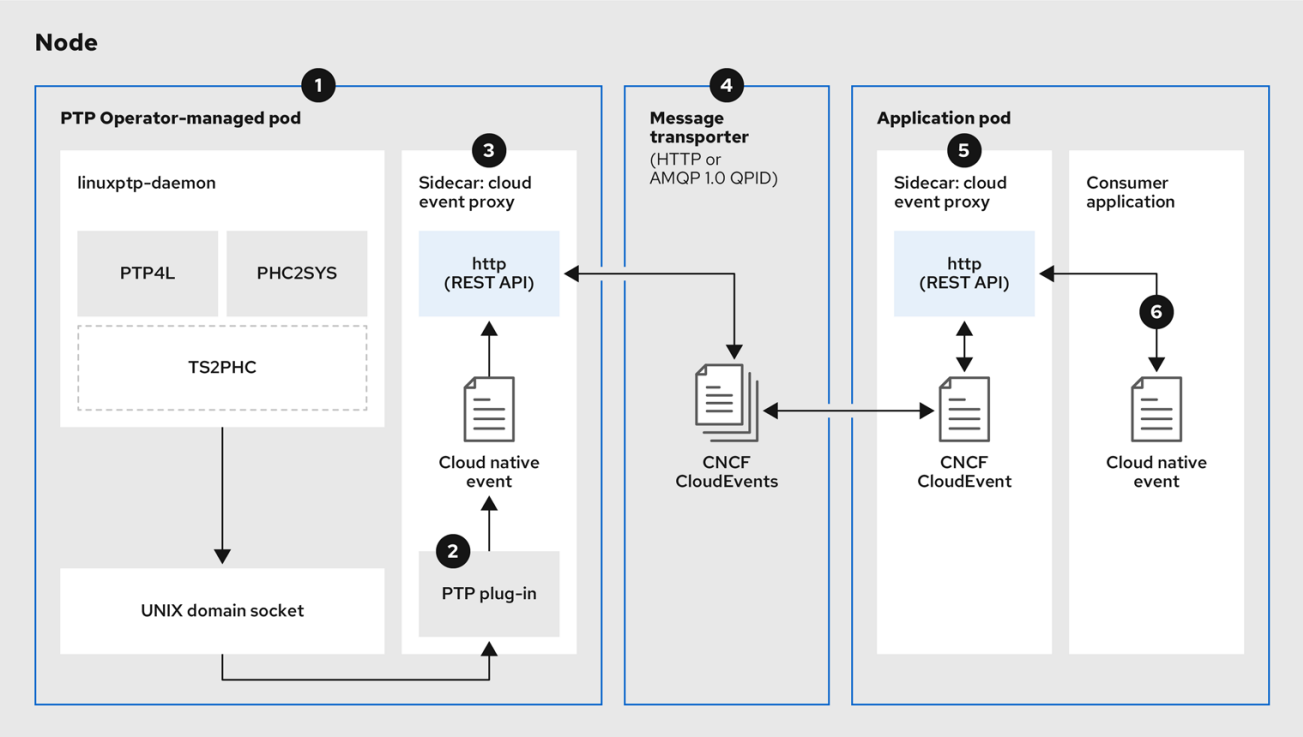
高速イベント通知フレームワークは、通信に REST API を使用します。REST API は、O-RAN ALLIANCE 仕様 から入手できる **O-RAN O-Cloud Notification API Specification for Event Consumers 3.0** に基づいています。

このフレームワークは、パブリッシャー、サブスクライバー、および AMQ または HTTP メッセージングプロトコルで構成され、パブリッシャーとサブスクライバーのアプリケーション間の通信を処理します。アプリケーションは、**cloud-event-proxy** コンテナーをサイドカーパターンで実行して、PTP イベントをサブスクライブします。**cloud-event-proxy** サイドカーコンテナーは、プライマリーアプリケーションのリソースをまったく使用せずに、大幅な待機時間なしで、プライマリーアプリケーションコンテナーと同じリソースにアクセスできます。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

図21.4 PTP 高速イベントの概要



1 イベントはクラスターホストで生成されます。

PTP Operator が管理する Pod の **linuxptp-daemon は**、Kubernetes **DaemonSet** として実行され、さまざまな **linuxptp** プロセス (**ptp4l**、**phc2sys**、およびオプションでグランドマスタークロック用の **ts2phc**) を管理します。**linuxptp-daemon** は、イベントを UNIX ドメインソケットに渡します。

2 イベントが cloud-event-proxy サイドカーに渡されます。

PTP プラグインは、UNIX ドメインソケットからイベントを読み取り、PTP Operator が管理する Pod 内の **cloud-event-proxy** サイドカーに渡します。**cloud-event-proxy** は、イベントを Kubernetes インフラストラクチャーから Cloud-Native Network Functions (CNF) に低レイテンシーで配信します。

3 **イベントが永続化される**

PTP Operator が管理する Pod 内の **cloud-event-proxy** サイドカーは、REST API を使用してイベントを処理し、クラウドネイティブイベントを発行します。

4 **メッセージはトランスポートされます。**

メッセージトランスポーターは、HTTP または AMQP 1.0 QPID を介して、アプリケーション Pod 内の **cloud-event-proxy** サイドカーにイベントを転送します。

5 **イベントは REST API から入手できます。**

アプリケーション Pod の **cloud-event-proxy** サイドカーはイベントを処理し、REST API を使用して利用できるようにします。

6 **コンシューマーアプリケーションがサブスクリプションをリクエストし、サブスクライブされたイベントを受信します**

コンシューマーアプリケーションは、API 要求をアプリケーション Pod の **cloud-event-proxy** サイドカーに送信して、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で指定されたリソースの AMQ または HTTP メッセージングリスナープロトコルを作成します。

アプリケーション Pod の **cloud-event-proxy** サイドカーは、PTP Operator が管理する Pod からイベントを受信し、クラウドイベントオブジェクトをラッピング解除してデータを取得し、イベントをコンシューマーアプリケーションにポストします。コンシューマーアプリケーションは、リソース修飾子で指定されたアドレスをリッスンし、PTP イベントを受信して処理します。

21.3.3. 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 がインストールされている。

手順

1. デフォルトの PTP Operator 設定を変更して、PTP 高速イベントを有効にします。
   a. 次の YAML を**ptp-operatorconfig.yaml**ファイルに保存します。

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
```

```
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    enableEventPublisher: true (1)
```

(1) **enableEventPublisher** を **true** に設定して、PTP 高速イベント通知を有効にします。

注記

OpenShift Container Platform 4.13 以降では、PTP イベントに HTTP トランスポートを使用するときに、**PtpOperatorConfig** リソースの **spec.ptpEventConfig.transportHost** フィールドを設定する必要はありません。PTP イベントに AMQP トランスポートを使用する場合にのみ、**transportHost** を設定します。

a. **PtpOperatorConfig** CR を更新します。

```
$ oc apply -f ptp-operatorconfig.yaml
```

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** および **ptp4lOpts** に必要な値を設定します。次の YAML は、**PtpConfig** CR で設定する必要のある値 (必須) を示しています。

```
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 **ptpClockThreshold** 値を示します。**ptpClockThreshold** 値は、PTP マスタークロックが PTP イベントが発生する前に切断されてからの期間を設定します。**holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。**maxOffsetThreshold** および **minOffsetThreshold** 設定は、**CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4l**) の値と比較するナノ秒単位のオフセット値を設定します。**ptp4l** または **phc2sys** のオフセット値がこの範囲外の場合、PTP クロックの状態が **FREERUN** に設定されま

す。オフセット値がこの範囲内にある場合、PTP クロックの状態が **LOCKED** に設定されます。

関連情報

- PTP 高速イベントを使用して **linuxptp** サービスを通常のクロックとして設定する CR の詳細な例は、linuxptp サービスを通常のクロックとして設定する を参照してください。

21.3.4. PTP またはベアメタルイベントに HTTP トランスポートを使用するためのコンシューマーアプリケーションの移行

以前に PTP またはベアメタルイベントのコンシューマーアプリケーションをデプロイしている場合は、HTTP メッセージトランスポートを使用するようにアプリケーションを更新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。
- PTP Operator または Bare Metal Event Relay を、デフォルトで HTTP トランスポートを使用するバージョン 4.13 以降に更新している。

手順

1. HTTP トランスポートを使用するようにイベントコンシューマーアプリケーションを更新します。クラウドイベントサイドカーデプロイメントの **http-event-publishers** 変数を設定します。
   たとえば、PTP イベントが設定されているクラスターでは、以下の YAML スニペットはクラウドイベントサイドカーデプロイメントを示しています。

```
containers:
  - name: cloud-event-sidecar
    image: cloud-event-sidecar
    args:
      - "--metrics-addr=127.0.0.1:9091"
      - "--store-path=/store"
      - "--transport-host=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
      - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043" (1)
      - "--api-port=8089"
```

   (1) PTP Operator は、PTP イベントを生成するホストに対して **NODE_NAME** を自動的に解決します。**compute-1.example.com** はその例です。

   ベアメタルイベントが設定されているクラスターでは、クラウドイベントサイドカーデプロイメント CR で **http-event-publishers** フィールドを **hw-event-publisher-service.openshift-bare-metal-events.svc.cluster.local:9043** に設定します。

2. **consumer-events-subscription-service** サービスをイベントコンシューマーアプリケーションと併せてデプロイします。以下に例を示します。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

21.3.5. 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ライバー間で 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。AMQ メッセージングを使用するには、AMQ Interconnect Operator をインストールする必要があります。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

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
```

21.3.6. REST API を使用して PTP イベントに DU アプリケーションをサブスクライブする

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-consumerDUアプリケーションコンテナーと**cloud-event-proxy**サイドカーコンテナーを別々の 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**コンテナー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、DU アプリケーション Pod の **http://localhost:8089/api/ocloudNotifications/v1/** にある **cloud-event-proxy** コンテナーによってポストされた PTP イベントに **cloud-event-consumer** DU アプリケーションをサブスクライブします。

- **/api/ocloudNotifications/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/ocloudNotifications/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/ocloudNotifications/v1/health**
  - **GET**: **ocloudNotifications** API の正常性ステータスを返します
- **api/ocloudNotifications/v1/publishers**
  - **GET**: クラスターノードの **os-clock-sync-state**、**ptp-clock-class-change**、**lock-state**、および **gnss-sync-status** メッセージの配列を返します。
- **/api/ocloudnotifications/v1/<resource_address>/CurrentState**
  - **GET**: **os-clock-sync-state**、**ptp-clock-class-change**、**lock-state**、または **gnss-state-change** イベントタイプの現在の状態を返します。

注記

9089 は、アプリケーション Pod にデプロイされた **cloud-event-consumer** コンテナー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。

21.3.6.1. PTP イベント REST API リファレンス

PTP イベント通知 REST API を使用して、親ノードで生成された PTP イベントにクラスターアプリケーションをサブスクライブします。

21.3.6.1.1. api/ocloudNotifications/v1/subscriptions

HTTP メソッド
GET api/ocloudNotifications/v1/subscriptions

説明
サブスクリプションのリストを返します。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/ocloudNotifications/v1/subscriptions/75b1ad8f-c807-4c23-
acf5-56f4b7ee3826",
  "resource": "/cluster/node/compute-1.example.com/ptp"
 }
]
```

HTTP メソッド
POST api/ocloudNotifications/v1/subscriptions

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21.11 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/ocloudNotifications/v1/subscriptions",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

21.3.6.1.2. api/ocloudNotifications/v1/subscriptions/<subscription_id>

HTTP メソッド
GET api/ocloudNotifications/v1/subscriptions/<subscription_id>

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21.12 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/ocloudNotifications/v1/subscriptions/48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "resource":"/cluster/node/compute-1.example.com/ptp"
}
```

21.3.6.1.3. api/ocloudNotifications/v1/health

HTTP メソッド
GET api/ocloudNotifications/v1/health/

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

API 応答の例

```
OK
```

21.3.6.1.4. api/ocloudNotifications/v1/publishers

HTTP メソッド
GET api/ocloudNotifications/v1/publishers

説明
クラスターノードの **os-clock-sync-state**、**ptp-clock-class-change**、**lock-state**、および **gnss-sync-status** の詳細の配列を返します。関連する機器の状態が変化すると、システムは通知を生成します。

- **os-clock-sync-state** 通知は、ホストオペレーティングシステムのクロック同期状態を示します。**LOCKED** または **FREERUN** 状態になります。
- **ptp-clock-class-change** 通知は、PTP クロッククラスの現在の状態を示します。
- **lock-state** 通知は、PTP 機器のロック状態の現在のステータスを示します。**LOCKED**、**HOLDOVER**、または **FREERUN** 状態になります。
- **gnss-sync-status** 通知は、外部 GNSS クロック信号に関する GPS 同期状態を示します。**LOCKED** または **FREERUN** 状態になります。

機器の同期ステータスのサブスクリプションを組み合わせて使用すると、システム全体の同期状態の詳細なビューを提供できます。

API 応答の例

```
[
  {
    "id": "0fa415ae-a3cf-4299-876a-589438bacf75",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/0fa415ae-a3cf-4299-876a-589438bacf75",
    "resource": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state"
  },
  {
    "id": "28cd82df-8436-4f50-bbd9-7a9742828a71",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/28cd82df-8436-4f50-bbd9-7a9742828a71",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change"
  },
  {
    "id": "44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state"
  },
  {
    "id": "778da345d-4567-67b0-a43f0-rty885a456",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/778da345d-4567-67b0-a43f0-rty885a456",
    "resource": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status"
  }
]
```

cloud-event-proxy コンテナーのログには、**os-clock-sync-state**、**ptp-clock-class-change**、**lock-state**、および **gnss-sync-status** イベントが含まれています。以下に例を示します。

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

os-clock-sync-state イベントの例

```
{
  "id":"c8a784d1-5f4a-4c16-9a81-a3b4313affe5",
  "type":"event.sync.sync-status.os-clock-sync-state-change",
  "source":"/cluster/compute-1.example.com/ptp/CLOCK_REALTIME",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.906277159Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/sync-status/os-clock-sync-state",
        "dataType":"notification",
```

```
            "valueType":"enumeration",
            "value":"LOCKED"
          },
          {
            "resource":"/sync/sync-status/os-clock-sync-state",
            "dataType":"metric",
            "valueType":"decimal64.3",
            "value":"-53"
          }
        ]
      }
    }
```


ptp-clock-class-change イベントの例


```
{
  "id":"69eddb52-1650-4e56-b325-86d44688d02b",
  "type":"event.sync.ptp-status.ptp-clock-class-change",
  "source":"/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.147100033Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/ptp-status/ptp-clock-class-change",
        "dataType":"metric",
        "valueType":"decimal64.3",
        "value":"135"
      }
    ]
  }
}
```


lock-state イベントの例


```
{
  "id":"305ec18b-1472-47b3-aadd-8f37933249a9",
  "type":"event.sync.ptp-status.ptp-state-change",
  "source":"/cluster/compute-1.example.com/ptp/ens2fx/master",
  "dataContentType":"application/json",
  "time":"2022-05-06T15:31:23.467684081Z",
  "data":{
    "version":"v1",
    "values":[
      {
        "resource":"/sync/ptp-status/lock-state",
        "dataType":"notification",
        "valueType":"enumeration",
        "value":"LOCKED"
      },
      {
        "resource":"/sync/ptp-status/lock-state",
        "dataType":"metric",
        "valueType":"decimal64.3",
```

```
          "value":"62"
        }
      ]
    }
}
```

gnss-sync-status イベントの例

```
{
  "id": "435e1f2a-6854-4555-8520-767325c087d7",
  "type": "event.sync.gnss-status.gnss-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",
  "dataContentType": "application/json",
  "time": "2023-09-27T19:35:33.42347206Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens2fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/ens2fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "5"
      }
    ]
  }
}
```

21.3.6.1.5. api/ocloudNotifications/v1/<resource_address>/CurrentState

HTTP メソッド
GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/lock-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change/CurrentState

説明
クラスターノードの **os-clock-sync-state**、**ptp-clock-class-change**、**lock-state** イベントの現在の状態を返すように **CurrentState** API エンドポイントを設定します。

- **os-clock-sync-state** 通知は、ホストオペレーティングシステムのクロック同期状態を示します。**LOCKED** または **FREERUN** 状態になります。
- **ptp-clock-class-change** 通知は、PTP クロッククラスの現在の状態を示します。

- **lock-state** 通知は、PTP 機器のロック状態の現在のステータスを示します。**LOCKED**、**HOLDOVER**、または **FREERUN** 状態になります。

表21.13 クエリーパラメーター

パラメーター	型
<resource_address>	string

ロック状態 API レスポンスの例

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

os-clock-sync-state API レスポンスの例

```
{
  "specversion": "0.3",
  "id": "4f51fe99-feaa-4e66-9112-66c5c9b9afcb",
  "source": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "subject": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "datacontenttype": "application/json",
  "time": "2022-11-29T17:44:22.202Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
```

```
      },
      {
        "resource": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "27"
      }
    ]
  }
}
```


ptp-clock-class-change API レスポンスの例


```
{
  "id": "064c9e67-5ad4-4afb-98ff-189c6aa9c205",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
```


21.3.7. PTP 高速イベントメトリックのモニタリング

linuxptp-daemon が実行されているクラスターノードから PTP 高速イベントメトリクスを監視できます。事前に設定された自己更新型の Prometheus モニタ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。
- PTP 対応ハードウェアを搭載したノードに PTP Operator をインストールし、設定します。

手順

1. 次のコマンドを実行して、ノードのデバッグ Pod を起動します。

```
$ oc debug node/<node_name>
```

2. **linuxptp-daemon** コンテナーによって公開された PTP メトリックを確認します。たとえば、以下のコマンドを実行します。

```
sh-4.4# curl http://localhost:9091/metrics
```

出力例

```
# HELP cne_api_events_published Metric to get number of events published by the rest api
# TYPE cne_api_events_published gauge
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",status="success"} 1
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",status="success"} 94
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/ptp-clock-class-change",status="success"} 18
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",status="success"} 27
```

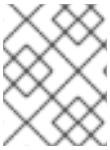
3. OpenShift Container Platform Web コンソールで PTP イベントを表示するには、クエリーする PTP メトリクスの名前 (例: **openshift_ptp_offset_ns**) をコピーします。
4. OpenShift Container Platform Web コンソールで、**Observe → Metrics** をクリックします。
5. PTP メトリクスを **Expression** フィールドに貼り付け、**Run queries** をクリックします。

関連情報

- メトリクスの管理

21.3.8. PTP 高速イベントメトリクスのリファレンス

次の表では、**linuxptp-daemon** サービスが実行されているクラスターノードから利用できる PTP 高速イベントメトリクスについて説明します。



注記

次のメトリクスの一部は、PTP グランドマスタークロック (T-GM) にのみ適用されます。

表21.14 PTP 高速イベントメトリクス

メトリクス	説明	例
openshift_ptp_clock_class	インターフェイスの PTP クロッククラスを返します。PTP クロッククラスの可能な値は、6 (**LOCKED**)、7 (**PRC UNLOCKED IN-SPEC**)、52 (**PRC UNLOCKED OUT-OF-SPEC**)、187 (**PRC UNLOCKED OUT-OF-SPEC**)、135 (**T-BC HOLDOVER IN-SPEC**)、165 (**T-BC HOLDOVER OUT-OF-SPEC**)、248 (**DEFAULT**)、または 255 (**SLAVE ONLY CLOCK**) です。T-GM クロックにのみ適用されます。	**{node="compute-1.example.com",process="ptp4l"} 6**

メトリクス	説明	例
openshift_ptp_clock_state	インターフェイスの現在の PTP クロック状態を返します。PTP クロック状態の可能な値は、**FREERUN**、**LOCKED**、または **HOLDOVER** です。	**{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1**
openshift_ptp_delay_ns	タイミングパケットを送信するプライマリークロックとタイミングパケットを受信するセカンダリークロックの間の遅延をナノ秒単位で返します。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0**
openshift_ptp_frequency_adjustment_ns	2 つの PTP クロック間の周波数調整をナノ秒単位で返します。たとえば、アップストリームクロックと NIC の間、システムクロックと NIC の間、または PTP ハードウェアクロック (**phc**) と NIC の間などです。T-GM クロックにのみ適用されます。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -6768**
openshift_ptp_frequency_status	NIC の Digital Phase-Locked Loop (DPLL) 周波数の現在のステータスを返します。可能な値は、-1 (**UNKNOWN**)、0 (**INVALID**)、1 (**FREERUN**)、2 (**LOCKED**)、3 (**LOCKED_HO_ACQ**)、または 4 (**HOLDOVER**) です。	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_phase_status	NIC の DPLL 位相のステータスを返します。可能な値は、-1 (**UNKNOWN**)、0 (**INVALID**)、1 (**FREERUN**)、2 (**LOCKED**)、3 (**LOCKED_HO_ACQ**)、または 4 (**HOLDOVER**) です。	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_interface_role	インターフェイスに設定された PTP クロックの役割を返します。可能な値は、0 (**PASSIVE**)、1 (**SLAVE**)、2 (**MASTER**)、3 (**FAULTY**)、4 (**UNKNOWN**)、または 5 (**LISTENING**) です。	**{iface="ens2f0", node="compute-1.example.com", process="ptp4l"} 2**
openshift_ptp_max_offset_ns	2 つのクロックまたはインターフェイス間の最大オフセットをナノ秒単位で返します。たとえば、アップストリーム GNSS クロックと NIC (**ts2phc**) の間、または PTP ハードウェアクロック (**phc**) とシステムクロック (**phc2sys**) の間などです。T-GM クロックにのみ適用されます。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 1.038099569e+09**
openshift_ptp_offset_ns	DPLL クロックまたは GNSS クロックソースと NIC ハードウェアクロック間のオフセットをナノ秒単位で返します。T-GM クロックにのみ適用されます。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -9**

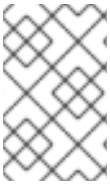
メトリクス	説明	例
openshift_ptp_process_restart_count	**ptp4l** プロセスが再起動した回数を返します。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_process_status	PTP プロセスが実行中かどうかを示すステータスコードを返します。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_threshold	**HoldOverTimeout**、**MaxOffsetThreshold**、および **MinOffsetThreshold** の値を返します。 ● **holdOverTimeout** は、PTP マスタークロックが切断されたときに、PTP クロックイベントの状態が **FREERUN** に変わるまでの時間値 (秒単位) です。 ● **maxOffsetThreshold** および **minOffsetThreshold** は、NIC の **PtpConfig** CR で設定した **CLOCK_REALTIME** (**phc2sys**) またはマスターオフセット (**ptp4l**) の値と比較されるナノ秒単位のオフセット値です。	**{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5**
openshift_ptp_pps_status	NIC 1PPS 接続の現在のステータスを返します。1PPS 接続は、接続された NIC 間のタイミングを同期するために使用します。可能な値は 0 (**UNAVAILABLE**) と 1 (**AVAILABLE**) です。	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1**
openshift_ptp_nmea_status	NMEA 接続の現在のステータスを返します。NMEA は、1PPS NIC 接続に使用されるプロトコルです。可能な値は 0 (**UNAVAILABLE**) と 1 (**AVAILABLE**) です。	**{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1**
openshift_ptp_gnss_status	Global Navigation Satellite System (GNSS) 接続の現在のステータスを返します。GNSS は、衛星ベースの測位、ナビゲーション、およびタイミングサービスを世界中に提供します。可能な値は、0 (**NOFIX**)、1 (**DEAD RECKONING ONLY**)、2 (**2D-FIX**)、3 (**3D-FIX**)、4 (**GPS+DEAD RECKONING FIX**)、5 (**TIME ONLY FIX**) です。	**{from="gnss",iface="ens2fx",node="compute-1.example.com",process="gnss"} 3**

21.4. PTP イベントコンシューマーアプリケーションの開発

ベアメタルクラスターノードで Precision Time Protocol (PTP)イベントを使用するコンシューマーアプリケーションを開発する場合は、コンシューマーアプリケーションと **cloud-event-proxy** コンテナーを別のアプリケーション Pod にデプロイする必要があります。**cloud-event-proxy** コンテナーは、PTP

Operator Pod からイベントを受信し、これをコンシューマーアプリケーションに渡します。コンシューマーアプリケーションは、REST API を使用して **cloud-event-proxy** コンテナーで投稿されたイベントにサブスクライブします。

PTP イベントアプリケーションのデプロイに関する詳細は、PTP 高速イベント通知フレームワークについて を参照してください。



注記

以下の情報は、PTP イベントを使用するコンシューマーアプリケーションを開発するための一般的なガイダンスです。完全なイベントコンシューマーアプリケーションの例は、この情報の範囲外です。

21.4.1. PTP イベントコンシューマーアプリケーションのリファレンス

PTP イベントコンシューマーアプリケーションには次の機能が必要です。

1. **POST** ハンドラーで実行され、クラウドネイティブ PTP イベントの JSON ペイロードを受信する Web サービス
2. PTP イベントプロデューサーをサブスクライブするための **createSubscription** 関数
3. PTP イベントプロデューサーの現在の状態をポーリングする **getCurrentState** 関数

次の Go スニペットの例は、これらの要件を示しています。

Go での PTP イベントコンシューマーサーバー関数の例

```
func server() {
  http.HandleFunc("/event", getEvent)
  http.ListenAndServe("localhost:8989", nil)
}

func getEvent(w http.ResponseWriter, req *http.Request) {
  defer req.Body.Close()
  bodyBytes, err := io.ReadAll(req.Body)
  if err != nil {
    log.Errorf("error reading event %v", err)
  }
  e := string(bodyBytes)
  if e != "" {
    processEvent(bodyBytes)
    log.Infof("received event %s", string(bodyBytes))
  } else {
    w.WriteHeader(http.StatusNoContent)
  }
}
```

PTP イベントの例 Go の createSubscription 関数

```
import (
"github.com/redhat-cne/sdk-go/pkg/pubsub"
"github.com/redhat-cne/sdk-go/pkg/types"
v1pubsub "github.com/redhat-cne/sdk-go/v1/pubsub"
)
```

```
// Subscribe to PTP events using REST API
s1,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state") (1)
s2,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change")
s3,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/lock-state")

// Create PTP event subscriptions POST
func createSubscription(resourceAddress string) (sub pubsub.PubSub, err error) {
 var status int
    apiPath:= "/api/ocloudNotifications/v1/"
    localAPIAddr:=localhost:8989 // vDU service API address
    apiAddr:= "localhost:8089" // event framework API address

 subURL := &types.URI{URL: url.URL{Scheme: "http",
  Host: apiAddr
  Path: fmt.Sprintf("%s%s", apiPath, "subscriptions")}}
 endpointURL := &types.URI{URL: url.URL{Scheme: "http",
  Host: localAPIAddr,
  Path: "event"}}

 sub = v1pubsub.NewPubSub(endpointURL, resourceAddress)
 var subB []byte

 if subB, err = json.Marshal(&sub); err == nil {
  rc := restclient.New()
  if status, subB = rc.PostWithReturn(subURL, subB); status != http.StatusCreated {
   err = fmt.Errorf("error in subscription creation api at %s, returned status %d", subURL, status)
  } else {
   err = json.Unmarshal(subB, &sub)
  }
 } else {
  err = fmt.Errorf("failed to marshal subscription for %s", resourceAddress)
 }
 return
}
```

(1) **<node_name>** を、PTP イベントを生成しているノードの FQDN に置き換えます。 **compute-1.example.com** はその例です。

Go の PTP イベントコンシューマー getCurrentState 関数の例

```
//Get PTP event state for the resource
func getCurrentState(resource string) {
 //Create publisher
 url := &types.URI{URL: url.URL{Scheme: "http",
  Host: localhost:8989,
  Path: fmt.SPrintf("/api/ocloudNotifications/v1/%s/CurrentState",resource)}}
 rc := restclient.New()
 status, event := rc.Get(url)
 if status != http.StatusOK {
  log.Errorf("CurrentState:error %d from url %s, %s", status, url.String(), event)
 } else {
  log.Debugf("Got CurrentState: %s ", event)
 }
}
```

-

21.4.2. cloud-event-proxy のデプロイメントとサービス CR を参照する

PTP イベントコンシューマーアプリケーションをデプロイするときは、次の **cloud-event-proxy** デプロイメントとサブスクライバサービス CR の例を参考として使用してください。

注記

HTTP トランスポートは、PTP およびベアメタルイベントのデフォルトのトランスポートです。可能な場合、PTP およびベアメタルイベントには AMQP ではなく HTTP トランスポートを使用してください。AMQ Interconnect は、2024 年 6 月 30 日で EOL になります。AMQ Interconnect の延長ライフサイクルサポート (ELS) は 2029 年 11 月 29 日に終了します。詳細は、Red Hat AMQ Interconnect のサポートステータス を参照してください。

HTTP トランスポートを使用した cloud-event-proxy デプロイメントの参照

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: event-consumer-deployment
  namespace: <namespace>
  labels:
    app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
    spec:
      serviceAccountName: sidecar-consumer-sa
      containers:
        - name: event-subscriber
          image: event-subscriber-app
        - name: cloud-event-proxy-as-sidecar
          image: openshift4/ose-cloud-event-proxy
          args:
            - "--metrics-addr=127.0.0.1:9091"
            - "--store-path=/store"
            - "--transport-host=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
            - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043"
            - "--api-port=8089"
          env:
            - name: NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: NODE_IP
              valueFrom:
```

```
            fieldRef:
              fieldPath: status.hostIP
          volumeMounts:
            - name: pubsubstore
              mountPath: /store
        ports:
          - name: metrics-port
            containerPort: 9091
          - name: sub-port
            containerPort: 9043
        volumes:
          - name: pubsubstore
            emptyDir: {}
```

AMQ トランスポートを使用した cloud-event-proxy デプロイメントの参照

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cloud-event-proxy-sidecar
  namespace: cloud-events
  labels:
    app: cloud-event-proxy
spec:
  selector:
    matchLabels:
      app: cloud-event-proxy
  template:
    metadata:
      labels:
        app: cloud-event-proxy
    spec:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      containers:
        - name: cloud-event-sidecar
          image: openshift4/ose-cloud-event-proxy
          args:
            - "--metrics-addr=127.0.0.1:9091"
            - "--store-path=/store"
            - "--transport-host=amqp://router.router.svc.cluster.local"
            - "--api-port=8089"
          env:
            - name: <node_name>
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: <node_ip>
              valueFrom:
                fieldRef:
                  fieldPath: status.hostIP
          volumeMounts:
            - name: pubsubstore
              mountPath: /store
          ports:
            - name: metrics-port
```

```
        containerPort: 9091
      - name: sub-port
        containerPort: 9043
    volumes:
      - name: pubsubstore
        emptyDir: {}
```

cloud-event-proxy サブスクライバーサービスの参照

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  clusterIP: None
  sessionAffinity: None
  type: ClusterIP
```

21.4.3. c loud-event-proxy サイドカー REST API から利用可能な PTP イベント

PTP イベントコンシューマーアプリケーションは、以下の PTP タイミングイベントの PTP イベントプロデューサーをポーリングできます。

表21.15 cloud-event-proxy サイドカーから利用可能な PTP イベント

リソース URI	説明
/cluster/node/<node_name>/sync/ptp-status/lock-state	現在の PTP 機器のロック状態を説明します。**LOCKED**、**HOLDOVER**、または **FREERUN** の状態にできます。
/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state	ホストオペレーティングシステムのクロック同期状態について説明します。**LOCKED** または **FREERUN** 状態になります。
/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change	PTP クロッククラスの現在の状態を説明します。

21.4.4. コンシューマーアプリケーションを PTP イベントにサブスクライブする

PTP イベントコンシューマーアプリケーションがイベントをポーリングできるようにするには、アプリケーションをイベントプロデューサーにサブスクライブする必要があります。

21.4.4.1. PTP lock-state イベントのサブスクライブ

PTP **lock-state** イベントのサブスクリプションを作成するには、次のペイロードを使用して、**http://localhost:8081/api/ocloudNotifications/v1/subscriptions** のクラウドイベント API に **POST** アクションを送信します。

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/ptp-status/lock-state",
}
```

応答の例

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/ptp-status/lock-state",
}
```

21.4.4.2. PTP os-lock-sync-state イベントのサブスクライブ

PTP **os-clock-sync-state** イベントのサブスクリプションを作成するには、次のペイロードを使用して、**http://localhost:8081/api/ocloudNotifications/v1/subscriptions** のクラウドイベント API に **POST** アクションを送信します。

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state",
}
```

応答の例

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state",
}
```

21.4.4.3. PTP ptp-lock-class-change イベントのサブスクライブ

PTP **ptp-clock-class-change** イベントのサブスクリプションを作成するには、次のペイロードを使用して、**http://localhost:8081/api/ocloudNotifications/v1/subscriptions** のクラウドイベント API に **POST** アクションを送信します。

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change",
}
```

応答の例

```
{
"id": "e23473d9-ba18-4f78-946e-401a0caeff90",
"endpointUri": "http://localhost:8989/event",
"uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/e23473d9-ba18-4f78-946e-401a0caeff90",
"resource": "/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change",
}
```

21.4.5. 現在の PTP クロックステータスの取得

ノードの現在の PTP ステータスを取得するには、**GET** アクションを以下のイベント REST API のいずれかに送信します。

- **http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/lock-state/CurrentState**
- **http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state/CurrentState**
- **http://localhost:8081/api/ocloudNotifications/v1/cluster/node/<node_name>/sync/ptp-status/ptp-clock-class-change/CurrentState**

応答はクラウドネイティブイベント JSON オブジェクトです。以下に例を示します。

ロック状態 API レスポンスの例

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "v1",
    "values": [
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "notification",
        "valueType": "enumeration",
        "value": "LOCKED"
      },
      {
        "resource": "/cluster/node/compute-1.example.com/ens5fx/master",
        "dataType": "metric",
        "valueType": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

21.4.6. PTP イベントコンシューマーアプリケーションがイベントを受信していることの確認

アプリケーション Pod の **cloud-event-proxy** コンテナーが PTP イベントを受信していることを確認します。

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。
- PTP Operator をインストールして設定している。

手順

1. アクティブな **linuxptp-daemon** Pod の一覧を取得します。以下のコマンドを実行します。

```
$ 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. 次のコマンドを実行して、必要なコンシューマー側 **cloud-event-proxy** コンテナーのメトリックにアクセスします。

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

ここでは、以下のようになります。

<linuxptp-daemon>

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

出力例

```
# HELP cne_transport_connections_resets Metric to get number of connection resets
# TYPE cne_transport_connections_resets gauge
cne_transport_connection_reset 1
# HELP cne_transport_receiver Metric to get number of receiver created
# TYPE cne_transport_receiver gauge
cne_transport_receiver{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 2
cne_transport_receiver{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 2
# HELP cne_transport_sender Metric to get number of sender created
# TYPE cne_transport_sender gauge
cne_transport_sender{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 1
cne_transport_sender{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 1
```

```
# HELP cne_events_ack Metric to get number of events produced
# TYPE cne_events_ack gauge
cne_events_ack{status="success",type="/cluster/node/compute-1.example.com/ptp"} 18
cne_events_ack{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP cne_events_transport_published Metric to get number of events published by the
transport
# TYPE cne_events_transport_published gauge
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_transport_received Metric to get number of events received  by the
transport
# TYPE cne_events_transport_received gauge
cne_events_transport_received{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_received{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_api_published Metric to get number of events published by the rest
api
# TYPE cne_events_api_published gauge
cne_events_api_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 19
cne_events_api_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 19
# HELP cne_events_received Metric to get number of events received
# TYPE cne_events_received gauge
cne_events_received{status="success",type="/cluster/node/compute-1.example.com/ptp"}
18
cne_events_received{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being
served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status
code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 4
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

第22章 外部 DNS OPERATOR

22.1. 外部 DNS OPERATOR のリリースノート

外部 DNS Operator は、**ExternalDNS** をデプロイおよび管理し、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行います。

これらのリリースノートでは、OpenShift Container Platform での外部 DNS Operator の開発を追跡しています。

22.1.1. 外部 DNS Operator 1.2.0

外部 DNS Operator バージョン 1.2.0 では、以下のアドバイザリーを利用できます。

- RHEA-2022:5867 ExternalDNS Operator 1.2 operator/operand containers

22.1.1.1. 新機能

- 外部 DNS Operator が AWS 共有 VPC をサポートするようになりました。詳細は、共有 VPC を使用して別の AWS アカウントに DNS レコードを作成する を参照してください。

22.1.1.2. バグ修正

- オペランドの更新ストラテジーが、**Rolling** から **Recreate** に変更されました。(OCPBUGS-3630)

22.1.2. 外部 DNS Operator 1.1.1

外部 DNS Operator バージョン 1.1.1 では、以下のアドバイザリーを利用できます。利用できます。

- RHEA-2024:0536 ExternalDNS Operator 1.1 operator/operand containers

22.1.3. 外部 DNS Operator 1.1.0

このリリースには、アップストリームプロジェクトバージョン 0.13.1 からのオペランドのリベースが含まれていました。外部 DNS Operator バージョン 1.1.0 では、以下のアドバイザリーを利用できます。

- RHEA-2022:9086-01 ExternalDNS Operator 1.1 operator/operand containers

22.1.3.1. バグ修正

- 以前は、ExternalDNS Operator がボリュームに空の **defaultMode** 値を強制していたため、OpenShift API との競合により更新が随時行われていました。現在は、**defaultMode** 値は強制されず、オペランドのデプロイは随時更新されなくなりました。(OCPBUGS-2793)

22.1.4. 外部 DNS Operator 1.0.1

外部 DNS Operator バージョン 1.0.1 では、以下のアドバイザリーを利用できます。

- RHEA-2024:0537 ExternalDNS Operator 1.0 operator/operand containers

22.1.5. 外部 DNS Operator 1.0.0

外部 DNS Operator バージョン 1.0.0 では、以下のアドバイザリーを利用できます。

- RHEA-2022:5867 ExternalDNS Operator 1.0 operator/operand containers

22.1.5.1. バグ修正

- 以前は、External DNS Operator は、ExternalDNS オペランド Pod のデプロイメント中に制限付き SCC ポリシーの違反に関する警告を発していました。この問題は解決されています。(BZ#2086408)

22.2. OPENSHIFT CONTAINER PLATFORM の外部 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。

22.2.1. 外部 DNS Operator

外部 DNS Operator は、**olm.openshift.io** API グループから外部 DNS API を実装します。外部 DNS Operator は、サービス、ルート、外部 DNS プロバイダーを更新します。

前提条件

- **yq** CLI ツールがインストールされている。

手順

OperatorHub からオンデマンドで外部 DNS Operator をデプロイできます。外部 DNS Operator をデプロイすると、**Subscription** オブジェクトが作成されます。

1. 次のコマンドを実行して、インストールプランの名前を確認します。

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

出力例

```
install-zcvlr
```

2. 次のコマンドを実行して、インストールプランのステータスが **Complete** になっているか確認します。

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

出力例

```
Complete
```

3. 次のコマンドを実行して、**external-dns-operator** デプロイメントのステータスを表示します。

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

出力例

```
```

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

22.2.2. 外部 DNS Operator ログ

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

手順

1. 次のコマンドを実行して、外部 DNS Operator のログを表示します。

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

22.2.2.1. 外部 DNS Operator のドメイン名の制限

外部 DNS Operator は、TXT レコードの接頭辞を追加する TXT レジストリーを使用します。これにより、TXT レコードのドメイン名の最大長が短くなります。DNS レコードは対応する TXT レコードなしでは存在できないため、DNS レコードのドメイン名は TXT レコードと同じ制限に従う必要があります。たとえば、DNS レコードが **<domain_name_from_source>** の場合、TXT レコードは **external-dns-<record_type>-<domain_name_from_source>** になります。

外部 DNS Operator によって生成される DNS レコードのドメイン名には、次の制限があります。

レコードの種類	文字数
CNAME	44
AzureDNS のワイルドカード CNAME レコード	42
A	48
AzureDNS のワイルドカード A レコード	46

生成されたドメイン名がドメイン名の制限を超える場合、外部 DNS Operator のログに次のエラーが表示されます。

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

22.3. クラウドプロバイダーへの外部 DNS OPERATOR のインストール

AWS、Azure、GCP などのクラウドプロバイダーに外部 DNS Operator をインストールできます。

22.3.1. External DNS Operator のインストール

OpenShift Container Platform OperatorHub を使用して、外部 DNS Operator をインストールできます。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
2. **外部 DNS Operator**をクリックします。**Filter by keyword** のテキストボックスまたはフィルターリストを使用して、Operator のリストから外部 DNS Operator を検索できます。
3. **external-dns-operator** namespace を選択します。
4. **External DNS Operator** ページで **Install** をクリックします。
5. **Install Operator** ページで、次のオプションを選択していることを確認してください。
    a. チャネルを **stable-v1.0** として更新している。
    b. インストールモードに **A specific name on the cluster**を選択している。
    c. namespace を**external-dns-operator**としてインストールしている。namespace **external-dns-operator** が存在しない場合は、Operator のインストール中に作成されます。
    d. **承認ストラテジー** を **Automatic** または **Manual** として選択している。承認ストラテジーはデフォルトで **Automatic** に設定されます。
    e. **Install** をクリックします。

Automatic (自動) 更新を選択した場合、Operator Lifecycle Manager (OLM) は介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。

Manual 更新を選択した場合、OLM は更新要求を作成します。クラスター管理者は、Operator が新規バージョンに更新されるように更新要求を手動で承認する必要があります。

検証

外部 DNS Operator で、**Installed Operators** ダッシュボードの **Status** が **Succeeded** と表示されることを確認します。

22.4. 外部 DNS OPERATOR 設定パラメーター

外部 DNS Operator には、次の設定パラメーターが含まれています。

22.4.1. 外部 DNS Operator 設定パラメーター

外部 DNS Operator には、次の設定パラメーターが含まれています。

パラメーター	説明

パラメーター	説明
spec	クラウドプロバイダーのタイプを有効にします。 spec: provider: type: AWS (1) aws: credentials: name: aws-access-key (2) (1) AWS、GCP、Azure、Infoblox などの利用可能なオプションを定義します。 (2) クラウドプロバイダーのシークレット名を定義します。
zones	ドメインごとに DNS ゾーンを指定できます。ゾーンを指定しない場合、**ExternalDNS** リソースはクラウドプロバイダーアカウントに存在するすべてのゾーンを検出します。 zones: - "myzoneid" (1) (1) DNS ゾーンの名前を指定します。

パラメーター	説明
domains	ドメインごとに AWS ゾーンを指定できます。ドメインを指定しない場合、**ExternalDNS** リソースはクラウドプロバイダーアカウントに存在するすべてのゾーンを検出します。 domains: - filterType: Include (1) matchType: Exact (2) name: "myzonedomain1.com" (3) - filterType: Include matchType: Pattern (4) pattern: ".*\\.otherzonedomain\\.com" (5) (1) **ExternalDNS** リソースにドメイン名が含まれていることを確認します。 (2) ドメインは、正規表現での照合ではなく、完全に一致する必要があることを、**ExtrnalDNS** に指示します。 (3) ドメインの名前を定義します。 (4) **ExternalDNS** リソースに **regex-domain-filter** フラグを設定します。正規表現フィルターを使用して、使用できるドメインに限定します。 (5) ターゲットゾーンのドメインをフィルタリングするために **ExternalDNS** リソースが使用する正規表現パターンを定義します。
source	DNS レコードのソース (**サービス**または**ルート**) を指定できます。 source: (1) type: Service (2) service: serviceType: (3) - LoadBalancer - ClusterIP labelFilter: (4) matchLabels: external-dns.mydomain.org/publish: "yes" hostnameAnnotation: "Allow" (5) fqdnTemplate: - "{{.Name}}.myzonedomain.com" (6) (1) DNS レコードのソースの設定を定義します。 (2) **ExternalDNS** リソースは、DNS レコードを作成するためのソースとして **Service** タイプを使用します。 (3) **ExternalDNS** リソースに **service-type-filter** フラグを設定します。**service Type** には、次のフィールドが含まれています。 • **デフォルト**: **LoadBalancer** • **expected**: **ClusterIP**

パラメーター	説明
	• NodePort • LoadBalancer • ExternalName 4 コントローラーで考慮するのは、ラベルフィルターと一致するリソースのみにします。 5 **hostnameAnnotation**のデフォルト値は**Ignore**で、フィールド**fqdnTemplates**で指定されたテンプレートを使用して DNS レコードを生成するように**ExternalDNS**に指示します。値が**Allow**の場合には、**external-dns.alpha.kubernetes.io/hostname** アノテーションで指定された値をもとに DNS レコードが生成されます。 6 外部 DNS Operator は文字列を使用して、ホスト名を定義しないソースから DNS 名を生成するか、偽のソースとペアになっている場合はホスト名接尾辞を追加します。 source: type: OpenShiftRoute 1 openshiftRouteOptions: routerName: default 2 labelFilter: matchLabels: external-dns.mydomain.org/publish: "yes" 1 DNS レコードを作成します。 2 ソースタイプが **OpenShiftRoute** の場合は、Ingress Controller 名を指定できます。**ExternalDNS** リソースは、CNAME レコードのターゲットとして Ingress コントローラーの正規名を使用します。

22.5. AWS での DNS レコードの作成

外部 DNS Operator を使用して、AWS および AWS GovCloud で DNS レコードを作成できます。

22.5.1. Red Hat 外部 DNS Operator を使用した AWS のパブリックホストゾーンへの DNS レコードの作成

Red Hat 外部 DNS Operator を使用して、AWS のパブリックホストゾーンに DNS レコードを作成できます。同じ手順を使用して、AWS GovCloud のホストゾーンに DNS レコードを作成できます。

手順

1. ユーザーを確認してください。ユーザーは、 **kube-system**namespace にアクセスできる必要があります。クレデンシャルがない場合は、 **kube-system**namespace からクレデンシャルを取得すると、クラウドプロバイダークライアントを使用できます。

```
$ oc whoami
```

出力例

```
system:admin
```

2. **kube-system**namespace に存在する aws-creds シークレットから値を取得します。

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

3. ルートを取得して、ドメインを確認します。

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

出力例

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

4. DNS ゾーンのリストを取得して、以前に検出されたルートのドメインに対応するものを検索します。

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

出力例

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

5. **route** ソースの **ExternalDNS** リソースを作成します。

```
$ cat <<EOF | oc create -f -
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-aws 1
spec:
  domains:
  - filterType: Include 2
    matchType: Exact 3
    name: testextdnsoperator.apacshift.support 4
  provider:
    type: AWS 5
  source: 6
    type: OpenShiftRoute 7
    openshiftRouteOptions:
      routerName: default 8
EOF
```

1. 外部 DNS リソースの名前を定義します。
2. デフォルトでは、すべてのホストゾーンがターゲット候補として選択されます。必要なホストゾーンを追加できます。
3. ターゲットゾーンのドメインは、(正規表現の一致とは対照的に) 完全一致である必要があります。
4. 更新するゾーンのドメインを正確に指定します。ルートのホスト名は、指定されたドメインのサブドメインである必要があります。
5. **AWS Route53DNS**プロバイダーを定義します。
6. DNS レコードのソースのオプションを定義します。
7. 以前に指定された DNS プロバイダーで作成される DNS レコードのソースとして OpenShift **route** リソースを定義します。
8. ソースが **OpenShiftRoute** の場合に、OpenShift Ingress Controller 名を指定できます。外部 DNS Operator は、CNAME レコードの作成時に、そのルーターの正規のホスト名をターゲットとして選択します。

6. 次のコマンドを使用して、OCP ルート用に作成されたレコードを確認します。

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

22.5.2. 共有 VPC を使用して別の AWS アカウントに DNS レコードを作成する

ExternalDNS Operator を使用すると、共有 Virtual Private Cloud (VPC) を使用して別の AWS アカウントに DNS レコードを作成できます。共有 VPC を使用すると、組織は複数のプロジェクトのリソースを共通の VPC ネットワークに接続できます。その後、VPC 共有を使用して、複数の AWS アカウント間で単一の Route 53 インスタンスを使用できます。

前提条件

- 2 つの Amazon AWS アカウントを作成している。1 つは VPC と Route 53 プライベートホストゾーンが設定されたもの (アカウント A)、もう 1 つはクラスターをインストールするためのもの (アカウント B) です。
- アカウント B でアカウント A の Route 53 ホストゾーンに DNS レコードを作成するために、適切な権限を持つ IAM ポリシーと IAM ロールをアカウント A に作成している。
- アカウント B のクラスターをアカウント A の既存の VPC にインストールしている。
- アカウント B のクラスターに ExternalDNS Operator をインストールしている。

手順

1. 次のコマンドを実行して、アカウント B からアカウント A の Route 53 ホストゾーンにアクセスできるように作成した IAM ロールのロール ARN を取得します。

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

出力例

```
ROLE arn:aws:iam::1234567890123:role/user-rol1 2023-09-14T17:21:54+00:00 3600 /
AROA3SGB2ZRKRT5NISNJN user-rol1
```

2. 次のコマンドを実行して、アカウント A の認証情報で使用するプライベートホストゾーンを特定します。

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

出力例

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

3. 次のコマンドを実行して、**ExternalDNS** オブジェクトを作成します。

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

1 アカウント A に DNS レコードを作成するには、ロール ARN を指定します。

4. 次のコマンドを使用して、OpenShift Container Platform (OCP) ルートに対して作成されたレコードを確認します。

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

22.6. AZURE での DNS レコードの作成

外部 DNS Operator を使用して、Azure 上に DNS レコードを作成できます。

22.6.1. Azure のパブリック DNS ゾーン上で DNS レコードを作成する

Red Hat 外部 DNS Operator を使用して、Azure のパブリック DNS ゾーンに DNS レコードを作成できます。

前提条件

- 管理者権限を持っている。
- **admin** ユーザーの場合、**kube-system** namespace にアクセスできる。

手順

1. クラウドプロバイダークライアントを使用するために、次のコマンドを実行して **kube-system** namespace から認証情報を取得します。

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

2. 次のコマンドを実行して、Azure にログインします。

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

3. 次のコマンドを実行して、ルートのリストを取得します。

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

出力例

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

4. 次のコマンドを実行して、DNS ゾーンのリストを取得します。

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

5. **ExternalDNS** オブジェクトを定義する YAML ファイル (例: **external-dns-sample-azure.yaml**) を作成します。

external-dns-sample-azure.yaml ファイルの例

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

1. 外部 DNS 名を指定します。
2. ゾーン ID を定義します。
3. プロバイダータイプを定義します。
4. DNS レコードのソースのオプションを定義できます。
5. ソースタイプが **OpenShiftRoute** の場合、OpenShift Ingress Controller 名を渡すことができます。外部 DNS は、CNAME レコードの作成時に、そのルーターの正規のホスト名をターゲットとして選択します。
6. **route** リソースを Azure DNS レコードのソースとして定義します。

6. 次のコマンドを実行して、OpenShift Container Platform ルートに対して作成された DNS レコードを確認します。

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

注記

プライベート Azure DNS のホストされたプライベートゾーンにレコードを作成するには、**zones** フィールドの下にプライベートゾーンを指定する必要があります。これにより、プロバイダータイプが **ExternalDNS** 引数の **azure-private-dns** に入力されます。

22.7. GCP での DNS レコードの作成

外部 DNS Operator を使用して、GCP 上に DNS レコードを作成できます。

22.7.1. GCP のパブリックマネージドゾーン上で DNS レコードを作成する

外部 DNS Operator を使用して、GCP のパブリックマネージドゾーンに DNS レコードを作成できます。

前提条件

- 管理者権限を持っている。

手順

1. 次のコマンドを実行して、**encoded-gcloud.json** ファイル内の **gcp-credentials** シークレットをコピーします。

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

2. 次のコマンドを実行して、Google の認証情報をエクスポートします。

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

3. 次のコマンドを使用して、アカウントをアクティブ化します。

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

4. 次のコマンドを実行して、プロジェクトを設定します。

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

5. 次のコマンドを実行して、ルートのリストを取得します。

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

出力例

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

6. 次のコマンドを実行して、マネージドゾーンのリストを取得します。

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

出力例

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

7. **ExternalDNS** オブジェクトを定義する YAML ファイル (例: **external-dns-sample-gcp.yaml**) を作成します。

external-dns-sample-gcp.yaml ファイルの例

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
```

```
  name: sample-gcp (1)
spec:
  domains:
    - filterType: Include (2)
      matchType: Exact (3)
      name: test.gcp.example.com (4)
  provider:
    type: GCP (5)
  source:
    openshiftRouteOptions: (6)
      routerName: default (7)
    type: OpenShiftRoute (8)
```

1. 外部 DNS 名を指定します。
2. デフォルトでは、すべてのホストされたゾーンがターゲット候補として選択されます。ホストされたゾーンを含めることができます。
3. ターゲットのドメインは、**name** キーで定義された文字列と一致する必要があります。
4. 更新するゾーンのドメインを正確に指定します。ルートのホスト名は、指定されたドメインのサブドメインである必要があります。
5. プロバイダータイプを定義します。
6. DNS レコードのソースのオプションを定義できます。
7. ソースタイプが **OpenShiftRoute** の場合、OpenShift Ingress Controller 名を渡すことができます。外部 DNS は、CNAME レコードの作成時に、そのルーターの正規のホスト名をターゲットとして選択します。
8. **route** リソースを GCP DNS レコードのソースとして定義します。

8. 次のコマンドを実行して、OpenShift Container Platform ルートに対して作成された DNS レコードを確認します。

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

22.8. INFOBLOX での DNS レコードの作成

外部 DNS Operator を使用して、Infoblox に DNS レコードを作成できます。

22.8.1. Infoblox のパブリック DNS ゾーンでの DNS レコードの作成

外部 DNS Operator を使用して、Infoblox のパブリック DNS ゾーンに DNS レコードを作成できます。

前提条件

- OpenShift CLI (**oc**) にアクセスできる。
- Infoblox UI にアクセスできる。

手順

1. 次のコマンドを実行して、Infoblox クレデンシャルを使用して **secret** オブジェクトを作成します。

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

2. 次のコマンドを実行して、ルートのリストを取得します。

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

出力例

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

3. **ExternalDNS** オブジェクトを定義する YAML ファイル (例: **external-dns-sample-infoblox.yaml**) を作成します。

external-dns-sample-infoblox.yaml ファイルの例

```
apiVersion: externaldns.olm.openshift.io/v1beta1
kind: ExternalDNS
metadata:
  name: sample-infoblox (1)
spec:
  provider:
    type: Infoblox (2)
    infoblox:
      credentials:
        name: infoblox-credentials
      gridHost: ${INFOBLOX_GRID_PUBLIC_IP}
      wapiPort: 443
      wapiVersion: "2.3.1"
  domains:
  - filterType: Include
    matchType: Exact
    name: test.example.com
  source:
    type: OpenShiftRoute (3)
    openshiftRouteOptions:
      routerName: default (4)
```

(1) 外部 DNS 名を指定します。

(2) プロバイダータイプを定義します。

(3) DNS レコードのソースのオプションを定義できます。

4 ソースタイプが **OpenShiftRoute** の場合、OpenShift Ingress Controller 名を渡すことができます。外部 DNS は、CNAME レコードの作成時に、そのルーターの正規のホスト名をターゲットとして選択します。

4. 次のコマンドを実行して、Infoblox に **ExternalDNS** リソースを作成します。

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

5. Infoblox UI から、**console** ルート用に作成された DNS レコードを確認します。
   a. **Data Management → DNS → Zones** をクリックします。
   b. ゾーン名を選択します。

22.9. 外部 DNS OPERATOR でのクラスター全体のプロキシーの設定

クラスター全体のプロキシーを設定した後、Operator Lifecycle Manager (OLM) はデプロイされたすべての Operator に対して、**HTTP_PROXY**、**HTTPS_PROXY**、および **NO_PROXY** 環境変数の新しい内容の自動更新をトリガーします。

22.9.1. クラスター全体のプロキシーの認証局を信頼する

外部 DNS Operator を設定して、クラスター全体のプロキシーの認証局を信頼できます。

手順

1. 次のコマンドを実行して、**external-dns-operator** namespace に CA バンドルを含める config map を作成します。

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

2. 信頼できる CA バンドルを config map に挿入するには、次のコマンドを実行して、**config.openshift.io/inject-trusted-cabundle=true** ラベルを config map に追加します。

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

3. 次のコマンドを実行して、外部 DNS Operator のサブスクリプションを更新します。

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

検証

- 外部 DNS Operator のデプロイ後、次のコマンドを実行して、信頼できる CA 環境変数が **external-dns-operator** デプロイメントに追加されていることを確認します。

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

出力例

trusted-ca

第23章 ネットワークポリシー

23.1. 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。

23.1.1. 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ネットワークプラグインを使用するクラスターでは、ネットワーク分離は **NetworkPolicy** オブジェクトによって完全に制御されます。OpenShift Container Platform 4.15 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。

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。ただし、ホストネットワークの Pod に接続する Pod はネットワークポリシールールの影響を受ける可能性があります。

ネットワークポリシーは、ローカルホストまたは常駐ノードからのトラフィックをブロックすることはできません。

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。

ネットワークポリシーは、TCP、UDP、ICMP、および SCTP プロトコルにのみ適用されます。他のプロトコルは影響を受けません。

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 Controller からの接続のみを許可します。
プロジェクトで OpenShift Container Platform Ingress Controller 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て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て複雑なネットワーク要件を満すことができます。

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。

23.1.1.1. allow-from-router ネットワークポリシーの使用

次の **NetworkPolicy** を使用して、ルーターの設定に関係なく外部トラフィックを許可します。

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

(1) **policy-group.network.openshift.io/ingress:""** ラベルは、OpenShift-SDN と OVN-Kubernetes の両方をサポートします。

23.1.1.2. allow-from-hostnetwork ネットワークポリシーの使用

次の **allow-from-hostnetwork NetworkPolicy** オブジェクトを追加して、ホストネットワーク Pod からのトラフィックを転送します。

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

23.1.2. OpenShift SDN を使用したネットワークポリシー最適化

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。

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のある 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、分離する必要のある 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 から許可する必要のある特定のトラフィックを可能にします。

23.1.3. OVN-Kubernetes ネットワークプラグインによる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。

- 同じ **spec.podSelector** 仕様を持つネットワークポリシーの場合、**ingress** ルールまたは **egress** ルールを持つ複数のネットワークポリシーを使用するよりも、複数の **Ingress** ルールまたは **egress** ルールを持つ 1 つのネットワークポリシーを使用する方が効率的です。

- **podSelector** または **namespaceSelector** 仕様に基づくすべての **Ingress** または **egress** ルールは、**number of pods selected by network policy + number of pods selected by ingress or egress rule** に比例する数の OVS フローを生成します。そのため、Pod ごとに個別のルールを作成するのではなく、1 つのルールで必要な数の Pod を選択できる **podSelector** または **namespaceSelector** 仕様を使用することが推奨されます。
たとえば、以下のポリシーには 2 つのルールが含まれています。

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

以下のポリシーは、上記と同じ 2 つのルールを 1 つのルールとして表現しています。

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

同じガイドラインが **spec.podSelector** 仕様に適用されます。異なるネットワークポリシーに同じ **ingress** ルールまたは **egress** ルールがある場合、共通の **spec.podSelector** 仕様で 1 つのネットワークポリシーを作成する方が効率的な場合があります。たとえば、以下の 2 つのポリシーには異なるルールがあります。

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

```
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: policy2
spec:
  podSelector:
    matchLabels:
      role: client
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

以下のネットワークポリシーは、上記と同じ 2 つのルールを1 つのルールとして表現しています。

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

この最適化は、複数のセレクターを 1 つのセレクターとして表現する場合に限り適用できます。セレクターが異なるラベルに基づいている場合、この最適化は適用できない可能性があります。その場合は、ネットワークポリシーの最適化に特化して新規ラベルをいくつか適用することを検討してください。

23.1.4. 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

23.1.5. 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

23.2. 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。

23.2.1. 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 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

1. NetworkPolicy オブジェクトの名前。
2. 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。
3. 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 namaspace にある Pod を照合して検索します。
4. 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。

23.2.2. CLI を使用した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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。

   これは基本的なポリシーであり、他のネットワークポリシーの設定によって許可されたクロス Pod トラフィック以外のすべてのクロス Pod ネットワーキングをブロックします。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: deny-by-default
   spec:
     podSelector: {}
     policyTypes:
     - Ingress
     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: {}
   ```

   特定のnamespaceから 1つの Pod への上りトラフィックを許可する

   このポリシーは、**namespace-y** で実行されている Pod から **pod-a** というラベルの付いた Pod へのトラフィックを許可します。

   ```
   kind: NetworkPolicy
   apiVersion: networking.k8s.io/v1
   metadata:
     name: allow-traffic-pod
   spec:
     podSelector:
       matchLabels:
   ```

```
    pod: pod-a
policyTypes:
- Ingress
ingress:
- from:
  - namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: namespace-y
```

2. 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>
```

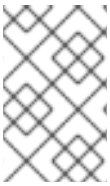
ここでは、以下のようになります。

<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/deny-by-default created
```



注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールのフォームから、クラスターの任意の namespace でネットワークポリシーを直接作成できます。

23.2.3. デフォルトの全拒否ネットワークポリシーの作成

これは基本的なポリシーであり、他のデプロイメントされたネットワークポリシーの設定によって許可されたネットワークトラフィック以外のすべてのクロス Pod ネットワークをブロックします。この手順では、デフォルトの **deny-by-default** ポリシーを適用します。

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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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. すべての namespace におけるすべての Pod からの ingress を拒否する **deny-by-default** ポリシーを定義する次の YAML を作成します。YAML を **deny-by-default.yaml** ファイルに保存します。

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

(1) **namespace: default** は、このポリシーを **default** namespace にデプロイします。

(2) **podSelector:** は空です。これは、すべての Pod に一致することを意味します。したがって、ポリシーはデフォルトnamespaceのすべての Pod に適用されます。

(3) 指定された **ingress** ルールはありません。これにより、着信トラフィックがすべての Pod にドロップされます。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

23.2.4. 外部クライアントからのトラフィックを許可するネットワークポリシーの作成

deny-by-default ポリシーを設定すると、外部クライアントからラベル **app=web** を持つ Pod へのトラフィックを許可するポリシーの設定に進むことができます。

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。

この手順に従って、パブリックインターネットから直接、またはロードバランサーを使用して Pod にアクセスすることにより、外部サービスを許可するポリシーを設定します。トラフィックは、ラベル **app=web** を持つ Pod にのみ許可されます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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. パブリックインターネットからのトラフィックが直接、またはロードバランサーを使用して Pod にアクセスできるようにするポリシーを作成します。YAML を **web-allow-external.yaml** ファイルに保存します。

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

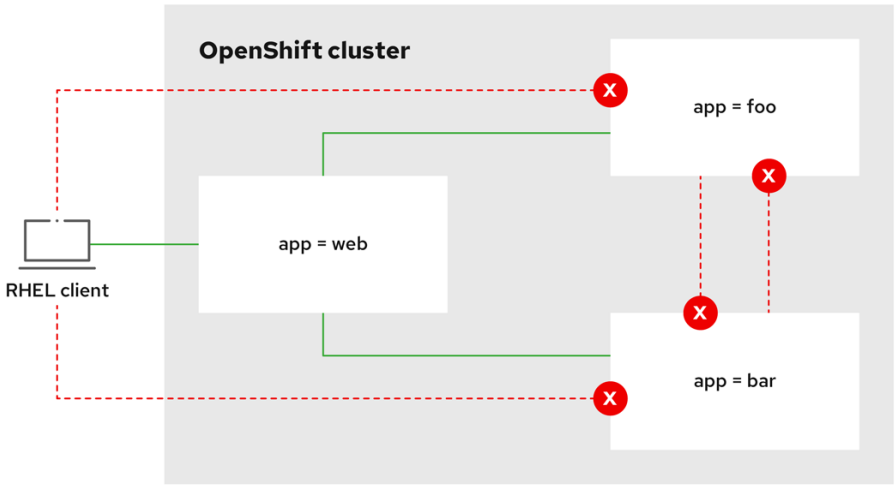
2. 次のコマンドを入力して、ポリシーを適用します。

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

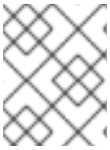
出力例

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

このポリシーは、次の図に示すように、外部トラフィックを含むすべてのリソースからのトラフィックを許可します。



23.2.5. すべてのnamespaceからアプリケーションへのトラフィックを許可するネットワークポリシーを作成する



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。

この手順に従って、すべてのnamespace内のすべての Pod から特定のアプリケーションへのトラフィックを許可するポリシーを設定します。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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. すべてのnamespaceのすべての Pod から特定のアプリケーションへのトラフィックを許可するポリシーを作成します。YAML を **web-allow-all-namespaces.yaml** ファイルに保存します。

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

(1) デフォルトの namespace の **app:web** Pod にのみポリシーを適用します。

(2) すべてのnamespaceのすべての Pod を選択します。

注記

デフォルトでは、**namespaceSelector** の指定を省略した場合、namespace は選択されません。つまり、ポリシーは、ネットワークポリシーがデプロイされている namespace からのトラフィックのみを許可します。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

検証

1. 次のコマンドを入力して、**default** namespace で Web サービスを開始します。

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

2. 次のコマンドを実行して、**alpine** イメージを **secondary** namespace にデプロイし、シェルを開始します。

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

3. シェルで次のコマンドを実行し、リクエストが許可されていることを確認します。

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

予想される出力

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

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

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

23.2.6. namespaceからアプリケーションへのトラフィックを許可するネットワークポリシーの作成

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。

特定の namespace からラベル **app=web** を持つ Pod へのトラフィックを許可するポリシーを設定するには、次の手順に従います。以下の場合にこれを行うことができます。

- 運用データベースへのトラフィックを、運用ワークロードがデプロイされているnamespaceのみに制限します。
- 特定のnamespaceにデプロイされた監視ツールを有効にして、現在のnamespaceからメトリックをスクレイピングします。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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. ラベルが **purpose=production** の特定の namespace 内にあるすべての Pod からのトラフィックを許可するポリシーを作成します。YAML を **web-allow-prod.yaml** ファイルに保存します。

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

(1) デフォルトの namespace の **app:web** Pod にのみポリシーを適用します。

(2) ラベルが **purpose=production** の namespace 内にある Pod のみにトラフィックを制限します。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

検証

1. 次のコマンドを入力して、**default** namespace で Web サービスを開始します。

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

2. 次のコマンドを実行して、**prod** namespace を作成します。

```
$ oc create namespace prod
```

3. 次のコマンドを実行して、**prod** namespace にラベルを付けます。

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

4. 次のコマンドを実行して、**dev** namespace を作成します。

```
$ oc create namespace dev
```

5. 次のコマンドを実行して、**dev** namespace にラベルを付けます。

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

6. 次のコマンドを実行して、**alpine** イメージを **dev** namespace にデプロイし、シェルを開始します。

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

7. シェルで次のコマンドを実行し、リクエストがブロックされていることを確認します。

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

予想される出力

```
wget: download timed out
```

8. 次のコマンドを実行して、**alpine** イメージを **prod** namespace にデプロイし、シェルを開始します。

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

9. シェルで次のコマンドを実行し、リクエストが許可されていることを確認します。

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

予想される出力

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

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

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

23.2.7. 関連情報

- Web コンソールへのアクセス
- Egress ファイアウォールとネットワークポリシールールのロギング

23.3. 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。

23.3.1. 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 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
```

```
    matchLabels:
      app: app
  ports: 4
  - protocol: TCP
    port: 27017
```

1. NetworkPolicy オブジェクトの名前。
2. 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。
3. 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 namaspace にある Pod を照合して検索します。
4. 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。

23.3.2. CLI を使用した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
```

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールのフォームから、クラスターの任意の namespace でネットワークポリシーを直接表示できます。

23.4. 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。

23.4.1. 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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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リスト表示するには、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。

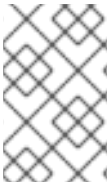
3. ネットワークポリシーオブジェクトが更新されていることを確認します。

```
$ 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。



注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールの **Actions** メニューのポリシーから、クラスターの任意の namespace でネットワークポリシーを直接編集できます。

23.4.2. 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 (1)
spec:
  podSelector: (2)
    matchLabels:
      app: mongodb
  ingress:
  - from:
    - podSelector: (3)
        matchLabels:
          app: app
    ports: (4)
    - protocol: TCP
      port: 27017
```

1. NetworkPolicy オブジェクトの名前。
2. 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。
3. 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 namaspace にある Pod を照合して検索します。
4. 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。

23.4.3. 関連情報

- ネットワークポリシーの作成

23.5. 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。

23.5.1. CLI を使用した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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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
```

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールの **Actions** メニューのポリシーから、クラスターの任意の namespace でネットワークポリシーを直接削除できます。

23.6. 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。

23.6.1. 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。

前提条件

- **cluster-admin** 権限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。

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、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。

23.6.2. 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、**mode: NetworkPolicy** が設定された OpenShift SDN ネットワークプラグインなど、**NetworkPolicy** オブジェクトをサポートするデフォルトの CNI ネットワークプラグインを使用します。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
- apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
```

```
  metadata:
    name: allow-from-kube-apiserver-operator
  spec:
    ingress:
    - from:
      - namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-kube-apiserver-operator
        podSelector:
          matchLabels:
            app: kube-apiserver-operator
    policyTypes:
    - Ingress
...
```

3. 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
   ```

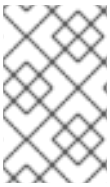
   1 **<project>** を、作成しているプロジェクトの名前に置き換えます。

   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
   ```

23.7. 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 ネットワークプラグイン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す。

23.7.1. 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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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
```

d. **allow-from-kube-apiserver-operator** 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-kube-apiserver-operator
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-kube-apiserver-operator
      podSelector:
        matchLabels:
          app: kube-apiserver-operator
  policyTypes:
  - Ingress
EOF
```

詳細は、新規の New **kube-apiserver-operator** webhook controller validating health of webhook を参照してください。

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
```

23.7.2. 次のステップ

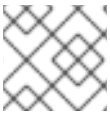
- デフォルトのネットワークポリシーの定義

23.7.3. 関連情報

- OpenShift SDN ネットワーク分離モード

第24章 CIDR 範囲の定義

次の CIDR 範囲には、重複しない範囲を指定する必要があります。



注記

クラスターの作成後にマシンの CIDR 範囲を変更することはできません。

重要

OpenShift Container Platform 4.11 以降のデフォルトのネットワークプロバイダーである OVN-Kubernetes は、IP アドレス範囲 **100.64.0.0/16** を内部的に使用します。クラスターで OVN-Kubernetes を使用している場合は、クラスター内の他の CIDR 定義に IP アドレス範囲 **100.64.0.0/16** を含めないでください。

24.1. MACHINE CIDR

Machine CIDR フィールドで、マシンまたはクラスターノードの IP アドレス範囲を指定する必要があります。

デフォルトは **10.0.0.0/16** です。この範囲は、接続されているネットワークと競合しないようにする必要があります。

24.2. SERVICE CIDR

Service CIDR フィールドで、サービスの IP アドレス範囲を指定する必要があります。範囲は、ワークロードに対応するのに十分な大きさである必要があります。アドレスブロックは、クラスター内からアクセスする外部サービスと重複してはいけません。デフォルトは **172.30.0.0/16** です。

24.3. POD CIDR

Pod CIDR フィールドで、Pod の IP アドレス範囲を指定する必要があります。

Pod CIDR は、**clusterNetwork** CIDR およびクラスター CIDR と同じです。範囲は、ワークロードに対応するのに十分な大きさである必要があります。アドレスブロックは、クラスター内からアクセスする外部サービスと重複してはいけません。デフォルトは **10.128.0.0/14** です。クラスターをインストールした後に、範囲を拡張できます。

関連情報

- Cluster Network Operator の設定
- クラスターネットワーク範囲の設定

24.4. ホスト接頭辞

Host Prefix フィールドで、個々のマシンにスケジュールされた Pod に割り当てられたサブネット接頭辞の長さを指定する必要があります。ホスト接頭辞は、各マシンの Pod IP アドレスプールを決定します。

例えば、ホスト接頭辞を **/23** に設定した場合、各マシンには Pod CIDR アドレス範囲から **/23** のサブネットが割り当てられます。デフォルトは **/23** で、510 台のクラスターノードと、ノードごとに 510 個の Pod IP アドレスを許可します。

第25章 AWS LOAD BALANCER OPERATOR

25.1. AWS LOAD BALANCER OPERATOR リリースノート

AWS Load Balancer (ALB) Operator は、**AWSLoadBalancerController** リソースのインスタンスをデプロイおよび管理します。

これらのリリースノートは、OpenShift Container Platform での AWS Load Balancer Operator の開発を追跡します。

AWS Load Balancer Operator の概要は、OpenShift Container Platform の AWS Load Balancer Operator を参照してください。

注記

AWS Load Balancer Operator は現在、AWS GovCloud をサポートしていません。

25.1.1. AWS Load Balancer Operator 1.1.1

AWS Load Balancer Operator バージョン 1.1.1 では、以下のアドバイザリーを利用できます。

- RHEA-2024:0555 Release of AWS Load Balancer Operator 1.1.z on OperatorHub

25.1.2. AWS Load Balancer Operator 1.1.0

AWS Load Balancer Operator バージョン 1.1.0 は、AWS Load Balancer Controller バージョン 2.4.4 をサポートします。

AWS Load Balancer Operator バージョン 1.1.0 では、以下のアドバイザリーを利用できます。

- RHEA-2023:6218 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

25.1.2.1. 大きな変更

- このリリースでは、Kubernetes API バージョン 0.27.2 を使用します。

25.1.2.2. 新機能

- AWS Load Balancer Operator は、Cloud Credential Operator を使用して標準化された Security Token Service (STS) フローをサポートするようになりました。

25.1.2.3. バグ修正

- FIPS 準拠のクラスターでは、TLS バージョン 1.2 を使用する必要があります。以前は、AWS Load Balancer Controller の Webhook は最小バージョンとして TLS 1.3 のみを受け入れていたため、FIPS 準拠のクラスターで次のようなエラーが発生しました。

  ```
  remote error: tls: protocol version not supported
  ```

  現在、AWS Load Balancer Controller は TLS 1.2 を最小 TLS バージョンとして受け入れ、この問題は解決されています。(OCPBUGS-14846)

25.1.3. AWS Load Balancer Operator 1.0.1

AWS Load Balancer Operator バージョン 1.0.1 では、以下のアドバイザリーを利用できます。

- Release of AWS Load Balancer Operator 1.0.1 on OperatorHub

25.1.4. AWS Load Balancer Operator 1.0.0

このリリースで、AWS Load Balancer Operator の一般提供が開始されました。AWS Load Balancer Operator バージョン 1.0.0 は、AWS Load Balancer Controller バージョン 2.4.4 をサポートします。

AWS Load Balancer Operator バージョン 1.0.0 では、以下のアドバイザリーを利用できます。

- RHEA-2023:1954 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

重要

AWS Load Balancer (ALB) Operator バージョン 1.x.x は、テクノロジープレビューバージョン 0.x.x から自動的にアップグレードできません。以前のバージョンからアップグレードするには、ALB オペランドをアンインストールし、**aws-load-balancer-operator** namespace を削除する必要があります。

25.1.4.1. 大きな変更

- このリリースでは、新しい **v1** API バージョンを使用しています。

25.1.4.2. バグ修正

- 以前のバージョンでは、AWS Load Balancer Operator によってプロビジョニングされたコントローラーは、クラスター全体のプロキシー設定を適切に使用しませんでした。これらの設定は、コントローラーに正しく適用されるようになりました。(OCPBUGS-4052、OCPBUGS-5295)

25.1.5. 以前のバージョン

AWS Load Balancer Operator の最初の 2 つのバージョンは、テクノロジープレビュー機能として利用できます。これらのバージョンは、実稼働クラスターで使用しないでください。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。

AWS Load Balancer Operator バージョン 0.2.0 では、以下のアドバイザリーを利用できます。

- RHEA-2022:9084 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

AWS Load Balancer Operator バージョン 0.0.1 では、以下のアドバイザリーを利用できます。

- RHEA-2022:5780 Release of AWS Load Balancer Operator on OperatorHub Enhancement Advisory Update

25.2. OPENSHIFT CONTAINER PLATFORM の AWS LOAD BALANCER OPERATOR

AWS Load Balancer Operator は、AWS Load Balancer Controller をデプロイおよび管理します。OpenShift Container Platform Web コンソールまたは CLI を使用して、OperatorHub から AWS Load Balancer Operator をインストールできます。

25.2.1. AWS Load Balancer Operator に関する考慮事項

AWS Load Balancer Operator をインストールして使用する前に、次の制限事項を確認してください。

- IP トラフィックモードは、AWS Elastic Kubernetes Service (EKS) でのみ機能します。AWS Load Balancer Operator は、AWS Load Balancer Controller の IP トラフィックモードを無効にします。IP トラフィックモードを無効にすると、AWS Load Balancer Controller は Pod readiness ゲートを使用できません。
- AWS Load Balancer Operator は **--disable-ingress-class-annotation** や **--disable-ingress-group-name-annotation** などのコマンドラインフラグを AWS Load Balancer Controller に追加します。したがって、AWS Load Balancer Operator では、**Ingress** リソースで **kubernetes.io/ingress.class** および **alb.ingress.kubernetes.io/group.name** アノテーションを使用できません。

25.2.2. AWS Load Balancer Operator

AWS Load Balancer Operator は **kubernetes.io/role/elb** タグがない場合に、パブリックサブネットにタグを付けることができます。また、AWS Load Balancer Operator は、基盤となる AWS クラウドから次の情報を検出します。

- Operator をホストするクラスターがデプロイされる仮想プライベートクラウド (VPC) の ID。
- 検出された VPC のパブリックおよびプライベートサブネット。

AWS Load Balancer Operator は、**インスタンス** ターゲットタイプのみで Network Load Balancer (NLB) を使用することにより、タイプ **LoadBalancer** の Kubernetes サービスリソースをサポートします。

手順

1. 次のコマンドを実行して **Subscription** オブジェクトを作成することにより、OperatorHub からオンデマンドで AWS Load Balancer Operator をデプロイできます。

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

出力例

```
install-zlfbt
```

2. 次のコマンドを実行して、インストールプランのステータスが **Complete** になっているか確認します。

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

出力例

```
Complete
```

3. 次のコマンドを実行して、**aws-load-balancer-operator-controller-manager** デプロイメントのステータスを表示します。

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

出力例

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

25.2.3. Outpost に拡張された AWS VPC クラスターでの AWS Load Balancer Operator の使用

Outpost に拡張された AWS VPC クラスター内で AWS Application Load Balancer をプロビジョニングするように AWS Load Balancer Operator を設定できます。AWS Outposts は AWS Network Load Balancer をサポートしていません。そのため、AWS Load Balancer Operator は Outpost に Network Load Balancer をプロビジョニングできません。

AWS Application Load Balancer は、クラウドサブネットか Outpost サブネットのどちらかに作成できます。クラウドの Application Load Balancer はクラウドベースのコンピュートノードに接続でき、Outpost の Application Load Balancer はエッジコンピュートノードに接続できます。Ingress リソースには Outpost サブネットまたは VPC サブネットのアノテーションを付ける必要がありますが、両方を付けることはできません。

前提条件

- AWS VPC クラスターを Outpost に拡張した。
- OpenShift CLI (**oc**) がインストールされている。
- AWS Load Balancer Operator をインストールし、AWS Load Balancer Controller を作成した。

手順

- 指定のサブネットを使用するように **Ingress** リソースを設定します。

  Ingress リソース設定の例

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

```
      service:
        name: <application_name>
        port:
          number: 80
```

1. 使用するサブネットを指定します。
   - Outpost で Application Load Balancer を使用するには、Outpost のサブネット ID を指定します。
   - クラウドで Application Load Balancer を使用するには、別々のアベイラビリティーゾーンに少なくとも 2 つのサブネットを指定する必要があります。

25.2.4. AWS Load Balancer Operator ログ

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

手順

- 次のコマンドを実行して、AWS Load Balancer Operator のログを表示します。

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

25.3. AWS LOAD BALANCER OPERATOR のインストール

AWS Load Balancer Operator は、AWS Load Balancer Controller をデプロイおよび管理します。OpenShift Container Platform Web コンソールまたは CLI を使用して、OperatorHub から AWS Load Balancer Operator をインストールできます。

25.3.1. Web コンソールを使用した AWS Load Balancer Operator のインストール

Web コンソールを使用して、AWS Load Balancer Operator をインストールできます。

前提条件

- **cluster-admin** パーミッションのあるユーザーとして OpenShift Container Platform Web コンソールにログインしていること。
- クラスターのプラットフォームタイプとクラウドプロバイダーが AWS に設定されている。
- セキュリティートークンサービス (STS) または user-provisioned infrastructure を使用している場合は、関連する準備手順に従います。たとえば、AWS Security Token Service を使用している場合は、"AWS Security Token Service (STS) を使用してクラスター上で AWS Load Balancer Operator を準備する" を参照してください。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** に移動します。

2. **AWS Load Balancer Operator** を選択します。**Filter by keyword** テキストボックスを使用するか、フィルターリストを使用して、Operator のリストから AWS Load Balancer Operator を検索できます。

3. **aws-load-balancer-operator** namespace を選択します。

4. **Install Operator** ページで、次のオプションを選択します。

   a. **Update the channel** で **stable-v1** を選択します。

   b. **Installation mode** で **All namespaces on the cluster (default)**を選択します。

   c. **Installed Namespace** で **aws-load-balancer-operator** を選択します。**aws-load-balancer-operator** namespace が存在しない場合は、Operator のインストール中に作成されます。

   d. **Update approval** で **Automatic** または **Manual** を選択します。デフォルトでは、**Update approval** は **Automatic** に設定されています。Automatic (自動) 更新を選択した場合、Operator Lifecycle Manager (OLM) は介入なしに、Operator の実行中のインスタンスを自動的にアップグレードします。手動更新を選択した場合、OLM は更新要求を作成します。クラスター管理者は、Operator を新規バージョンに更新できるように更新要求を手動で承認する必要があります。

5. **Install** をクリックします。

検証

- AWS Load Balancer Operator で、インストール済み Operator ダッシュボードの **Status** が **Succeeded** と表示されることを確認します。

25.3.2. CLI を使用した AWS Load Balancer Operator のインストール

CLI を使用して AWS Load Balancer Operator をインストールできます。

前提条件

- **cluster-admin** 権限を持つユーザーとして OpenShift Container Platform Web コンソールにログインしている。
- クラスターのプラットフォームタイプとクラウドプロバイダーが AWS に設定されている。
- OpenShift CLI (**oc**) にログイン済みである。

手順

1. **Namespace** オブジェクトを作成します。

   a. **Namespace** オブジェクトを定義する YAML ファイルを作成します。

   namespace.yaml ファイルの例

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

   b. 次のコマンドを実行して、**Namespace** オブジェクトを作成します。

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

2. **OperatorGroup** オブジェクトを作成します。
   a. **OperatorGroup** オブジェクトを定義する YAML ファイルを作成します。

   operatorgroup.yaml ファイルの例

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

   b. 以下のコマンドを実行して **OperatorGroup** オブジェクトを作成します。

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

3. **Subscription** オブジェクトを作成します。
   a. **Subscription** オブジェクトを定義する YAML ファイルを作成します。

   subscription.yaml ファイルの例

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

   b. 以下のコマンドを実行して **Subscription** オブジェクトを作成します。

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

検証

1. サブスクリプションからインストールプランの名前を取得します。

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

2. インストールプランのステータスを確認します。

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

出力は **Complete** でなければなりません。

25.4. AWS SECURITY TOKEN SERVICE を使用するクラスターに AWS LOAD BALANCER OPERATOR をインストールする

STS を使用するクラスターに AWS Load Balancer Operator をインストールできます。

AWS Load Balancer Operator は、**CredentialsRequest** オブジェクトに依存して Operator と AWS Load Balancer Controller をブートストラップします。AWS Load Balancer Operator は、必要なシークレットが作成されて利用可能になるまで待機します。

25.4.1. AWS Load Balancer Operator 用の IAM ロールの作成

STS を使用するクラスターに AWS Load Balancer Operator を正常にインストールするには、追加の AWS アイデンティティーおよびアクセス管理 (IAM) ロールが必要です。この IAM ロールは、サブネットおよび Virtual Private Cloud (VPC) と対話するために必要です。AWS Load Balancer Operator は、自身をブートストラップするために IAM ロールを持つ **CredentialsRequest** オブジェクトを生成します。

IAM ロールは次の方法で作成できます。

- Cloud Credential Operator ユーティリティー (**ccoctl**) と事前定義済みの **CredentialsRequest** オブジェクトを使用します。
- AWS CLI と事前定義された AWS マニフェストを使用します。

環境が **ccoctl** コマンドをサポートしていない場合は、AWS CLI を使用します。

25.4.1.1. Cloud Credential Operator ユーティリティーを使用して AWS IAM ロールを作成する

Cloud Credential Operator ユーティリティー (**ccoctl**) を使用して、AWS Load Balancer Operator 用の AWS IAM ロールを作成できます。AWS IAM ロールは、サブネットおよび Virtual Private Cloud (VPC) と対話するために使用されます。

前提条件

- **ccoctl** バイナリーを抽出して準備する必要があります。

手順

1. 次のコマンドを実行して、**CredentialsRequest** カスタムリソース (CR) をダウンロードし、ディレクトリーに保存します。

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

2. **ccoctl** ユーティリティーを使用して次のコマンドを実行し、AWS IAM ロールを作成します。

