Unified Load Balancer 2026.02

Cisco Ultra Cloud Core - Subscriber Microservices Infrastructure
Updated April 23, 2026

第 1 章 : Unified Load Balancer の概要....4
ULB の概要....5
ULB の主な特長と機能....5
ULB を使用する利点....6
制限事項....6

第 2 章 : ULB デプロイメント アーキテクチャ....8
ULB の主なコンポーネント....10

第 3 章 : スタート ガイド....12
要　件....13
オンプレミスデプロイメントの初期構成....13
Cilium の構成....13

第 4 章 : Ops センター経由で SMP をデプロイして構成する....16
ULB Ops センター....17
ULB デプロイメントの前提条件....17
ULB をデプロイする....17
ULB Ops センターにアクセスする....19

第 5 章 : Unified Load Balancer の通信分散....22
ULB の仕組み....23
ULB Pod の可用性と構成の調整....23

第 6 章 : ULB の機能と構成....26
BGP および BFD によるレジリエンスと通信コンバージェンス....27
レジリエンスと通信コンバージェンスの仕組み....27
ネイバー探索....32
ネイバー探索の仕組み....32
更新間隔、インターフェイス、ノードラベルを構成する....32
ネイバーエントリを監視する....34
コアダンプの生成....34
サービス アウトバウンド ルート（SOR）....35
SOR を実装する利点....35
SOR の制限事項と制約事項....35
SOR 機能の仕組み....35
eBPF に基づくデータパスメトリックのサポート....36

eBPF に基づくサービスのパケットおよびバイトメトリクス....36
eBPF に基づく ULB 運用データパスメトリック....38
その他の構成....40
cilium BPF マップを構成する....40
Cilium Hubible 統計を有効にする....42
ULB データパスロギングを有効または無効にする....42
ULB クラスターポリシー（UCP）....43
ULB クラスターポリシーを構成する....43
ULB ノードインターフェイスの運用状態....45
IP プールを使用して ULB クラスターポリシーを構成する....45
ULB ネットワークポリシー（UNP）....46
ULB ネットワークポリシーを構成する....46
ダイレクトサービス....48
ULB で SCTP ロードバランシングをサポートする....48
SCTP ダイレクトサービスのロードバランサを構成する....50
SCTP サービス配布を管理する....51
ULB で GTP-C ロードバランシングをサポートする....52
GTP- C ロードバランサのカスタムリソース....52
GTP-C 通信管理ポリシーのカスタムリソース....53
GTP-C エグレス管理ポリシーカスタムリソース....54
セッションレベルのメッセージ....55
ノードレベルのメッセージ....55
制限事項：....55
ULB で PFCP ロードバランシングをサポートする....56
PFCP ロードバランサのカスタムリソース....56
PFCP 通信管理ポリシーのカスタムリソース....57
PFCP エグレス管理ポリシーカスタムリソース....58

第 7 章：評価指標と KPI....60
ULB オペレータメトリック....61
ULB エージェントメトリック....63

付録 A：ロードバランサとエグレス管理ポリシーのカスタムリソース定義の例....68
ロードバランサ CR（IPv4）....69
ロードバランサ CR（IPv6）....69
ステートフル Pod のエグレス管理ポリシー CR（IPv4）....69
ステートフル Pod のエグレス管理ポリシー CR（IPv6）....70
ステートレス Pod のエグレス管理ポリシー CR（IPv4）....70
ステートレス Pod のエグレス管理ポリシー CR（IPv6）....71
ポート範囲のないステートフル Pod のエグレス管理ポリシー CR（IPv4）....71
サービス アウトバウンド ルート（SOR）IPv4....71
サービス アウトバウンド ルート（SOR）IPv6....72

第

1 Unified Load Balancer

章

:

- ULB □□□
- ULB □□□□□□□□
- ULB □□□□□□□
- □□□□

このトピックでは、Unified Load Balancer（ULB）の概要、主な機能、能力、利点、および制限事項などについて説明します。

このトピックでは、Unified Load Balancer（ULB）の概要について説明し、機能、利点、制限事項に焦点を当てます。

ULB

Unified Load Balancer（ULB）について説明します。これは、ワーカーノード間の通信分散を管理して、ネットワーク インフラストラクチャの高可用性と信頼性を確保するシスコ独自のネットワーク機能です。

Unified Load Balancer（ULB）は、すべてのワーカーノード間で受信ネットワーク通信の通信分散を管理し、ネットワーク インフラストラクチャ全体で高可用性と信頼性を確保する、シスコ独自のノードまたはネットワーク機能（NF）です。