```
$ ccoctl aws create-iam-roles \
    --name <name> \
    --region=<aws_region> \
    --credentials-requests-dir=<credrequests-dir> \
    --identity-provider-arn <oidc-arn>
```

出力例

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

(1) AWS IAM ロールの Amazon Resource Name (ARN) をメモします。

注記

AWS IAM ロール名の長さは 12 文字以下にする必要があります。

25.4.1.2. AWS CLI を使用して AWS IAM ロールを作成する

AWS コマンドラインインターフェイスを使用して、AWS Load Balancer Operator 用の IAM ロールを作成できます。IAM ロールは、サブネットおよび Virtual Private Cloud (VPC) と対話するために使用されます。

前提条件

- AWS コマンドラインインターフェイス (**aws**) にアクセスできる。

手順

1. 次のコマンドを実行して、アイデンティティープロバイダーを使用して信頼ポリシーファイルを生成します。

```
$ cat <<EOF > albo-operator-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::777777777777:oidc-provider/<oidc-provider-id>" (1)
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "<oidc-provider-id>:sub": "system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-operator-controller-manager" (2)
                }
            }
```

```
      }
    ]
}
EOF
```

1. アイデンティティープロバイダーの Amazon Resource Name (ARN) を指定します。
2. AWS Load Balancer Operator のサービスアカウントを指定します。

2. 次のコマンドを実行して、生成された信頼ポリシーを使用して IAM ロールを作成します。

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

出力例

```
ROLE arn:aws:iam::777777777777:role/albo-operator 2023-08-02T12:13:22Z (1)
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-controller-manager
PRINCIPAL arn:aws:iam:777777777777:oidc-provider/<oidc-provider-id>
```

1. 作成した IAM ロールの ARN をメモします。

3. 次のコマンドを実行して、AWS Load Balancer Operator の権限ポリシーをダウンロードします。

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

4. 次のコマンドを実行して、AWS Load Balancer Controller の権限ポリシーを IAM ロールに割り当てます。

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

25.4.2. AWS Load Balancer Operator 用の ARN ロールの設定

AWS Load Balancer Operator 用の Amazon Resource Name (ARN) ロールを環境変数として設定できます。CLI を使用して ARN ロールを設定できます。

前提条件

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

手順

1. 次のコマンドを実行して、**aws-load-balancer-operator** プロジェクトを作成します。

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

2. 以下のコマンドを実行して **OperatorGroup** オブジェクトを作成します。

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

3. 以下のコマンドを実行して **Subscription** オブジェクトを作成します。

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

(1) AWS Load Balancer Operator の AWS 認証情報をプロビジョニングするために **CredentialsRequest** で使用する ARN ロールを指定します。

注記

AWS Load Balancer Operator は、シークレットが作成されるまで待機してから、**Available** ステータスに移行します。

25.4.3. AWS Load Balancer Controller 用の IAM ロールの作成

AWS Load Balancer Controller の **CredentialsRequest** オブジェクトは、手動でプロビジョニングした IAM ロールを使用して設定する必要があります。

IAM ロールは次の方法で作成できます。

- Cloud Credential Operator ユーティリティー (**ccoctl**) と事前定義済みの **CredentialsRequest** オブジェクトを使用します。
- AWS CLI と事前定義された AWS マニフェストを使用します。

環境が **ccoctl** コマンドをサポートしていない場合は、AWS CLI を使用します。

25.4.3.1. Cloud Credential Operator ユーティリティーを使用してコントローラー用の AWS IAM ロールを作成する

Cloud Credential Operator ユーティリティー (**ccoctl**) を使用して、AWS Load Balancer Controller 用の AWS IAM ロールを作成できます。AWS IAM ロールは、サブネットおよび Virtual Private Cloud (VPC) と対話するために使用されます。

前提条件

- **ccoctl** バイナリーを抽出して準備する必要があります。

手順

1. 次のコマンドを実行して、**CredentialsRequest** カスタムリソース (CR) をダウンロードし、ディレクトリーに保存します。

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

2. **ccoctl** ユーティリティーを使用して次のコマンドを実行し、AWS IAM ロールを作成します。

```
$ ccoctl aws create-iam-roles \
    --name <name> \
    --region=<aws_region> \
    --credentials-requests-dir=<credrequests-dir> \
    --identity-provider-arn <oidc-arn>
```

出力例

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

1 AWS IAM ロールの Amazon Resource Name (ARN) をメモします。

注記

AWS IAM ロール名の長さは 12 文字以下にする必要があります。

25.4.3.2. AWS CLI を使用してコントローラー用の AWS IAM ロールを作成する

AWS コマンドラインインターフェイスを使用して、AWS Load Balancer Controller 用の AWS IAM ロールを作成できます。AWS IAM ロールは、サブネットおよび Virtual Private Cloud (VPC) と対話するために使用されます。

前提条件

- AWS コマンドラインインターフェイス (**aws**) にアクセスできる。

手順

1. 次のコマンドを実行して、アイデンティティプロバイダーを使用して信頼ポリシーファイルを生成します。

```
$ cat <<EOF > albo-controller-trust-policy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::777777777777:oidc-provider/<oidc-provider-id>" (1)
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "<oidc-provider-id>:sub": "system:serviceaccount:aws-load-balancer-
operator:aws-load-balancer-controller-cluster" (2)
                }
            }
        }
    ]
}
EOF
```

(1) アイデンティティープロバイダーの Amazon Resource Name (ARN) を指定します。

(2) AWS Load Balancer Controller のサービスアカウントを指定します。

2. 次のコマンドを実行して、生成された信頼ポリシーを使用して AWS IAM ロールを作成します。

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

出力例

```
ROLE arn:aws:iam::777777777777:role/albo-controller 2023-08-02T12:13:22Z (1)
ASSUMEROLEPOLICYDOCUMENT 2012-10-17
STATEMENT sts:AssumeRoleWithWebIdentity Allow
STRINGEQUALS system:serviceaccount:aws-load-balancer-operator:aws-load-balancer-
controller-cluster
PRINCIPAL arn:aws:iam:777777777777:oidc-provider/<oidc-provider-id>
```

(1) AWS IAM ロールの ARN をメモします。

3. 次のコマンドを実行して、AWS Load Balancer Controller の権限ポリシーをダウンロードします。

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

4. 次のコマンドを実行して、AWS Load Balancer Controller の権限ポリシーを AWS IAM ロールに割り当てます。

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

5. **AWSLoadBalancerController** オブジェクトを定義する YAML ファイルを作成します。

sample-aws-lb-manual-creds.yaml ファイルの例:

```
apiVersion: networking.olm.openshift.io/v1
kind: AWSLoadBalancerController (1)
metadata:
  name: cluster (2)
spec:
  credentialsRequestConfig:
    stsIAMRoleARN: <role-arn> (3)
```

(1) **AWSLoadBalancerController** オブジェクトを定義します。

(2) AWS Load Balancer Controller 名を定義します。このインスタンス名は、関連するすべてのリソースの接尾辞として使用されます。

(3) ARN ロールを指定します。**CredentialsRequest** オブジェクトは、この ARN ロールを使用して AWS 認証情報をプロビジョニングします。

25.4.4. 関連情報

- Cloud Credential Operator ユーティリティーの設定

25.5. AWS LOAD BALANCER CONTROLLER のインスタンスを作成する

AWS Load Balancer Operator をインストールしたら、AWS Load Balancer Controller を作成できます。

25.5.1. AWS Load Balancer Controller の作成

クラスターにインストールできる **AWSLoadBalancerController** オブジェクトのインスタンスは 1 つだけです。CLI を使用して AWS Load Balancer Controller を作成できます。AWS Load Balancer Operator は、**cluster** という名前のリソースのみを調整します。

前提条件

- **echoserver** namespace を作成している。
- OpenShift CLI (**oc**) にアクセスできる。

手順

1. **AWSLoadBalancerController** オブジェクトを定義する YAML ファイルを作成します。

   sample-aws-lb.yaml ファイルの例

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

   (1) **AWSLoadBalancerController** オブジェクトを定義します。

   (2) AWS Load Balancer Controller 名を定義します。このインスタンス名は、関連するすべてのリソースの接尾辞として追加されます。

   (3) AWS Load Balancer Controller のサブネットのタグ付け方法を設定します。次の値が有効です。

   - **Auto**: AWS Load Balancer Operator は、クラスターに属するサブネットを決定し、適切にタグ付けします。内部サブネットタグが内部サブネットに存在しない場合、Operator はロールを正しく判別できません。
   - **Manual**: クラスターに属するサブネットに適切なロールタグを手動でタグ付けします。ユーザー提供のインフラストラクチャーにクラスターをインストールした場合は、このオプションを使用します。

   (4) AWS Load Balancer Controller が AWS リソースをプロビジョニングするときに使用するタグを定義します。

   (5) Ingress クラス名を定義します。デフォルト値は **alb** です。

   (6) AWS Load Balancer Controller のレプリカの数を指定します。

   (7) AWS Load Balancer Controller のアドオンとしてアノテーションを指定します。

   (8) **alb.ingress.kubernetes.io/wafv2-acl-arn** アノテーションを有効にします。

2. 次のコマンドを実行して、**AWSLoadBalancerController** オブジェクトを作成します。

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

3. **Deployment** リソースを定義する YAML ファイルを作成します。

sample-aws-lb.yaml ファイルの例

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

(1) デプロイメントリソースを定義します。

(2) デプロイメント名を指定します。

(3) デプロイメントのレプリカの数を指定します。

4. **Service** リソースを定義する YAML ファイルを作成します。

service-albo.yaml ファイルの例:

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

1 サービスリソースを定義します。

2 サービス名を指定します。

5. **Ingress** リソースを定義する YAML ファイルを作成します。

Ingress-albo.yaml ファイルの例:

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

1 **Ingress** リソースの名前を指定します。

2 サービス名を指定します。

検証

- 次のコマンドを実行して、**Ingress** リソースのステータスを **HOST** 変数に保存します。

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

- 次のコマンドを実行して、**Ingress** リソースのステータスを確認します。

```
$ curl $HOST
```

25.6. 1 つの AWS ロードバランサーを介して複数の INGRESS リソースを提供する

1 つの AWS Load Balancer を介して、1 つのドメインに含まれるさまざまなサービスにトラフィックをルーティングできます。各 Ingress リソースは、ドメインの異なるエンドポイントを提供します。

25.6.1. 1 つの AWS ロードバランサーを介して複数の Ingress リソースを作成する

CLI を使用すると、1 つの AWS ロードバランサーを介して複数の Ingress リソースにトラフィックをルーティングできます。

前提条件

- OpenShift CLI (**oc**) にアクセスできる。

手順

1. 次のように、**IngressClassParams** リソースの YAML ファイル (例: **sample-single-lb-params.yaml**) を作成します。

```
apiVersion: elbv2.k8s.aws/v1beta1 1
kind: IngressClassParams
metadata:
  name: single-lb-params 2
spec:
  group:
    name: single-lb 3
```

1. **IngressClassParams** リソースの API グループとバージョンを定義します。
2. **IngressClassParams** リソース名を指定します。
3. **IngressGroup** リソース名を指定します。このクラスのすべての **Ingress** リソースは、この **IngressGroup** に属します。

2. 次のコマンドを実行して、**IngressClassParams** リソースを作成します。

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

3. 次のように、**IngressClass** リソースの YAML ファイル (例: **sample-single-lb-class.yaml**) を作成します。

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

1. API グループと **IngressClass** リソースのバージョンを定義します。
2. Ingress クラス名を指定します。
3. コントローラー名を定義します。**ingress.k8s.aws/alb** という値は、このクラスのすべての Ingress リソースを AWS Load Balancer Controller によって管理することを意味します。

4 **IngressClassParams** リソースの API グループを定義します。

5 **IngressClassParams** リソースのリソースタイプを定義します。

6 **IngressClassParams** リソース名を定義します。

4. 次のコマンドを実行して **IngressClass** リソースを作成します。

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

5. 次のように、**AWSLoadBalancerController** リソース YAML ファイル (例: **sample-single-lb.yaml**) を作成します。

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

1 **IngressClass** リソースの名前を定義します。

6. 次のコマンドを実行して、**AWSLoadBalancerController** リソースを作成します。

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

7. 次のように、**Ingress** リソースの YAML ファイル (例: **sample-multiple-ingress.yaml**) を作成します。

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

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

1. Ingress 名を指定します。
2. インターネットにアクセスするためにパブリックサブネットにプロビジョニングするロードバランサーを示します。
3. ロードバランサーで要求を受信したときに、複数の Ingress リソースからのルールをマッチする順序を指定します。
4. ロードバランサーがサービスに到達するために OpenShift Container Platform ノードをターゲットにすることを示します。
5. この Ingress に属する Ingress クラスを指定します。

6 要求のルーティングに使用するドメイン名を定義します。

7 サービスにルーティングする必要があるパスを定義します。

8 **Ingress** リソースで設定されたエンドポイントを提供するサービス名を定義します。

9 エンドポイントにサービスを提供するサービスのポートを定義します。

8. 次のコマンドを実行して **Ingress** リソースを作成します。

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

25.7. TLS TERMINATION の追加

AWS Load Balancer に TLS Termination を追加できます。

25.7.1. AWS Load Balancer への TLS Termination の追加

ドメインのトラフィックをサービスの Pod にルーティングし、AWS Load Balancer に TLS Termination を追加できます。

前提条件

- OpenShift CLI (**oc**) にアクセスできる。

手順

1. **AWSLoadBalancerController** リソースを定義する YAML ファイルを作成します。

   add-tls-termination-albc.yaml ファイルの例

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

1 Ingress クラス名を定義します。クラスター内に Ingress クラスが存在しない場合は、AWS Load Balancer Controller によって作成されます。**spec.controller** が **ingress.k8s.aws/alb** に設定されている場合、AWS Load Balancer Controller は追加の Ingress クラス値を調整します。

2. **Ingress** リソースを定義する YAML ファイルを作成します。

   add-tls-termination-ingress.yaml ファイルの例

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <example> 1
```

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

1. Ingress 名を指定します。
2. コントローラーは、パブリックサブネット内の Ingress のロードバランサーをプロビジョニングし、インターネット経由でロードバランサーにアクセスします。
3. ロードバランサーに割り当てる証明書の Amazon Resource Name (ARN)。
4. Ingress クラス名を定義します。
5. トラフィックルーティングのドメインを定義します。
6. トラフィックルーティングのサービスを定義します。

25.8. クラスター全体のプロキシーの設定

AWS Load Balancer Operator でクラスター全体のプロキシーを設定できます。クラスター全体のプロキシーを設定すると、Operator Lifecycle Manager (OLM) が、**HTTP_PROXY**、**HTTPS_PROXY**、**NO_PROXY** などの環境変数を使用して、Operator のすべてのデプロイメントを自動的に更新します。これらの変数は、AWS Load Balancer Operator によってマネージドコントローラーに入力されます。

25.8.1. クラスター全体のプロキシーの認証局を信頼する

1. 次のコマンドを実行して、**aws-load-balancer-operator** namespace に認証局 (CA) バンドルを含める config map を作成します。

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

2. 信頼できる CA バンドルを config map に挿入するには、次のコマンドを実行して、**config.openshift.io/inject-trusted-cabundle=true** ラベルを config map に追加します。

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

3. 次のコマンドを実行して、AWS Load Balancer Operator デプロイメントの config map にアクセスできるように AWS Load Balancer Operator サブスクリプションを更新します。

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

4. AWS Load Balancer Operator がデプロイされたら、次のコマンドを実行して、CA バンドルが **aws-load-balancer-operator-controller-manager** デプロイメントに追加されていることを確認します。

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

出力例

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

5. オプション: config-map が変更されるたびに、次のコマンドを実行して、AWS Load Balancer Operator のデプロイを再開します。

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

25.8.2. 関連情報

- Operator を使用した証明書の挿入

第26章 複数ネットワーク

26.1. 複数ネットワークについて

Kubernetes では、コンテナーネットワークは 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。これにより、スイッチングやルーティングなどのネットワーク機能を提供する Pod を設定する際に柔軟性が得られます。

26.1.1. 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ー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。

セキュリティー

機密トラフィックは、セキュリティー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。

26.1.2. 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。

- **vlan**: VLAN ベースの追加ネットワークを設定 して、VLAN ベースのネットワークの分離と Pod の接続を可能にします。
- **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。
- **Tap**: タップベースの追加ネットワークを設定 して、コンテナー namespace 内にタップデバイスを作成します。タップデバイスを使用すると、ユーザー空間プログラムがネットワークパケットを送受信できるようになります。
- **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す。

26.2. 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
- VLAN
- IPVLAN
- MACVLAN
- TAP
- OVN-Kubernetes

26.2.1. 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。

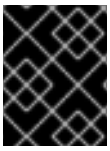
注記

OVN SDN を使用して、Red Hat OpenStack Platform (RHOSP) に複数のネットワークインターフェイスを持つ OpenShift Container Platform ノードをデプロイすると、セカンダリーインターフェイスの DNS 設定がプライマリーインターフェイスの DNS 設定よりも優先される場合があります。この場合は、セカンダリーインターフェイスに接続されているサブネット ID の DNS ネームサーバーを削除します。

```
$ openstack subnet set --dns-nameserver 0.0.0.0 <subnet_id>
```

26.2.2. 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 **NetworkAttachmentDefinition** API を使用して設定されます。



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26.1 NetworkAttachmentDefinition API フィールド

フィールド	型	説明
metadata.name	**string**	追加のネットワークの名前です。
metadata.namespace	**string**	オブジェクトが関連付けられる namespace。
spec.config	**string**	JSON 形式の CNI プラグイン設定。

26.2.2.1. 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: 1
  - name: <name> 2
    namespace: <namespace> 3
    rawCNIConfig: |- 4
```

```
    {
      ...
    }
  type: Raw
```

1. 1 つまたは複数の追加ネットワーク設定の配列。
2. 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。
3. 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。
4. JSON 形式の CNI プラグイン設定。

26.2.2.2. 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> 1
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。

26.2.3. 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。

26.2.3.1. 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26.2 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**	設定する CNI プラグインの名前: **bridge**。

フィールド	型	説明
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。
bridge	**string**	オプション: 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。
ipMasq	**boolean**	オプション: 仮想ネットワークから外すトラフィックの 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**	オプション: 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**	オプション: 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。
forceAddress	**boolean**	オプション: 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**	オプション: 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**	オプション: 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**	オプション: 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。
preserveDefaultVlan	**string**	オプション: デフォルトの VLAN をブリッジに接続されている **veth** 側で保持する必要があるか示します。デフォルトは true です。
vlanTrunk	**list**	オプション: VLAN トランクタグを割り当てます。デフォルト値は **none** です。

フィールド	型	説明
mtu	**string**	オプション: 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。
enabledad	**boolean**	オプション: コンテナー側の **veth** の重複アドレス検出を有効にします。デフォルト値は **false** です。
macspoofchk	**boolean**	オプション: MAC スプーフィングチェックを有効にして、コンテナーから発信されるトラフィックをインターフェイスの MAC アドレスに制限します。デフォルト値は **false** です。

注記

VLAN パラメーターは、**veth** のホスト側に VLAN タグを設定し、ブリッジインターフェイスで **vlan_filtering** 機能を有効にします。

注記

L2 ネットワークのアップリンクを設定するには、以下のコマンドを使用してアップリンクインターフェイスで vlan を許可する必要があります。

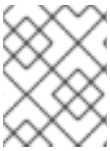
```
$ bridge vlan add vid VLAN_ID dev DEV
```

26.2.3.1.1. 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": "bridge-net",
  "type": "bridge",
  "isGateway": true,
  "vlan": 2,
  "ipam": {
    "type": "dhcp"
    }
}
```

26.2.3.2. 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26.3 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。

26.2.3.2.1. 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": "hostdev-net",
  "type": "host-device",
  "device": "eth1"
}
```

26.2.3.3. VLAN 追加ネットワークの設定

以下のオブジェクトは、VLAN 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26.4 VLAN 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: **vlan**
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
vlanId	**integer**	vlan の ID を設定します。
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。
mtu	**integer**	オプション: 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。
dns	**integer**	オプション: 返す DNS 情報 (DNS ネームサーバーの優先順位リストなど)。
linkInContainer	**boolean**	オプション: マスターインターフェイスが、コンテナーネットワーク namespace とメインネットワーク namespace のどちらにあるかを指定します。コンテナー namespace マスターインターフェイスの使用を要求するには、値を **true** に設定します。

26.2.3.3.1. vlan 設定例

以下の例では、**ipvlan-net** という名前の追加ネットワークを設定します。

```
{
  "name": "vlan-net",
  "cniVersion": "0.3.1",
  "type": "vlan",
  "master": "eth0",
  "mtu": 1500,
  "vlanId": 5,
  "linkInContainer": false,
  "ipam": {
     "type": "host-local",
     "subnet": "10.1.1.0/24"
  },
  "dns": {
     "nameservers": [ "10.1.1.1", "8.8.8.8" ]
  }
}
```

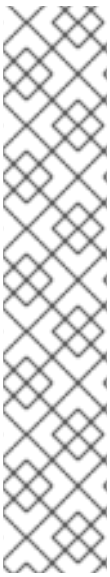
26.2.3.4. IPVLAN 追加ネットワークの設定

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26.5 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。

フィールド	型	説明
type	**string**	設定する CNI プラグインの名前: **ipvlan**。
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。これは、プラグインが連鎖している場合を除き必要です。
mode	**string**	オプション: 仮想ネットワークの操作モードを指定します。この値は、**l2**、**l3**、または **l3s** である必要があります。デフォルト値は **l2** です。
master	**string**	オプション: 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を指定します。**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。
mtu	**integer**	オプション: 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。
linkInContainer	**boolean**	オプション: マスターインターフェイスが、コンテナーネットワーク namespace とメインネットワーク namespace のどちらにあるかを指定します。コンテナー namespace マスターインターフェイスの使用を要求するには、値を **true** に設定します。



注記

- **ipvlan** オブジェクトは、仮想インターフェイスが **master** インターフェイスと通信することを許可しません。したがって、コンテナーは **ipvlan** インターフェイスを使用してホストに到達できなくなります。コンテナーが、Precision Time Protocol (**PTP**) をサポートするネットワークなど、ホストへの接続を提供するネットワークに参加していることを確認してください。
- 1 つのの **master** インターフェイスを、**macvlan** と **ipvlan** の両方を使用するように同時に設定することはできません。
- インターフェイスに依存できない IP 割り当てスキームの場合、**ipvlan** プラグインは、このロジックを処理する以前のプラグインと連鎖させることができます。**master** が省略された場合、前の結果にはスレーブにする **ipvlan** プラグインのインターフェイス名が 1 つ含まれていなければなりません。**ipam** が省略された場合、**ipvlan** インターフェイスの設定には前の結果が使用されます。

26.2.3.4.1. 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": "ipvlan-net",
  "type": "ipvlan",
  "master": "eth1",
```

```
  "linkInContainer": false,
  "mode": "l3",
  "ipam": {
    "type": "static",
    "addresses": [
      {
        "address": "192.168.10.10/24"
      }
    ]
  }
}
```

26.2.3.5. 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26.6 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**。
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。
mode	**string**	オプション: 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。
master	**string**	オプション: 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**	オプション: 指定された値への最大転送単位 (MTU)。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。
linkInContainer	**boolean**	オプション: マスターインターフェイスが、コンテナーネットワーク namespace とメインネットワーク namespace のどちらにあるかを指定します。コンテナー namespace マスターインターフェイスの使用を要求するには、値を **true** に設定します。

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。

26.2.3.5.1. macvlan 設定の例

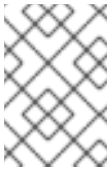
以下の例では、**macvlan-net** という名前の追加のネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "macvlan-net",
  "type": "macvlan",
  "master": "eth1",
  "linkInContainer": false,
  "mode": "bridge",
  "ipam": {
    "type": "dhcp"
    }
}
```

26.2.3.6. TAP 追加ネットワークの設定

以下のオブジェクトは、TAP 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26.7 TAP 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: **tap**
mac	**string**	オプション: インターフェイスの指定された MAC アドレスを要求します。
mtu	**integer**	オプション: 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。
selinuxcontext	**string**	オプション: タップデバイスに関連付ける SELinux コンテキスト。  **注記** OpenShift Container Platform には、値 **system_u:system_r:container_t:s0** が必要です。

フィールド	型	説明
multiQueue	**boolean**	オプション: マルチキューを有効にするには **true** に設定します。
owner	**integer**	オプション: タップデバイスを所有するユーザー。
group	**integer**	オプション: タップデバイスを所有するグループ。
bridge	**string**	オプション: タップデバイスを既存のブリッジのポートとして設定します。

26.2.3.6.1. Tap 設定の例

以下の例では、**mynet** という名前の追加ネットワークを設定します。

```
{
 "name": "mynet",
 "cniVersion": "0.3.1",
 "type": "tap",
 "mac": "00:11:22:33:44:55",
 "mtu": 1500,
 "selinuxcontext": "system_u:system_r:container_t:s0",
 "multiQueue": true,
 "owner": 0,
 "group": 0
 "bridge": "br1"
}
```

26.2.3.6.2. TAP CNI プラグインの SELinux ブール値の設定

Container_t SELinux コンテキストを使用して Tap デバイスを作成するには、Machine Config Operator (MCO) を使用してホスト上で **container_use_devices** ブール値を有効にします。

前提条件

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

手順

1. 次の詳細を含む、**setsebool-container-use-devices.yaml** などの名前の新しい YAML ファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-worker-setsebool
spec:
```

```
config:
  ignition:
    version: 3.2.0
  systemd:
    units:
    - enabled: true
      name: setsebool.service
      contents: |
        [Unit]
        Description=Set SELinux boolean for the TAP CNI plugin
        Before=kubelet.service

        [Service]
        Type=oneshot
        ExecStart=/usr/sbin/setsebool container_use_devices=on
        RemainAfterExit=true

        [Install]
        WantedBy=multi-user.target graphical.target
```

2. 次のコマンドを実行して、新しい **MachineConfig** オブジェクトを作成します。

```
$ oc apply -f setsebool-container-use-devices.yaml
```

注記

MachineConfig オブジェクトに変更を適用すると、変更が適用された後、影響を受けるすべてのノードが正常に再起動します。この更新が適用されるまでに、時間がかかる場合があります。

3. 次のコマンドを実行して、変更が適用されていることを確認します。

```
$ oc get machineconfigpools
```

予想される出力

```
NAME       CONFIG                                             UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
DEGRADEDMACHINECOUNT   AGE
master     rendered-master-e5e0c8e8be9194e7c5a882e047379cfa     True      False
False      3              3                 3                   0                   7d2h
worker     rendered-worker-d6c9ca107fba6cd76cdcbfcedcafa0f2     True      False      False
3              3                 3                   0                   7d
```

注記

すべてのノードが更新され、準備完了状態になっている必要があります。

関連情報

- ノード上で SELinux ブール値を有効にする方法の詳細は、SELinux ブール値の設定 を参照してください。

26.2.3.7. OVN-Kubernetes 追加ネットワークの設定

Red Hat OpenShift Networking OVN-Kubernetes ネットワークプラグインを使用すると、Pod のセカンダリーネットワークインターフェイスを設定できます。セカンダリーネットワークインターフェイスを設定するには、**NetworkAttachmentDefinition** カスタムリソース (CR) で設定を定義する必要があります。

注記

Pod およびマルチネットワークポリシーの作成は、ノード内の OVN-Kubernetes コントロールプレーンエージェントが関連する **network-attachment-definition** CR を処理するまで、保留状態のままになる場合があります。

OVN-Kubernetes 追加ネットワークは、**レイヤー 2** または **ローカルネット** トポロジーで設定できます。

- レイヤ 2 トポロジーは、East-West クラスタートラフィックをサポートしますが、基礎となる物理ネットワークへのアクセスは許可しません。
- ローカルネットトポロジーでは物理ネットワークへの接続が可能ですが、クラスターノード上の基盤となる Open vSwitch (OVS) ブリッジの追加設定が必要です。

次のセクションでは、OVN-Kubernetes で現在セカンダリーネットワークに許可されている各トポロジーの設定例を示します。

注記

ネットワーク名は一意である必要があります。たとえば、同じネットワークを参照する異なる設定を持つ複数の **NetworkAttachmentDefinition** CR の作成はサポートされていません。

26.2.3.7.1. OVN-Kubernetes 追加ネットワークでサポートされるプラットフォーム

OVN-Kubernetes 追加ネットワークは、次のサポートされているプラットフォームで使用できます。

- ベアメタル
- IBM Power®
- IBM Z®
- IBM® LinuxONE
- VMware vSphere
- Red Hat OpenStack Platform (RHOSP)

26.2.3.7.2. OVN-Kubernetes ネットワークプラグインの JSON 設定テーブル

次の表では、OVN-Kubernetes CNI ネットワークプラグインの設定パラメーターについて説明します。

表26.8 OVN-Kubernetes ネットワークプラグインの JSON 設定テーブル

フィールド	型	説明
cniVersion	**string**	CNI 仕様のバージョン。必要な値は **0.3.1** です。
name	**string**	ネットワークの名前。これらのネットワークには namespace が使用されていません。たとえば、**l2-network** という名前のネットワークを、2 つの異なる namespace に存在する 2 つの異なる **NetworkAttachmentDefinition** から参照させることができます。これにより、独自の異なる namespace で **NetworkAttachmentDefinition** を使用する Pod が同じセカンダリーネットワーク上で通信できるようになります。ただし、これら 2 つの異なる **NetworkAttachmentDefinition** は、**topology**、**subnets**、**mtu**、**excludeSubnets** などの同じネットワーク固有のパラメーターも共有する必要があります。
type	**string**	設定する CNI プラグインの名前。この値は **ovn-k8s-cni-overlay** に設定する必要があります。
topology	**string**	ネットワークのトポロジー設定。**layer2** または **localnet** のいずれかである必要があります。
subnets	**string**	クラスター全体のネットワークに使用するサブネット。 **"topology":"layer2"** デプロイメントでは、IPv6 (**2001:DBB::/64**) およびデュアルスタック (**192.168.100.0/24,2001:DBB::/64**) サブネットがサポートされています。 省略した場合、ネットワークを実装する論理スイッチはレイヤー 2 通信のみを提供し、ユーザーは Pod の IP アドレスを設定する必要があります。ポートセキュリティーは、MAC スプーフィングのみを防止します。
mtu	**string**	最大伝送単位 (MTU)。デフォルト値 **1300** は、カーネルによって自動的に設定されます。
netAttachDefName	**string**	この設定が含まれるネットワーク接続定義オブジェクトのメタデータ **namespace** と **name**。たとえば、この設定が **l2-network** という namespace **ns1** の **NetworkAttachmentDefinition** で定義されている場合、これを **ns1/l2-network** に設定する必要があります。
excludeSubnets	**string**	CIDR と IP アドレスのコンマ区切りのリスト。IP アドレスは割り当て可能な IP アドレスプールから削除され、Pod に渡されることはありません。
vlanID	**integer**	トポロジーが **localnet** に設定されている場合、指定された VLAN タグがこの追加ネットワークからのトラフィックに割り当てられます。デフォルトでは、VLAN タグは割り当てられません。

26.2.3.7.3. マルチネットワークポリシーとの互換性

k8s.cni.cncf.io API グループの **MultiNetworkPolicy** カスタムリソース定義 (CRD) によって提供されるマルチネットワークポリシー API は、OVN-Kubernetes セカンダリーネットワークと互換性があります。ネットワークポリシーを定義する場合、使用できるネットワークポリシールールは、OVN-Kubernetes セカンダリーネットワークが **subnets** フィールドを定義しているかどうかによって異なります。詳細は、次の表を参照してください。

表26.9 subnets CNI 設定に基づいてサポートされるマルチネットワークポリシーセレクター

subnets フィールドの指定	許可されたマルチネットワークポリシーセレクター
はい	• **podSelector** と **namespaceSelector** • **ipBlock**
いいえ	• **ipBlock**

たとえば、次のマルチネットワークポリシーは、**blue2** という名前の追加ネットワークの追加ネットワーク CNI 設定で **subnets** フィールドが定義されている場合にのみ有効です。

Pod セレクターを使用するマルチネットワークポリシーの例

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-same-namespace
  annotations:
    k8s.v1.cni.cncf.io/policy-for: blue2
spec:
  podSelector:
  ingress:
  - from:
    - podSelector: {}
```

次の例では、**ipBlock** ネットワークポリシーセレクターを使用します。これは、OVN-Kubernetes 追加ネットワークに対して常に有効です。

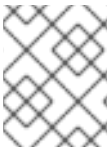
IP ブロックセレクターを使用するマルチネットワークポリシーの例

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name:  ingress-ipblock
  annotations:
    k8s.v1.cni.cncf.io/policy-for: default/flatl2net
spec:
  podSelector:
    matchLabels:
      name: access-control
  policyTypes:
```

```
  - Ingress
  ingress:
  - from:
    - ipBlock:
        cidr: 10.200.0.0/30
```

26.2.3.7.4. レイヤー 2 スイッチドトポロジーの設定

スイッチド (レイヤー 2) トポロジーネットワークは、クラスター全体の論理スイッチを介してワークロードを相互接続します。この設定は、IPv6 およびデュアルスタックデプロイメントに使用できます。



注記

レイヤー 2 スイッチドトポロジーネットワークでは、クラスター内の Pod 間のデータパケットの転送のみが許可されます。

次の JSON 例では、スイッチドセカンダリーネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "l2-network",
  "type": "ovn-k8s-cni-overlay",
  "topology":"layer2",
  "subnets": "10.100.200.0/24",
  "mtu": 1300,
  "netAttachDefName": "ns1/l2-network",
  "excludeSubnets": "10.100.200.0/29"
}
```

26.2.3.7.5. ローカルネットトポロジーの設定

スイッチド (ローカルネット) トポロジーは、クラスター全体の論理スイッチを介してワークロードを物理ネットワークに相互接続します。

26.2.3.7.5.1. OVN-Kubernetes 追加ネットワークを設定するための前提条件

- NMState Operator がインストールされている。詳細は、Kubernetes NMState Operator について を参照してください。

26.2.3.7.5.2. OVN-Kubernetes 追加ネットワークマッピングの設定

OVN-Kubernetes 追加ネットワークとして使用するには、追加ネットワークを OVN ブリッジにマップする必要があります。ブリッジマッピングにより、ネットワークトラフィックが物理ネットワークに到達できるようになります。ブリッジマッピングは、インターフェイスラベルとも呼ばれる物理ネットワーク名を、Open vSwitch (OVS) で作成されたブリッジに関連付けます。

nmstate.io/v1 API グループの一部である **NodeNetworkConfigurationPolicy** オブジェクトを作成して、宣言的にマッピングを作成できます。この API は NMState Operator によって提供されます。この API を使用すると、指定した **nodeSelector** 式 (**node-role.kubernetes.io/worker: ''** など) に一致するノードにブリッジマッピングを適用できます。

追加のネットワークを接続する場合、既存の **br-ex** ブリッジを使用することも、新しいブリッジを作成することもできます。どのアプローチを使用するかは、特定のネットワークインフラストラクチャーによって異なります。

- ノードにネットワークインターフェイスが 1 つしか含まれていない場合は、既存のブリッジを使用する必要があります。このネットワークインターフェイスは OVN-Kubernetes によって所有および管理されているため、**br-ex** ブリッジから削除したり、インターフェイス設定を変更したりしないでください。ネットワークインターフェイスを削除または変更すると、クラスターネットワークは正しく動作しなくなります。

- ノードに複数のネットワークインターフェイスが含まれている場合は、別のネットワークインターフェイスを新しいブリッジに接続して、追加のネットワークに使用できます。このアプローチでは、プライマリークラスターネットワークからトラフィックが分離されます。

次の例では、**localnet1** ネットワークが **br-ex** ブリッジにマッピングされています。

ブリッジを共有するためのマッピングの例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: mapping 1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: '' 2
  desiredState:
    ovn:
      bridge-mappings:
      - localnet: localnet1 3
        bridge: br-ex 4
        state: present 5
```

1. 設定オブジェクトの名前。
2. ノードネットワーク設定ポリシーを適用するノードを指定するノードセレクター。
3. トラフィックが OVS ブリッジに転送される追加ネットワークの名前。この追加ネットワークは、OVN-Kubernetes 追加ネットワークを定義する **NetworkAttachmentDefinition** オブジェクトの **spec.config.name** フィールドの名前と一致する必要があります。
4. ノード上の OVS ブリッジの名前。この値は、**state: present** を指定する場合にのみ必要です。
5. マッピングの状態。ブリッジを追加する場合は **present**、ブリッジを削除する場合は **absent** である必要があります。デフォルト値は **present** です。

次の例では、**localnet2** ネットワークインターフェイスが **ovs-br1** ブリッジに接続されています。この接続を使って、ネットワークインターフェイスを OVN-Kubernetes ネットワークプラグインで追加のネットワークとして利用できるようになります。

複数のインターフェイスを持つノードのマッピング例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ovs-br1-multiple-networks 1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: '' 2
```

```
  desiredState:
    interfaces:
    - name: ovs-br1 3
      description: |-
        A dedicated OVS bridge with eth1 as a port
        allowing all VLANs and untagged traffic
      type: ovs-bridge
      state: up
      bridge:
        options:
          stp: true
        port:
        - name: eth1 4
    ovn:
      bridge-mappings:
      - localnet: localnet2 5
        bridge: ovs-br1 6
        state: present 7
```

1 設定オブジェクトの名前。

2 ノードネットワーク設定ポリシーを適用するノードを指定するノードセレクター。

3 OVN-Kubernetes がすべてのクラスタートラフィックに使用するデフォルトブリッジとは別の、新しい OVS ブリッジ。

4 この新しい OVS ブリッジに関連付けるホストシステム上のネットワークデバイス。

5 トラフィックが OVS ブリッジに転送される追加ネットワークの名前。この追加ネットワークは、OVN-Kubernetes 追加ネットワークを定義する **NetworkAttachmentDefinition** オブジェクトの **spec.config.name** フィールドの名前と一致する必要があります。

6 ノード上の OVS ブリッジの名前。この値は、**state: present** を指定する場合にのみ必要です。

7 マッピングの状態。ブリッジを追加する場合は **present**、ブリッジを削除する場合は **absent** である必要があります。デフォルト値は **present** です。

NMState Operator は、ノードセレクターによって指定されたすべてのノードに追加のネットワーク設定を自動的かつ透過的に適用するため、この宣言的アプローチが推奨されます。

次の JSON 例では、localnet セカンダリーネットワークを設定します。

```
{
  "cniVersion": "0.3.1",
  "name": "ns1-localnet-network",
  "type": "ovn-k8s-cni-overlay",
  "topology":"localnet",
  "subnets": "202.10.130.112/28",
  "vlanID": 33,
  "mtu": 1500,
  "netAttachDefName": "ns1/localnet-network"
  "excludeSubnets": "10.100.200.0/29"
}
```

26.2.3.7.6. 追加ネットワーク用の Pod の設定

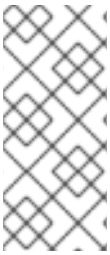
k8s.v1.cni.cncf.io/networks アノテーションを使用して、セカンダリーネットワーク割り当てを指定する必要があります。

次の例では、このガイドに示されている割り当て設定ごとに 1 つずつ、2 つのセカンダリー割り当てを持つ Pod をプロビジョニングします。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: l2-network
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
    - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container
```

26.2.3.7.7. 静的 IP アドレスを使用して Pod を設定する

次の例では、静的 IP アドレスを使用して Pod をプロビジョニングします。



注記

- レイヤー 2 割り当てに対する Pod のセカンダリーネットワーク割り当ての IP アドレスのみを指定できます。
- Pod の静的 IP アドレスを指定できるのは、割り当て設定にサブネットが含まれていない場合のみです。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "l2-network", (1)
        "mac": "02:03:04:05:06:07", (2)
        "interface": "myiface1", (3)
        "ips": [
          "192.0.2.20/24"
          ] (4)
      }
    ]'
  name: tinypod
  namespace: ns1
spec:
  containers:
  - args:
```

```
    - pause
    image: k8s.gcr.io/e2e-test-images/agnhost:2.36
    imagePullPolicy: IfNotPresent
    name: agnhost-container
```

1. ネットワークの名前。この値は、すべての **NetworkAttachmentDefinitions** で一意である必要があります。
2. インターフェイスに割り当てられる MAC アドレス。
3. Pod 用に作成されるネットワークインターフェイスの名前。
4. ネットワークインターフェイスに割り当てられる IP アドレス。

26.2.4. 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。

26.2.4.1. 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26.10 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26.11 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26.12 **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26.13 **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。
search	**array**	DNS ルックアップのクエリー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"
        }
      ]
  }
}
```

26.2.4.2. 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26.14 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"
  }
}
```

26.2.4.3. 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26.15 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) のリスト。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"
    ]
  }
}
```

26.2.4.4. whereabouts-reconciler デーモンセットの作成

Whereabouts reconciler は、Whereabouts IP アドレス管理 (IPAM) ソリューションを使用して、クラスター内の Pod の動的 IP アドレス割り当てを管理します。これにより、各 Pod が指定の IP アドレス範囲から一意の IP アドレスを確実に取得します。また、Pod が削除またはスケールダウンされた場合の IP アドレスの解放も処理します。

注記

NetworkAttachmentDefinition カスタムリソース (CR) を使用して動的 IP アドレスを割り当てることもできます。

whereabouts-reconciler デーモンセットは、Cluster Network Operator を通じて追加のネットワークを設定するときに自動的に作成されます。YAML マニフェストから追加のネットワークを設定する場合、これは自動的には作成されません。

whereabouts-reconciler デーモンセットのデプロイをトリガーするには、Cluster Network Operator のカスタムリソース (CR) ファイルを編集して、**whereabouts-shim** ネットワーク割り当てを手動で作成する必要があります。

whereabouts-reconciler デーモンセットをデプロイするには、次の手順を使用します。

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. この例で展開されている YAML の **additionalNetworks** セクションを、カスタムリソース (CR) の **spec** 定義内に含めます。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
# ...
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    rawCNIConfig: |-
      {
       "name": "whereabouts-shim",
       "cniVersion": "0.3.1",
       "type": "bridge",
       "ipam": {
         "type": "whereabouts"
       }
      }
    type: Raw
# ...
```

3. ファイルを保存し、テキストエディターを編集します。

4. 次のコマンドを実行して、**whereabouts-reconciler** デーモンセットが正常にデプロイされたことを確認します。

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

出力例

```
pod/whereabouts-reconciler-jnp6g 1/1 Running 0 6s
pod/whereabouts-reconciler-k76gg 1/1 Running 0 6s
pod/whereabouts-reconciler-k86t9 1/1 Running 0 6s
pod/whereabouts-reconciler-p4sxw 1/1 Running 0 6s
pod/whereabouts-reconciler-rvfdv 1/1 Running 0 6s
pod/whereabouts-reconciler-svzw9 1/1 Running 0 6s
daemonset.apps/whereabouts-reconciler 6 6 6 6 6 kubernetes.io/os=linux 6s
```

26.2.4.5. Whereabouts IP リコンサイラーのスケジュールの設定

Whereabouts IPAM CNI プラグインは、IP リコンサイラーを毎日実行します。このプロセスは、IP が枯渇して新しい Pod に IP が割り当てられなくなる状態を避けるために、完了せずに残っている IP 割り当てをクリーンアップします。

IP リコンサイラーを実行する頻度を変更するには、次の手順を使用します。

前提条件

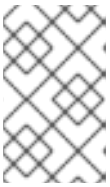
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- **whereabouts-reconciler** デーモンセットがデプロイされており、**whereabouts-reconciler** Pod が起動して実行されている。

手順

1. 次のコマンドを実行して、IP リコンサイラー用の特定の cron 式を使用し、**openshift-multus** namespace に **whereabouts-config** という名前の **ConfigMap** オブジェクトを作成します。

```
$ oc create configmap whereabouts-config -n openshift-multus --from-
literal=reconciler_cron_expression="*/15 * * * *"
```

この cron 式は、IP リコンサイラーを 15 分ごとに実行するよう指定します。この式は固有の要件に基づいて調整してください。



注記

whereabouts-reconciler デーモンセットは、5 つのアスタリスクを含む cron 式パターンのみを使用できます。秒を表すために使用される 6 番目のアスタリスクは、現在サポートされていません。

2. 次のコマンドを実行して、**openshift-multus** namespace 内の **whereabouts-reconciler** デーモンセットおよび Pod に関連するリソースに関する情報を取得します。

```
$ oc get all -n openshift-multus | grep whereabouts-reconciler
```

出力例

```
pod/whereabouts-reconciler-2p7hw                1/1     Running   0          4m14s
pod/whereabouts-reconciler-76jk7                1/1     Running   0          4m14s
pod/whereabouts-reconciler-94zw6                1/1     Running   0          4m14s
pod/whereabouts-reconciler-mfh68                1/1     Running   0          4m14s
pod/whereabouts-reconciler-pgshz                1/1     Running   0          4m14s
pod/whereabouts-reconciler-xn5xz                1/1     Running   0          4m14s
daemonset.apps/whereabouts-reconciler       6         6         6       6          6
kubernetes.io/os=linux   4m16s
```

3. 次のコマンドを実行して、設定した間隔で **whereabouts-reconciler** Pod が IP リコンサイラーを実行していることを確認します。

```
$ oc -n openshift-multus logs whereabouts-reconciler-2p7hw
```

出力例

```
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_33_54.1375928161": CREATE
```

```
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_33_54.1375928161": CHMOD
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/..data_tmp": RENAME
2024-02-02T16:33:54Z [verbose] using expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] configuration updated to file "/cron-schedule/..data". New
cron expression: */15 * * * *
2024-02-02T16:33:54Z [verbose] successfully updated CRON configuration id "00c2d1c9-
631d-403f-bb86-73ad104a6817" - new cron expression: */15 * * * *
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-schedule/config": CREATE
2024-02-02T16:33:54Z [debug] event not relevant: "/cron-
schedule/..2024_02_02_16_26_17.3874177937": REMOVE
2024-02-02T16:45:00Z [verbose] starting reconciler run
2024-02-02T16:45:00Z [debug] NewReconcileLooper - inferred connection data
2024-02-02T16:45:00Z [debug] listing IP pools
2024-02-02T16:45:00Z [debug] no IP addresses to cleanup
2024-02-02T16:45:00Z [verbose] reconciler success
```

26.2.4.6. デュアルスタック IP アドレスを動的に割り当てる設定の作成

デュアルスタックの IP アドレスの割り当ては、**ipRanges** パラメーターで設定できます。

- IPv4 アドレス
- IPv6 アドレス
- 複数の IP アドレスの割り当て

手順

1. **type** を **whereabouts** に設定します。
2. 以下の例のように、**ipRanges** を使用して IP アドレスを割り当てます。

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
       "name": "whereabouts-dual-stack",
       "cniVersion": "0.3.1,
       "type": "bridge",
       "ipam": {
         "type": "whereabouts",
         "ipRanges": [
                {"range": "192.168.10.0/24"},
                {"range": "2001:db8::/64"}
            ]
       }
      }
```

3. ネットワークを Pod にアタッチします。詳細は、追加のネットワークへの Pod の追加を参照してください。
4. すべての IP アドレスが割り当てられていることを確認します。
5. 以下のコマンドを実行して、IP アドレスがメタデータとして割り当てられることを確認します。

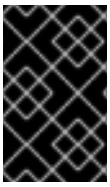
```
$ oc exec -it mypod -- ip a
```

関連情報

- Pod の追加のネットワークへの割り当て

26.2.5. 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. オプション: 追加のネットワークの namespace を作成します。

```
$ oc create namespace <namespace_name>
```

2. 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
```

3. 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: namespace2
```

```
    type: Raw
    rawCNIConfig: |-
     {
       "cniVersion": "0.3.1",
       "name": "tertiary-net",
       "type": "ipvlan",
       "master": "eth1",
       "mode": "l2",
       "ipam": {
         "type": "static",
         "addresses": [
           {
             "address": "192.168.1.23/24"
           }
         ]
       }
     }
```

4. 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
```

26.2.6. 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。

26.2.7. コンテナーネットワーク namespace でのマスターインターフェイスの設定について

OpenShift Container Platform 4.14 以降では、ユーザーがコンテナー namespace 内のマスターインターフェイスに基づいて MAC-VLAN、IP-VLAN、および VLAN サブインターフェイスを作成できる機能が一般提供されるようになりました。

この機能を使用すると、別のネットワーク割り当て定義で、Pod ネットワーク設定の一部としてマスターインターフェイスを作成できます。その後、ノードのネットワーク設定の知識を必要とせずに、このインターフェイスに基づいて VLAN、MACVLAN、または IPVLAN を構築できます。

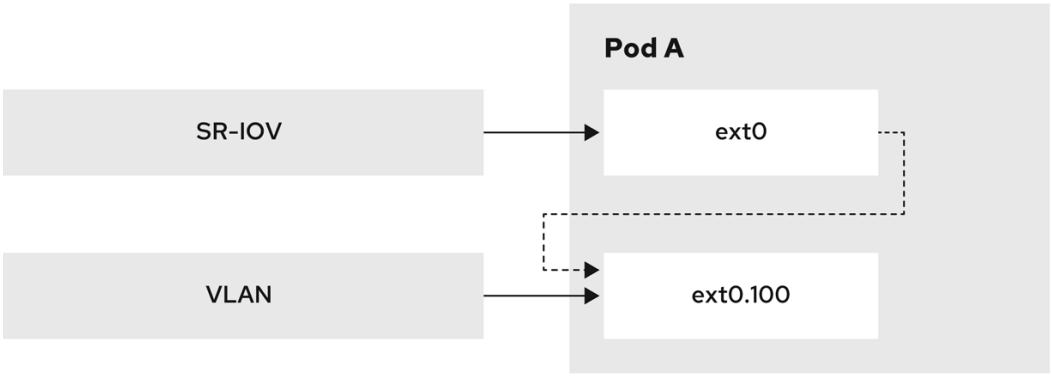
コンテナー namespace マスターインターフェイスを確実に使用するには、追加ネットワークの特定の種類に応じて、VLAN、MACVLAN、または IPVLAN プラグイン設定で **linkInContainer** を指定し、値を **true** に設定します。

26.2.7.1. SR-IOV VF 上で複数の VLAN を作成する

この機能を利用するユースケースの例として、SR-IOV VF に基づいて複数の VLAN を作成することが挙げられます。これを行うには、まず SR-IOV ネットワークを作成し、次に VLAN インターフェイスのネットワーク割り当てを定義します。

次の例は、この図に示されているセットアップを設定する方法を示しています。

図26.1 VLAN の作成



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。

手順

1. 次のコマンドを使用して、Pod をデプロイする専用のコンテナー namespace を作成します。

```
$ oc new-project test-namespace
```

2. SR-IOV ノードポリシーを作成します。
   a. **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **sriov-node-network-policy.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: sriovnic
 namespace: openshift-sriov-network-operator
spec:
 deviceType: netdevice
 isRdma: false
 needVhostNet: true
 nicSelector:
   vendor: "15b3" 1
   deviceID: "101b" 2
   rootDevices: ["00:05.0"]
 numVfs: 10
 priority: 99
 resourceName: sriovnic
 nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

注記

deviceType: netdevice を設定した SR-IOV ネットワークノードポリシーの設定例は、Mellanox ネットワークインターフェイスカード (NIC) 向けに特別に調整されています。

1. SR-IOV ネットワークデバイスのベンダーの 16 進数コード。**15b3** の値は Mellanox NIC に関連付けられています。
2. SR-IOV ネットワークデバイスのデバイスの 16 進数コード。

b. 以下のコマンドを実行して YAML を適用します。

```
$ oc apply -f sriov-node-network-policy.yaml
```

注記

ノードの再起動が必要なため、YAML の適用には時間がかかる場合があります。

3. SR-IOV ネットワークを作成します。

a. 次の CR の例のように、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: sriov-network
 namespace: openshift-sriov-network-operator
spec:
 networkNamespace: test-namespace
 resourceName: sriovnic
 spoofChk: "off"
 trust: "on"
```

b. 以下のコマンドを実行して YAML を適用します。

```
$ oc apply -f sriov-network-attachment.yaml
```

4. VLAN 追加ネットワークを作成します。

a. 以下の YAML の例を使用して、**ipvlan100-additional-network-configuration.yaml** という名前のファイルを作成します。

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: vlan-100
  namespace: test-namespace
spec:
  config: |
```

```
{
  "cniVersion": "0.4.0",
  "name": "vlan-100",
  "plugins": [
    {
      "type": "vlan",
      "master": "ext0", (1)
      "mtu": 1500,
      "vlanId": 100,
      "linkInContainer": true, (2)
      "ipam": {"type": "whereabouts", "ipRanges": [{"range": "1.1.1.0/24"}]}
    }
  ]
}
```

1. VLAN 設定ではマスター名を指定する必要があります。これは Pod ネットワークアノテーションで設定できます。
2. **linkInContainer** パラメーターを指定する必要があります。

b. 以下のコマンドを実行して、YAML ファイルを適用します。

```
$ oc apply -f vlan100-additional-network-configuration.yaml
```

5. 前に指定したネットワークを使用して、Pod 定義を作成します。

a. 次の YAML の例を使用して、**pod-a.yaml** という名前のファイルを作成します。

注記

以下のマニフェストには 2 つのリソースが含まれています。

- セキュリティーラベルのある namespace
- 適切なネットワークアノテーションを含む Pod 定義

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
```

```
        "name": "sriov-network",
        "namespace": "test-namespace",
        "interface": "ext0" 1
      },
      {
        "name": "vlan-100",
        "namespace": "test-namespace",
        "interface": "ext0.100"
      }
    ]'
spec:
  securityContext:
    runAsNonRoot: true
  containers:
    - name: nginx-container
      image: nginxinc/nginx-unprivileged:latest
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: ["ALL"]
      ports:
        - containerPort: 80
      seccompProfile:
        type: "RuntimeDefault"
```

1 VLAN インターフェイスのマスターとして使用される名前。

b. 以下のコマンドを実行して、YAML ファイルを適用します。

```
$ oc apply -f pod-a.yaml
```

6. 次のコマンドを実行して、**test-namespace** 内の **nginx-pod** に関する詳細情報を取得します。

```
$ oc describe pods nginx-pod -n test-namespace
```

出力例

```
Name:         nginx-pod
Namespace:    test-namespace
Priority:     0
Node:         worker-1/10.46.186.105
Start Time:   Mon, 14 Aug 2023 16:23:13 -0400
Labels:       <none>
Annotations:  k8s.ovn.org/pod-networks:
                {"default":{"ip_addresses":
["10.131.0.26/23"],"mac_address":"0a:58:0a:83:00:1a","gateway_ips":["10.131.0.1"],"routes":
[{"dest":"10.128.0.0...
              k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "ovn-kubernetes",
                    "interface": "eth0",
                    "ips": [
                        "10.131.0.26"
                    ],
```

```
                "mac": "0a:58:0a:83:00:1a",
                "default": true,
                "dns": {}
            },{
                "name": "test-namespace/sriov-network",
                "interface": "ext0",
                "mac": "6e:a7:5e:3f:49:1b",
                "dns": {},
                "device-info": {
                    "type": "pci",
                    "version": "1.0.0",
                    "pci": {
                        "pci-address": "0000:d8:00.2"
                    }
                }
            },{
                "name": "test-namespace/vlan-100",
                "interface": "ext0.100",
                "ips": [
                    "1.1.1.1"
                ],
                "mac": "6e:a7:5e:3f:49:1b",
                "dns": {}
            }]
        k8s.v1.cni.cncf.io/networks:
          [ { "name": "sriov-network", "namespace": "test-namespace", "interface": "ext0" }, {
"name": "vlan-100", "namespace": "test-namespace", "i...
        openshift.io/scc: privileged
Status:       Running
IP:           10.131.0.26
IPs:
  IP:  10.131.0.26
```

26.2.7.2. コンテナー namespace のブリッジマスターインターフェイスをベースにしてサブインターフェイスを作成する

サブインターフェイスの作成は、他のタイプのインターフェイスに適用できます。コンテナー namespace のブリッジマスターインターフェイスをベースとしてサブインターフェイスを作成するには、次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 OpenShift Container Platform クラスターにログインしている。

手順

1. 次のコマンドを実行して、Pod のデプロイ先となる専用のコンテナー namespace を作成します。

```
$ oc new-project test-namespace
```

2. 以下の YAML の例を使用して、**bridge-nad.yaml** という名前のブリッジ **NetworkAttachmentDefinition** カスタムリソース (CR) ファイルを作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
spec:
  config: '{
    "cniVersion": "0.4.0",
    "name": "bridge-network",
    "type": "bridge",
    "bridge": "br-001",
    "isGateway": true,
    "ipMasq": true,
    "hairpinMode": true,
    "ipam": {
      "type": "host-local",
      "subnet": "10.0.0.0/24",
      "routes": [{"dst": "0.0.0.0/0"}]
    }
  }'
```

3. 以下のコマンドを実行して、**NetworkAttachmentDefinition** CR を OpenShift Container Platform クラスターに適用します。

```
$ oc apply -f bridge-nad.yaml
```

4. 以下のコマンドを実行して、**NetworkAttachmentDefinition** CR が正常に作成されていることを確認します。

```
$ oc get network-attachment-definitions
```

出力例

```
NAME            AGE
bridge-network   15s
```

5. 以下の YAML の例を使用して、追加の IPVLAN ネットワーク設定用に **ipvlan-additional-network-configuration.yaml** という名前のファイルを作成します。

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: ipvlan-net
  namespace: test-namespace
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "ipvlan-net",
    "type": "ipvlan",
    "master": "ext0", 1
    "mode": "l3",
```

```
    "linkInContainer": true, 2
    "ipam": {"type": "whereabouts", "ipRanges": [{"range": "10.0.0.0/24"}]}
  }'
```

1 ネットワーク接続に関連付けるイーサネットインターフェイスを指定します。これはその後、Pod ネットワークアノテーションで設定されます。

2 マスターインターフェイスがコンテナーネットワーク namespace にあることを指定します。

6. 以下のコマンドを実行して、YAML ファイルを適用します。

```
$ oc apply -f ipvlan-additional-network-configuration.yaml
```

7. 以下のコマンドを実行して、**NetworkAttachmentDefinition** CR が正常に作成されていることを確認します。

```
$ oc get network-attachment-definitions
```

出力例

```
NAME             AGE
bridge-network   87s
ipvlan-net       9s
```

8. 以下の YAML の例を使用して、Pod 定義用に **pod-a.yaml** という名前のファイルを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-a
  namespace: test-namespace
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {
        "name": "bridge-network",
        "interface": "ext0" 1
      },
      {
        "name": "ipvlan-net",
        "interface": "ext1"
      }
    ]'
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: test-pod
    image: quay.io/openshifttest/hello-
sdn@sha256:c89445416459e7adea9a5a416b3365ed3d74f2491beb904d61dc8d1eb89a72a4
```

```
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

1 IPVLAN インターフェイスのマスターとして使用する名前を指定します。

9. 以下のコマンドを実行して、YAML ファイルを適用します。

```
$ oc apply -f pod-a.yaml
```

10. 以下のコマンドを使用して、Pod が実行されていることを確認します。

```
$ oc get pod -n test-namespace
```

出力例

```
NAME    READY   STATUS    RESTARTS   AGE
pod-a   1/1     Running   0          2m36s
```

11. 次のコマンドを実行して、**test-namespace** 内の **pod-a** リソースに関するネットワークインターフェイス情報を表示します。

```
$ oc exec -n test-namespace pod-a -- ip a
```

出力例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if105: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1400 qdisc noqueue
state UP group default
    link/ether 0a:58:0a:d9:00:5d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.217.0.93/23 brd 10.217.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::488b:91ff:fe84:a94b/64 scope link
       valid_lft forever preferred_lft forever
4: ext0@if107: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.0.0.2/24 brd 10.0.0.255 scope global ext0
       valid_lft forever preferred_lft forever
    inet6 fe80::bcda:bdff:fe7e:f437/64 scope link
       valid_lft forever preferred_lft forever
5: ext1@ext0: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UNKNOWN group default
    link/ether be:da:bd:7e:f4:37 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 brd 10.0.0.255 scope global ext1
```

```
    valid_lft forever preferred_lft forever
inet6 fe80::beda:bd00:17e:f437/64 scope link
    valid_lft forever preferred_lft forever
```

この出力は、ネットワークインターフェイス **ext1** が物理インターフェイス **ext0** に関連付けられていることを示しています。

26.3. 仮想ルーティングおよび転送について

26.3.1. 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パケットは、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。

26.3.1.1. 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。

26.4. 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用にマルチネットワークを設定できます。SR-IOV、macvlan、および OVN-Kubernetes の追加ネットワークに対してマルチネットワークポリシーを指定できます。Macvlan 追加ネットワークは完全にサポートされています。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。

重要

SR-IOV 追加ネットワークのマルチネットワークポリシー設定のサポートはテクノロジープレビュー機能であり、カーネルネットワークインターフェイスカード (NIC) でのみサポートされます。SR-IOV は、データプレーン開発キット (DPDK) アプリケーションで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。

26.4.1. 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 または SR-IOV 追加ネットワークを定義するネットワーク割り当て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>
ネットワーク割り当て定義の名前を指定します。

26.4.2. 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
```

2. 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
```

26.4.3. IPv6 ネットワークでのマルチネットワークポリシーのサポート

ICMPv6 近隣探索プロトコル (NDP) は、デバイスが近隣ノードに関する情報を検出して維持できるようにするための、メッセージとプロセスのセットです。NDP は IPv6 ネットワークで重要な役割を果たし、同じリンク上のデバイス間の対話を促進します。

useMultiNetworkPolicy パラメーターが **true** に設定されている場合、Cluster Network Operator (CNO) はマルチネットワークポリシーの iptables 実装をデプロイします。

IPv6 ネットワークでマルチネットワークポリシーをサポートするために、Cluster Network Operator は、マルチネットワークポリシーの影響を受けるすべての Pod に次のルールのセットをデプロイします。

マルチネットワークポリシーのカスタムルール

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: multi-networkpolicy-custom-rules
  namespace: openshift-multus
data:

  custom-v6-rules.txt: |
    # accept NDP
    -p icmpv6 --icmpv6-type neighbor-solicitation -j ACCEPT 1
    -p icmpv6 --icmpv6-type neighbor-advertisement -j ACCEPT 2
    # accept RA/RS
    -p icmpv6 --icmpv6-type router-solicitation -j ACCEPT 3
    -p icmpv6 --icmpv6-type router-advertisement -j ACCEPT 4
```

1. このルールは、近隣探索プロトコル (NDP) の一部である ICMPv6 ネイバー要請メッセージの受信を許可します。これらのメッセージは、近隣ノードのリンクレイヤーアドレスを決定するのに役立ちます。
2. このルールは、NDP の一部であり、送信者のリンクレイヤーアドレスに関する情報を提供する、受信 ICMPv6 近隣アドバタイズメントメッセージを許可します。
3. このルールは、ICMPv6 ルーター要請メッセージの受信を許可します。ホストは、これらのメッセージを使用してルーター設定情報を要求します。
4. このルールは、ホストに設定情報を提供する ICMPv6 ルーターアドバタイズメントメッセージの受信を許可します。

注記

これらの事前定義されたルールは編集できません。

これらのルールが集合することで、IPv6 環境でのアドレス解決やルーター通信などを含め、ネットワークが正しく機能するために不可欠な ICMPv6 トラフィックが有効になります。これらのルールが適用さ

れ、トラフィックを拒否するマルチネットワークポリシーがあれば、アプリケーションで接続の問題が発生することは考えられません。

26.4.4. 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。

26.4.4.1. 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。

26.4.4.2. CLI を使用した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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。
- 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

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>
      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。

      これは基本的なポリシーであり、他のネットワークポリシーの設定によって許可されたクロス Pod トラフィック以外のすべてのクロス Pod ネットワーキングをブロックします。

      ```
      apiVersion: k8s.cni.cncf.io/v1beta1
      kind: MultiNetworkPolicy
      metadata:
      ```

```
  name: deny-by-default
  annotations:
    k8s.v1.cni.cncf.io/policy-for:<namespace_name>/<network_name>
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  ingress: []
```

ここでは、以下のようになります。

<network_name>

ネットワーク割り当て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>

ネットワーク割り当て定義の名前を指定します。

特定のnamespaceから 1つの Pod への上りトラフィックを許可する

このポリシーは、**namespace-y** で実行されている Pod から **pod-a** というラベルの付いた Pod へのトラフィックを許可します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: allow-traffic-pod
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
   matchLabels:
      pod: pod-a
  policyTypes:
  - Ingress
  ingress:
  - from:
```

```
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: namespace-y
```

ここでは、以下のようになります。

<network_name>
ネットワーク割り当て定義の名前を指定します。

サービスへのトラフィックを制限する

このポリシーを適用すると、**app=bookstore** と **role=api** の両方のラベルを持つすべての Pod に、**app=bookstore** というラベルを持つ Pod のみがアクセスできるようになります。この例では、アプリケーションは、ラベル **app=bookstore** および **role=api** でマークされた REST API サーバーである可能性があります。

この例では、次のユースケースに対応します。

- サービスへのトラフィックを、それを使用する必要がある他のマイクロサービスのみに制限します。
- データベースへの接続を制限して、それを使用するアプリケーションのみを許可します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: api-allow
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: bookstore
      role: api
  ingress:
  - from:
      - podSelector:
          matchLabels:
            app: bookstore
```

ここでは、以下のようになります。

<network_name>
ネットワーク割り当て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。

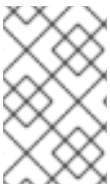
<policy_name>
マルチネットワークポリシーのファイル名を指定します。

<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/deny-by-default created
```



注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールのフォームから、クラスターの任意の namespace でネットワークポリシーを直接作成できます。

26.4.4.3. 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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。
- 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リスト表示するには、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. 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。

- 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。

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールの **Actions** メニューのポリシーから、クラスターの任意の namespace でネットワークポリシーを直接編集できます。

26.4.4.4. CLI を使用した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。
- 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リスト表示します。

- 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。

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールのフォームから、クラスターの任意の namespace でネットワークポリシーを直接表示できます。

26.4.4.5. CLI を使用した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

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。
- 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

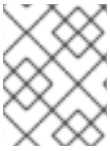
```
multinetworkpolicy.k8s.cni.cncf.io/default-deny deleted
```

注記

cluster-admin 権限で Web コンソールにログインする場合、YAML で、または Web コンソールの **Actions** メニューのポリシーから、クラスターの任意の namespace でネットワークポリシーを直接削除できます。

26.4.4.6. デフォルトのすべてのマルチネットワーク拒否ポリシーの作成

これは基本的なポリシーであり、他のデプロイメントされたネットワークポリシーの設定によって許可されたネットワークトラフィック以外のすべてのクロス Pod ネットワークをブロックします。この手順では、デフォルトの **deny-by-default** ポリシーを適用します。



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ネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。
- 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

1. すべての namespace におけるすべての Pod からの ingress を拒否する **deny-by-default** ポリシーを定義する次の YAML を作成します。YAML を **deny-by-default.yaml** ファイルに保存します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: deny-by-default
  namespace: default 1
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <namespace_name>/<network_name> 2
spec:
  podSelector: {} 3
```

```
policyTypes: 4
- Ingress 5
ingress: [] 6
```

1. **namespace: default** は、このポリシーを **default** namespace にデプロイします。
2. **network_name**: ネットワーク割り当て定義の名前を指定します。
3. **podSelector:** は空です。これは、すべての Pod に一致することを意味します。したがって、ポリシーはデフォルトnamespaceのすべての Pod に適用されます。
4. **policyTypes: NetworkPolicy** が関連するルールタイプのリスト。
5. **Ingress** のみの **policyType** として指定します。
6. 指定された **ingress** ルールはありません。これにより、着信トラフィックがすべての Pod にドロップされます。

2. 次のコマンドを入力して、ポリシーを適用します。

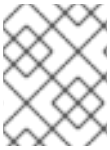
```
$ oc apply -f deny-by-default.yaml
```

出力例

```
multinetworkpolicy.k8s.cni.cncf.io/deny-by-default created
```

26.4.4.7. 外部クライアントからのトラフィックを許可するマルチネットワークポリシーの作成

deny-by-default ポリシーを設定すると、外部クライアントからラベル **app=web** を持つ Pod へのトラフィックを許可するポリシーの設定に進むことができます。



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。

この手順に従って、パブリックインターネットから直接、またはロードバランサーを使用して Pod にアクセスすることにより、外部サービスを許可するポリシーを設定します。トラフィックは、ラベル **app=web** を持つ Pod にのみ許可されます。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。
- 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

1. パブリックインターネットからのトラフィックが直接、またはロードバランサーを使用して Pod にアクセスできるようにするポリシーを作成します。YAML を **web-allow-external.yaml** ファイルに保存します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-external
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: web
  ingress:
    - {}
```

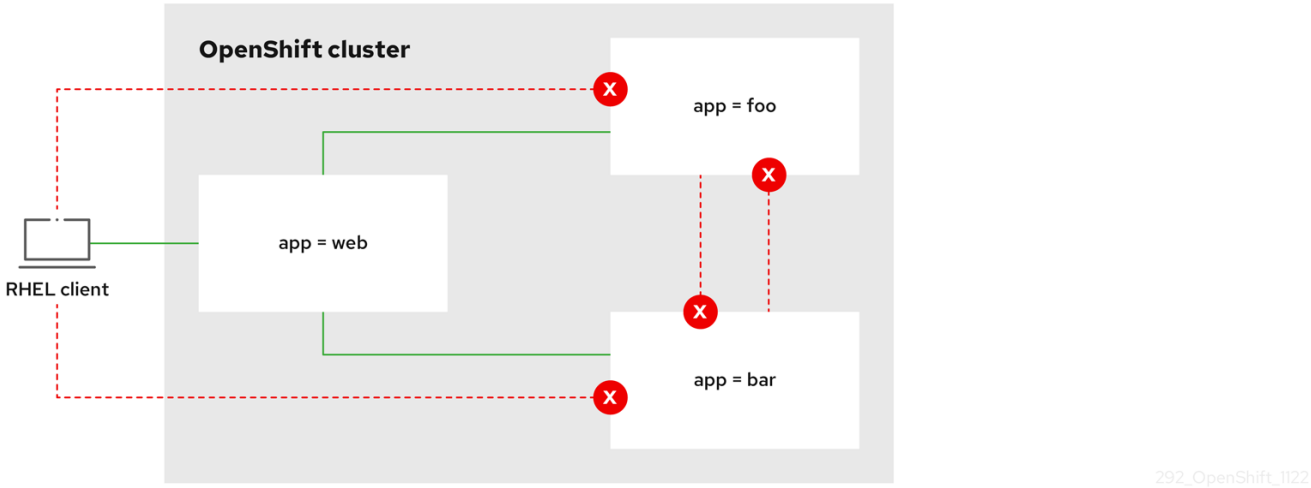
2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-external created
```

このポリシーは、次の図に示すように、外部トラフィックを含むすべてのリソースからのトラフィックを許可します。



26.4.4.8. すべてのnamespaceからアプリケーションへのトラフィックを許可するマルチネットワークポリシーの作成

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。

この手順に従って、すべてのnamespace内のすべての Pod から特定のアプリケーションへのトラフィックを許可するポリシーを設定します。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。
- 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

1. すべてのnamespaceのすべての Pod から特定のアプリケーションへのトラフィックを許可するポリシーを作成します。YAML を **web-allow-all-namespaces.yaml** ファイルに保存します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-all-namespaces
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: web (1)
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector: {} (2)
```

(1) デフォルトの namespace の **app:web** Pod にのみポリシーを適用します。

(2) すべてのnamespaceのすべての Pod を選択します。

注記

デフォルトでは、**namespaceSelector** の指定を省略した場合、namespace は選択されません。つまり、ポリシーは、ネットワークポリシーがデプロイされている namespace からのトラフィックのみを許可します。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-all-namespaces created
```

検証

1. 次のコマンドを入力して、**default** namespace で Web サービスを開始します。

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

2. 次のコマンドを実行して、**alpine** イメージを **secondary** namespace にデプロイし、シェルを開始します。

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

3. シェルで次のコマンドを実行し、リクエストが許可されていることを確認します。

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

予想される出力

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

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

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

26.4.4.9. namespaceからアプリケーションへのトラフィックを許可するマルチネットワークポリシーの作成

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。

特定の namespace からラベル **app=web** を持つ Pod へのトラフィックを許可するポリシーを設定するには、次の手順に従います。以下の場合にこれを行うことができます。

- 運用データベースへのトラフィックを、運用ワークロードがデプロイされているnamespaceのみに制限します。
- 特定のnamespaceにデプロイされた監視ツールを有効にして、現在のnamespaceからメトリックをスクレイピングします。

前提条件

- クラスターは、**NetworkPolicy** オブジェクトをサポートするネットワークプラグイン (**mode: NetworkPolicy** が設定された OVN-Kubernetes ネットワークプラグインや OpenShift SDN ネットワークプラグインなど) を使用します。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。
- 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

1. ラベルが **purpose=production** の特定の namespace 内にあるすべての Pod からのトラフィックを許可するポリシーを作成します。YAML を **web-allow-prod.yaml** ファイルに保存します。

```
apiVersion: k8s.cni.cncf.io/v1beta1
kind: MultiNetworkPolicy
metadata:
  name: web-allow-prod
  namespace: default
  annotations:
    k8s.v1.cni.cncf.io/policy-for: <network_name>
spec:
  podSelector:
    matchLabels:
      app: web (1)
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: production (2)
```

(1) デフォルトの namespace の **app:web** Pod にのみポリシーを適用します。

(2) ラベルが **purpose=production** の namespace 内にある Pod のみにトラフィックを制限します。

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

```
multinetworkpolicy.k8s.cni.cncf.io/web-allow-prod created
```

検証

1. 次のコマンドを入力して、**default** namespace で Web サービスを開始します。

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

2. 次のコマンドを実行して、**prod** namespace を作成します。

```
$ oc create namespace prod
```

3. 次のコマンドを実行して、**prod** namespace にラベルを付けます。

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

4. 次のコマンドを実行して、**dev** namespace を作成します。

```
$ oc create namespace dev
```

5. 次のコマンドを実行して、**dev** namespace にラベルを付けます。

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

6. 次のコマンドを実行して、**alpine** イメージを **dev** namespace にデプロイし、シェルを開始します。

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

7. シェルで次のコマンドを実行し、リクエストがブロックされていることを確認します。

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

予想される出力

```
wget: download timed out
```

8. 次のコマンドを実行して、**alpine** イメージを **prod** namespace にデプロイし、シェルを開始します。

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

9. シェルで次のコマンドを実行し、リクエストが許可されていることを確認します。

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

予想される出力

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

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

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

26.4.5. 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
- SR-IOV ネットワークデバイスの設定

26.5. 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。

26.5.1. 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ることはできません。

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
   ```

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/network-status: |- 1
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
      }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

1 **k8s.v1.cni.cncf.io/network-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。

26.5.1.1. 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ー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。
- クラスター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、追加のプロパティー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リストの 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。

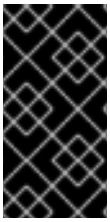
3. 以下の例では、アノテーションで **default-route** パラメーターを使用して、デフォルトルートを持つネットワーク割り当てを指定します。

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
    {
      "name": "net1"
    },
    {
      "name": "net2", 1
      "default-route": ["192.0.2.1"] 2
    }]'
spec:
  containers:
  - name: example-pod
    command: ["/bin/bash", "-c", "sleep 2000000000000"]
    image: centos/tools
```

1 **name** キーは、Pod に関連付ける追加ネットワークの名前です。

2 **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ーを確認する場合、**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/network-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 rawCNIConfig 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> 1
namespace: <namespace> 2
rawCNIConfig: '{ 3
  ...
}'
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 **rawCNIConfig** 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 プロパティーを検証するには、**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
```

26.6. 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。

26.6.1. 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。

26.7. 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。

26.7.1. 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。

1. 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
```

2. **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。
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、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"]}} }
```

26.8. 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。

26.8.1. 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
```

26.9. 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、仮想ルーティングおよび転送 (VRF) ドメインの追加ネットワークを設定できます。このプラグインが作成する仮想ネットワークは、指定した物理インターフェイスに関連付けられます。

VRF インスタンスでセカンダリーネットワークを使用すると、次の利点があります。

ワークロードの分離

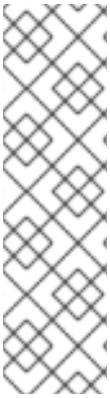
追加のネットワークの VRF インスタンスを設定して、ワークロードトラフィックを分離します。

セキュリティーの向上

VRF ドメイン内の分離されたネットワークパスを通じて、セキュリティーを向上させます。

マルチテナンシーのサポート

各テナントの VRF ドメイン内で、一意のルーティングテーブルを使用したネットワークセグメンテーションを通じて、マルチテナントをサポートします。



注記

VRF を使用するアプリケーションは、特定のデバイスに対して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** オプションは、渡されたインターフェイス名 (例: **eth1**) で指定されたデバイスにソケットをバインドします。**SO_BINDTODEVICE** オプションを使用するには、アプリケーションに **CAP_NET_RAW** 機能が必要です。

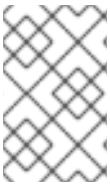
ip vrf exec コマンドを使用した VRF の使用は、OpenShift Container Platform Pod ではサポートされません。VRF を使用するには、アプリケーションを VRF インターフェイスに直接バインドします。

関連情報

- 仮想ルーティングおよび転送について

26.9.1. 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": [ 1
    {
      "type": "macvlan",
      "master": "eth1",
      "ipam": {
         "type": "static",
         "addresses": [
         {
            "address": "191.168.1.23/24"
         }
         ]
      }
    },
    {
      "type": "vrf", 2
      "vrfname": "vrf-1", 3
      "table": 1001 4
    }]
  }'
```

1. **plugins** は一覧である必要があります。リストの最初の項目は、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。
2. **type** は **vrf** に設定する必要があります。
3. **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。
4. 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。

検証

1. Pod を作成し、VRF インスタンスを使用して追加のネットワークに割り当てます。
   a. **Pod** リソースを定義する YAML ファイルを作成します。

      pod-additional-net.yam ファイルの例

```
apiVersion: v1
kind: Pod
metadata:
 name: pod-additional-net
 annotations:
   k8s.v1.cni.cncf.io/networks: '[
      {
              "name": "test-network-1" 1
      }
 ]'
spec:
 containers:
 - name: example-pod-1
   command: ["/bin/bash", "-c", "sleep 9000000"]
   image: centos:8
```

      1 VRF インスタンスを使用する追加ネットワークの名前を指定します。

   b. 次のコマンドを実行して、**Pod** リソースを作成します。

```
$ oc create -f pod-additional-net.yaml
```

      出力例

```
pod/test-pod created
```

2. 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 とのリモートセッションを開始し、次のコマンドを実行します。

```
$ ip vrf show
```

   出力例

```
Name              Table
-----------------------
vrf-1             1001
```

3. VRF インターフェイスが追加インターフェイスのコントローラ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27章 ハードウェアネットワーク

27.1. 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ーは、VF がコンテナーで公開される方法を判別します。

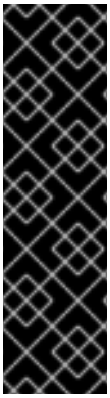
- **netdevice** ドライバー: コンテナーの **netns** 内の通常のカーネルネットワークデバイス
- **vfio-pci** ドライバー: コンテナーにマウントされるキャラクターデバイス

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 クラスター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のあるアプリケーションに使用できます。

SR-IOV ネットワークのマルチネットワークポリシーを設定できます。これのサポートはテクノロジープレビューであり、SR-IOV 追加ネットワークはカーネル NIC でのみサポートされます。データプレーン開発キット (DPDK) アプリケーションではサポートされていません。

注記

SR-IOV ネットワークでマルチネットワークポリシーを作成しても、マルチネットワークポリシーが設定されていない SR-IOV ネットワークと比較して、アプリケーションに同じパフォーマンスが提供されない場合があります。



重要

SR-IOV ネットワークのマルチネットワークポリシーは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

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"
```

27.1.1. 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、クラスターで SR-IOV ネットワークデバイスを検出し、初期化します。

SR-IOV Network 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ーのみに **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 をスケジュールできます。

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。

27.1.1.1. 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)

27.1.1.2. 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27.1 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
Broadcom	BCM57504	14e4	1751
Intel	X710	8086	1572
Intel	X710 Backplane	8086	1581
Intel	X710 Base T	8086	15ff
Intel	XL710	8086	1583
Intel	XXV710	8086	158b
Intel	E810-CQDA2	8086	1592
Intel	E810-2CQDA2	8086	1592
Intel	E810-XXVDA2	8086	159b
Intel	E810-XXVDA4	8086	1593
Intel	E810-XXVDA4T	8086	1593
Mellanox	MT27700 Family [ConnectX-4]	15b3	1013
Mellanox	MT27710 Family [ConnectX-4 Lx]	15b3	1015
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT28908 Family [ConnectX-6]	15b3	101b
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 Family [ConnectX-6 Lx]	15b3	101f

製造元	モデル	ベンダー ID	デバイス ID
Mellanox	Mellanox MT2910 ファミリー [ConnectX-7]	15b3	1021
Mellanox	ConnectX-6 NIC モードの MT42822 BlueField-2	15b3	a2d6
Pensando [1]	Ionic ドライバー用 DSC-25 デュアルポート 25G 分散サービスカード	0x1dd8	0x1002
Pensando [1]	Ionic ドライバー用 DSC-100 デュアルポート 100G 分散サービスカード	0x1dd8	0x1003
Silicom	STS ファミリー	8086	1591

1. OpenShift SR-IOV はサポートされますが、SR-IOV を使用する際に SR-IOV CNI config ファイルを使用して静的な Virtual Function (VF) メディアアクセス制御 (MAC) アドレスを設定する必要があります。

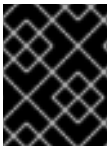
注記

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。

27.1.1.3. 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** リストは、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。

27.1.1.3.1. 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 1
```

```
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: 2
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f1
    pciAddress: 0000:81:00.1
    totalvfs: 64
    vendor: "8086"
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens803f0
    pciAddress: 0000:86:00.0
    totalvfs: 64
    vendor: "8086"
  syncStatus: Succeeded
```

1 **name** フィールドの値はワーカーノードの名前と同じです。

2 **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 デバイスのリストが含まれます。

27.1.1.4. 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
```

27.1.1.5. コンテナー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ーから 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) のコンテナー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、コンテナーで利用可能な CPU を判別し、リストを返します。

GetHugepages()

この機能は、各コンテナーの **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、コンテナー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リストを返します。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ーには、コンテナーイメージ **dpdk-app-centos** をビルドするためのサンプル Dockerfile が含まれます。コンテナーイメージは、Pod 仕様の環境変数に応じて、**l2fwd**、**l3wd** 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。コンテナーイメージは、**app-netutil** ライブラリーをコンテナー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 コンテナーに統合することもできます。init コンテナー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。

27.1.1.6. 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 情報をコンテナーに提供します。

Network Resources Injector は、**podnetinfo** という名前のボリュームを追加し、Pod の各コンテナー用に **/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>**

直前のリスト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リスト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。

27.1.2. 関連情報

- マルチネットワークポリシーの設定

27.1.3. 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

27.2. 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。

27.2.1. 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、Single Root I/O Virtualization (SR-IOV) Network Operator をインストールできます。

27.2.1.1. 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 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: stable
  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.4.15.0-202310121402   Succeeded
```

27.2.1.2. 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** プロジェクトにリスト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。

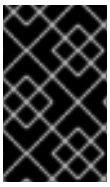
27.2.2. 次のステップ

- オプション: SR-IOV Network Operator の設定

27.3. 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。

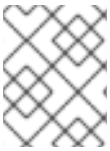
27.3.1. SR-IOV Network Operator の設定



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。

27.3.1.1. 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27.2 SR-IOV Network Operator config カスタムリソース

フィールド	型	説明
metadata.name	**string**	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。
metadata.namespace	**string**	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。

フィールド	型	説明
spec.configDaemonNodeSelector	**string**	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 デーモンセットを配置します。
spec.disableDrain	**boolean**	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。
spec.enableInjector	**boolean**	Network Resources Injector デー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。
spec.enableOperatorWebhook	**boolean**	Operator Admission Controller の Webhook デー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。
spec.logLevel	**integer**	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。

27.3.1.2. 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ーは、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

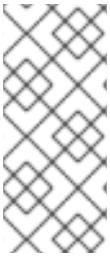
```
NAME                                  READY   STATUS    RESTARTS   AGE
network-resources-injector-5cz5p      1/1     Running   0          10m
network-resources-injector-dwqpx      1/1     Running   0          10m
network-resources-injector-lktz5      1/1     Running   0          10m
```

27.3.1.3. SR-IOV Network Operator Admission Controller Webhook について

SR-IOV Network Operator Admission Controller Webbook 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。サポート対象外のデバイスの設定については、サポート対象外の NIC を使用するための SR-IOV Network Operator の設定 を参照してください。

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
```

27.3.1.4. カスタムノードセレクターについて

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。

27.3.1.5. 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、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、**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>
```

27.3.1.6. 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、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

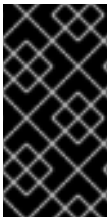
または、以下の YAML を適用して Operator を更新することもできます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovOperatorConfig
metadata:
  name: default
  namespace: openshift-sriov-network-operator
spec:
  enableOperatorWebhook: <value>
```

27.3.1.7. 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>
```

27.3.1.8. 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
```

27.3.1.9. Hosted Control Plane 用の SR-IOV Operator のデプロイ

重要

AWS プラットフォーム上の Hosted Control Plane は、テクノロジープレビュー機能としてのみ利用できます。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

ホスティングサービスクラスターを設定してデプロイすると、ホストされたクラスターで SR-IOV Operator へのサブスクリプションを作成できます。SR-IOV Pod は、コントロールプレーンではなくワーカーマシンで実行されます。

前提条件

AWS 上でホストされたクラスターを設定およびデプロイしている。詳細は、AWS 上でのホストクラスターの設定 (テクノロジープレビュー) を参照してください。

手順

1. namespace と Operator グループを作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sriov-network-operator
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: sriov-network-operators
  namespace: openshift-sriov-network-operator
spec:
  targetNamespaces:
  - openshift-sriov-network-operator
```

2. SR-IOV Operator へのサブスクリプションを作成します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: sriov-network-operator-subsription
  namespace: openshift-sriov-network-operator
spec:
  channel: stable
  name: sriov-network-operator
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
  source: s/qe-app-registry/redhat-operators
  sourceNamespace: openshift-marketplace
```

検証

1. SR-IOV Operator の準備ができていることを確認するには、次のコマンドを実行し、結果の出力を表示します。

```
$ oc get csv -n openshift-sriov-network-operator
```

出力例

```
NAME                                         DISPLAY                   VERSION               REPLACES
PHASE
sriov-network-operator.4.15.0-202211021237   SR-IOV Network Operator   4.15.0-
202211021237   sriov-network-operator.4.15.0-202210290517   Succeeded
```

2. SR-IOV Pod がデプロイされていることを確認するには、次のコマンドを実行します。

```
$ oc get pods -n openshift-sriov-network-operator
```

27.3.2. 次のステップ

- SR-IOV ネットワークデバイスの設定

27.4. 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。

27.4.1. 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
  externallyManaged: false 9
  nicSelector: 10
    vendor: "<vendor_code>" 11
    deviceID: "<device_id>" 12
    pfNames: ["<pf_name>", ...] 13
```

```
  rootDevices: ["<pci_bus_id>", ...] 14
  netFilter: "<filter_string>" 15
deviceType: <device_type> 16
isRdma: false 17
linkType: <link_type> 18
eSwitchMode: "switchdev" 19
excludeTopology: false 20
```

1. 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。
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。

   名前を指定するときは、**resourceName** で使用できる構文式 **^a-zA-Z0-9_+$** を必ず使用してください。
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。

   重要

   SR-IOV Network Operator は、ノードネットワーク設定ポリシーを順番にノードに適用します。ノードネットワーク設定ポリシーを適用する前に、SR-IOV Network Operator は、ノードのマシン設定プール (MCP) が **Degraded** または **Updating** などの正常ではない状態にないか確認します。ノード正常ではない MCP にある場合、ノードネットワーク設定ポリシーをクラスター内のすべての対象ノードに適用するプロセスは、MCP が正常な状態に戻るまで一時停止します。

   正常でない MCP 内にあるノードが、他のノード (他の MCP 内のノードを含む) にノードネットワーク設定ポリシーを適用することを阻害しないようにするには、MCP ごとに別のノードネットワーク設定ポリシーを作成する必要があります。
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. **externallyManaged** を **true** に設定すると、SR-IOV Network Operator が外部管理の Virtual Function (VF) のすべてまたは一部を使用し、それらを Pod に割り当てることができるようになります。値を **false** に設定すると、SR-IOV Network Operator は割り当てられたすべての VF を管理および設定します。

注記

externallyManaged を **true** に設定した場合、ポリシーを適用する前に Virtual Function (VF) を作成する必要があります。そうでない場合、Webhook がリクエストをブロックします。**externallyManaged** を **false** に設定した場合、SR-IOV Network Operator が、必要に応じて VF をリセットするなど、VF の作成と管理を処理します。したがって、ホストシステム上で VF を使用するには、VF を手動で作成する必要があり、**nicSelector** ポリシーで定義されていない PF **および** VF に対して SR-IOV Network Operator がアクションを実行しないように、externallyManaging を **true** に設定する必要があります。

10 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て正確に特定することが推奨されます。

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。

11 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。

12 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。

13 オプション: 1 つ以上のデバイスの物理機能 (PF) 名の配列。

14 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**

15 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。

16 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 **vfio-pci** 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。

17 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。

18 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、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 **vfio-pci** 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。

19 オプション: ハードウェアオフロードを有効にするには、**eSwitchMode** フィールドを **"switchdev"** に設定する必要があります。

20 オプション: SR-IOV ネットワークリソースの NUMA ノードを Topology Manager にアドバタイスする場合を除外するには、値を **true** に設定します。デフォルト値は **false** です。

27.4.1.1. 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 1
  nicSelector:
```

```
vendor: "15b3"
deviceID: "101b"
netFilter: "openstack/NetworkID:ea24bd04-8674-4f69-b0ee-fa0b3bd20509" (2)
```

(1) 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。

(2) **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す。

27.4.1.2. 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 **vfio-pci** 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、**vfio** VF ドライバーと共に PF **netpf0** の VF **8** から **15** までが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
```

インターフェイスが正常にパーティショニングされていることを確認します

次のコマンドを実行して、インターフェイスが SR-IOV デバイスの Virtual Function (VF) にパーティショニングされていることを確認します。

```
$ ip link show <interface> 1
```

1 **<interface>** を、SR-IOV デバイスの VF にパーティショニングするときに指定したインターフェイス (例: **ens3f1**) に置き換えます。

出力例

```
5: ens3f1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
link/ether 3c:fd:fe:d1:bc:01 brd ff:ff:ff:ff:ff:ff

vf 0     link/ether 5a:e7:88:25:ea:a0 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 1     link/ether 3e:1d:36:d7:3d:49 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 2     link/ether ce:09:56:97:df:f9 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 3     link/ether 5e:91:cf:88:d1:38 brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
vf 4     link/ether e6:06:a1:96:2f:de brd ff:ff:ff:ff:ff:ff, spoof checking on, link-state auto, trust off
```

27.4.2. 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

27.4.3. 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"
```

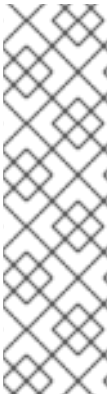
ノードがメモリーを割り当てることができないことを示す場合は、以下の項目を確認します。

- 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。

27.4.4. 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。

27.4.4.1. 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: intelnics
  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

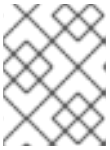
- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。

```
$ oc get network-attachment-definitions -n <namespace> (1)
```

1 **<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。

1. VRF CNI を使用する SR-IOV 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 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
```

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
...
```

27.4.5. NUMA 対応スケジューリング用の SR-IOV ネットワークトポロジーを除外する場合

NUMA 対応 Pod のスケジューリングでより柔軟な SR-IOV ネットワークデプロイメントを実現するために、SR-IOV ネットワークの Non-Uniform Memory Access (NUMA) ノードを Topology Manager にアドバタイズする場合を除外できます。

一部のシナリオでは、シングル NUMA ノード上の Pod の CPU およびメモリーリソースを最大化する

ことが優先されます。Topology Manager に Pod の SR-IOV ネットワークリソースの NUMA ノードに関するヒントを提供しないことで、Topology Manager は SR-IOV ネットワークリソースと Pod の CPU およびメモリーリソースを異なる NUMA ノードにデプロイできます。その場合、NUMA ノード間のデータ転送により、ネットワーク遅延が増加する可能性があります。ただし、ワークロードが最適な CPU とメモリーのパフォーマンスを必要とするシナリオでは許容されます。

たとえば、2 つの NUMA ノード (**uma0** と **uma1)** を備えたコンピュートノード **compute-1** があるとします。SR-IOV が有効な NIC は **numa0** にあります。Pod のスケジューリングに使用できる CPU は、**numa1** にしかありません。**excludeTopology** 仕様を **true** に設定すると、Topology Manager は Pod の CPU およびメモリーリソースを **numa1** に割り当て、同じ Pod の SR-IOV ネットワークリソースを **numa0** に割り当てることができます。これは、**excludeTopology** 仕様を **true** に設定した場合にのみ可能です。そうではない場合、Topology Manager はすべてのリソースを同じ NUMA ノードに配置しようとします。

27.4.5.1. NUMA 対応スケジューリングのための SR-IOV ネットワークトポロジーの除外

SR-IOV ネットワークリソースの Non-Uniform Memory Access (NUMA) ノードを Topology Manager にアドバタイズする場合を除外するには、**SriovNetworkNodePolicy** カスタムリソースで **excludeTopology** 仕様を設定できます。NUMA 対応 Pod のスケジューリングでより柔軟な SR-IOV ネットワークデプロイメントを行うには、この設定を使用します。

前提条件

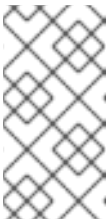
- OpenShift CLI (**oc**) がインストールされている。
- CPU マネージャーのポリシーを **static** に設定している。CPU マネージャーの詳細は、**関連情報** セクションを参照してください。
- Topology Manager ポリシーを **single-numa-node** に設定している。
- SR-IOV Network Operator がインストールされている。

手順

1. **SriovNetworkNodePolicy** CR を作成します。
   a. 次の YAML を **sriov-network-node-policy.yaml** ファイルに保存し、環境に合わせて YAML 内の値を置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <policy_name>
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 (1)
  nodeSelector:
    kubernetes.io/hostname: <node_name>
  numVfs: <number_of_Vfs>
  nicSelector: (2)
    vendor: "<vendor_ID>"
    deviceID: "<device_ID>"
  deviceType: netdevice
  excludeTopology: true (3)
```

1. SR-IOV ネットワークデバイスプラグインのリソース名。この YAML は、サンプルの **resourceName** 値を使用します。
2. NIC セレクターを使用して、Operator が設定するデバイスを特定します。
3. SR-IOV ネットワークリソースの NUMA ノードを Topology Manager にアドバタイスする場合を除外するには、値を **true** に設定します。デフォルト値は **false** です。



注記

複数の **SriovNetworkNodePolicy** リソースが同じ SR-IOV ネットワークリソースをターゲットとする場合、**SriovNetworkNodePolicy** リソースの値は **excludeTopology** 仕様と同じである必要があります。そうでない場合、矛盾するポリシーは拒否されます。

b. 次のコマンドを実行して、**SriovNetworkNodePolicy** リソースを作成します。

```
$ oc create -f sriov-network-node-policy.yaml
```

出力例

```
sriovnetworknodepolicy.sriovnetwork.openshift.io/policy-for-numa-0 created
```

2. **SriovNetwork** CR を作成します。

a. 次の YAML を **sriov-network.yaml** ファイルに保存します。その場合、YAML 内の値は環境に合わせて置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: sriov-numa-0-network 1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: sriovnuma0 2
  networkNamespace: <namespace> 3
  ipam: |- 4
    {
      "type": "<ipam_type>",
    }
```

1. **sriov-uma-0-network** を SR-IOV ネットワークリソースの名前に置き換えます。
2. 前の手順で作成した **SriovNetworkNodePolicy** CR のリソース名を指定します。この YAML は、サンプルの **resourceName** 値を使用します。
3. SR-IOV ネットワークリソースの namespace を入力します。
4. SR-IOV ネットワークの IP アドレス管理設定を入力します。

b. 次のコマンドを実行して、**SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network.yaml
```

出力例

```
sriovnetwork.sriovnetwork.openshift.io/sriov-numa-0-network created
```

3. Pod を作成し、前の手順で作成した SR-IOV ネットワークリソースを割り当てます。

   a. 次の YAML を **sriov-network-pod.yaml** ファイルに保存します。その場合、YAML 内の値は環境に合わせて置き換えます。

```
apiVersion: v1
kind: Pod
metadata:
  name: <pod_name>
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "sriov-numa-0-network", (1)
        }
      ]
spec:
  containers:
  - name: <container_name>
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

(1) これは、**SriovNetworkNodePolicy** リソースを使用する **SriovNetwork** リソースの名前です。

   b. 次のコマンドを実行して、**Pod** リソースを作成します。

```
$ oc create -f sriov-network-pod.yaml
```

出力例

```
pod/example-pod created
```

検証

1. 次のコマンドを実行して、Pod のステータスを確認します。その場合、**<pod_name>** は Pod の名前に置き換えます。

```
$ oc get pod <pod_name>
```

出力例

```
NAME                                     READY   STATUS    RESTARTS   AGE
test-deployment-sriov-76cbbf4756-k9v72   1/1     Running   0          45h
```

2. ターゲット Pod とのデバッグセッションを開き、SR-IOV ネットワークリソースがメモリーおよび CPU リソースとは異なるノードにデプロイされていることを確認します。

   a. 次のコマンドを実行して、Pod とのデバッグセッションを開きます。その場合、<pod_name> はターゲット Pod の名前に置き換えます。

   ```
   $ oc debug pod/<pod_name>
   ```

   b. **/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** にホストから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、ホストファイルシステムからのバイナリーを実行できます。

   ```
   $ chroot /host
   ```

   c. 次のコマンドを実行して、CPU 割り当てに関する情報を表示します。

   ```
   $ lscpu | grep NUMA
   ```

   出力例

   ```
   NUMA node(s):                2
   NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,...
   NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,...
   ```

   ```
   $ cat /proc/self/status | grep Cpus
   ```

   出力例

   ```
   Cpus_allowed: aa
   Cpus_allowed_list: 1,3,5,7
   ```

   ```
   $ cat  /sys/class/net/net1/device/numa_node
   ```

   出力例

   ```
   0
   ```

   この例では、CPU 1、3、5、7 が **NUMA node1** に割り当てられていますが、SR-IOV ネットワークリソースは **NUMA node0** の NIC を使用できます。

注記

excludeTopology 仕様が **True** に設定されている場合、必要なリソースが同じ NUMA ノードに存在する可能性があります。

関連情報

- CPU マネージャーの使用

27.4.6. 次のステップ

- SR-IOV ネットワーク割り当ての設定

27.5. 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。

27.5.1. 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。
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

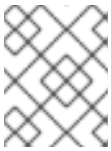
指定する値は引用符で囲む必要があります。引用符で囲まないと、オブジェクトが SR-IOV Network Operator によって拒否されます。

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。**'{ "ips": true }'** を指定して IP アドレスのサポートを有効にするか、**'{ "mac": true }'** を指定して MAC アドレスのサポートを有効にすることができます。

27.5.1.1. 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。

27.5.1.1.1. 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27.3 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。

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27.4 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27.5 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27.6 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ー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"
        }
      ]
  }
}
```

27.5.1.1.2. 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 Network 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27.7 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"
  }
}
```

27.5.1.1.3. 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27.8 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) のリスト。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"
    ]
  }
}
```

27.5.1.2. デュアルスタック IP アドレスを動的に割り当てる設定の作成

デュアルスタックの IP アドレスの割り当ては、**ipRanges** パラメーターで設定できます。

- IPv4 アドレス
- IPv6 アドレス
- 複数の IP アドレスの割り当て

手順

1. **type** を **whereabouts** に設定します。
2. 以下の例のように、**ipRanges** を使用して IP アドレスを割り当てます。

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
```

```
rawCNIConfig: |-
 {
  "name": "whereabouts-dual-stack",
  "cniVersion": "0.3.1,
  "type": "bridge",
  "ipam": {
    "type": "whereabouts",
    "ipRanges": [
           {"range": "192.168.10.0/24"},
           {"range": "2001:db8::/64"}
        ]
  }
 }
```

3. ネットワークを Pod にアタッチします。詳細は、追加のネットワークへの Pod の追加を参照してください。
4. すべての IP アドレスが割り当てられていることを確認します。
5. 以下のコマンドを実行して、IP アドレスがメタデータとして割り当てられることを確認します。

```
$ oc exec -it mypod -- ip a
```

関連情報

- Pod の追加のネットワークへの割り当て

27.5.2. 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: SriovNetwork
metadata:
```

```
    name: attach1
    namespace: openshift-sriov-network-operator
  spec:
    resourceName: net1
    networkNamespace: project2
    ipam: |-
      {
        "type": "host-local",
        "subnet": "10.56.217.0/24",
        "rangeStart": "10.56.217.171",
        "rangeEnd": "10.56.217.181",
        "gateway": "10.56.217.1"
      }
```

2. 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。

3. 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 **SriovNetwork** 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 **SriovNetwork** 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>
```

27.5.3. 次のステップ

- Pod の SR-IOV の追加ネットワークへの追加

27.5.4. 関連情報

- SR-IOV ネットワークデバイスの設定

27.6. 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。

27.6.1. InfiniBand デバイス設定オブジェクト

SriovIBNetwork 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、**SriovIBNetwork** オブジェクトについて説明しています。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
```

```
resourceName: <sriov_resource_name> 3
networkNamespace: <target_namespace> 4
ipam: |- 5
  {}
linkState: <link_state> 6
capabilities: <capabilities> 7
```

1. 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 Operator がインストールされている namespace。
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. **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。
5. 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。
6. 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。
7. 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) のサポートを有効にすることができます。

27.6.1.1. 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。

27.6.1.1.1. 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27.9 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27.10 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27.11 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27.12 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。
search	**array**	DNS ルックアップのクエリー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"
       }
     ]
  }
}
```

27.6.1.1.2. 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27.13 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"
  }
}
```

27.6.1.1.3. 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27.14 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) のリスト。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"
    ]
  }
}
```

27.6.1.2. デュアルスタック IP アドレスを動的に割り当てる設定の作成

デュアルスタックの IP アドレスの割り当ては、**ipRanges** パラメーターで設定できます。

- IPv4 アドレス
- IPv6 アドレス
- 複数の IP アドレスの割り当て

手順

1. **type** を **whereabouts** に設定します。
2. 以下の例のように、**ipRanges** を使用して IP アドレスを割り当てます。

```
cniVersion: operator.openshift.io/v1
kind: Network
=metadata:
  name: cluster
spec:
  additionalNetworks:
  - name: whereabouts-shim
    namespace: default
    type: Raw
    rawCNIConfig: |-
      {
       "name": "whereabouts-dual-stack",
       "cniVersion": "0.3.1,
       "type": "bridge",
       "ipam": {
         "type": "whereabouts",
         "ipRanges": [
                {"range": "192.168.10.0/24"},
                {"range": "2001:db8::/64"}
            ]
       }
      }
```

3. ネットワークを Pod にアタッチします。詳細は、追加のネットワークへの Pod の追加を参照してください。
4. すべての IP アドレスが割り当てられていることを確認します。
5. 以下のコマンドを実行して、IP アドレスがメタデータとして割り当てられることを確認します。

```
$ oc exec -it mypod -- ip a
```

関連情報

- Pod の追加のネットワークへの割り当て

27.6.2. 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"
    }
```

2. 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。

3. 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>
```

27.6.3. 次のステップ

- Pod の SR-IOV の追加ネットワークへの追加

27.6.4. 関連情報

- SR-IOV ネットワークデバイスの設定

27.7. 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。

27.7.1. 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。

27.7.1.1. 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"]
```

27.7.1.2. 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、**SriovIBNetwork** 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

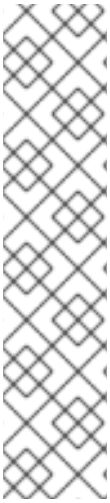
```
apiVersion: v1
kind: Pod
metadata:
  name: sample-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "ib1",
          "infiniband-guid": "c2:11:22:33:44:55:66:77",
          "ips": ["192.168.10.1/24", "2001::1/64"]
        }
      ]
spec:
  containers:
  - name: sample-container
    image: <image>
    imagePullPolicy: IfNotPresent
    command: ["sleep", "infinity"]
```

27.7.2. 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ることはできません。

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 の最初のコンテナーに **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ーのみが 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 **vfio-pci** 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ーが **Pod** オブジェクトで定義された最初のコンテナー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。
- クラスターにログインする。
- 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
```

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/network-status: |- 1
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.128.2.14"
          ],
          "default": true,
          "dns": {}
      },{
          "name": "macvlan-bridge",
          "interface": "net1",
          "ips": [
              "20.2.2.100"
          ],
          "mac": "22:2f:60:a5:f8:00",
          "dns": {}
      }]
  name: example-pod
  namespace: default
spec:
  ...
status:
  ...
```

1 **k8s.v1.cni.cncf.io/network-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

27.7.3. 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 。より柔軟な SR-IOV ネットワークリソーススケジューリングについては、**関連情報** セクションの **NUMA 対応スケジューリングにおける SR-IOV ネットワークトポロジーの除外** を参照してください。

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> 1
spec:
  containers:
  - name: sample-container
    image: <image> 2
    command: ["sleep", "infinity"]
    resources:
      limits:
        memory: "1Gi" 3
        cpu: "2" 4
      requests:
        memory: "1Gi"
        cpu: "2"
```

1 **<name>** 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。

2 **<image>** を **sample-pod** イメージの名前に置き換えます。

3 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。

4 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> 1
```

1 **<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
```

27.7.4. OpenStack で SR-IOV を使用するクラスター用のテスト Pod テンプレート

次の **testpmd** Pod では、ヒュージページ、予約済み CPU、および SR-IOV ポートを使用したコンテナーの作成を紹介します。

testpmd Pod の例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
```

```
    requests:
      memory: 1000Mi
      hugepages-1Gi: 1Gi
      cpu: '2'
      openshift.io/sriov1: 1
    limits:
      hugepages-1Gi: 1Gi
      cpu: '2'
      memory: 1000Mi
      openshift.io/sriov1: 1
  volumeMounts:
    - mountPath: /dev/hugepages
      name: hugepage
      readOnly: False
runtimeClassName: performance-cnf-performanceprofile (1)
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

(1) この例では、パフォーマンスプロファイルの名前が **cnf-performance profile** であると想定しています。

27.7.5. 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
- NUMA 対応スケジューリング用 SR-IOV ネットワークトポロジーの除外

27.8. SR-IOV ネットワークのインターフェイスレベルのネットワーク SYSCTL 設定とオールマルチキャストモードを設定する

クラスター管理者は、SR-IOV ネットワークデバイスに接続されている Pod のチューニング Container Network Interface (CNI) メタプラグインを使用して、インターフェイスレベルのネットワーク sysctl と、プロミスキャスモード、オールマルチキャストモード、MTU、MAC アドレスなどのいくつかのインターフェイス属性を変更できます。

27.8.1. SR-IOV 対応 NIC を使用したノードのラベル付け

SR-IOV 対応ノードのみで SR-IOV を有効にしたい場合は、いくつかの方法があります。

1. Node Feature Discovery (NFD) Operator をインストールします。NFD は SR-IOV 対応の NIC の存在を検出し、ノードに **node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = true** ラベルを付けます。
2. 各ノードの **SriovNetworkNodeState** CR を調べます。**interfaces** スタンザには、ワーカーノード上の SR-IOV Network 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。次のコマンドを使用して、各ノードに **feature.node.kubernetes.io/network-sriov.capable: "true"** というラベルを付けます。

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

注記

任意の名前でノードにラベルを付けることができます。

27.8.2. 1 つの sysctl フラグの設定

SR-IOV ネットワークデバイスに接続された Pod のインターフェイスレベルのネットワーク **sysctl** 設定を設定できます。

この例では、作成された仮想インターフェイスで **net.ipv4.conf.IFNAME.accept_redirects** が **1** に設定されます。

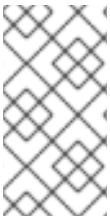
sysctl-tuning-test は、この例で使用される namespace です。

- 次のコマンドを使用して、**sysctl-tuning-test** namespace を作成します。

```
$ oc create namespace sysctl-tuning-test
```

27.8.2.1. SR-IOV ネットワークデバイスを持つノードで 1 つの sysctl フラグを設定する

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** カスタムリソース定義 (CRD) 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と、SR-IOV Operator がノードをドレインして再起動する場合があります。

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。

この手順に従って、**SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。

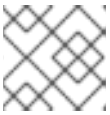
手順

1. **SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。たとえば、次の YAML をファイル **policyoneflag-sriov-node-network.yaml** として保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyoneflag 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyoneflag 3
  nodeSelector: 4
    feature.node.kubernetes.io/network-sriov.capable="true"
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
```

```
  pfNames: ["ens5"] 8
deviceType: "netdevice" 9
isRdma: false 10
```

1. 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。
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. 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。
7. 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て正確に特定することが推奨されます。**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。
8. オプション: 1 つ以上のデバイスの物理機能 (PF) 名の配列。
9. オプション: Virtual Function (VF) のドライバータイプ。許可される唯一の値は **netdevice** です。ベアメタルノードで Mellanox NIC を DPDK モードで動作させるには、**isRdma** を **true** に設定します。
10. 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。



注記

vfio-pci ドライバータイプはサポートされていません。

2. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f policyoneflag-sriov-node-network.yaml
```

設定の更新が適用された後に、**sriov-network-operator** namespace 変更のすべての Pod が **Running** ステータスに移行します。

3. 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}'
```

出力例

```
Succeeded
```

27.8.2.2. SR-IOV ネットワークでの sysctl の設定

SriovNetwork リソースのオプションの **metaPlugins** パラメーターにチューニング設定を追加することで、SR-IOV により作成された仮想インターフェイスにインターフェイス固有の **sysctl** 設定を設定できます。

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。

インターフェイスレベルのネットワーク **net.ipv4.conf.IFNAME.accept_redirects sysctl** 設定を変更するには、Container Network Interface (CNI) チューニングプラグインを使用して追加の SR-IOV ネットワークを作成します。

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-interface-sysctl.yaml** ファイルとして保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: onevalidflag 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyoneflag 3
```

```
networkNamespace: sysctl-tuning-test 4
ipam: '{ "type": "static" }' 5
capabilities: '{ "mac": true, "ips": true }' 6
metaPlugins : | 7
 {
  "type": "tuning",
  "capabilities":{
    "mac":true
  },
  "sysctl":{
     "net.ipv4.conf.IFNAME.accept_redirects": "1"
  }
 }
```

1. 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。
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. 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。
6. オプション: 追加のネットワークの機能を設定します。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。
7. オプション: metaPlugins パラメーターは、デバイスに機能を追加するために使用されます。このユースケースでは、**type** フィールドを **tuning** に設定します。設定したいインターフェイスレベルのネットワーク **sysctl** を **sysctl** フィールドに指定します。

2. **SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network-interface-sysctl.yaml
```

NetworkAttachmentDefinition CR が正常に作成されることの確認

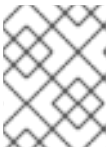
- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1. **<namespace>** を、**SriovNetwork** オブジェクトで指定した **networkNamespace** の値に置き換えます。たとえば、**sysctl-tuning-test** です。

出力例

```
NAME                 AGE
onevalidflag         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

チューニング 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。

1. **Pod** CR を作成します。次の YAML を **examplepod.yaml** ファイルとして保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "onevalidflag", (1)
          "mac": "0a:56:0a:83:04:0c", (2)
          "ips": ["10.100.100.200/24"] (3)
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

(1) SR-IOV 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。

2. **Pod** CR を作成します。

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

3. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod -n sysctl-tuning-test
```

出力例

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

4. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5. 設定された sysctl フラグの値を確認します。次のコマンドを実行して、**net.ipv4.conf.IFNAME.accept_redirects** の値を見つけます。

```
$ sysctl net.ipv4.conf.net1.accept_redirects
```

出力例

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

27.8.3. ボンディングされた SR-IOV インターフェイスフラグに関連付けられた Pod の sysctl 設定の設定

ボンディングされた SR-IOV ネットワークデバイスに接続された Pod のインターフェイスレベルのネットワーク **sysctl** 設定を設定できます。

この例では、設定可能な特定のネットワークインターフェイスレベルの **sysctl** 設定がボンドインターフェイスに設定されています。

sysctl-tuning-test は、この例で使用される namespace です。

- 次のコマンドを使用して、**sysctl-tuning-test** namespace を作成します。

```
$ oc create namespace sysctl-tuning-test
```

27.8.3.1. SR-IOV ネットワークデバイスがボンドされたノードですべての sysctl フラグを設定する

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** カスタムリソース定義 (CRD) 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。

この手順に従って、**SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。

手順

1. **SriovNetworkNodePolicy** カスタムリソース (CR) を作成します。次の YAML を **policyallflags-sriov-node-network.yaml** ファイルとして保存します。**policyallflags** を設定の名前に置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policyallflags 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: policyallflags 3
  nodeSelector: 4
    node.alpha.kubernetes-incubator.io/nfd-network-sriov.capable = `true`
  priority: 10 5
  numVfs: 5 6
  nicSelector: 7
    pfNames: ["ens1f0"] 8
  deviceType: "netdevice" 9
  isRdma: false 10
```

1. 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。
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. 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。
7. 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

8 オプション: 1 つ以上のデバイスの物理機能 (PF) 名の配列。

9 オプション: Virtual Function (VF) のドライバータイプ。許可される唯一の値は **netdevice** です。ベアメタルノードで Mellanox NIC を DPDK モードで動作させるには、**isRdma** を **true** に設定します。

10 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。

注記

vfio-pci ドライバータイプはサポートされていません。

2. SriovNetworkNodePolicy オブジェクトを作成します。

```
$ oc create -f policyallflags-sriov-node-network.yaml
```

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。

3. 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。

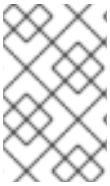
```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

出力例

```
Succeeded
```

27.8.3.2. ボンディングされた SR-IOV ネットワークでの sysctl の設定

2 つの SR-IOV インターフェイスから作成されたボンドインターフェイスで、インターフェイス固有の **sysctl** 設定を設定できます。これを行うには、bond ネットワーク接続定義のオプションの **Plugins** パラメーターにチューニング設定を追加します。



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。

特定のインターフェイスレベルのネットワーク **sysctl** 設定を変更するには、次の手順を使用して、Container Network Interface (CNI) チューニングプラグインを使用して、**SriovNetwork** カスタムリソース (CR) を作成します。

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. 次の例の CR のように、ボンドされたインターフェイスの **SriovNetwork** カスタムリソース (CR) を作成します。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: allvalidflags (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: policyallflags (3)
  networkNamespace: sysctl-tuning-test (4)
  capabilities: '{ "mac": true, "ips": true }' (5)
```

(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。

(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: 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。

2. **SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-network-attachment.yaml
```

3. 次の例の CR のように、ボンドネットワーク接続定義を作成します。YAML を **sriov-bond-network-interface.yaml** ファイルとして保存します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bond-sysctl-network
  namespace: sysctl-tuning-test
spec:
  config: '{
  "cniVersion":"0.4.0",
  "name":"bound-net",
  "plugins":[
    {
      "type":"bond", (1)
```

```
      "mode": "active-backup", 2
      "failOverMac": 1, 3
      "linksInContainer": true, 4
      "miimon": "100",
      "links": [ 5
        {"name": "net1"},
        {"name": "net2"}
      ],
      "ipam":{ 6
        "type":"static"
      }
    },
    {
      "type":"tuning", 7
      "capabilities":{
        "mac":true
      },
      "sysctl":{
        "net.ipv4.conf.IFNAME.accept_redirects": "0",
        "net.ipv4.conf.IFNAME.accept_source_route": "0",
        "net.ipv4.conf.IFNAME.disable_policy": "1",
        "net.ipv4.conf.IFNAME.secure_redirects": "0",
        "net.ipv4.conf.IFNAME.send_redirects": "0",
        "net.ipv6.conf.IFNAME.accept_redirects": "0",
        "net.ipv6.conf.IFNAME.accept_source_route": "1",
        "net.ipv6.neigh.IFNAME.base_reachable_time_ms": "20000",
        "net.ipv6.neigh.IFNAME.retrans_time_ms": "2000"
      }
    }
  ]
}'
```

1 タイプは**bond**です。

2 **mode** 属性は、ボンドモードを指定します。サポートされているボンドモードは次のとおりです。

- **balance-rr** - 0
- **active-backup** - 1
- **balance-xor** - 2
  balance-rr または **balance-xor** モードの場合には、SR-IOV Virtual Function の **trust** モードを **on** に設定する必要があります。

3 **failover** 属性は、active-backup モードでは必須です。

4 **linksInContainer=true** フラグは、必要なインターフェイスがコンテナー内にあることをボンディング CNI に通知します。デフォルトでは、ボンディング CNI は、SRIOV および Multus との統合で機能しないホストで、このようなインターフェイスを検索します。

5 **links** セクションは、結合の作成に使用するインターフェイスを定義します。デフォルトでは、Multus は接続されたインターフェイスに net と 1 から始まる連続した番号の名前を付けます。

6 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。この Pod の例では、

7 デバイスに追加の機能を追加します。たとえば、**type** フィールドを **tuning** に設定します。設定したいインターフェイスレベルのネットワーク **sysctl** を sysctl フィールドに指定します。この例では、設定可能なすべてのインターフェイスレベルのネットワーク **sysctl** 設定を設定します。

4. ボンドネットワーク接続リソースを作成します。

```
$ oc create -f sriov-bond-network-interface.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> 1
```

1 **<namespace>** は、ネットワーク割り当ての設定時に指定した networkNamespace に置き換えます (例: **sysctl-tuning-test**)。

出力例

```
NAME                    AGE
bond-sysctl-network     22m
allvalidflags           47m
```

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。

SR-IOV ネットワークリソースの追加が成功したことの確認

チューニング 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。

1. **Pod** CR を作成します。たとえば、次の YAML を **examplepod.yaml** ファイルとして保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: tunepod
  namespace: sysctl-tuning-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {"name": "allvalidflags"}, 1
        {"name": "allvalidflags"},
        {
```

```
          "name": "bond-sysctl-network",
          "interface": "bond0",
          "mac": "0a:56:0a:83:04:0c", (2)
          "ips": ["10.100.100.200/24"] (3)
        }
       ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

(1) SR-IOV 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。

2. YAML を適用します。

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

3. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod -n sysctl-tuning-test
```

出力例

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

4. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh -n sysctl-tuning-test tunepod
```

5. 設定された **sysctl** フラグの値を確認します。次のコマンドを実行して、**net.ipv6.neigh.IFNAME.base_reachable_time_ms** の値を見つけます。

```
$ sysctl net.ipv6.neigh.bond0.base_reachable_time_ms
```

出力例

```
net.ipv6.neigh.bond0.base_reachable_time_ms = 20000
```

27.8.4. オールマルチキャストモード

特にルートレスアプリケーションのコンテキストでは、オールマルチキャストモードを有効にすることが重要です。このモードを有効にしない場合は、Pod のセキュリティーコンテキスト制約 (SCC) に **NET_ADMIN** ケイパビリティーを付与する必要があります。**NET_ADMIN** ケイパビリティーを使用して、特定の要件を超える変更を行う権限を Pod に付与すると、セキュリティーの脆弱性が露呈する可能性があります。

チューニング CNI プラグインは、オールマルチキャストモードを含め、いくつかのインターフェイス属性の変更をサポートしています。このモードを有効にすると、SR-IOV ネットワークデバイス上で設定された Virtual Function (VF) 上で実行されているアプリケーションは、接続されている物理機能が同じか異なるかにかかわらず、他の VF 上のアプリケーションからマルチキャストトラフィックを受信できます。

27.8.4.1. SR-IOV ネットワーク上でオールマルチキャストモードを有効にする

SR-IOV インターフェイスでオールマルチキャストモードを有効にするには、次の方法があります。

- **SriovNetwork** リソースの **metaPlugins** パラメーターにチューニング設定を追加します。
- チューニング設定で、**allmulti** フィールドを **true** に設定します。

注記

信頼を有効にして Virtual Function (VF) を作成していることを確認してください。

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。

このガイダンスに従って、SR-IOV ネットワーク上でオールマルチキャストモードを有効にします。

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 OpenShift Container Platform クラスターにログインしている。
- SR-IOV Network Operator がインストールされている。

- 適切な **SriovNetworkNodePolicy** オブジェクトを設定している。

手順

1. Mellanox ConnectX-5 デバイスの **SriovNetworkNodePolicy** オブジェクトを定義する次の設定を使用して、YAML ファイルを作成します。YAML ファイルを **sriovnetpolicy-mlx.yaml** として保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriovnetpolicy-mlx
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  nicSelector:
    deviceID: "1017"
    pfNames:
      - ens8f0np0#0-9
    rootDevices:
      - 0000:d8:00.0
    vendor: "15b3"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 10
  priority: 99
  resourceName: resourcemlx
```

2. オプション: SR-IOV 対応クラスターノードにラベルが付けられていない場合は、**SriovNetworkNodePolicy.Spec.NodeSelector** ラベルを追加します。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. 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 sriovnetpolicy-mlx.yaml
```

設定の更新を適用すると、**sriov-network-operator** namespace 内のすべての Pod が自動的に **Running** ステータスに移行します。

4. 次のコマンドを実行して、**enable-allmulti-test** namespace を作成します。

```
$ oc create namespace enable-allmulti-test
```

5. 追加の SR-IOV ネットワーク接続用の **SriovNetwork** カスタムリソース (CR) を作成し、以下のサンプル CR YAML のように **metaPlugins** 設定を挿入し、そのファイルを **sriov-enable-all-multicast.yaml** として保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: enableallmulti (1)
  namespace: openshift-sriov-network-operator (2)
spec:
  resourceName: enableallmulti (3)
```

```
networkNamespace: enable-allmulti-test 4
ipam: '{ "type": "static" }' 5
capabilities: '{ "mac": true, "ips": true }' 6
trust: "on" 7
metaPlugins : | 8
 {
  "type": "tuning",
  "capabilities":{
   "mac":true
  },
  "allmulti": true
  }
 }
```

1. オブジェクトの名前を指定します。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** パラメーター値を指定します。
4. **SriovNetwork** オブジェクトのターゲット namespace を指定します。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. 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。
6. オプション: 追加のネットワークの機能を設定します。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。
7. Virtual Function の信頼モードを指定します。これは "on" に設定する必要があります。
8. **metaPlugins** パラメーターを使用して、デバイスにケイパビリティーをさらに追加します。このユースケースでは、**type** フィールドを **tuning** に設定し、**allmulti** フィールドを追加して **true** に設定します。

6. 次のコマンドを実行して、**SriovNetwork** リソースを作成します。

```
$ oc create -f sriov-enable-all-multicast.yaml
```

NetworkAttachmentDefinition CR の検証

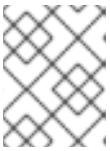
- 以下のコマンドを実行して、SR-IOV Network Operator が **NetworkAttachmentDefinition** CR を作成していることを確認します。

```
$ oc get network-attachment-definitions -n <namespace> 1
```

1. **<namespace>** を、**SriovNetwork** オブジェクトで指定した **networkNamespace** の値に置き換えます。この例では、**enable-allmulti-test** です。

出力例

```
NAME                    AGE
enableallmulti          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。

1. 次のコマンドを実行して、SR-IOV ネットワークリソースに関する情報を表示します。

```
$ oc get sriovnetwork -n openshift-sriov-network-operator
```

追加の SR-IOV ネットワーク接続の検証

チューニング CNI が正しく設定されていること、および追加の SR-IOV ネットワーク接続が割り当てられていることを確認するには、次の手順を実行します。

1. **Pod** CR を作成します。次のサンプル YAML を **examplepod.yaml** という名前のファイルに保存します。

```
apiVersion: v1
kind: Pod
metadata:
  name: samplepod
  namespace: enable-allmulti-test
  annotations:
    k8s.v1.cni.cncf.io/networks: |-
      [
        {
          "name": "enableallmulti",  1
          "mac": "0a:56:0a:83:04:0c",  2
          "ips": ["10.100.100.200/24"]  3
        }
      ]
spec:
  containers:
  - name: podexample
    image: centos
    command: ["/bin/bash", "-c", "sleep INF"]
    securityContext:
      runAsUser: 2000
      runAsGroup: 3000
      allowPrivilegeEscalation: false
      capabilities:
        drop: ["ALL"]
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

1. SR-IOV network attachment definition CR の名前を指定します。

2.

オプション: SR-IOV network attachment definition 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 アドレスを指定します。この機能を使用するに

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 アドレスを指定します。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。

2. 以下のコマンドを実行して **Pod** を作成します。

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

3. 次のコマンドを実行して、Pod が作成されていることを確認します。

```
$ oc get pod -n enable-allmulti-test
```

出力例

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

4. 次のコマンドを実行して、Pod にログインします。

```
$ oc rsh -n enable-allmulti-test samplepod
```

5. 次のコマンドを実行して、Pod に関連付けられているすべてのインターフェイスをリスト表示します。

```
sh-4.4# ip link
```

出力例

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8901 qdisc noqueue state
UP mode DEFAULT group default
    link/ether 0a:58:0a:83:00:10 brd ff:ff:ff:ff:ff:ff link-netnsid 0 1
3: net1@if24: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP mode DEFAULT group default
    link/ether ee:9b:66:a4:ec:1d brd ff:ff:ff:ff:ff:ff link-netnsid 0 2
```

1 **eth0@if22** は、プライマリーインターフェイスです。

2 **net1@if24** は、オールマルチキャストモード (**ALLMULTI**l フラグ) をサポートするネットワーク接続定義で設定されたセカンダリーインターフェイスです。

27.9. 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す。

27.9.1. 高パフォーマンスのマルチキャスト

OpenShift SDN ネットワークプラグインは、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。

27.9.2. 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"] 1
    command: [ "sleep", "infinity"]
```

1 **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。

27.10. DPDK および RDMA の使用

コンテナー化された 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) で利用できます。

サポートされているデバイスについては、サポートされているデバイス を参照してください。

27.10.1. 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 (1)
```

(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に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。

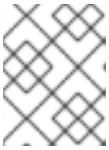
```
$ oc create -f intel-dpdk-node-policy.yaml
```

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: intelnics
```

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、コンテナーの親 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> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: intel-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
    resources:
      limits:
        openshift.io/intelnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
```

```
      openshift.io/intelnics: "1"
      memory: "1Gi"
      cpu: "4"
      hugepages-1Gi: "4Gi"
  command: ["sleep", "infinity"]
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

1. **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。

2. 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。

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ー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 **/mnt/huge** の下の 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。

5. 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
```

27.10.2. Mellanox NIC を使用した DPDK モードでの Virtual Function の使用

Mellanox NIC で DPDK モードの Virtual Function を使用して、ネットワークノードポリシーを作成し、Data Plane Development Kit (DPDK) Pod を作成できます。

前提条件

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

- Single Root I/O Virtualization (SR-IOV) Network Operator 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. 次の **SriovNetworkNodePolicy** YAML 設定を **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: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

(1) 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。

(2) Virtual Function (VF) のドライバータイプを **netdevice** に指定します。Mellanox SR-IOV Virtual Function (VF) は、**vfio-pci** デバイスタイプを使用せずに DPDK モードで機能します。VF デバイスは、コンテナー内のカーネルネットワークインターフェイスとして表示されます。

(3) リモートダイレクトメモリーアクセス (RDMA) モードを有効にします。これは、DPDK モードで機能させるために Mellanox カードで必要です。

注記

SriovNetworkNodePolicy オブジェクトの各オプションの詳細な説明については **SR-IOV ネットワークデバイスの設定** 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に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-dpdk-node-policy.yaml
```

3. 次の **SriovNetwork** YAML 設定を **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: |- 1
...
  vlan: <vlan>
  resourceName: mlxnics
```

1 IP アドレス管理 (IPAM) コンテナーネットワークインターフェイス (CNI) プラグインの設定オブジェクトを YAML ブロックスカ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 オブジェクトの各オプションの詳細な説明については **SR-IOV ネットワークデバイスの設定** を参照してください。

app-netutil オプションライブラリーには、コンテナーの親 Pod に関するネットワーク情報を収集するための API メソッドが複数あります。

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 mlx-dpdk-network.yaml
```

5. 次の **Pod** YAML 設定を **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> 1
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-dpdk-network
spec:
  containers:
  - name: testpmd
    image: <DPDK_image> 2
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] 3
    volumeMounts:
    - mountPath: /mnt/huge 4
      name: hugepage
```

```
    resources:
      limits:
        openshift.io/mlxnics: "1" 5
        memory: "1Gi"
        cpu: "4" 6
        hugepages-1Gi: "4Gi" 7
      requests:
        openshift.io/mlxnics: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1. **SriovNetwork** 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。別の namespace で Pod を作成するには、**Pod** 仕様と **SriovNetwork** オブジェクトの両方で **target_namespace** を変更します。
2. 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。
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ー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 **/mnt/huge** の下の 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。
5. 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。これを行うには、CPU マネージャーポリシーを **static** に設定し、サービス品質 (QoS) が **Guaranteed** の Pod を作成します。
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。

6. 以下のコマンドを実行して DPDK Pod を作成します。

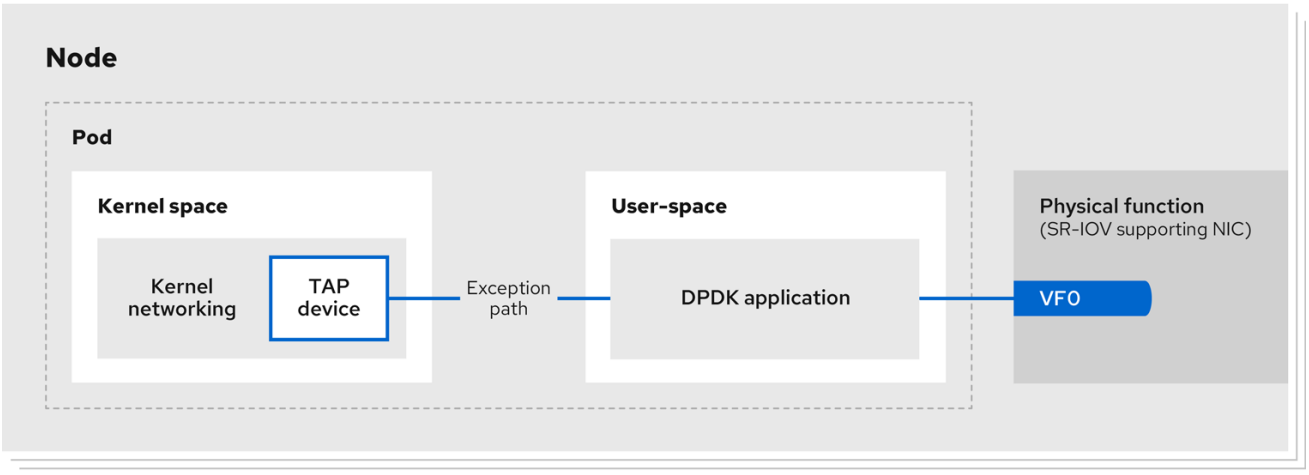
```
$ oc create -f mlx-dpdk-pod.yaml
```

27.10.3. TAP CNI を使用したカーネルアクセスでのルートレス DPDK ワークロード実行

DPDK アプリケーションは、ログメッセージなどの特定の種類のパケットを処理のためにカーネルに挿入するための例外パスとして **virtio-user** を使用できます。この機能の詳細は、例外パスとしての Virtio_user を参照してください。

OpenShift Container Platform バージョン 4.14 以降では、非特権 Pod を使用して、tap CNI プラグインと一緒に DPDK アプリケーションを実行できます。この機能を有効にするには、**SriovNetworkNodePolicy** オブジェクト内で **needVhostNet** パラメーターを **true** に設定して、**vhost-net** デバイスをマウントする必要があります。

図27.1 DPDK と TAP の設定例



前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator 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。
- すべてのノードで **setsebools container_use_devices=on** が root として設定されていることを確認します。

注記

Machine Config Operator を使用して、この SELinux ブール値を設定します。

手順

1. 次の例のような内容を含むファイル (**test-namespace.yaml** など) を作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: test-namespace
  labels:
    pod-security.kubernetes.io/enforce: privileged
    pod-security.kubernetes.io/audit: privileged
    pod-security.kubernetes.io/warn: privileged
    security.openshift.io/scc.podSecurityLabelSync: "false"
```

2. 次のコマンドを実行して、**Namespace** オブジェクトを新規作成します。

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

3. 次の例のようなコンテンツを含むファイル (**sriov-node-network-policy.yaml** など) を作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
 name: sriovnic
 namespace: openshift-sriov-network-operator
spec:
 deviceType: netdevice (1)
 isRdma: true (2)
 needVhostNet: true (3)
 nicSelector:
   vendor: "15b3" (4)
   deviceID: "101b" (5)
   rootDevices: ["00:05.0"]
 numVfs: 10
 priority: 99
 resourceName: sriovnic
 nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

(1) これは、プロファイルが Mellanox ネットワークインターフェイスコントローラー (NIC) 専用に調整されていることを示します。

(2) **isRdma** を **true** に設定する必要があるのは、Mellanox NIC の場合のみです。

(3) これにより、**/dev/net/tun** および **/dev/vhost-net** デバイスがコンテナーにマウントされ、アプリケーションがタップデバイスを作成し、タップデバイスを DPDK ワークロードに接続できるようになります。

(4) SR-IOV ネットワークデバイスのベンダーの 16 進数コード。値 15b3 は Mellanox NIC に関連付けられています。

(5) SR-IOV ネットワークデバイスのデバイスの 16 進数コード。

4. 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 sriov-node-network-policy.yaml
```

5. 次の **SriovNetwork** オブジェクトを作成し、YAML を **sriov-network-attachment.yaml** ファイルに保存します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
 name: sriov-network
 namespace: openshift-sriov-network-operator
spec:
 networkNamespace: test-namespace
```

```
resourceName: sriovnic
spoofChk: "off"
trust: "on"
```

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、コンテナーの親 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。

6. 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 sriov-network-attachment.yaml
```

7. 次の例のような内容を含む、ネットワーク割り当て定義を指定するファイル (**Tap-example.yaml** など) を作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: tap-one
 namespace: test-namespace (1)
spec:
 config: '{
   "cniVersion": "0.4.0",
   "name": "tap",
   "plugins": [
    {
      "type": "tap",
      "multiQueue": true,
      "selinuxcontext": "system_u:system_r:container_t:s0"
    },
    {
      "type":"tuning",
      "capabilities":{
        "mac":true
      }
    }
   ]
}'
```

(1) **SriovNetwork** オブジェクトが作成されるのと同じ **target_namespace** を指定します。

8. 次のコマンドを実行して、**NetworkAttachmentDefinition** オブジェクトを作成します。

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

9. 次の例のような内容を含むファイル **(dpdk-pod-rootless.yaml** など) を作成します。

```
apiVersion: v1
kind: Pod
```

```
metadata:
  name: dpdk-app
  namespace: test-namespace (1)
  annotations:
    k8s.v1.cni.cncf.io/networks: '[
      {"name": "sriov-network", "namespace": "test-namespace"},
      {"name": "tap-one", "interface": "ext0", "namespace": "test-namespace"}]'
spec:
  nodeSelector:
    kubernetes.io/hostname: "worker-0"
  securityContext:
      fsGroup: 1001 (2)
      runAsGroup: 1001 (3)
      seccompProfile:
        type: RuntimeDefault
  containers:
  - name: testpmd
    image: <DPDK_image> (4)
    securityContext:
      capabilities:
        drop: ["ALL"] (5)
        add: (6)
          - IPC_LOCK
          - NET_RAW #for mlx only (7)
      runAsUser: 1001 (8)
      privileged: false (9)
      allowPrivilegeEscalation: true (10)
      runAsNonRoot: true (11)
    volumeMounts:
    - mountPath: /mnt/huge (12)
      name: hugepages
    resources:
      limits:
        openshift.io/sriovnic: "1" (13)
        memory: "1Gi"
        cpu: "4" (14)
        hugepages-1Gi: "4Gi" (15)
      requests:
        openshift.io/sriovnic: "1"
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  runtimeClassName: performance-cnf-performanceprofile (16)
  volumes:
  - name: hugepages
    emptyDir:
      medium: HugePages
```

(1) **SriovNetwork** オブジェクトが作成されるのと同じ **target_namespace** を指定します。Pod を別の namespace に作成する場合は、**target_namespace** を **Pod** 仕様と **SriovNetwork** オブジェクトの両方で変更します。

(2)

ボリュームにマウントされたディレクトリーおよびそれらのボリューム内に作成されたファイルのグループ所有権を設定します。

3 コンテナーの実行に使用するプライマリーグループ ID を指定します。

4 アプリケーションを含む DPDK イメージとアプリケーションで使用される DPDK ライブラリーを指定します。

5 コンテナーの securityContext からすべての機能 (**ALL**) を削除すると、通常の操作に必要とされる権限以上の特権がコンテナーからなくなります。

6 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ー内のアプリケーションに必要な追加機能を指定します。これらの機能は、**setcap** コマンドを使用してバイナリーファイルでも設定する必要があります。

7 Mellanox ネットワークインターフェイスコントローラー (NIC) には、**NET_RAW** 機能が必要です。

8 コンテナーの実行に使用するユーザー ID を指定します。

9 この設定で、Pod 内のコンテナー (複数可) にホストシステムへの特権アクセスを許可しないように指定します。

10 この設定を使用すると、コンテナーは、割り当てられている初期の root 以外の権限を超えて権限を昇格できます。

11 また、この設定により、コンテナーは root 以外のユーザーで実行されます。これにより、最小特権の原則が適用され、コンテナーが不正アクセスされる可能性を最小限に抑えるとともに、攻撃対象領域を減少させます。

12 hugepage ボリュームを **/mnt/huge** の下の 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。

13 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。

14 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。

15 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。

16 パフォーマンスプロファイルの名前が **cnf-performance profile** でない場合は、その文字列を正しいパフォーマンスプロファイル名に置き換えます。

10. 以下のコマンドを実行して DPDK Pod を作成します。

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

関連情報

- container_use_devices ブール値の有効化
- パフォーマンスプロファイルの作成
- SR-IOV ネットワークデバイスの設定

27.10.4. 特定の DPDK ラインレート達成に関する概要

特定の Data Plane Development Kit (DPDK) ラインレートを実現するには、Node Tuning Operator をデプロイし、Single Root I/O Virtualization (SR-IOV) を設定します。次のリソースの DPDK 設定も調整する必要があります。

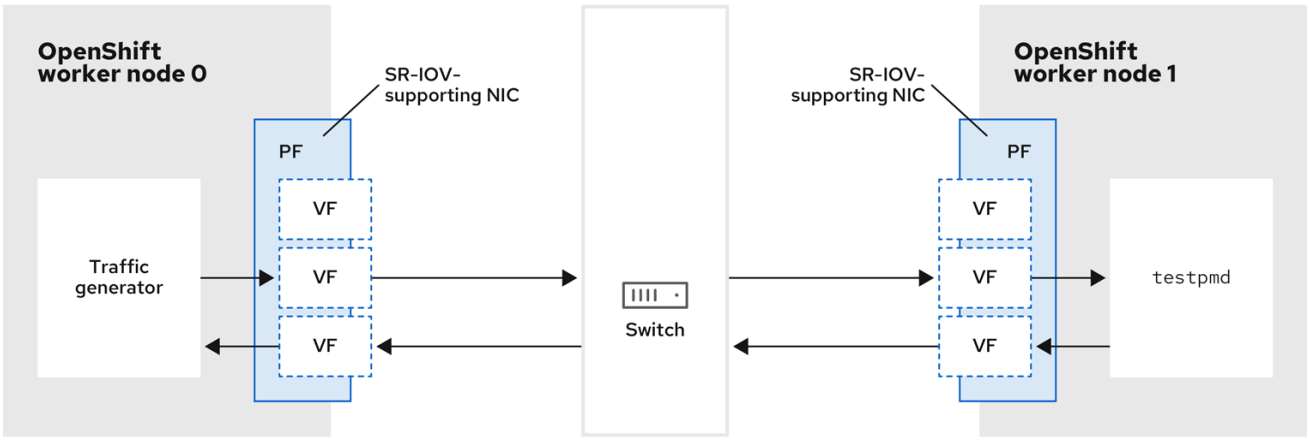
- 分離された CPU
- hugepage
- トポロジースケジューラー

注記

OpenShift Container Platform の以前のバージョンでは、パフォーマンスアドオン Operator を使用して自動チューニングを実装し、OpenShift Container Platform アプリケーションの低レイテンシーパフォーマンスを実現していました。OpenShift Container Platform 4.11 以降では、この機能は Node Tuning Operator の一部です。

DPDK テスト環境

次の図は、トラフィックテスト環境のコンポーネントを示しています。



- **トラフィックジェネレーター**: 大量のパケットトラフィックを生成できるアプリケーション。
- **SR-IOV 対応 NIC**: SR-IOV に対応したネットワークインターフェイスカードです。カードは、物理インターフェイス上で多数の Virtual Function を実行します。

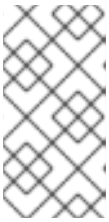
- **Physical Function (PF)**: SR-IOV インターフェイスをサポートするネットワークアダプターの PCI Express (PCIe) 機能。
- **Virtual Function (VF)**: SR-IOV をサポートするネットワークアダプター上の軽量の PCIe 機能。VF は、ネットワークアダプターの PCIe PF に関連付けられています。VF は、ネットワークアダプターの仮想化されたインスタンスを表します。
- **スイッチ**: ネットワークスイッチ。ノードは中断なしに接続することもできます。
- **testpmd**: DPDK に含まれるサンプルアプリケーション。**testpmd** アプリケーションを使用して、パケット転送モードで DPDK をテストできます。**testpmd** アプリケーションは、DPDK ソフトウェア開発キット (SDK) を使用して本格的なアプリケーションを構築する方法の例でもあります。
- **worker 0** および **worker 1**: OpenShift Container Platform ノード。

27.10.5. SR-IOV と Node Tuning Operator を使用した DPDK ラインレートの実現

Node Tuning Operator を使用して、分離された CPU、ヒュージページ、およびトポロジースケジューラーを設定できます。その後、Node Tuning Operator と Single Root I/O Virtualization (SR-IOV) を使用して、特定の Data Plane Development Kit (DPDK) ラインレートを実現できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator 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。
- スタンドアロン Node Tuning Operator をデプロイしている。



注記

OpenShift Container Platform の以前のバージョンでは、パフォーマンスアドオン Operator を使用して自動チューニングを実装し、OpenShift アプリケーションの低レイテンシーパフォーマンスを実現していました。OpenShift Container Platform 4.11 以降では、この機能は Node Tuning Operator の一部です。

手順

1. 次の例に基づいて **PerformanceProfile** オブジェクトを作成します。

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: performance
spec:
  globallyDisableIrqLoadBalancing: true
  cpu:
    isolated: 21-51,73-103 (1)
    reserved: 0-20,52-72 (2)
  hugepages:
    defaultHugepagesSize: 1G (3)
    pages:
```

```
    - count: 32
      size: 1G
net:
  userLevelNetworking: true
numa:
  topologyPolicy: "single-numa-node"
nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
```

1. システムでハイパースレッディングが有効になっている場合は、関連するシンボリックリンクを **isolated** および **reserved** の CPU グループに割り当てます。システムに複数の Non-Uniform Memory Access (NUMA) ノードが含まれている場合は、両方の NUMA から両方のグループに CPU を割り当てます。このタスクには Performance Profile Creator を使用することもできます。詳細は、**コントロールプレーンプロファイルの作成** について参照してください。

2. キューが予約済みの CPU 数に設定されているデバイスのリストを指定することもできます。詳細については **Node Tuning Operator を使用した NIC キューの削減**を参照してください。

3. 必要なヒュージページの数とサイズを割り当てます。ヒュージページの NUMA 設定を指定できます。デフォルトでは、システムは、そのシステムにあるすべての NUMA ノードに偶数分を割り当てます。必要に応じて、ノードのリアルタイムカーネルの使用をリクエストできます。詳しくは、 **リアルタイム機能を備えたワーカーのプロビジョニング** を参照してください。

2. **yaml** ファイルを **mlx-dpdk-perfprofile-policy.yaml** として保存します。

3. 次のコマンドを使用して、パフォーマンスプロファイルを適用します。

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

27.10.5.1. Virtual Function の SR-IOV Network Operator の例

Single Root I/O Virtualization (SR-IOV) ネットワーク Operator を使用して、ノード上の SR-IOV をサポートする Physical Function NIC から Virtual Function (VF) を割り当てて設定できます。

Operator のデプロイの詳細については、**SR-IOV Network Operator のインストール** を参照してください。SR-IOV ネットワークデバイスの設定の詳細については、**SR-IOV ネットワークデバイスの設定** を参照してください。

Intel VF と Mellanox VF での Data Plane Development Kit (DPDK) ワークロードの実行にはいくつかの違いがあります。このセクションでは、両方の VF タイプのオブジェクト設定の例を示します。以下は、Intel NIC で DPDK アプリケーションを実行するために使用される **sriovNetworkNodePolicy** オブジェクトの例です。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci 1
  needVhostNet: true 2
```

```
  nicSelector:
    pfNames: ["ens3f0"]
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: vfio-pci
  needVhostNet: true
  nicSelector:
    pfNames: ["ens3f1"]
  nodeSelector:
  node-role.kubernetes.io/worker-cnf: ""
  numVfs: 10
  priority: 99
  resourceName: dpdk_nic_2
```

1 Intel NIC の場合、**deviceType** は **vfio-pci** である必要があります。

2 DPDK ワークロードとのカーネル通信が必要な場合は、**needVhostNet: true** を追加します。これにより、**/dev/net/tun** および **/dev/vhost-net** デバイスがコンテナーにマウントされ、アプリケーションがタップデバイスを作成し、タップデバイスを DPDK ワークロードに接続できるようになります。

以下は、Mellanox NIC の **sriovNetworkNodePolicy** オブジェクトの例です。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-1
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice 1
  isRdma: true 2
  nicSelector:
    rootDevices:
      - "0000:5e:00.1"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_1
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: dpdk-nic-2
  namespace: openshift-sriov-network-operator
```

```
spec:
  deviceType: netdevice
  isRdma: true
  nicSelector:
    rootDevices:
      - "0000:5e:00.0"
  nodeSelector:
    node-role.kubernetes.io/worker-cnf: ""
  numVfs: 5
  priority: 99
  resourceName: dpdk_nic_2
```

1. Mellanox デバイスの場合、**deviceType** は **netdevice** である必要があります。
2. Mellanox デバイスの場合、**isRdma** は **true** である必要があります。Mellanox カードは、Flow Bifurcation を使用して DPDK アプリケーションに接続されます。このメカニズムは、Linux ユーザー空間とカーネル空間の間でトラフィックを分割し、ラインレートの処理能力を高めることができます。

27.10.5.2. SR-IOV Network Operator の例

以下は、**sriovNetwork** オブジェクトの定義例です。この場合、Intel と Mellanox の設定は同じです。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-1
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.1.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}' 1
  networkNamespace: dpdk-test 2
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_1 3
---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-network-2
  namespace: openshift-sriov-network-operator
spec:
  ipam: '{"type": "host-local","ranges": [[{"subnet": "10.0.2.0/24"}]],"dataDir":
   "/run/my-orchestrator/container-ipam-state-1"}'
  networkNamespace: dpdk-test
  spoofChk: "off"
  trust: "on"
  resourceName: dpdk_nic_2
```

1. Whereabouts など、別の IP Address Management (IPAM) 実装を使用できます。詳細については **Whereabouts を使用した動的 IP アドレス割り当ての設定** を参照してください。
2. ネットワーク接続定義が作成される **networkNamespace** を要求する必要があります。**openshift-sriov-network-operator** namespace で **sriovNetwork** CR を作成する必要があります。

3 **resourceName** の値は、**sriovNetworkNodePolicy** で作成された **resourceName** の値と一致する必要があります。

27.10.5.3. DPDK ベースワークロードの例

以下は、Data Plane Development Kit (DPDK) コンテナーの例です。

```
apiVersion: v1
kind: Namespace
metadata:
  name: dpdk-test
---
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/networks: '[ 1
     {
      "name": "dpdk-network-1",
      "namespace": "dpdk-test"
     },
     {
      "name": "dpdk-network-2",
      "namespace": "dpdk-test"
     }
    ]'
    irq-load-balancing.crio.io: "disable" 2
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
  labels:
    app: dpdk
  name: testpmd
  namespace: dpdk-test
spec:
  runtimeClassName: performance-performance 3
  containers:
    - command:
        - /bin/bash
        - -c
        - sleep INF
      image: registry.redhat.io/openshift4/dpdk-base-rhel8
      imagePullPolicy: Always
      name: dpdk
      resources: 4
        limits:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 2Gi
        requests:
          cpu: "16"
          hugepages-1Gi: 8Gi
          memory: 2Gi
      securityContext:
        capabilities:
```

```
        add:
          - IPC_LOCK
          - SYS_RESOURCE
          - NET_RAW
          - NET_ADMIN
      runAsUser: 0
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepages
terminationGracePeriodSeconds: 5
volumes:
  - emptyDir:
      medium: HugePages
    name: hugepages
```

1. 必要な SR-IOV ネットワークをリクエストします。デバイスのリソースは自動挿入されます。
2. CPU と IRQ 負荷分散ベースを無効にします。詳しくは **個々の Pod の割り込み処理の無効化** を参照してください。
3. **runtimeClass** は **performance-performance** に設定します。**runtimeClass** は **HostNetwork** または **privileged** に設定しないでください。
4. サービスの品質 (QoS) が **Guaranteed** さの Pod を開始するには、要求と制限に対して同じ数のリソースを要求します。

注記

SLEEP 状態の Pod を起動し、その Pod で exec 操作を実行して testpmd または DPDK ワークロードを開始しないでください。これにより、**exec** プロセスがどの CPU にも固定されていないため、割り込みが追加される可能性があります。

27.10.5.4. testpmd スクリプトの例

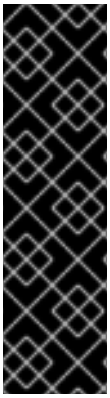
以下は、**testpmd** を実行するスクリプトの例です。

```
#!/bin/bash
set -ex
export CPU=$(cat /sys/fs/cgroup/cpuset/cpuset.cpus)
echo ${CPU}

dpdk-testpmd -l ${CPU} -a ${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_1} -a
${PCIDEVICE_OPENSHIFT_IO_DPDK_NIC_2} -n 4 -- -i --nb-cores=15 --rxd=4096 --txd=4096 --
rxq=7 --txq=7 --forward-mode=mac --eth-peer=0,50:00:00:00:00:01 --eth-peer=1,50:00:00:00:00:02
```

この例では、2 つの異なる **sriovNetwork** CR を使用しています。環境変数には、Pod に割り当てられた Virtual Function (VF) PCI アドレスが含まれています。Pod 定義で同じネットワークを使用する場合は、**pciAddress** を分割する必要があります。トラフィックジェネレータの正しい MAC アドレスを設定することが重要です。この例では、カスタム MAC アドレスを使用しています。

27.10.6. 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 製品サポート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場合があります。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: mlxnics
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  priority: <priority>
  numVfs: <num>
  nicSelector:
    vendor: "15b3"
    deviceID: "1015" (1)
    pfNames: ["<pf_name>", ...]
    rootDevices: ["<pci_bus_id>", "..."]
  deviceType: netdevice (2)
  isRdma: true (3)
```

(1) 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。

(2) Virtual Function (VF) のドライバータイプを **netdevice** に指定します。

(3) 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に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: |- 1
# ...
  vlan: <vlan>
  resourceName: mlxnics
```

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、コンテナーの親 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 **SriovNetworkNodePolicy** オブジェクトを作成します。

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

5. 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> (1)
  annotations:
    k8s.v1.cni.cncf.io/networks: mlx-rdma-network
spec:
  containers:
  - name: testpmd
    image: <RDMA_image> (2)
    securityContext:
      runAsUser: 0
      capabilities:
        add: ["IPC_LOCK","SYS_RESOURCE","NET_RAW"] (3)
    volumeMounts:
    - mountPath: /mnt/huge (4)
      name: hugepage
    resources:
      limits:
        memory: "1Gi"
        cpu: "4" (5)
        hugepages-1Gi: "4Gi" (6)
      requests:
        memory: "1Gi"
        cpu: "4"
        hugepages-1Gi: "4Gi"
    command: ["sleep", "infinity"]
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

(1) **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 に作成する場合は、**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。

(2) 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ー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 **/mnt/huge** の下の 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
```

-

27.10.7. OpenStack で OVS-DPDK を使用するクラスター用のテスト Pod テンプレート

次の **testpmd** Pod では、ヒュージページ、予約済み CPU、および SR-IOV ポートを使用したコンテナーの作成を紹介します。

testpmd Pod の例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-dpdk
  namespace: mynamespace
  annotations:
    cpu-load-balancing.crio.io: "disable"
    cpu-quota.crio.io: "disable"
# ...
spec:
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
        openshift.io/dpdk1: 1 1
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
        openshift.io/dpdk1: 1
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepage
        readOnly: False
  runtimeClassName: performance-cnf-performanceprofile 2
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

1 この例の **dpdk1** という名前は、ユーザーが作成した **SriovNetworkNodePolicy** リソースです。この名前は、作成したリソースの名前に置き換えることができます。

2 パフォーマンスプロファイルの名前が **cnf-performance profile** でない場合は、その文字列を正しいパフォーマンスプロファイル名に置き換えます。

27.10.8. OpenStack で OVS ハードウェアオフロードを使用するクラスター用のテスト Pod テンプレート

次の **testpmd** Pod は、Red Hat OpenStack Platform (RHOSP) での Open vSwitch (OVS) ハードウェアオフロードを示しています。

testpmd Pod の例

```
apiVersion: v1
kind: Pod
metadata:
  name: testpmd-sriov
  namespace: mynamespace
  annotations:
    k8s.v1.cni.cncf.io/networks: hwoffload1
spec:
  runtimeClassName: performance-cnf-performanceprofile (1)
  containers:
  - name: testpmd
    command: ["sleep", "99999"]
    image: registry.redhat.io/openshift4/dpdk-base-rhel8:v4.9
    securityContext:
      capabilities:
        add: ["IPC_LOCK","SYS_ADMIN"]
      privileged: true
      runAsUser: 0
    resources:
      requests:
        memory: 1000Mi
        hugepages-1Gi: 1Gi
        cpu: '2'
      limits:
        hugepages-1Gi: 1Gi
        cpu: '2'
        memory: 1000Mi
    volumeMounts:
      - mountPath: /mnt/huge
        name: hugepage
        readOnly: False
  volumes:
  - name: hugepage
    emptyDir:
      medium: HugePages
```

(1) パフォーマンスプロファイルの名前が **cnf-performance profile** でない場合は、その文字列を正しいパフォーマンスプロファイル名に置き換えます。

27.10.9. 関連情報

- パフォーマンスプロファイルの作成
- パフォーマンスプロファイルによる NIC キューの調整
- リアルタイムおよび低レイテンシーワークロードのプロビジョニング

- SR-IOV Network Operator のインストール
- SR-IOV ネットワークデバイスの設定
- Whereabouts を使用した動的 IP アドレス割り当ての設定
- 個別の Pod の割り込み処理の無効化
- SR-IOV イーサネットネットワーク割り当ての設定
- app-netutil library ライブラリーは、コンテナーの親 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。

27.11. POD レベルのボンディングの使用

Pod レベルでのボンディングは、高可用性とスループットを必要とする Pod 内のワークロードを有効にするために不可欠です。Pod レベルのボンディングでは、カーネルモードインターフェイスで複数の Single Root I/O Virtualization (SR-IOV) Virtual Function インターフェイスからボンドインターフェイスを作成できます。SR-IOV Virtual Function は Pod に渡され、カーネルドライバーに割り当てられます。

Pod レベルのボンディングが必要なシナリオには、異なる Physical Function 上の複数の SR-IOV Virtual Function からのボンディングインターフェイスの作成が含まれます。ホストの 2 つの異なる Physical Function からボンディングインターフェイスを作成して、Pod レベルで高可用性およびスループットを実現するために使用できます。

SR-IOV ネットワークの作成、ネットワークポリシー、ネットワーク接続定義、Pod などのタスクのガイダンスはSR-IOV ネットワークデバイスの設定を参照してください。

27.11.1. 2 つの SR-IOV インターフェイスからのボンドインターフェイスの設定

ボンディングを使用して、複数のネットワークインターフェイスを、1 つの論理的なボンディングされたインターフェイスに集約できます。Bond Container Network Interface (Bond-CNI) により、コンテナーでボンディング機能を使用できます。

Bond-CNI は、Single Root I/O Virtualization (SR-IOV) Virtual Function を使用して作成し、それらをコンテナーネットワーク namespace に配置できます。

OpenShift Container Platform は、SR-IOV Virtual Functions を使用する Bond-CNI のみをサポートします。SR-IOV Network Operator は、Virtual Function の管理に必要な SR-IOV CNI プラグインを提供します。他の CNI またはインターフェイスのタイプはサポートされていません。

前提条件

- SR-IOV Network Operator をインストールおよび設定して、コンテナー内の Virtual Functions を取得する必要があります。
- SR-IOV インターフェイスを設定するには、インターフェイスごとに SR-IOV ネットワークとポリシーを作成する必要があります。
- SR-IOV Network Operator は、定義された SR-IOV ネットワークとポリシーをもとに、各 SR-IOV インターフェイスのネットワーク接続定義を作成します。
- **linkState** は、SR-IOV Virtual Function のデフォルト値 **auto** に設定されます。

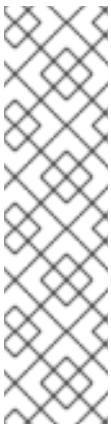
27.11.1.1. ボンドネットワーク接続定義の作成

SR-IOV Virtual Function が使用可能になったので、ボンドネットワーク接続定義を作成できます。

```
apiVersion: "k8s.cni.cncf.io/v1"
   kind: NetworkAttachmentDefinition
   metadata:
     name: bond-net1
     namespace: demo
   spec:
     config: '{
     "type": "bond", (1)
     "cniVersion": "0.3.1",
     "name": "bond-net1",
     "mode": "active-backup", (2)
     "failOverMac": 1, (3)
     "linksInContainer": true, (4)
     "miimon": "100",
     "mtu": 1500,
     "links": [ (5)
          {"name": "net1"},
          {"name": "net2"}
       ],
     "ipam": {
          "type": "host-local",
          "subnet": "10.56.217.0/24",
          "routes": [{
          "dst": "0.0.0.0/0"
          }],
          "gateway": "10.56.217.1"
       }
     }'
```

(1) cni-type は常に **bond** に設定されます。

(2) **mode** 属性は、ボンドモードを指定します。



注記

サポートされているボンドモードは次のとおりです。

- **balance-rr** - 0
- **active-backup** - 1
- **balance-xor** - 2

balance-rr または **balance-xor** モードの場合には、SR-IOV Virtual Function の **trust** モードを **on** に設定する必要があります。

(3) active-backup モードでは **フェイルオーバー** 属性が必須であり、1 に設定する必要があります。

(4) **linksInContainer=true** フラグは、必要なインターフェイスがコンテナー内にあることをボンディング CNI に通知します。デフォルトでは、ボンディング CNI は、SRIOV および Multus との統合で機能しないホストで、このようなインターフェイスを検索します。

5 **links** セクションは、結合の作成に使用するインターフェイスを定義します。デフォルトでは、Multus は接続されたインターフェイスに net と 1 から始まる連続した番号の名前を付けます。

27.11.1.2. ボンディングインターフェイスを使用した Pod の作成

1. **podbonding.yaml** などの名前の YAMLファイル以下の内容を追加して Pod を作成し、この設定をテストします。

```
apiVersion: v1
  kind: Pod
  metadata:
    name: bondpod1
    namespace: demo
    annotations:
      k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1 1
  spec:
    containers:
    - name: podexample
      image: quay.io/openshift/origin-network-interface-bond-cni:4.11.0
      command: ["/bin/bash", "-c", "sleep INF"]
```

1 ネットワークのアノテーションに注意してください。これには、SR-IOV ネットワーク割り当てが 2 つとボンドネットワーク割り当てが 1 つ含まれています。ボンド割り当ては、2 つの SR-IOV インターフェイスをボンドポートインターフェイスとして使用します。

2. 以下のコマンドを実行して yaml を適用します。

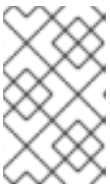
```
$ oc apply -f podbonding.yaml
```

3. 次のコマンドを使用して Pod インターフェイスを検査します。

```
$ oc rsh -n demo bondpod1
sh-4.4#
sh-4.4# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
valid_lft forever preferred_lft forever
3: eth0@if150: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc
noqueue state UP
link/ether 62:b1:b5:c8:fb:7a brd ff:ff:ff:ff:ff:ff
inet 10.244.1.122/24 brd 10.244.1.255 scope global eth0
valid_lft forever preferred_lft forever
4: net3: <BROADCAST,MULTICAST,UP,LOWER_UP400> mtu 1500 qdisc noqueue state
UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff 1
inet 10.56.217.66/24 scope global bond0
valid_lft forever preferred_lft forever
43: net1: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff 2
```

```
44: net2: <BROADCAST,MULTICAST,UP,LOWER_UP800> mtu 1500 qdisc mq master
bond0 state UP qlen 1000
link/ether 9e:23:69:42:fb:8a brd ff:ff:ff:ff:ff:ff 3
```

1. 結合インターフェイスには、自動的に **net3** という名前が付けられます。特定のインターフェイス名を設定するには、Pod の **k8s.v1.cni.cncf.io/networks** アノテーションに **@name** 接尾辞を追加します。
2. **net1**インターフェイスは、SR-IOV Virtual Function に基づいています。
3. **net2**インターフェイスは、SR-IOV Virtual Function に基づいています。



注記

Pod アノテーションでインターフェイス名が設定されていない場合、インターフェイス名は **net<n>** として自動的に割り当てられます (**<n>** は **1** から始まります)。

4. オプション: たとえば **bond0** などの特定のインターフェイス名を設定する場合は、次のように **k8s.v1.cni.cncf.io/networks** アノテーションを編集し、**bond0** をインターフェイス名として設定します。

```
annotations:
    k8s.v1.cni.cncf.io/networks: demo/sriovnet1, demo/sriovnet2, demo/bond-net1@bond0
```

27.12. ハードウェアオフロードの設定

クラスター管理者は、互換性のあるノードでハードウェアオフロードを設定して、データ処理パフォーマンスを向上させ、ホスト CPU の負荷を軽減できます。

27.12.1. ハードウェアのオフロードについて

Open vSwitch ハードウェアオフロードは、ネットワークタスクを CPU から迂回させ、ネットワークインターフェイスコントローラー上の専用プロセッサーにオフロードすることにより、ネットワークタスクを処理する方法です。その結果、クラスターは、データ転送速度の高速化、CPU ワークロードの削減、およびコンピューティングコストの削減の恩恵を受けることができます。

この機能の重要な要素は、SmartNIC と呼ばれる最新クラスのネットワークインターフェイスコントローラーです。SmartNIC は、計算量の多いネットワーク処理タスクを処理できるネットワークインターフェイスコントローラーです。専用のグラフィックカードがグラフィックパフォーマンスを向上させるのと同じように、SmartNIC はネットワークパフォーマンスを向上させることができます。いずれの場合も、専用プロセッサーにより、特定のタイプの処理タスクのパフォーマンスが向上します。

OpenShift Container Platform では、互換性のある SmartNIC を持つベアメタルノードのハードウェアオフロードを設定できます。ハードウェアオフロードは、SR-IOV Network Operator によって設定および有効化されます。

ハードウェアのオフロードは、すべてのワークロードまたはアプリケーションタイプと互換性があるわけではありません。次の 2 つの通信タイプのみがサポートされています。

- pod-to-pod
- pod-to-service。サービスは通常の Pod に基づく ClusterIP サービスです。

すべての場合において、ハードウェアのオフロードは、それらの Pod とサービスが互換性のある SmartNIC を持つノードに割り当てられている場合にのみ行われます。たとえば、ハードウェアをオフロードしているノードの Pod が、通常のノードのサービスと通信しようとしているとします。通常のノードでは、すべての処理がカーネルで行われるため、Pod からサービスへの通信の全体的なパフォーマンスは、その通常のノードの最大パフォーマンスに制限されます。ハードウェアオフロードは、DPDK アプリケーションと互換性がありません。

ノードでのハードウェアのオフロードを有効にし、使用する Pod を設定しないと、Pod トラフィックのスループットパフォーマンスが低下する可能性があります。OpenShift Container Platform で管理される Pod のハードウェアオフロードを設定することはできません。

27.12.2. サポートされるデバイス

ハードウェアオフロードは、次のネットワークインターフェイスコントローラーでサポートされています。

表27.15 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
Mellanox	MT27800 Family [ConnectX-5]	15b3	1017
Mellanox	MT28880 Family [ConnectX-5 Ex]	15b3	1019
Mellanox	MT2892 Family [ConnectX-6 Dx]	15b3	101d
Mellanox	MT2894 ファミリー ConnectX-6 Lx	15b3	101f
Mellanox	ConnectX-6 NIC モードの MT42822 BlueField-2	15b3	a2d6

27.12.3. 前提条件

- クラスターに、ハードウェアのオフロードがサポートされているネットワークインターフェイスコントローラーを備えたベアメタルマシンが少なくとも 1 台ある。
- SR-IOV Network Operator をインストール している。
- クラスターで OVN-Kubernetes ネットワークプラグインを使用 している。
- OVN-Kubernetes ネットワークプラグイン設定 で、**gatewayConfig.routingViaHost** フィールドが **false** に設定されています。

27.12.4. ハードウェアオフロード用のマシン設定プールの設定

ハードウェアオフロードを有効にするには、最初に専用のマシン設定プールを作成し、SR-IOV Network Operator と連携するように設定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** 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. ハードウェアオフロードを使用するマシンのマシン設定プールを作成します。

   a. 次の例のようなコンテンツを含む **mcp-offloading.yaml** などのファイルを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: mcp-offloading 1
spec:
  machineConfigSelector:
    matchExpressions:
      - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,mcp-
offloading]} 2
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/mcp-offloading: "" 3
```

   1 2 ハードウェアオフロード用のマシン設定プールの名前。

   3 このノードロールラベルは、マシン設定プールにノードを追加するために使用されます。

   b. マシン設定プールの設定を適用します。

```
$ oc create -f mcp-offloading.yaml
```

2. マシン設定プールにノードを追加します。プールのノードロールラベルで各ノードにラベルを付けます。

```
$ oc label node worker-2 node-role.kubernetes.io/mcp-offloading=""
```

3. オプション: 新しいプールが作成されたことを確認するには、次のコマンドを実行します。

```
$ oc get nodes
```

出力例

```
NAME       STATUS   ROLES                  AGE   VERSION
master-0   Ready    master                 2d    v1.28.5
master-1   Ready    master                 2d    v1.28.5
master-2   Ready    master                 2d    v1.28.5
worker-0   Ready    worker                 2d    v1.28.5
worker-1   Ready    worker                 2d    v1.28.5
worker-2   Ready    mcp-offloading,worker  47h   v1.28.5
worker-3   Ready    mcp-offloading,worker  47h   v1.28.5
```

4. このマシン設定プールを **SriovNetworkPoolConfig** カスタムリソースに追加します。

   a. 次の例のようなコンテンツを含むファイル (**sriov-pool-config.yaml**など) を作成します。

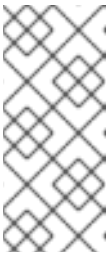
```
apiVersion: sriovnetwork.openshift.io/v1
```

```
kind: SriovNetworkPoolConfig
metadata:
  name: sriovnetworkpoolconfig-offload
  namespace: openshift-sriov-network-operator
spec:
  ovsHardwareOffloadConfig:
    name: mcp-offloading 1
```

1 ハードウェアオフロード用のマシン設定プールの名前。

b. 設定を適用します。

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



注記

SriovNetworkPoolConfig オブジェクトで指定された設定を適用すると、SR-IOV Operator は、マシン設定プール内のノードをドレインして再起動します。

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。

27.12.5. SR-IOV ネットワークノードポリシーの設定

SR-IOV ネットワークノードポリシーを作成することにより、ノードの SR-IOV ネットワークデバイス設定を作成できます。ハードウェアオフロードを有効にするには、値 **"switchdev"** を使用して **.spec.eSwitchMode** フィールドを定義する必要があります。

次の手順では、ハードウェアをオフロードするネットワークインターフェイスコントローラー用の 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アクセスできる。

手順

1. 次の例のようなコンテンツを含むファイル (**sriov-node-policy.yaml**など) を作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy <.>
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice <.>
  eSwitchMode: "switchdev" <.>
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
```

```
    vendor: "15b3"
    pfNames:
    - ens8f0
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  numVfs: 6
  priority: 5
  resourceName: mlxnics
```

<.> カスタムリソースオブジェクトの名前。<.> 必須。ハードウェアのオフロードは **vfio-pci** ではサポートされていません。<.> 必須。

2. ポリシーの設定を適用します。

```
$ oc create -f sriov-node-policy.yaml
```

注記

SriovNetworkPoolConfig オブジェクトで指定された設定を適用すると、SR-IOV Operator は、マシン設定プール内のノードをドレインして再起動します。

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。

27.12.5.1. OpenStack の SR-IOV ネットワークノードポリシーの例

次の例では、Red Hat OpenStack Platform (RHOSP) でハードウェアオフロードを使用するネットワークインターフェイスコントローラー (NIC) の SR-IOV インターフェイスについて説明します。

RHOSP でのハードウェアオフロードを備えた NIC の SR-IOV インターフェイス

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ${name}
  namespace: openshift-sriov-network-operator
spec:
  deviceType: switchdev
  isRdma: true
  nicSelector:
    netFilter: openstack/NetworkID:${net_id}
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: 'true'
  numVfs: 1
  priority: 99
  resourceName: ${name}
```

27.12.6. Virtual Function を使用したネットワークトラフィックのパフォーマンスの向上

この手順に従って、OVN-Kubernetes 管理ポートに Virtual Function を割り当て、そのネットワークトラフィックパフォーマンスを向上させます。

この手順により 2 つのプールが作成されます。1 つ目は OVN-Kubernetes によって使用される Virtual Function が設定され、2 つ目は残りの Virtual Function で設定されます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 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. 次のコマンドを実行して、SmartNIC が存在する各ワーカーノードに **network.operator.openshift.io/smart-nic** ラベルを追加します。

```
$ oc label node <node-name> network.operator.openshift.io/smart-nic=
```

oc get nodes コマンドを使用して、使用可能なノードのリストを取得します。

2. 次の例のような内容を含む、管理ポート用の **sriov-node-mgmt-vf-policy.yaml** という名前のポリシーを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-mgmt-vf-policy
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  eSwitchMode: "switchdev"
  nicSelector:
    deviceID: "1019"
    rootDevices:
    - 0000:d8:00.0
    vendor: "15b3"
    pfNames:
    - ens8f0#0-0 <.>
  nodeSelector:
    network.operator.openshift.io/smart-nic: ""
  numVfs: 6 <.>
  priority: 5
  resourceName: mgmtvf
```

<.> このデバイスは、ユースケースに適したネットワークデバイスに置き換えます。**pfNames** 値の **#0-0** の部分は、OVN-Kubernetes によって使用される単一の Virtual Function を予約します。<.> ここで指定する値は一例です。この値は、要件を満たす値に置き換えます。詳細は、**関連情報** セクションの **SR-IOV ネットワークノード設定オブジェクト** を参照してください。

3. 次の例のような内容を含む **sriov-node-policy.yaml** という名前のポリシーを作成します。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: sriov-node-policy
  namespace: openshift-sriov-network-operator
spec:
  deviceType: netdevice
  eSwitchMode: "switchdev"
  nicSelector:
    deviceID: "1019"
```

```
  rootDevices:
  - 0000:d8:00.0
  vendor: "15b3"
  pfNames:
  - ens8f0#1-5 <.>
nodeSelector:
  network.operator.openshift.io/smart-nic: ""
numVfs: 6 <.>
priority: 5
resourceName: mlxnics
```

<.> このデバイスは、ユースケースに適したネットワークデバイスに置き換えます。<.> ここで指定する値は一例です。この値は **sriov-node-mgmt-vf-policy.yaml** ファイルで指定された値に置き換えます。詳細は、**関連情報** セクションの **SR-IOV ネットワークノード設定オブジェクト** を参照してください。

注記

sriov-node-mgmt-vf-policy.yaml ファイルには、**pfNames** キーと **resourceName** キーの値が **sriov-node-policy.yaml** ファイルとは異なります。

4. 両方のポリシーの設定を適用します。

```
$ oc create -f sriov-node-policy.yaml
```

```
$ oc create -f sriov-node-mgmt-vf-policy.yaml
```

5. 管理設定用にクラスター内に Cluster Network Operator (CNO) ConfigMap を作成します。

   a. 次の内容を含む **hardware-offload-config.yaml** という名前の ConfigMap を作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: hardware-offload-config
    namespace: openshift-network-operator
data:
    mgmt-port-resource-name: openshift.io/mgmtvf
```

   b. ConfigMap の設定を適用します。

```
$ oc create -f hardware-offload-config.yaml
```

関連情報

- SR-IOV ネットワークノード設定オブジェクト

27.12.7. ネットワーク接続定義の作成

マシン設定プールと SR-IOV ネットワークノードポリシーを定義した後、指定したネットワークインターフェイスカードのネットワーク接続定義を作成できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **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. 次の例のようなコンテンツを含むファイル (**net-attach-def.yaml**など) を作成します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net-attach-def <.>
  namespace: net-attach-def <.>
  annotations:
    k8s.v1.cni.cncf.io/resourceName: openshift.io/mlxnics <.>
spec:
  config: '{"cniVersion":"0.3.1","name":"ovn-kubernetes","type":"ovn-k8s-cni-overlay","ipam":
{},"dns":{}}'
```

<.> ネットワーク接続定義の名前。<.> ネットワーク接続定義の namespace。<.> これは、**SriovNetworkNodePolicy** オブジェクトで指定した **spec.resourceName** フィールドの値です。

2. ネットワーク接続定義の設定を適用します。

```
$ oc create -f net-attach-def.yaml
```

検証

- 次のコマンドを実行して、新しい定義が存在するかどうかを確認します。

```
$ oc get net-attach-def -A
```

出力例

```
NAMESPACE         NAME             AGE
net-attach-def    net-attach-def   43h
```

27.12.8. ネットワーク接続定義を Pod へ追加

マシン設定プール、**SriovNetworkPoolConfig** および **SriovNetworkNodePolicy** カスタムリソース、およびネットワーク接続定義を作成した後、ネットワーク接続定義を Pod 仕様に追加することにより、これらの設定を Pod に適用できます。

手順

- Pod 仕様で、**.metadata.annotations.k8s.v1.cni.cncf.io/networks** フィールドを追加し、ハードウェアオフロード用に作成したネットワーク接続定義を指定します。

```
....
metadata:
  annotations:
    v1.multus-cni.io/default-network: net-attach-def/net-attach-def <.>
```

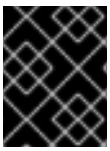
<.> 値は、ハードウェアオフロード用に作成したネットワーク接続定義の名前と namespace である必要があります。

27.13. BLUEFIELD-2 の DPU から NIC への切り替え

Bluefield-2 ネットワークデバイスをデータ処理ユニット (DPU) モードからネットワークインターフェイスコントローラー (NIC) モードに切り替えることができます。

27.13.1. Bluefield-2 を DPU モードから NIC モードに切り替える

以下の手順を使用して、Bluefield-2 をデータ処理ユニット (DPU) モードからネットワークインターフェイスコントローラー (NIC) モードに切り替えます。



重要

現在、DPU から NIC モードへの Bluefield-2 の切り替えのみがサポートされています。NIC モードから DPU モードへの切り替えはサポートされていません。

前提条件

- SR-IOV Network Operator がインストールされている。詳細については、SR-IOV Network Operator のインストールを参照してください。
- Bluefield-2 を最新のファームウェアに更新している。詳細については 、NVIDIA BlueField-2 のファームウェア を参照してください。

手順

1. 次のコマンドを入力して、各ワーカーノードに次のラベルを追加します。

```
$ oc label node <example_node_name_one> node-role.kubernetes.io/sriov=
```

```
$ oc label node <example_node_name_two> node-role.kubernetes.io/sriov=
```

2. SR-IOV Network Operator のマシン設定プールを作成します。次に例を示します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: sriov
spec:
  machineConfigSelector:
    matchExpressions:
    - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,sriov]}
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/sriov: ""
```

3. 次の **machineconfig.yaml** ファイルをワーカーノードに適用します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
```

```
  labels:
    machineconfiguration.openshift.io/role: sriov
  name: 99-bf2-dpu
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,ZmluZF9jb250YWluZXIoKSB7CiAgY3JpY3RsIHBzIC1vIGpzb24gfCBqcSAtciAnLmNv
bnRhaW5lcnNbXSB8IHNlbGVjdCgubWV0YWRhdGEubmFtZT09InNyaW92LW5ldHdvcmstY29
uZmlnLWRhZW1vbiIpIHwgLmlkJwp9CnVudGlsIG91dHB1dD0kKGZpbmRfY29udGFpbmVyKT
sgW1sgLW4gIiRvdXRwdXQiIF1dOyBkbwogIGVjaG8gIndhaXRpbmcgZm9yIGNvbnRhaW5lciB
0byBjb21lIHVwIgogIHNsZWVwIDE7CmRvbmUKISBzdWRvIGNyaWN0bCBleGVjICRvdXRwdX
QgL2JpbmRhdGEvc2NyaXB0cy9iZjItc3dpdGNoLW1vZGUuc2ggIiRAIgo=
        mode: 0755
        overwrite: true
        path: /etc/default/switch_in_sriov_config_daemon.sh
    systemd:
      units:
      - name: dpu-switch.service
        enabled: true
        contents: |
          [Unit]
          Description=Switch BlueField2 card to NIC/DPU mode
          RequiresMountsFor=%t/containers
          Wants=network.target
          After=network-online.target kubelet.service
          [Service]
          SuccessExitStatus=0 120
          RemainAfterExit=True
          ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic || shutdown
-r now' (1)
          Type=oneshot
          [Install]
          WantedBy=multi-user.target
```

(1) オプションです。オプションで、特定のカードの PCI アドレスを指定することができます。 例えば **ExecStart=/bin/bash -c '/etc/default/switch_in_sriov_config_daemon.sh nic 0000:5e:00.0 || echo done'**。デフォルトでは、最初のデバイスが選択されています。複数のデバイスがある場合は、使用する PCI アドレスを指定する必要があります。Bluefield-2 を DPU モードから NIC モードに切り替えるすべてのノードで、PCI アドレスが同じである必要があります。

4. ワーカーノードが再起動するまで待ちます。再起動後、ワーカーノードの Bluefield-2 ネットワークデバイスは NIC モードに切り替わります。

関連情報

SR-IOV Network Operator のインストール

27.14. 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。

27.14.1. 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** 権限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 sriovibnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodepolicies.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworknodestates.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworkpoolconfigs.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovnetworks.sriovnetwork.openshift.io
```

```
$ oc delete crd sriovoperatorconfigs.sriovnetwork.openshift.io
```

4. SR-IOV 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28章 OVN-KUBERNETES ネットワークプラグイン

28.1. OVN-KUBERNETES ネットワークプラグインについて

OpenShift Container Platform クラスターは、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。

Red Hat OpenShift Networking の一部である OVN-Kubernetes ネットワークプラグインは、OpenShift Container Platform のデフォルト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 プラグインを使用するクラスターは、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。

注記

OVN-Kubernetes は、OpenShift Container Platform および単一ノードの OpenShift デプロイメントのデフォルトのネットワークソリューションです。

OVS プロジェクトから生まれた OVN-Kubernetes は、オープンフロールールなど、同じコンストラクトの多くを使用して、パケットがネットワークを通過する方法を決定します。詳細は、Open Virtual Network の Web サイト を参照してください。

OVN-Kubernetes は、仮想ネットワーク設定を **OpenFlow** ルールに変換する OVS 用の一連のデーモンです。**OpenFlow** は、ネットワークスイッチおよびルーターと通信するためのプロトコルであり、ネットワークデバイス上のネットワークトラフィックのフローをリモートで制御する手段を提供し、ネットワーク管理者がネットワークトラフィックのフローを設定、管理、および監視できるようにします。

OVN-Kubernetes は、**OpenFlow** では利用できない高度な機能をさらに提供します。OVN は、分散仮想ルーター、分散論理スイッチ、アクセス制御、DHCP および DNS をサポートします。OVN は、オープンフローと同等のロジックフロー内に分散仮想ルーターを実装します。たとえば、ネットワーク上に DHCP リクエストを送信する Pod がある場合、Pod はそのブロードキャストを送信して DHCP アドレスを探します。また、そのパケットに一致するロジックフロールールが存在し、応答としてゲートウェイ、DNS サーバー、IP アドレスなどを提供します。

OVN-Kubernetes は、各ノードでデーモンを実行します。すべてのノードで実行されるデータベースおよび OVN コントローラー用のデーモンセットがあります。OVN コントローラーは、ネットワークプロバイダーの機能 (egress IP、ファイアウォール、ルーター、ハイブリッドネットワーク、IPSEC 暗号化、IPv6、ネットワークポリシー、ネットワークポリシーログ、ハードウェアオフロード、およびマルチキャスト) をサポートするために、ノード上で Open vSwitch デーモンをプログラムします。

28.1.1. OVN-Kubernetes の目的

OVN-Kubernetes ネットワークプラグインは、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する、オープンソースのフル機能の Kubernetes CNI プラグインです。OVN はコミュニティーで開発され、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。OVN-Kubernetes ネットワークプラグイン:

- 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 はコミュニティーで開発され、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。

OVN-Kubernetes ネットワークプラグインは、OpenShift SDN よりも次の利点があります。

- サポートされているプラットフォームでの IPv6 シングルスタックおよび IPv4/IPv6 デュアルスタックネットワークの完全サポート
- Linux と Microsoft Windows の両方のワークロードによるハイブリッドクラスターのサポート
- クラスター内通信のオプションの IPsec 暗号化
- ホスト CPU から互換性のあるネットワークカードおよびデータ処理ユニット (DPU) へのネットワークデータ処理のオフロード

28.1.2. サポートされているネットワークプラグイン機能のマトリックス

Red Hat OpenShift Networking は、ネットワークプラグイン用に OpenShift SDN と OVN-Kubernetes の 2 つのオプションを提供します。以下の表は、両方のネットワークプラグインの現在の機能サポートをまとめたものです。

表28.1 デフォルトの CNI ネットワークプラグイン機能の比較

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
ハイブリッドネットワーク	サポート対象	サポート対象外
クラスター内通信の 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][4]	サポート対象外
Kubernetes ネットワークポリシー	サポート対象	サポート対象
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
ハードウェアのオフロード	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、vSphere、IBM Power®、IBM Z®、および Red Hat OpenStack クラスターでのみサポートされます。

4. IPv6 シングルスタックは、IBM Power®、IBM Z®、および Red Hat OpenStack クラスターではサポートされません。

28.1.3. OVN-Kubernetes IPv6 とデュアルスタックの制限

OVN-Kubernetes ネットワークプラグインには、次の制限があります。

- 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** デー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** デー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。

28.1.4. セッションアフィニティー

セッションアフィニティーは、Kubernetes **Service** オブジェクトに適用される機能です。<service_VIP>:<Port> に接続するたびに、トラフィックが常に同じバックエンドに負荷分散されるようにする場合は、**セッションアフィニティー** を使用できます。クライアントの IP アドレスに基づいてセッションアフィニティーを設定する方法など、詳細は、セッションアフィニティー を参照してください。

セッションアフィニティーのスティッキタイムアウト
OpenShift Container Platform の OVN-Kubernetes ネットワークプラグインは、最後のパケットに基づいて、クライアントからのセッションのスティッキタイムアウトを計算します。たとえば、**curl** コマンドを 10 回実行すると、スティッキーセッションタイマーは最初のパケットではなく 10 番目のパケット

から開始します。その結果、クライアントが継続的にサービスに接続している場合でも、セッションがタイムアウトすることはありません。タイムアウトは、**timeoutSeconds** パラメーターで設定された時間、サービスがパケットを受信しなかった場合に開始されます。

関連情報

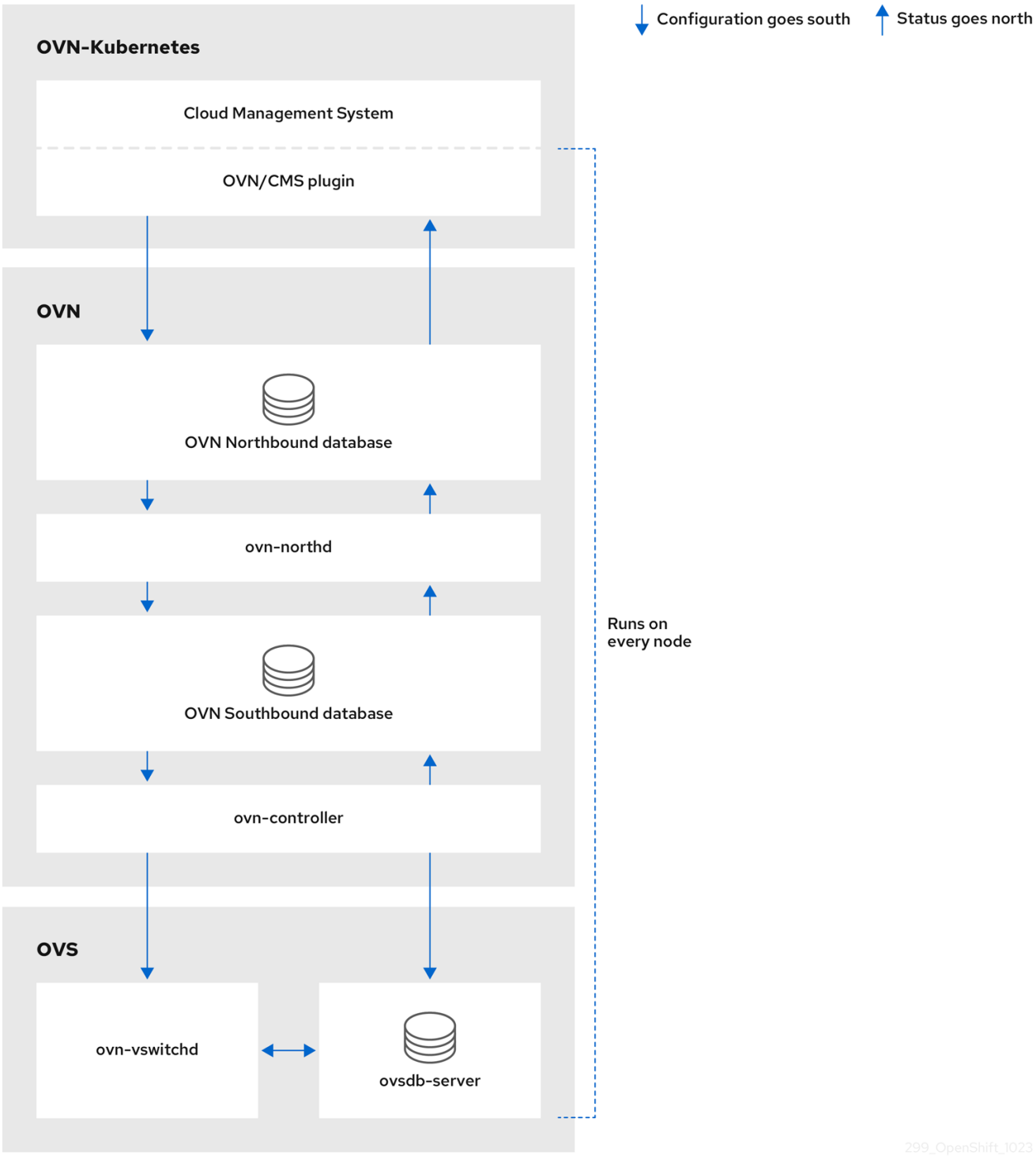
- プロジェクトの egress ファイアウォールの設定
- ネットワークポリシーについて
- ネットワークポリシーイベントのロギング
- プロジェクトのマルチキャストの有効化
- IPsec 暗号化の設定
- Network [operator.openshift.io/v1]

28.2. OVN-KUBERNETES のアーキテクチャー

28.2.1. OVN-Kubernetes のアーキテクチャーの紹介

次の図は、OVN-Kubernetes のアーキテクチャーを示しています。

図28.1 OVN-Kubernetes のアーキテクチャー



主なコンポーネントは次のとおりです。

- **Cloud Management System (CMS)** - OVN 統合用の CMS 固有のプラグインを提供する OVN 用のプラットフォーム固有のクライアント。このプラグインは、CMS 固有の形式で CMS 設定データベースに格納されているクラウド管理システムの論理ネットワーク設定の概念を、OVN が理解できる中間表現に変換します。
- **OVN ノースバウンドデータベース (nbdb) コンテナー** - CMS プラグインによって渡された論理ネットワーク設定を格納します。

- **OVN サウスバウンドデータベース (sbdb) コンテナー** - 各ノードの Open vSwitch (OVS) システムの物理および論理ネットワーク設定状態を、それらをバインドするテーブルを含めて格納します。

- **OVN North デーモン (ovn-northd)** - これは、**nbdb** コンテナーと **sbdb** コンテナーの間の仲介クライアントです。これは、論理ネットワーク設定を、**nbdb** コンテナーから取得した従来のネットワーク概念の観点から、その下の **sbdb** コンテナーの論理データパスフローに変換します。**ovn-northd** デーモンのコンテナー名は **Northd** で、**ovnkube-node** Pod 内で実行されます。

- **ovn-controller** - **sbdb** コンテナーに必要な情報または更新のために、OVS およびハイパーバイザーと対話する OVN エージェントです。**ovn-controller** は **sbdb** コンテナーから論理フローを読み取り、それらを **OpenFlow** フローに変換して、ノードの OVS デーモンに送信します。コンテナー名は **ovn-controller** で、**ovnkube-node** Pod で実行されます。

OVN Northd、ノースバウンドデータベース、および サウスバウンドデータベースはクラスター内の各ノードで実行され、主にそのノードのローカルにある情報で構成され、このような情報を処理します。

OVN ノースバウンドデータベースには、クラウド管理システム (CMS) によって渡された論理ネットワーク設定があります。OVN ノースバウンドデータベースには、ネットワークの現在の望ましい状態が含まれており、論理ポート、論理スイッチ、論理ルーターなどのコレクションとして提示されます。**ovn-northd** (**northd** コンテナー) は、OVN ノースバウンドデータベースと OVN サウスバウンドデータベースに接続します。これは、論理ネットワーク設定を、OVN ノースバウンドデータベースから取得した従来のネットワーク概念の観点から、OVN サウスバウンドデータベースの論理データパスフローに変換します。

OVN サウスバウンドデータベースには、ネットワークの物理的および論理的表現と、それらをリンクするバインディングテーブルがあります。これには、ノードのシャーシ情報と、クラスター内の他のノードに接続するために必要なリモート中継スイッチポートなどの他の設定要素が含まれます。OVN サウスバウンドデータベースには、すべてのロジックフローも含まれています。このロジックフローは、各ノードで実行される **ovn-controller** プロセスと共有され、**ovn-controller** はそれらを **OpenvSwitch** (OVS) をプログラムする **OpenFlow** ルールに変換します。

Kubernetes コントロールプレーンノードでは、別々のノードに 2 つの **ovnkube-control-plane** Pod が含まれています。これらの Pod は、クラスター内の各ノードに一元的な IP アドレス管理 (IPAM) 割り当てを実行します。どの時点でも、1 つの **ovnkube-control-plane** Pod がリーダーです。

28.2.2. OVN-Kubernetes プロジェクト内のすべてのリソースの一覧表示

OVN-Kubernetes プロジェクトで実行されるリソースとコンテナーを見つけることは、OVN-Kubernetes ネットワークの実装を理解するのに役立ちます。

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. 次のコマンドを実行して、OVN-Kubernetes プロジェクト内のすべてのリソース、エンドポイント、および **ConfigMap** を取得します。

```
$ oc get all,ep,cm -n openshift-ovn-kubernetes
```

出力例

```
Warning: apps.openshift.io/v1 DeploymentConfig is deprecated in v4.14+, unavailable in
v4.10000+
NAME                                     READY   STATUS    RESTARTS      AGE
pod/ovnkube-control-plane-65c6f55656-6d55h   2/2     Running   0             114m
pod/ovnkube-control-plane-65c6f55656-fd7vw   2/2     Running   2 (104m ago)   114m
pod/ovnkube-node-bcvts                       8/8     Running   0             113m
pod/ovnkube-node-drgvv                       8/8     Running   0             113m
pod/ovnkube-node-f2pxt                       8/8     Running   0             113m
pod/ovnkube-node-frqsb                       8/8     Running   0             105m
pod/ovnkube-node-lbxkk                       8/8     Running   0             105m
pod/ovnkube-node-tt7bx                       8/8     Running   1 (102m ago)   105m

NAME                                  TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)           AGE
service/ovn-kubernetes-control-plane   ClusterIP   None         <none>        9108/TCP
114m
service/ovn-kubernetes-node           ClusterIP   None         <none>        9103/TCP,9105/TCP
114m

NAME                         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE
NODE SELECTOR                AGE
daemonset.apps/ovnkube-node   6         6         6       6            6
beta.kubernetes.io/os=linux   114m

NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ovnkube-control-plane   3/3     3            3           114m

NAME                                             DESIRED   CURRENT   READY   AGE
replicaset.apps/ovnkube-control-plane-65c6f55656   3         3         3       114m

NAME                                   ENDPOINTS                                         AGE
endpoints/ovn-kubernetes-control-plane   10.0.0.3:9108,10.0.0.4:9108,10.0.0.5:9108
114m
endpoints/ovn-kubernetes-node            10.0.0.3:9105,10.0.0.4:9105,10.0.0.5:9105 + 9 more...
114m

NAME                                DATA   AGE
configmap/control-plane-status      1      113m
configmap/kube-root-ca.crt          1      114m
configmap/openshift-service-ca.crt   1      114m
configmap/ovn-ca                    1      114m
configmap/ovnkube-config            1      114m
configmap/signer-ca                 1      114m
```

クラスター内の各ノードには、1つの **ovnkube-node** Pod があります。**ovnkube-config** config map には、OpenShift Container Platform OVN-Kubernetes 設定が含まれています。

2. 次のコマンドを実行して、**ovnkube-node** Pod 内のすべてのコンテナーを一覧表示します。

```
$ oc get pods ovnkube-node-bcvts -o jsonpath='{.spec.containers[*].name}' -n openshift-ovn-
kubernetes
```

予想される出力

```
ovn-controller ovn-acl-logging kube-rbac-proxy-node kube-rbac-proxy-ovn-metrics northd
nbdb sbdb ovnkube-controller
```

ovnkube-node Pod は、複数のコンテナーで設定されています。これは、ノースバウンドデータベース (**nbdb** コンテナー)、サウスバウンドデータベース (**sbdb** コンテナー)、ノースデーモン (**northd** コンテナー)、**ovn-controller** および **ovnkube-controller** コンテナーをホストします。**ovnkube-controller** コンテナーは、Pod、Egress IP、namespace、サービス、エンドポイント、Egress ファイアウォール、ネットワークポリシーなどの API オブジェクトを監視します。また、そのノードの利用可能なサブネットプールから Pod IP への割り当ても行います。

3. 次のコマンドを実行して、**ovnkube-control-plane** Pod 内のすべてのコンテナーをリスト表示します。

```
$ oc get pods ovnkube-control-plane-65c6f55656-6d55h -o
jsonpath='{.spec.containers[*].name}' -n openshift-ovn-kubernetes
```

予想される出力

```
kube-rbac-proxy ovnkube-cluster-manager
```

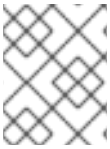
ovnkube-control-plane Pod には、各 OpenShift Container Platform ノードに常駐するコンテナー (**ovnkube-cluster-manager**) があります。**ovnkube-cluster-manager** コンテナーは、Pod サブネット、トランジットスイッチサブネット IP、およびジョインスイッチサブネット IP をクラスター内の各ノードに割り当てます。**kube-rbac-proxy** コンテナーは **ovnkube-cluster-manager** コンテナーのメトリックを監視します。

28.2.3. OVN-Kubernetes ノースバウンドデータベースの内容の一覧表示

各ノードは、そのノード上の **ovnkube-node** Pod で実行されている **ovnkube-controller** コンテナーによって制御されます。OVN 論理ネットワークエンティティーを理解するには、そのノード上の **ovnkube-node** Pod 内でコンテナーとして実行されているノースバウンドデータベースを調べて、表示するノード内にどのようなオブジェクトがあるかを確認する必要があります。

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

クラスター内で ovn **nbctl** または **sbctl** コマンドを実行するには、関連するノード上の **nbdb** または **sbdb** コンテナーにリモートシェルを開く必要があります。

1. 次のコマンドを実行して Pod をリスト表示します。

```
$ oc get po -n openshift-ovn-kubernetes
```

出力例

```
NAME                                  READY   STATUS    RESTARTS      AGE
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0             27m
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0             27m
ovnkube-node-55xs2                       8/8     Running   0             26m
ovnkube-node-7r84r                       8/8     Running   0             16m
ovnkube-node-bqq8p                       8/8     Running   0             17m
```

```
ovnkube-node-mkj4f                  8/8     Running   0          26m
ovnkube-node-mlr8k                  8/8     Running   0          26m
ovnkube-node-wqn2m                    8/8     Running   0          16m
```

2. オプション: Pod とノード情報をリストするには、次のコマンドを実行します。

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

出力例

```
NAME                                READY   STATUS    RESTARTS      AGE   IP          NODE
NOMINATED NODE   READINESS GATES
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0          27m   10.0.0.3    ci-ln-
t487nnb-72292-mdcnq-master-1        <none>          <none>
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0          27m   10.0.0.4    ci-ln-
t487nnb-72292-mdcnq-master-2        <none>          <none>
ovnkube-node-55xs2                  8/8     Running   0          26m   10.0.0.4    ci-ln-t487nnb-
72292-mdcnq-master-2        <none>          <none>
ovnkube-node-7r84r                  8/8     Running   0          17m   10.0.128.3   ci-ln-t487nnb-
72292-mdcnq-worker-b-wbz7z   <none>          <none>
ovnkube-node-bqq8p                  8/8     Running   0          17m   10.0.128.2   ci-ln-
t487nnb-72292-mdcnq-worker-a-lh7ms   <none>          <none>
ovnkube-node-mkj4f                  8/8     Running   0          27m   10.0.0.5    ci-ln-t487nnb-
72292-mdcnq-master-0        <none>          <none>
ovnkube-node-mlr8k                  8/8     Running   0          27m   10.0.0.3    ci-ln-t487nnb-
72292-mdcnq-master-1        <none>          <none>
ovnkube-node-wqn2m                    8/8     Running   0          17m   10.0.128.4   ci-ln-
t487nnb-72292-mdcnq-worker-c-przlm   <none>          <none>
```

3. 次のコマンドを実行して、Pod に移動してノースバウンドデータベースを確認します。

```
$ oc rsh -c nbdb -n openshift-ovn-kubernetes ovnkube-node-55xs2
```

4. 次のコマンドを実行して、ノースバウンドデータベース内のすべてのオブジェクトを表示します。

```
$ ovn-nbctl show
```

出力は長すぎてここにリストできません。リストには、NAT ルール、論理スイッチ、ロードバランサーなどが含まれます。

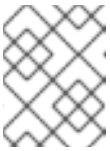
次の任意のコマンドのいくつかを使用すると、特定のコンポーネントに絞り込むことができます。

a. 次のコマンドを実行して、論理ルーターのリストを表示します。

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c northd -- ovn-nbctl lr-list
```

出力例

```
45339f4f-7d0b-41d0-b5f9-9fca9ce40ce6 (GR_ci-ln-t487nnb-72292-mdcnq-master-2)
96a0a0f0-e7ed-4fec-8393-3195563de1b8 (ovn_cluster_router)
```



注記

この出力から、各ノードにルーターと **ovn_cluster_router** があることがわかります。

b. 次のコマンドを実行して、論理スイッチのリストを表示します。

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb -- ovn-nbctl ls-list
```

出力例

```
bdd7dc3d-d848-4a74-b293-cc15128ea614 (ci-ln-t487nnb-72292-mdcnq-master-2)
b349292d-ee03-4914-935f-1940b6cb91e5 (ext_ci-ln-t487nnb-72292-mdcnq-master-2)
0aac0754-ea32-4e33-b086-35eeabf0a140 (join)
992509d7-2c3f-4432-88db-c179e43592e5 (transit_switch)
```

注記

この出力から、各ノードの ext スイッチに加えて、ノード名自体を持つスイッチと結合スイッチがあることがわかります。

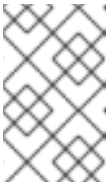
c. 次のコマンドを実行して、ロードバランサーのリストを表示します。

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb -- ovn-nbctl lb-list
```

出力例

```
UUID                                    LB                  PROTO      VIP                  IPs
7c84c673-ed2a-4436-9a1f-9bc5dd181eea    Service_default/    tcp        172.30.0.1:443
10.0.0.3:6443,169.254.169.2:6443,10.0.0.5:6443
4d663fd9-ddc8-4271-b333-4c0e279e20bb    Service_default/    tcp        172.30.0.1:443
10.0.0.3:6443,10.0.0.4:6443,10.0.0.5:6443
292eb07f-b82f-4962-868a-4f541d250bca    Service_openshif    tcp
172.30.105.247:443      10.129.0.12:8443
034b5a7f-bb6a-45e9-8e6d-573a82dc5ee3    Service_openshif    tcp
172.30.192.38:443       10.0.0.3:10259,10.0.0.4:10259,10.0.0.5:10259
a68bb53e-be84-48df-bd38-bdd82fcd4026    Service_openshif    tcp
172.30.161.125:8443     10.129.0.32:8443
6cc21b3d-2c54-4c94-8ff5-d8e017269c2e    Service_openshif    tcp        172.30.3.144:443
10.129.0.22:8443
37996ffd-7268-4862-a27f-61cd62e09c32    Service_openshif    tcp
172.30.181.107:443      10.129.0.18:8443
81d4da3c-f811-411f-ae0c-bc6713d0861d    Service_openshif    tcp
172.30.228.23:443       10.129.0.29:8443
ac5a4f3b-b6ba-4ceb-82d0-d84f2c41306e    Service_openshif    tcp
172.30.14.240:9443      10.129.0.36:9443
c88979fb-1ef5-414b-90ac-43b579351ac9    Service_openshif    tcp
172.30.231.192:9001
10.128.0.5:9001,10.128.2.5:9001,10.129.0.5:9001,10.129.2.4:9001,10.130.0.3:9001,10.13
1.0.3:9001
fcb0a3fb-4a77-4230-a84a-be45dce757e8    Service_openshif    tcp
```

```
172.30.189.92:443       10.130.0.17:8440
67ef3e7b-ceb9-4bf0-8d96-b43bde4c9151    Service_openshif    tcp
172.30.67.218:443       10.129.0.9:8443
d0032fba-7d5e-424a-af25-4ab9b5d46e81    Service_openshif    tcp
172.30.102.137:2379     10.0.0.3:2379,10.0.0.4:2379,10.0.0.5:2379
                                        tcp         172.30.102.137:9979
10.0.0.3:9979,10.0.0.4:9979,10.0.0.5:9979
7361c537-3eec-4e6c-bc0c-0522d182abd4    Service_openshif    tcp
172.30.198.215:9001
10.0.0.3:9001,10.0.0.4:9001,10.0.0.5:9001,10.0.128.2:9001,10.0.128.3:9001,10.0.128.4:9
001
0296c437-1259-410b-a6fd-81c310ad0af5    Service_openshif    tcp
172.30.198.215:9001
10.0.0.3:9001,169.254.169.2:9001,10.0.0.5:9001,10.0.128.2:9001,10.0.128.3:9001,10.0.1
28.4:9001
5d5679f5-45b8-479d-9f7c-08b123c688b8    Service_openshif    tcp
172.30.38.253:17698     10.128.0.52:17698,10.129.0.84:17698,10.130.0.60:17698
2adcbab4-d1c9-447d-9573-b5dc9f2efbfa    Service_openshif    tcp
172.30.148.52:443       10.0.0.4:9202,10.0.0.5:9202
                                        tcp         172.30.148.52:444
10.0.0.4:9203,10.0.0.5:9203
                                        tcp         172.30.148.52:445
10.0.0.4:9204,10.0.0.5:9204
                                        tcp         172.30.148.52:446
10.0.0.4:9205,10.0.0.5:9205
2a33a6d7-af1b-4892-87cc-326a380b809b    Service_openshif    tcp
172.30.67.219:9091      10.129.2.16:9091,10.131.0.16:9091
                                        tcp         172.30.67.219:9092
10.129.2.16:9092,10.131.0.16:9092
                                        tcp         172.30.67.219:9093
10.129.2.16:9093,10.131.0.16:9093
                                        tcp         172.30.67.219:9094
10.129.2.16:9094,10.131.0.16:9094
f56f59d7-231a-4974-99b3-792e2741ec8d    Service_openshif    tcp
172.30.89.212:443       10.128.0.41:8443,10.129.0.68:8443,10.130.0.44:8443
08c2c6d7-d217-4b96-b5d8-c80c4e258116    Service_openshif    tcp
172.30.102.137:2379     10.0.0.3:2379,169.254.169.2:2379,10.0.0.5:2379
                                        tcp         172.30.102.137:9979
10.0.0.3:9979,169.254.169.2:9979,10.0.0.5:9979
60a69c56-fc6a-4de6-bd88-3f2af5ba5665    Service_openshif    tcp
172.30.10.193:443       10.129.0.25:8443
ab1ef694-0826-4671-a22c-565fc2d282ec    Service_openshif    tcp
172.30.196.123:443      10.128.0.33:8443,10.129.0.64:8443,10.130.0.37:8443
b1fb34d3-0944-4770-9ee3-2683e7a630e2    Service_openshif    tcp
172.30.158.93:8443      10.129.0.13:8443
95811c11-56e2-4877-be1e-c78ccb3a82a9    Service_openshif    tcp
172.30.46.85:9001       10.130.0.16:9001
4baba1d1-b873-4535-884c-3f6fc07a50fd    Service_openshif    tcp         172.30.28.87:443
10.129.0.26:8443
6c2e1c90-f0ca-484e-8a8e-40e71442110a    Service_openshif    udp         172.30.0.10:53
10.128.0.13:5353,10.128.2.6:5353,10.129.0.39:5353,10.129.2.6:5353,10.130.0.11:5353,1
0.131.0.9:5353
```



注記

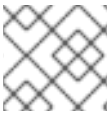
この出力 (一部省略あり) から、多くの OVN-Kubernetes ロードバランサーがあることがわかります。OVN-Kubernetes のロードバランサーはサービスの表現です。

5. 次のコマンドを実行して、コマンド **ovn-nbctl** で使用可能なオプションを表示します。

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c nbdb ovn-nbctl --help
```

28.2.4. ノースバウンドデータベースの内容を調べるための ovn-nbctl のコマンドライン引数

次の表に、ノースバウンドデータベースの内容を調べるために **ovn-nbctl** で使用できるコマンドライン引数を示します。



注記

内容を表示する Pod でリモートシェルを開き、**ovn-nbctl** コマンドを実行します。

表28.2 ノースバウンドデータベースの内容を調べるためのコマンドライン引数

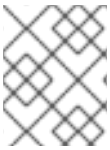
引数	説明
ovn-nbctl show	特定のノードから見たノースバウンドデータベースの内容の概要。
ovn-nbctl show <switch_or_router>	指定されたスイッチまたはルーターに関連付けられた詳細を表示します。
ovn-nbctl lr-list	論理ルーターを表示します。
ovn-nbctl lrp-list <router>	**ovn-nbctl lr-list** からのルーター情報を使用して、ルーターポートを表示します。
ovn-nbctl lr-nat-list <router>	指定されたルーターのネットワークアドレス変換の詳細を表示します。
ovn-nbctl ls-list	論理スイッチを表示します。
ovn-nbctl lsp-list <switch>	**ovn-nbctl ls-list** からのスイッチ情報を使用して、スイッチポートを表示します。
ovn-nbctl lsp-get-type <port>	論理ポートのタイプを取得します。
ovn-nbctl lb-list	ロードバランサーを表示します。

28.2.5. OVN-Kubernetes サウスバウンドデータベースの内容の一覧表示

各ノードは、そのノード上の **ovnkube-node** Pod で実行されている **ovnkube-controller** コンテナーによって制御されます。OVN 論理ネットワークエンティティーを理解するには、そのノード上の **ovnkube-node** Pod 内でコンテナーとして実行されているノースバウンドデータベースを調べて、表示するノード内にどのようなオブジェクトがあるかを確認する必要があります。

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

クラスター内で ovn **nbctl** または **sbctl** コマンドを実行するには、関連するノード上の **nbdb** または **sbdb** コンテナーにリモートシェルを開く必要があります。

1. 次のコマンドを実行して、Pod を一覧表示します。

```
$ oc get po -n openshift-ovn-kubernetes
```

出力例

```
NAME                                  READY   STATUS    RESTARTS      AGE
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0             27m
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0             27m
ovnkube-node-55xs2                       8/8     Running   0             26m
ovnkube-node-7r84r                       8/8     Running   0             16m
ovnkube-node-bqq8p                       8/8     Running   0             17m
ovnkube-node-mkj4f                       8/8     Running   0             26m
ovnkube-node-mlr8k                       8/8     Running   0             26m
ovnkube-node-wqn2m                       8/8     Running   0             16m
```

2. オプション: Pod とノード情報をリストするには、次のコマンドを実行します。

```
$ oc get pods -n openshift-ovn-kubernetes -owide
```

出力例

```
NAME                                  READY   STATUS    RESTARTS      AGE   IP           NODE
NOMINATED NODE   READINESS GATES
ovnkube-control-plane-8444dff7f9-4lh9k   2/2     Running   0             27m   10.0.0.3     ci-ln-
t487nnb-72292-mdcnq-master-1   <none>           <none>
ovnkube-control-plane-8444dff7f9-5rjh9   2/2     Running   0             27m   10.0.0.4     ci-ln-
t487nnb-72292-mdcnq-master-2   <none>           <none>
ovnkube-node-55xs2                       8/8     Running   0             26m   10.0.0.4     ci-ln-t487nnb-
72292-mdcnq-master-2   <none>           <none>
ovnkube-node-7r84r                       8/8     Running   0             17m   10.0.128.3   ci-ln-t487nnb-
72292-mdcnq-worker-b-wbz7z   <none>           <none>
ovnkube-node-bqq8p                       8/8     Running   0             17m   10.0.128.2   ci-ln-
t487nnb-72292-mdcnq-worker-a-lh7ms   <none>           <none>
ovnkube-node-mkj4f                       8/8     Running   0             27m   10.0.0.5     ci-ln-t487nnb-
72292-mdcnq-master-0   <none>           <none>
ovnkube-node-mlr8k                       8/8     Running   0             27m   10.0.0.3     ci-ln-t487nnb-
```

```
72292-mdcnq-master-1        <none>          <none>
ovnkube-node-wqn2m                  8/8     Running   0          17m   10.0.128.4   ci-ln-
t487nnb-72292-mdcnq-worker-c-przlm   <none>          <none>
```

3. Pod に移動してサウスバウンドデータベースを確認します。

```
$ oc rsh -c sbdb -n openshift-ovn-kubernetes ovnkube-node-55xs2
```

4. 次のコマンドを実行して、サウスバウンドデータベース内のすべてのオブジェクトを表示します。

```
$ ovn-sbctl show
```

出力例

```
Chassis "5db31703-35e9-413b-8cdf-69e7eecb41f7"
    hostname: ci-ln-9gp362t-72292-v2p94-worker-a-8bmwz
    Encap geneve
        ip: "10.0.128.4"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-worker-a-8bmwz
Chassis "070debed-99b7-4bce-b17d-17e720b7f8bc"
    hostname: ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Encap geneve
        ip: "10.0.128.2"
        options: {csum="true"}
    Port_Binding k8s-ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding rtoe-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding openshift-monitoring_alertmanager-main-1
    Port_Binding rtoj-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding etor-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding cr-rtos-ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding openshift-e2e-loki_loki-promtail-qcrcz
    Port_Binding jtor-GR_ci-ln-9gp362t-72292-v2p94-worker-b-svmp6
    Port_Binding openshift-multus_network-metrics-daemon-mkd4t
    Port_Binding openshift-ingress-canary_ingress-canary-xtvj4
    Port_Binding openshift-ingress_router-default-6c76cbc498-pvlqk
    Port_Binding openshift-dns_dns-default-zz582
    Port_Binding openshift-monitoring_thanos-querier-57585899f5-lbf4f
    Port_Binding openshift-network-diagnostics_network-check-target-tn228
    Port_Binding openshift-monitoring_prometheus-k8s-0
    Port_Binding openshift-image-registry_image-registry-68899bd877-xqxjj
Chassis "179ba069-0af1-401c-b044-e5ba90f60fea"
    hostname: ci-ln-9gp362t-72292-v2p94-master-0
    Encap geneve
        ip: "10.0.0.5"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-master-0
Chassis "68c954f2-5a76-47be-9e84-1cb13bd9dab9"
    hostname: ci-ln-9gp362t-72292-v2p94-worker-c-mjf9w
    Encap geneve
        ip: "10.0.128.3"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-worker-c-mjf9w
Chassis "2de65d9e-9abf-4b6e-a51d-a1e038b4d8af"
```

```
    hostname: ci-ln-9gp362t-72292-v2p94-master-2
    Encap geneve
        ip: "10.0.0.4"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-master-2
Chassis "1d371cb8-5e21-44fd-9025-c4b162cc4247"
    hostname: ci-ln-9gp362t-72292-v2p94-master-1
    Encap geneve
        ip: "10.0.0.3"
        options: {csum="true"}
    Port_Binding tstor-ci-ln-9gp362t-72292-v2p94-master-1
```

この詳細な出力は、シャーシとシャーシに接続されているポート (この場合、すべてのルーターポートとホストネットワークのように動作するもの) を示しています。すべての Pod は、ソースネットワークアドレス変換 (SNAT) を使用して、より広いネットワークと通信します。Pod の IP アドレスは、Pod が実行されているノードの IP アドレスに変換され、ネットワークに送信されます。

シャーシ情報に加えて、サウスバウンドデータベースにはすべてのロジックフローがあります。これらのロジックフローは各ノードで実行されている **ovn-controller** に送信されます。**ovn-controller** は、ロジックフローをオープンフロールールに変換し、最終的に **OpenvSwitch** をプログラムして、Pod がオープンフロールールに従ってネットワークの外に出られるようにします。

5. 次のコマンドを実行して、コマンド **ovn-sbctl** で使用可能なオプションを表示します。

```
$ oc exec -n openshift-ovn-kubernetes -it ovnkube-node-55xs2 \
-c sbdb ovn-sbctl --help
```

28.2.6. サウスバウンドデータベースの内容を調べるための ovn-sbctl のコマンドライン引数

次の表に、サウスバウンドデータベースの内容を調べるために **ovn-sbctl** で使用できるコマンドライン引数を示します。

注記

内容を表示する Pod でリモートシェルを開き、**ovn-sbctl** コマンドを実行します。

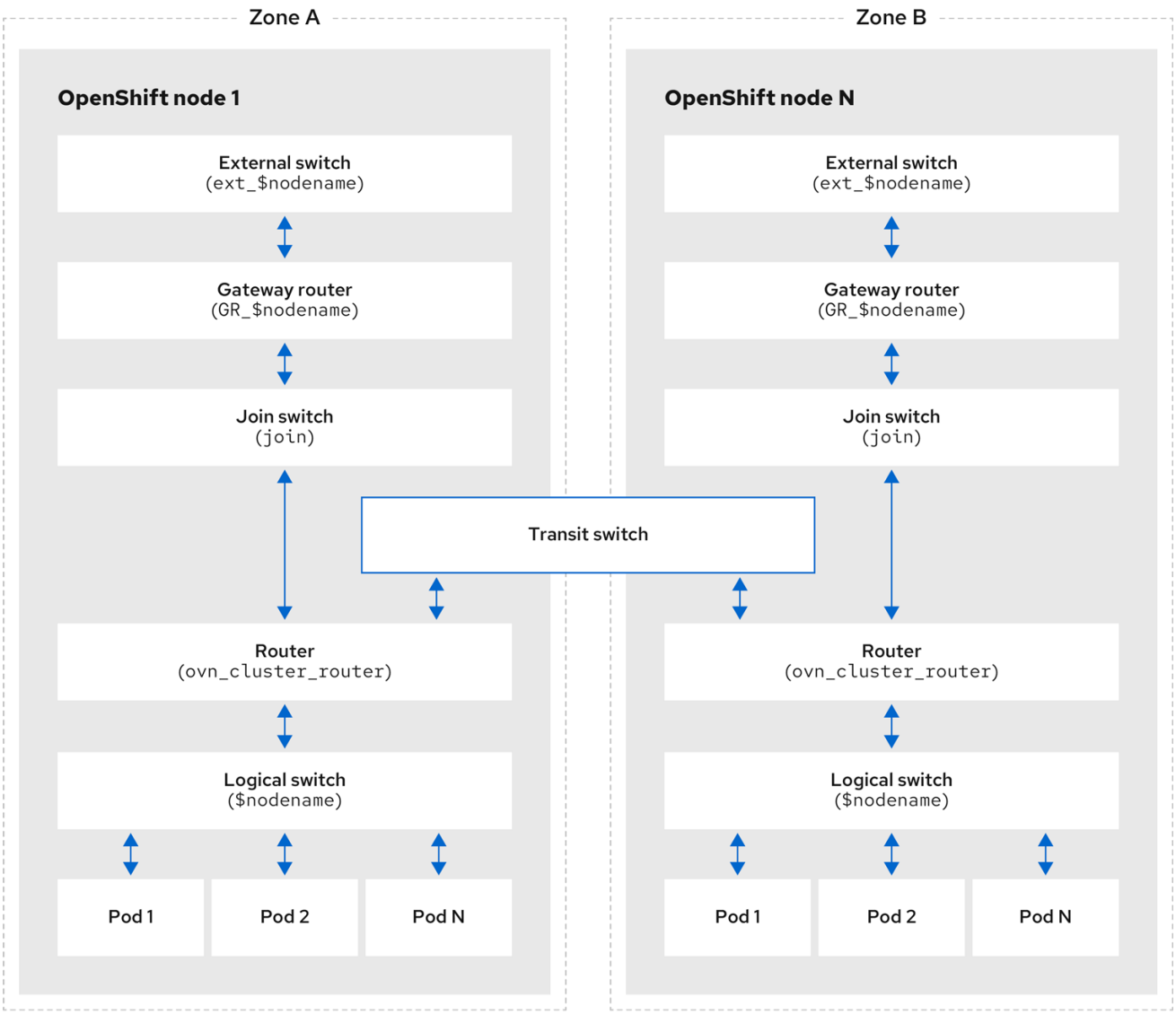
表28.3 サウスバウンドデータベースの内容を調べるためのコマンドライン引数

引数	説明
ovn-sbctl show	特定のノードから見たサウスバウンドデータベースの内容の概要。
ovn-sbctl list Port_Binding <port>	指定されたポートのサウスバウンドデータベースの内容を一覧表示します。
ovn-sbctl dump-flows	論理フローを一覧表示します。

28.2.7. OVN-Kubernetes の論理アーキテクチャー

OVN はネットワーク仮想化ソリューションです。OVN は論理スイッチとルーターを作成します。これらのスイッチとルーターは相互接続され、任意のネットワークトポロジーを作成します。ログレベルを 2 または 5 に設定して **ovnkube-trace** を実行すると、OVN-Kubernetes 論理コンポーネントが公開されます。以下の図は、ルーターとスイッチが OpenShift Container Platform でどのように接続されているかを示しています。

図28.2 OVN-Kubernetes のルーターおよびスイッチコンポーネント



パケット処理に関係する主要なコンポーネントは次のとおりです。

ゲートウェイルーター

L3 ゲートウェイルーターとも呼ばれるゲートウェイルーターは、通常、分散ルーターと物理ネットワークの間で使用されます。論理パッチポートを含むゲートウェイルーターは、(分散されていない) 物理的な場所またはシャーシにバインドされます。このルーターのパッチポートは、ovn-southbound データベース (**ovn-sbdb**) では l3gateway ポートと呼ばれます。

分散論理ルーター

分散論理ルーターと、仮想マシンとコンテナーが接続されるその背後にある論理スイッチは、事実上、各ハイパーバイザーに常駐します。

結合ローカルスイッチ

結合ローカルスイッチは、分散ルーターとゲートウェイルーターを接続するために使用されます。これにより、分散ルーターで必要な IP アドレスの数が減ります。

パッチポートを備えた論理スイッチ

パッチポートを備えた論理スイッチは、ネットワークスタックを仮想化するために使用されます。これらは、トンネルを介してリモート論理ポートを接続します。

localnet ポートを備えた論理スイッチ

localnet ポートを備えた論理スイッチは、OVN を物理ネットワークに接続するために使用されます。これらは、localnet ポートを使用して直接接続された物理 L2 セグメントにパケットをブリッジすることにより、リモート論理ポートを接続します。

パッチポート

パッチポートは、論理スイッチと論理ルーターの間、およびピア論理ルーター間の接続を表します。1 つの接続には、このような接続ポイントごとに、両側に 1 つずつ、1 組のパッチポートがあります。

l3gateway ポート

l3gateway ポートは、ゲートウェイルーターで使用される論理パッチポートの **ovn-sbdb** 内のポートバインディングエントリーです。これらのポートは、ゲートウェイルーター本体と同様にシャーシにバインドされているという事実を表すために、パッチポートではなく l3gateway ポートと呼ばれます。

localnet ポート

localnet ポートは、各 **ovn-controller** インスタンスからローカルにアクセス可能なネットワークへの接続を可能にするブリッジ論理スイッチに存在します。これは、論理スイッチから物理ネットワークへの直接接続をモデル化するのに役立ちます。論理スイッチに接続できる localnet ポートは 1 つだけです。

28.2.7.1. ローカルホストへの network-tools のインストール

ローカルホストに **network-tools** をインストールして、OpenShift Container Platform クラスターネットワークの問題をデバッグするための一連のツールを使用できるようにします。

手順

1. 次のコマンドを使用して、**network-tools** リポジトリーのクローンをワークステーションに作成します。

```
$ git clone git@github.com:openshift/network-tools.git
```

2. クローン作成したリポジトリーのディレクトリーに移動します。

```
$ cd network-tools
```

3. オプション: 使用可能なすべてのコマンドをリストします。

```
$ ./debug-scripts/network-tools -h
```

28.2.7.2. network-tools の実行

network-tools を実行して、論理スイッチとルーターに関する情報を取得します。

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クラスターにログインしている。
- ローカルホストに **network-tools** がインストールされている。

手順

1. 次のコマンドを実行して、Pod へのリモートシェルを開きます。

```
$ oc rsh -n openshift-ovn-kubernetes ovnkube-node-2hsbt
```

2. 次のコマンドを実行して、ルーターを一覧表示します。

```
$ ./debug-scripts/network-tools ovn-db-run-command ovn-nbctl lr-list
```

出力例

```
944a7b53-7948-4ad2-a494-82b55eeccf87 (GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99)
84bd4a4c-4b0b-4a47-b0cf-a2c32709fc53 (ovn_cluster_router)
```

3. 次のコマンドを実行して、localnet ポートを一覧表示します。

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=localnet
```

出力例

```
_uuid               : d05298f5-805b-4838-9224-1211afc2f199
additional_chassis  : []
additional_encap    : []
chassis             : []
datapath            : f3c2c959-743b-4037-854d-26627902597c
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : br-ex_ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac                 : [unknown]
mirror_rules        : []
nat_addresses       : []
options             : {network_name=physnet}
parent_port         : []
port_security       : []
requested_additional_chassis: []
requested_chassis   : []
tag                 : []
tunnel_key          : 2
type                : localnet
up                  : false
virtual_parent      : []

[...]
```

4. 次のコマンドを実行して、**l3gateway** ポートを一覧表示します。

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=l3gateway
```

出力例

```
_uuid               : 5207a1f3-1cf3-42f1-83e9-387bbb06b03c
additional_chassis  : []
additional_encap    : []
chassis             : ca6eb600-3a10-4372-a83e-e0d957c4cd92
datapath            : f3c2c959-743b-4037-854d-26627902597c
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : etor-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac                 : ["42:01:0a:00:80:04"]
mirror_rules        : []
nat_addresses       : ["42:01:0a:00:80:04 10.0.128.4"]
options             : {l3gateway-chassis="84737c36-b383-4c83-92c5-2bd5b3c7e772", peer=rtoe-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99}
parent_port         : []
port_security       : []
requested_additional_chassis: []
requested_chassis   : []
tag                 : []
tunnel_key          : 1
type                : l3gateway
up                  : true
virtual_parent      : []

_uuid               : 6088d647-84f2-43f2-b53f-c9d379042679
additional_chassis  : []
additional_encap    : []
chassis             : ca6eb600-3a10-4372-a83e-e0d957c4cd92
datapath            : dc9cea00-d94a-41b8-bdb0-89d42d13aa2e
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : jtor-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac                 : [router]
mirror_rules        : []
nat_addresses       : []
options             : {l3gateway-chassis="84737c36-b383-4c83-92c5-2bd5b3c7e772", peer=rtoj-GR_ci-ln-54932yb-72292-kd676-worker-c-rzj99}
parent_port         : []
port_security       : []
requested_additional_chassis: []
requested_chassis   : []
tag                 : []
tunnel_key          : 2
type                : l3gateway
up                  : true
```

```
virtual_parent      : []

[...]
```

5. 次のコマンドを実行して、パッチポートを一覧表示します。

```
$ ./debug-scripts/network-tools ovn-db-run-command \
ovn-sbctl find Port_Binding type=patch
```

出力例

```
_uid               : 785fb8b6-ee5a-4792-a415-5b1cb855dac2
additional_chassis  : []
additional_encap    : []
chassis             : []
datapath            : f1ddd1cc-dc0d-43b4-90ca-12651305acec
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : stor-ci-ln-54932yb-72292-kd676-worker-c-rzj99
mac                 : [router]
mirror_rules        : []
nat_addresses       : ["0a:58:0a:80:02:01 10.128.2.1 is_chassis_resident(\"cr-rtos-ci-ln-
54932yb-72292-kd676-worker-c-rzj99\")"]
options             : {peer=rtos-ci-ln-54932yb-72292-kd676-worker-c-rzj99}
parent_port         : []
port_security       : []
requested_additional_chassis: []
requested_chassis   : []
tag                 : []
tunnel_key          : 1
type                : patch
up                  : false
virtual_parent      : []

_uid               : c01ff587-21a5-40b4-8244-4cd0425e5d9a
additional_chassis  : []
additional_encap    : []
chassis             : []
datapath            : f6795586-bf92-4f84-9222-efe4ac6a7734
encap               : []
external_ids        : {}
gateway_chassis     : []
ha_chassis_group    : []
logical_port        : rtoj-ovn_cluster_router
mac                 : ["0a:58:64:40:00:01 100.64.0.1/16"]
mirror_rules        : []
nat_addresses       : []
options             : {peer=jtor-ovn_cluster_router}
parent_port         : []
port_security       : []
requested_additional_chassis: []
requested_chassis   : []
tag                 : []
```

```
tunnel_key          : 1
type                : patch
up                  : false
virtual_parent      : []
[...]
```

28.2.8. 関連情報

- ovnkube-trace を使用した Openflow のトレース
- OVN のアーキテクチャー
- ovn-nbctl linux man ページ
- ovn-sbctl linux man ページ

28.3. OVN-KUBERNETES のトラブルシューティング

OVN-Kubernetes には、組み込みのヘルスチェックとログのソースが多数あります。以下のセクションの手順に従ってクラスターを調査してください。サポートケースが必要な場合は、サポートガイド に従って、**must-gather** を使用して追加情報を収集してください。**-- gather_network_logs** は、サポートから指示された場合にのみ使用してください。

28.3.1. readiness プローブを使用した OVN-Kubernetes の正常性の監視

ovnkube-control-plane および **ovnkube-node** Pod には、readiness プローブが設定されたコンテナーがあります。

前提条件

- OpenShift CLI (**oc**) へのアクセスがある。
- **cluster-admin** 権限でクラスターにアクセスできる。
- **jq** がインストールされている。

手順

1. 次のコマンドを実行して、**ovnkube-node** readiness プローブの詳細を確認します。

```
$ oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-node \
-o json | jq '.items[0].spec.containers[] | .name,.readinessProbe'
```

ovnkube-node Pod 内のノースバウンドおよびサウスバウンドのデータベースコンテナーの Readiness プローブは、データベースと **ovnkube-controller** コンテナーの正常性をチェックします。

ovnkube-node Pod の **ovnkube-controller** コンテナーには、OVN-Kubernetes CNI 設定ファイルの存在を確認するための readiness プローブがあります。この設定ファイルがない場合、Pod が実行中でないか、Pod を設定するリクエストを受け入れる準備ができていません。

2. 次のコマンドを使用して、プローブの失敗を含む namespace のすべてのイベントを表示します。

```
$ oc get events -n openshift-ovn-kubernetes
```

3. 特定の Pod のみのイベントを表示します。

```
$ oc describe pod ovnkube-node-9lqfk -n openshift-ovn-kubernetes
```

4. クラスターネットワーク Operator からのメッセージとステータスを表示します。

```
$ oc get co/network -o json | jq '.status.conditions[]'
```

5. 次のスクリプトを実行して、**ovnkube-node** Pod 内の各コンテナーの **ready** ステータスを表示します。

```
$ for p in $(oc get pods --selector app=ovnkube-node -n openshift-ovn-kubernetes \
-o jsonpath='{range.items[*]}{" "}{.metadata.name}'); do echo === $p ===;  \
oc get pods -n openshift-ovn-kubernetes $p -o json | jq '.status.containerStatuses[] | .name, .ready'; \
done
```

注記

すべてのコンテナーのステータスが **true** として報告されることが期待されます。readiness プローブが失敗すると、ステータスが **false** に設定されます。

関連情報

- ヘルスチェックの使用によるアプリケーションの正常性の監視

28.3.2. コンソールでの OVN-Kubernetes アラートの表示

アラート UI は、アラートおよびそれらを規定するアラートルールおよびサイレンスについての詳細情報を提供します。

前提条件

- 開発者として、またはメトリクスで表示しているプロジェクトの表示パーミッションを持つユーザーとしてクラスターへのアクセスがある。

手順 (UI)

1. **Administrator** パースペクティブで、**Observe → Alerting** を選択します。このパースペクティブのアラート UI の主なページには、**Alerts**、**Silences**、および **Alerting Rules** という 3 つのページがあります。

2. **Observe → Alerting → Alerting Rules** を選択して、OVN-Kubernetes アラートのルールを表示します。

28.3.3. CLI での OVN-Kubernetes アラートの表示

コマンドラインから、アラートとその管理アラートルールおよびサイレンスに関する情報を取得できます。

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。
- **jq** がインストールされている。

手順

1. 次のコマンドを実行して、アクティブまたは発生中のアラートを表示します。
   a. 次のコマンドを実行して、アラートマネージャーのルート環境変数を設定します。

```
$ ALERT_MANAGER=$(oc get route alertmanager-main -n openshift-monitoring \
-o jsonpath='{@.spec.host}')
```

   b. 以下のコマンドを実行してアラートマネージャールート API に **curl** リクエストを実行します。**$ALERT_MANAGER** は、**Alertmanager** インスタンスの URL に置き換えます。

```
$ curl -s -k -H "Authorization: Bearer $(oc create token prometheus-k8s -n openshift-
monitoring)" https://$ALERT_MANAGER/api/v1/alerts | jq '.data[] | "\(.labels.severity) \
(.labels.alertname) \(.labels.pod) \(.labels.container) \(.labels.endpoint) \
(.labels.instance)"'
```

2. 次のコマンドを実行して、アラートルールを表示します。

```
$ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -s
'http://localhost:9090/api/v1/rules' | jq '.data.groups[].rules[] | select(((.name|contains("ovn"))
or (.name|contains("OVN")) or (.name|contains("Ovn")) or (.name|contains("North")) or
(.name|contains("South"))) and .type=="alerting")'
```

28.3.4. CLI を使用した OVN-Kubernetes ログの表示

OpenShift CLI (**oc**) を使用して、**ovnkube-master** および **ovnkube-node** Pod 内の各 Pod のログを表示できます。

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**) へのアクセスがある。
- **jq** がインストールされている。

手順

1. 特定の Pod のログを表示します。

```
$ oc logs -f <pod_name> -c <container_name> -n <namespace>
```

   ここでは、以下のようになります。

   -f

オプション: ログに書き込まれている内容に沿って出力することを指定します。

<pod_name>

Pod の名前を指定します。

<container_name>

オプション: コンテナーの名前を指定します。Pod に複数のコンテナーがある場合、コンテナー名を指定する必要があります。

<namespace>

Pod が実行されている namespace を指定します。

以下に例を示します。

```
$ oc logs ovnkube-node-5dx44 -n openshift-ovn-kubernetes
```

```
$ oc logs -f ovnkube-node-5dx44 -c ovnkube-controller -n openshift-ovn-kubernetes
```

ログファイルの内容が出力されます。

2. **ovnkube-node** Pod 内のすべてのコンテナーの最新のエントリーを調べます。

```
$ for p in $(oc get pods --selector app=ovnkube-node -n openshift-ovn-kubernetes \
-o jsonpath='{range.items[*]}{" "}{.metadata.name}'); \
do echo === $p ===; for container in $(oc get pods -n openshift-ovn-kubernetes $p \
-o json | jq -r '.status.containerStatuses[] | .name');do echo ---$container---; \
oc logs -c $container $p -n openshift-ovn-kubernetes --tail=5; done; done
```

3. 次のコマンドを使用して、**ovnkube-node** Pod 内のすべてのコンテナーのすべてのログの最後の 5 行を表示します。

```
$ oc logs -l app=ovnkube-node -n openshift-ovn-kubernetes --all-containers --tail 5
```

28.3.5. Web コンソールを使用した OVN-Kubernetes ログの表示

Web コンソールで、**ovnkube-master** Pod と **ovnkube-node** Pod の各 Pod のログを表示できます。

前提条件

- OpenShift CLI (**oc**) へのアクセスがある。

手順

1. OpenShift Container Platform コンソールで **Workloads → Pods** に移動するか、調査するリソースから Pod に移動します。
2. ドロップダウンメニューから **openshift-ovn-kubernetes** プロジェクトを選択します。
3. 調査する Pod の名前をクリックします。
4. **Logs** をクリックします。**ovnkube-master** のデフォルトでは、**northd** コンテナーに関連付けられたログが表示されます。
5. ドロップダウンメニューを使用して、各コンテナーのログを順番に選択します。

28.3.5.1. OVN-Kubernetes のログレベルの変更

OVN-Kubernetes のデフォルトのログレベルは 4 です。OVN-Kubernetes をデバッグするには、ログレベルを 5 に設定します。次の手順に従って OVN-Kubernetes のログレベルを上げることで、問題のデバッグに役立てることができます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. 次のコマンドを実行して、OVN-Kubernetes プロジェクト内のすべての Pod の詳細情報を取得します。

```
$ oc get po -o wide -n openshift-ovn-kubernetes
```

出力例

```
NAME                                  READY   STATUS    RESTARTS      AGE    IP           NODE
NOMINATED NODE   READINESS GATES
ovnkube-control-plane-65497d4548-9ptdr   2/2     Running   2 (128m ago)   147m   10.0.0.3
ci-ln-3njdr9b-72292-5nwkp-master-0         <none>          <none>
ovnkube-control-plane-65497d4548-j6zfk   2/2     Running   0              147m   10.0.0.5     ci-
ln-3njdr9b-72292-5nwkp-master-2         <none>          <none>
ovnkube-node-5dx44                    8/8     Running   0              146m   10.0.0.3     ci-ln-
3njdr9b-72292-5nwkp-master-0         <none>          <none>
ovnkube-node-dpfn4                    8/8     Running   0              146m   10.0.0.4     ci-ln-3njdr9b-
72292-5nwkp-master-1         <none>          <none>
ovnkube-node-kwc9l                    8/8     Running   0              134m   10.0.128.2   ci-ln-
3njdr9b-72292-5nwkp-worker-a-2fjcj   <none>          <none>
ovnkube-node-mcrhl                    8/8     Running   0              134m   10.0.128.4   ci-ln-
3njdr9b-72292-5nwkp-worker-c-v9x5v   <none>          <none>
ovnkube-node-nsct4                    8/8     Running   0              146m   10.0.0.5     ci-ln-3njdr9b-
72292-5nwkp-master-2         <none>          <none>
ovnkube-node-zrj9f                    8/8     Running   0              134m   10.0.128.3   ci-ln-3njdr9b-
72292-5nwkp-worker-b-v78h7   <none>          <none>
```

2. 次の例のような **ConfigMap** ファイルを作成し、**env-overrides.yaml** などのファイル名を使用します。

ConfigMap ファイルの例

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: env-overrides
  namespace: openshift-ovn-kubernetes
data:
  ci-ln-3njdr9b-72292-5nwkp-master-0: | (1)
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
```

```
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
  ci-ln-3njdr9b-72292-5nwkp-master-2: |
    # This sets the log level for the ovn-kubernetes node process:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for ovn-controller:
    OVN_LOG_LEVEL=dbg
  _master: | 2
    # This sets the log level for the ovn-kubernetes master process as well as the ovn-
dbchecker:
    OVN_KUBE_LOG_LEVEL=5
    # You might also/instead want to enable debug logging for northd, nbdb and sbdb on all
masters:
    OVN_LOG_LEVEL=dbg
```

1 デバッグログレベルを設定するノードの名前を指定します。

2 **_master** を指定して、**ovnkube-master** コンポーネントのログレベルを設定します。

3. 次のコマンドを使用して、**ConfigMap** ファイルを適用します。

```
$ oc apply -n openshift-ovn-kubernetes -f env-overrides.yaml
```

出力例

```
configmap/env-overrides.yaml created
```

4. 次のコマンドを使用して **ovnkube** Pod を再起動し、新しいログレベルを適用します。

```
$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ci-ln-3njdr9b-72292-5nwkp-master-0 -l app=ovnkube-node
```

```
$ oc delete pod -n openshift-ovn-kubernetes \
--field-selector spec.nodeName=ci-ln-3njdr9b-72292-5nwkp-master-2 -l app=ovnkube-node
```

```
$ oc delete pod -n openshift-ovn-kubernetes -l app=ovnkube-node
```

5. ConfigMap ファイルが特定の Pod のすべてのノードに適用されていることを確認するには、次のコマンドを実行します。

```
$ oc logs -n openshift-ovn-kubernetes --all-containers --prefix ovnkube-node-<xxxx> | grep -
E -m 10 '(Logging config:|vconsole|DBG)'
```

ここでは、以下のようになります。

<XXXX>

前の手順の Pod の文字のランダムなシーケンスを指定します。

出力例

```
[pod/ovnkube-node-2cpjc/sbdb] + exec /usr/share/ovn/scripts/ovn-ctl --no-monitor '--ovn-
sb-log=-vconsole:info -vfile:off -vPATTERN:console:%D{%Y-%m-
```

```
%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' run_sb_ovsdb
[pod/ovnkube-node-2cpjc/ovnkube-controller] I1012 14:39:59.984506   35767
config.go:2247] Logging config: {File: CNIFile:/var/log/ovn-kubernetes/ovn-k8s-cni-
overlay.log LibovsdbFile:/var/log/ovnkube/libovsdb.log Level:5 LogFileMaxSize:100
LogFileMaxBackups:5 LogFileMaxAge:0 ACLLoggingRateLimit:20}
[pod/ovnkube-node-2cpjc/northd] + exec ovn-northd --no-chdir -vconsole:info -vfile:off '-
vPATTERN:console:%D{%Y-%m-%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' --pidfile
/var/run/ovn/ovn-northd.pid --n-threads=1
[pod/ovnkube-node-2cpjc/nbdb] + exec /usr/share/ovn/scripts/ovn-ctl --no-monitor '--ovn-
nb-log=-vconsole:info -vfile:off -vPATTERN:console:%D{%Y-%m-
%dT%H:%M:%S.###Z}|%05N|%c%T|%p|%m' run_nb_ovsdb
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.552Z|00002|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes, including 1
bucket with 6 nodes (32 nodes total across 32 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00003|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes, including 1
bucket with 6 nodes (64 nodes total across 64 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00004|hmap|DBG|lib/shash.c:114: 1 bucket with 6+ nodes, including 1
bucket with 7 nodes (32 nodes total across 32 buckets)
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00005|reconnect|DBG|unix:/var/run/openvswitch/db.sock: entering
BACKOFF
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00007|reconnect|DBG|unix:/var/run/openvswitch/db.sock: entering
CONNECTING
[pod/ovnkube-node-2cpjc/ovn-controller] 2023-10-
12T14:39:54.553Z|00008|ovsdb_cs|DBG|unix:/var/run/openvswitch/db.sock:
SERVER_SCHEMA_REQUESTED -> SERVER_SCHEMA_REQUESTED at lib/ovsdb-
cs.c:423
```

6. オプション: 次のコマンドを実行して、**ConfigMap** ファイルが適用されていることを確認します。

```
for f in $(oc -n openshift-ovn-kubernetes get po -l 'app=ovnkube-node' --no-headers -o
custom-columns=N:.metadata.name) ; do echo "---- $f ----" ; oc -n openshift-ovn-kubernetes
exec -c ovnkube-controller $f --  pgrep -a -f  init-ovnkube-controller | grep -P -o
'^.*loglevel\s+\d' ; done
```

出力例

```
---- ovnkube-node-2dt57 ----
60981 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-c-vmh5n.c.openshift-
qe.internal --init-node xpst8-worker-c-vmh5n.c.openshift-qe.internal --config-
file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-4zznh ----
178034 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-2.c.openshift-qe.internal --
init-node xpst8-master-2.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-548sx ----
77499 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-a-fjtnb.c.openshift-qe.internal -
-init-node xpst8-worker-a-fjtnb.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-6btrf ----
```

```
73781 /usr/bin/ovnkube --init-ovnkube-controller xpst8-worker-b-p8rww.c.openshift-
qe.internal --init-node xpst8-worker-b-p8rww.c.openshift-qe.internal --config-
file=/run/ovnkube-config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
---- ovnkube-node-fkc9r ----
130707 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-0.c.openshift-qe.internal --
init-node xpst8-master-0.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 5
---- ovnkube-node-tk9l4 ----
181328 /usr/bin/ovnkube --init-ovnkube-controller xpst8-master-1.c.openshift-qe.internal --
init-node xpst8-master-1.c.openshift-qe.internal --config-file=/run/ovnkube-
config/ovnkube.conf --ovn-empty-lb-events --loglevel 4
```

28.3.6. OVN-Kubernetes Pod ネットワーク接続のチェック

OpenShift Container Platform 4.10 以降の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。これには、Kubernetes API、OpenShift API、および個々のノードが含まれます。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。

前提条件

- OpenShift CLI (**oc**) へのアクセスがある。
- **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。
- **jq** がインストールされている。

手順

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

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

2. 次のコマンドを使用して、各接続オブジェクトの最新の成功を表示します。

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]| .spec.targetEndpoint,.status.successes[0]'
```

3. 次のコマンドを使用して、各接続オブジェクトの最新のエラーを表示します。

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]| .spec.targetEndpoint,.status.failures[0]'
```

4. 次のコマンドを使用して、各接続オブジェクトの最新の停止を表示します。

```
$ oc get podnetworkconnectivitychecks -n openshift-network-diagnostics \
-o json | jq '.items[]| .spec.targetEndpoint,.status.outages[0]'
```

   接続チェックコントローラーは、これらのチェックからのメトリクスも Prometheus に記録します。

5. 次のコマンドを実行して、すべてのメトリクスを表示します。

```
$ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
promtool query instant  http://localhost:9090 \
'{component="openshift-network-diagnostics"}'
```

6. 過去 5 分間のソース Pod と openshift api サービス間のレイテンシーを表示します。

```
$ oc exec prometheus-k8s-0 -n openshift-monitoring -- \
promtool query instant  http://localhost:9090 \
'{component="openshift-network-diagnostics"}'
```

28.3.7. 関連情報

- Red Hat サポート用のクラスターについてのデータの収集
- 接続ヘルスチェックの実装
- エンドポイントのネットワーク接続の確認

28.4. OVN-KUBERNETES ネットワークポリシー

重要

AdminNetworkPolicy リソースはテクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

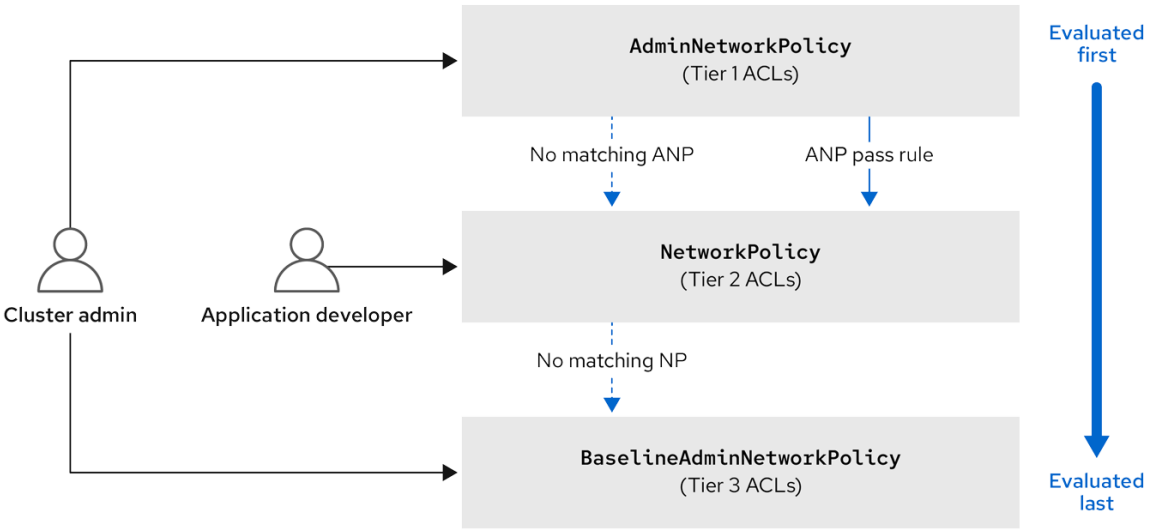
Kubernetes は、ネットワークセキュリティーの強化に使用できる 2 つの機能を提供します。機能の 1 つは、ユーザーによるネットワークポリシーの適用を可能にする **NetworkPolicy** API です。これは主にアプリケーション開発者と namespace テナント向けの機能で、namespace スコープのポリシーを作成して namespace を保護することを目的としています。詳細は、ネットワークポリシーについて を参照してください。

もう 1 つの機能は **AdminNetworkPolicy** です。これは、**AdminNetworkPolicy** (ANP) API と **BaselineAdminNetworkPolicy** (BANP) API の 2 つの API で構成されます。ANP と BANP は、クラスターおよびネットワーク管理者向けの機能で、クラスタースコープのポリシーを作成してクラスター全体を保護することを目的としています。クラスター管理者は、ANP を使用すると、**NetworkPolicy** オブジェクトよりも優先されるオーバーライド不可能なポリシーを適用できます。BANP を使用すると、**NetworkPolicy** オブジェクトを使用して必要に応じてユーザーがオーバーライドできるオプションのクラスタースコープのネットワークポリシールールを設定および適用できます。ANP と BANP を一緒に使用すると、クラスターの保護に使用できるマルチテナンシーポリシーを作成できます。

OpenShift Container Platform の OVN-Kubernetes CNI は、アクセス制御リスト (ACL) の階層を使用してこれらのネットワークポリシーを実装し、それらを評価して適用します。ACL は、階層 1 から 階層 3 まで降順で評価されます。

階層 1 では **AdminNetworkPolicy** (ANP) オブジェクトを評価します。階層 2 では **NetworkPolicy** オブジェクトを評価します。階層 3 では **BaselineAdminNetworkPolicy** (BANP) オブジェクトを評価します。

図28.3 OVK-Kubernetes アクセス制御リスト (ACL)



トラフィックが ANP ルールに一致した場合、その ANP 内のルールが最初に評価されます。一致が ANP の **allow** または **deny** ルールの場合、クラスター内の既存の **NetworkPolicies** および **BaselineAdminNetworkPolicy** (BANP) が評価から意図的に省略されます。一致が ANP の **pass** ルールの場合、評価が ACL 階層 1 から階層 2 に進み、そこで **NetworkPolicy** ポリシーが評価されます。

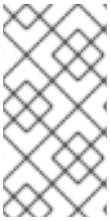
28.4.1. AdminNetworkPolicy

AdminNetworkPolicy (ANP) は、クラススコープのカスタムリソース定義 (CRD) です。OpenShift Container Platform 管理者は、namespace を作成する前にネットワークポリシーを作成することで、ANP を使用してネットワークを保護できます。さらに、**NetworkPolicy** オブジェクトによってオーバーライドできないネットワークポリシーをクラススコープのレベルで作成できます。

AdminNetworkPolicy オブジェクトと **NetworkPolicy** オブジェクトの主な違いは、前者は管理者用でクラススコープであるのに対し、後者はテナント所有者用で namespace スコープであることです。

ANP を使用すると、管理者は以下を指定できます。

- 評価の順序を決定する **priority** 値。値が小さいほど優先度が高くなります。
- 一連の namespace または namespace で構成される **subject**。
- **subject** へのすべての Ingress トラフィックに適用される Ingress ルールのリスト。
- **subject** からのすべての Egress トラフィックに適用される Egress ルールのリスト。



注記

AdminNetworkPolicy リソースは、実稼働環境ではないテストクラスターで有効にできる **TechnologyPreviewNoUpgrade** 機能です。フィーチャーゲートと **TechnologyPreviewNoUpgrade** 機能の詳細には、このセクションの「関連情報」の「フィーチャーゲートの使用による各種機能の有効化」を参照してください。

AdminNetworkPolicy の例

例28.1 ANP の YAML ファイルの例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: sample-anp-deny-pass-rules (1)
spec:
  priority: 50 (2)
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: example.name (3)
  ingress: (4)
  - name: "deny-all-ingress-tenant-1" (5)
    action: "Deny"
    from:
    - pods:
        namespaces: (6)
          namespaceSelector:
            matchLabels:
              custom-anp: tenant-1
        podSelector:
          matchLabels:
            custom-anp: tenant-1 (7)
  egress:(8)
  - name: "pass-all-egress-to-tenant-1"
    action: "Pass"
    to:
    - pods:
        namespaces:
          namespaceSelector:
            matchLabels:
              custom-anp: tenant-1
        podSelector:
          matchLabels:
            custom-anp: tenant-1
```

(1) ANP の名前を指定します。

(2) **spec.priority** フィールドは、0 から 99 までの値を受け入れ、クラスター内で最大 100 個の ANP をサポートします。値が小さいほど優先度が高くなります。同じ優先度の **AdminNetworkPolicy** を作成すると、非決定的な結果が生じます。

(3) ANP リソースを適用する namespace を指定します。

(4) ANP には Ingress ルールと Egress ルールの両方があります。**spec.ingress** フィールドの ANP ルールは、**action** フィールドの **Pass**、**Deny**、および **Allow** の値を受け入れます。

(5) **ingress.name** の名前を指定します。

(6) ANP リソースを適用する Pod の選択元の namespace を指定します。

(7) ANP リソースを適用する Pod の **podSelector.matchLabels** 名を指定します。

(8) ANP には Ingress ルールと Egress ルールの両方があります。**spec.egress** フィールドの ANP ルールは、**action** フィールドの **Pass**、**Deny**、および **Allow** の値を受け入れます。

関連情報

- 機能ゲートの使用による各種機能の有効化
- Network Policy API Working Group

28.4.1.1. ルールの AdminNetworkPolicy アクション

管理者は、**AdminNetworkPolicy** ルールの **action** フィールドに **Allow**、**Deny**、または **Pass** を設定できます。OVN-Kubernetes は階層型 ACL を使用してネットワークトラフィックルールを評価するため、ANP を使用すると、非常に強力なポリシールールを設定できます。このポリシールールを変更するには、管理者がルールを変更、削除するか、より高い優先度のルールを設定してオーバーライドする必要があります。

AdminNetworkPolicy の Allow の例
優先度 9 で定義されている次の ANP は、**monitoring** namespace からクラスター内の任意のテナント (他のすべての namespace) への Ingress トラフィックをすべて許可します。

例28.2 強力な Allow ANP の YAML ファイルの例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: allow-monitoring
spec:
  priority: 9
  subject:
    namespaces: {}
  ingress:
  - name: "allow-ingress-from-monitoring"
    action: "Allow"
    from:
    - namespaces:
        namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: monitoring
# ...
```

これは、関係者全員がオーバーライドできない強力な **Allow** ANP の例です。テナントは、**NetworkPolicy** オブジェクトを使用してテナント自体の監視をブロックすることはできません。また、監視を実行するテナントが監視の対象を決定することもできません。

AdminNetworkPolicy の Deny の例
優先度 5 で定義されている次の ANP は、**monitoring** namespace から制限付きテナント (**security: restricted** ラベルを持つ namespace) への Ingress トラフィックをすべてブロックします。

例28.3 強力な Deny ANP の YAML ファイルの例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: block-monitoring
```

```
spec:
  priority: 5
  subject:
    namespaces:
      matchLabels:
        security: restricted
  ingress:
  - name: "deny-ingress-from-monitoring"
    action: "Deny"
    from:
    - namespaces:
        namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: monitoring
# ...
```

これは、関係者全員がオーバーライドできない強力な **Deny** ANP です。制限付きテナントの所有者は、トラフィックの監視を許可する権限を自分自身に付与できません。また、インフラストラクチャーの監視サービスは、これらの機密性の高い namespace から何も収集できません。

強力な **Allow** の例と組み合わせると、**block-monitoring** ANP の優先度の値よりも Allow の優先度の値のほうが高いため、制限付きテナントが監視されなくなります。

AdminNetworkPolicy の Pass の例

優先度 7 で定義されている次の ANP は、**monitoring** namespace から内部インフラストラクチャーテナント (**security: internal** ラベルを持つ namespace) への Ingress トラフィックをすべて ACL の階層 2 に渡し、トラフィックが namespace の **NetworkPolicy** オブジェクトによって評価されるようにします。

例28.4 強力な Pass ANP の YAML ファイルの例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: AdminNetworkPolicy
metadata:
  name: pass-monitoring
spec:
  priority: 7
  subject:
    namespaces:
      matchLabels:
        security: internal
  ingress:
  - name: "pass-ingress-from-monitoring"
    action: "Pass"
    from:
    - namespaces:
        namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: monitoring
# ...
```

この例は、テナント所有者によって定義された **NetworkPolicy** オブジェクトに決定を委譲する強力な **Pass** アクション ANP です。この **pass-monitoring** ANP により、**internal** セキュリティーレベルでグ

ループ化されたすべてのテナント所有者は、インフラストラクチャーの監視サービスによって namespace スコープの **NetworkPolicy** オブジェクトを使用してメトリクスを収集する必要があるかどうかを選択できます。

28.4.2. BaselineAdminNetworkPolicy

BaselineAdminNetworkPolicy (BANP) は、クラスタースコープのカスタムリソース定義 (CRD) です。OpenShift Container Platform 管理者は、BANP を使用すると、**NetworkPolicy** オブジェクトを使用してユーザーが必要に応じてオーバーライドできるオプションのベースラインネットワークポリシールールを設定および適用できます。BANP のルールアクションは、**allow** または **deny** です。

BaselineAdminNetworkPolicy リソースは、クラスターのシングルトンオブジェクトであり、渡されたトラフィックポリシーがクラスター内のどの **NetworkPolicy** オブジェクトにも一致しない場合にガードレールポリシーとして使用できます。BANP は、クラスター内トラフィックをデフォルトでブロックするガードレールを提供するデフォルトのセキュリティーモデルとしても使用できます。その場合、ユーザーが **NetworkPolicy** オブジェクトを使用して既知のトラフィックを許可する必要があります。BANP リソースを作成するときは、名前として **default** を使用する必要があります。

BANP を使用すると、管理者は以下を指定できます。

- 一連の namespace または namespace で構成される **subject**。
- **subject** へのすべての Ingress トラフィックに適用される Ingress ルールのリスト。
- **subject** からのすべての Egress トラフィックに適用される Egress ルールのリスト。

注記

BaselineAdminNetworkPolicy は、実稼働環境ではないテストクラスターで有効にできる **TechnologyPreviewNoUpgrade** 機能です。

BaselineAdminNetworkPolicy の例

例28.5 BANP の YAML ファイルの例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default 1
spec:
  subject:
    namespaces:
      matchLabels:
        kubernetes.io/metadata.name: example.name 2
  ingress: 3
  - name: "deny-all-ingress-from-tenant-1" 4
    action: "Deny"
    from:
    - pods:
        namespaces:
          namespaceSelector:
            matchLabels:
              custom-banp: tenant-1 5
        podSelector:
          matchLabels:
```

```
          custom-banp: tenant-1 6
  egress:
  - name: "allow-all-egress-to-tenant-1"
    action: "Allow"
    to:
    - pods:
        namespaces:
          namespaceSelector:
            matchLabels:
              custom-banp: tenant-1
        podSelector:
          matchLabels:
            custom-banp: tenant-1
```

1. BANP はシングルトンオブジェクトであるため、ポリシー名は **default** にする必要があります。
2. ANP を適用する namespace を指定します。
3. BANP には Ingress ルールと Egress ルールの両方があります。**spec.ingress** フィールドと **spec.egress** フィールドの BANP ルールは、**action** フィールドの **Deny** と **Allow** の値を受け入れます。
4. **ingress.name** の名前を指定します。
5. BANP リソースを適用する Pod の選択元の namespace を指定します。
6. BANP リソースを適用する Pod の **podSelector.matchLabels** 名を指定します。

BaselineAdminNetworkPolicy の Deny の例

次の BANP シングルトンは、**internal** セキュリティーレベルのテナントに着信するすべての Ingress 監視トラフィックに対してデフォルトの拒否ポリシーを設定します。「AdminNetworkPolicy の Pass の例」と組み合わせると、この拒否ポリシーは、ANP **pass-monitoring** ポリシーによって渡されるすべての Ingress トラフィックに対するガードレールポリシーとして機能します。

例28.6 Deny ガードレールルールの YAML ファイルの例

```
apiVersion: policy.networking.k8s.io/v1alpha1
kind: BaselineAdminNetworkPolicy
metadata:
  name: default
spec:
  subject:
    namespaces:
      matchLabels:
        security: internal
  ingress:
  - name: "deny-ingress-from-monitoring"
    action: "Deny"
    from:
    - namespaces:
        namespaceSelector:
```

```
          matchLabels:
            kubernetes.io/metadata.name: monitoring
# ...
```

action フィールドに **Pass** 値を指定した **AdminNetworkPolicy** リソースを **BaselineAdminNetworkPolicy** リソースと組み合わせて使用すると、マルチテナントポリシーを作成できます。このマルチテナントポリシーを使用すると、あるテナントのアプリケーションの監視データを収集しながら、別のテナントのデータを収集しないことが可能になります。

管理者が「AdminNetworkPolicy の **Pass** アクションの例」と「BaselineAdminNetworkPolicy の **Deny** の例」の両方を適用すると、BANP の前に評価される **NetworkPolicy** リソースを作成するかどうかをテナントが選択できるようになります。

たとえば、テナント 1 が Ingress トラフィックを監視する次の **NetworkPolicy** リソースを設定したとします。

例28.7 NetworkPolicy の例

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-monitoring
  namespace: tenant 1
spec:
  podSelector:
  policyTypes:
    - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: monitoring
# ...
```

この場合、テナント 1 のポリシーは、「AdminNetworkPolicy の **Pass** アクションの例」の後、**security** レベルが **internal** のテナントに着信する Ingress 監視トラフィックをすべて拒否する「BaselineAdminNetworkPolicy の **Deny** の例」の前に評価されます。テナント 1 の **NetworkPolicy** オブジェクトを設定すると、テナント 1 はアプリケーションのデータを収集できるようになります。一方、**NetworkPolicy** オブジェクトが設定されていないテナント 2 は、データを収集できません。管理者はデフォルトでは内部のテナントを監視していませんでした。その代わりに BANP を作成し、テナントが **NetworkPolicy** オブジェクトを使用して BANP のデフォルト動作をオーバーライドできるようにしました。

28.5. OVNKUBE-TRACE を使用した OPENFLOW のトレース

OVN と OVS のトラフィックフローは、**ovnkube-trace** という単一のユーティリティーでシミュレートできます。**ovnkube-trace** ユーティリティーは、**ovn-trace**、**ovs-appctl ofproto/trace**、および **ovn-detrace** を実行し、その情報を 1 つの出力に関連付けます。

専用コンテナーから **ovnkube-trace** バイナリーを実行できます。OpenShift Container Platform 4.7 以降のリリースでは、バイナリーをローカルホストにコピーして、そのホストから実行することもできます。

28.5.1. ローカルホストへの ovnkube-trace のインストール

ovnkube-trace ツールは、OVN-Kubernetes で動作する OpenShift Container Platform クラスター内のポイント間における任意の UDP または TCP トラフィックのパケットシミュレーションをトレースします。**ovnkube-trace** バイナリーをローカルホストにコピーして、クラスターに対して実行できるようにします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 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. 次のコマンドを使用して Pod 変数を作成します。

```
$  POD=$(oc get pods -n openshift-ovn-kubernetes -l app=ovnkube-control-plane -o name | head -1 | awk -F '/' '{print $NF}')
```

2. ローカルホストで次のコマンドを実行して、**ovnkube-control-plane** Pod からバイナリーをコピーします。

```
$  oc cp -n openshift-ovn-kubernetes $POD:/usr/bin/ovnkube-trace -c ovnkube-cluster-manager ovnkube-trace
```

注記

Red Hat Enterprise Linux (RHEL) 8 を使用して **ovnkube-trace** ツールを実行している場合は、**/usr/lib/rhel8/ovnkube-trace** ファイルをローカルホストにコピーする必要があります。

3. 次のコマンドを実行して、**ovnkube-trace** を実行可能にします。

```
$  chmod +x ovnkube-trace
```

4. 次のコマンドを実行して、**ovnkube-trace** で使用可能なオプションを表示します。

```
$  ./ovnkube-trace -help
```

予想される出力

```
Usage of ./ovnkube-trace:
  -addr-family string
    Address family (ip4 or ip6) to be used for tracing (default "ip4")
  -dst string
    dest: destination pod name
  -dst-ip string
    destination IP address (meant for tests to external targets)
  -dst-namespace string
    k8s namespace of dest pod (default "default")
  -dst-port string
    dst-port: destination port (default "80")
```

```
-kubeconfig string
  absolute path to the kubeconfig file
-loglevel string
  loglevel: klog level (default "0")
-ovn-config-namespace string
  namespace used by ovn-config itself
-service string
  service: destination service name
-skip-detrace
  skip ovn-detrace command
-src string
  src: source pod name
-src-namespace string
  k8s namespace of source pod (default "default")
-tcp
  use tcp transport protocol
-udp
  use udp transport protocol
```

サポートされているコマンドライン引数は、namespace、Pod、サービスなど、よく知られた Kubernetes コンストラクトであるため、MAC アドレス、宛先ノードの IP アドレス、または ICMP タイプを見つける必要はありません。

ログレベルは次のとおりです。

- 0 (最小出力)
- 2 (トレースコマンドの結果を示すより詳細な出力)
- 5 (デバッグ出力)

28.5.2. ovnkube-trace の実行

ovn-trace を実行して、OVN 論理ネットワーク内のパケット転送をシミュレートします。

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クラスターにログインしている。
- ローカルホストに **ovnkube-trace** がインストールされている。

例: デプロイされた Pod からの DNS 解決が機能することをテストする

この例は、デプロイされた Pod からクラスターで実行されるコア DNS Pod への DNS 解決をテストする方法を示しています。

手順

1. 次のコマンドを入力して、default namespace で Web サービスを開始します。

```
$ oc run web --namespace=default --image=quay.io/openshifttest/nginx --labels="app=web" --expose --port=80
```

2. **openshift-dns** namespace で実行されている Pod を一覧表示します。

```
oc get pods -n openshift-dns
```

出力例

```
NAME                READY   STATUS    RESTARTS   AGE
dns-default-8s42x     2/2     Running   0          5h8m
dns-default-mdw6r     2/2     Running   0          4h58m
dns-default-p8t5h     2/2     Running   0          4h58m
dns-default-rl6nk     2/2     Running   0          5h8m
dns-default-xbgqx     2/2     Running   0          5h8m
dns-default-zv8f6     2/2     Running   0          4h58m
node-resolver-62jjb   1/1     Running   0          5h8m
node-resolver-8z4cj   1/1     Running   0          4h59m
node-resolver-bq244   1/1     Running   0          5h8m
node-resolver-hc58n   1/1     Running   0          4h59m
node-resolver-lm6z4   1/1     Running   0          5h8m
node-resolver-zfx5k   1/1     Running   0          5h
```

3. 次の **ovnkube-trace** コマンドを実行して、DNS 解決が機能していることを確認します。

```
$ ./ovnkube-trace \
  -src-namespace default \ 1
  -src web \ 2
  -dst-namespace openshift-dns \ 3
  -dst dns-default-p8t5h \ 4
  -udp -dst-port 53 \ 5
  -loglevel 0 6
```

1. ソース Pod の namespace
2. ソース Pod 名
3. 宛先 Pod の namespace
4. 宛先 Pod 名
5. **udp** トランスポートプロトコルを使用します。ポート 53 は、DNS サービスが使用するポートです。
6. ログレベルを 0 に設定します (0 は最小限で、5 はデバッグです)。

src&dst Pod が同じノードに配置された場合の出力例:

```
ovn-trace source pod to destination pod indicates success from web to dns-default-p8t5h
ovn-trace destination pod to source pod indicates success from dns-default-p8t5h to web
ovs-appctl ofproto/trace source pod to destination pod indicates success from web to dns-default-p8t5h
ovs-appctl ofproto/trace destination pod to source pod indicates success from dns-default-p8t5h to web
ovn-detrace source pod to destination pod indicates success from web to dns-default-p8t5h
ovn-detrace destination pod to source pod indicates success from dns-default-p8t5h to web
```

src&dst Pod が別のノードに配置された場合の出力例:

```
ovn-trace source pod to destination pod indicates success from web to dns-default-8s42x
ovn-trace (remote) source pod to destination pod indicates success from web to dns-default-8s42x
ovn-trace destination pod to source pod indicates success from dns-default-8s42x to web
ovn-trace (remote) destination pod to source pod indicates success from dns-default-8s42x to web
ovs-appctl ofproto/trace source pod to destination pod indicates success from web to dns-default-8s42x
ovs-appctl ofproto/trace destination pod to source pod indicates success from dns-default-8s42x to web
ovn-detrace source pod to destination pod indicates success from web to dns-default-8s42x
ovn-detrace destination pod to source pod indicates success from dns-default-8s42x to web
```

この出力は、デプロイされた Pod から DNS ポートへの解決が成功し、その反対方向への解決も成功したことを示しています。つまり、Web Pod がコア DNS からの DNS 解決を行う場合に、UDP ポート 53 で双方向のトラフィックがサポートされていることがわかります。

たとえば、これが機能せず、**ovn-trace** を取得する必要がある場合は、**proto/trace** と **ovn-detrace** の **ovs-appctl**、およびデバッグのタイプ情報が、ログレベルを 2 に上げて、以下のようにコマンドを再度実行します。

```
$ ./ovnkube-trace \
  -src-namespace default \
  -src web \
  -dst-namespace openshift-dns \
  -dst dns-default-467qw \
  -udp -dst-port 53 \
  -loglevel 2
```

このログレベルの出力は多すぎるため、ここにはリストできません。障害状況では、このコマンドの出力は、どのフローがそのトラフィックを破棄しているかを示します。たとえば、egress または ingress ネットワークポリシーが、そのトラフィックを許可しないクラスターで設定されている場合などがあります。

例: デバッグ出力を使用して設定済みのデフォルトの拒否を確認する

この例は、デバッグ出力を使用して、デフォルトのingress 拒否ポリシーがトラフィックをブロックしていることを特定する方法を示しています。

手順

1. すべての namespace におけるすべての Pod からの ingress を拒否する **deny-by-default** ポリシーを定義する次の YAML を作成します。YAML を **deny-by-default.yaml** ファイルに保存します。

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

2. 次のコマンドを入力して、ポリシーを適用します。

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

出力例

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

3. 次のコマンドを入力して、**default** namespace で Web サービスを開始します。

```
$ oc run web --namespace=default --image=quay.io/openshifttest/nginx --labels="app=web" --expose --port=80
```

4. 次のコマンドを実行して、**prod** namespace を作成します。

```
$ oc create namespace prod
```

5. 次のコマンドを実行して、**prod** namespace にラベルを付けます。

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

6. 次のコマンドを実行して、**alpine** イメージを **prod** namespace にデプロイし、シェルを開始します。

```
$ oc run test-6459 --namespace=prod --rm -i -t --image=alpine -- sh
```

7. 別のターミナルセッションを開きます。

8. この新しいターミナルセッションで **ovn-trace** を実行して、namespace **prod** で実行されているソース Pod **test-6459** と **default** namespace で実行されている宛先 Pod 間の通信の失敗を確認します。

```
$ ./ovnkube-trace \
 -src-namespace prod \
 -src test-6459 \
 -dst-namespace default \
 -dst web \
 -tcp -dst-port 80 \
 -loglevel 0
```

出力例

```
ovn-trace source pod to destination pod indicates failure from test-6459 to web
```

9. 次のコマンドを実行して、ログレベルを 2 に上げて、失敗の理由を明らかにします。

```
$ ./ovnkube-trace \
 -src-namespace prod \
 -src test-6459 \
 -dst-namespace default \
```

```
 -dst web \
 -tcp -dst-port 80 \
 -loglevel 2
```

出力例

```
...
------------------------------------------------
 3. ls_out_acl_hint (northd.c:7454): !ct.new && ct.est && !ct.rpl && ct_mark.blocked == 0,
priority 4, uuid 12efc456
    reg0[8] = 1;
    reg0[10] = 1;
    next;
 5. ls_out_acl_action (northd.c:7835): reg8[30..31] == 0, priority 500, uuid 69372c5d
    reg8[30..31] = 1;
    next(4);
 5. ls_out_acl_action (northd.c:7835): reg8[30..31] == 1, priority 500, uuid 2fa0af89
    reg8[30..31] = 2;
    next(4);
 4. ls_out_acl_eval (northd.c:7691): reg8[30..31] == 2 && reg0[10] == 1 && (outport ==
@a16982411286042166782_ingressDefaultDeny), priority 2000, uuid 447d0dab
    reg8[17] = 1;
    ct_commit { ct_mark.blocked = 1; }; (1)
    next;
...
```

(1) デフォルトの拒否ポリシーが設定されているため、ingress トラフィックがブロックされています。

10. ラベルが **purpose=production** の特定の namespace 内にあるすべての Pod からのトラフィックを許可するポリシーを作成します。YAML を **web-allow-prod.yaml** ファイルに保存します。

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

11. 次のコマンドを入力して、ポリシーを適用します。

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

12. 次のコマンドを入力して、**ovnkube-trace** を実行し、トラフィックが許可されていることを確認します。

```
$ ./ovnkube-trace \
 -src-namespace prod \
 -src test-6459 \
 -dst-namespace default \
 -dst web \
 -tcp -dst-port 80 \
 -loglevel 0
```

予想される出力

```
ovn-trace source pod to destination pod indicates success from test-6459 to web
ovn-trace destination pod to source pod indicates success from web to test-6459
ovs-appctl ofproto/trace source pod to destination pod indicates success from test-6459 to
web
ovs-appctl ofproto/trace destination pod to source pod indicates success from web to test-
6459
ovn-detrace source pod to destination pod indicates success from test-6459 to web
ovn-detrace destination pod to source pod indicates success from web to test-6459
```

13. 手順 6 で開いたシェルで次のコマンドを実行して、nginx を Web サーバーに接続します。

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

予想される出力

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

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

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

28.5.3. 関連情報

- ovnkube-trace ユーティリティーを使用した Openflow のトレース
- ovnkube-trace

28.6. OPENSHIFT SDN ネットワークプラグインからの移行

クラスター管理者は、OpenShift SDN ネットワークプラグインから OVN-Kubernetes ネットワークプラグインに移行できます。

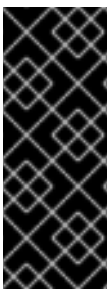
OVN-Kubernetes の詳細は、OVN-Kubernetes ネットワークプラグインについて を参照してください。

28.6.1. OVN-Kubernetes ネットワークプラグインへの移行

OVN-Kubernetes ネットワークプラグインへの移行は、クラスターに到達できないダウンタイムを含む手動プロセスです。ロールバック手順が提供されますが、移行は一方向プロセスとなることが意図されています。

OVN-Kubernetes ネットワークプラグインへの移行は、次のプラットフォームでサポートされています。

- ベアメタルハードウェア
- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- IBM Cloud®
- Microsoft Azure
- Red Hat OpenStack Platform (RHOSP)
- VMware vSphere



重要

OVN-Kubernetes ネットワークプラグインとの間の移行は、Red Hat OpenShift Dedicated、Azure Red Hat OpenShift (ARO)、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。

OpenShift SDN ネットワークプラグインから OVN-Kubernetes ネットワークプラグインへの移行は、Nutanix ではサポートされていません。

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

28.6.1.1. OVN-Kubernetes ネットワークプラグインへの移行についての考慮点

OpenShift Container Platform クラスターに 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 ネットワークプラグインは、OpenShift SDN ネットワークプラグインに存在する多くの機能を実装していますが、設定は同じではありません。

- クラスターが次の OpenShift SDN ネットワークプラグイン機能のいずれかを使用する場合、OVN-Kubernetes ネットワークプラグイン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 ルーター Pod
- クラスターまたは周囲のネットワークが **100.64.0.0/16** アドレス範囲の一部を使用している場合、**spec.defaultNetwork.ovnKubernetesConfig** オブジェクト定義で **v4InternalSubnet** 仕様を指定して、別の未使用の IP 範囲を選択する必要があります。OVN-Kubernetes は、デフォルトで IP 範囲 **100.64.0.0/16** を内部的に使用します。

以下のセクションでは、OVN-Kubernetes と OpenShift SDN ネットワークプラグインの前述の機能の設定の違いを強調しています。

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

クラスター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 ネットワークプラグインに移行することはできません。

Egress IP アドレス
OpenShift SDN は、2 つの異なる Egress IP モードをサポートしています。

- **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。

移行プロセスでは、自動割り当てモードを使用する Egress IP 設定の移行がサポートされています。

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28.4 egress IP アドレス設定の違い

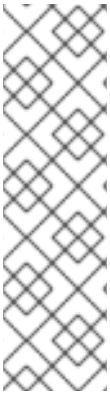
OVN-Kubernetes	OpenShift SDN
• **EgressIPs** オブジェクトを作成します。 • アノテーションを **Node** オブジェクトに追加します。	• **NetNamespace** オブジェクトにパッチを適用します。 • **HostSubnet** オブジェクトにパッチを適用します。

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28.5 egress ネットワークポリシー設定の相違点

OVN-Kubernetes	OpenShift SDN
• **EgressFirewall** オブジェクトを namespace に作成します。	• **EgressNetworkPolicy** オブジェクトを namespace に作成します。



注記

EgressFirewall オブジェクトの名前は **default** にしか設定できないため、移行後は、OpenShift SDN での名前に関係なく、移行されたすべての **EgressNetworkPolicy** オブジェクトに **default** という名前が付けられます。

その後、OpenShift SDN にロールバックすると、以前の名前が失われるため、すべての **EgressNetworkPolicy** オブジェクトに **default** という名前が付けられます。

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 プロキシーモードまたは DNS プロキシーモードでは 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28.6 マルチキャスト設定の相違点

OVN-Kubernetes	OpenShift SDN
• アノテーションを **Namespace** オブジェクトに追加します。	• 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。

28.6.1.2. 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28.7 OpenShift SDN から OVN-Kubernetes への移行

ユーザーが開始する手順	移行アクティビティー
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。	Cluster Network Operator (CNO) **cluster** という名前の **Network.config.openshift.io** CR のステータスを更新します。 Machine Config Operator (MCO) OVN-Kubernetes に必要な systemd 設定の更新をロールアウトします。デフォルトでは、MCO はプールごとに一度に 1 台のマシンを更新するため、クラスターのサイズに応じて移行にかかる合計時間が長くなります。
Network.config.openshift.io CR の **networkType** フィールドを更新します。	CNO 以下のアクションを実行します。 • 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。 • Multus オブジェクトを更新して、新しいネットワークプラグインを反映します。

ユーザーが開始する手順	移行アクティビティー
クラスターの各ノードを再起動します。	**クラスター** 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。

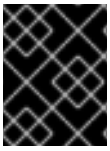
OpenShift SDN へのロールバックが必要な場合、以下の表がプロセスについて説明します。

表28.8 OpenShift SDN へのロールバックの実行

ユーザーが開始する手順	移行アクティビティー
MCO を一時停止し、移行が中断されないようにします。	MCO が停止します。
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。	CNO **cluster** という名前の **Network.config.openshift.io** CR のステータスを更新します。
networkType フィールドを更新します。	CNO 以下のアクションを実行します。 ● 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。 ● Multus オブジェクトを更新して、新しいネットワークプラグインを反映します。
クラスターの各ノードを再起動します。	**クラスター** 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。
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。	MCO OpenShift SDN に必要な systemd 設定の更新をロールアウトします。デフォルトでは、MCO はプールごとに一度に 1 台のマシンを更新するため、移行にかかる合計時間はクラスターのサイズに応じて増加します。

28.6.2. OVN-Kubernetes ネットワークプラグインへの移行

クラスター管理者は、クラスターのネットワークプラグインを 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分離モードの OpenShift SDN CNI ネットワークプラグイン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。
- ソフトウェア更新後のクラウドプラットフォームでは、すべてのノードに対してポート **6081** で UDP パケットを許可するセキュリティーグループルールを設定する必要があります。

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: いくつかの OpenShift SDN 機能の OVN-Kubernetes 同等機能への自動移行を無効にすることができます。
   - Egress IP
   - Egress ファイアウォール

- マルチキャスト

前述の OpenShift SDN 機能の設定の自動移行を無効にするには、次のキーを指定します。

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{
    "spec": {
      "migration": {
        "networkType": "OVNKubernetes",
        "features": {
          "egressIP": <bool>,
          "egressFirewall": <bool>,
          "multicast": <bool>
        }
      }
    }
  }'
```

ここでは、以下のようになります。

bool: 機能の移行を有効にするかどうかを指定します。デフォルトは **true** です。

4. 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。

   - 最大伝送単位 (MTU)。このオプションの手順で MTU をカスタマイズする前に、以下を考慮してください。

     - デフォルトの MTU を使用しており、移行中にデフォルトの MTU を維持したい場合は、この手順を無視できます。

     - カスタム MTU を使用しており、移行中にカスタム MTU を維持する必要がある場合は、この手順でカスタム MTU 値を宣言する必要があります。

     - 移行中に MTU 値を変更する場合、この手順は機能しません。代わりに、まず「クラスター MTU の変更」に記載された指示に従う必要があります。その後、この手順を実行してカスタム MTU 値を宣言すると、カスタム MTU 値を維持できます。

       注記

       OpenShift-SDN と OVN-Kubernetes のオーバーレイオーバーヘッドは異なります。MTU 値は、「MTU 値の選択」ページにあるガイドラインに従って選択する必要があります。

   - Geneve (Generic Network Virtualization Encapsulation) オーバーレイネットワークポート

   - OVN-Kubernetes IPv4 内部サブネット

   - OVN-Kubernetes IPv6 内部サブネット

   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>,
        "v4InternalSubnet":"<ipv4_subnet>",
        "v6InternalSubnet":"<ipv6_subnet>"
  }}}}'
```

ここでは、以下のようになります。

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。

ipv4_subnet

OVN-Kubernetes による内部使用のための IPv4 アドレス範囲。IP アドレス範囲が、OpenShift Container Platform インストールで使用される他のサブネットと重複しないようにする必要があります。IP アドレス範囲は、クラスターに追加できるノードの最大数より大きくする必要があります。デフォルト値は **100.64.0.0/16** です。

ipv6_subnet

OVN-Kubernetes による内部使用のための IPv6 アドレス範囲。IP アドレス範囲が、OpenShift Container Platform インストールで使用される他のサブネットと重複しないようにする必要があります。IP アドレス範囲は、クラスターに追加できるノードの最大数より大きくする必要があります。デフォルト値は **fd98::/48** 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
    }}}}'
```

5. 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ーのサイズと共に増加します。

6. 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リスト表示するには、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
   ```

   <config_name> は **machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前です。

   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 をリスト表示するには、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。

7. 移行を開始するには、次のいずれかのコマンドを使用して OVN-Kubernetes ネットワークプラグインを設定します。

- クラスターネットワークの 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" } }'
```

- 別のクラスターネットワーク 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。

8. Multus デー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-<xxxxx>** です。ここで、 **<xxxxx>** 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
```

9. ネットワークプラグインの変更を完了するには、クラスター内の各ノードを再起動します。次のいずれかの方法で、クラスター内のノードを再起動できます。

   - **oc rsh** コマンドでは、次のような bash スクリプトを使用できます。

     ```
     #!/bin/bash
     readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator -o wide| grep daemon|awk '{print $1" "$7}')"

     for i in "${POD_NODES[@]}"
     do
       read -r POD NODE <<< "$i"
       until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs shutdown -r +1
         do
           echo "cannot reboot node $NODE, retry" && sleep 3
         done
     done
     ```

   - **ssh** コマンドでは、次のような bash スクリプトを使用できます。このスクリプトは、パスワードの入力を求めないように sudo が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
     ```

10. 移行が正常に完了したことを確認します。

    a. ネットワークプラグインが OVN-Kubernetes であることを確認するには、次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}{"\n"}'
       ```

    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、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。

11. 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
```

28.6.3. 関連情報

- Red Hat OpenShift Network Calculator
- OVN-Kubernetes ネットワークプラグイン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
- クラスター MTU の変更
- MTU 値の選択
- 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]

28.7. OPENSHIFT 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 への移行が失敗した場合に、OVN-Kubernetes ネットワークプラグインから OpenShift SDN ネットワークプラグインにロールバックできます。

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

28.7.1. OpenShift SDN ネットワークプラグインへの移行

クラスター管理者は、OpenShift SDN Container Network Interface (CNI) ネットワークプラグインに移行できます。移行中は、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限り、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。
- インフラストラクチャーにインストールされたクラスターが OVN-Kubernetes ネットワークプラグインで設定されている。
- 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. 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. 移行の準備をするには、次のコマンドを入力して移行フィールドを **null** に設定します。

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

3. 移行を開始するには、次のコマンドを入力して、ネットワークプラグインを 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" } }'
```

4. オプション: いくつかの OVN-Kubernetes 機能の OpenShift SDN 同等機能への自動移行を無効にすることができます。

   - Egress IP
   - Egress ファイアウォール
   - マルチキャスト

   前述の OpenShift SDN 機能の設定の自動移行を無効にするには、次のキーを指定します。

```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{
    "spec": {
      "migration": {
        "networkType": "OpenShiftSDN",
        "features": {
          "egressIP": <bool>,
          "egressFirewall": <bool>,
          "multicast": <bool>
        }
      }
    }
  }'
```

   ここでは、以下のようになります。

   bool: 機能の移行を有効にするかどうかを指定します。デフォルトは **true** です。

5. 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
    }}}}'
```

6. クラスター内の各ノードを再起動します。次のいずれかの方法で、クラスター内のノードを再起動できます。

- **oc rsh** コマンドでは、次のような bash スクリプトを使用できます。

```
#!/bin/bash
readarray -t POD_NODES <<< "$(oc get pod -n openshift-machine-config-operator -o
wide| grep daemon|awk '{print $1" "$7}')"

for i in "${POD_NODES[@]}"
do
  read -r POD NODE <<< "$i"
  until oc rsh -n openshift-machine-config-operator "$POD" chroot /rootfs shutdown -r +1
    do
      echo "cannot reboot node $NODE, retry" && sleep 3
    done
done
```

- **ssh** コマンドでは、次のような bash スクリプトを使用できます。このスクリプトは、パスワードの入力を求めないように sudo が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
```

7. Multus デーモンセットのロールアウトが完了するまで待機します。次のコマンドを実行して、ロールアウトのステータスを確認します。

```
$ oc -n openshift-multus rollout status daemonset/multus
```

Multus Pod の名前の形式は **multus-<xxxxx>** です。ここで、 **<xxxxx>** 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. クラスター内のノードが再起動し、multus Pod がロールアウトされたら、次のコマンドを実行して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。

9. 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リスト表示するには、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
```

<config_name> は **machineconfiguration.openshift.io/currentConfig** フィールドのマシン設定の名前です。

10. 移行が正常に完了したことを確認します。

a. ネットワークプラグインが OpenShift SDN であることを確認するには、次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}{"\n"}'
```

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 をリスト表示するには、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。

11. 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。

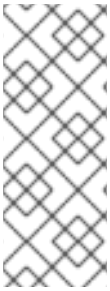
```
$ oc patch Network.operator.openshift.io cluster --type='merge' \
  --patch '{ "spec": { "defaultNetwork": { "ovnKubernetesConfig":null } } }'
```

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
```

28.8. 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。

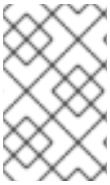


注記

- デュアルスタックネットワークを使用している場合、IPv6 を必要とする、IPv4 にマッピングされ IPv6 アドレス (例: **::FFFF:198.51.100.1**) は使用できません。
- デュアルスタックネットワークは、ベアメタル、IBM Power®、IBM Z® インフラストラクチャー、シングルノード OpenShift、および VMware vSphere に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。

28.8.1. 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。
- クラスターは OVN-Kubernetes ネットワークプラグインを使用します。
- クラスターノードに IPv6 アドレスがある。
- インフラストラクチャーに基づいて IPv6 対応ルーターを設定している。

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
```

28.8.2. 単一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、デュアルスタッククラスターネットワークを単一スタッククラスターネットワークに変換できます。

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。
- クラスターは OVN-Kubernetes ネットワークプラグインを使用します。
- クラスターノードに IPv6 アドレスがある。
- デュアルスタックネットワークを有効にしている。

手順

1. 以下のコマンドを実行して、**networks.config.openshift.io** カスタムリソース (CR) を編集します。

```
$ oc edit networks.config.openshift.io
```

2. 前の手順で **cidr** および **hostPrefix** フィールドに追加した IPv6 固有の設定を削除します。

28.9. EGRESS ファイアウォールとネットワークポリシールールのロギング

クラスター管理者は、クラスターの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。OpenShift Container Platform は、Egress ファイアウォールとネットワークポリシーの両方の監査ログを生成します。

注記

監査ログは、OVN-Kubernetes ネットワークプラグイン でのみ使用できます。

28.9.1. 監査ロギング

OVN-Kubernetes ネットワークプラグインは、Open Virtual Network (OVN) ACL を使用して、egress ファイアウォールと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、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。

ネットワークポリシーの ACL 拒否ログエントリーの例

```
2023-11-02T16:28:54.139Z|00004|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=1
0.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn
2023-11-02T16:28:55.187Z|00005|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=1
0.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn
2023-11-02T16:28:57.235Z|00006|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,nw_dst=1
0.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp_flags=syn
```

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28.9 監査ログの namespace の注釈

Annotation	値
k8s.ovn.org/acl-logging	namespace の監査ログを有効にするには、**allow**、**deny**、またはその両方の少なくとも 1 つを指定する必要があります。 **deny** オプション: **alert**、**warning**、**notice**、**info**、または **debug** を指定します。 **allow** オプション: **alert**、**warning**、**notice**、**info**、または **debug** を指定します。

28.9.2. 監査設定

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。次の YAML は、監査ログ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
```

次の表では、監査ログの設定フィールドについて説明します。

表28.10 policyAuditConfig オブジェクト

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** (50MB) です。
maxLogFiles	integer	保持されるログファイルの最大数。

フィールド	型	説明
比較先	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 ファシリティー。デフォルト値は **local0** です。

28.9.3. クラスターの Egress ファイアウォールとネットワークポリシー監査の設定

クラスター管理者は、クラスターの監査ロ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

- 監査ロギング設定をカスタマイズするには、次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. 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
  namespace: verify-audit-logging
spec:
  podSelector: {}
  policyTypes:
   - Ingress
   - Egress
  ingress:
    - from:
        - podSelector: {}
  egress:
    - to:
       - namespaceSelector:
           matchLabels:
             kubernetes.io/metadata.name: 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ーを作成するには、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

```
2023-11-02T16:28:54.139Z|00004|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,n
w_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp
_flags=syn
2023-11-02T16:28:55.187Z|00005|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,n
w_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp
_flags=syn
2023-11-02T16:28:57.235Z|00006|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:Ingress", verdict=drop, severity=alert, direction=to-lport:
tcp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:01,dl_dst=0a:58:0a:81:02:23,nw_src=10.131.0.39,n
w_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=62,nw_frag=no,tp_src=58496,tp_dst=8080,tcp
_flags=syn
2023-11-02T16:49:57.909Z|00028|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:57.909Z|00029|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
```

```
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:58.932Z|00030|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:58.932Z|00031|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
```

28.9.4. namespace の Egress ファイアウォールとネットワークポリシーの監査ログを有効にする

クラスター管理者は、namespace の監査ログ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

- namespace の監査ログを有効にするには、次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

- 監査ログの最新の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

```
2023-11-02T16:49:57.909Z|00028|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:57.909Z|00029|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:58.932Z|00030|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Egress:0", verdict=allow, severity=alert, direction=from-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
2023-11-02T16:49:58.932Z|00031|acl_log(ovn_pinctrl0)|INFO|name="NP:verify-audit-
logging:allow-from-same-namespace:Ingress:0", verdict=allow, severity=alert, direction=to-
lport:
icmp,vlan_tci=0x0000,dl_src=0a:58:0a:81:02:22,dl_dst=0a:58:0a:81:02:23,nw_src=10.129.2.34,
nw_dst=10.129.2.35,nw_tos=0,nw_ecn=0,nw_ttl=64,nw_frag=no,icmp_type=8,icmp_code=0
```

28.9.5. namespace の Egress ファイアウォールとネットワークポリシーの監査ログを無効にする

クラスター管理者は、namespace の監査ログ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** 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 の監査ログを無効にするには、次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
```

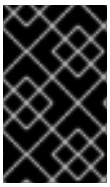
28.9.6. 関連情報

- ネットワークポリシーについて
- プロジェクトの egress ファイアウォールの設定

28.10. IPSEC 暗号化の設定

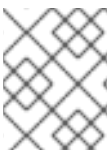
IPsec を有効にすると、ノード間の内部 Pod 間のクラストートラフィックと、Pod とクラスター外部の IPsec エンドポイント間の外部トラフィックの両方を暗号化できます。OVN-Kubernetes クラスターネットワーク上のノード間のすべての Pod 間ネットワークトラフィックが、**トランスポートモード** の IPsec で暗号化されます。

IPsec はデフォルトで無効にされています。クラスターのインストール中またはインストール後に有効にできます。クラスターのインストールの詳細は、OpenShift Container Platform インストールの概要を参照してください。



重要

クラスターが Red Hat OpenShift Container Platform の Hosted Control Plane を使用している場合、IPsec は、Pod 間のトラフィックや外部ホストへのトラフィックの IPsec 暗号化でサポートされません。



注記

IBM Cloud® 上の IPsec は NAT-T のみをサポートします。ESP の使用はサポートされていません。

次のドキュメントの手順を使用して、次のことを行います。

- クラスターのインストール後に IPSec を有効または無効にする

- クラスターと外部ホスト間のトラフィックの IPsec 暗号化を設定する
- IPsec が異なるノード上の Pod 間のトラフィックを暗号化することを確認する

28.10.1. 動作モード

OpenShift Container Platform クラスターで IPsec を使用する場合、以下の動作モードから選択できます。

表28.11 IPsec の動作モード

モード	説明	デフォルト
Disabled	トラフィックは暗号化されません。これはクラスターのデフォルトです。	はい
Full	Pod 間のトラフィックが、「Pod 間の IPsec によって暗号化されるネットワークトラフィックフローのタイプ」の説明のとおりに暗号化されます。IPsec に必要な設定手順を完了すると、外部ノードへのトラフィックを暗号化できます。	いいえ
External	IPsec に必要な設定手順を完了すると、外部ノードへのトラフィックを暗号化できます。	いいえ

28.10.2. 前提条件

外部ホストへのトラフィックを暗号化するために IPsec をサポートする場合は、次の前提条件が満たされていることを確認してください。

- OVN-Kubernetes ネットワークプラグインが、ローカルゲートウェイモード (**ovnKubernetesConfig.gatewayConfig.routingViaHost=true**) で設定されている。
- NMState Operator がインストールされている。この Operator は、IPsec 設定を指定するために必要です。詳細は、Kubernetes NMState Operator について を参照してください。

注記

NMState Operator は、Google Cloud Platform (GCP) で IPsec を設定する場合にのみサポートされます。

- ブタンツール (**butane**) がインストールされている。Butane をインストールするには、Butane のインストール を参照してください。

これらの前提条件は、証明書をホストの NSS データベースに追加するために、および外部ホストと通信するように IPsec を設定するために必要です。

28.10.3. 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できる必要があります。

表28.12 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)

28.10.4. Pod 間のトラフィックの IPsec 暗号化

Pod 間のトラフィックの IPsec 暗号化については、次のセクションで、暗号化される Pod 間のトラフィック、使用される暗号化プロトコルの種類、および X.509 証明書の処理の仕組みについて説明します。以下のセクションは、クラスターと外部ホスト間の IPsec 暗号化には適用されません。この種の暗号化は、特定の外部ネットワークインフラストラクチャーに合わせて手動で設定する必要があります。

28.10.4.1. Pod 間の IPsec によって暗号化されるネットワークトラフィックフローのタイプ

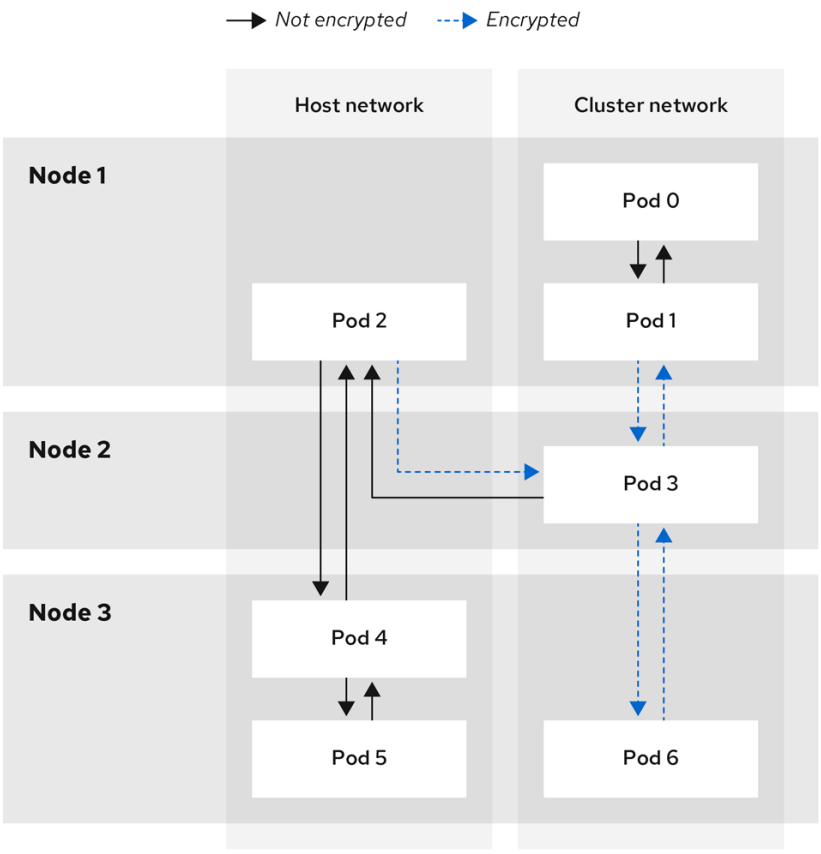
IPsec を有効にすると、Pod 間の以下のネットワークトラフィックフローのみが暗号化されます。

- 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。



28.10.4.2. 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。

28.10.4.3. 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。

28.10.5. 外部トラフィックの IPsec 暗号化

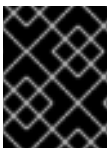
OpenShift Container Platform は、TLS 証明書を使用した外部ホストへのトラフィックの IPsec 暗号化をサポートします。TLS 証明書は管理者が提供する必要があります。

28.10.5.1. サポート対象のプラットフォーム

この機能は次のプラットフォームでサポートされています。

- ベアメタル

- Google Cloud Platform (GCP)
- Red Hat OpenStack Platform (RHOSP)
- VMware vSphere



重要

Red Hat Enterprise Linux (RHEL) ワーカーノードがある場合、これらのプラットフォームは外部トラフィックの IPsec 暗号化をサポートしません。

クラスターが Red Hat OpenShift Container Platform の Hosted Control Plane を使用している場合、外部ホストへのトラフィックを暗号化するための IPsec の設定はサポートされません。

28.10.5.2. 制限事項

次の禁止事項が遵守されていることを確認してください。

- 外部トラフィックの IPsec を設定する場合、IPv6 設定は現在 NMState Operator によってサポートされません。
- 提供する証明書バンドル内の証明書共通名 (CN) には、接頭辞 **ovs_** を指定できません。この名前は、各ノードのネットワークセキュリティーサービス (NSS) データベース内の Pod 間の IPsec CN 名と競合する可能性があるためです。

28.10.6. IPsec 暗号化の有効化

クラスター管理者は、Pod 間の IPsec 暗号化、およびクラスターと外部 IPsec エンドポイント間の IPsec 暗号化を有効にすることができます。

次のいずれかのモードで IPsec を設定できます。

- **Full**: Pod 間のトラフィックおよび外部トラフィックの暗号化
- **External**: 外部トラフィックの暗号化

Pod 間のトラフィックに加えて外部トラフィックの暗号化を設定する必要がある場合は、「外部トラフィックの IPsec 暗号化の設定」の手順も完了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クラスターにログインしている。
- クラスター MTU のサイズを **46** バイト減らして、IPsec ESP ヘッダーにオーバーヘッドを設けている。

手順

1. IPsec 暗号化を有効にするには、次のコマンドを入力します。

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

```
  "defaultNetwork":{
    "ovnKubernetesConfig":{
      "ipsecConfig":{
        "mode":<mode>
      }}}}}'
```

ここでは、以下のようになります。

mode

外部ホストへのトラフィックのみを暗号化するには **External** を指定します。Pod 間のトラフィックを暗号化し、必要に応じて外部ホストへのトラフィックを暗号化するには **Full** を指定します。デフォルトでは、IPsec は無効になっています。

2. オプション: 外部ホストへのトラフィックを暗号化する必要がある場合は、「外部トラフィックの IPsec 暗号化の設定」の手順を実行します。

検証

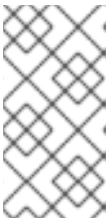
1. OVN-Kubernetes データプレーン Pod の名前を見つけるには、次のコマンドを入力します。

```
$ oc get pods -n openshift-ovn-kubernetes -l=app=ovnkube-node
```

出力例

```
ovnkube-node-5xqbf                 8/8     Running   0          28m
ovnkube-node-6mwcx                 8/8     Running   0          29m
ovnkube-node-ck5fr                 8/8     Running   0          31m
ovnkube-node-fr4ld                 8/8     Running   0          26m
ovnkube-node-wgs4l                 8/8     Running   0          33m
ovnkube-node-zfvcl                 8/8     Running   0          34m
```

2. 次のコマンドを実行して、クラスターで IPsec が有効になっていることを確認します。



注記

クラスター管理者は、IPsec が **Full** モードで設定されている場合に、クラスター上の Pod 間で IPsec が有効になっていることを確認できます。この手順では、クラスターと外部ホストの間で IPsec が機能しているかどうかは検証されません。

```
$ oc -n openshift-ovn-kubernetes rsh ovnkube-node-<XXXXX> ovn-nbctl --no-leader-only get nb_global . ipsec
```

ここでは、以下のようになります。

<XXXXX>

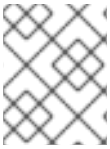
前の手順の Pod の文字のランダムなシーケンスを指定します。

出力例

```
true
```

28.10.7. 外部トラフィックの IPsec 暗号化の設定

クラスター管理者は、IPsec を使用して外部トラフィックを暗号化するには、PKCS#12 証明書の提供を含め、ネットワークインフラストラクチャーに IPsec を設定する必要があります。この手順では Butane を使用してマシン設定を作成するため、**butane** コマンドがインストールされている必要があります。



注記

マシン設定を適用した後、Machine Config Operator はクラスター内の影響を受けるノードを再起動し、新しいマシン設定をロールアウトします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- ローカルコンピューターに **butane** ユーティリティーがインストールされている。
- NMState Operator がクラスターに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。
- IPsec エンドポイント用の既存の PKCS#12 証明書と PEM 形式の CA 証明書があります。
- クラスターで **Full** または **External** モードの IPsec が有効になっている。
- OVN-Kubernetes ネットワークプラグインが、ローカルゲートウェイモード (**ovnKubernetesConfig.gatewayConfig.routingViaHost=true**) で設定されている。

手順

1. NMState Operator ノードネットワーク設定ポリシーを使用して IPsec 設定を作成します。詳細は、IPsec VPN 実装としての Libreswan を参照してください。

   a. IPsec エンドポイントであるクラスターノードの IP アドレスを特定するために、次のコマンドを入力します。

   ```
   $ oc get nodes
   ```

   b. 次の例のように、NMState Operator のノードネットワーク設定ポリシーを含む **ipsec-config.yaml** という名前のファイルを作成します。**NodeNetworkConfigurationPolicy** オブジェクトの概要については、The Kubernetes NMState project を参照してください。

   NMState IPsec トランスポート設定の例

   ```
   apiVersion: nmstate.io/v1
   kind: NodeNetworkConfigurationPolicy
   metadata:
     name: ipsec-config
   spec:
     nodeSelector:
       kubernetes.io/hostname: "<hostname>" 1
     desiredState:
       interfaces:
       - name: <interface_name> 2
   ```

```
    type: ipsec
    libreswan:
      left: <cluster_node> 3
      leftid: '%fromcert'
      leftrsasigkey: '%cert'
      leftcert: left_server
      leftmodecfgclient: false
      right: <external_host> 4
      rightid: '%fromcert'
      rightrsasigkey: '%cert'
      rightsubnet: <external_address>/32 5
      ikev2: insist
      type: transport
```

1 ポリシーを適用するホスト名を指定します。このホストは、IPsec 設定の左側のホストとして機能します。

2 ホスト上に作成するインターフェイスの名前を指定します。

3 クラスター側の IPsec トンネルを終端するクラスターノードのホスト名を指定します。この名前は、提供した PKCS#12 証明書の SAN **[Subject Alternate Name]** と一致する必要があります。

4 外部ホスト名 (**host.example.com** など) を指定します。この名前は、提供した PKCS#12 証明書の SAN **[Subject Alternate Name]** と一致する必要があります。

5 外部ホストの IP アドレス (**10.1.2.3/32** など) を指定します。

NMState IPsec トンネル設定の例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ipsec-config
spec:
  nodeSelector:
    kubernetes.io/hostname: "<hostname>" 1
  desiredState:
    interfaces:
    - name: <interface_name> 2
      type: ipsec
      libreswan:
        left: <cluster_node> 3
        leftid: '%fromcert'
        leftmodecfgclient: false
        leftrsasigkey: '%cert'
        leftcert: left_server
        right: <external_host> 4
        rightid: '%fromcert'
        rightrsasigkey: '%cert'
        rightsubnet: <external_address>/32 5
        ikev2: insist
        type: tunnel
```

1. ポリシーを適用するホスト名を指定します。このホストは、IPsec 設定の左側のホストとして機能します。
2. ホスト上に作成するインターフェイスの名前を指定します。
3. クラスター側の IPsec トンネルを終端するクラスターノードのホスト名を指定します。この名前は、提供した PKCS#12 証明書の SAN **[Subject Alternate Name]** と一致する必要があります。
4. 外部ホスト名 (**host.example.com** など) を指定します。この名前は、提供した PKCS#12 証明書の SAN **[Subject Alternate Name]** と一致する必要があります。
5. 外部ホストの IP アドレス (**10.1.2.3/32** など) を指定します。

c. IPsec インターフェイスを設定するために、次のコマンドを入力します。

```
$ oc create -f ipsec-config.yaml
```

2. 次の証明書ファイルを提供して、各ホストのネットワークセキュリティーサービス (NSS) データベースに追加します。これらのファイルは、後続の手順で Butane 設定の一部としてインポートされます。
   - **left_server.p12**: IPsec エンドポイントの証明書バンドル
   - **ca.pem**: 証明書に署名した認証局
3. 証明書をクラスターに追加するためのマシン設定を作成します。

   a. コントロールプレーンとワーカーノードの Butane 設定ファイルを作成するには、次のコマンドを入力します。

```
$ for role in master worker; do
  cat >> "99-ipsec-${role}-endpoint-config.bu" <<-EOF
  variant: openshift
  version: 4.15.0
  metadata:
    name: 99-${role}-import-certs
    labels:
      machineconfiguration.openshift.io/role: $role
  systemd:
    units:
    - name: ipsec-import.service
      enabled: true
      contents: |
        [Unit]
        Description=Import external certs into ipsec NSS
        Before=ipsec.service

        [Service]
        Type=oneshot
        ExecStart=/usr/local/bin/ipsec-addcert.sh
        RemainAfterExit=false
        StandardOutput=journal

        [Install]
        WantedBy=multi-user.target
```

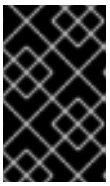
```
  storage:
    files:
    - path: /etc/pki/certs/ca.pem
      mode: 0400
      overwrite: true
      contents:
        local: ca.pem
    - path: /etc/pki/certs/left_server.p12
      mode: 0400
      overwrite: true
      contents:
        local: left_server.p12
    - path: /usr/local/bin/ipsec-addcert.sh
      mode: 0740
      overwrite: true
      contents:
        inline: |
          #!/bin/bash -e
          echo "importing cert to NSS"
          certutil -A -n "CA" -t "CT,C,C" -d /var/lib/ipsec/nss/ -i /etc/pki/certs/ca.pem
          pk12util -W "" -i /etc/pki/certs/left_server.p12 -d /var/lib/ipsec/nss/
          certutil -M -n "left_server" -t "u,u,u" -d /var/lib/ipsec/nss/
EOF
done
```

b. 前の手順で作成した Butane ファイルをマシン設定に変換するには、次のコマンドを入力します。

```
$ for role in master worker; do
  butane 99-ipsec-${role}-endpoint-config.bu -o ./99-ipsec-$role-endpoint-config.yaml
done
```

4. マシン設定をクラスターに適用するには、次のコマンドを入力します。

```
$ for role in master worker; do
  oc apply -f 99-ipsec-${role}-endpoint-config.yaml
done
```



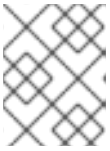
重要

Machine Config Operator (MCO) は各マシン設定プール内のマシンを更新するときに、各ノードを 1 つずつ再起動します。外部 IPsec 接続が使用可能になる前に、すべてのノードが更新されるまで待つ必要があります。

5. 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ーのサイズと共に増加します。

6. IPsec マシン設定が正常にロールアウトされたことを確認するために、次のコマンドを入力します。

   a. IPsec マシン設定が作成されたことを確認します。

   ```
   $ oc get mc | grep ipsec
   ```

   出力例

   ```
   80-ipsec-master-extensions      3.2.0      6d15h
   80-ipsec-worker-extensions      3.2.0      6d15h
   ```

   b. IPsec 拡張機能がコントロールプレーンノードに適用されていることを確認します。

   ```
   $ oc get mcp master -o yaml | grep 80-ipsec-master-extensions -c
   ```

   予想される出力

   ```
   2
   ```

   c. IPsec 拡張機能がワーカーノードに適用されていることを確認します。

   ```
   $ oc get mcp worker -o yaml | grep 80-ipsec-worker-extensions -c
   ```

   予想される出力

   ```
   2
   ```

関連情報

- nmstate IPsec API の詳細は、IPsec Encryption を参照してください。

28.10.8. 外部 IPsec エンドポイントの IPsec 暗号化の無効化

クラスター管理者は、外部ホストへの既存の IPsec トンネルを削除できます。

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。
- クラスターで **Full** または **External** モードの IPsec が有効になっている。

手順

1. 次の YAML を使用して、**remove-ipsec-tunnel.yaml** という名前のファイルを作成します。

```
kind: NodeNetworkConfigurationPolicy
apiVersion: nmstate.io/v1
metadata:
  name: <name>
spec:
  nodeSelector:
    kubernetes.io/hostname: <node_name>
  desiredState:
    interfaces:
    - name: <tunnel_name>
      type: ipsec
      state: absent
```

ここでは、以下のようになります。

name
ノードネットワーク設定ポリシーの名前を指定します。

node_name
削除する IPsec トンネルが存在するノードの名前を指定します。

tunnel_name
既存の IPsec トンネルのインターフェイス名を指定します。

2. IPsec トンネルを削除するために、次のコマンドを入力します。

```
$ oc apply -f remove-ipsec-tunnel.yaml
```

28.10.9. IPsec 暗号化の無効化

クラスター管理者は、IPsec 暗号化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** 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. IPsec 暗号化を無効にするには、次のコマンドを入力します。

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
-p '{
  "spec":{
    "defaultNetwork":{
      "ovnKubernetesConfig":{
        "ipsecConfig":{
          "mode":"Disabled"
        }}}}}'
```

2. オプション: IP パケットの IPsec ESP ヘッダーからのオーバーヘッドがなくなるため、クラスター MTU のサイズを **46** バイト増やすことができます。

28.10.10. 関連情報

- Red Hat Enterprise Linux (RHEL) 9 での IPsec を使用した VPN の設定
- Butane のインストール
- OVN-Kubernetes Container Network Interface (CNI) ネットワークプラグインについて
- クラスターネットワークの MTU 変更
- Network [operator.openshift.io/v1] API

28.11. デフォルトネットワークに外部ゲートウェイを設定する

クラスター管理者は、デフォルトネットワークに外部ゲートウェイを設定できます。

この機能には次の利点があります。

- namespace ごとの egress トラフィックの詳細制御
- 静的および動的外部ゲートウェイ IP アドレスの柔軟な設定
- IPv4 と IPv6 の両方のアドレスファミリーのサポート

28.11.1. 前提条件

- クラスターは OVN-Kubernetes ネットワークプラグインを使用します。
- インフラストラクチャーは、セカンダリー外部ゲートウェイからのトラフィックをルーティングするように設定されています。

28.11.2. OpenShift Container Platform が外部ゲートウェイ IP アドレスを決定する方法

k8s.ovn.org API グループの **AdminPolicyBasedExternalRoute** カスタムリソース (CR) を使用してセカンダリー外部ゲートウェイを設定します。この CR は、外部ゲートウェイの IP アドレスを指定するための静的アプローチと動的アプローチをサポートしています。

AdminPolicyBasedExternalRoute CR の対象となる各 namespace は、他の **AdminPolicyBasedExternalRoute** CR で選択できません。namespace には同時にセカンダリー外部ゲートウェイを含めることはできません。

ポリシーへの変更はコントローラー内で分離されます。あるポリシーの適用が失敗した場合に、他のポリシーを変更しても、他のポリシーの再試行はトリガーされません。ポリシーが再評価され、変更によって発生した可能性のある差分が適用されるのは、ポリシー自体またはポリシーに関連するオブジェクト (対象の namespace、Pod ゲートウェイ、または動的ホップからそれらをホストする namespace など) の更新が行われたときのみです。

静的割り当て

IP アドレスを直接指定します。

動的割り当て

namespace と Pod セレクター、およびオプションのネットワーク割り当て定義を使用して、IP アドレスを間接的に指定します。

- ネットワーク割り当て定義の名前が指定されている場合は、ネットワーク割り当ての外部ゲートウェイ IP アドレスが使用されます。

- ネットワーク割り当て定義の名前が指定されていない場合は、Pod 自体の外部ゲートウェイ IP アドレスが使用されます。ただし、このアプローチは、Pod が **hostNetwork** を **true** に指定して設定されている場合にのみ機能します。

28.11.3. AdminPolicyBasedExternalRoute オブジェクトの設定

次のプロパティーを使用して、クラスタースコープの **AdminPolicyBasedExternalRoute** オブジェクトを定義できます。namespace は、一度に 1 つの **AdminPolicyBasedExternalRoute** CR でのみ選択できます。

表28.13 AdminPolicyBasedExternalRoute オブジェクト

フィールド	型	説明
metadata.name	**string**	**AdminPolicyBasedExternalRoute** オブジェクトの名前を指定します。
spec.from	**string**	ルーティングポリシーが適用される namespace セレクターを指定します。外部トラフィックでは **namespaceSelector** のみがサポートされます。以下に例を示します。 `from:` `  namespaceSelector:` `  matchLabels:` `  kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059` namespace を対象にできるのは、1 つの **AdminPolicyBasedExternalRoute** CR のみです。namespace が複数の **AdminPolicyBasedExternalRoute** CR によって選択されている場合、同じ namespace を対象とする 2 番目以降の CR で **failed** エラーステータスが発生します。更新を適用するには、ポリシーが再評価され、変更が適用されるように、ポリシー自体またはポリシーに関連するオブジェクト (対象の namespace 、Pod ゲートウェイ、または動的ホップからそれらをホストする namespace など) を変更する必要があります。
spec.nextHops	**object**	パケットの転送先を指定します。**static** と **dynamic** のいずれか、または両方である必要があります。少なくとも 1 つのネクストホップを定義する必要があります。

表28.14 nextHops オブジェクト

フィールド	型	説明
static	**array**	静的 IP アドレスの配列を指定します。

フィールド	型	説明
dynamic	**array**	外部ゲートウェイターゲットとして使用するネットワーク割り当て定義で設定された Pod に対応する Pod セレクターの配列を指定します。

表28.15 **nextHops.static** オブジェクト

フィールド	型	説明
ip	**string**	次の宛先ホップの IPv4 アドレスまたは IPv6 アドレスを指定します。
bfdEnabled	**boolean**	オプション: ネットワークで双方向転送検出 (BFD) がサポートされているかどうかを指定します。デフォルト値は **false** です。

表28.16 **nextHops.dynamic** オブジェクト

フィールド	型	説明
podSelector	**string**	[set-based] (https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/#set-based-requirement) ラベルセレクターを指定して、このネットワークに一致する namespace 内の Pod をフィルターします設定。
namespaceSelector	**string**	**set-based** セレクターを指定して、**podSelector** が適用される namespace をフィルターします。このフィールドには値を指定する必要があります。
bfdEnabled	**boolean**	オプション: ネットワークで双方向転送検出 (BFD) がサポートされているかどうかを指定します。デフォルト値は **false** です。
networkAttachmentName	**string**	オプション: ネットワーク接続定義の名前を指定します。名前は、Pod に関連付けられた論理ネットワークのリストと一致する必要があります。このフィールドが指定されていない場合は、Pod のホストネットワークが使用されます。ただし、ホストネットワークを使用するには、Pod をホストネットワーク Pod として設定する必要があります。

28.11.3.1. セカンダリー外部ゲートウェイ設定の例

次の例では、**AdminPolicyBasedExternalRoute** オブジェクトは、ラベルが **kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059** の namespace 内にある Pod の外部ゲートウェイとして 2 つの静的 IP アドレスを設定します。

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: default-route-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
        kubernetes.io/metadata.name: novxlan-externalgw-ecmp-4059
  nextHops:
    static:
    - ip: "172.18.0.8"
    - ip: "172.18.0.9"
```

次の例では、**AdminPolicyBasedExternalRoute** オブジェクトが動的外部ゲートウェイを設定します。外部ゲートウェイに使用される IP アドレスは、選択した各 Pod に関連付けられた追加のネットワーク割り当てから派生します。

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: shadow-traffic-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
        externalTraffic: ""
  nextHops:
    dynamic:
    - podSelector:
        matchLabels:
          gatewayPod: ""
      namespaceSelector:
        matchLabels:
          shadowTraffic: ""
      networkAttachmentName: shadow-gateway
    - podSelector:
        matchLabels:
          gigabyteGW: ""
      namespaceSelector:
        matchLabels:
          gatewayNamespace: ""
      networkAttachmentName: gateway
```

次の例では、**AdminPolicyBasedExternalRoute** オブジェクトは静的外部ゲートウェイと動的外部ゲートウェイの両方を設定します。

```
apiVersion: k8s.ovn.org/v1
kind: AdminPolicyBasedExternalRoute
metadata:
  name: multi-hop-policy
spec:
  from:
    namespaceSelector:
      matchLabels:
```

```
      trafficType: "egress"
nextHops:
  static:
  - ip: "172.18.0.8"
  - ip: "172.18.0.9"
  dynamic:
  - podSelector:
      matchLabels:
        gatewayPod: ""
    namespaceSelector:
      matchLabels:
        egressTraffic: ""
    networkAttachmentName: gigabyte
```

28.11.4. セカンダリー外部ゲートウェイの設定

クラスター内の namespace のデフォルトネットワークに外部ゲートウェイを設定できます。

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クラスターにログインしている。

手順

1. **AdminPolicyBasedExternalRoute** オブジェクトを含む YAML ファイルを作成します。
2. 管理ポリシーベースの外部ルートを作成するには、次のコマンドを入力します。

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

   ここでは、以下のようになります。

   <file>
   前の手順で作成した YAML ファイルの名前を指定します。

   出力例

   ```
   adminpolicybasedexternalroute.k8s.ovn.org/default-route-policy created
   ```

3. 管理ポリシーベースの外部ルートが作成されたことを確認するには、次のコマンドを入力します。

   ```
   $ oc describe apbexternalroute <name> | tail -n 6
   ```

   ここでは、以下のようになります。

   <name>
   AdminPolicyBasedExternalRoute オブジェクトの名前を指定します。

   出力例

```
Status:
  Last Transition Time:  2023-04-24T15:09:01Z
  Messages:
  Configured external gateway IPs: 172.18.0.8
  Status:  Success
Events:  <none>
```

28.11.5. 関連情報

- 追加のネットワーク接続の詳細は、複数のネットワークについて を参照してください。

28.12. 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。

28.12.1. 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。

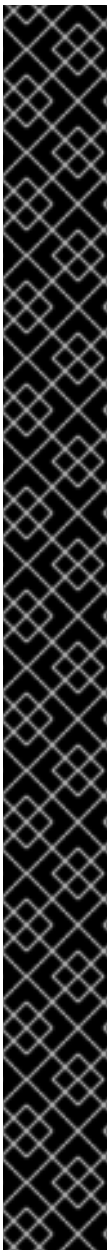
注記

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。API サーバーに接続するには、IP アドレスごとに許可ルールを追加するか、egress ポリシールールで **nodeSelector** タイプの許可ルールを使用する必要があります。

次の例は、API サーバーへのアクセスを確保するために必要な Egress ファイアウォールルールの順序を示しています。

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
  namespace: <namespace> 1
spec:
  egress:
  - to:
      cidrSelector: <api_server_address_range> 2
    type: Allow
# ...
  - to:
      cidrSelector: 0.0.0.0/0 3
    type: Deny
```

1. Egress ファイアウォールの namespace。
2. OpenShift Container Platform API サーバーを含む IP アドレス範囲。
3. 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。

28.12.1.1. 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。

これらの制限のいずれかに違反すると、プロジェクトの Egress ファイアウォールが壊れます。その結果、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、組織にセキュリティーリスクが生じ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。

28.12.1.2. 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。

28.12.1.3. 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 ファイアウォールポリシーで DNS 名を使用しても、CoreDNS を介したローカル DNS 解決には影響しません。

ただし、egress ファイアウォールポリシーでドメイン名を使用し、外部 DNS サーバーで関連する Pod の DNS 解決を処理する場合は、DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールルールを含める必要があります。

28.12.2. 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> 1
spec:
  egress: 2
    ...
```

1 オブジェクトの名前は **default** である必要があります。

2 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。

28.12.2.1. 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。ユーザーは、CIDR 形式の IP アドレス範囲またはドメイン名を選択するか、**nodeSelector** を使用して、送信トラフィックを許可または拒否できます。**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> 1
  to: 2
    cidrSelector: <cidr> 3
    dnsName: <dns_name> 4
    nodeSelector: <label_name>: <label_value> 5
  ports: 6
     ...
```

1 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。

2 **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。

3 CIDR 形式の IP アドレス範囲。

4 DNS ドメイン名。

5 ラベルは、ユーザーが定義するキーと値のペアです。ラベルは Pod などのオブジェクトに添付されます。**nodeSelector** を使用すると、1 つ以上のノードラベルを選択して、Pod に添付できます。

6 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> 1
  protocol: <protocol> 2
```

1 **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。

2 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。

28.12.2.2. 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
```

28.12.2.3. EgressFirewall の nodeSelector の例

クラスター管理者は、**nodeSelector** を使用して、ラベルを指定することにより、クラスター内のノードへのegress トラフィックを許可または拒否できます。ラベルは、1 つ以上のノードに適用できます。以下は、**region=east** ラベルを使用した例です。

```
apiVersion: k8s.ovn.org/v1
kind: EgressFirewall
metadata:
  name: default
spec:
    egress:
    - to:
        nodeSelector:
          matchLabels:
            region: east
      type: Allow
```

ヒント

ノード IP アドレスごとに手動でルールを追加する代わりに、ノードセレクターを使用して、egress ファイアウォールの背後にある Pod がホストネットワーク Pod にアクセスできるようにするラベルを作成します。

28.12.3. 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 ネットワーク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。

28.13. 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リスト表示し、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。

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

28.13.1. 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 ネットワーク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
```

28.14. 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。

28.14.1. 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 ネットワーク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
   ```

28.15. 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。

28.15.1. 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 ネットワーク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>
```

28.16. 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 の Container Network Interface (CNI) ネットワークプラグインを設定することができます。

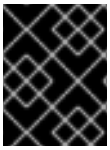
28.16.1. Egress IP アドレスアーキテクチャーの設計および実装

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。

namespace に割り当てられた egress IP アドレスは、特定の宛先にトラフィックを送信するために使用されるスロールーターとは異なります。

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。



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。

28.16.1.1. 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。

プラットフォーム	サポート対象
ベアメタル	はい
VMware vSphere	はい
Red Hat OpenStack Platform (RHOSP)	はい
Amazon Web Services (AWS)	はい
Google Cloud Platform (GCP)	はい
Microsoft Azure	はい
IBM Z® および IBM® LinuxONE	はい
IBM Z® および IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM	はい
IBM Power®	はい
Nutanix	はい

重要

EgressIP 機能を持つコントロールプレーンノードへの egress IP アドレスの割り当ては、Amazon Web Services (AWS) でプロビジョニングされるクラスターではサポートされません。(BZ#2039656)。

28.16.1.2. パブリッククラウドプラットフォームに関する考慮事項

パブリッククラウドインフラストラクチャーでプロビジョニングされたクラスターの場合は、ノードごとに割り当て可能な IP アドレスの絶対数に制約があります。ノードごとに割り当て可能な IP アドレスの最大数、つまり**IP 容量**は、次の式で表すことができます。

```
IP capacity = public cloud default capacity - sum(current IP assignments)
```

egress IP 機能はノードごとの IP アドレス容量を管理しますが、デプロイメントでこの制約を計画することが重要です。たとえば、8 ノードのベアメタルインフラストラクチャーにインストールされたクラスターの場合は、150 の egress IP アドレスを設定できます。ただし、パブリッククラウドプロバイダーが IP アドレスの容量をノードあたり 10 IP アドレスに制限している場合、割り当て可能な IP アドレスの総数はわずか 80 です。この例のクラウドプロバイダーで同じ IP アドレス容量を実現するには、7 つの追加ノードを割り当てる必要があります。

パブリッククラウド環境内の任意のノードの IP 容量とサブネットを確認するには、**oc get node <node_name> -o yaml** コマンドを入力します。**cloud.network.openshift.io/egress-ipconfig** アノテーションには、ノードの容量とサブネット情報が含まれています。

アノテーション値は、プライマリーネットワークインターフェイスに次の情報を提供するフィールドを持つ単一のオブジェクトを持つ配列です。

- **interface**: AWS と Azure のインターフェイス ID と GCP のインターフェイス名を指定します。
- **ifaddr**: 一方または両方の IP アドレスファミリーのサブネットマスクを指定します。
- **capacity**: ノードの IP アドレス容量を指定します。AWS では、IP アドレス容量は IP アドレスファミリーごとに提供されます。Azure と GCP では、IP アドレスの容量には IPv4 アドレスと IPv6 アドレスの両方が含まれます。

ノード間のトラフィックの送信 IP アドレスの自動アタッチおよびデタッチが可能です。これにより、namespace 内の多くの Pod からのトラフィックが、クラスター外の場所への一貫した送信元 IP アドレスを持つことができます。これは、OpenShift Container Platform 4.15 の Red Hat OpenShift Networking におけるデフォルトのネットワーキングプラグインである OpenShift SDN および OVN-Kubernetes もサポートします。

注記

RHOSP egress IP アドレス機能は、**egressip-<IP address>** と呼ばれる Neutron 予約ポートを作成します。OpenShift Container Platform クラスターのインストールに使用したものと同じ RHOSP ユーザーを使用して、Floating IP アドレスをこの予約ポートに割り当て、egress トラフィック用の予測可能な SNAT アドレスを指定できます。RHOSP ネットワーク上の egress IP アドレスが、ノードのフェイルオーバーなどのためにあるノードから別のノードに移動されると、Neutron 予約ポートが削除され、再作成されます。これは、フローティング IP の関連付けが失われ、フローティング IP アドレスを新しい予約ポートに手動で再割り当てする必要があることを意味します。

注記

RHOSP クラスター管理者が Floating IP を予約ポートに割り当てると、OpenShift Container Platform は予約ポートを削除できません。RHOSP クラスター管理者が予約ポートから Floating IP の割り当てを解除するまで、**CloudPrivateIPConfig** オブジェクトは削除および移動操作を実行できません。

次の例は、いくつかのパブリッククラウドプロバイダーのノードからのアノテーションを示しています。アノテーションは、読みやすくするためにインデントされています。

AWS での cloud.network.openshift.io/egress-ipconfig アノテーションの例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

GCP での cloud.network.openshift.io/egress-ipconfig アノテーションの例

```
cloud.network.openshift.io/egress-ipconfig: [
```

```
  {
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]
```

次のセクションでは、容量計算で使用するためにサポートされているパブリッククラウド環境の IP アドレス容量を説明します。

28.16.1.2.1. Amazon Web Services (AWS) の IP アドレス容量の制限

AWS では、IP アドレスの割り当てに関する制約は、設定されているインスタンスタイプによって異なります。詳細は、IP addresses per network interface per instance type を参照してください。

28.16.1.2.2. Google Cloud Platform (GCP) の IP アドレス容量の制限

GCP では、ネットワークモデルは、IP アドレスの割り当てではなく、IP アドレスのエイリアス作成を介して追加のノード IP アドレスを実装します。ただし、IP アドレス容量は IP エイリアス容量に直接マッピングされます。

IP エイリアスの割り当てには、次の容量制限があります。

- ノードごとに、IPv4 と IPv6 の両方の IP エイリアスの最大数は 100 です。
- VPC ごとに、IP エイリアスの最大数は指定されていませんが、OpenShift Container Platform のスケーラビリティーテストでは、最大数が約 15,000 であることが明らかになっています。

詳細は、インスタンスごとのクォータとエイリアス IP 範囲の概要を参照してください。

28.16.1.2.3. Microsoft Azure IP アドレスの容量制限

Azure では、IP アドレスの割り当てに次の容量制限があります。

- NIC ごとに、IPv4 と IPv6 の両方で割り当て可能な IP アドレスの最大数は 256 です。
- 仮想ネットワークごとに、割り当てられる IP アドレスの最大数は 65,536 を超えることはできません。

詳細は、ネットワークの制限を参照してください。

28.16.1.3. 追加のネットワークインターフェイスで egress IP を使用する場合の考慮事項

OpenShift Container Platform では、egress IP は管理者にネットワークトラフィックを制御する方法を提供します。Egress IP は、Open vSwitch に関連付けられた Linux ブリッジインターフェイスである **br-ex** (プライマリー) ネットワークインターフェイスで使用することも、追加のネットワークインターフェイスで使用することもできます。

次のコマンドを実行して、ネットワークインターフェイスのタイプを検査できます。

```
$ ip -details link show
```

プライマリーネットワークインターフェイスには、サブネットマスクも含まれるノード IP アドレスが割り当てられます。このノードの IP アドレスの情報は、**k8s.ovn.org/node-primary-ifaddr** アノテーションを検査して、クラスター内の各ノードの Kubernetes ノードオブジェクトから取得できます。

IPv4 クラスターでは、このアノテーションは **"k8s.ovn.org/node-primary-ifaddr: {"ipv4":"192.168.111.23/24"}"** の例に似ています。

egress IP がプライマリーネットワークインターフェイスのサブネット内にない場合は、プライマリーネットワークインターフェイスタイプではない別の Linux ネットワークインターフェイスで egress IP を使用できます。これにより、OpenShift Container Platform 管理者は、ルーティング、アドレス指定、セグメンテーション、セキュリティーポリシーなどのネットワーク側面をより高度に制御できるようになります。この機能は、トラフィックのセグメント化や特殊な要件への対応などの目的で、ワークロードトラフィックを特定のネットワークインターフェイス経由でルーティングするオプションもユーザーに提供します。

egress IP がプライマリーネットワークインターフェイスのサブネット内にない場合、ノード上に egress トラフィック用の別のネットワークインターフェイスが存在すると、そのネットワークインターフェイスが選択される可能性があります。

k8s.ovn.org/host-cidrs Kubernetes ノードのアノテーションを検査することで、他のどのネットワークインターフェイスが egress IP をサポートしているかを判断できます。このアノテーションには、プライマリーネットワークインターフェイスで見つかったアドレスとサブネットマスクが含まれています。また、追加のネットワークインターフェイスアドレスとサブネットマスク情報も含まれます。これらのアドレスとサブネットマスクは、最長接頭辞マッチルーティング メカニズムを使用して、どのネットワークインターフェイスが egress IP をサポートするかを決定するネットワークインターフェイスに割り当てられます。

注記

OVN-Kubernetes は、特定の namespace および Pod からのアウトバウンドネットワークトラフィックを制御および送信するメカニズムを提供します。これにより、特定のネットワークインターフェイス経由で、特定の egress IP アドレスを使用してクラスターからトラフィックが出ていきます。

プライマリーネットワークインターフェイスではないネットワークインターフェイスに egress IP を割り当てる場合の要件

egress IP とトラフィックをプライマリーネットワークインターフェイスではない特定のインターフェイス経由でルーティングすることを希望する場合は、次の条件を満たす必要があります。

- OpenShift Container Platform がベアメタルクラスターにインストールされている。この機能は、クラウドまたはハイパーバイザー環境では無効になります。
- OpenShift Container Platform Pod はホストネットワークとして設定されていません。
- ネットワークインターフェイスが削除された場合、またはインターフェイス上で egress IP をホストできるようにする IP アドレスとサブネットマスクが削除された場合、egress IP は再設定されます。その結果、別のノードおよびインターフェイスに割り当てられる可能性があります。
- Egress IP は IPv4 である必要があります。IPv6 は現在サポートされていません。
- ネットワークインターフェイスに対して IP フォワーディングを有効にする必要があります。IP 転送を有効にするには、**oc edit network.operator** コマンドを使用し、次の例のようにオブジェクトを編集します。

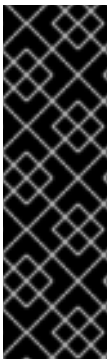
```
# ...
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
```

```
  defaultNetwork:
    ovnKubernetesConfig:
      gatewayConfig:
        ipForwarding: Global
# ...
```

28.16.1.4. 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。

28.16.1.5. 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。

28.16.1.6. 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。

28.16.2. 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> 1
spec:
  egressIPs: 2
  - <ip_address>
  namespaceSelector: 3
    ...
  podSelector: 4
    ...
```

1. **EgressIPs** オブジェクトの名前。
2. 1 つ以上の IP アドレスの配列。
3. egress IP アドレスを関連付ける namespace の 1 つ以上のセレクター。
4. 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: 1
  matchLabels:
    <label_name>: <label_value>
```

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

1 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
```

28.16.3. EgressIPconfig オブジェクト

egress IP の機能として、**reachabilityTotalTimeoutSeconds** パラメーターは、プローブによって egress IP ノードに送信されるチェックの合計タイムアウトを設定します。**egressIPConfig** オブジェクトを使用すると、ユーザーは **reachabilityTotalTimeoutSeconds spec** を設定できます。このタイムアウト内に EgressIP ノードに到達できない場合、ノードはダウンしていると宣言されます。

ネットワークが現在のデフォルト値である 1 秒を処理できるほど安定していない場合は、この値を増やすことができます。

次の YAML は、**reachabilityTotalTimeoutSeconds** をデフォルトの 1 秒プローブから 5 秒プローブに変更する方法を記述しています。

```
apiVersion: operator.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      egressIPConfig: (1)
        reachabilityTotalTimeoutSeconds: 5 (2)
      gatewayConfig:
        routingViaHost: false
      genevePort: 6081
```

(1) **egressIPConfig** は、**EgressIP** オブジェクトのオプション設定を保持します。これらの設定を変更すると、**EgressIP** オブジェクトを拡張できます。

(2) **reachabilityTotalTimeoutSeconds** の値としては、**0** から **60** までの整数値が許可されます。値が 0 の場合、egressIP ノードの到達性チェックは無効になります。値が **1** から **60** の場合、ノードの到達性チェックを送信するプローブ間の時間 (秒単位) を表します。

28.16.4. 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>
```

28.16.5. 次のステップ

- egress IP の割り当て

28.16.6. 関連情報

- LabelSelector meta/v1
- LabelSelectorRequirement meta/v1

28.17. 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。

28.17.1. 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
```

2. 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 1
```

   1 **<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
```

3. 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。

4. 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 1
```

   1 **<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。

28.17.2. 関連情報

- egress IP アドレスの設定

28.18. 出力サービスの設定

クラスター管理者は、egress サービスを使用して、ロードバランサーサービスの背後にある Pod の egress トラフィックを設定できます。

重要

Egress サービス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 製品サポート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場合があります。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。

EgressService カスタムリソース (CR) を使用して、次の方法で送信トラフィックを管理できます。

- ロードバランサーサービスの IP アドレスを、ロードバランサーサービスの背後にある Pod の送信トラフィックの送信元 IP アドレスとして割り当てます。
  このコンテキストでは、ロードバランサーの IP アドレスを送信元 IP アドレスとして割り当てると、単一の egress と ingress を提供するのに役立ちます。たとえば、一部のシナリオでは、ロードバランサーサービスの背後でアプリケーションと通信する外部システムは、アプリケーションの送信元 IP アドレスと宛先 IP アドレスが同じであると想定できます。

  注記

  ロードバランサーサービスの IP アドレスをサービスの背後にある Pod の egress トラフィックに割り当てると、OVN-Kubernetes は ingress ポイントと egress ポイントを単一のノードに制限します。これにより、MetalLB が通常提供するトラフィックのロードバランシングが制限されます。

- ロードバランサーの背後にある Pod の Egress トラフィックを、デフォルトノードネットワークとは異なるネットワークに割り当てます。
  これは、ロードバランサーの背後にあるアプリケーションの egress トラフィックを、デフォルトネットワークとは異なるネットワークに割り当てる場合に便利です。通常、別のネットワークは、ネットワークインターフェイスに関連付けられた VRF インスタンスを使用して実装されます。

28.18.1. Egress サービスのカスタムリソース

EgressService カスタムリソースで egress サービスの設定を定義します。次の YAML は、egress サービスの設定のフィールドを説明します。

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: <egress_service_name> 1
  namespace: <namespace> 2
spec:
  sourceIPBy: <egress_traffic_ip> 3
  nodeSelector: 4
    matchLabels:
      node-role.kubernetes.io/<role>: ""
  network: <egress_traffic_network> 5
```

1 Egress サービスの名前を指定します。**EgressService** リソースの名前は、変更するロードバランサーサービスの名前と一致する必要があります。

2 Egress サービスの namespace を指定します。**EgressService** の namespace は、変更するロードバランサーサービスの namespace と一致する必要があります。egress サービスは namespace ス

3 サービスの背後にある Pod の egress トラフィックの送信元 IP アドレスを指定します。有効な値は、**LoadBalancerIP** または **Network** です。**LoadBalancerIP** 値を使用して、**LoadBalancer** サービスの ingress IP アドレスを egress トラフィックの送信元 IP アドレスとして割り当てます。**ネットワーク** を指定して、ネットワークインターフェイス IP アドレスを egress トラフィックの送信元 IP アドレスとして割り当てます。

4 オプション: **sourceIPBy** 指定に **LoadBalancerIP** 値を使用する場合、単一ノードが **LoadBalancer** サービストラフィックを処理します。このタスクを割り当て可能なノードを制限するには、**nodeSelector** フィールドを使用します。サービストラフィックを処理するノードが選択されると、OVN-Kubernetes はそのノードに **egress-service.k8s.ovn.org/<svc-namespace>-<svc-name>: ""** という形式でラベルを付けます。**nodeSelector** フィールドが指定されていない場合、どのノードでも **LoadBalancer** サービストラフィックを管理できます。

5 オプション: 出力トラフィックのルーティングテーブルを指定します。**network** 仕様を含めない場合、egress サービスはデフォルトのホストネットワークを使用します。

egress サービス仕様の例

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: test-egress-service
  namespace: test-namespace
spec:
  sourceIPBy: "LoadBalancerIP"
  nodeSelector:
    matchLabels:
      vrf: "true"
  network: "2"
```

28.18.2. Egress サービスのデプロイ

Egress サービスをデプロイして、**LoadBalancer** サービスの背後にある Pod の Egress トラフィックを管理できます。

次の例では、**LoadBalancer** サービスの ingress IP アドレスと同じ送信元 IP アドレスを持つように egress トラフィックを設定します。

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 **BGPPeer** リソースを設定している。

手順

1. サービスに必要な IP を使用して **IPAddressPool** CR を作成します。
   a. 次の例のような内容を含むファイル (**ip-addr-pool.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: example-pool
  namespace: metallb-system
spec:
  addresses:
  - 172.19.0.100/32
```

b. 次のコマンドを実行して、IP アドレスプールの設定を適用します。

```
$ oc apply -f ip-addr-pool.yaml
```

2. **Service** CR および **EgressService** CR を作成します。

a. 次の例のような内容を含むファイル **(service-egress-service.yaml** など) を作成します。

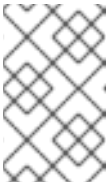
```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
  annotations:
    metallb.universe.tf/address-pool: example-pool 1
spec:
  selector:
    app: example
  ports:
    - name: http
      protocol: TCP
      port: 8080
      targetPort: 8080
  type: LoadBalancer
---
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: example-service
  namespace: example-namespace
spec:
  sourceIPBy: "LoadBalancerIP" 2
  nodeSelector: 3
    matchLabels:
      node-role.kubernetes.io/worker: ""
```

1 **LoadBalancer** サービスは 、**example-pool** IP アドレスプールから MetalLB によって割り当てられた IP アドレスを使用します。

2 この例では、**LoadBalancerIP** 値を使用して、**LoadBalancer** サービスの ingress IP アドレスを egress トラフィックの送信元 IP アドレスとして割り当てます。

3 **LoadBalancerIP** 値を指定すると、単一ノードが **LoadBalancer** サービスのトラフィックを処理します。この例では、トラフィックを処理する場合に **worker** ラベルが割り当てられたノードのみを選択できます。ノードが選択されると、OVN-Kubernetes はそのノードに **egress-service.k8s.ovn.org/<svc-namespace>-<svc-**

name>: "" という形式でラベルを付けます。



注記

sourceIPBy: "LoadBalancerIP" 設定を使用する場合は、**BGPAdvertisement** カスタムリソース (CR) でロードバランサーノードを指定する必要があります。

b. 次のコマンドを実行して、サービスと egress サービスの設定を適用します。

```
$ oc apply -f service-egress-service.yaml
```

3. **BGPAdvertisement** CR を作成してサービスをアドバタイズします。

a. 次の例のような内容を含むファイル **(service-bgp-advertisement.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: example-bgp-adv
  namespace: metallb-system
spec:
  ipAddressPools:
  - example-pool
  nodeSelector:
  - matchLabels:
      egress-service.k8s.ovn.org/example-namespace-example-service: "" (1)
```

(1) この例では、**EgressService** CR は、ロードバランサーサービス IP アドレスを使用するように、egress トラフィックの送信元 IP アドレスを設定します。したがって、Pod から発信されるトラフィックに同じリターンパスを使用するには、リターントラフィックのロードバランサーノードを指定する必要があります。

検証

1. 次のコマンドを実行して、MetalLB サービスの背後で実行されている Pod のアプリケーションエンドポイントにアクセスできることを確認します。

```
$ curl <external_ip_address>:<port_number> (1)
```

(1) アプリケーションのエンドポイントに合わせて外部 IP アドレスとポート番号を更新します。

2. **LoadBalancer** サービスの ingress IP アドレスを egress トラフィックの送信元 IP アドレスとして割り当てた場合は、**tcpdump** などのツールを使用して外部クライアントで受信したパケットを分析し、この設定を確認します。

関連情報

- ネットワーク VRF を介したサービスの公開

- 例: VRF インスタンスノードのネットワーク設定ポリシーを使用したネットワークインターフェイス
- MetalLB を使用した対称ルーティングの管理
- 仮想ルーティングおよび転送について

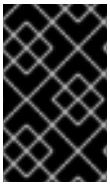
28.19. EGRESS ルーター POD の使用についての考慮事項

28.19.1. 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ー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。

28.19.1.1. 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。予約された送信元 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、スルールーターのサービスにアクセスするように設定する必要があります。**curl** コマンドを使用して、アプリケーション Pod から宛先サービスとポートにアクセスできます。以下に例を示します。

```
$ curl <router_service_IP> <port>
```

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 プロキシーモードまたは DNS プロキシーモード をサポートしません。

28.19.1.2. 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リストの 2 つのオブジェクトを削除します。

28.19.1.3. 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>
```

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

28.19.1.4. 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
```

28.19.2. 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

28.20. 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。

28.20.1. 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
    }
  ]
}
```

28.20.2. 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のあるクライアント Pod は、宛先 IP に直接接続するのでなく、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 {Ifindex: 3 Dst: <nil> Src: <nil> Gw:
10.128.10.1 Flags: [] Table: 254}
2021-04-26T12:27:20Z [debug] Added new default route with gateway 192.168.12.1
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p
UDP --dport 80 -j DNAT --to-destination 10.0.0.99
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p
TCP --dport 8080 -j DNAT --to-destination 203.0.113.26:80
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat PREROUTING -i eth0 -p
TCP --dport 8443 -j DNAT --to-destination 203.0.113.27:443
2021-04-26T12:27:20Z [debug] Added iptables rule: iptables -t nat -o net1 -j SNAT --to-
source 192.168.12.99
```

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、コンテナー ID を取得します。

```
# crictl ps --name egress-router-cni-pod | awk '{print $1}'
```

出力例

```
CONTAINER
bac9fae69ddb6
```

f. コンテナーのプロセス ID を判別します。この例では、コンテナー ID は **bac9fae69ddb6** です。

```
# crictl inspect -o yaml bac9fae69ddb6 | grep 'pid:' | awk '{print $2}'
```

出力例

```
68857
```

g. コンテナーのネットワーク 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
```

28.21. プロジェクトのマルチキャストの有効化

28.21.1. 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。
- デフォルトでは、ネットワークポリシーは namespace 内のすべての接続に影響します。ただし、マルチキャストはネットワークポリシーの影響を受けません。マルチキャストがネットワークポリシーと同じ namespace で有効にされている場合、**deny-all** ネットワークポリシーがある場合でも、マルチキャストは常に許可されます。クラスター管理者は、これを有効にする前に、ネットワークポリシーからマルチキャストが除外されることの影響を考慮する必要があります。

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 ネットワークプラグイン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。

28.21.2. 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/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
```

```
        name: mlistener
        protocol: UDP
EOF
```

3. マルチキャストセンダーとして機能する Pod を作成します。

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. 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
   ```

28.22. プロジェクトのマルチキャストの無効化

28.22.1. 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
```

28.23. 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ー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。

28.23.1. 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28.17 ネットワークフローの設定

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.exportNetworkFlows	**object**	1 つ以上の **netFlow**、**sFlow**、または **ipfix**。
spec.exportNetworkFlows.netFlow.collectors	**array**	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リスト。
spec.exportNetworkFlows.sFlow.collectors	**array**	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リスト。
spec.exportNetworkFlows.ipfix.collectors	**array**	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リスト。

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
```

28.23.2. 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-controller $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"]-

...
```

28.23.3. 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
```

28.23.4. 関連情報

- Network [operator.openshift.io/v1]

28.24. ハイブリッドネットワークの設定

クラスター管理者は、Red Hat OpenShift Networking OVN-Kubernetes ネットワークプラグインを設定して、Linux および Windows ノードがそれぞれ Linux および Windows ワークロードをホストできるようにすることができます。

28.24.1. OVN-Kubernetes を使用したハイブリッドネットワークの設定

OVN-Kubernetes ネットワークプラグインを使用して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。

注記

この設定は、同じクラスター内で Linux ノードと Windows ノードの両方を実行するために必要です。

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。
- クラスターが OVN-Kubernetes ネットワークプラグインを使用していることを確認します。

手順

1. OVN-Kubernetes ハイブリッドネットワークオーバーレイを設定するには、次のコマンドを入力します。

```
$ oc patch networks.operator.openshift.io cluster --type=merge \
  -p '{
    "spec":{
      "defaultNetwork":{
        "ovnKubernetesConfig":{
          "hybridOverlayConfig":{
            "hybridClusterNetwork":[
              {
                "cidr": "<cidr>",
                "hostPrefix": <prefix>
              }
            ],
```

```
          "hybridOverlayVXLANPort": <overlay_port>
        }
      }
    }
  }
}'
```

ここでは、以下のようになります。

cidr

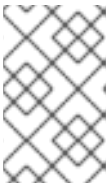
追加のオーバーレイネットワーク上のノードに使用される CIDR 設定を指定します。この CIDR はクラスターネットワーク CIDR と重複できません。

hostPrefix

それぞれの個別ノードに割り当てるサブネット接頭辞の長さを指定します。たとえば、**hostPrefix** が **23** に設定されている場合、各ノードに指定の **cidr** から **/23** サブネットが割り当てられます。これにより、510 (2^(32 - 23) - 2) Pod IP アドレスが許可されます。外部ネットワークからのノードへのアクセスを提供する必要がある場合には、ロードバランサーおよびルーターを、トラフィックを管理するように設定します。

hybridOverlayVXLANPort

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。

出力例

```
network.operator.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 network.operator.openshift.io -o jsonpath="
{.items[0].spec.defaultNetwork.ovnKubernetesConfig}"
```

28.24.2. 関連情報

- Windows コンテナーワークロードについて
- Windows コンテナー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29章 OPENSHIFT SDN ネットワークプラグイン

29.1. OPENSHIFT SDN ネットワークプラグインについて

Red Hat OpenShift Networking の一部である OpenShift SDN は、ソフトウェア定義ネットワーキング (SDN) アプローチを使用して、OpenShift Container Platform クラスター全体の Pod 間の通信を可能にする統合クラスターネットワークを提供するネットワークプラグインです。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。

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

29.1.1. 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 4.15 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。

29.1.2. サポートされているネットワークプラグイン機能のマトリックス

Red Hat OpenShift Networking は、 ネットワークプラグイン用に OpenShift SDN と OVN-Kubernetes の 2 つのオプションを提供します。以下の表は、両方のネットワークプラグインの現在の機能サポートをまとめたものです。

表29.1 デフォルトの CNI ネットワークプラグイン機能の比較

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

機能	OpenShift SDN	OVN-Kubernetes
Egress ルーター	サポート対象	サポート対象 [2]
ハイブリッドネットワーク	サポート対象外	サポート対象
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][4]
Kubernetes ネットワークポリシー	サポート対象	サポート対象
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
ハードウェアのオフロード	サポート対象外	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、vSphere、IBM Power®、IBM Z®、および Red Hat OpenStack クラスターでのみサポートされます。
4. IPv6 シングルスタックは、IBM Power®、IBM Z®、および Red Hat OpenStack クラスターではサポートされません。

29.2. プロジェクトの EGRESS IP の設定

クラスター管理者は、OpenShift SDN の Container Network Interface (CNI) ネットワークプラグインが 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。

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

29.2.1. 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。

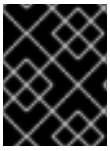
たとえば、クラスター外のサーバーでホストされるデータベースを定期的にクエリーする Pod がある

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。

namespace に割り当てられた egress IP アドレスは、特定の宛先にトラフィックを送信するために使用されるスロールーターとは異なります。

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。

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。



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。

29.2.1.1. 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。

プラットフォーム	サポート対象
ベアメタル	はい
VMware vSphere	はい
Red Hat OpenStack Platform (RHOSP)	はい
Amazon Web Services (AWS)	はい
Google Cloud Platform (GCP)	はい
Microsoft Azure	はい
IBM Z® および IBM® LinuxONE	はい
IBM Z® および IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM	はい
IBM Power®	はい
Nutanix	はい

重要

EgressIP 機能を持つコントロールプレーンノードへの egress IP アドレスの割り当ては、Amazon Web Services (AWS) でプロビジョニングされるクラスターではサポートされません。(BZ#2039656)。

29.2.1.2. パブリッククラウドプラットフォームに関する考慮事項

パブリッククラウドインフラストラクチャーでプロビジョニングされたクラスターの場合は、ノードごとに割り当て可能な IP アドレスの絶対数に制約があります。ノードごとに割り当て可能な IP アドレスの最大数、つまり**IP 容量**は、次の式で表すことができます。

```
IP capacity = public cloud default capacity - sum(current IP assignments)
```

egress IP 機能はノードごとの IP アドレス容量を管理しますが、デプロイメントでこの制約を計画することが重要です。たとえば、8 ノードのベアメタルインフラストラクチャーにインストールされたクラスターの場合は、150 の egress IP アドレスを設定できます。ただし、パブリッククラウドプロバイダーが IP アドレスの容量をノードあたり 10 IP アドレスに制限している場合、割り当て可能な IP アドレスの総数はわずか 80 です。この例のクラウドプロバイダーで同じ IP アドレス容量を実現するには、7 つの追加ノードを割り当てる必要があります。

パブリッククラウド環境内の任意のノードの IP 容量とサブネットを確認するには、**oc get node <node_name> -o yaml** コマンドを入力します。**cloud.network.openshift.io/egress-ipconfig** アノテーションには、ノードの容量とサブネット情報が含まれています。

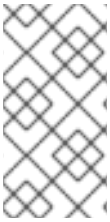
アノテーション値は、プライマリーネットワークインターフェイスに次の情報を提供するフィールドを持つ単一のオブジェクトを持つ配列です。

- **interface**: AWS と Azure のインターフェイス ID と GCP のインターフェイス名を指定します。
- **ifaddr**: 一方または両方の IP アドレスファミリーのサブネットマスクを指定します。
- **capacity**: ノードの IP アドレス容量を指定します。AWS では、IP アドレス容量は IP アドレスファミリーごとに提供されます。Azure と GCP では、IP アドレスの容量には IPv4 アドレスと IPv6 アドレスの両方が含まれます。

ノード間のトラフィックの送信 IP アドレスの自動アタッチおよびデタッチが可能です。これにより、namespace 内の多くの Pod からのトラフィックが、クラスター外の場所への一貫した送信元 IP アドレスを持つことができます。これは、OpenShift Container Platform 4.15 の Red Hat OpenShift Networking におけるデフォルトのネットワーキングプラグインである OpenShift SDN および OVN-Kubernetes もサポートします。

注記

RHOSP egress IP アドレス機能は、**egressip-<IP address>** と呼ばれる Neutron 予約ポートを作成します。OpenShift Container Platform クラスターのインストールに使用したものと同じ RHOSP ユーザーを使用して、Floating IP アドレスをこの予約ポートに割り当て、egress トラフィック用の予測可能な SNAT アドレスを指定できます。RHOSP ネットワーク上の egress IP アドレスが、ノードのフェイルオーバーなどのためにあるノードから別のノードに移動されると、Neutron 予約ポートが削除され、再作成されます。これは、フローティング IP の関連付けが失われ、フローティング IP アドレスを新しい予約ポートに手動で再割り当てする必要があることを意味します。



注記

RHOSP クラスター管理者が Floating IP を予約ポートに割り当てると、OpenShift Container Platform は予約ポートを削除できません。RHOSP クラスター管理者が予約ポートから Floating IP の割り当てを解除するまで、**CloudPrivateIPConfig** オブジェクトは削除および移動操作を実行できません。

次の例は、いくつかのパブリッククラウドプロバイダーのノードからのアノテーションを示しています。アノテーションは、読みやすくするためにインデントされています。

AWS での cloud.network.openshift.io/egress-ipconfig アノテーションの例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"eni-078d267045138e436",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ipv4":14,"ipv6":15}
  }
]
```

GCP での cloud.network.openshift.io/egress-ipconfig アノテーションの例

```
cloud.network.openshift.io/egress-ipconfig: [
  {
    "interface":"nic0",
    "ifaddr":{"ipv4":"10.0.128.0/18"},
    "capacity":{"ip":14}
  }
]
```

次のセクションでは、容量計算で使用するためにサポートされているパブリッククラウド環境の IP アドレス容量を説明します。

29.2.1.2.1. Amazon Web Services (AWS) の IP アドレス容量の制限

AWS では、IP アドレスの割り当てに関する制約は、設定されているインスタンスタイプによって異なります。詳細は、IP addresses per network interface per instance type を参照してください。

29.2.1.2.2. Google Cloud Platform (GCP) の IP アドレス容量の制限

GCP では、ネットワークモデルは、IP アドレスの割り当てではなく、IP アドレスのエイリアス作成を介して追加のノード IP アドレスを実装します。ただし、IP アドレス容量は IP エイリアス容量に直接マッピングされます。

IP エイリアスの割り当てには、次の容量制限があります。

- ノードごとに、IPv4 と IPv6 の両方の IP エイリアスの最大数は 100 です。
- VPC ごとに、IP エイリアスの最大数は指定されていませんが、OpenShift Container Platform のスケーラビリティーテストでは、最大数が約 15,000 であることが明らかになっています。

詳細は、インスタンスごとのクォータとエイリアス IP 範囲の概要を参照してください。

29.2.1.2.3. Microsoft Azure IP アドレスの容量制限

Azure では、IP アドレスの割り当てに次の容量制限があります。

- NIC ごとに、IPv4 と IPv6 の両方で割り当て可能な IP アドレスの最大数は 256 です。
- 仮想ネットワークごとに、割り当てられる IP アドレスの最大数は 65,536 を超えることはできません。

詳細は、ネットワークの制限を参照してください。

29.2.1.3. 追加のネットワークインターフェイスで egress IP を使用する場合の考慮事項

OpenShift Container Platform では、egress IP は管理者にネットワークトラフィックを制御する方法を提供します。Egress IP は、Open vSwitch に関連付けられた Linux ブリッジインターフェイスである **br-ex** (プライマリー) ネットワークインターフェイスで使用することも、追加のネットワークインターフェイスで使用することもできます。

次のコマンドを実行して、ネットワークインターフェイスのタイプを検査できます。

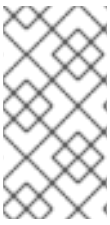
```
$ ip -details link show
```

プライマリーネットワークインターフェイスには、サブネットマスクも含まれるノード IP アドレスが割り当てられます。このノードの IP アドレスの情報は、**k8s.ovn.org/node-primary-ifaddr** アノテーションを検査して、クラスター内の各ノードの Kubernetes ノードオブジェクトから取得できます。IPv4 クラスターでは、このアノテーションは **"k8s.ovn.org/node-primary-ifaddr: {"ipv4":"192.168.111.23/24"}"** の例に似ています。

egress IP がプライマリーネットワークインターフェイスのサブネット内にない場合は、プライマリーネットワークインターフェイスタイプではない別の Linux ネットワークインターフェイスで egress IP を使用できます。これにより、OpenShift Container Platform 管理者は、ルーティング、アドレス指定、セグメンテーション、セキュリティーポリシーなどのネットワーク側面をより高度に制御できるようになります。この機能は、トラフィックのセグメント化や特殊な要件への対応などの目的で、ワークロードトラフィックを特定のネットワークインターフェイス経由でルーティングするオプションもユーザーに提供します。

egress IP がプライマリーネットワークインターフェイスのサブネット内にない場合、ノード上に egress トラフィック用の別のネットワークインターフェイスが存在すると、そのネットワークインターフェイスが選択される可能性があります。

k8s.ovn.org/host-cidrs Kubernetes ノードのアノテーションを検査することで、他のどのネットワークインターフェイスが egress IP をサポートしているかを判断できます。このアノテーションには、プライマリーネットワークインターフェイスで見つかったアドレスとサブネットマスクが含まれています。また、追加のネットワークインターフェイスアドレスとサブネットマスク情報も含まれます。これらのアドレスとサブネットマスクは、最長接頭辞マッチルーティング メカニズムを使用して、どのネットワークインターフェイスが egress IP をサポートするかを決定するネットワークインターフェイスに割り当てられます。



注記

OVN-Kubernetes は、特定の namespace および Pod からのアウトバウンドネットワークトラフィックを制御および送信するメカニズムを提供します。これにより、特定のネットワークインターフェイス経由で、特定の egress IP アドレスを使用してクラスターからトラフィックが出ていきます。

プライマリーネットワークインターフェイスではないネットワークインターフェイスに egress IP を割り当てる場合の要件

egress IP とトラフィックをプライマリーネットワークインターフェイスではない特定のインターフェイス経由でルーティングすることを希望する場合は、次の条件を満たす必要があります。

- OpenShift Container Platform がベアメタルクラスターにインストールされている。この機能は、クラウドまたはハイパーバイザー環境では無効になります。
- OpenShift Container Platform Pod はホストネットワークとして設定されていません。
- ネットワークインターフェイスが削除された場合、またはインターフェイス上で egress IP をホストできるようにする IP アドレスとサブネットマスクが削除された場合、egress IP は再設定されます。その結果、別のノードおよびインターフェイスに割り当てられる可能性があります。
- Egress IP は IPv4 である必要があります。IPv6 は現在サポートされていません。
- ネットワークインターフェイスに対して IP フォワーディングを有効にする必要があります。IP 転送を有効にするには、**oc edit network.operator** コマンドを使用し、次の例のようにオブジェクトを編集します。

```
# ...
spec:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  defaultNetwork:
    ovnKubernetesConfig:
      gatewayConfig:
        ipForwarding: Global
# ...
```

== 制限

OpenShift SDN ネットワークプラグインで egress IP アドレスを使用する場合、次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 ネットワークプラグインの 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。

29.2.1.4. IP アドレス割り当てアプローチ

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 アドレス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 アドレスをホストに割り当てることを可能にします。

- **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 アドレスのリスト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。

29.2.1.4.1. 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。

29.2.1.4.2. 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

このアプローチにより、egress IP アドレスをホストできるノードを制御できます。

注記

クラスターがパブリッククラウドインフラストラクチャーにインストールされている場合は、egress IP アドレスを割り当てる各ノードに、IP アドレスをホストするための十分な予備容量があることを確認する必要があります。詳細については、前のセクションのプラットフォームに関する考慮事項を参照してください。

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。

29.2.2. 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。

2. 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、以下のようになります。

<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。

29.2.3. 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

1. 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 アドレスをノードホストに手動で割り当てます。
クラスターがパブリッククラウドインフラストラクチャーにインストールされている場合は、ノードに使用可能な IP アドレス容量があることを確認する必要があります。

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"]}'
```

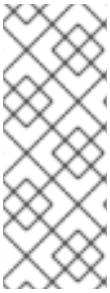
直前の例では、**project1** のすべての egress トラフィックは、指定された egress IP をホストするノードにルーティングされてから、その IP アドレスに Network Address Translation (NAT) を使用して接続されます。

29.2.4. 関連情報

- 手動の egress IP アドレス割り当てを設定している場合は、IP 容量計画の情報について、プラットフォームの考慮事項を参照してください。

29.3. 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。



注記

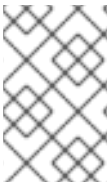
OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

29.3.1. 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。

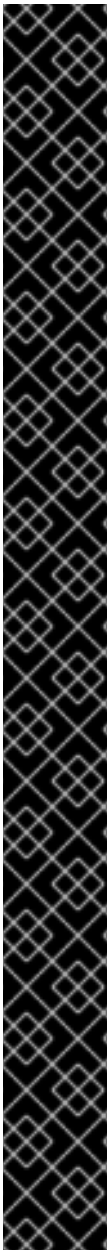


注記

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。API サーバーに接続するには、IP アドレスごとに許可ルールを追加するか、egress ポリシールールで **nodeSelector** タイプの許可ルールを使用する必要があります。

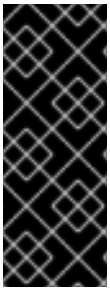
次の例は、API サーバーへのアクセスを確保するために必要な Egress ファイアウォールルールの順序を示しています。

```
apiVersion: network.openshift.io/v1
kind: EgressNetworkPolicy
metadata:
  name: default
  namespace: <namespace> 1
spec:
  egress:
  - to:
      cidrSelector: <api_server_address_range> 2
    type: Allow
# ...
  - to:
      cidrSelector: 0.0.0.0/0 3
    type: Deny
```

1. Egress ファイアウォールの namespace。
2. OpenShift Container Platform API サーバーを含む IP アドレス範囲。
3. 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。

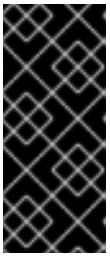
警告

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。

29.3.1.1. 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。



重要

複数の EgressNetworkPolicy オブジェクトの作成は可能ですが、作成するべきではありません。複数の EgressNetworkPolicy オブジェクトを作成すると、**dropping all rules** というメッセージが返されます。実際には、すべての外部トラフィックがドロップされるため、組織にセキュリティーリスクが生じる可能性があります。

- 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 ネットワークプラグインを使用する場合、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。

これらの制限のいずれかに違反すると、プロジェクトの Egress ファイアウォールが壊れます。その結果、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、組織にセキュリティーリスクが生じ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。

29.3.1.2. 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。

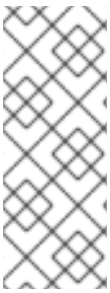
29.3.1.3. DNS (Domain Name Server) 解決の仕組み

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 ファイアウォールポリシーで DNS 名を使用しても、CoreDNS を介したローカル DNS 解決には影響しません。

ただし、egress ファイアウォールポリシーでドメイン名を使用し、外部 DNS サーバーで関連する Pod の DNS 解決を処理する場合は、DNS サーバーの IP アドレスへのアクセスを許可する egress ファイアウォールルールを含める必要があります。

29.3.2. 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> 1
spec:
  egress: 2
    ...
```

1 egress ファイアウォールポリシーの名前。

2 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。

29.3.2.1. 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。ユーザーは、CIDR 形式の IP アドレス範囲またはドメイン名を選択するか、**nodeSelector** を使用して、送信トラフィックを許可または拒否できます。**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> 1
  to: 2
```

```
  cidrSelector: <cidr> 3
  dnsName: <dns_name> 4
```

1 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。

2 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。

3 CIDR 形式の IP アドレス範囲。

4 ドメイン名。

29.3.2.2. 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。

29.3.3. 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す。

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 ネットワーク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 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
```

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。

29.4. 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。

29.4.1. 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

- OpenShiftSDN ネットワーク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 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
```

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。

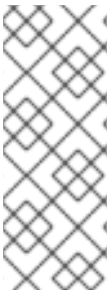
```
$ oc describe egressnetworkpolicy <policy_name>
```

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
```

29.5. 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。



注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

29.5.1. 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

- OpenShiftSDN ネットワーク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
```

29.6. 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。

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

29.6.1. 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

- OpenShiftSDN ネットワーク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>
```

29.7. EGRESS ルーター POD の使用についての考慮事項

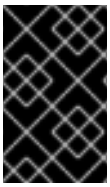
29.7.1. egress ルーター Pod について

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ー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。

29.7.1.1. 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。予約された送信元 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、スロールーターのサービスにアクセスするように設定する必要があります。**curl** コマンドを使用して、アプリケーション Pod から宛先サービスとポートにアクセスできます。以下に例を示します。

```
$ curl <router_service_IP> <port>
```

HTTP プロキシーモードでは、egress ルーター Pod はポート **8080** で HTTP プロキシー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 プロキシーを使用するように指示されます。

DNS プロキシーモードでは、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 プロキシー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 プロキシー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 プロキシーモードを使用します。

29.7.1.2. egress ルーター Pod の実装

egress ルーター Pod の設定は、初期化コンテナーで実行されます。このコンテナーは特権付きコンテキストで実行され、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。初期化コンテナーが **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 はコンテナーを実行して 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に設定します。

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に設定されます。

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。

29.7.1.3. 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>
```

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

29.7.1.4. 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。

29.7.2. 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 プロキシーモードでの egress ルーターのデプロイ
- DNS プロキシーモードでの egress ルーターのデプロイ

29.8. 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。

29.8.1. 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" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE 2
      value: <egress_router>
    - name: EGRESS_GATEWAY 3
      value: <egress_gateway>
    - name: EGRESS_DESTINATION 4
```

```
    value: <egress_destination>
  - name: EGRESS_ROUTER_MODE
    value: init
containers:
- name: egress-router-wait
  image: registry.redhat.io/openshift4/ose-pod
```

1. 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。

2. 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。

3. 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。

4. 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
```

29.8.2. 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
```

29.8.3. 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。予約された送信元 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、スルールーターのサービスにアクセスするように設定する必要があります。**curl** コマンドを使用して、アプリケーション Pod から宛先サービスとポートにアクセスできます。以下に例を示します。

```
$ curl <router_service_IP> <port>
```

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。

29.8.4. 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

29.9. HTTP プロキシーモードでの 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ー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。

29.9.1. 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" (1)
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE (2)
      value: <egress-router>
    - name: EGRESS_GATEWAY (3)
```

```
    value: <egress-gateway>
  - name: EGRESS_ROUTER_MODE
    value: http-proxy
containers:
- name: egress-router-pod
  image: registry.redhat.io/openshift4/ose-egress-http-proxy
  env:
  - name: EGRESS_HTTP_PROXY_DESTINATION 4
    value: |-
      ...
  ...
```

1 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。

2 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。

3 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。

4 プロキシー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 コンテナーの他の環境変数ではなく、HTTP プロキシーコンテナーの環境変数として指定されることに注意してください。

29.9.2. egress 宛先設定形式

egress ルーター Pod が HTTP プロキシー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ーを許可します。
- ***.** 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ーを許可します。
- 先の一致 (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
*
```

29.9.3. HTTP プロキシーモードでの egress ルーター Pod のデプロイ

HTTP プロキシーモードでは、egress ルーター Pod はポート **8080** で HTTP プロキシー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 プロキシー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 (1)
  type: ClusterIP
  selector:
    name: egress-1
```

(1) **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 プロキシー Pod ではない) を HTTP プロキシー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/ (1)
```

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

1 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。

29.9.4. 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

29.10. DNS プロキシーモードでの 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ー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。

29.10.1. 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" 1
spec:
  initContainers:
  - name: egress-router
    image: registry.redhat.io/openshift4/ose-egress-router
    securityContext:
      privileged: true
    env:
    - name: EGRESS_SOURCE 2
      value: <egress-router>
    - name: EGRESS_GATEWAY 3
      value: <egress-gateway>
    - name: EGRESS_ROUTER_MODE
      value: 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 4
    value: |-
      ...
  - name: EGRESS_DNS_PROXY_DEBUG 5
    value: "1"
  ...
```

1 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。

2 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。

3 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。

4 1 つ以上のプロキシー宛先のリストを指定します。

5 オプション: DNS プロキシーログ出力を **stdout** に出力するために指定します。

29.10.2. egress 宛先設定形式

ルーターが DNS プロキシー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リストを指定します。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、プロキシーは、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
```

29.10.3. DNS プロキシーモードでの egress ルーター Pod のデプロイ

DNS プロキシーモードでは、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 プロキシーとして機能します。

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リストに設定します。

```
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ー送信されます。

29.10.4. 関連情報

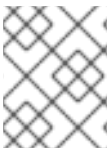
- ConfigMap を使用した egress ルーターの宛先マッピングの設定

29.11. 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。

29.11.1. 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、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ーを必要とするため、**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
...
```

29.11.2. 関連情報

- リダイレクトモード
- HTTP_PROXY
- DNS プロキシーモード

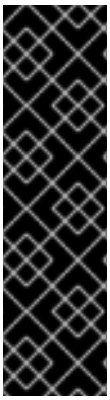
29.12. プロジェクトのマルチキャストの有効化

注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

29.12.1. 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。
- デフォルトでは、ネットワークポリシーは namespace 内のすべての接続に影響します。ただし、マルチキャストはネットワークポリシーの影響を受けません。マルチキャストがネットワークポリシーと同じ namespace で有効にされている場合、**deny-all** ネットワークポリシーがある場合でも、マルチキャストは常に許可されます。クラスター管理者は、これを有効にする前に、ネットワークポリシーからマルチキャストが除外されることの影響を考慮する必要があります。

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 ネットワークプラグインを使用している場合、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。

multinenant 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 に送信されます。
- 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。

29.12.2. 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/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat hostname && sleep inf"]
      ports:
        - containerPort: 30102
          name: mlistener
          protocol: UDP
EOF
```

3. マルチキャストセンダーとして機能する Pod を作成します。

```
$ cat <<EOF| oc create -f -
apiVersion: v1
kind: Pod
metadata:
  name: msender
```

```
  labels:
    app: multicast-verify
spec:
  containers:
    - name: msender
      image: registry.access.redhat.com/ubi9
      command: ["/bin/sh", "-c"]
      args:
        ["dnf -y install socat && sleep inf"]
EOF
```

4. 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
   ```

29.13. プロジェクトのマルチキャストの無効化

29.13.1. 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。

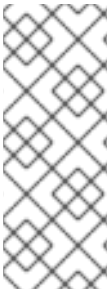
手順

- 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**。

29.14. OPENSHIFT SDN を使用したネットワーク分離の設定



注記

OpenShift SDN CNI は、OpenShift Container Platform 4.14 以降非推奨になりました。OpenShift Container Platform 4.15 以降の新規インストールでは、ネットワークプラグインというオプションはなくなりました。今後のリリースでは、OpenShift SDN ネットワークプラグインは削除され、サポートされなくなる予定です。Red Hat は、この機能が削除されるまでバグ修正とサポートを提供しますが、この機能は拡張されなくなります。OpenShift SDN CNI の代わりに、OVN Kubernetes CNI を使用できます。

クラスターが OpenShift SDN ネットワーク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。

29.14.1. 前提条件

- マルチテナント分離モードで OpenShift SDN ネットワークプラグインを使用するように設定されたクラスターが必要です。

29.14.2. 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ー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。

29.14.3. 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ーにログインする必要があります。

手順

- クラスター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。

29.14.4. 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。

29.15. KUBE-PROXY の設定

Kubernetes メットワークプロキシー (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。

29.15.1. iptables ルールの同期について

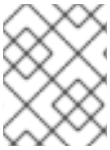
同期の期間は、Kubernetes ネットワークプロキシー (kube-proxy) がノードで iptables ルールを同期する頻度を定めます。

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。

29.15.2. 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29.2 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**
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**

29.15.3. kube-proxy 設定の変化

クラスターの Kubernetes ネットワークプロキシー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30章 ルートの作成

30.1. ルート設定

30.1.1. 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ー設定に応じて、セキュリティー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ー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
```

検証

- 作成した **route** リソースを確認するには、次のコマンドを実行します。

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

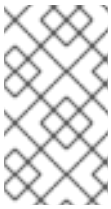
1 この例では、ルートの名前は **hello-openshift** です。

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> 1
  port:
    targetPort: 8080 2
  to:
    kind: Service
    name: hello-openshift
```

1 **<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。

2 **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}
```

30.1.2. Ingress Controller シャーディングのルート作成

ルートを使用すると、URL でアプリケーションをホストできます。この場合、ホスト名は設定されず、ルートは代わりにサブドメインを使用します。サブドメインを指定すると、ルートを公開する Ingress Controller のドメインが自動的に使用されます。ルートが複数の Ingress Controller によって公開されている状況では、ルートは複数の URL でホストされます。

以下の手順では、例として **hello-openshift** アプリケーションを使用して、Ingress Controller シャーディングのルートを作成する方法について説明します。

Ingress Controller のシャード化は、一連の Ingress Controller 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 Controller 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 Controller に指定し、Company B を別の Ingress Controller に指定できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- プロジェクト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 HTTP または TCP エンドポイントがある。
- シャーディング用に Ingress Controller を設定している。

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. **hello-openshift-route.yaml** というルート定義を作成します。

シャーディング用に作成されたルートの YAML 定義:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded (1)
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift (2)
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

(1) ラベルキーとそれに対応するラベル値の両方が、Ingress Controller で指定されたものと一致する必要があります。この例では、Ingress Controller にはラベルキーと値 **type: sharded** があります。

(2) ルートは、**subdomain** フィールドの値を使用して公開されます。**subdomain** フィールドを指定するときは、ホスト名を未設定のままにしておく必要があります。**host** フィールドと **subdomain** フィールドの両方を指定すると、ルートは **host** フィールドの値を使用し、**subdomain** フィールドを無視します。

5. 次のコマンドを実行し、**hello-openshift-route.yaml** を使用して **hello-openshift** アプリケーションへのルートを作成します。

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

検証

- 次のコマンドを使用して、ルートのステータスを取得します。

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

結果の **Route** リソースは次のようになります。

出力例

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1. Ingress Controller またはルーターがルートを公開するために使用するホスト名。**host** フィールドの値は、Ingress Controller によって自動的に決定され、そのドメインを使用します。この例では、Ingress Controller のドメインは **<apps-sharded.basedomain.example.net>** です。
2. Ingress Controller のホスト名。
3. Ingress Controller の名前。この例では、Ingress Controller の名前は **sharded** です。

30.1.3. 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 Controller 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> 1
```

1. 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
```

30.1.4. 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ークライアントに通知するセキュリティーの拡張機能です。また、HSTS は、HTTP リダイレクトを使用せずに HTTPS トランスポート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。

30.1.4.1. 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ー (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

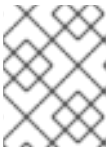
- プロジェクトの管理者権限があるユーザーで、クラスターにログインしている。
- **oc** CLI がインストールされている。

手順

- ルートで HSTS を有効にするには、**haproxy.router.openshift.io/hsts_header** 値を edge-termed 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 プリロードリスト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。ブラウザーはこれらのリストを使用し、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、ブラウザー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。

30.1.4.2. 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 がインストールされている。

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

1. 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}}{{""}}{{end}}{{end}}
{{end}}'
```

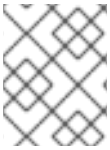
出力例

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

30.1.4.3. ドメインごとに HTTP Strict Transport Security の強制

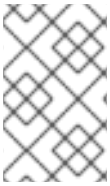
安全なルートのドメインごとに HTTPStrict Transport Security(HSTS) を適用するには、**requiredHSTSPolicies** レコードを Ingress 仕様に追加して、HSTS ポリシーの設定を取得します。

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。



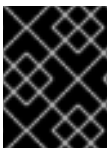
注記

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 がインストールされている。

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リスト表示できます。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 プリロードリストに含めることができます。ブラウザー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、ブラウザー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** で必要です。
- **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
```

30.1.5. スループットの問題のトラブルシューティング方法

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ノードで **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、これらの 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。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。最初に Pod からツールを実行し、次にノードから実行して、ボトルネックを特定します。
  - **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。
- 場合によっては、レイテンシーの問題が原因で、クラスターがルーター Pod を含むノードを異常としてマークすることがあります。ワーカーレイテンシープロファイルを使用して、アクションを実行する前にクラスターがノードからステータスの最新情報を受け取る頻度を調節します。

- クラスターでレイテンシーの低いノードとレイテンシーの高いノードが指定されている場合は、Ingress Controller の **spec.nodePlacement** フィールドを設定して、ルーター Pod の配置を制御します。

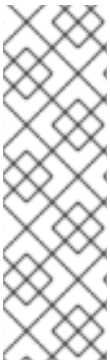
関連情報

- リモートワーカーへのレイテンシーの急上昇またはスループットの一時的な低下
- Ingress Controller 設定パラメーター

30.1.6. Cookie の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、スケーリング、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 Controller 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 Controller 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。

30.1.6.1. 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、以下のようになります。

<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、以下のようになります。

<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
```

30.1.7. 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。

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ーを示しています。

表30.1 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/text	Yes (ルートではなく、ホストで一致)
	www.example.com	はい

パスが 1 つでセキュリティー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" 1
  to:
    kind: Service
    name: service-name
```

1 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ことができないので、パ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。

30.1.8. HTTP ヘッダーの設定

OpenShift Container Platform は、HTTP ヘッダーを操作するためのさまざまな方法を提供します。ヘッダーを設定または削除する場合、Ingress Controller の特定のフィールドまたは個々のルートを使用して、リクエストヘッダーと応答ヘッダーを変更できます。ルートアノテーションを使用して特定のヘッダーを設定することもできます。ヘッダーを設定するさまざまな方法は、連携時に課題となる可能性があります。

注記

IngressController または **Route** CR 内のヘッダーは設定または削除のみ可能で、追加はできません。HTTP ヘッダーに値が設定されている場合、その値は完全である必要があるため、今後追加する必要はありません。X-Forwarded-For ヘッダーなどのヘッダーを追加することが適切な状況では、**spec.httpHeaders.actions** の代わりに **spec.httpHeaders.forwardedHeaderPolicy** フィールドを使用します。

30.1.8.1. 優先順位

同じ HTTP ヘッダーを Ingress Controller とルートの両方で変更すると、HAProxy は、それがリクエストヘッダーであるか応答ヘッダーであるかに応じて、特定の方法でアクションの優先順位を付けます。

- HTTP 応答ヘッダーの場合、Ingress Controller で指定されたアクションは、ルートで指定されたアクションの後に実行されます。これは、Ingress Controller で指定されたアクションが優先されることを意味します。
- HTTP リクエストヘッダーの場合、ルートで指定されたアクションは、Ingress Controller で指定されたアクションの後に実行されます。これは、ルートで指定されたアクションが優先されることを意味します。

たとえば、クラスター管理者は、次の設定を使用して、Ingress Controller で X-Frame-Options 応答ヘッダーに値 **DENY** を設定します。

IngressController 仕様の例

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

```
kind: IngressController
# ...
spec:
  httpHeaders:
    actions:
      response:
      - name: X-Frame-Options
        action:
          type: Set
          set:
            value: DENY
```

ルート所有者は、クラスター管理者が Ingress Controller に設定したものと同じ応答ヘッダーを設定しますが、次の設定を使用して値 **SAMEORIGIN** を設定します。

Route 仕様の例

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

IngressController 仕様と **Route** 仕様の両方で X-Frame-Options ヘッダーを設定している場合、特定のルートでフレームが許可されている場合でも、Ingress Controller のグローバルレベルでこのヘッダーに設定された値が優先されます。

この優先順位付けは、**haproxy.config** ファイルが次のロジックを使用するために発生します。このロジックでは、Ingress Controller がフロントエンドとみなされ、個々のルートがバックエンドとみなされます。フロントエンド設定に適用されるヘッダー値 **DENY** は、バックエンドで設定されている値 **SAMEORIGIN** で同じヘッダーをオーバーライドします。

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

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

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

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

さらに、Ingress Controller またはルートのいずれかで定義されたアクションは、ルートアノテーションを使用して設定された値をオーバーライドします。

30.1.8.2. 特殊なケースのヘッダー

次のヘッダーは、設定または削除が完全に禁止されているか、特定の状況下で許可されています。

表30.2 特殊な場合のヘッダー設定オプション

ヘッダー名	IngressController 仕様を使用して設定可能かどうか	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか
proxy	いいえ	いいえ	**プロキシー** HTTP リクエストヘッダーを使用して、ヘッダー値を **HTTP_PROXY** 環境変数に挿入して、脆弱な CGI アプリケーションを悪用できます。**プロキシー** HTTP リクエストヘッダーも標準ではないため、設定中にエラーが発生しやすくなります。	いいえ
host	いいえ	はい	**IngressController** CR を使用して **ホスト** HTTP 要求ヘッダーが設定されている場合、HAProxy は正しいルートを検索するときに失敗する可能性があります。	いいえ
strict-transport-security	いいえ	いいえ	**strict-transport-security** HTTP 応答ヘッダーはルートアノテーションを使用してすでに処理されているため、別の実装は必要ありません。	はい: **haproxy.router.openshift.io/hsts_header** ルートアノテーション

ヘッダー名	IngressController 仕様を使用して設定可能かどうか	Route 仕様を使用して設定可能かどうか	不許可の理由	別の方法で設定可能かどうか
cookie と **set-cookie**	いいえ	いいえ	HAProxy が設定する Cookie は、クライアント接続を特定のバックエンドサーバーにマップするセッション追跡に使用されます。これらのヘッダーの設定を許可すると、HAProxy のセッションアフィニティーが妨げられ、HAProxy の Cookie の所有権が制限される可能性があります。	はい: • **haproxy.router.openshift.io/disable_cookie** ルートアノテーション • **haproxy.router.openshift.io/cookie_name** ルートアノテーション

30.1.9. ルート内の HTTP リクエストおよびレスポンスヘッダーの設定または削除

コンプライアンス目的またはその他の理由で、特定の HTTP 要求および応答ヘッダーを設定または削除できます。これらのヘッダーは、Ingress Controller によって提供されるすべてのルート、または特定のルートに対して設定または削除できます。

たとえば、ルートを提供する Ingress Controller によってデフォルトのグローバルな場所が指定されている場合でも、コンテンツが複数の言語で記述されていると、Web アプリケーションが特定のルートの別の場所でコンテンツを提供するように指定できます。

以下の手順では Content-Location HTTP リクエストヘッダーを設定するルートを作成し、アプリケーション (**https://app.example.com**) に URL が関連付けられ、**https://app.example.com/lang/en-us** のロケーションにダイレクトされるようにします。アプリケーショントラフィックをこの場所にダイレクトすると、特定のルートを使用する場合はすべて、アメリカ英語で記載された Web コンテンツにアクセスすることになります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- プロジェクト管理者として OpenShift Container Platform クラスターにログインしている。
- 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 HTTP または TCP エンドポイントがある。

手順

1. ルート定義を作成し、**app-example-route.yaml** というファイルに保存します。

HTTP ヘッダーディレクティブを使用して作成されたルートの YAML 定義

```
apiVersion: route.openshift.io/v1
kind: Route
# ...
spec:
  host: app.example.com
  tls:
    termination: edge
  to:
    kind: Service
    name: app-example
  httpHeaders:
    actions: 1
      response: 2
      - name: Content-Location 3
        action:
          type: Set 4
          set:
            value: /lang/en-us 5
```

1. HTTP ヘッダーに対して実行するアクションのリスト。
2. 変更するヘッダーのタイプ。この場合は、応答ヘッダーです。
3. 変更するヘッダーの名前。設定または削除できる使用可能なヘッダーのリストについては、**HTTP ヘッダーの設定** を参照してください。
4. ヘッダーに対して実行されるアクションのタイプ。このフィールドには、**Set** または **Delete** の値を指定できます。
5. HTTP ヘッダーの設定時は、**値** を指定する必要があります。値は、そのヘッダーで使用可能なディレクティブのリストからの文字列 (例: **DENY)** にすることも、HAProxy の動的値構文を使用して解釈される動的値にすることもできます。この場合、値はコンテンツの相対位置に設定されます。

2. 新しく作成したルート定義を使用して、既存の Web アプリケーションへのルートを作成します。

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

HTTP リクエストヘッダーの場合、ルート定義で指定されたアクションは、Ingress Controller の HTTP リクエストヘッダーに対して実行されたアクションの後に実行されます。これは、ルート内のこれらのリクエストヘッダーに設定された値が、Ingress Controller に設定された値よりも優先されることを意味します。HTTP ヘッダーの処理順序の詳細は、**HTTP ヘッダーの設定** を参照してください。

30.1.10. ルート固有のアノテーション

Ingress Controller 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、リストが警告やエラーメッセージなしに無視されます。

表30.3 ルートアノテーション

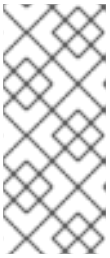
変数	説明	デフォルトで使用される環境変数
haproxy.router.openshift.io/balance	ロードバランシングアルゴリズムを設定します。使用できるオプションは、**random**、**source**、**roundrobin**、および **leastconn** です。デフォルト値は、TLS パススルールートの場合、**source** です。他のすべてのルートの場合、デフォルトは **random** です。	パススルールートの **ROUTER_TCP_BALANCE_SCHEME** です。それ以外の場合は **ROUTER_LOAD_BALANCE_ALGORITHM** を使用します。
haproxy.router.openshift.io/disable_cookies	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。	
router.openshift.io/cookie_name	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。	
haproxy.router.openshift.io/pod-concurrent-connections	ルーターからバッキングされる Pod に対して許容される接続最大数を設定します。 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。	

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	**'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。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
haproxy.router.openshift.io/rate-limit-connections.concurrent-tcp	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。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
haproxy.router.openshift.io/rate-limit-connections.rate-http	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。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
haproxy.router.openshift.io/rate-limit-connections.rate-tcp	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。 注記: このアノテーションを使用すると、サービス拒否攻撃に対する基本的な保護が提供されます。	
haproxy.router.openshift.io/timeout	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/timeout-tunnel	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。	**ROUTER_DEFAULT_TUNNEL_TIMEOUT**

変数	説明	デフォルトで使用される環境変数
ingresses.config/cluster ingress.operator.openshift.io /hard-stop-after	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 プロキシーが haproxy**hard-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、クリーンなソフトストップ実行で最大許容される時間を定義します。	**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**
haproxy.router.openshift.io/ip_whitelist	ルートの許可リストを設定します。許可リストは、承認したソースアドレスの IP アドレスおよび CIDR 範囲のリストをスペース区切りにしたリストです。許可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。 **haproxy.config** ファイルで直接表示される IP アドレスと CIDR 範囲の最大数は 61 です [1]。	
haproxy.router.openshift.io/hsts_header	edge terminated または re-encrypt ルートの Strick-Transport-Security ヘッダーを設定します。	
haproxy.router.openshift.io/rewrite-target	バックエンドの要求の書き換えパスを設定します。	

変数	説明	デフォルトで使用される環境変数
router.openshift.io/cookie-same-site	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。 **Lax**: ブラウザーは、クロスサイト要求では Cookie を送信しませんが、ユーザーが外部サイトから元のサイトに移動するときに Cookie を送信します。これは、**SameSite** 値が指定されていない場合のブラウザーのデフォルトの動作です。 **Strict**: ブラウザーは、同じサイトのリクエストに対してのみ Cookie を送信します。 **None**: ブラウザーは、クロスサイト要求と同一サイト要求の両方に対して Cookie を送信します。 この値は、re-encrypt および edge ルートにのみ適用されます。詳細は、SameSite cookie のドキュメント を参照してください。	
haproxy.router.openshift.io/set-forwarded-headers	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。値は以下のようになります。 **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。 **Replace**: ヘッダーを設定し、既存のヘッダーを削除します。 **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。 **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。	**ROUTER_SET_FORWARDED_HEADERS**

1. 許可リストの IP アドレスと CIDR 範囲の数が 61 を超えると、それらは別のファイルに書き込まれます。このファイルは **haproxy.config** から参照されます。このファイルは、**var/lib/haproxy/router/whitelists** フォルダーに保存されます。



注記

アドレスが許可リストに書き込まれることを確認するには、CIDR 範囲の完全なリストが Ingress Controller 設定ファイルに記載されていることを確認します。etcd オブジェクトサイズ制限は、ルートアノテーションのサイズを制限します。このため、許可リストに追加できる IP アドレスと CIDR 範囲の最大数のしきい値が作成されます。

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**)

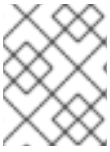
変数	デフォルト	説明
ROUTER_BACKEND_CHECK_INTERVAL	**5000ms**	バックエンドでの後続の liveness チェックの時間の長さ。
ROUTER_CLIENT_FIN_TIMEOUT	**1s**	クライアントがルートに接続する場合の TCP FIN タイムアウトの期間を制御します。接続切断のために送信された FIN が指定の時間内に応答されない場合は、HAProxy が接続を切断します。小さい値を設定し、ルーターでリソースをあまり使用していない場合には、リスクはありません。
ROUTER_DEFAULT_CLIENT_TIMEOUT	**30s**	ク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。
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バッキングする 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。

変数	デフォルト	説明
ROUTER_SLOWLORIS_HTTP_KEEPALIVE	**300s**	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、ブラウザー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。 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_KEEPALIVE** 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** を加えたことになります。
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 (1)
...
```

(1) 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

パススルールートのサーバー側のタイムアウト値を低く設定し過ぎると、WebSocket 接続がそのルートで頻繁にタイムアウトする可能性があります。

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: / (1)
...
```

(1) 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 Controller 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30.4 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

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/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

30.1.11. 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
```

30.1.12. 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、**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
    route.openshift.io/destination-ca-certificate-secret: secret-ca-cert 2
spec:
  rules:
  - host: www.example.com 3
    http:
      paths:
      - backend:
          service:
            name: frontend
            port:
              number: 443
        path: /
        pathType: Prefix
  tls:
```

```
- hosts:
  - www.example.com
  secretName: example-com-tls-certificate
```

1 **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。

3 **Ingress** オブジェクトを操作する場合、ルートを操作する場合とは異なり、明示的なホスト名を指定する必要があります。**<host_name>.<cluster_ingress_domain>** 構文 (**apps.openshiftdemos.com** など) を使用して、***.<cluster_ingress_domain>** ワイルドカード DNS レコードとクラスターのサービング証明書を利用できます。それ以外の場合は、選択したホスト名の DNS レコードがあることを確認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 **route.openshift.io/destination-ca-certificate-secret** を Ingress オブジェクトで使用して、カスタム宛先証明書 (CA) でルートを定義できます。アノテーションは、生成されたルートに挿入される kubernetes シークレット **secret-ca-cert** を参照します。

a. Ingress オブジェクトから宛先 CA を使用してルートオブジェクトを指定するには、シークレットの **data.tls.crt** 指定子に PEM エンコード形式の証明書を使用して **kubernetes.io/tls** または **Opaque** タイプのシークレットを作成する必要があります。

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
    destinationCACertificate: |
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
  to:
    kind: Service
    name: frontend
```

30.1.13. 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:
  - {} 1
```

1 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 1
    ...
  spec:
  ...
    tls: 2
      insecureEdgeTerminationPolicy: Redirect
      termination: edge 3
  ...
```

1 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。

2 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。

3 ルートは、**edge** の終了ポリシーを指定する必要があります。

30.1.14. Ingress アノテーションでの宛先 CA 証明書を使用したルート作成

route.openshift.io/destination-ca-certificate-secret アノテーションを Ingress オブジェクトで使用して、カスタム宛先 CA 証明書でルートを定義できます。

前提条件

- PEM エンコードされたファイルで証明書/キーのペアを持つことができます。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。
- 公開する必要のあるサービスが必要です。

手順

1. **route.openshift.io/destination-ca-certificate-secret** を Ingress アノテーションに追加します。

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

(1) アノテーションは kubernetes シークレットを参照します。

2. このアノテーションで参照されているシークレットは、生成されたルートに挿入されます。

出力例

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

30.1.15. デュアルスタックネットワーク用の OpenShift Container Platform Ingress Controller 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 Controller 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 Controller を設定できます。

前提条件

- ベアメタルに OpenShift Container Platform クラスター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 Controller 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 Controller 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

30.2. 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 が制限する語のリストは、Azure ドキュメントの Resolve reserved resource name errors を参照してください。

30.2.1. 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。

- 公開する必要の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フレーズを削除するには、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 Controller 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、公開する必要のある **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。

30.2.2. 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 Controller 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 Controller 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。
- 公開する必要の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フレーズを削除するには、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、公開する必要の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。

30.2.3. 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

- 公開する必要の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。
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31章 INGRESS クラスタートラフィックの設定

31.1. 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用し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 Controller 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 Controller 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 Controller 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 アドレスを使用し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 アドレスを使用し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。

31.1.1. 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。

31.2. 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。

31.2.1. 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。

31.2.2. 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。

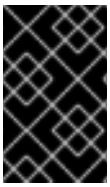
ネットワークインフラストラクチャーを設定し、定義する外部 IP アドレスブロックがクラスターにルーティングされるようにする必要があります。そのため、IP アドレスがノードのネットワークインターフェイスに設定されません。トラフィックを処理するには、静的な Address Resolution Protocol (ARP) エントリーなどの方法を使用して、ルーティングと外部 IP へのアクセスを設定する必要があります。

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、セキュリティー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。

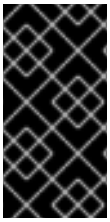
31.2.2.1. ExternalIP の設定

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もあります。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: highport
    nodePort: 31903
    port: 30102
    protocol: TCP
    targetPort: 30102
  selector:
    app: web
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer:
    ingress:
    - ip: 192.168.132.253
```

31.2.2.2. 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。

31.2.2.3. 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
  ...
```

31.2.3. 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

クラスター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

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 アドレス範囲のリスト。

2 CIDR 形式の拒否される IP アドレス範囲のリスト。

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.externalIP.autoAssignCIDRs が設定された設定例

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    ...
    externalIP:
      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.externalIP.policy が設定された設定例

  ```
  apiVersion: config.openshift.io/v1
  kind: Network
  metadata:
    name: cluster
  spec:
    ...
    externalIP:
      policy:
        allowedCIDRs:
        - 192.168.132.0/29
        - 192.168.132.8/29
        rejectedCIDRs:
        - 192.168.132.7/32
  ```

31.2.4. 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 ExternalIP 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 ExternalIP 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 ExternalIP 設定を変更します。

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  ...
  externalIP: 1
  ...
```

1 **externalIP** スタンザの設定を指定します。

4. 更新された ExternalIP 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.externalIP}}{{"\n"}}'
```

31.2.5. 次のステップ

- サービスの外部 IP を使用した ingress クラストラフィックの設定

31.3. INGRESS CONTROLLER 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用し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 Controller を使用します。

31.3.1. Ingress Controller およびルートの使用

Ingress Operator は Ingress Controller およびワイルドカード DNS を管理します。

Ingress Controller の使用は、最も一般的な、OpenShift Container Platform クラスターへの外部アクセスを許可する方法です。

Ingress Controller 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ー送信するよう設定されます。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。

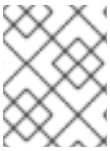
管理者と連携して Ingress Controller を設定します。外部要求を許可し、設定されたルートに基づいてそれらをプロキシー送信するように Ingress Controller を設定します。

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 Controller 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 Controller 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 Controller 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 Controller:

- 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。

31.3.2. 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。

31.3.3. 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。

31.3.4. 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 などのツールを使用して、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
```

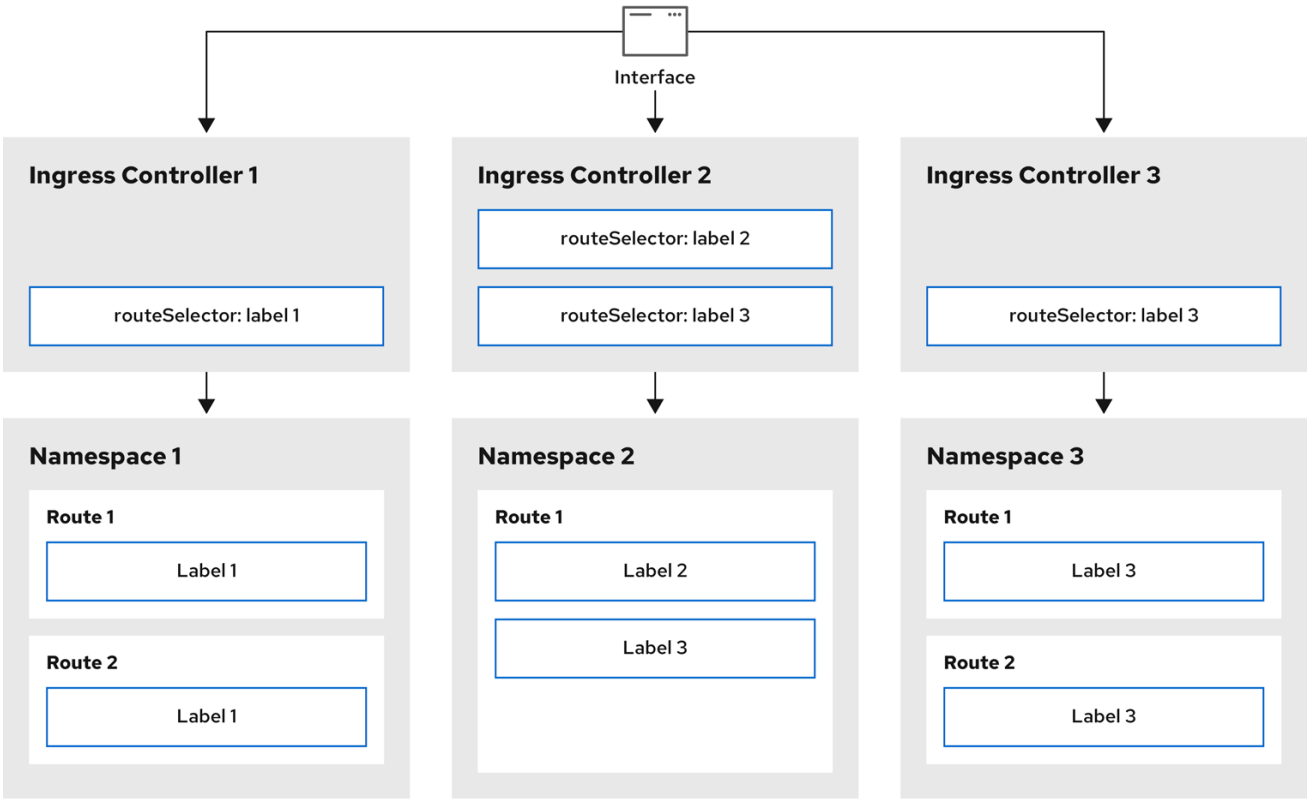
出力例

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

31.3.5. ルートラベルを使用した Ingress Controller のシャード化の設定

ルートラベルを使用した Ingress Controller のシャード化とは、Ingress Controller 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。

図31.1 ルートラベルを使用した Ingress シャーディング



Ingress Controller のシャード化は、一連の Ingress Controller 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 Controller 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 Controller に指定し、Company B を別の Ingress Controller 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: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  routeSelector:
    matchLabels:
      type: sharded
```

1 Ingress Controller が使用するドメインを指定します。このドメインは、デフォルトの Ingress Controller ドメインとは異なる必要があります。

2. Ingress Controller の **router-internal.yaml** ファイルを適用します。

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

Ingress Controller は、**type: sharded** というラベルのある namespace のルートを選択します。

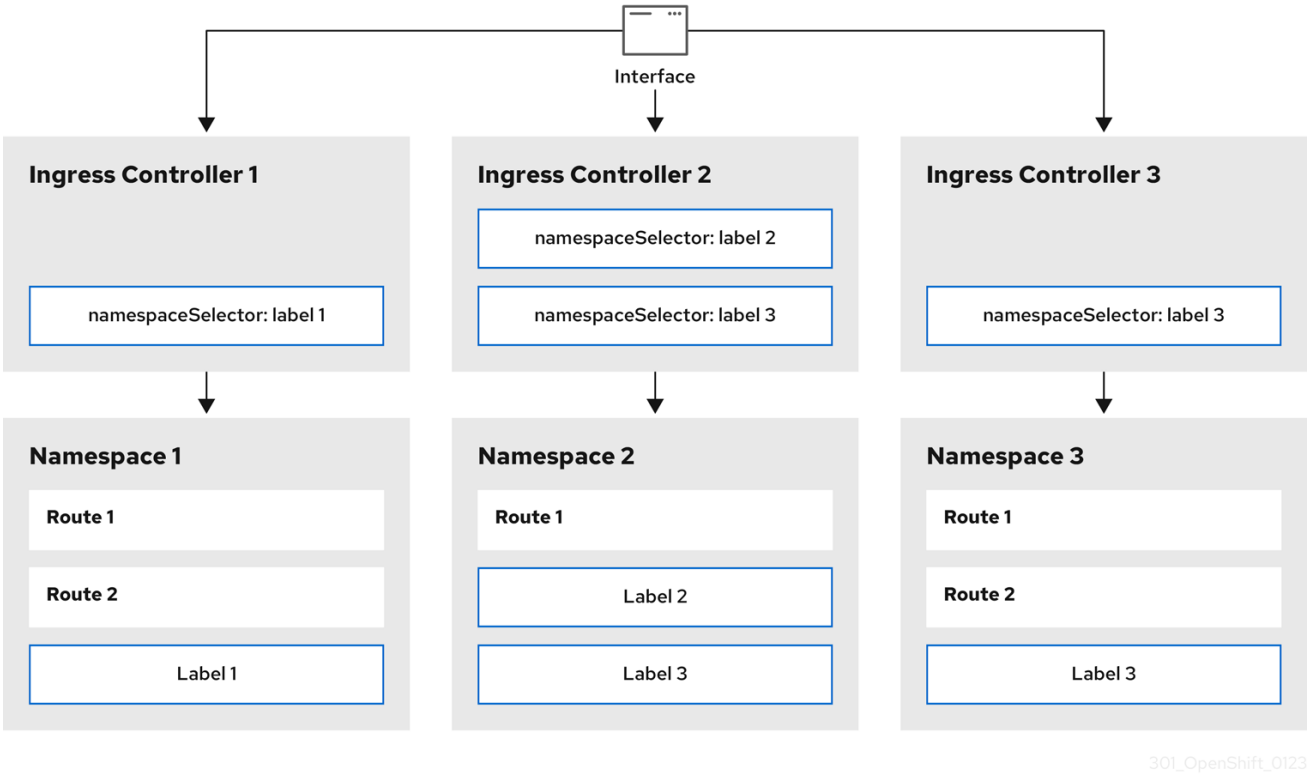
3. **router-internal.yaml** で設定されたドメインを使用して新しいルートを作成します。

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

31.3.6. namespace ラベルを使用した Ingress Controller のシャード化の設定

namespace ラベルを使用した Ingress Controller のシャード化とは、Ingress Controller 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。

図31.2 namespace ラベルを使用した Ingress シャーディング



Ingress Controller のシャード化は、一連の Ingress Controller 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 Controller 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 Controller に指定し、Company B を別の Ingress Controller 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
```

出力例

```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: sharded
  namespace: openshift-ingress-operator
spec:
  domain: <apps-sharded.basedomain.example.net> 1
  nodePlacement:
    nodeSelector:
      matchLabels:
        node-role.kubernetes.io/worker: ""
  namespaceSelector:
    matchLabels:
      type: sharded
```

1 Ingress Controller が使用するドメインを指定します。このドメインは、デフォルトの Ingress Controller ドメインとは異なる必要があります。

2. Ingress Controller の **router-internal.yaml** ファイルを適用します。

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

Ingress Controller 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。

3. **router-internal.yaml** で設定されたドメインを使用して新しいルートを作成します。

```
$ oc expose svc <service-name> --hostname <route-name>.apps-
sharded.basedomain.example.net
```

31.3.7. Ingress Controller シャーディングのルート作成

ルートを使用すると、URL でアプリケーションをホストできます。この場合、ホスト名は設定されず、ルートは代わりにサブドメインを使用します。サブドメインを指定すると、ルートを公開する Ingress Controller のドメインが自動的に使用されます。ルートが複数の Ingress Controller によって公開されている状況では、ルートは複数の URL でホストされます。

以下の手順では、例として **hello-openshift** アプリケーションを使用して、Ingress Controller シャーディングのルートを作成する方法について説明します。

Ingress Controller のシャード化は、一連の Ingress Controller 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 Controller 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 Controller に指定し、Company B を別の Ingress Controller に指定できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- プロジェクト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 HTTP または TCP エンドポイントがある。
- シャーディング用に Ingress Controller を設定している。

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. **hello-openshift-route.yaml** というルート定義を作成します。

シャーディング用に作成されたルートの YAML 定義:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  labels:
    type: sharded (1)
  name: hello-openshift-edge
  namespace: hello-openshift
spec:
  subdomain: hello-openshift (2)
  tls:
    termination: edge
  to:
    kind: Service
    name: hello-openshift
```

(1) ラベルキーとそれに対応するラベル値の両方が、Ingress Controller で指定されたものと一致する必要があります。この例では、Ingress Controller にはラベルキーと値 **type: sharded** があります。

(2) ルートは、**subdomain** フィールドの値を使用して公開されます。**subdomain** フィールドを指定するときは、ホスト名を未設定のままにしておく必要があります。**host** フィールドと **subdomain** フィールドの両方を指定すると、ルートは **host** フィールドの値を使用し、**subdomain** フィールドを無視します。

5. 次のコマンドを実行し、**hello-openshift-route.yaml** を使用して **hello-openshift** アプリケーションへのルートを作成します。

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

検証

- 次のコマンドを使用して、ルートのステータスを取得します。

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

結果の **Route** リソースは次のようになります。

出力例

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

```
    kind: Service
    name: hello-openshift
status:
  ingress:
  - host: hello-openshift.<apps-sharded.basedomain.example.net> 1
    routerCanonicalHostname: router-sharded.<apps-sharded.basedomain.example.net> 2
    routerName: sharded 3
```

1 Ingress Controller またはルーターがルートを公開するために使用するホスト名。**host** フィールドの値は、Ingress Controller によって自動的に決定され、そのドメインを使用します。この例では、Ingress Controller のドメインは **<apps-sharded.basedomain.example.net>** です。

2 Ingress Controller のホスト名。

3 Ingress Controller の名前。この例では、Ingress Controller の名前は **sharded** です。

31.3.8. 関連情報

Ingress Operator はワイルドカード DNS を管理します。詳細は、以下を参照してください。

- OpenShift Container Platform の Ingress Operator
- ベアメタルにクラスターをインストールする
- vSphere にクラスターをインストールする
- ネットワークポリシーについて

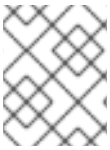
31.4. ロードバランサーを使用した INGRESS クラスターの設定

OpenShift Container Platform は、クラスター内で実行されるサービスを使用し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。

31.4.1. 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。

31.4.2. 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。

31.4.3. 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。

31.4.4. 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 などのツールを使用して、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
...
```

31.4.5. 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。
- クラウドプロバイダーがロードバランサーをサポートしている。

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。

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を読み込みます。

```
$ oc project project1
```

3. 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 アドレスのリストを入力して、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通過するトラフィックを特定の IP アドレスに制限するには、Ingress Controller フィールド **spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges** を使用することを推奨します。**loadBalancerSourceRanges** フィールドを設定しないでください。

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になります。

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)]>
```

31.5. 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用し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、AWS のロードバランサー、具体的には Network Load Balancer (NLB) またはク Classic Load Balancer (CLB) を使用します。どちらのタイプのロードバランサーもクライアントの IP アドレスをノードに転送できますが、CLB にはプロキシープロトコルのサポートが必要です。これは OpenShift Container Platform によって自動的に有効になります。

NLB を使用するように Ingress Controller を設定するには、次の 2 つの方法があります。

1. 現在 CLB を使用している Ingress Controller を強制的に置き換える。これにより、**IngressController** オブジェクトが削除され、新しい DNS レコードが伝達され、NLB がプロビジョニングされている間、停止が発生します。

2. CLB を使用する既存の Ingress Controller を編集して、NLB を使用する。これにより、**IngressController** オブジェクトを削除して再作成することなく、ロードバランサーが変更されます。

どちらの方法も、NLB から CLB への切り替えに使用できます。

これらのロードバランサーは、新規または既存の AWS クラスターで設定できます。

31.5.1. AWS での Classic Load Balancer タイムアウトの設定

OpenShift Container Platform は、特定のルートまたは Ingress Controller のカスタムタイムアウト期間を設定するためのメソッドを提供します。さらに、AWS Classic Load Balancer (CLB) には独自のタイムアウト期間があり、デフォルトは 60 秒です。

CLB のタイムアウト期間がルートタイムアウトまたは Ingress Controller タイムアウトよりも短い場合、ロードバランサーは接続を途中で終了する可能性があります。ルートと CLB の両方のタイムアウト期間を増やすことで、この問題を防ぐことができます。

31.5.1.1. 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 Controller 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> 1
```

1 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
```

31.5.1.2. Classic Load Balancer タイムアウトの設定

Classic Load Balancer (CLB) のデフォルトのタイムアウトを設定して、アイドル接続を延長できます。

前提条件

- 実行中のクラスターにデプロイ済みの Ingress Controller がある。

手順

1. 次のコマンドを実行して、デフォルト **ingresscontroller** の AWS 接続アイドルタイムアウトを 5 分に設定します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"type":"LoadBalancerService", "loadBalancer": \
    {"scope":"External", "providerParameters":{"type":"AWS", "aws": \
    {"type":"Classic", "classicLoadBalancer": \
    {"connectionIdleTimeout":"5m"}}}}}}'
```

2. オプション: 次のコマンドを実行して、タイムアウトのデフォルト値を復元します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"providerParameters":{"aws":{"classicLoadBalancer": \
    {"connectionIdleTimeout":null}}}}}}'
```

注記

現在のスコープがすでに設定されている場合を除き、接続タイムアウト値を変更するには **scope** フィールドを指定する必要があります。デフォルトのタイムアウト値に戻す場合は、**scope** フィールドを設定する際に再度設定する必要はありません。

31.5.2. 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用し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。そのような方法の 1 つでは、Network Load Balancer (NLB) を使用します。NLB を新規または既存の AWS クラスターに設定することができます。

31.5.2.1. Ingress Controller を Classic Load Balancer から Network Load Balancer に切り替える

Classic Load Balancer (CLB) を使用している Ingress Controller を、AWS の Network Load Balancer (NLB) を使用する Ingress Controller に切り替えることができます。

これらのロードバランサーを切り替えても、**IngressController** オブジェクトは削除されません。

警告

この手順により、次の問題が発生する可能性があります。

- 新しい DNS レコードの伝播、新しいロードバランサーのプロビジョニング、およびその他の要因により、数分間続く可能性のある停止。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。
- サービスのアノテーションの変更により、ロードバランサーリソースがリークする。

手順

1. NLB を使用して切り替える既存の Ingress Controller を変更します。この例では、デフォルトの Ingress Controller に **External** スコープがあり、他のカスタマイズがないことを前提としています。

   ingresscontroller.yaml 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
```

注記

spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type フィールドの値を指定しない場合、Ingress Controller は、インストール時に設定されたクラスター **Ingress** 設定の **spec.loadBalancer.platform.aws.type** 値を使用します。

ヒント

Ingress Controller に、ドメインの変更など、更新したい他のカスタマイズがある場合は、代わりに Ingress Controller 定義ファイルを強制的に置き換えることを検討してください。

2. 次のコマンドを実行して、Ingress Controller YAML ファイルに変更を適用します。

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

Ingress Controller の更新中は、数分間の停止が予想されます。

31.5.2.2. Network Load Balancer の使用から Classic Load Balancer への Ingress Controller の切り替え

Network Load Balancer (NLB) を使用している Ingress Controller を、AWS の Classic Load Balancer (CLB) を使用する Ingress Controller に切り替えることができます。

これらのロードバランサーを切り替えても、**IngressController** オブジェクトは削除されません。

警告

この手順により、新しい DNS レコードの伝播、新しいロードバランサーのプロビジョニング、およびその他の要因により、数分間続く停止が発生する可能性があります。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. CLB を使用して切り替える既存の Ingress Controller を変更します。この例では、デフォルトの Ingress Controller に **External** スコープがあり、他のカスタマイズがないことを前提としています。

   ingresscontroller.yaml 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: Classic
    type: LoadBalancerService
```

注記

spec.endpointPublishingStrategy.loadBalancer.providerParameters.aws.type フィールドの値を指定しない場合、Ingress Controller は、インストール時に設定されたクラスター **Ingress** 設定の **spec.loadBalancer.platform.aws.type** 値を使用します。

ヒント

Ingress Controller に、ドメインの変更など、更新したい他のカスタマイズがある場合は、代わりに Ingress Controller 定義ファイルを強制的に置き換えることを検討してください。

2. 次のコマンドを実行して、Ingress Controller YAML ファイルに変更を適用します。

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

Ingress Controller の更新中は、数分間の停止が予想されます。

31.5.2.3. 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

この手順により、次の問題が発生する可能性があります。

- 新しい DNS レコードの伝播、新しいロードバランサーのプロビジョニング、およびその他の要因により、数分間続く可能性のある停止。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。
- サービスのアノテーションの変更により、ロードバランサーリソースがリークする。

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。

ヒント

Ingress Controller に他のカスタマイズがなく、ロードバランサータイプのみを更新する場合は、Switching the Ingress Controller from using a Classic Load Balancer to a Network Load Balancer に記載の手順に従ってください。

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。数分間の停止が予想されます。

31.5.2.4. 既存 AWS クラスターでの Ingress Controller 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 Controller を既存のクラスターに作成できます。

前提条件

- AWS クラスター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

既存のクラスターの AWS NLB がサポートする Ingress Controller を作成します。

1. Ingress Controller のマニフェストを作成します。

```
$ cat ingresscontroller-aws-nlb.yaml
```

出力例

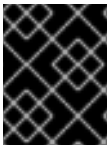
```
apiVersion: operator.openshift.io/v1
kind: IngressController
metadata:
  name: $my_ingress_controller 1
  namespace: openshift-ingress-operator
spec:
  domain: $my_unique_ingress_domain 2
  endpointPublishingStrategy:
    type: LoadBalancerService
```

```
  loadBalancer:
    scope: External 3
    providerParameters:
      type: AWS
      aws:
        type: NLB
```

1. **$my_ingress_controller** を Ingress Controller の一意の名前に置き換えます。
2. **$my_unique_ingress_domain** を、クラスター内のすべての Ingress Controller 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。
3. **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 Controller 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。

31.5.2.5. 新規 AWS クラスターでの Ingress Controller 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 Controller 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 Controller 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> 1
```

1. **<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。

31.5.3. 関連情報

- ネットワークをカスタマイズして AWS にクラスターをインストールする
- NLB のサポートの詳細は、Network Load Balancer support on AWS を参照してください。
- CLB のプロキシープロトコルサポートの詳細は、Configure proxy protocol support for your Classic Load Balancer を参照してください。

31.6. 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。

31.6.1. 前提条件

- クラスターは 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 を参照してください。

注記

egress IP に同じ ExternalIP を使用しないでください。

31.6.2. 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。クラスターが 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}{"\n"}'
```

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
```

31.6.3. 関連情報

- サービスの ExternalIP の設定

31.7. NODEPORT を使用した INGRESS クラス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用し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。

31.7.1. 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。

31.7.2. 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。

31.7.3. 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。

31.7.4. 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、次のコマンドを入力して、サービスのカスタムリソース定義 (CRD) を変更します。

```
$ oc edit svc <service_name>
```

出力例

```
spec:
  ports:
  - name: 8443-tcp
    nodePort: 30327 1
    port: 8443
    protocol: TCP
    targetPort: 8443
  sessionAffinity: None
  type: NodePort 2
```

1. オプション: アプリケーションのノードポート範囲を指定します。デフォルトでは、OpenShift Container Platform は **30000-32767** 範囲で使用可能なポートを選択します。
2. サービスタイプを定義します。

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
```

検証

- サービスノードポートが **30000** ~ **32767** の範囲のポートで更新されていることを確認するには、次のコマンドを入力します。

```
$ oc get svc
```

次の出力例では、更新されたポートは **30327** です。

出力例

```
NAME    TYPE      CLUSTER-IP      EXTERNAL-IP   PORT(S)         AGE
httpd   NodePort  172.xx.xx.xx    <none>        8443:30327/TCP  109s
```

31.7.5. 関連情報

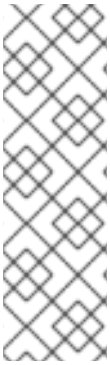
- ノードポートサービス範囲の設定

31.8. ロードバランサーの許可された送信元範囲を使用した INGRESS クラスタートラフィックの設定

IngressController の IP アドレス範囲の一覧を指定できます。**endpointPublishingStrategy** が **LoadBalancerService** の場合、これにより、ロードバランサーサービスへのアクセスが制限されます。

31.8.1. ロードバランサーの許可されるソース範囲の設定

spec.endpointPublishingStrategy.loadBalancer.allowedSourceRanges フィールドを有効にして設定できます。ロードバランサーの許可されるソース範囲を設定することで、Ingress Contoller のロードバランサーへのアクセスを、指定された IP アドレス範囲のリストに制限できます。Ingress Operator はロードバランサー Service を調整し、**AllowedSourceRanges** に基づいて **spec.loadBalancerSourceRanges** フィールドを設定します。



注記

以前のバージョンの OpenShift Container Platform で **spec.loadBalancerSourceRanges** フィールドまたはロードバランサーサービスアノテーション **service.beta.kubernetes.io/load-balancer-source-ranges** をすでに設定している場合、Ingress Controller はアップグレード後に **Progressing=True** のレポートを開始します。.これを修正するには、**spec.loadBalancerSourceRanges** フィールドを上書きし、**service.beta.kubernetes.io/load-balancer-source-ranges** アノテーションをクリアする **AllowedSourceRanges** を設定します。Ingress Controller は、再び **Progressing=False** の報告を開始します。

前提条件

- 実行中のクラスターにデプロイされた Ingress Controller があります。

手順

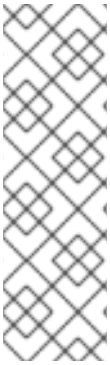
- 次のコマンドを実行して、Ingress Controller の許可されるソース範囲 API を設定します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}}' 1
```

1 例の値 **0.0.0.0/0** は、許可されるソース範囲を指定します。

31.8.2. ロードバランサーの許可されたソース範囲への移行

注釈 **service.beta.kubernetes.io/load-balancer-source-ranges** をすでに設定している場合は、ロードバランサーの許可されたソース範囲に移行できます。**AllowedSourceRanges** を設定すると、Ingress Controller は **AllowedSourceRanges** 値に基づいて **spec.loadBalancerSourceRanges** フィールドを設定し、**service.beta.kubernetes.io/load-balancer-source-ranges** アノテーションを設定解除します。



注記

以前のバージョンの OpenShift Container Platform で **spec.loadBalancerSourceRanges** フィールドまたはロードバランサーサービスアノテーション **service.beta.kubernetes.io/load-balancer-source-ranges** をすでに設定している場合、Ingress Controller はアップグレード後に **Progressing=True** のレポートを開始します。.これを修正するには、**spec.loadBalancerSourceRanges** フィールドを上書きし、**service.beta.kubernetes.io/load-balancer-source-ranges** アノテーションをクリアする **AllowedSourceRanges** を設定します。Ingress Controller は再び **Progressing=False** の報告を開始します。

前提条件

- **service.beta.kubernetes.io/load-balancer-source-ranges** アノテーションを設定しました。

手順

1. **service.beta.kubernetes.io/load-balancer-source-ranges** が設定されていることを確認します。

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

出力例

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/load-balancer-source-ranges: 192.168.0.1/32
```

2. **spec.loadBalancerSourceRanges** フィールドが設定されていないことを確認します。

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

出力例

```
...
spec:
  loadBalancerSourceRanges:
  - 0.0.0.0/0
...
```

3. クラスターを OpenShift Container Platform 4.15 に更新します。

4. 次のコマンドを実行して、**ingresscontroller** の許可されるソース範囲 API を設定します。

```
$ oc -n openshift-ingress-operator patch ingresscontroller/default \
    --type=merge --patch='{"spec":{"endpointPublishingStrategy": \
    {"loadBalancer":{"allowedSourceRanges":["0.0.0.0/0"]}}}}' 1
```

1 例の値 **0.0.0.0/0** は、許可されるソース範囲を指定します。

31.8.3. 関連情報

- OpenShift の更新の概要

第32章 KUBERNETES NMSTATE

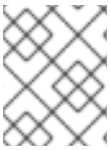
32.1. KUBERNETES NMSTATE OPERATOR について

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

Red Hat は、ベアメタル、IBM Power®、IBM Z®、IBM® LinuxONE、VMware vSphere、および OpenStack インストール上の実稼働環境で Kubernetes NMState Operator をサポートします。

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。



注記

Kubernetes NMState Operator は、セカンダリー NIC のネットワーク設定を更新します。プライマリー NIC または **br-ex** ブリッジのネットワーク設定を更新できません。

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、たとえばすべてのノードに Linux ブリッジを作成するなど、単一の設定マニフェストをクラスターに適用して、ネットワークポリシー設定を変更できるようになります。

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。

32.1.1. Kubernetes NMState Operator のインストール

ウェブコンソールまたは CLI を使用して、Kubernetes NMState Operator をインストールできます。

32.1.1.1. Web コンソールを使用した Kubernetes NMState Operator のインストール

Web コンソールを使用して Kubernetes NMState Operator をインストールできます。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。

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

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。

32.1.1.2. CLI を使用した Kubernetes NMState Operator のインストール

OpenShift CLI (**oc)** を使用して、Kubernetes NMState Operator をインストールできます。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。

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. **nmstateOperator** namespace を作成します。

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
```

```
  labels:
    kubernetes.io/metadata.name: openshift-nmstate
    name: openshift-nmstate
  name: openshift-nmstate
spec:
  finalizers:
  - kubernetes
EOF
```

2. **OperatorGroup** を作成します。

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

3. **nmstate** Operator にサブスクライブします。

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

4. **nmstate** Operator デプロイメントの **ClusterServiceVersion** (CSV) ステータスが **Succeeded** であることを確認します。

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

出力例

```
Name                                         Phase
kubernetes-nmstate-operator.4.15.0-202210210157   Succeeded
```

5. **nmstate** Operator のインスタンスを作成します。

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

6. NMState Operator の Pod が稼働していることを確認します。

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

出力例

```
Name                                  Ready   Status  Restarts  Age
pod/nmstate-cert-manager-5b47d8dddf-5wnb5   1/1   Running  0        77s
pod/nmstate-console-plugin-d6b76c6b9-4dcwm  1/1   Running  0        77s
pod/nmstate-handler-6v7rm                   1/1   Running  0        77s
pod/nmstate-handler-bjcxw                   1/1   Running  0        77s
pod/nmstate-handler-fv6m2                   1/1   Running  0        77s
pod/nmstate-handler-kb8j6                   1/1   Running  0        77s
pod/nmstate-handler-wn55p                   1/1   Running  0        77s
pod/nmstate-operator-f6bb869b6-v5m92        1/1   Running  0        4m51s
pod/nmstate-webhook-66d6bbd84b-6n674        1/1   Running  0        77s
pod/nmstate-webhook-66d6bbd84b-vlzrd        1/1   Running  0        77s
```

32.2. ノードのネットワーク状態と設定の監視と更新

32.2.1. CLI を使用したノードのネットワーク状態の表示

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。**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** オブジェクトをリスト表示します。

```
$ 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/v1
kind: NodeNetworkState
metadata:
  name: node01 (1)
status:
  currentState: (2)
```

```
    dns-resolver:
# ...
    interfaces:
# ...
    route-rules:
# ...
    routes:
# ...
  lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" 3
```

1. **NodeNetworkState** 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。

32.2.2. Web コンソールからのノードのネットワーク状態の表示

管理者は、OpenShift Container Platform Web コンソールを使用して、**NodeNetworkState** リソースとネットワークインターフェイスを観察し、ネットワークの詳細にアクセスできます。

手順

1. Networking → NodeNetworkState に移動します。
   NodeNetworkState ページでは、**NodeNetworkState** リソースと、ノード上に作成された対応するインターフェイスのリストを表示できます。Interface state、Interface type、および IP に基づく **フィルター**、または Name または Label の条件に基づく検索バーを使用して、表示される **NodeNetworkState** リソースを絞り込むことができます。
2. **NodeNetworkState** リソースに関する詳細情報にアクセスするには、Name 列にリストされている **NodeNetworkState** リソース名をクリックします。
3. **NodeNetworkState** リソースの Network Details セクションをデプロイメントして表示するには、> アイコンをクリックします。あるいは、Network interface 列の下の各インターフェイスタイプをクリックして、ネットワークの詳細を表示することもできます。

32.2.3. Web コンソールからのポリシーの管理

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。Web コンソールからポリシーを管理するには、Networking メニューの NodeNetworkConfigurationPolicy ページで作成されたポリシーのリストにアクセスします。このページでは、ポリシーを作成、更新、監視、削除できます。

32.2.3.1. ポリシーステータスの監視

NodeNetworkConfigurationPolicy ページからポリシーのステータスを監視できます。このページには、クラスター内に作成されたすべてのポリシーが次の列を含む表形式で表示されます。

名前

作成されたポリシーの名前。

一致したノード

ポリシーが適用されるノードの数。これは、ノードセレクターに基づくノードのサブセット、またはクラスター上のすべてのノードのいずれかになります。

ノードのネットワーク状態

一致したノードの実行状態。制定状態をクリックすると、ステータスの詳細情報が表示されます。

目的のポリシーを見つけるには、**Filter** オプションを使用するか、検索オプションを使用して、制定状態に基づいてリストをフィルターできます。

32.2.3.2. ポリシーの作成

Web コンソールでフォームまたは YAML を使用してポリシーを作成できます。

手順

1. **Networking → NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで **Create** をクリックし、**From Form**オプションを選択します。
   既存のポリシーがない場合は、**Create NodeNetworkConfigurationPolicy** をクリックして、フォームを使用してポリシーを作成することもできます。

注記

YAML を使用してポリシーを作成するには、**Create** をクリックし、**With YAML** オプションを選択します。次の手順は、フォームを使用してポリシーを作成する場合にのみ適用されます。

3. オプション: **Apply this NodeNetworkConfigurationPolicy only to specific subsets of nodes using the node selector** チェックボックスをオンにして、ポリシーを適用する必要があるノードを指定します。
4. **Policy name** フィールドにポリシー名を入力します。
5. オプション: **Description** フィールドにポリシーの説明を入力します。
6. オプション: **Policy Interface(s)** セクションでは、編集可能なフィールドにプリセット値が設定されたブリッジインターフェイスがデフォルトで追加されます。次の手順を実行して値を編集します。
   a. **Interface name** フィールドにインターフェイスの名前を入力します。
   b. **Network state** ドロップダウンからネットワーク状態を選択します。デフォルトで選択されている値は **Up** です。
   c. **Type** ドロップダウンからインターフェイスのタイプを選択します。使用可能な値は、**Bridge**、**Bonding**、および **Ethernet** です。デフォルトで選択されている値は **Bridge** です。

注記

フォームを使用した VLAN インターフェイスの追加はサポートされていません。VLAN インターフェイスを追加するには、YAML を使用してポリシーを作成する必要があります。ポリシーを追加すると、フォームを使用してポリシーを編集できません。

d. オプション: IP 設定セクションで、**IPv4** チェックボックスをオンにしてインターフェイスに IPv4 アドレスを割り当て、IP アドレス割り当ての詳細を設定します。

i. **IP address** をクリックしてインターフェイスを静的 IP アドレスで設定するか、**DHCP** をクリックして IP アドレスを自動割り当てします。

ii. **IP address** オプションを選択した場合は、**IPV4 address** フィールドに IPv4 アドレスを、**Prefix length** フィールドに接頭辞長を入力します。
DHCP オプションを選択した場合は、無効にするオプションのチェックを外します。
使用可能なオプションは、**Auto-DNS**、**Auto-routes**、および **Auto-gateway** です。すべてのオプションがデフォルトで選択されています。

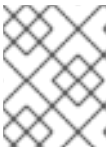
e. オプション: **Port** フィールドにポート番号を入力します。

f. オプション: STP を有効にするには、**Enable STP** チェックボックスをオンにします。

g. オプション: ポリシーにインターフェイスを追加するには、**Add another interface to the policy** をクリックします。

h. オプション: ポリシーからインターフェイスを削除するには、インターフェイスの横にある

アイコン をクリックします。



注記

または、ページ上部の **Edit YAML** をクリックして、YAML を使用してフォームの編集を続けることもできます。

7. **Create** をクリックしてポリシーの作成を完了します。

32.2.3.3. ポリシーの更新

32.2.3.3.1. フォームを使用してポリシーを更新する

手順

1. **Networking** → **NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで、編集するポリシーの横にあるアイコン を選択し、**Edit** をクリックします。
3. 更新するフィールドを編集します。
4. **Save** をクリックします。

注記

フォームを使用した VLAN インターフェイスの追加はサポートされていません。VLAN インターフェイスを追加するには、YAML を使用してポリシーを作成する必要があります。ポリシーを追加すると、フォームを使用してポリシーを編集することはできません。

32.2.3.3.2. YAML を使用したポリシーの更新

手順

1. **Networking** → **NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで、編集するポリシーの **Name** 列の下にあるポリシー名をクリックします。
3. **YAML** タブをクリックし、YAML を編集します。
4. **Save** をクリックします。

32.2.3.4. ポリシーの削除

手順

1. **Networking** → **NodeNetworkConfigurationPolicy** に移動します。
2. **NodeNetworkConfigurationPolicy** ページで、削除するポリシーの横にあるアイコン を選択し、**Delete** をクリックします。
3. ポップアップウィンドウで、削除を確認するポリシー名を入力し、**Delete** をクリックします。

32.2.4. CLI を使用したポリシーの管理

32.2.4.1. 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、ノードセレクターの **spec: nodeSelector** パラメーターおよび適切な **<key>:<value>** を追加します。

複数の nmstate 対応ノードを同時に設定できます。この設定は、並列のノードの 50% に適用されます。このストラテジーでは、ネットワーク接続に障害が発生した場合にクラスター全体が使用できなくなるのを回避します。クラスターの特定の部分に、ポリシーの並行設定を適用するには、**maxUnavailable**フィールドを使用します。

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、DNS リゾルバーを設定します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: "" (3)
  maxUnavailable: 3 (4)
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port (5)
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
          auto-dns: false
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1
    dns-resolver: (6)
      config:
        search:
        - example.com
        - example.org
        server:
        - 8.8.8.8
```

(1) 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 **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。

(4) オプション: ポリシー設定を同時に適用できる nmstate 対応ノードの最大数を指定します。このパラメーターは、 **10%**などのパーセンテージ値 (文字列)、または**3**などの絶対値 (数値) のいずれかに設定できます。

(5) オプション: インターフェイスの人間が判読できる説明。

(6) オプション:DNS サーバーの検索およびサーバー設定を指定します。

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 (1)
```

(1) 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

32.2.4.2. 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リスト表示します。

```
$ 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) のステータスをリスト表示できます。

```
$ 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
```

32.2.4.3. 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、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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
    - name: br1
      type: linux-bridge
      state: absent 4
    - name: eth1 5
      type: ethernet 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

1. 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 **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。
4. 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。
5. 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。
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. 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。

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> 1
```

1 ポリシーマニフェストのファイル名。

32.2.5. 異なるインターフェイスのポリシー設定の例

次の例は、さまざまな **NodeNetworkConfigurationPolicy** マニフェスト設定を示しています。

最高のパフォーマンスを得るには、ポリシーを適用するときに次の要素を考慮してください。

- ポリシーを複数のノードに適用する必要がある場合は、ターゲットノードごとに **NodeNetworkConfigurationPolicy** マニフェストを作成します。ポリシーの範囲を 1 つのノードに設定すると、Kubernetes NMState Operator によるポリシー適用にかかる総時間が短縮されます。
  対照的に、1 つのポリシーに複数のノードの設定が含まれている場合、Kubernetes NMState Operator はポリシーを各ノードに順番に適用するため、ポリシー適用にかかる総時間が長くなります。

- 関連するすべてのネットワーク設定を 1 つのポリシーで指定する必要があります。
  ノードが再起動すると、Kubernetes NMState Operator はポリシーを適用する順序を制御できません。したがって、Kubernetes NMState Operator が相互依存するポリシーを順番に適用すると、ネットワークオブジェクトがデグレード状態になる可能性があります。

32.2.5.1. 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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: br1 4
        description: Linux bridge with eth1 as a port 5
        type: linux-bridge 6
        state: up 7
        ipv4:
          dhcp: true 8
          enabled: true 9
        bridge:
          options:
            stp:
```

```
      enabled: false 10
    port:
      - name: eth1 11
```

1 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、ブリッジを作成します。

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。

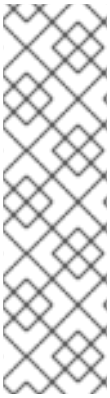
9 この例では **ipv4** を有効にします。

10 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。

32.2.5.2. 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。



注記

1 つの **NodeNetworkConfigurationPolicy** マニフェストで、ノードの VLAN インターフェイスに関連するすべての設定を定義します。たとえば、同じ **NodeNetworkConfigurationPolicy** マニフェストで、ノードの VLAN インターフェイスと VLAN インターフェイスの関連ルートを定義します。

ノードが再起動すると、Kubernetes NMState Operator はポリシーを適用する順序を制御できません。したがって、関連するネットワーク設定に別々のポリシーを使用すると、Kubernetes NMState Operator がこれらのポリシーを順番に適用するため、ネットワークオブジェクトがデグレード状態になる可能性があります。

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。

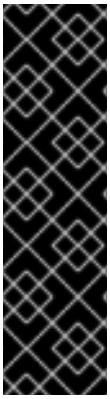
```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan-eth1-policy 1
spec:
  nodeSelector: 2
```

```
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: eth1.102 4
      description: VLAN using eth1 5
      type: vlan 6
      state: up 7
      vlan:
        base-iface: eth1 8
        id: 102 9
```

1. 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。ベアメタルにデプロイする場合、**<interface_name>.<vlan_number>** VLAN 形式のみがサポートされます。
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。

32.2.5.3. 例: Virtual Function のノードネットワーク設定ポリシー (テクノロジープレビュー)

NodeNetworkConfigurationPolicy マニフェストを適用して、既存のクラスター内の Single Root I/O Virtualization (SR-IOV) ネットワーク Virtual Function (VF) のホストネットワーク設定を更新します。



重要

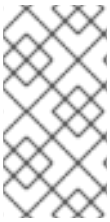
SR-IOV ネットワーク VF のホストネットワーク設定の更新は、テクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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。

NodeNetworkConfigurationPolicy マニフェストを既存のクラスターに適用して、次のタスクを完了できます。

- VF の QoS または MTU ホストネットワークを設定して、パフォーマンスを最適化します。
- ネットワークインターフェイスの VF を追加、削除、または更新します。

- VF ボンディング設定を管理します。



注記

SR-IOV Network Operator を通じて管理もされる物理機能で NMState を使用して SR-IOV VF のホストネットワーク設定を更新するには、関連する **SriovNetworkNodePolicy** リソースの **externallyManaging** パラメーターを **true** に設定する必要があります。詳細は、**関連情報** セクションを参照してください。

次の YAML ファイルは、VF の QoS ポリシーを定義するマニフェストの例です。このファイルにはサンプル値が含まれていますが、これは独自の情報に置き換える必要があります。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: qos 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
      - name: ens1f0 4
        description: Change QOS on VF0 5
        type: ethernet 6
        state: up 7
        ethernet:
          sr-iov:
            total-vfs: 3 8
            vfs:
            - id: 0 9
              max-tx-rate: 200 10
```

1. 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. この例は、**worker** ロールを持つすべてのノードに適用されます。
4. 物理機能 (PF) ネットワーク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。
7. 設定後のインターフェイスの要求された状態。
8. VF の総数。
9. ID **0** を持つ VF を識別します。
10. VF の最大伝送速度 (Mbps) を設定します。このサンプル値は、200 Mbps のレートを設定します。

次の YAML ファイルは、VF 上に VLAN インターフェイスを作成し、それをボンディングされたネットワークインターフェイスに追加する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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: addvf (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: "" (3)
  maxUnavailable: 3
  desiredState:
    interfaces:
      - name: ens1f0v1 (4)
        type: ethernet
        state: up
      - name: ens1f0v1.477 (5)
        type: vlan
        state: up
        vlan:
          base-iface: ens1f0v1 (6)
          id: 477
      - name: bond0 (7)
        description: Add vf (8)
        type: bond (9)
        state: up (10)
        link-aggregation:
          mode: active-backup (11)
          options:
            primary: ens1f1v0.477 (12)
          port: (13)
            - ens1f1v0.477
            - ens1f0v0.477
            - ens1f0v1.477 (14)
```

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. この例は、**worker** ロールを持つすべてのノードに適用されます。
4. VF ネットワークインターフェイスの名前。
5. VLAN ネットワークインターフェイスの名前。
6. VLAN インターフェイスがアタッチされている VF ネットワークインターフェイス。
7. ボンディングネットワークインターフェイスの名前。
8. オプション: 人間が判読できるインターフェイスの説明。
9. インターフェイスのタイプ。

10 設定後のインターフェイスの要求された状態。

11 ボンディングのボンディングポリシー。

12 割り当てられたプライマリーボンディングポート。

13 ボンディングされたネットワークインターフェイスのポート。

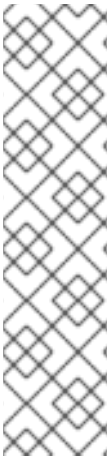
14 この例では、この VLAN ネットワークインターフェイスが、追加インターフェイスとしてボンディングされたネットワークインターフェイスに追加されています。

関連情報

- SR-IOV ネットワークデバイスの設定

32.2.5.4. 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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: bond0 4
      description: Bond with ports eth1 and eth2 5
      type: bond 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

```
    link-aggregation:
      mode: active-backup (10)
      options:
        miimon: '140' (11)
      port: (12)
      - eth1
      - eth2
    mtu: 1450 (13)
```

(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、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。

32.2.5.5. 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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1-policy (1)
spec:
  nodeSelector: (2)
    kubernetes.io/hostname: <node01> (3)
  desiredState:
```

```
  interfaces:
  - name: eth1 4
    description: Configuring eth1 on node01 5
    type: ethernet 6
    state: up 7
    ipv4:
      dhcp: true 8
      enabled: true 9
```

1 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。

32.2.5.6. 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イス

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イスを作成できます。これらのインターフェイ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 YAML ファイルの例では、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 **bond10.103** という名前の VLAN を作成します。

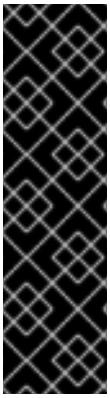
```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond-vlan 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: bond10 4
      description: Bonding eth2 and eth3 5
      type: bond 6
      state: up 7
      link-aggregation:
```

```
      mode: balance-rr 8
      options:
        miimon: '140' 9
      port: 10
      - eth2
      - eth3
  - name: bond10.103 11
    description: vlan using bond10 12
    type: vlan 13
    state: up 14
    vlan:
      base-iface: bond10 15
      id: 103 16
    ipv4:
      dhcp: true 17
      enabled: true 18
```

1. 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. 11 インターフェイスの名前。
5. 12 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. 13 インターフェイスのタイプ。
7. 14 作成後のインターフェイスの要求された状態。
8. ボンドのドライバーモード。
9. 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。
10. ボンド内の下位ノードの NIC。
15. VLAN が接続されているノードの NIC。
16. VLAN タグ。
17. 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。
18. この例では ipv4 を有効にします。

32.2.5.7. 例: VRF インスタンスノードのネットワーク設定ポリシーを使用したネットワークインターフェイス

NodeNetworkConfigurationPolicy カスタムリソース (CR) を適用して、仮想ルーティングおよび転送 (VRF) インスタンスをネットワークインターフェイスに関連付けます。



重要

VRF インスタンスとネットワークインターフェイスの関連付けは、テクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

VRF インスタンスをネットワークインターフェイスに関連付けることにより、トラフィックの分離、独立したルーティングの決定、およびネットワークリソースの論理的な分離をサポートできます。

ベアメタル環境では、MetalLB を使用して、VRF インスタンスに属するインターフェイスを通じてロードバランサーサービスを通知できます。詳細は、**関連情報** セクションを参照してください。

次の YAML ファイルは、VRF インスタンスをネットワークインターフェイスに関連付ける例です。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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vrfpolicy 1
spec:
  nodeSelector:
    vrf: "true" 2
  maxUnavailable: 3
  desiredState:
    interfaces:
      - name: ens4vrf 3
        type: vrf 4
        state: up
        vrf:
          port:
            - ens4 5
          route-table-id: 2 6
```

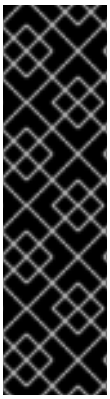
1. ポリシーの名前。
2. この例では、**vrf:true** のラベルが割り当てられたべてのノードにポリシーを適用します。
3. インターフェイスの名前。
4. インターフェイスのタイプ。この例では VRF インスタンスを作成します。
5. VRF が接続されるノードインターフェイス。
6. VRF のルートテーブル ID の名前。

関連情報

- 仮想ルーティングおよび転送について

- ネットワーク VRF を介したサービスの公開

32.2.6. ブリッジに接続された NIC の静的 IP の取得



重要

NIC の静的 IP のキャプチャーは、テクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

32.2.6.1. 例: ブリッジに接続された NIC から静的 IP アドレスを継承する 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

クラスター内のノードに Linux ブリッジインターフェイスを作成し、単一の **NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用して NIC の静的 IP 設定をブリッジに転送します。

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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-copy-ipv4-policy (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: ""
  capture:
    eth1-nic: interfaces.name=="eth1" (3)
    eth1-routes: routes.running.next-hop-interface=="eth1"
    br1-routes: capture.eth1-routes | routes.running.next-hop-interface := "br1"
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port
        type: linux-bridge (4)
        state: up
        ipv4: "{{ capture.eth1-nic.interfaces.0.ipv4 }}" (5)
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1 (6)
    routes:
      config: "{{ capture.br1-routes.routes.running }}"
```

1. 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。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。
3. ブリッジを接続するノード NIC への参照。
4. 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。
5. ブリッジインターフェイスの IP アドレス。この値は、**spec.capture.eth1-nic** エントリーにより参照される NIC の IP アドレスと一致します。
6. ブリッジが接続されるノードの NIC。

関連情報

- The NMPolicy project - Policy syntax

32.2.7. 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。

32.2.7.1. 静的

以下のスニペットは、イーサネットインターフェイスで 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。

32.2.7.2. 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
# ...
```

32.2.7.3. 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
# ...
```

32.2.7.4. DNS

DNS 設定の定義は、**/etc/resolv.conf** ファイルの変更に類似しています。以下のスニペットは、ホストに DNS 設定を定義します。

```
# ...
    interfaces: 1
      ...
      ipv4:
        ...
        auto-dns: false
        ...
    dns-resolver:
      config:
```

```
      search:
      - example.com
      - example.org
      server:
      - 8.8.8.8
# ...
```

1 Kubernetes NMState がカスタム DNS 設定を保存するためには、インターフェイスを **auto-dns: false** で設定するか、インターフェイスで静的 IP 設定を使用する必要があります。

重要

DNS リゾルバーの設定時に、インターフェイスとして OVNKubernetes が管理する Open vSwitch ブリッジである **br-ex** を使用することはできません。

32.2.7.5. 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。

32.3. 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。

32.3.1. 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 ブリッジポリシーが、3 つのコントロールプレーンノードと 3 つのコンピュートノードを持つサンプルクラスターに適用されます。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。その後に、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/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with the wrong port
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: ens01
```

```
$ oc apply -f ens01-bridge-testfail.yaml
```

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、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) をリスト表示し、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** をリスト表示します。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33章 クラスター全体のプロキシーの設定

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 プロキシーを使用することができます。既存クラスターのプロキシー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ー設定を行うことにより、OpenShift Container Platform をプロキシーを使用するように設定できます。

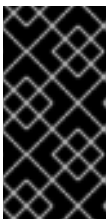
33.1. 前提条件

- クラスターがアクセスする必要のあるサイト を確認し、プロキシー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) はプロキシーされます。システム全体のプロキシーは、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。プロキシーオブジェクトの **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ーをバイパスします。

注記

プロキシーオブジェクトの **status.noProxy** フィールドには、ほとんどのインストールタイプのインストール設定における **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。



重要

使用しているインストールタイプに **networking.machineNetwork[].cidr** フィールドの設定が含まれていない場合は、ノード間のトラフィックがプロキシーをバイパスできるようにするために、**.status.noProxy** フィールドにマシンの IP アドレスを手動で含める必要があります。

33.2. クラスター全体のプロキシーの有効化

Proxy オブジェクトは、クラスター全体の egress プロキシーを管理するために使用されます。プロキシー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 のプロキシーを設定できます。

注記

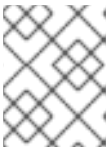
cluster という名前の **Proxy** オブジェクトのみがサポートされ、追加のプロキシーを作成することはできません。

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 接続のプロキシーに必要な追加の CA 証明書が含まれる config map を作成します。



注記

プロキシーのアイデンティティー証明書が 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) プロキシーのアイデンティティー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. プロキシー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ー 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ー 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ーの URL を指定します。たとえば、ほとんどのプロキシーは、**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ーを使用している場合は、プロキシーが使用する 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. プロキシー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ーをバイパスします。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、それらをこのリストに追加し、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 接続のプロキシーに必要な追加の 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 が存在している必要があります。このフィールドは、プロキシーのアイデンティティー証明書が 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。

33.3. クラスター全体のプロキシーの削除

cluster プロキシー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ーを削除するには、プロキシー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ーを変更します。

```
$ oc edit proxy/cluster
```

2. プロキシー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
- プロキシー証明書のカスタマイズ

第34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。

プロキシー 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ーを設定します。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 はプロキシーオブジェクトの **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ー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ー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** でプロキシー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ーリソースの **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、プロキシーの **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。

34.1. インストール時のクラスター全体のプロキシーの設定

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 プロキシーを使用することができます。プロキシー設定を **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ー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ー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) はプロキシーされます。プロキシー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。

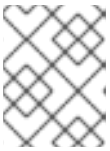
Amazon Web Services (AWS)、Google Cloud Platform (GCP)、Microsoft Azure、および Red Hat OpenStack Platform (RHOSP) へのインストールの場合、**Proxy** オブジェクトの **status.noProxy** フィールドには、インスタンスメタデータのエンドポイント (**169.254.169.254**) も設定されます。

手順

1. **install-config.yaml** ファイルを編集し、プロキシー設定を追加します。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: ec2.<aws_region>.amazonaws.com,elasticloadbalancing.
<aws_region>.amazonaws.com,s3.<aws_region>.amazonaws.com 3
additionalTrustBundle: | 4
    -----BEGIN CERTIFICATE-----
    <MY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
additionalTrustBundlePolicy: <policy_to_add_additionalTrustBundle> 5
```

1. 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ー 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ー URL。
3. プロキシー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 のコンマ区切りのリスト。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ーをバイパスします。Amazon **EC2**、**Elastic Load Balancing**、および **S3** VPC エンドポイントを VPC に追加した場合は、これらのエンドポイントを **noProxy** フィールドに追加する必要があります。
4. 指定されている場合、インストールプログラムは HTTPS 接続のプロキシーに必要な 1 つ以上の追加の CA 証明書が含まれる **user-ca-bundle** という名前の設定マップを **openshift-config** namespace に生成します。次に Cluster Network Operator は、これらのコンテンツを Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージする **trusted-ca-bundle** 設定マップを作成し、この設定マップは **Proxy** オブジェクトの **trustedCA** フィールドで参照されます。**additionalTrustBundle** フィールドは、プロキシーのアイデンティティー証明書が 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。
5. オプション： **trustedCA** フィールドの **user-ca-bundle** 設定マップを参照する **Proxy** オブジェクトの設定を決定するポリシー。許可される値は **Proxyonly** および **Always** です。**Proxyonly** を使用して、**http/https** プロキシーが設定されている場合にのみ **user-ca-bundle** 設定マップを参照します。**Always** を使用して、常に **user-ca-bundle** 設定マップを参照します。デフォルト値は **Proxyonly** です。



注記

インストールプログラムは、プロキシーの **readinessEndpoints** フィールドをサポートしません。

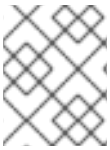
注記

インストーラーがタイムアウトした場合は、インストーラーの **wait-for** コマンドを使用してデプロイメントを再起動してからデプロイメントを完了します。以下に例を示します。

```
$ ./openshift-install wait-for install-complete --log-level debug
```

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ー設定を使用する **cluster** という名前のクラスター全体のプロキシーを作成します。プロキシー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、追加のプロキシーを作成することはできません。

34.2. クラスター全体のプロキシーの有効化

Proxy オブジェクトは、クラスター全体の egress プロキシーを管理するために使用されます。プロキシー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 のプロキシーを設定できます。

注記

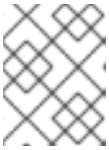
cluster という名前の **Proxy** オブジェクトのみがサポートされ、追加のプロキシーを作成することはできません。

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 接続のプロキシーに必要な追加の CA 証明書が含まれる config map を作成します。



注記

プロキシーのアイデンティティー証明書が 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. プロキシーのアイデンティティー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. プロキシー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ー 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ー 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ーの URL を指定します。たとえば、ほとんどのプロキシーは、**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ーを使用している場合は、プロキシーが使用する 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. プロキシー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ーをバイパスします。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、それらをこのリストに追加し、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 接続のプロキシーに必要な追加の 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 が存在している必要があります。このフィールドは、プロキシーのアイデンティティー証明書が 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。

34.3. 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。

重要

config.openshift.io/inject-trusted-cabundle="true" ラベルを config map に追加すると、そこに格納されている既存データが削除されます。Cluster Network Operator は config map の所有権を取得し、**ca-bundle** をデータとしてのみ受け入れます。**service.beta.openshift.io/inject-cabundle=true** アノテーションまたは同様の設定を使用して **service-ca.crt** を保存するには、別の config map を使用する必要があります。同じ config map に **config.openshift.io/inject-trusted-cabundle="true"** ラベルと **service.beta.openshift.io/inject-cabundle=true** アノテーションを追加すると、問題が発生する可能性があります。

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。

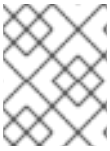
```
config.openshift.io/inject-trusted-cabundle="true"
```

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 をコンテナー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 内の各コンテナー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35章 RHOSP での負荷分散

35.1. ロードバランサーサービスの制限

Red Hat OpenStack Platform (RHOSP) 上の OpenShift Container Platform クラスターは、Octavia を使用してロードバランサーサービスを処理します。その結果、該当するクラスターには多くの機能的制限が生じます。

RHOSP Octavia では、Amphora と OVN の 2 つのプロバイダーがサポートされています。これらのプロバイダーでは、利用可能な機能と実装の詳細が異なります。そのような差異は、クラスター上に作成されるロードバランサーサービスに影響を及ぼします。

35.1.1. ローカルの外部トラフィックポリシー

ロードバランサーサービスで外部トラフィックポリシー (ETP) パラメーター **.spec.externalTrafficPolicy** を設定して、受信トラフィックがサービスエンドポイント Pod に到達する際に、そのソース IP アドレスを保存できます。ただし、クラスターが Amphora Octavia プロバイダーを使用している場合じゃ、トラフィックのソース IP は Amphora 仮想マシンの IP アドレスに置き換えられます。クラスターが OVN Octavia プロバイダーを使用している場合、この動作は発生しません。

ETP オプションを **Local** に設定する場合は、ロードバランサー用にヘルスモニターを作成する必要があります。ヘルスモニターがないと、トラフィックは機能エンドポイントを持たないノードにルーティングされる可能性があり、そうなると接続が切断されます。ヘルスモニターの作成を Cloud Provider OpenStack に強制するには、クラウドプロバイダー設定の **create-monitor** オプションの値を **true** に設定する必要があります。

RHOSP 16.2 では、OVN Octavia プロバイダーはヘルスモニターをサポートしません。そのため、EPP をローカルに設定することはサポートされていません。

RHOSP 16.2 では、Amphora Octavia プロバイダーは UDP プールでの HTTP モニターをサポートしません。その結果、UDP ロードバランサーサービスには **UDP-CONNECT** モニターが代わりに作成されます。実装の詳細に基づき、この設定は OVN-Kubernetes CNI プラグインでのみ適切に機能します。OpenShift SDN CNI プラグインを使用している場合、UDP サービスのアクティブなノードの検出が不確実になります。この問題は、ドライバーが HTTP ヘルスモニターをサポートしていないため、RHOSP バージョンの OVN Octavia プロバイダーにも影響します。

35.2. 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。

アプリケーションのネットワークのスケーリングに使用する、独自の Octavia ロードバランサーを作成する必要があります。

35.2.1. Octavia を使用したクラスターのスケーリング

複数の API ロードバランサーを使用する場合、Octavia ロードバランサーを作成し、それを使用するようにクラスターを設定します。

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、ヘルスモ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
```

7. 設定を解除された **API_FIP**、または新規アドレスを、作成されたロードばランサー 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。

35.3. 外部ロードバランサー用のサービス

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。

重要

外部ロードバランサーの設定は、ベンダーのロードバランサーによって異なります。

このセクションの情報と例は、ガイドラインのみを目的としています。ベンダーのロードバランサーに関する詳細は、ベンダーのドキュメントを参照してください。

Red Hat は、外部ロードバランサーに対して次のサービスをサポートしています。

- Ingress Controller
- OpenShift API
- OpenShift MachineConfig API

外部ロードバランサーに対して、これらのサービスの 1 つまたはすべてを設定するように選択できます。一般的な設定オプションは、Ingress Controller サービスのみを設定することです。次の図は、各サービスの詳細を示しています。

図35.1 OpenShift Container Platform 環境で動作する Ingress Controller を示すネットワークワークフローの例

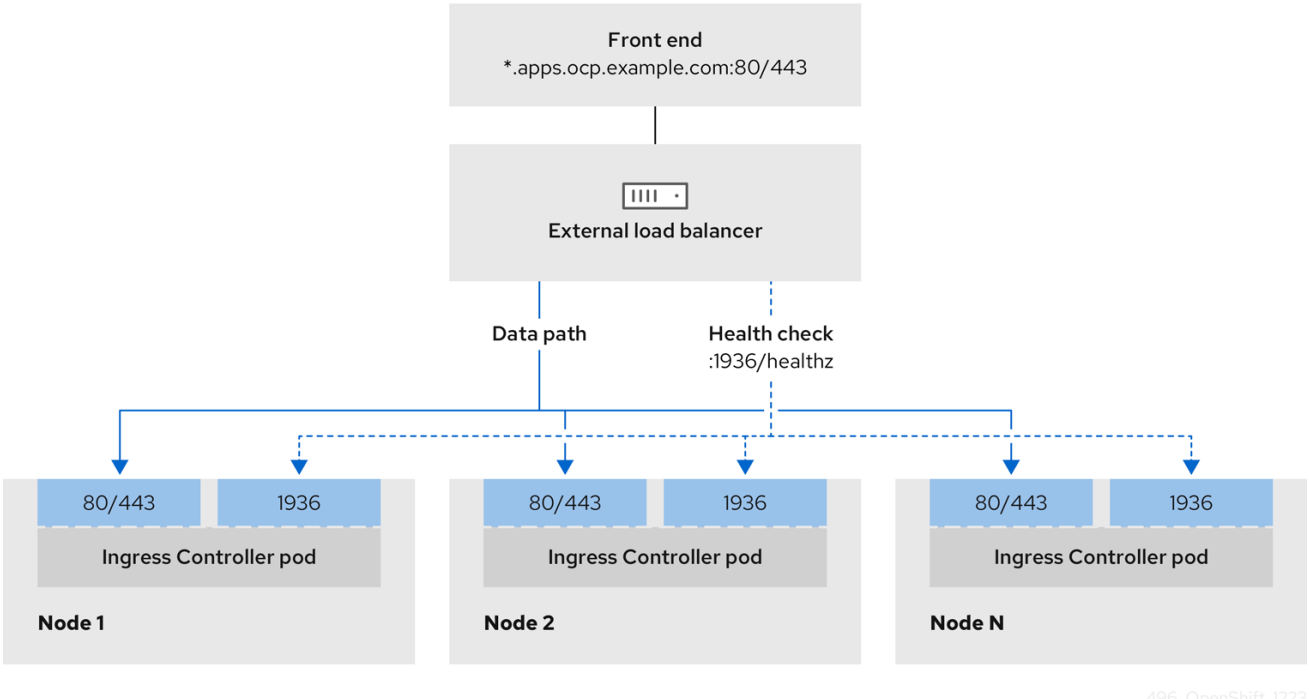


図35.2 OpenShift Container Platform 環境で動作する OpenShift API を示すネットワークワークフローの例

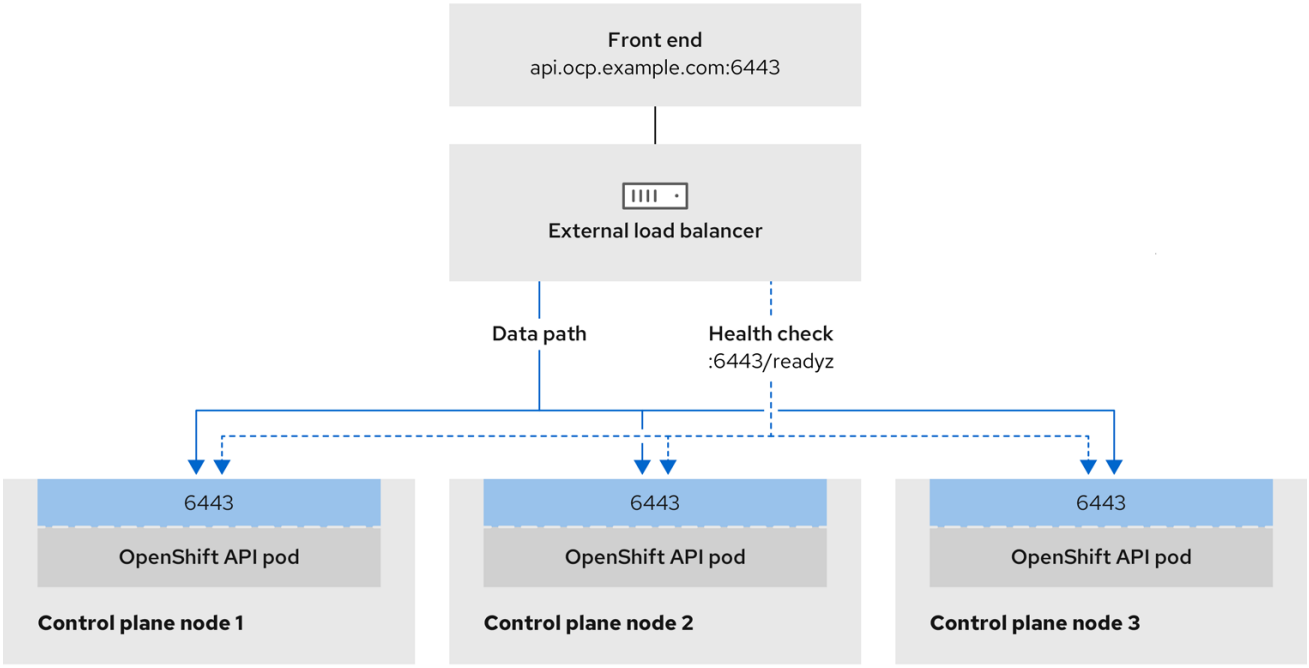
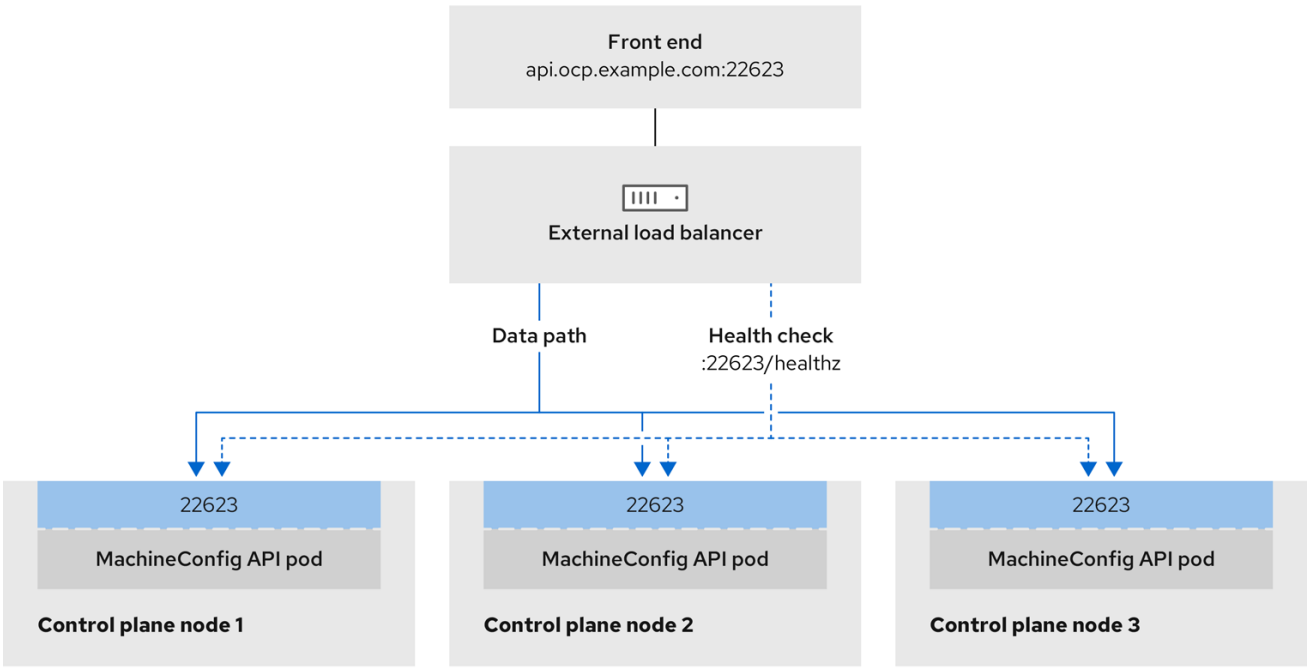


図35.3 OpenShift Container Platform 環境で動作する OpenShift MachineConfig API を示すネットワークワークフローの例



外部ロードバランサーでは、次の設定オプションがサポートされています。

- ノードセレクターを使用して、Ingress Controller を特定のノードのセットにマッピングします。このセットの各ノードに静的 IP アドレスを割り当てるか、 Dynamic Host Configuration Protocol (DHCP) から同じ IP アドレスを受け取るように各ノードを設定する必要があります。インフラストラクチャーノードは通常、このタイプの設定を受け取ります。

- サブネット上のすべての IP アドレスをターゲットにします。この設定では、ロードバランサーターゲットを再設定せずにネットワーク内でノードを作成および破棄できるため、メンテナンスオーバーヘッドを削減できます。**/27** や **/28** などの小規模なネットワーク上に設定されたマシンを使用して Ingress Pod をデプロイする場合、ロードバランサーのターゲットを簡素化できます。

  ヒント

  マシン config プールのリソースを確認することで、ネットワーク内に存在するすべての IP アドレスをリスト表示できます。

OpenShift Container Platform クラスターの外部ロードバランサーを設定する前に、以下の情報を考慮してください。

- フロントエンド IP アドレスの場合、フロントエンド IP アドレス、Ingress Controller のロードバランサー、および API ロードバランサーに同じ IP アドレスを使用できます。この機能については、ベンダーのドキュメントを確認してください。

- バックエンド IP アドレスの場合、OpenShift Container Platform コントロールプレーンノードの IP アドレスが、外部ロードバランサーの存続期間中に変更されないようにください。次のいずれかのアクションを実行すると、これを実現できます。

  - 各コントロールプレーンノードに静的 IP アドレスを割り当てます。

  - ノードが DHCP リースを要求するたびに、DHCP から同じ IP アドレスを受信するように各ノードを設定します。ベンダーによっては、DHCP リースは IP 予約または静的 DHCP 割り当ての形式になる場合があります。

- Ingress Controller バックエンドサービスの外部ロードバランサーで、Ingress Controller を実行する各ノードを手動で定義します。たとえば、Ingress Controller が未定義のノードに移動すると、接続が停止する可能性があります。

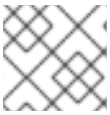
35.3.1. 外部ロードバランサーの設定

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。

重要

外部ロードバランサーを設定する前に、「外部ロードバランサー用のサービス」セクションを必ず確認してください。

外部ロードバランサー用に設定するサービスに適用される次の前提条件を確認してください。



注記

クラスター上で動作する MetalLB は、外部ロードバランサーとして機能します。

OpenShift API の前提条件

- フロントエンド IP アドレスを定義している。

- TCP ポート 6443 および 22623 は、ロードバランサーのフロントエンド IP アドレスで公開されている。以下の項目を確認します。

- ポート 6443 が OpenShift API サービスにアクセスできる。
- ポート 22623 が Ignition 起動設定をノードに提供できる。

- フロントエンド IP アドレスとポート 6443 へは、OpenShift Container Platform クラスターの外部の場所にいるシステムのすべてのユーザーがアクセスできる。
- フロントエンド IP アドレスとポート 22623 は、OpenShift Container Platform ノードからのみ到達できる。
- ロードバランサーバックエンドは、ポート 6443 および 22623 の OpenShift Container Platform コントロールプレーンノードと通信できる。

Ingress Controller の前提条件

- フロントエンド IP アドレスを定義している。
- TCP ポート 443 および 80 はロードバランサーのフロントエンド IP アドレスで公開されている。
- フロントエンドの IP アドレス、ポート 80、ポート 443 へは、OpenShift Container Platform クラスターの外部の場所にあるシステムの全ユーザーがアクセスできる。
- フロントエンドの IP アドレス、ポート 80、ポート 443 は、OpenShift Container Platform クラスターで動作するすべてのノードから到達できる。
- ロードバランサーバックエンドは、ポート 80、443、および 1936 で Ingress Controller を実行する OpenShift Container Platform ノードと通信できる。

ヘルスチェック URL 仕様の前提条件

ほとんどのロードバランサーは、サービスが使用可能か使用不可かを判断するヘルスチェック URL を指定して設定できまうｓ．OpenShift Container Platform は、OpenShift API、Machine Configuration API、および Ingress Controller バックエンドサービスのこれらのヘルスチェックを提供します。

次の例は、前にリスト表示したバックエンドサービスのヘルスチェック仕様を示しています。

Kubernetes API ヘルスチェック仕様の例

```
Path: HTTPS:6443/readyz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

Machine Config API ヘルスチェック仕様の例

```
Path: HTTPS:22623/healthz
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 10
Interval: 10
```

Ingress Controller のヘルスチェック仕様の例

```
Path: HTTP:1936/healthz/ready
Healthy threshold: 2
Unhealthy threshold: 2
Timeout: 5
Interval: 10
```

手順

1. HAProxy Ingress Controller を設定して、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化できるようにします。

   HAProxy 設定の例

```
#...
listen my-cluster-api-6443
    bind 192.168.1.100:6443
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /readyz
  http-check expect status 200
    server my-cluster-master-2 192.168.1.101:6443 check inter 10s rise 2 fall 2
    server my-cluster-master-0 192.168.1.102:6443 check inter 10s rise 2 fall 2
    server my-cluster-master-1 192.168.1.103:6443 check inter 10s rise 2 fall 2

listen my-cluster-machine-config-api-22623
    bind 192.168.1.100:22623
    mode tcp
    balance roundrobin
  option httpchk
  http-check connect
  http-check send meth GET uri /healthz
  http-check expect status 200
    server my-cluster-master-2 192.168.1.101:22623 check inter 10s rise 2 fall 2
    server my-cluster-master-0 192.168.1.102:22623 check inter 10s rise 2 fall 2
    server my-cluster-master-1 192.168.1.103:22623 check inter 10s rise 2 fall 2

listen my-cluster-apps-443
        bind 192.168.1.100:443
        mode tcp
        balance roundrobin
    option httpchk
    http-check connect
    http-check send meth GET uri /healthz/ready
    http-check expect status 200
        server my-cluster-worker-0 192.168.1.111:443 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-1 192.168.1.112:443 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-2 192.168.1.113:443 check port 1936 inter 10s rise 2 fall 2

listen my-cluster-apps-80
        bind 192.168.1.100:80
        mode tcp
        balance roundrobin
    option httpchk
```

```
    http-check connect
    http-check send meth GET uri /healthz/ready
    http-check expect status 200
        server my-cluster-worker-0 192.168.1.111:80 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-1 192.168.1.112:80 check port 1936 inter 10s rise 2 fall 2
        server my-cluster-worker-2 192.168.1.113:80 check port 1936 inter 10s rise 2 fall 2
# ...
```

2. **curl** CLI コマンドを使用して、外部ロードバランサーとそのリソースが動作していることを確認します。

   a. 次のコマンドを実行して応答を観察し、クラスターマシン設定 API が Kubernetes API サーバーリソースにアクセスでき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"
   }
   ```

   b. 次のコマンドを実行して出力を確認し、クラスターマシン設定 API がマシン設定サーバーリソースからアクセスできることを確認します。

   ```
   $ curl -v https://<loadbalancer_ip_address>:22623/healthz --insecure
   ```

   設定が正しい場合、コマンドの出力には次の応答が表示されます。

   ```
   HTTP/1.1 200 OK
   Content-Length: 0
   ```

   c. 次のコマンドを実行して出力を確認し、コントローラーがポート 80 の Ingress Controller リソースにアクセスできることを確認します。

   ```
   $ curl -I -L -H "Host: console-openshift-console.apps.<cluster_name>.<base_domain>"
   http://<load_balancer_front_end_IP_address>
   ```

   設定が正しい場合、コマンドの出力には次の応答が表示されます。

   ```
   HTTP/1.1 302 Found
   content-length: 0
   location: https://console-openshift-console.apps.ocp4.private.opequon.net/
   cache-control: no-cache
   ```

d. 次のコマンドを実行して出力を確認し、コントローラーがポート 443 の Ingress Controller リソースにアクセスできることを確認します。

```
$ curl -I -L --insecure --resolve console-openshift-console.apps.<cluster_name>.
<base_domain>:443:<Load Balancer Front End IP Address> https://console-openshift-
console.apps.<cluster_name>.<base_domain>
```

設定が正しい場合、コマンドの出力には次の応答が表示されます。

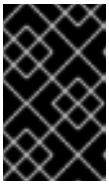
```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja261dG
LgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

3. 外部ロードバランサーのフロントエンド IP アドレスをターゲットにするように、クラスターの DNS レコードを設定します。ロードバランサー経由で、クラスター API およびアプリケーションの DNS サーバーのレコードを更新する必要があります。

変更された DNS レコードの例

```
<load_balancer_ip_address>  A  api.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```

```
<load_balancer_ip_address>   A apps.<cluster_name>.<base_domain>
A record pointing to Load Balancer Front End
```



重要

DNS の伝播では、各 DNS レコードが使用可能になるまでに時間がかかる場合があります。各レコードを検証する前に、各 DNS レコードが伝播されることを確認してください。

4. **curl** CLI コマンドを使用して、外部ロードバランサーと DNS レコード設定が動作していることを確認します。

a. 次のコマンドを実行して出力を確認し、クラスター API にアクセスできることを確認します。

```
$ curl https://api.<cluster_name>.<base_domain>: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"
  }
```

b. 次のコマンドを実行して出力を確認し、クラスターマシン設定にアクセスできることを確認します。

```
$ curl -v https://api.<cluster_name>.<base_domain>:22623/healthz --insecure
```

設定が正しい場合、コマンドの出力には次の応答が表示されます。

```
HTTP/1.1 200 OK
Content-Length: 0
```

c. 以下のコマンドを実行して出力を確認し、ポートで各クラスターアプリケーションにアクセスできることを確認します。

```
$ curl http://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --
insecure
```

設定が正しい場合、コマンドの出力には次の応答が表示されます。

```
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/JuLJMeoKNXlfiVv2YgZc09c3TBOBU4NI6kDXaJH1LdicNhN1UsQ
Wzon4Dor9GWGfopaTEQ==; Path=/; Secure
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Tue, 17 Nov 2020 08:42:10 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=9b714eb87e93cf34853e87a92d6894be; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

d. 次のコマンドを実行して出力を確認し、ポート 443 で各クラスターアプリケーションにアクセスできることを確認します。

```
$ curl https://console-openshift-console.apps.<cluster_name>.<base_domain> -I -L --
insecure
```

-

設定が正しい場合、コマンドの出力には次の応答が表示されます。

```
HTTP/1.1 200 OK
referrer-policy: strict-origin-when-cross-origin
set-cookie: csrf-
token=UlYWOyQ62LWjw2h003xtYSKlh1a0Py2hhctw0WmV2YEdhJjFyQwWcGBsja261dG
LgaYO0nxzVErhiXt6QepA7g==; Path=/; Secure; SameSite=Lax
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-frame-options: DENY
x-xss-protection: 1; mode=block
date: Wed, 04 Oct 2023 16:29:38 GMT
content-type: text/html; charset=utf-8
set-cookie:
1e2670d92730b515ce3a1bb65da45062=1bf5e9573c9a2760c964ed1659cc1673; path=/;
HttpOnly; Secure; SameSite=None
cache-control: private
```

第36章 METALLB を使用した負荷分散

36.1. 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。

36.1.1. 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。

レイヤ 2 モードで動作する MetalLB は、IP フェイルオーバーと同様のメカニズムを利用してフェイルオーバーをサポートします。ただし、仮想ルーター冗長プロトコル (VRRP) とキープアライブに依存する代わりに、MetalLB はゴシップベースのプロトコルを利用してノード障害のインスタンスを識別します。フェイルオーバーが検出されると、別のノードがリーダーノードのロールを引き継ぎ、Gratuitous ARP メッセージがディスパッチされて、この変更がブロードキャストされます。

レイヤ 3 またはボーダーゲートウェイプロトコル (BGP) モードで動作する MetalLB は、障害検出をネットワークに委任します。OpenShift Container Platform ノードが接続を確立した BGP ルーターは、ノードの障害を識別し、そのノードへのルートを終了します。

Pod とサービスの高可用性を確保するには、IP フェイルオーバーの代わりに MetalLB を使用することを推奨します。

36.1.2. MetalLB Operator カスタムリソース

Metal LB Operator は、次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。

IPAddressPool

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。**IPAddressPool** には、IP アドレスのリストが含まれています。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。**IPAddressPool** 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。MetalLB **controller** は、**IPAddressPool** 内のアドレスのプールから IP アドレスを割り当てます。**L2Advertisement** および **BGPAdvertisement** カスタムリソースは、指定されたプールからの指定された IP のアドバタイズメントを有効にします。**IPAddressPool** カスタムリソースの **IPAddressPool** 仕様を使用して、**spec.serviceAllocation** からサービスと namespace に IP アドレスを割り当てることができます。

注記

単一の **IPAddressPool** は、L2 アドバタイズメントと BGP アドバタイズメントによって参照できます。

BGPPeer

BGP ピアカスタムリソースは、通信する MetalLB の BGP ルーター、ルーターの AS 番号、MetalLB の AS 番号、およびルートアドバタイズメントのカスタマイズを識別します。MetalLB は、サービスロードバランサーの IP アドレスのルートを 1 つ以上の BGP ピアにアドバタイズします。

BFDProfile

BFD プロファイルカスタムリソースは、BGP ピアの双方向フォワーディング検出 (BFD) を設定します。BFD は、BGP のみよりも、パスの障害検出が高速になります。

L2Advertisement

L2Advertisement カスタムリソースは、L2 プロトコルを使用して **IPAddressPool** からの IP をアドバタイズします。

BGPAdvertisement

BGPAdvertisement カスタムリソースは、BGP プロトコルを使用して **IPAddressPool** からの IP をアドバタイズします。

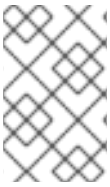
MetalLB カスタムリソースをクラスターに追加し、Operator が MetalLB をデプロイすると、**controller** および **speaker** MetalLB ソフトウェアコンポーネントは実行を開始します。

MetalLB は、関連するすべてのカスタムリソースを検証します。

36.1.3. 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、関連するすべてのリソースへの変更を監視します。

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。



注記

MetalLB カスタムリソースでデプロイメント仕様を設定して、**controller** および **speaker** Pod がクラスターへのデプロイおよび実行方法を管理できます。これらの展開仕様の詳細については、**その他のリソース** セクションを参照してください。

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、**speaker**Pod 用に設定されたデーモンを起動します。デフォルトでは、Pod はクラスター内の各ノードで起動されます。Metal LB の起動時に**MetalLB**カスタムリソースでノードセレクターを指定して、Pod を特定のノードに制限できます。**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 モードの場合には、**コントローラー** がサービスに IP アドレスを割り当てた後に、**speaker**Pod はアルゴリズムを使用して、どのノードの、どの**speaker**Pod がロードバランサーの IP アドレスをアナウンスするかを決定します。このアルゴリズムには、ノード名とロードバランサーの IP アドレスのハッシュが含まれます。詳細は、MetalLB と外部トラフィックポリシーを参照してください。**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。

Border Gateway Protocol (BGP) モードの場合、**コントローラー**がサービスに IP アドレスを割り当てた後に、各**speaker** Pod はロードバランサーの IP アドレスを BGP ピアにアドバタイズします。どのノードが BGP ピアとの BGP セッションを開始するかを設定できます。

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、サービスプロキシー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。サービスプロキシーは、接続が確立されるたびにエンドポイントを選択します。

36.1.4. MetalLB と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。BGP モードでは、ホストネットワーク上のルーターが、新しいクライアントが接続を確立できるように、クラスター内のノードの 1 つに接続を開きます。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、サービスプロキシー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、サービスプロキシー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、サービスプロキシーはノード 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。

注記

次の情報は、BGP モードで外部トラフィックポリシーを設定する場合に重要です。

MetalLB は、適格なすべてのノードからロードバランサーの IP アドレスをアドバタイズしますが、サービスのロードバランシングを行うノードの数は、等コストマルチパス (ECMP) ルートを確立するルーターの容量によって制限される場合があります。IP をアドバタイズするノードの数がルーターの ECMP グループ制限よりも多い場合、ルーターは IP をアドバタイズするノードよりも少ないノードを使用します。

たとえば、外部トラフィックポリシーが **local** に設定され、ルーターの ECMP グループ制限が 16 に設定され、LoadBalancer サービスを実装する Pod が 30 ノードにデプロイされている場合、14 ノードにデプロイされた Pod はトラフィックを受信しません。この状況では、サービスの外部トラフィックポリシーを **cluster** に設定することを推奨します。

36.1.5. 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

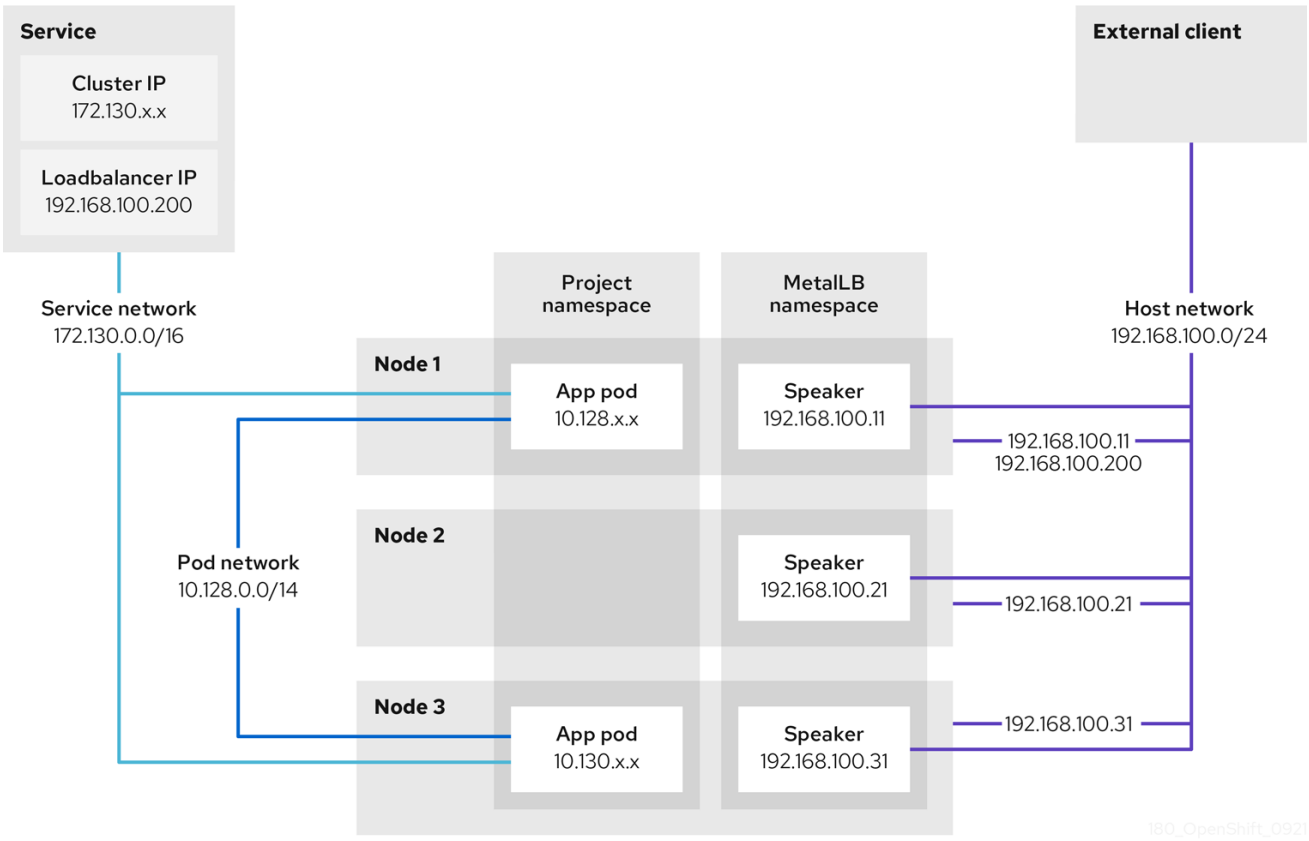
レイヤ 2 モードでは、MetalLB は ARP と NDP に依存します。これらのプロトコルは、特定のサブネット内でローカルアドレス解決を実装します。このコンテキストでは、MetalLB が機能するために、クライアントは、サービスをアナウンスするノードと同じサブネット上に存在する MetalLB によって割り当てられた VIP に到達できなければなりません。

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ー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。



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** デーモンセットは、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、サービスプロキシーは、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。
  - サービスの外部トラフィックポリシーが **cluster** に設定されている場合、**speaker** Pod が実行されているノードから **192.168.100.200** ロードバランサーの IP アドレスをアドバタイズするノードが選択されます。そのノードのみがサービスのトラフィックを受信できます。
  - サービスの外部トラフィックポリシーが **local** に設定されている場合、**speaker** Pod が実行されているノードと少なくとも 1 つのサービスエンドポイントから **192.168.100.200** ロードバランサーの IP アドレスをアドバタイズするノードが選択されます。そのノードのみが

サービスのトラフィックを受信できます。前の図では、ノード 1 または 3 のいずれかが **192.168.100.200** をアドバタイズします。

- 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。

36.1.6. BGP モードの MetalLB の概念

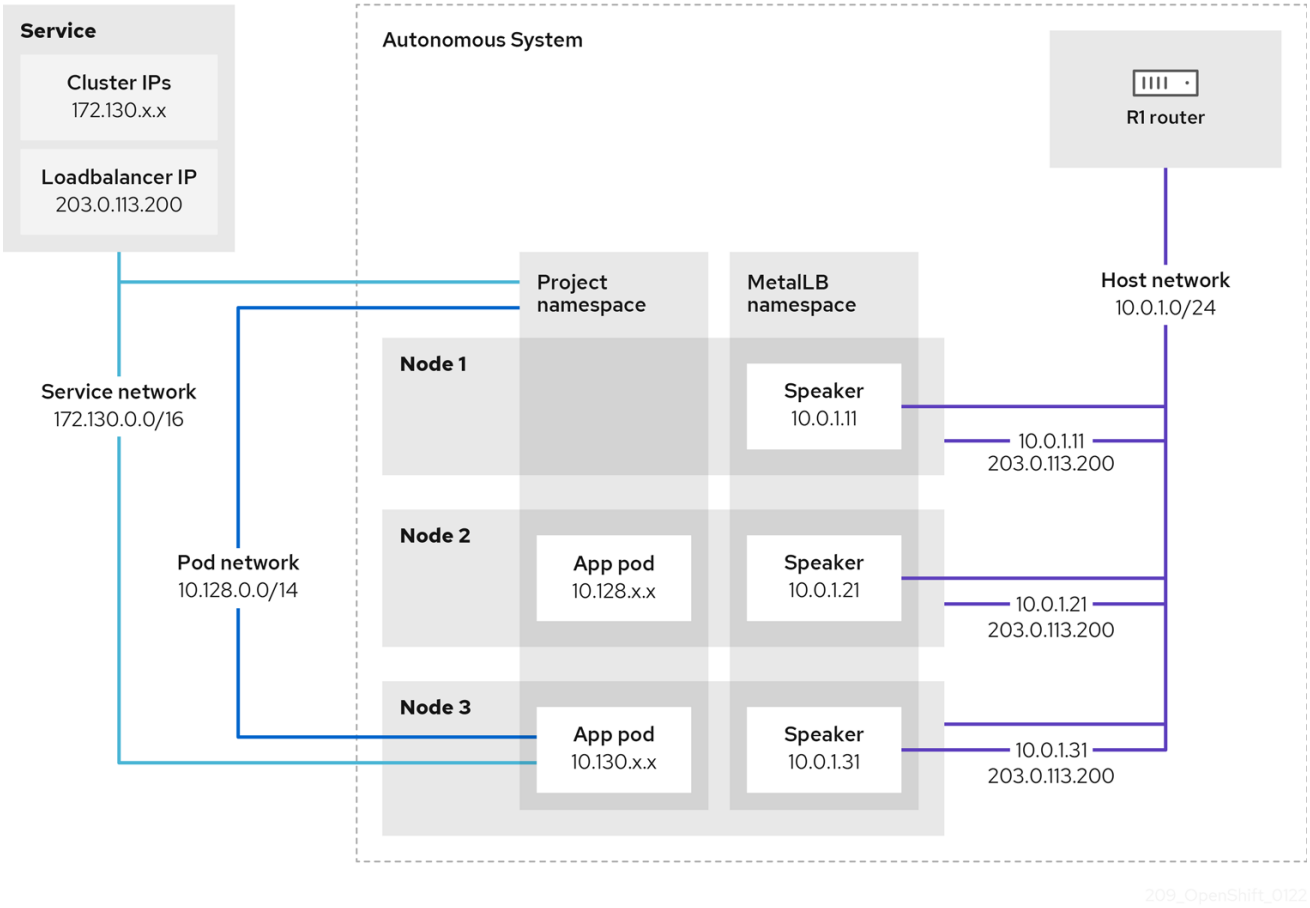
BGP モードでは、デフォルトで各 **speaker** Pod がサービスのロードバランサー IP アドレスを各 BGP ピアにアドバタイズします。オプションの BGP ピアのリストを追加すると、指定されたプールからの IP を指定されたピアセットにアドバタイズすることもできます。BGP ピアは通常、BGP プロトコルを使用するように設定されたネットワークルーターです。ルーターがロードバランサーの IP アドレスのトラフィックを受信すると、ルーターは IP アドレスをアドバタイズした **speaker** Pod が含まれるノードの 1 つを選択します。ルーターはトラフィックをそのノードに送信します。トラフィックがノードに入ると、CNI ネットワークプラグインのサービスプロキシーはトラフィックをサービスのすべての Pod に配信します。

クラスターノードと同じレイヤー 2 のネットワークセグメントに直接接続されたルーターは、BGP ピアとして設定できます。直接接続されたルーターが BGP ピアとして設定されていない場合は、ロードバランサーの IP アドレスのパケットが BGP ピアと**speaker**Pod を実行するクラスターノードの間でルーティングされるようにネットワークを設定する必要があります。

ルーターは、ロードバランサーの IP アドレスの新しいトラフィックを受信するたびに、ノードへの新しい接続を作成します。各ルーターのメーカーには、接続開始ノードを選択する実装固有のアルゴリズムがあります。ただし、アルゴリズムは通常、ネットワーク負荷のバランスをとるために、使用可能なノード間でトラフィックを分散するように設計されています。

ノードが使用できなくなった場合に、ルーターは、ロードバランサーの IP アドレスをアドバタイズする**speaker** Pod が含まれる別のノードとの新しい接続を開始します。

図36.1 BGP モードの MetalLB トポロジーの図



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 IPv4 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 **203.0.113.200** 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 2 および 3 には、アプリケーションの Pod があります。
- **speaker** デーモンセットは、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。MetalLB を設定して、**speaker** Pod を実行するノードを指定できます。
- 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。
- 各**speaker** Pod は、すべての BGP ピアとの BGP セッションを開始し、ロードバランサーの IP アドレスまたは集約されたルートを BGP ピアにアドバタイズします。**speaker** Pod は、Autonomous System 65010 の一部であることをアドバタイズします。この図ではルーター R1 を示しており、これは同じ Autonomous System 内の BGP ピアです。ただし、他の Autonomous System に属するピアとの BGP セッションを開始するように MetalLB を設定できます。
- ノードに、ロードバランサーの IP アドレスをアドバタイズする**speaker** Pod がある場合にはすべて、サービスのトラフィックを受信できます。
  - サービスの外部トラフィックポリシーが **cluster** に設定されている場合、スピーカー Pod が実行されているすべてのノードが **203.0.113.200** ロードバランサーの IP アドレスをアドバタイズし、**speaker** Pod を持つすべてのノードがサービスのトラフィックを受信できま

す。ホストの接頭辞は、外部トラフィックポリシーが cluster に設定されている場合にのみ、ルーターピアにアドバタイズされます。

- サービスの外部トラフィックポリシーが **local** に設定されている場合、**speaker** Pod が実行されているノードとサービスが実行されている少なくとも 1 つのエンドポイントが、**203.0.113.200** ロードバランサーの IP アドレスをアドバタイズできます。これらのノードのみがサービスのトラフィックを受信できます。前の図では、ノード 2 と 3 は**203.0.113.200**をアドバタイズします。

- BGP ピアカスタムリソースの追加時にノードセレクターを指定して、特定の BGP ピアとの BGP セッションを開始する**speaker**Pod を制御するように MetalLB を設定できます。
- BGP を使用するように設定されている R1 などのルーターは、BGP ピアとして設定できます。
- クライアントトラフィックは、ホストネットワーク上のノードの 1 つにルーティングされます。トラフィックがノードに入ると、サービスプロキシーは、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。
- ノードが使用できなくなった場合に、ルーターは障害を検出し、別のノードとの新しい接続を開始します。BGP ピアに双方向フォワーディング検出 (BFD) プロファイルを使用するように MetalLB を設定できます。BFD は、リンク障害検出がより高速であるため、ルーターは BFD がない場合よりも早く新しい接続を開始できます。

36.1.7. 制限および制限

36.1.7.1. 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
- IBM Z® および IBM® LinuxONE
- IBM Z® および IBM® LinuxONE for Red Hat Enterprise Linux (RHEL) KVM
- IBM Power®

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。

36.1.7.2. レイヤー 2 モードの制限

36.1.7.2.1. 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。

36.1.7.2.2. 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ーを更新するまでサービス IP に到達できません。

36.1.7.2.3. 追加ネットワークと MetalLB は同じネットワークを使用できない

MetalLB とソース Pod 上に設定された追加のネットワークインターフェイスの両方に同じ VLAN を使用すると、接続エラーが発生する可能性があります。これは、MetalLB IP とソース Pod が同じノード上に存在する場合に発生します。

接続エラーを回避するには、ソース Pod が存在するサブネットとは異なるサブネットに MetalLB IP を配置します。この設定により、ソース Pod からのトラフィックがデフォルトゲートウェイを経由するようになります。その結果、トラフィックは OVN オーバーレイネットワークを使用して宛先に到達でき、接続が確実に意図したとおりに機能するようになります。

36.1.7.3. BGP モードの制限

36.1.7.3.1. ノードに障害が発生すると、アクティブなすべての接続が切断される可能性があります

MetalLB には、BGP ベースのロードバランシングに共通する制限があります。ノードに障害が発生した場合や **speaker** Pod が再起動した場合など、BGP セッションが中断されると、すべてのアクティブな接続がリセットされる可能性があります。エンドユーザーに、**Connection reset by peer** のメッセージが表示される可能性があります。

BGP セッションが中断された場合にどうなるかは、各ルーターの製造元の実装によります。ただし、**speaker** Pod の数を変更すると、BGP セッションの数に影響し、BGP ピアとのアクティブな接続が切断されることが予想されます。

サービスの中断の可能性を回避または低減するために、BGP ピアの追加時にノードセレクターを指定できます。BGP セッションを開始するノードの数を制限すると、BGP セッションのないノードでの障害が発生しても、サービスへの接続に影響はありません。

36.1.7.3.2. 単一の ASN とルーター ID のみのサポート

BGP ピアカスタムリソースを追加するときは、**spec.myASN**フィールドを指定して、MetalLB が属する Autonomous System Number (ASN) を特定します。OpenShift Container Platform は、MetalLB を使用した BGP の実装を使用しますが、この実装は MetalLB が単一の ASN に所属する必要があります。BGP ピアを追加し、**spec.myASN**に既存の BGP ピアカスタムリソースとは異なる値を指定しようとするとエラーが発生します。

同様に、BGP ピアカスタムリソースを追加する場合には、**spec.router D**フィールドはオプションです。このフィールドに値を指定する場合は、追加する他の BGP ピアカスタムリソースすべてに、同じ値を指定する必要があります。

単一の ASN と単一のルーター ID のサポートに制限がある点が、コミュニティーがサポートする MetalLB の実装との違いです。

36.1.8. 関連情報

- Comparison: Fault tolerant access to external IP addresses
- IP フェイルオーバーの削除
- MetalLB のデプロイメント仕様

36.2. 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。

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。

36.2.1. Web コンソールを使用した OperatorHub からの MetalLB Operator のインストール

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

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

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. キーワードを **Filter by keyword** ボックスに入力するか、目的の Operator までスクロールします。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もしても知られる) で機能する Operator を表示するには、**Disconnected** を選択します。
3. **Install Operator** ページで、デフォルトを受け入れて **Install** をクリックします。

検証

1. インストールが正常に行われたことを確認するには、以下を実行します。
   a. **Operators → Installed Operators** ページに移動します。
   b. Operator が **openshift-operators** の namespace 内に設置されていることと、その状態が **Succeeded** となっていることを確認してください。
2. Operator が正常にインストールされない場合は、Operator のステータスを確認し、ログを確認してください。
   a. **Operators → Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
   b. **Workloads → Pods**ページにナビゲートし、問題を報告している **openshift-operators** プロジェクトの Pod のログを確認します。

36.2.2. 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。OpenShift CLI (**oc**) を使用して、MetalLB Operator をインストールできます。

CLI を使用する場合は、**metallb-system** namespace に Operator をインストールすることを推奨します。

前提条件

- 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. 次のコマンドを入力して、MetalLB Operator の namespace を作成します。

```
$ cat << EOF | oc apply -f -
apiVersion: v1
kind: Namespace
metadata:
  name: metallb-system
EOF
```

2. 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
EOF
```

3. 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
```

4. **Subscription**CR を作成します。

   a. **Subscription** CR を定義し、YAML ファイルを保存します (例: **metallb-sub.yaml**)。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: metallb-operator-sub
  namespace: metallb-system
spec:
  channel: stable
  name: metallb-operator
  source: redhat-operators 1
  sourceNamespace: openshift-marketplace
```

   1 **redhat-operators** 値を指定する必要があります。

   b. **Subscription**CR を作成するには、次のコマンドを実行します。

```
$ oc create -f metallb-sub.yaml
```

5. オプション: BGP および BFD メトリックが Prometheus に表示されるようにするには、次のコマンドのように namespace にラベルを付けることができます。

```
$ oc label ns metallb-system "openshift.io/cluster-monitoring=true"
```

検証

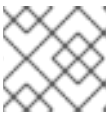
検証手順は、MetalLB Operator が **metallb-system** namespace にインストールされていることを前提としています。

1. 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.4.15.0-nnnnnnnnnnnn   Automatic   true
```



注記

Operator のインストールには数秒かかる場合があります。

2. 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.4.15.0-nnnnnnnnnnnn   Succeeded
```

36.2.3. クラスターでの 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 はクラスターで 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

この手順は、MetallB Operator が **metallb-system** namespace にインストールされていることを前提としています。Web コンソールを使用してインストールした場合は、namespace の代わりに **openshift-operators** を使用してください。

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検証します。

```
$ 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されていることを検証します。

```
$ 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。

36.2.4. MetalLB のデプロイメント仕様

MetalLB カスタムリソースを使用して MetalLB のインスタンスを起動すると、**MetalLB** カスタムリソースでデプロイメント仕様を設定して、**controller** または **speaker** Pod がクラスターにデプロイし、実行する方法を管理できます。これらのデプロイメント仕様を使用して、以下のタスクを管理します。

- MetalLB Pod デプロイメントのノードの選択
- Pod の優先順位および Pod のアフィニティーを使用してたケジューリングの管理
- MetalLB Pod の CPU 制限の割り当て
- MetalLB Pod のコンテナー RuntimeClass の割り当て
- MetalLB Pod のメタデータの割り当て

36.2.4.1. speaker Pod の特定のノードへの限定

デフォルトでは、MetalLB Operator を使用して MetalLB を開始すると、Operator はクラスター内の各ノードで**speaker**Pod のインスタンスを開始します。ロードバランサーの IP アドレスをアドバタイズできるのは、**speaker** Pod を備えたノードのみです。ノードセレクターを使用して **MetalLB**カスタムリソースを設定し、**speaker**Pod を実行するノードを指定できます。

speaker Pod を特定のノードに制限する最も一般的な理由として、特定のネットワークにネットワークインターフェイスがあるノードのみがロードバランサーの IP アドレスをアドバタイズするようにすることが挙げられます。ロードバランサーの IP アドレスの宛先として、**speaker** Pod が実行されているノードのみがアドバタイズされます。

speaker Pod を特定のノードに制限し、サービスの外部トラフィックポリシーに**ローカル** を指定する場合は、サービスのアプリケーション Pod が同じノードにデプロイされていることを確認する必要があります。

speaker Pod をワーカーノードに制限する設定例

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  nodeSelector:  <.>
    node-role.kubernetes.io/worker: ""
  speakerTolerations:   <.>
```

```
  - key: "Example"
    operator: "Exists"
    effect: "NoExecute"
```

<.> 設定例では、スピーカー Pod をワーカーノードに割り当てるように指定していますが、ノードまたは任意の有効なノードセレクターに割り当てたラベルを指定できます。<.> この設定例では、この容認がアタッチされている Pod は、**operator** を使用して **キー** 値と **effect** 値に一致するテイントを容認します。

spec.nodeSelectorフィールドを使用してマニフェストを適用した後に、**oc get daemonset -n metallb-systemspeaker** コマンドを使用して Operator がデプロイした Pod の数を確認できます。同様に、**oc get node -l node-role.kubernetes.io/worker =**のようなコマンドを使用して、ラベルに一致するノードを表示できます。

オプションで、アフィニティールールを使用して、ノードがどの speaker Pod をスケジュールするか、スケジュールしないかを制御することができます。また、容認の一覧を適用してこれらの Pod を制限することもできます。アフィニティールール、テイント、および容認の詳細は、追加のリソースを参照してください。

36.2.4.2. MetalLB デプロイメントでのコンテナーランタイムクラスの設定

必要に応じて、MetalLB カスタムリソースを設定して、コンテナーランタイムクラスを **controller** Pod および **speaker** Pod に割り当てることができます。たとえば、Windows ワークロードでは、Windows ランタイムクラスを Pod に割り当てることができ、Pod のすべてのコンテナーに対してこのランタイムクラスが使用されるようになります。

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。
- MetalLB Operator がインストールされている。

手順

1. **myRuntimeClass.yaml** などの **RuntimeClass** カスタムリソースを作成して、ランタイムクラスを定義します。

```
apiVersion: node.k8s.io/v1
kind: RuntimeClass
metadata:
  name: myclass
handler: myconfiguration
```

2. **RuntimeClass** カスタムリソース設定を適用します。

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

3. **MetalLBRuntime.yaml** などの **MetalLB** カスタムリソースを作成して、**runtimeClassName** 値を指定します。

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
```

```
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    runtimeClassName: myclass
    annotations: 1
      controller: demo
  speakerConfig:
    runtimeClassName: myclass
    annotations: 2
      speaker: demo
```

1 2 この例では、**アノテーション** を使用してビルドリリース情報や GitHub プルリクエスト情報などのメタデータを追加します。ラベルで許可されていない文字をアノテーションに入力できます。ただし、アノテーションを使用してオブジェクトを特定または選択することはできません。

4. **MetalLB** カスタムリソース設定を適用します。

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

検証

- Pod のコンテナーランタイムを表示するには、以下のコマンドを実行します。

```
$ oc get pod -o custom-
columns=NAME:metadata.name,STATUS:.status.phase,RUNTIME_CLASS:.spec.runtimeClass
Name
```

36.2.4.3. MetalLB デプロイメントでの Pod の優先順位および Pod アフィニティーの設定

オプションで、**MetalLB** カスタムリソースを設定して、Pod の優先順位と Pod のアフィニティールールを **controller** Pod および **speaker** Pod に割り当てることができます。Pod の優先順位は、ノード上の Pod の相対的な重要度を示し、この優先順位に基づいて Pod をスケジュールします。**controller** または **speaker** Pod に高い優先順位を設定して、ノード上の他の Pod よりも優先的にスケジューリングされるようにします。

Pod のアフィニティーは Pod 間の関係を管理します。Pod のアフィニティーを **controller** または **speaker** Pod に割り当て、スケジューラーが Pod 関係のコンテキストで Pod を配置するノードを制御します。たとえば、Pod アフィニティールールを使用して、複数の特定 Pod を同じノードまたは別のノードに配置するようにできます。これにより、ネットワーク通信が改善され、これらのコンポーネント間の遅延が縮小されます。

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。
- MetalLB Operator がインストールされている。
- クラスター上で MetalLB Operator を開始している。

手順

1. **myPriorityClass.yaml** などの **PriorityClass** カスタムリソースを作成して、優先度レベルを設定します。この例では、**high-priority** という名前の **PriorityClass** を、値 **1000000** で定義します。この優先クラスが割り当てられた Pod は、スケジューリングにおいて、それより低い優先クラスの Pod より優先順位が高いとみなされます。

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
```

2. **PriorityClass** カスタムリソース設定を適用します。

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

3. **MetalLBPodConfig.yaml** などの **MetalLB** カスタムリソースを作成して、**priorityClassName** と **podAffinity** の値を指定します。

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
    priorityClassName: high-priority (1)
    affinity:
      podAffinity: (2)
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: metallb
          topologyKey: kubernetes.io/hostname
  speakerConfig:
    priorityClassName: high-priority
    affinity:
      podAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: metallb
          topologyKey: kubernetes.io/hostname
```

(1) MetalLB コントローラー Pod の優先クラスを指定します。この場合、**high-priority** に設定されます。

(2) Pod アフィニティールールを設定していることを指定します。これらのルールは、他の Pod またはノードに関連して Pod がどのようにスケジュールされるかを決定します。この設定は、**app: metallb** ラベルを持つ Pod を同じホスト名を共有するノードにスケジュールするようにスケジューラーに指示します。これは、MetalLB 関連の Pod を同じノード上に配置するのに役立ち、これらの Pod 間のネットワーク通信、遅延、リソース使用量を最適化できる可能性があります。

4. **MetalLB** カスタムリソース設定を適用します。

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

検証

- **metallb-system** namespace の Pod に割り当てた優先クラスを表示するには、次のコマンドを実行します。

```
$ oc get pods -n metallb-system -o custom-
columns=NAME:.metadata.name,PRIORITY:.spec.priorityClassName
```

出力例

```
NAME                                          PRIORITY
controller-584f5c8cd8-5zbvg                   high-priority
metallb-operator-controller-manager-9c8d9985-szkqg   <none>
metallb-operator-webhook-server-c895594d4-shjgx      <none>
speaker-dddf7                                 high-priority
```

- スケジューラーが Pod アフィニティールールに従って Pod を配置したことを確認するには、次のコマンドを実行して Pod のノードのメタデータを表示します。

```
$ oc get pod -o=custom-columns=NODE:.spec.nodeName,NAME:.metadata.name -n
metallb-system
```

36.2.4.4. MetalLB デプロイメントでの Pod CPU 制限の設定

オプションで、**MetalLB** カスタムリソースを設定することで、Pod の CPU 制限を **controller** Pod と **speaker** Pod に割り当てることができます。**controller** Pod または **speaker** Pod の CPU 制限を定義すると、ノード上のコンピュートリソースを管理するのに役立ちます。これにより、ノード上のすべての Pod に、ワークロードとクラスターのハウスキーピングを管理するために必要なコンピューティングリソースが確保されます。

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。
- MetalLB Operator がインストールされている。

手順

1. **CPULimits.yaml** などの **MetalLB** カスタムリソースファイルを作成し、**コントローラー** および **speaker** Pod の **cpu** 値を指定します。

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  controllerConfig:
```

```
    resources:
      limits:
        cpu: "200m"
  speakerConfig:
    resources:
      limits:
        cpu: "300m"
```

2. **MetalLB** カスタムリソース設定を適用します。

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

検証

- Pod のコンピューティングリソースを表示するには、次のコマンドを実行し、**<pod_name>** をターゲット Pod に置き換えます。

```
$ oc describe pod <pod_name>
```

36.2.5. 関連情報

- ノードセレクターの使用による特定ノードへの Pod の配置
- テイントおよび容認 (Toleration) について
- Pod の優先順位について
- Pod のアフィニティーについて

36.2.6. 次のステップ

- MetalLB アドレスプールの設定

36.3. METALLB のアップグレード

現在、MetalLB Operator のバージョン 4.10 以前のバージョンを実行している場合、4.10 以降のバージョンへの自動更新は機能しないことに注意してください。4.11 以降の任意のバージョンの MetalLB Operator から新しいバージョンへのアップグレードは成功します。たとえば、バージョン 4.12 からバージョン 4.13 へのアップグレードはスムーズに行われます。

4.10 以前からの MetalLB Operator のアップグレード手順の概要は次のとおりです。

1. インストールされている MetalLB Operator バージョン (4.10 など) を削除します。namespace と **metallb** カスタムリソースが削除されていないことを確認してください。
2. CLI を使用して、以前のバージョンの MetalLB Operator がインストールされていた namespace に MetalLB Operator 4.15 をインストールします。

注記

この手順は、標準の簡単な方法に従う MetalLB Operator の自動 z ストリーム更新には適用されません。

MetalLB Operator を 4.10 以前からアップグレードする詳細な手順については、次のガイダンスを参照してください。クラスター管理者は、OpenShift CLI (**oc**) または Web コンソールを使用して MetalLB Operator を削除し、アップグレードプロセスを開始します。

36.3.1. Web コンソールを使用してクラスターから MetalLB Operator を削除

クラスター管理者は Web コンソールを使用して、選択した namespace からインストールされた Operator を削除できます。

前提条件

- **cluster-admin** 権限を持つアカウントを使用して OpenShift Container Platform クラスター Web コンソールにアクセスできる。

手順

1. **Operators → Installed Operators** ページに移動します。
2. MetalLB Operator を検索します。次に、それをクリックします。
3. **Operator Details** ページの右側で、**Actions** ドロップダウンメニューから **Uninstall Operator** を選択します。
   Uninstall Operator? ダイアログボックスが表示されます。
4. **Uninstall** を選択し、Operator、Operator デプロイメント、および Pod を削除します。このアクションの後には、Operator は実行を停止し、更新を受信しなくなります。

注記

このアクションは、カスタムリソース定義 (CRD) およびカスタムリソース (CR) など、Operator が管理するリソースは削除されません。Web コンソールおよび継続して実行されるクラスター外のリソースによって有効にされるダッシュボードおよびナビゲーションアイテムには、手動でのクリーンアップが必要になる場合があります。Operator のアンインストール後にこれらを削除するには、Operator CRD を手動で削除する必要があります。

36.3.2. CLI を使用してクラスターから MetalLB Operator を削除

クラスター管理者は CLI を使用して、選択した namespace からインストールされた Operator を削除できます。

前提条件

- **cluster-admin** 権限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。
- **oc** コマンドがワークステーションにインストールされていること。

手順

1. **currentCSV** フィールドでサブスクライブされた MetalLB Operator の現在のバージョンを確認します。

```
$ oc get subscription metallb-operator -n metallb-system -o yaml | grep currentCSV
```

出力例

```
currentCSV: metallb-operator.4.10.0-202207051316
```

2. サブスクリプションを削除します。

```
$ oc delete subscription metallb-operator -n metallb-system
```

出力例

```
subscription.operators.coreos.com "metallb-operator" deleted
```

3. 直前の手順で **currentCSV** 値を使用し、ターゲット namespace の Operator の CSV を削除します。

```
$ oc delete clusterserviceversion metallb-operator.4.10.0-202207051316 -n metallb-system
```

出力例

```
clusterserviceversion.operators.coreos.com "metallb-operator.4.10.0-202207051316" deleted
```

36.3.3. MetalLB Operator Operator グループの編集

4.10 以前の MetalLB Operator バージョンから 4.11 以降にアップグレードする場合は、Operator グループのカスタムリソース (CR) から **spec.targetNamespaces** を削除します。MetalLB Operator の削除に Web コンソールや CLI を使用したかにかかわらず、仕様を削除する必要があります。

注記

MetalLB Operator バージョン 4.11 以降は **AllNamespaces** インストールモードのみをサポートしますが、4.10 以前のバージョンは **OwnNamespace** モードまたは **SingleNamespace** モードをサポートします。

前提条件

- **cluster-admin** 権限を使用して OpenShift Container Platform 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. 次のコマンドを実行して、**metallb-system** namespace 内の Operator グループをリスト表示します。

```
$ oc get operatorgroup -n metallb-system
```

出力例

```
NAME                 AGE
metallb-system-7jc66   85m
```

2. 次のコマンドを実行して、**metallb-system** namespace に関連付けられた Operator グループ CR に **spec.targetNamespaces** が存在することを確認します。

```
$ oc get operatorgroup metallb-system-7jc66 -n metallb-system -o yaml
```

出力例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: ""
  creationTimestamp: "2023-10-25T09:42:49Z"
  generateName: metallb-system-
  generation: 1
  name: metallb-system-7jc66
  namespace: metallb-system
  resourceVersion: "25027"
  uid: f5f644a0-eef8-4e31-a306-e2bbcfaffab3
spec:
  targetNamespaces:
  - metallb-system
  upgradeStrategy: Default
status:
  lastUpdated: "2023-10-25T09:42:49Z"
  namespaces:
  - metallb-system
```

3. 次のコマンドを実行して、Operator グループを編集し、**spec** セクション配下の **targetNamespaces** と **metallb-system** を削除します。

```
$ oc edit n metallb-system
```

出力例

```
operatorgroup.operators.coreos.com/metallb-system-7jc66 edited
```

4. 次のコマンドを実行して、**metallb-system** namespace に関連付けられた Operator グループのカスタムリソースから **spec.targetNamespaces** が削除されていることを確認します。

```
$ oc get operatorgroup metallb-system-7jc66 -n metallb-system -o yaml
```

出力例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  annotations:
    olm.providedAPIs: ""
  creationTimestamp: "2023-10-25T09:42:49Z"
  generateName: metallb-system-
  generation: 2
  name: metallb-system-7jc66
```

```
  namespace: metallb-system
  resourceVersion: "61658"
  uid: f5f644a0-eef8-4e31-a306-e2bbcfaffab3
spec:
  upgradeStrategy: Default
status:
  lastUpdated: "2023-10-25T14:31:30Z"
  namespaces:
  - ""
```

36.3.4. MetalLB Operator のアップグレード

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します。

手順

1. **metallb-system** namespace がまだ存在することを確認します。

```
$ oc get namespaces | grep metallb-system
```

出力例

```
metallb-system                         Active   31m
```

2. **metallb** カスタムリソースがまだ存在することを確認します。

```
$ oc get metallb -n metallb-system
```

出力例

```
NAME      AGE
metallb   33m
```

3. 「CLI を使用して OperatorHub からインストールする」に記載されたガイダンスに従い、MetalLB Operator の最新の 4.15 バージョンをインストールします。

注記

MetalLB Operator の最新の 4.15 バージョンをインストールする場合、以前にインストールしたのと同じ namespace に Operator をインストールする必要があります。

4. アップグレード後の Operator バージョンが 4.15 であることを確認します。

```
$ oc get csv -n metallb-system
```

出力例

```
NAME                                         DISPLAY          VERSION               REPLACES   PHASE
metallb-operator.4.15.0-202207051316   MetalLB Operator   4.15.0-202207051316
Succeeded
```

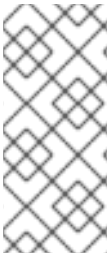
36.3.5. 関連情報

- クラスターからの Operator の削除
- MetalLB Operator のインストール

36.4. 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。例で使用されている namespace は、namespace が **metallb-system** であることを前提としています。

36.4.1. IPAddressPool カスタムリソースについて



注記

OpenShift Container Platform 4.10 の 「MetalLB を使用した負荷分散」に記載されているアドレスプールカスタムリソース定義 (CRD) および API は、4.15 でも引き続き使用できます。ただし、**AddressPool** CRD を使用する場合、レイヤ 2 プロトコルまたは BGP プロトコルを使用した **IPAddressPool** からの IP アドレスのアドバタイズに関連する拡張機能はサポートされません。

次の表では、**IPAddressPool** カスタムリソースのフィールドについて説明します。

表36.1 MetalLB IPAddressPool プールのカスタムリソース

フィールド	型	説明
metadata.name	**string**	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、**metallb.universe.tf/address-pool** 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、**doc-example**、**silver**、および **gold** の名前が使用されます。
metadata.namespace	**string**	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。
metadata.label	**string**	オプション: **IPAddressPool** に割り当てられたキーと値のペアを指定します。これは、**BGPAdvertisement** および **L2Advertisement** CRD の **ipAddressPoolSelectors** によって参照され、**IPAddressPool** をアドバタイズメントに関連付けることができます。

フィールド	型	説明
spec.addresses	**string**	MetalLB Operator がサービスに割り当てる IP アドレスのリストを指定します。1 つのプールで複数の範囲を指定できます。それらはすべて同じ設定を共有します。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。
spec.autoAssign	**boolean**	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。**metallb.universe.tf/address-pool** 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、**false** を指定します。デフォルト値は **true** です。
spec.avoidBuggyIPs	**boolean**	オプション: これを有効にすると、.0 および .255 で終わる IP アドレスがプールから割り当てられなくなります。デフォルト値は **false** です。一部の古い消費者ネットワーク機器は、.0 および .255 で終わる IP アドレスを誤ってブロックします。

spec.serviceAllocation 仕様を設定することにより、**IPAddressPool** からサービスおよび namespace に IP アドレスを割り当てることができます。

表36.2 MetalLB IPAddressPool カスタムリソース spec.serviceAllocation サブフィールド

フィールド	型	説明
priority	**int**	オプション: 複数の IP アドレスプールがサービスまたは namespace に一致する場合の IP アドレスプール間の優先度を定義します。数字が小さいほど優先度が高いことを示します。
namespace	**array (string)**	オプション: IP アドレスプール内の IP アドレスに割り当てることができる namespace のリストを指定します。
namespaceSelectors	**配列 (LabelSelector)**	オプション: リスト形式のラベルセレクターを使用して、IP アドレスプールから IP アドレスに割り当てることができる namespace ラベルを指定します。
serviceSelectors	**配列 (LabelSelector)**	オプション: リスト形式のラベルセレクターを使用して、アドレスプールから IP アドレスに割り当てることができるサービスラベルを指定します。

36.4.2. 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のできる 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、**ipaddresspool.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example
  labels: 1
    zone: east
spec:
  addresses:
  - 203.0.113.1-203.0.113.10
  - 203.0.113.65-203.0.113.75
```

1 **IPAddressPool** に割り当てられたこのラベルは、**BGPAdvertisement** CRD の **ipAddressPoolSelectors** によって参照され、**IPAddressPool** をアドバタイズメントに関連付けることができます。

2. IP アドレスプールの設定を適用します。

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

検証

- アドレスプールを表示します。

```
$ oc describe -n metallb-system IPAddressPool doc-example
```

出力例

```
Name:         doc-example
Namespace:    metallb-system
Labels:       zone=east
Annotations:  <none>
API Version:  metallb.io/v1beta1
Kind:         IPAddressPool
Metadata:
  ...
Spec:
  Addresses:
    203.0.113.1-203.0.113.10
    203.0.113.65-203.0.113.75
  Auto Assign:  true
Events:         <none>
```

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。

36.4.3. アドレスプールの設定例

36.4.3.1. 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て下層と上限を分けることができます。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-cidr
  namespace: metallb-system
spec:
  addresses:
  - 192.168.100.0/24
  - 192.168.200.0/24
  - 192.168.255.1-192.168.255.5
```

36.4.3.2. 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、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: IPAddressPool
metadata:
  name: doc-example-reserved
  namespace: metallb-system
spec:
  addresses:
  - 10.0.100.0/28
  autoAssign: false
```

36.4.3.3. 例: IPv4 および IPv6 アドレス

IPv4 および IPv6 を使用するアドレスプールを追加できます。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。

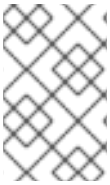
サービスに、単一の IPv4 アドレス、単一の IPv6 アドレス、またはその両方を割り当てるかどうかは、サービスの追加方法によって決まります。**spec.ip Families**フィールドと**spec.ip Family Policy**フィールドでは、IP アドレスをサービスに割り当てる方法を制御します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-combined
  namespace: metallb-system
spec:
  addresses:
  - 10.0.100.0/28
  - 2002:2:2::1-2002:2:2::100
```

36.4.3.4. 例: IP アドレスプールをサービスまたは namespace に割り当てる

IPAddressPool から指定したサービスと namespace に IP アドレスを割り当てることができます。

サービスまたは namespace を複数の IP アドレスプールに割り当てる場合、MetalLB は優先度の高い IP アドレスプールから使用可能な IP アドレスを使用します。割り当てられた優先度の高い IP アドレスプールから使用可能な IP アドレスがない場合、MetalLB は、優先度の低い、または優先度のない IP アドレスプールから使用可能な IP アドレスを使用します。



注記

namespaceSelectors と **serviceSelectors** の仕様には、**matchLabels** ラベルセレクター、**matchExpressions** ラベルセレクター、またはその両方を使用できます。この例は、仕様ごとに 1 つのラベルセレクターを示しています。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: doc-example-service-allocation
  namespace: metallb-system
spec:
  addresses:
    - 192.168.20.0/24
  serviceAllocation:
    priority: 50 1
    namespaces: 2
      - namespace-a
      - namespace-b
    namespaceSelectors: 3
      - matchLabels:
          zone: east
    serviceSelectors: 4
      - matchExpressions:
        - key: security
          operator: In
          values:
          - S1
```

1 アドレスプールに優先度を割り当てます。数字が小さいほど優先度が高いことを示します。

2 1 つ以上の namespace をリスト形式で IP アドレスプールに割り当てます。

3 リスト形式のラベルセレクターを使用して、1 つ以上の namespace ラベルを IP アドレスプールに割り当てます。

4 リスト形式のラベルセレクターを使用して、1 つ以上のサービスラベルを IP アドレスプールに割り当てます。

36.4.4. 関連情報

- L2 アドバタイズメントとラベルを使用した MetalLB の設定

36.4.5. 次のステップ

- BGP モードについては、 MetalLBBGP ピアの設定を参照してください。
- MetalLB を使用するためのサービスの設定

36.5. IP アドレスプールのアドバタイズメントについて

IP アドレスがレイヤー 2 プロトコル、BGP プロトコル、またはその両方でアドバタイズされるように MetalLB を設定できます。レイヤー 2 では、MetalLB ではフォールトトレラントな外部 IP アドレスを使用できます。BGP を使用すると、MetalLB で外部 IP アドレスに対するフォールトトレランス機能および負荷分散が提供されます。

MetalLB は、同じ IP アドレスのセットに対して L2 と BGP を使用したアドバタイズをサポートします。

MetalLB は、ネットワーク上のノードのサブセットに対して、特定の BGP ピアにアドレスプールを効果的に割り当てる柔軟性を提供します。これにより、たとえばノードの分離やネットワークのセグメンテーションを容易にするなど、より複雑な設定が可能になります。

36.5.1. BGPAdvertisement カスタムリソースについて

bgp Advertisements オブジェクトのフィールドは、次の表に定義されています。

表36.3 BGPAdvertisements の設定

フィールド	型	説明
metadata.name	**string**	BGP アドバタイズメントの名前を指定します。
metadata.namespace	**string**	BGP アドバタイズメントの 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。
spec.aggregationLength	**integer**	オプション: 32 ビット CIDR マスクに含めるビット数を指定します。マスクが複数のサービス IP アドレスのルートに適用され、speaker は集約されたルートをアドバタイズし、speaker が BGP ピアにアドバタイズするルートを集約します。たとえば、集約の長さが **24**の場合は、speaker は複数の**10.0.1.x/32**サービス IP アドレスを集約して、**10.0.1.0/24**ルートを 1 つアドバタイズできます。
spec.aggregationLengthV6	**integer**	オプション: 128 ビット CIDR マスクに含めるビット数を指定します。たとえば、集約の長さが **124**の場合は、speaker は複数の **fc00:f853:0ccd:e799::x/128** サービス IP アドレスを集約して、**fc00:f853:0ccd:e799::0/124** ルートを 1 つアドバタイズできます。

フィールド	型	説明
spec.communities	**string**	オプション:1 つ以上の BGP コミュニティーを指定します。各コミュニティーは、16 ビット値 2 つをコロン文字で区切って指定します。一般的なコミュニティーは、16 ビット値として指定する必要があります。 • **NO_EXPORT**: **65535:65281** • **NO_ADVERTISE**: **65535:65282** • **NO_EXPORT_SUBCONFED**: **65535:65283** **注記** 文字列とともに作成されたコミュニティーオブジェクトを使用することもできます。
spec.localPref	**integer**	オプション: このアドバタイズメントのローカル設定を指定します。この BGP 属性は、Autonomous System 内の BGP セッションに適用されます。
spec.ipAddressPools	**string**	オプション: 名前で選択された、このアドバタイズメントでアドバタイズする **IPAddressPools** のリスト。
spec.ipAddressPoolSelectors	**string**	オプション: このアドバタイズメントでアドバタイズされる **IPAddressPools** のセレクター。これは、名前自体ではなく、**IPAddressPool** に割り当てられたラベルに基づいて **IPAddressPool** をアドバタイズメントに関連付けるためのものです。これやリストで **IPAddressPool** が選択されていない場合、アドバタイズメントはすべての **IPAddressPools** に適用されます。
spec.nodeSelectors	**string**	オプション: **NodeSelectors** を使用すると、ロードバランサー IP のネクストホップとしてアナウンスするノードを制限できます。空の場合、すべてのノードがネクストホップとしてアナウンスされます。
spec.peers	**string**	オプション: ピアは、選択したプールの IP をアドバタイズする BGP ピアを制限します。空の場合、ロードバランサー IP は設定されているすべての BGP ピアにアナウンスされます。

36.5.2. BGP アドバタイズメントと基本的なユースケースを使用する MetalLB の設定

MetalLB を次のとおり設定し、ピア BGP ルーターが、MetalLB がサービスに割り当てるロードバランサー IP アドレスごとに、**203.0.113.200/32** ルート 1 つ、**fc00:f853:ccd:e799::1/128** ルート 1 つを受信するようにします。**local Pref**および**communities**フィールドが指定されていないため、ルートは**local**

Prefをゼロに設定して BGP コミュニティーなしでアドバタイズされます。

36.5.2.1. 例: BGP を使用する基本的なアドレスプール設定のアドバタイズメント

IPAddressPool が BGP プロトコルでアドバタイズされるように、MetalLB を次のように設定します。

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. IP アドレスプールを作成します。
   a. 以下の例のような内容で、**ipaddresspool.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-basic
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
```

   b. IP アドレスプールの設定を適用します。

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

2. BGP アドバタイズメントを作成します。
   a. 以下の例のような内容で、**bgpadvertisement.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-basic
  namespace: metallb-system
spec:
  ipAddressPools:
  - doc-example-bgp-basic
```

   b. 設定を適用します。

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

36.5.3. BGP アドバタイズメントと高度なユースケースを使用する MetalLB の設定

MetalLB を次のように設定し、MetalLB が **203.0.113.200** と **203.0.113.203**、**fc00:f853:ccd:e799::0** と **fc00:f853:ccd:e799::f** の範囲の IP アドレスを割り当てるようにします。

MetalLB が **203.0.113.200**の IP アドレスをサービスに割り当てる例について見ていき、これら 2 つの BGP アドバタイズメントを説明します。この IP アドレスを例にとると、speaker は 2 つのルートを BGP ピアにアドバタイズします。

- **localPref** が **100** に、コミュニティーが **NO_ADVERTISE** コミュニティーの数値に設定されている **203.0.113.200/32**。この仕様は、ピアルーターにこのルートを使用できることを指定していますが、このルートに関する情報を BGP ピアに伝播しないようにします。
- MetalLB で割り当てられたロードバランサーの IP アドレスを 1 つのルートに集約する **203.0.113.200/30**。MetalLB は、コミュニティー属性が **8000:800**に設定された BGP ピアに集約ルートをアドバタイズします。BGP ピアは、 **203.0.113.200/30**ルートを他の BGP ピアに伝播します。トラフィックが speaker のあるノードにルーティングされる場合には、**203.0.113.200/32**ルートを使用して、トラフィックがクラスターに転送され、サービスに関連付けられている Pod に転送されます。

さらにサービスを追加し、MetalLB でプールからより多くのロードバランサー IP アドレスを割り当てると、ピアルーターはサービスごとにローカルルート**203.0.113.20x/32** を 1 つと、 **203.0.113.200/30** 集約ルートを受け取ります。追加する各サービスは/**30**ルートを生成しますが、MetalLB は、ピアルーターと通信する前に、ルートの重複を排除して 1 つの BGP アドバタイズにします。

36.5.3.1. 例: BGP を使用する高度なアドレスプール設定のアドバタイズメント

IPAddressPool が BGP プロトコルでアドバタイズされるように、MetalLB を次のように設定します。

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. IP アドレスプールを作成します。
   a. 以下の例のような内容で、**ipaddresspool.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-adv
  labels:
    zone: east
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
  autoAssign: false
```

   b. IP アドレスプールの設定を適用します。

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

2. BGP アドバタイズメントを作成します。

a. 以下の例のような内容で、**bgpadvertisement1.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 65535:65282
  aggregationLength: 32
  localPref: 100
```

b. 設定を適用します。

```
$ oc apply -f bgpadvertisement1.yaml
```

c. 以下の例のような内容で、**bgpadvertisement2.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-adv-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - doc-example-bgp-adv
  communities:
    - 8000:800
  aggregationLength: 30
  aggregationLengthV6: 124
```

d. 設定を適用します。

```
$ oc apply -f bgpadvertisement2.yaml
```

36.5.4. ノードのサブセットからの IP アドレスプールのアドバタイズ

特定のノードセットのみから IP アドレスプールから IP アドレスをアドバタイズするには、BGPAdvertisement カスタムリソースで **.spec.nodeSelector** 仕様を使用します。この仕様は、IP アドレスのプールをクラスター内の一連のノードに関連付けます。これは、クラスター内の異なるサブネット上にノードがあり、特定のサブネット (パブリックに面したサブネットのみなど) のアドレスプールから IP アドレスをアドバタイズしたい場合に役立ちます。

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. カスタムリソースを使用して IP アドレスプールを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400
```

2. BGPAdvertisement カスタムリソースで **.spec.nodeSelector** 値を定義することにより、**pool1** からの IP アドレスがアドバタイズするクラスター内のノードを制御します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: example
spec:
  ipAddressPools:
  - pool1
  nodeSelector:
  - matchLabels:
      kubernetes.io/hostname: NodeA
  - matchLabels:
      kubernetes.io/hostname: NodeB
```

この例では、pool1 の IP アドレスは **NodeA** と **NodeB** からの **み** アドバタイズします。

36.5.5. L2Advertisement カスタムリソースについて

l2Advertisements オブジェクトのフィールドは、次の表に定義されています。

表36.4 L2 アドバタイズメント設定

フィールド	型	説明
metadata.name	**string**	L2 アドバタイズメントの名前を指定します。
metadata.namespace	**string**	L2 アドバタイズメントの 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。
spec.ipAddressPools	**string**	オプション: 名前で選択された、このアドバタイズメントでアドバタイズする **IPAddressPools** のリスト。
spec.ipAddressPoolSelectors	**string**	オプション: このアドバタイズメントでアドバタイズされる **IPAddressPools** のセレクター。これは、名前自体ではなく、**IPAddressPool** に割り当てられたラベルに基づいて **IPAddressPool** をアドバタイズメントに関連付けるためのものです。これやリストで **IPAddressPool** が選択されていない場合、アドバタイズメントはすべての **IPAddressPools** に適用されます。

フィールド	型	説明
spec.nodeSelectors	**string**	オプション: **NodeSelectors** は、ロードバランサー IP のネクストホップとしてアナウンスするノードを制限します。空の場合、すべてのノードがネクストホップとしてアナウンスされます。  **重要** ネクストホップとしてアナウンスするノードの制限は、テクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。
spec.interfaces	**string**	オプション: ロードバランサー IP をアナウンスするために使用される **interfaces** のリスト。

36.5.6. L2 アドバタイズメントを使用した MetalLB の設定

IPAddressPool が L2 プロトコルでアドバタイズされるように、MetalLB を次のように設定します。

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. IP アドレスプールを作成します。
   a. 以下の例のような内容で、**ipaddresspool.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
  addresses:
    - 4.4.4.0/24
  autoAssign: false
```

b. IP アドレスプールの設定を適用します。

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

2. L2 アドバタイズメントを作成します。

a. 以下の例のような内容で、**l2advertisement.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
   - doc-example-l2
```

b. 設定を適用します。

```
$ oc apply -f l2advertisement.yaml
```

36.5.7. L2 アドバタイズメントとラベルを使用した MetalLB の設定

BGPAdvertisement および **L2Advertisement** カスタムリソース定義の **ipAddressPoolSelectors** フィールドは、名前自体ではなく、**IPAddressPool** に割り当てられたラベルに基づいて **IPAddressPool** をアドバタイズメントに関連付けるために使用されます。

この例は、**ipAddressPoolSelectors** フィールドを設定することにより、**IPAddressPool** が L2 プロトコルでアドバタイズされるように MetalLB を設定する方法を示しています。

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. IP アドレスプールを作成します。

a. 以下の例のような内容で、**ipaddresspool.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2-label
  labels:
    zone: east
spec:
  addresses:
    - 172.31.249.87/32
```

b. IP アドレスプールの設定を適用します。

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

2. **ipAddressPoolSelectors** を使用して IP をアドバタイズする L2 アドバタイズメントを作成します。

   a. 以下の例のような内容で、**l2advertisement.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement-label
  namespace: metallb-system
spec:
  ipAddressPoolSelectors:
    - matchExpressions:
        - key: zone
          operator: In
          values:
            - east
```

   b. 設定を適用します。

```
$ oc apply -f l2advertisement.yaml
```

36.5.8. 選択したインターフェイスの L2 アドバタイズを使用した MetalLB の設定

デフォルトでは、サービスに割り当てられた IP アドレスプールの IP アドレスが、すべてのネットワークインターフェイスからアドバタイズされます。**L2Advertisement** カスタムリソース定義の **interfaces** フィールドは、IP アドレスプールをアドバタイズするネットワークインターフェイスを制限するために使用されます。

この例では、すべてのノードの **interfaces** フィールドにリストされているネットワークインターフェイスからのみ IP アドレスプールがアドバタイズされるように、MetalLB を設定する方法を示します。

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. IP アドレスプールを作成します。

   a. **ipaddresspool.yaml** などのファイルを作成し、次の例のように設定の詳細を入力します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-l2
spec:
```

```
  addresses:
    - 4.4.4.0/24
  autoAssign: false
```

b. 次の例のように、IP アドレスプールの設定を適用します。

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

2. **interfaces** セレクターを使用して IP をアドバタイズする L2 アドバタイズメントを作成します。

a. **l2advertisement.yaml** などの YAML ファイルを作成し、次の例のように設定の詳細を入力します。

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
   - doc-example-l2
   interfaces:
   - interfaceA
   - interfaceB
```

b. 次の例のように、広告の設定を適用します。

```
$ oc apply -f l2advertisement.yaml
```

重要

インターフェイスセレクターは、MetalLB が L2 を使用して特定の IP をアナウンスするノードを選択する方法には影響しません。ノードが選択されたインターフェイスを持たない場合、選択されたノードはサービスをアナウンスしません。

36.5.9. 関連情報

- コミュニティーエイリアスの設定

36.6. METALLB BGP ピアの設定

クラスター管理者は、ボーダーゲートウェイプロトコル (BGP) ピアを追加、変更、および削除できます。MetalLB Operator は、BGP ピアカスタムリソースを使用して、MetalLB **speaker** Pod が BGP セッションを開始するために接続するピアを識別します。ピアは、Metall B がサービスに割り当てるロードバランサー IP アドレスのルートアドバタイズメントを受信します。

36.6.1. BGP ピアカスタムリソースについて

次の表で、BGP ピアカスタムリソースのフィールドについて説明します。

表36.5 MetalLB BGP ピアカスタムリソース

フィールド	型	説明
metadata.name	**string**	BGP ピアカスタムリソースの名前を指定します。
metadata.namespace	**string**	BGP ピアカスタムリソースの namespace を指定します。
spec.myASN	**integer**	BGP セッションのローカルエンドの Autonomous System 番号を指定します。追加するすべての BGP ピアカスタムリソースに同じ値を指定します。範囲は **0** から **4294967295** です。
spec.peerASN	**integer**	BGP セッションのリモートエンドの Autonomous System 番号を指定します。範囲は **0** から **4294967295** です。
spec.peerAddress	**string**	BGP セッションを確立するために接続するピアの IP アドレスを指定します。
spec.sourceAddress	**string**	オプション: BGP セッションの確立時に使用する IP アドレスを指定します。値は IPv4 アドレスである必要があります。
spec.peerPort	**integer**	オプション: BGP セッションを確立するために接続するピアのネットワークポートを指定します。範囲は **0** から **16384** です。
spec.holdTime	**string**	オプション: BGP ピアに提案するホールドタイムの期間を指定します。最小値は 3 秒 (**3s**) です。一般的には、 **3s**、**1m** および **5m30s** など、秒および分単位で指定します。パス障害をより迅速に検出するには、BFD も設定します。
spec.keepaliveTime	**string**	オプション: キープアライブメッセージを BGP ピアに送信する間の最大間隔を指定します。このフィールドを指定する場合は、 **holdTime**フィールドの値も指定する必要があります。指定する値は、 **holdTime**フィールドの値よりも小さくする必要があります。
spec.routerID	**string**	オプション: BGP ピアにアドバタイズするルーター ID を指定します。このフィールドを指定する場合は、追加するすべての BGP ピアカスタムリソースに同じ値を指定する必要があります。
spec.password	**string**	オプション: TCP MD5 認証が済んだ BGP セッションを実施するルーターのピアに送信する MD5 パスワードを指定します。
spec.passwordSecret	**string**	オプション: BGP ピアの認証シークレットの名前を指定します。シークレットは **metallb** namespace に存在し、basic-auth タイプである必要があります。
spec.bfdProfile	**string**	オプション: BFD プロファイルの名前を指定します。
spec.nodeSelectors	**object[]**	オプション: 一致式と一致ラベルを使用してセレクターを指定し、BGP ピアに接続できるノードを制御します。

フィールド	型	説明
spec.ebgpMultiHop	**boolean**	オプション: BGP ピアがネットワークホップ数回分を離れるように指定します。BGP ピアが同じネットワークに直接接続されていない場合には、このフィールドが**true**に設定されていないと、speaker は BGP セッションを確立できません。このフィールドは**外部 BGP**に適用されます。外部 BGP は、BGP ピアが別の Autonomous System に属する場合に使用される用語です。

注記

passwordSecret フィールドは、**password** フィールドと相互に排他的であり、使用するパスワードを含むシークレットへの参照が含まれています。両方のフィールドを設定すると、解析が失敗します。

36.6.2. BGP ピアの設定

クラスター管理者は、BGP ピアカスタムリソースを追加して、ネットワークルーターとルーティング情報を交換し、サービスの IP アドレスをアドバタイズできます。

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ログインしている。
- BGP アドバタイズメントを使用して MetalLB を設定します。

手順

1. 以下の例のような内容で、**bgppeer.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-peer
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

2. BGP ピアの設定を適用します。

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

36.6.3. 指定されたアドレスプールに対して特定の BGP ピアセットを設定

これは、以下を実行するための手順です。

- アドレスプールのセット (**pool1** および **pool2**) を設定します。
- BGP ピアセット (**peer1** および **peer2**) を設定します。
- **pool1** を **peer1** に、**pool2** を **peer2** に割り当てるように BGP アドバタイズメントを設定します。

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. アドレスプール **pool1** を作成します。
   a. 以下の例のような内容で、**ipaddresspool1.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool1
spec:
  addresses:
    - 4.4.4.100-4.4.4.200
    - 2001:100:4::200-2001:100:4::400
```

   b. IP アドレスプール **pool1** の設定を適用します。

```
$ oc apply -f ipaddresspool1.yaml
```

2. アドレスプール **pool2** を作成します。
   a. 以下の例のような内容で、**ipaddresspool2.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: pool2
spec:
  addresses:
    - 5.5.5.100-5.5.5.200
    - 2001:100:5::200-2001:100:5::400
```

   b. IP アドレスプール **pool2** の設定を適用します。

```
$ oc apply -f ipaddresspool2.yaml
```

3. BGP **peer1** を作成します。
   a. 以下の例のような内容で、**bgppeer1.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer1
spec:
  peerAddress: 10.0.0.1
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

b. BGP ピアの設定を適用します。

```
$ oc apply -f bgppeer1.yaml
```

4. BGP **peer2** を作成します。

a. 以下の例のような内容で、**bgppeer2.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: peer2
spec:
  peerAddress: 10.0.0.2
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

b. BGP peer2 の設定を適用します。

```
$ oc apply -f bgppeer2.yaml
```

5. BGP advertisement 1 を作成します。

a. 以下の例のような内容で、**bgpadvertisement1.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-1
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool1
  peers:
    - peer1
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
```

b. 設定を適用します。

```
$ oc apply -f bgpadvertisement1.yaml
```

6. BGP advertisement 2 を作成します。

a. 以下の例のような内容で、**bgpadvertisement2.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgpadvertisement-2
  namespace: metallb-system
spec:
  ipAddressPools:
    - pool2
  peers:
    - peer2
  communities:
    - 65535:65282
  aggregationLength: 32
  aggregationLengthV6: 128
  localPref: 100
```

b. 設定を適用します。

```
$ oc apply -f bgpadvertisement2.yaml
```

36.6.4. ネットワーク VRF を介したサービスの公開

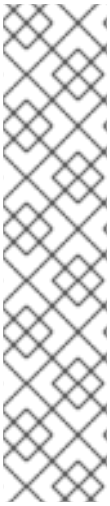
ネットワークインターフェイス上の VRF を BGP ピアに関連付けることにより、仮想ルーティングおよび転送 (VRF) インスタンスを通じてサービスを公開できます。

重要

BGP ピア上の VRF を介したサービスの公開は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

ネットワークインターフェイス上で VRF を使用して BGP ピア経由でサービスを公開すると、サービスへのトラフィックを分離し、独立したルーティング決定を設定し、ネットワークインターフェイス上でマルチテナントのサポートを有効化できます。



注記

ネットワーク VRF に属するインターフェイスを通じて BGP セッションを確立すると、MetalLB はそのインターフェイスを通じてサービスをアドバタイズし、外部トラフィックがこのインターフェイスを通じてサービスに到達させることができます。ただし、ネットワーク VRF ルーティングテーブルは、OVN-Kubernetes で使用されるデフォルトの VRF ルーティングテーブルとは異なります。したがって、トラフィックは OVN-Kubernetes ネットワークインフラストラクチャーに到達できません。

サービスに送信されたトラフィックが OVN-Kubernetes ネットワークインフラストラクチャーに到達できるようにするには、ルーティングルールを設定してネットワークトラフィックのネクストホップを定義する必要があります。詳細は、**関連情報** セクションの MetalLB を使用した対称ルーティングの管理の **NodeNetworkConfigurationPolicy** リソースを参照してください。

BGP ピアを使用してネットワーク VRF を介してサービスを公開するための概要手順は次のとおりです。

1. BGP ピアを定義し、ネットワーク VRF インスタンスを追加します。
2. MetalLB の IP アドレスプールを指定します。
3. MetalLB の BGP ルートアドバタイズメントを設定して、指定された IP アドレスプールと VRF インスタンスに関連付けられた BGP ピアを使用してルートをアドバタイズします。
4. サービスをデプロイして設定をテストします。

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。
- **NodeNetworkConfigurationPolicy** を定義して、仮想ルーティングおよび転送 (VRF) インスタンスをネットワークインターフェイスに関連付けている。この前提条件を満たす方法の詳細は、**関連情報** セクションを参照してください。
- MetalLB をクラスターにインストールしている。

手順

1. **BGPPeer** カスタムリソース (CR) を作成します。
   a. 次の例のような内容を含むファイル (**frrviavrf.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
  peerASN: 200
  peerAddress: 192.168.130.1
  vrf: ens4vrf (1)
```

1 BGP ピアに関連付けるネットワーク VRF インスタンスを指定します。MetalLB は、サービスをアドバタイズし、VRF 内のルーティング情報に基づいてルーティングを決

注記

このネットワーク VRF インスタンスは **NodeNetworkConfigurationPolicy** CR で設定する必要があります。詳細は、**関連情報** を参照してください。

b. 次のコマンドを実行して、BGP ピアの設定を適用します。

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

2. **IPAddressPool** CR を作成します。

a. 次の例のような内容を含むファイル (**first-pool.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.169.10.0/32
```

b. 次のコマンドを実行して、IP アドレスプールの設定を適用します。

```
$ oc apply -f first-pool.yaml
```

3. **BGPAdvertisement** CR を作成します。

a. 次の例のような内容を含むファイル (**first-adv.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: first-adv
  namespace: metallb-system
spec:
  ipAddressPools:
   - first-pool
  peers:
   - frrviavrf 1
```

1 この例では、MetalLB は、**first-pool** の IP アドレスプールから **frrviavrf** BGP ピアに IP アドレスの範囲をアドバタイズします。

b. 次のコマンドを実行して、BGP アドバタイズメントの設定を適用します。

```
$ oc apply -f first-adv.yaml
```

4. **Namespace**、**Deployment**、および **Service** CR を作成します。

a. 次の例のような内容を含むファイル (**deploy-service.yaml** など) を作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: test
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: server
  namespace: test
spec:
  selector:
    matchLabels:
      app: server
  template:
    metadata:
      labels:
        app: server
    spec:
      containers:
      - name: server
        image: registry.redhat.io/ubi9/ubi
        ports:
        - name: http
          containerPort: 30100
        command: ["/bin/sh", "-c"]
        args: ["sleep INF"]
---
apiVersion: v1
kind: Service
metadata:
  name: server1
  namespace: test
spec:
  ports:
  - name: http
    port: 30100
    protocol: TCP
    targetPort: 30100
  selector:
    app: server
  type: LoadBalancer
```

b. 次のコマンドを実行して、namespace 、デプロイメント、およびサービスの設定を適用します。

```
$ oc apply -f deploy-service.yaml
```

検証

1. 次のコマンドを実行して、MetalLB スピーカー Pod を識別します。

```
$ oc get -n metallb-system pods -l component=speaker
```

出力例

```
NAME          READY   STATUS    RESTARTS   AGE
speaker-c6c5f   6/6     Running   0          69m
```

2. 次のコマンドを実行して、BGP セッションの状態がスピーカー Pod で **Established** となっていることを確認します。設定に一致するように変数を置き換えます。

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf <vrf_name> neigh"
```

出力例

```
BGP neighbor is 192.168.30.1, remote AS 200, local AS 100, external link
  BGP version 4, remote router ID 192.168.30.1, local router ID 192.168.30.71
  BGP state = Established, up for 04:20:09

...
```

3. 次のコマンドを実行して、サービスが正しくアドバタイズされていることを確認します。

```
$ oc exec -n metallb-system <speaker_pod> -c frr -- vtysh -c "show bgp vrf <vrf_name> ipv4"
```

関連情報

- 仮想ルーティングおよび転送について
- 例: VRF インスタンスノードのネットワーク設定ポリシーを使用したネットワークインターフェイス
- 出力サービスの設定
- MetalLB を使用した対称ルーティングの管理

36.6.5. BGP ピア設定の例

36.6.5.1. 例: BGP ピアに接続するノードの制限

ノードセレクターフィールドを指定して、BGP ピアに接続できるノードを制御できます。

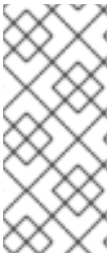
```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-nodesel
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  nodeSelectors:
  - matchExpressions:
```

```
    - key: kubernetes.io/hostname
      operator: In
      values: [compute-1.example.com, compute-2.example.com]
```

36.6.5.2. 例: BGP ピアの BFD プロファイル指定

BGP ピアに関連付ける BFD プロファイルを指定できます。BFD は、BGP のみの場合よりも、ピア間の通信障害をより迅速に検出して、BGP を補完します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-peer-bfd
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64501
  myASN: 64500
  holdTime: "10s"
  bfdProfile: doc-example-bfd-profile-full
```



注記

双方向転送検出 (BFD) プロファイルを削除し、ボーダーゲートウェイプロトコル (BGP) ピアリソースに追加された **bfdProfile** を削除しても、BFD は無効になりません。代わりに、BGP ピアはデフォルトの BFD プロファイルの使用を開始します。BGP ピアリソースから BFD をディセーブルにするには、BGP ピア設定を削除し、BFD プロファイルなしで再作成します。詳細は、BZ#2050824 を参照してください。

36.6.5.3. 例: デュアルスタックネットワーク用の BGP ピア指定

デュアルスタックネットワーキングをサポートするには、IPv4 用に BGP ピアカスタムリソース 1 つと IPv6 用に BGP ピアカスタムリソースを 1 つ追加します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv4
  namespace: metallb-system
spec:
  peerAddress: 10.0.20.1
  peerASN: 64500
  myASN: 64500
---
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: doc-example-dual-stack-ipv6
  namespace: metallb-system
spec:
  peerAddress: 2620:52:0:88::104
  peerASN: 64500
  myASN: 64500
```

36.6.6. 次のステップ

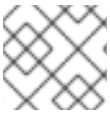
- MetalLB を使用するためのサービスの設定

36.7. コミュニティーエイリアスの設定

クラスター管理者は、コミュニティーエイリアスを設定して、さまざまなアドバタイズメントで使用できます。

36.7.1. コミュニティーカスタムリソースについて

community カスタムリソースは、コミュニティーのエイリアスのコレクションです。ユーザーは、**BGPAdvertisement** を使用して **ipAddressPools** をアドバタイズするときに使用される名前付きエイリアスを定義できます。次の表で、**community** カスタムリソースのフィールドについて説明します。



注記

community CRD は BGPAdvertisement にのみ適用されます。

表36.6 MetalLB コミュニティーカスタムリソース

フィールド	型	説明
metadata.name	**string**	**community** の名前を指定します。
metadata.namespace	**string**	**community** 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。
spec.communities	**string**	BGPAdvertisements で使用できる BGP コミュニティーエイリアスのリストを指定します。コミュニティーエイリアスは、名前 (エイリアス) と値 (番号:番号) のペアで構成されます。**spec.communities** フィールドのエイリアス名を参照して、BGPAdvertisement をコミュニティーエイリアスにリンクします。

表36.7 CommunityAlias

フィールド	型	説明
name	**string**	**community** のエイリアスの名前。
value	**string**	指定された名前に対応する BGP **community** 値。

36.7.2. BGP アドバタイズメントとコミュニティーエイリアスを使用した MetalLB の設定

MetalLB を次のように設定し、**IPAddressPool** が BGP プロトコルでアドバタイズされ、コミュニティーエイリアスが NO_ADVERTISE コミュニティーの数値に設定されるようにします。

次の例では、ピア BGP ルーター **doc-example-peer-community** は、MetalLB がサービスに割り当てるロードバランサー IP アドレスごとに 1 つの **203.0.113.200/32** ルートと 1 つの **fc00:f853:ccd:e799::1/128** ルートを受信します。コミュニティーエイリアスは、**NO_ADVERTISE** コミュニティーで設定されます。

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. IP アドレスプールを作成します。
   a. 以下の例のような内容で、**ipaddresspool.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: doc-example-bgp-community
spec:
  addresses:
    - 203.0.113.200/30
    - fc00:f853:ccd:e799::/124
```

   b. IP アドレスプールの設定を適用します。

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

2. **community1** という名前のコミュニティーエイリアスを作成します。

```
apiVersion: metallb.io/v1beta1
kind: Community
metadata:
  name: community1
  namespace: metallb-system
spec:
  communities:
    - name: NO_ADVERTISE
      value: '65535:65282'
```

3. **doc-example-bgp-peer** という名前の BGP ピアを作成します。
   a. 以下の例のような内容で、**bgppeer.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  namespace: metallb-system
  name: doc-example-bgp-peer
spec:
  peerAddress: 10.0.0.1
```

```
  peerASN: 64501
  myASN: 64500
  routerID: 10.10.10.10
```

b. BGP ピアの設定を適用します。

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

4. コミュニティーエイリアスを使用して BGP アドバタイズメントを作成します。

a. 以下の例のような内容で、**bgpadvertisement.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: bgp-community-sample
  namespace: metallb-system
spec:
  aggregationLength: 32
  aggregationLengthV6: 128
  communities:
    - NO_ADVERTISE (1)
  ipAddressPools:
    - doc-example-bgp-community
  peers:
    - doc-example-peer
```

(1) ここでは、コミュニティーのカスタムリソース (CR) 名ではなく、**CommunityAlias.name** を指定します。

b. 設定を適用します。

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

36.8. METALLB BFD プロファイルの設定

クラスター管理者は、双方向フォワーディング検出 (BFD) プロファイルを追加、変更、および削除できます。MetalLB Operator は、BFD プロファイルのカスタムリソースを使用して、BFD を使用する BGP セッションで、BGP だけの時よりも障害検出のパスを素早く見つけ出すセッションを特定します。

36.8.1. BFD プロファイルカスタムリソースについて

次の表で、BFD プロファイルのカスタムリソースのフィールドについて説明します。

表36.8 BFD プロファイルカスタムリソース

フィールド	型	説明
metadata.name	**string**	BFD プロファイルカスタムリソースの名前を指定します。

フィールド	型	説明
metadata.namespace	**string**	BFD プロファイルカスタムリソースの namespace を指定します。
spec.detectMultiplier	**integer**	パケット損失を決定するための検出乗数を指定します。リモート送信間隔にこの値を乗算して、接続損失検出タイマーを決定します。 たとえば、ローカルシステムの検出乗数が**3**に設定され、リモートシステムの送信間隔が**300**に設定されている場合に、ローカルシステムはパケットを受信せずに**900**ミリ秒後にのみ障害を検出します。 範囲は **2** から **255** です。デフォルト値は **3** です。
spec.echoMode	**boolean**	エコー送信モードを指定します。分散 BFD を使用していないと、エコー送信モードは、ピアが FRR でもある場合にのみ機能します。デフォルト値は**false**で、エコー送信モードは無効になっています。 エコー送信モードが有効になっている場合は、制御パケットの送信間隔を増やして、帯域幅の使用量を減らすことを検討してください。たとえば、送信間隔を**2000**ミリ秒に増やすことを検討してください。
spec.echoInterval	**integer**	このシステムがエコーパケットの送受信に使用する最小送信間隔 (ジッターの軽減) を指定します。範囲は **10** から **60000** です。デフォルト値は **50** ミリ秒です。
spec.minimumTtl	**integer**	着信制御パケットに最小限必要な TTL を指定します。このフィールドは、マルチホップセッションにのみ適用されます。 最小 TTL を設定する目的は、パケット検証要件をより厳しくし、他のセッションからの制御パケットの受信を回避することです。 デフォルト値は**254**で、システムでは、システムとピアの間のホップ数が 1 回のみとすると指定しています。
spec.passiveMode	**boolean**	セッションをアクティブまたはパッシブとしてマークするかどうかを指定します。パッシブセッションは接続の開始を試行しません。代わりに、パッシブセッションは、応答の開始前にピアからの制御パケットを待機します。 セッションをパッシブとしてマークすることは、スターネットワークの中央ノードとして機能するルーターがあり、システムが送信する必要のない制御パケットの送信を避ける場合に役立ちます。 デフォルト値は**false**で、セッションをアクティブとしてマークします。

フィールド	型	説明
spec.receiveInterval	**integer**	このシステムが制御パケットを受信できる最小間隔を指定します。範囲は **10** から **60000** です。デフォルト値は **300** ミリ秒です。
spec.transmitInterval	**integer**	このシステムが制御パケットの送信に使用する最小送信間隔 (ジッターの軽減) を指定します。範囲は **10** から **60000** です。デフォルト値は **300** ミリ秒です。

36.8.2. BFD プロファイルの設定

クラスター管理者は、BFD プロファイルを追加し、そのプロファイルを使用するように BGP ピアを設定できます。BFD は、BGP のみよりも、パスの障害検出が高速になります。

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. 次の例のようなコンテンツを含む **bfdprofile.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: BFDProfile
metadata:
  name: doc-example-bfd-profile-full
  namespace: metallb-system
spec:
  receiveInterval: 300
  transmitInterval: 300
  detectMultiplier: 3
  echoMode: false
  passiveMode: true
  minimumTtl: 254
```

2. BFD プロファイルの設定を適用します。

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

36.8.3. 次のステップ

- BFD プロファイルを使用するようにBGP ピアを設定します。

36.9. 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。

36.9.1. 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
```

36.9.2. 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。

36.9.3. 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
```

36.9.4. 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。

36.9.5. 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
```

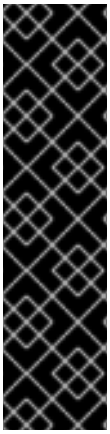
```
----    ------        ----             ----            -------
 Normal  nodeAssigned  32m (x2 over 32m)  metallb-speaker  announcing from node "
<node_name>"
```

<.> 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。

36.10. METALLB を使用した対称ルーティングの管理

クラスター管理者は、MetalLB、NMState、OVN-Kubernetes の機能を実装することで、複数のホストインターフェイスを備えた MetalLB ロードバランサーサービスの背後にある Pod のトラフィックを効果的に管理できます。このコンテキストでこれらの機能を組み合わせることで、対称ルーティング、トラフィック分離を提供し、重複する CIDR アドレスを持つ異なるネットワーク上のクライアントをサポートできます。

この機能を実現するには、MetalLB を使用して仮想ルーティングおよび転送 (VRF) インスタンスを実装し、出力サービスを設定する方法を学習します。



重要

MetalLB と出力サービスを備えた VRF インスタンスを使用した対称トラフィックの設定は、テクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品サポート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場合があります。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。

36.10.1. MetalLB を使用した対称ルーティング管理の課題

複数のホストインターフェイスで MetalLB を使用すると、MetalLB はホスト上で使用可能なすべてのインターフェイスを通じてサービスを公開し、通知します。これにより、ネットワークの分離、非対称リターントラフィック、CIDR アドレスの重複に関する課題が生じる可能性があります。

戻りトラフィックが正しいクライアントに確実に到達するための 1 つのオプションとして、静的ルートを使用します。ただし、このソリューションでは、MetalLB がサービスを分離して、別のインターフェイスを通じて各サービスを通知できません。さらに、静的ルーティングには手動設定が必要であり、リモートサイトが追加された場合はメンテナンスが必要になります。

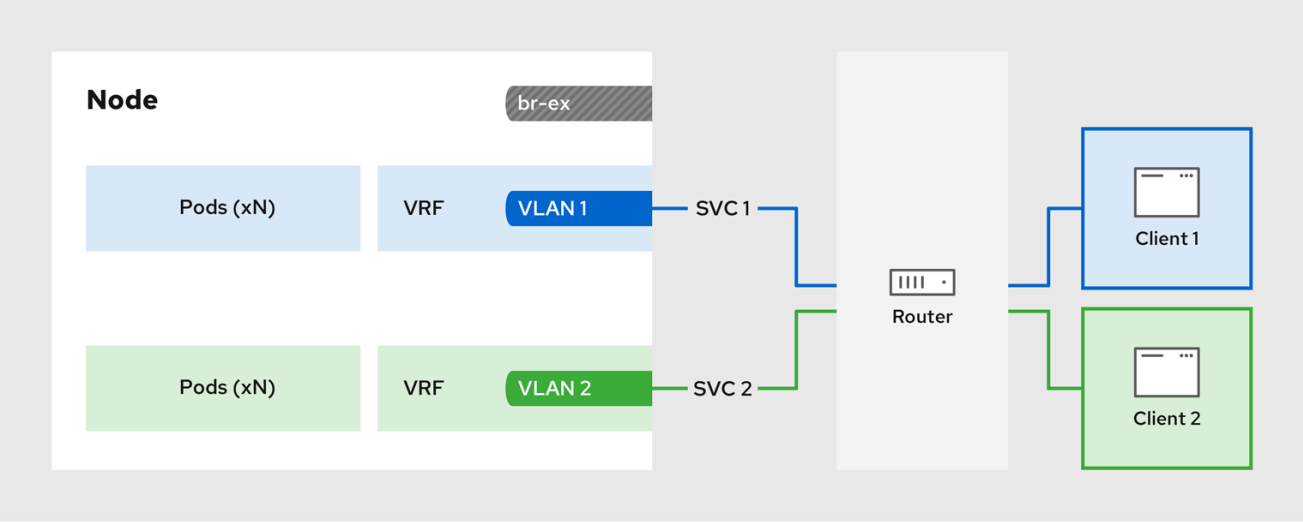
外部システムで、アプリケーションの送信元と宛先の IP アドレスが同じである必要があるシナリオなどが、MetalLB サービスの実装時における対称ルーティングのさらなる課題として挙げられます。OpenShift Container Platform のデフォルトの動作では、ホストネットワークインターフェイスの IP アドレスが、Pod から発信されるトラフィックの送信元 IP アドレスとして割り当てられます。これは、複数のホストインターフェイスがある場合に問題になります。

MetalLB、NMState、OVN-Kubernetes の機能を組み合わせた設定を実装することで、これらの課題を克服できます。

36.10.2. MetalLB で VRF を使用した対称ルーティング管理の概要

NMState を使用してホスト上で VRF インスタンスを設定し、VRF インスタンスを MetalLB **BGPPeer** リソースに関連付け、OVN-Kubernetes を使用して出力トラフィック用の出力サービスを設定することで、対称ルーティングの実装の課題を克服できます。

図36.2 MetalLB で VRF を使用して対称ルーティングを管理するネットワークの概要



設定プロセスには 3 つの段階が含まれます。

1.VRF とルーティングルールを定義する

- **NodeNetworkConfigurationPolicy** カスタムリソース (CR) を設定して、VRF インスタンスをネットワークインターフェイスに関連付けます。
- VRF ルーティングテーブルを使用して、受信トラフィックと送信トラフィックを送信します。

2.VRF を MetalLB BGPPeer にリンクする

- ネットワークインターフェイス上で VRF インスタンスを使用するように MetalLB **BGPPeer** リソースを設定します。
- **BGPPeer** リソースを VRF インスタンスに関連付けることにより、指定されたネットワークインターフェイスが BGP セッションのプライマリーインターフェイスになり、MetalLB はこのインターフェイスを通じてサービスをアドバタイズします。

3.Egress サービスを設定する

- egress サービスを設定して、egress トラフィック用の VRF インスタンスに関連付けられたネットワークを選択します。
- オプション: MetalLB ロードバランサーサービスの IP アドレスを egress トラフィックの送信元 IP として使用するように egress サービスを設定します。

36.10.3. MetalLB で VRF を使用した対称ルーティングの設定

同じ入力ネットワークパスと egress ネットワークパスを必要とする MetalLB サービスの背後にあるアプリケーションに対して対称ネットワークルーティングを設定できます。

この例では、VRF ルーティングテーブルを MetalLB および出力サービスに関連付けて、**LoadBalancer** サービスの背後にある Pod の ingress および egress トラフィックの対称ルーティングを有効にします。

注記

- **EgressService** CR で **sourceIPBy: "LoadBalancerIP"** 設定を使用する場合は、**BGPAdvertising** カスタムリソース (CR) でロードバランサーノードを指定する必要があります。
- **sourceIPBy: "Network"** 設定は、**gatewayConfig.routingViaHost** 仕様が **true** に設定された OVN-Kubernetes を使用するクラスターでのみ使用できます。さらに、**sourceIPBy: "Network"** 設定を使用する場合は、ネットワーク VRF インスタンスで設定されたノード上でアプリケーションワークロードをスケジュール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. **NodeNetworkConfigurationPolicy** CR を作成して VRF インスタンスを定義します。
   a. 次の例のような内容を含むファイル (**node-network-vrf.yaml** など) を作成します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vrfpolicy (1)
spec:
  nodeSelector:
    vrf: "true" (2)
  maxUnavailable: 3
  desiredState:
    interfaces:
      - name: ens4vrf (3)
        type: vrf (4)
        state: up
        vrf:
          port:
            - ens4 (5)
          route-table-id: 2 (6)
    routes: (7)
      config:
        - destination: 0.0.0.0/0
          metric: 150
          next-hop-address: 192.168.130.1
          next-hop-interface: ens4
          table-id: 2
    route-rules: (8)
      config:
```

```
    - ip-to: 172.30.0.0/16
      priority: 998
      route-table: 254
    - ip-to: 10.132.0.0/14
      priority: 998
      route-table: 254
```

1. ポリシーの名前。
2. この例では、**vrf:true** のラベルが割り当てられたべてのノードにポリシーを適用します。
3. インターフェイスの名前。
4. インターフェイスのタイプ。この例では VRF インスタンスを作成します。
5. VRF が接続されるノードインターフェイス。
6. VRF のルートテーブル ID の名前。
7. ネットワークルートの設定を定義します。**next-hop-address** フィールドは、ルートのネクストホップの IP アドレスを定義します。**next-hop-interface** フィールドは、ルートの送信インターフェイスを定義します。この例では、VRF ルーティングテーブルは **2** で、**EgressService** CR で定義した ID を参照します。
8. 追加のルートルールを定義します。**ip-to** フィールドは、**クラスターネットワーク** CIDR および **サービスネットワーク** CIDR と一致する必要があります。これらの CIDR アドレス仕様の値を表示するには、コマンド **oc description network.config/cluster** を実行します。

b. 以下のコマンドを実行してポリシーを適用します。

```
$ oc apply -f node-network-vrf.yaml
```

2. **BGPPeer** カスタムリソース (CR) を作成します。

a. 次の例のような内容を含むファイル (**frr-via-vrf.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta2
kind: BGPPeer
metadata:
  name: frrviavrf
  namespace: metallb-system
spec:
  myASN: 100
  peerASN: 200
  peerAddress: 192.168.130.1
  vrf: ens4vrf (1)
```

1. BGP ピアに関連付ける VRF インスタンスを指定します。MetalLB は、サービスをアドバタイズし、VRF 内のルーティング情報に基づいてルーティングを決定できます。

b. 次のコマンドを実行して、BGP ピアの設定を適用します。

```
$ oc apply -f frr-via-vrf.yaml
```

3. **IPAddressPool** CR を作成します。

   a. 次の例のような内容を含むファイル (**first-pool.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 192.169.10.0/32
```

   b. 次のコマンドを実行して、IP アドレスプールの設定を適用します。

```
$ oc apply -f first-pool.yaml
```

4. **BGPAdvertisement** CR を作成します。

   a. 次の例のような内容を含むファイル (**first-adv.yaml** など) を作成します。

```
apiVersion: metallb.io/v1beta1
kind: BGPAdvertisement
metadata:
  name: first-adv
  namespace: metallb-system
spec:
  ipAddressPools:
    - first-pool
  peers:
    - frrviavrf (1)
  nodeSelectors:
    - matchLabels:
        egress-service.k8s.ovn.org/test-server1: "" (2)
```

   (1) この例では、MetalLB は、**first-pool** の IP アドレスプールから **frrviavrf** BGP ピアに IP アドレスの範囲をアドバタイズします。

   (2) この例では、**EgressService** CR は、ロードバランサーサービス IP アドレスを使用するように、egress トラフィックの送信元 IP アドレスを設定します。したがって、Pod から発信されるトラフィックに同じリターンパスを使用するには、リターントラフィックのロードバランサーノードを指定する必要があります。

   b. 次のコマンドを実行して、BGP アドバタイズメントの設定を適用します。

```
$ oc apply -f first-adv.yaml
```

5. **EgressService** CR を作成します。

   a. 次の例のような内容を含むファイル (**egress-service.yaml** など) を作成します。

```
apiVersion: k8s.ovn.org/v1
kind: EgressService
metadata:
  name: server1 (1)
  namespace: test (2)
spec:
  sourceIPBy: "LoadBalancerIP" (3)
  nodeSelector:
    matchLabels:
      vrf: "true" (4)
  network: "2" (5)
```

1. Egress サービスの名前を指定します。**EgressService** リソースの名前は、変更するロードバランサーサービスの名前と一致する必要があります。
2. Egress サービスの namespace を指定します。**EgressService** の namespace は、変更するロードバランサーサービスの namespace と一致する必要があります。egress サービスは namespace スコープです。
3. この例では、**LoadBalancer** サービスの ingress IP アドレスを egress トラフィックの送信元 IP アドレスとして割り当てます。
4. **sourceIPBy** 仕様の **LoadBalancer** を指定すると、単一ノードが **LoadBalancer** サービストラフィックを処理します。この例では、ラベルが **vrf: "true"** のノードのみがサービストラフィックを処理できます。ノードを指定しない場合、OVN-Kubernetes はサービストラフィックを処理するワーカーノードを選択します。ノードが選択されると、OVN-Kubernetes はそのノードに **egress-service.k8s.ovn.org/<svc_namespace>-<svc_name>: ""** という形式でラベルを付けます。
5. egress トラフィックのルーティングテーブルを指定します。

b. 次のコマンドを実行して、egress サービスの設定を適用します。

```
$ oc apply -f egress-service.yaml
```

検証

1. 次のコマンドを実行して、MetalLB サービスの背後で実行されている Pod のアプリケーションエンドポイントにアクセスできることを確認します。

```
$ curl <external_ip_address>:<port_number> (1)
```

1. アプリケーションのエンドポイントに合わせて外部 IP アドレスとポート番号を更新します。

2. オプション: **LoadBalancer** サービスの ingress IP アドレスを egress トラフィックの送信元 IP アドレスとして割り当てた場合は、**tcpdump** などのツールを使用して外部クライアントで受信したパケットを分析し、この設定を確認します。

関連情報

- 仮想ルーティングおよび転送について
- ネットワーク VRF を介したサービスの公開
- 例: VRF インスタンスノードのネットワーク設定ポリシーを使用したネットワークインターフェイス
- 出力サービスの設定

36.11. METALLB のロギング、トラブルシューティング、サポート

MetalLB 設定のトラブルシューティングが必要な場合は、次のセクションで一般的に使用されるコマンドを参照してください。

36.11.1. MetalLB ログレベルの設定

MetalLB は、デフォルト設定の **info** を使用してコンテナーで FRRouting (FRR) を使用し、大量のログを生成します。この例に示すように **logLevel** を設定することにより、生成されるログの詳細度を制御できます。

次のように **logLevel** を **debug** に設定することで、MetalLB についてより深い洞察を得ることができます。

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

1. 以下の例のような内容で、**setdebugloglevel.yaml** などのファイルを作成します。

```
apiVersion: metallb.io/v1beta1
kind: MetalLB
metadata:
  name: metallb
  namespace: metallb-system
spec:
  logLevel: debug
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

2. 設定を適用します。

```
$ oc replace -f setdebugloglevel.yaml
```

注記

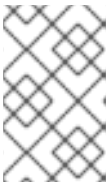
metallb CR はすでに作成されており、ここではログレベルを変更していることを理解たうえで、**ocr eplace** を使用します。

3. **speaker**Pod の名前を表示します。

```
$ oc get -n metallb-system pods -l component=speaker
```

出力例

```
NAME                READY   STATUS    RESTARTS   AGE
speaker-2m9pm         4/4     Running   0          9m19s
speaker-7m4qw         3/4     Running   0          19s
speaker-szlmx         4/4     Running   0          9m19s
```



注記

スピーカー Pod とコントローラー Pod が再作成され、更新されたログレベルが確実に適用されます。MetalLB のすべてのコンポーネントのログレベルが変更されます。

4. **speaker** ログを表示します。

```
$ oc logs -n metallb-system speaker-7m4qw -c speaker
```

出力例

```
{"branch":"main","caller":"main.go:92","commit":"3d052535","goversion":"gc / go1.17.1 /
amd64","level":"info","msg":"MetalLB speaker starting (commit 3d052535, branch
main)","ts":"2022-05-17T09:55:05Z","version":""}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"ens4","level":"info","m
sg":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"ens4","level":"info","m
sg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:110","event":"createARPResponder","interface":"tun0","level":"info","ms
g":"created ARP responder for interface","ts":"2022-05-17T09:55:05Z"}
{"caller":"announcer.go:119","event":"createNDPResponder","interface":"tun0","level":"info","m
sg":"created NDP responder for interface","ts":"2022-05-17T09:55:05Z"}
I0517 09:55:06.515686      95 request.go:665] Waited for 1.026500832s due to client-side
throttling, not priority and fairness, request:
GET:https://172.30.0.1:443/apis/operators.coreos.com/v1alpha1?timeout=32s
{"Starting Manager":"(MISSING)","caller":"k8s.go:389","level":"info","ts":"2022-05-
17T09:55:08Z"}
{"caller":"speakerlist.go:310","level":"info","msg":"node event - forcing sync","node
addr":"10.0.128.4","node event":"NodeJoin","node name":"ci-ln-qb8t3mb-72292-7s7rh-
worker-a-vvznj","ts":"2022-05-17T09:55:08Z"}
{"caller":"service_controller.go:113","controller":"ServiceReconciler","enqueueing":"openshift-
kube-controller-manager-operator/metrics","epslice":"{\"metadata\":{\"name\":\"metrics-
xtsxr\",\"generateName\":\"metrics-\",\"namespace\":\"openshift-kube-controller-manager-
operator\",\"uid\":\"ac6766d7-8504-492c-9d1e-
4ae8897990ad\",\"resourceVersion\":\"9041\",\"generation\":4,\"creationTimestamp\":\"2022-
05-17T07:16:53Z\",\"labels\":{\"app\":\"kube-controller-manager-
operator\",\"endpointslice.kubernetes.io/managed-by\":\"endpointslice-
controller.k8s.io\",\"kubernetes.io/service-name\":\"metrics\"},\"annotations\":
{\"endpoints.kubernetes.io/last-change-trigger-time\":\"2022-05-
17T07:21:34Z\"},\"ownerReferences\":
[{\"apiVersion\":\"v1\",\"kind\":\"Service\",\"name\":\"metrics\",\"uid\":\"0518eed3-6152-42be-
b566-0bd00a60faf8\",\"controller\":true,\"blockOwnerDeletion\":true}],\"managedFields\":
[{\"manager\":\"kube-controller-
```

manager\",\"operation\":\"Update\",\"apiVersion\":\"discovery.k8s.io/v1\",\"time\":\"2022-05-17T07:20:02Z\",\"fieldsType\":\"FieldsV1\",\"fieldsV1\":{\"f:addressType\":{},\"f:endpoints\":{},\"f:metadata\":{\"f:annotations\":{\".\":{},\"f:endpoints.kubernetes.io/last-change-trigger-time\":{}},\"f:generateName\":{},\"f:labels\":{\".\":{},\"f:app\":{},\"f:endpointslice.kubernetes.io/managed-by\":{},\"f:kubernetes.io/service-name\":{}},\"f:ownerReferences\":{\".\":{},\"k:{\\\"uid\\\":\\\"0518eed3-6152-42be-b566-0bd00a60faf8\\\"}\":{}}},\"f:ports\":{}}]},\"addressType\":\"IPv4\",\"endpoints\":[{\"addresses\":[\"10.129.0.7\"],\"conditions\":{\"ready\":true,\"serving\":true,\"terminating\":false},\"targetRef\":{\"kind\":\"Pod\",\"namespace\":\"openshift-kube-controller-manager-operator\",\"name\":\"kube-controller-manager-operator-6b98b89ddd-8d4nf\",\"uid\":\"dd5139b8-e41c-4946-a31b-1a629314e844\",\"resourceVersion\":\"9038\"},\"nodeName\":\"ci-ln-qb8t3mb-72292-7s7rh-master-0\",\"zone\":\"us-central1-a\"}],\"ports\":[{\"name\":\"https\",\"protocol\":\"TCP\",\"port\":8443}]}","level":"debug","ts":"2022-05-17T09:55:08Z"}

5. FRR ログを表示します。

```
$ oc logs -n metallb-system speaker-7m4qw -c frr
```

出力例

```
Started watchfrr
2022/05/17 09:55:05 ZEBRA: client 16 says hello and bids fair to announce only bgp routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 31 says hello and bids fair to announce only vnc routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 38 says hello and bids fair to announce only static routes vrf=0
2022/05/17 09:55:05 ZEBRA: client 43 says hello and bids fair to announce only bfd routes vrf=0
2022/05/17 09:57:25.089 BGP: Creating Default VRF, AS 64500
2022/05/17 09:57:25.090 BGP: dup addr detect enable max_moves 5 time 180 freeze disable freeze_time 0
2022/05/17 09:57:25.090 BGP: bgp_get: Registering BGP instance (null) to zebra
2022/05/17 09:57:25.090 BGP: Registering VRF 0
2022/05/17 09:57:25.091 BGP: Rx Router Id update VRF 0 Id 10.131.0.1/32
2022/05/17 09:57:25.091 BGP: RID change : vrf VRF default(0), RTR ID 10.131.0.1
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF br0
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ens4
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr 10.0.128.4/32
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF ens4 addr fe80::c9d:84da:4d86:5618/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF lo
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF ovs-system
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF tun0
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr 10.131.0.1/23
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF tun0 addr fe80::40f1:d1ff:feb6:5322/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2da49fed
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2da49fed addr fe80::24bd:d1ff:fec1:d88/64
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth2fa08c8c
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth2fa08c8c addr fe80::6870:ff:fe96:efc8/64
```

```
2022/05/17 09:57:25.091 BGP: Rx Intf add VRF 0 IF veth41e356b7
2022/05/17 09:57:25.091 BGP: Rx Intf address add VRF 0 IF veth41e356b7 addr
fe80::48ff:37ff:fede:eb4b/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth1295c6e2
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth1295c6e2 addr
fe80::b827:a2ff:feed:637/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth9733c6dc
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth9733c6dc addr
fe80::3cf4:15ff:fe11:e541/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF veth336680ea
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF veth336680ea addr
fe80::94b1:8bff:fe7e:488c/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vetha0a907b7
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vetha0a907b7 addr
fe80::3855:a6ff:fe73:46c3/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf35a4398
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf35a4398 addr
fe80::40ef:2fff:fe57:4c4d/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vethf831b7f4
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vethf831b7f4 addr
fe80::f0d9:89ff:fe7c:1d32/64
2022/05/17 09:57:25.092 BGP: Rx Intf add VRF 0 IF vxlan_sys_4789
2022/05/17 09:57:25.092 BGP: Rx Intf address add VRF 0 IF vxlan_sys_4789 addr
fe80::80c1:82ff:fe4b:f078/64
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Timer (start timer expire).
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] BGP_Start (Idle->Connect), fd -1
2022/05/17 09:57:26.094 BGP: Allocated bnc 10.0.0.1/32(0)(VRF default) peer
0x7f807f7631a0
2022/05/17 09:57:26.094 BGP: sendmsg_zebra_rnh: sending cmd
ZEBRA_NEXTHOP_REGISTER for 10.0.0.1/32 (vrf VRF default)
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] Waiting for NHT
2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Connect
established_peers 0
2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Idle to Connect
2022/05/17 09:57:26.094 BGP: 10.0.0.1 [FSM] TCP_connection_open_failed (Connect-
>Active), fd -1
2022/05/17 09:57:26.094 BGP: bgp_fsm_change_status : vrf default(0), Status: Active
established_peers 0
2022/05/17 09:57:26.094 BGP: 10.0.0.1 went from Connect to Active
2022/05/17 09:57:26.094 ZEBRA: rnh_register msg from client bgp: hdr->length=8,
type=nexthop vrf=0
2022/05/17 09:57:26.094 ZEBRA: 0: Add RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: Evaluate RNH, type Nexthop (force)
2022/05/17 09:57:26.094 ZEBRA: 0:10.0.0.1/32: NH has become unresolved
2022/05/17 09:57:26.094 ZEBRA: 0: Client bgp registers for RNH 10.0.0.1/32 type Nexthop
2022/05/17 09:57:26.094 BGP: VRF default(0): Rcvd NH update 10.0.0.1/32(0) - metric 0/0
#nhops 0/0 flags 0x6
2022/05/17 09:57:26.094 BGP: NH update for 10.0.0.1/32(0)(VRF default) - flags 0x6
chgflags 0x0 - evaluate paths
2022/05/17 09:57:26.094 BGP: evaluate_paths: Updating peer (10.0.0.1(VRF default)) status
with NHT
2022/05/17 09:57:30.081 ZEBRA: Event driven route-map update triggered
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-out
2022/05/17 09:57:30.081 ZEBRA: Event handler for route-map: 10.0.0.1-in
2022/05/17 09:57:31.104 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type
RTM_NEWNEIGH(28), len=76, seq=0, pid=0
```

```
2022/05/17 09:57:31.104 ZEBRA:  Neighbor Entry received is not on a VLAN or a BRIDGE, ignoring
2022/05/17 09:57:31.105 ZEBRA: netlink_parse_info: netlink-listen (NS 0) type RTM_NEWNEIGH(28), len=76, seq=0, pid=0
2022/05/17 09:57:31.105 ZEBRA:  Neighbor Entry received is not on a VLAN or a BRIDGE, ignoring
```

36.11.1.1. FRRouting (FRR) ログレベル

次の表で、FRR ログレベルについて説明します。

表36.9 ログレベル

ログレベル	説明
all	すべてのログレベルのすべてのログ情報を提供します。
debug	診断に役立つ情報。詳細なトラブルシューティング情報を提供するには、**debug** に設定します。
info	常にログに記録する必要がある情報を提供しますが、通常の状況ではユーザーの介入は必要ありません。これはデフォルトのログレベルです。
warn	一貫性のない **MetalLB** 動作を引き起こす可能性のあるもの。通常、**MetalLB** はこのタイプのエラーから自動的に回復します。
error	**MetalLB** の機能に対して致命的なエラー。通常、これらのエラーの修正には管理者の介入が必要です。
none	すべてのロギングをオフにします。

36.11.2. BGP の問題のトラブルシューティング

Red Hat がサポートする BGP 実装は、**speaker**Pod のコンテナーで FRRouting (FRR) を使用します。クラスター管理者は、BGP 設定の問題をトラブルシューティングする場合に、FRR コンテナーでコマンドを実行する必要があります。

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. **speaker**Pod の名前を表示します。

```
$ oc get -n metallb-system pods -l component=speaker
```

出力例

```
NAME          READY   STATUS    RESTARTS   AGE
speaker-66bth   4/4     Running   0          56m
speaker-gvfnf   4/4     Running   0          56m
...
```

2. FRR の実行設定を表示します。

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show running-config"
```

出力例

```
Building configuration...

Current configuration:
!
frr version 7.5.1_git
frr defaults traditional
hostname some-hostname
log file /etc/frr/frr.log informational
log timestamp precision 3
service integrated-vtysh-config
!
router bgp 64500 (1)
 bgp router-id 10.0.1.2
 no bgp ebgp-requires-policy
 no bgp default ipv4-unicast
 no bgp network import-check
 neighbor 10.0.2.3 remote-as 64500 (2)
 neighbor 10.0.2.3 bfd profile doc-example-bfd-profile-full (3)
 neighbor 10.0.2.3 timers 5 15
 neighbor 10.0.2.4 remote-as 64500 (4)
 neighbor 10.0.2.4 bfd profile doc-example-bfd-profile-full (5)
 neighbor 10.0.2.4 timers 5 15
 !
 address-family ipv4 unicast
  network 203.0.113.200/30 (6)
  neighbor 10.0.2.3 activate
  neighbor 10.0.2.3 route-map 10.0.2.3-in in
  neighbor 10.0.2.4 activate
  neighbor 10.0.2.4 route-map 10.0.2.4-in in
 exit-address-family
 !
 address-family ipv6 unicast
  network fc00:f853:ccd:e799::/124 (7)
  neighbor 10.0.2.3 activate
  neighbor 10.0.2.3 route-map 10.0.2.3-in in
  neighbor 10.0.2.4 activate
  neighbor 10.0.2.4 route-map 10.0.2.4-in in
 exit-address-family
!
route-map 10.0.2.3-in deny 20
!
route-map 10.0.2.4-in deny 20
!
```

```
ip nht resolve-via-default
!
ipv6 nht resolve-via-default
!
line vty
!
bfd
 profile doc-example-bfd-profile-full 8
  transmit-interval 35
  receive-interval 35
  passive-mode
  echo-mode
  echo-interval 35
  minimum-ttl 10
 !
!
end
```

<.> **ルーターの bgp** セクションは、MetalLB の ASN を示します。<.> 追加した各 BGP ピアカスタムリソースに対して、**neighbor <ip-address> remote-as <peer-ASN>** 行が存在することを確認します。<.> BFD を設定した場合は、BFD プロファイルが正しい BGP ピアに関連付けられていること、および BFD プロファイルがコマンド出力に表示されることを確認します。<.> **network <ip-address-range>** 行が、追加したアドレスプールカスタムリソースで指定した IP アドレス範囲と一致することを確認します。

3. BGP サマリーを表示します。

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp summary"
```

出力例

```
IPv4 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3        4      64500       387       389        0    0    0 00:32:02            0        1 1
10.0.2.4        4      64500         0         0        0    0    0    never       Active        0 2

Total number of neighbors 2

IPv6 Unicast Summary:
BGP router identifier 10.0.1.2, local AS number 64500 vrf-id 0
BGP table version 1
RIB entries 1, using 192 bytes of memory
Peers 2, using 29 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
10.0.2.3        4      64500       387       389        0    0    0 00:32:02 NoNeg 3
```

```
10.0.2.4       4     64500       0       0       0    0    0    never       Active      0 4

Total number of neighbors 2
```

1 1 3 追加した各 BGP ピアカスタムリソースの行が出力に含まれていることを確認します。

2 4 2 4 出力に、受信したメッセージと送信したメッセージが**0**が表示されている場合には、BGP ペアに BGP セッションがないことを示します。ネットワーク接続と BGP ピアの BGP 設定を確認します。

4. アドレスプールを受信した BGP ピアを表示します。

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bgp ipv4 unicast
203.0.113.200/30"
```

ipv4を**ipv6**に置き換えて、IPv6 アドレスプールを受信した BGP ピアを表示します。**203.0.113.200/30** は、アドレスプールの IPv4 または IPv6IP アドレス範囲に置き換えます。

出力例

```
BGP routing table entry for 203.0.113.200/30
Paths: (1 available, best #1, table default)
  Advertised to non peer-group peers:
  10.0.2.3  <.>
  Local
    0.0.0.0 from 0.0.0.0 (10.0.1.2)
      Origin IGP, metric 0, weight 32768, valid, sourced, local, best (First path received)
      Last update: Mon Jan 10 19:49:07 2022
```

<.> 出力に BGP ピアの IP アドレスが含まれていることを確認します。

36.11.3. BFD の問題のトラブルシューティング

Red Hat がサポートする双方向フォワーディング検出 (BFD) の実装では、**speaker**Pod のコンテナーで FR Routing (FRR) を使用します。BFD の実装は、BFD ピアに依存しており、このピアは、BGP セッションが確立されている BGP ピアとして設定されています。クラスター管理者は、BFD 設定の問題をトラブルシューティングする場合に、FRR コンテナーでコマンドを実行する必要があります。

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. **speaker**Pod の名前を表示します。

```
$ oc get -n metallb-system pods -l component=speaker
```

出力例

```
NAME          READY   STATUS    RESTARTS   AGE
speaker-66bth   4/4     Running   0          26m
speaker-gvfnf   4/4     Running   0          26m
...
```

2. BFD ピアを表示します。

```
$ oc exec -n metallb-system speaker-66bth -c frr -- vtysh -c "show bfd peers brief"
```

出力例

```
Session count: 2
SessionId  LocalAddress             PeerAddress             Status
=========  ============            ===========             ======
3909139637 10.0.1.2                10.0.2.3                up  <.>
```

<.> **PeerAddress** 列に各 BFD ピアが含まれていることを確認します。出力に含まれると予想される BFD ピア IP アドレスが出力にリストされていない場合は、ピアとの BGP 接続のトラブルシューティングを行います。ステータスフィールドが**down**と表示されている場合は、ノードとピア間のリンクと機器の接続を確認します。speaker Pod のノード名は、 **oc get pods -n metallb-system speaker-66bth -o jsonpath='{.spec.nodeName}'** などのコマンドで判断できます。

36.11.4. BGP および BFD の MetalLB メトリック

OpenShift Container Platform は、BGP ピアおよび BFD プロファイルに関連する、MetalLB の以下のメトリクスをキャプチャーします。

表36.10 MetalLB BFD メトリクス

名前	説明
metallb_bfd_control_packet_input	各 BFD ピアから受信した BFD 制御パケットの数をカウントします。
metallb_bfd_control_packet_output	各 BFD ピアに送信された BFD 制御パケットの数をカウントします。
metallb_bfd_echo_packet_input	各 BFD ピアから受信した BFD エコーパケットの数をカウントします。
metallb_bfd_echo_packet_output	各 BFD に送信された BFD エコーパケットの数をカウントします。
metallb_bfd_session_down_events	ピアとの BFD セッションが **down** 状態になった回数をカウントします。
metallb_bfd_session_up	BFD ピアとの接続状態を示します。**1**はセッションが**up** であること、 **0** は **down** であることを示します。

名前	説明
metallb_bfd_session_up_events	ピアとの BFD セッションが **up** 状態になった回数をカウントします。
metallb_bfd_zebra_notifications	各 BFD ピアの BFD Zebra 通知の数をカウントします。

表36.11 MetalLB BGP メトリクス

名前	説明
metallb_bgp_announced_prefixes_total	BGP ピアにアドバタイズされるロードバランサーの IP アドレス接頭辞の数をカウントします。**接頭辞**と**集約ルート**という用語は同じ意味です。
metallb_bgp_session_up	BGP ピアとの接続状態を示します。**1**はセッションが**up** であること、 **0** は **down** であることを示します。
metallb_bgp_updates_total	各 BGP ピアに送信された BGP 更新メッセージの数をカウントします。
metallb_bgp_opens_sent	各 BGP ピアに送信された BGP オープンメッセージの数をカウントします。
metallb_bgp_opens_received	各 BGP ピアから受信した BGP オープンメッセージの数をカウントします。
metallb_bgp_notifications_sent	各 BGP ピアに送信された BGP 通知メッセージの数をカウントします。
metallb_bgp_updates_total_received	各 BGP ピアから受信した BGP 更新メッセージの数をカウントします。
metallb_bgp_keepalives_sent	各 BGP ピアに送信された BGP keepalive メッセージの数をカウントします。
metallb_bgp_keepalives_received	各 BGP ピアから受信した BGP keepalive メッセージの数をカウントします。
metallb_bgp_route_refresh_sent	各 BGP ピアに送信された BGP ルートリフレッシュメッセージの数をカウントします。
metallb_bgp_total_sent	各 BGP ピアに送信された BGP メッセージの合計数をカウントします。
metallb_bgp_total_received	各 BGP ピアから受信した BGP メッセージの合計数をカウントします。

関連情報

- 監視ダッシュボードの使用については、メトリックのクエリーを参照してください。

36.11.5. Metal LB データの収集について

oc adm must-gather CLI コマンドを使用して、クラスター、MetalLB 設定、および MetalLB Operator に関する情報を収集できます。次の機能とオブジェクトは、MetalLB と MetalLB Operator に関連付けられています。

- MetalLB Operator がデプロイされている namespace と子オブジェクト
- すべての MetalLB Operator カスタムリソース定義 (CRD)

oc adm must-gatherCLI コマンドは、Red Hat が BGP および BFD 実装に使用する FR Routing (FRR) から次の情報を収集します。

- **/etc/frr/frr.conf**
- **/etc/frr/frr.log**
- **/etc/frr/daemons** 設定ファイル
- **/etc/frr/vtysh.conf**

上記のリストのログファイルと設定ファイルは、各**speaker** Pod の **frr** コンテナーから収集されます。

ログファイルと設定ファイル以外に、**oc adm must-gather** の CLI コマンドは、次の**vtysh**コマンドからの出力を収集します。

- **show running-config**
- **show bgp ipv4**
- **show bgp ipv6**
- **show bgp neighbor**
- **show bfd peer**

oc adm must-gather CLI コマンドを実行する場合、追加の設定は必要ありません。

関連情報

- クラスターに関するデータの収集

第37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

37.1. 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ック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。

37.1.1. Network Metrics Daemon

Network Metrics Daemon は、ネットワーク関連のメトリックを収集し、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ックを公開しています。以下は、これらのメトリックになります。

- **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**

これらのメトリックのラベルには、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**)

これらのメトリックは、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ック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。

37.1.2. ネットワーク名を持つメトリック

この daemonset は、固定の値が **0** の **pod_network_name_info** 測定メトリックを公開します。

```
pod_network_name_info{interface="net0",namespace="namespacename",network_name="nadnamespace/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/network-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)
```