- これは、Subscriber Microservices Infrastructure（SMI）、クラウドネイティブ（CN）アプリケーションを使用して開発されました。
- CN ベースのイメージとして配信され、SMI Cluster Manager（SMI CM）を使用して CN アプリケーションとしてデプロイされます。
- 複数のプロトコル Pod 間でロード バランシングを必要とする UDP と TCP を使用するプロトコルアプリケーションをサポートします。

ULB

通信分散、クライアント IP の保持、DSR、ダイナミックスケーリング、サービス IP の一貫性など、ULB の主要な機能について説明します。

ULB は次の機能をサポートします。

- 均等な通信分散とクライアント IP の保持：
  - Layer 4 プロトコル（UDP、TCP）のすべてのワーカーノードに、受信ネットワーク通信を均等に分散します。
  - 入力通信をバックエンド Pod に分散する際に、元のクライアントの IP アドレスを維持します。
- Direct Server Response（DSR）とサービス IP の一貫性：
  - バックエンド Pod が、ロードバランサをバイパスしてクライアントに直接応答できるようにして、遅延と負荷を軽減します。
  - 応答パケットに、イングレス用に公開されたサービス IP アドレスと一致する一貫した送信元 IP アドレスが含まれていることを確認します。
- ダイナミックスケーリングおよびリアルタイムの構成の更新：
  - ワーカーノードの追加や削除をシームレスに行い、ダウンタイムなしで新しいノードを通信分散に自動的に統合します。
  - 新しいバックエンド Pod がデプロイ時に、通信のルーティングスキームに動的に組み込まれます。
  - ワーカーノードまたはバックエンド Pod の変更に応じて、ロードバランサの構成に対してリアルタイムの更新をトリガーします。
- 出力通信のサービス IP の一意性と整合性：
  - 複数のワーカーノードを通過する場合でも、出力通信の一意のサービス IP アドレスを維持します。
  - イングレス要求とエグレス要求の両方に対して同じサービス IP アドレスを使用し、外部システムに対して一貫した IP アドレスを指定します。
- 出力通信の応答ルーティングと UDP 処理：

- エグレス要求を開始するバックエンドPodに応答を正確にルーティングし、元のクライアントIPアドレスを保持します。
- エグレス要求を送信したPodでない場合でも、UDPプロトコルのレスポンスをバックエンドPodにルーティングします。

この機能の構成の詳細については、「ULBの機能と構成（26ページ）」項を参照してください。

ULB

ULBを使用する利点について説明し、主要なパラメータ、要件、および参考情報についてまとめます。ネットワークパフォーマンス、可用性、拡張性、セキュリティ、最適化にULBが重要な理由について説明します。

ULBは、次の理由で重要です。

- 通信管理：ULBは受信ネットワーク通信を複数のサーバーに分散して、単一のサーバーが過負荷にならないようにします。このバランスの取れた通信分散により、ボトルネックが回避され、ネットワーク全体のパフォーマンスと信頼性が向上します。これは、UDPやTCPなどのレイヤ4（L4）プロトコルに適用されます。
- 高可用性：複数のサーバーに通信を分散することで、ULBは単一のサーバーの障害や使用不能がシステム全体に影響を及ぼさないようにします。この冗長性は、ユーザーに対する継続的なサービスの可用性を維持するために重要です。
- 拡張性：ULBは、水平スケーリングをサポートしており、既存のサービスを中断することなく、より多くのサーバーを追加して通信負荷の増加を処理できるようにします。この拡張性は、通信の増加と突然のスパイクに対応するために不可欠です。
- セキュリティ：ULBは、内部サーバーIPアドレスをマスキングし、ソース接続を管理することでセキュリティに追加の層を提供できます。これにより、バックエンドインフラストラクチャを潜在的な脅威から直接保護することができます。
- パフォーマンスの最適化：ULBは、Direct Server Response（DSR）を容易にすることで、遅延とロードバランサノードの負荷を軽減し、ネットワーク通信のパフォーマンスを最適化します。

これらの制限はIPv4およびStatefulSetワークロードにのみ適用されるため、portRangeのないステートフルアプリケーションにEgressManagementPolicyを設定する場合は、次のガイドラインを使用します。

- portRangeのないステートフルアプリケーションのEgressManagementPolicyは、IPv4にのみ適用されます。
- 現在、portRangesが設定されていないステートフルアプリケーションのEgressManagementPolicy（EMP）は、StatefulSetワークロードによって管理されるリソースに対してのみ機能します。

第

2 ULB

章

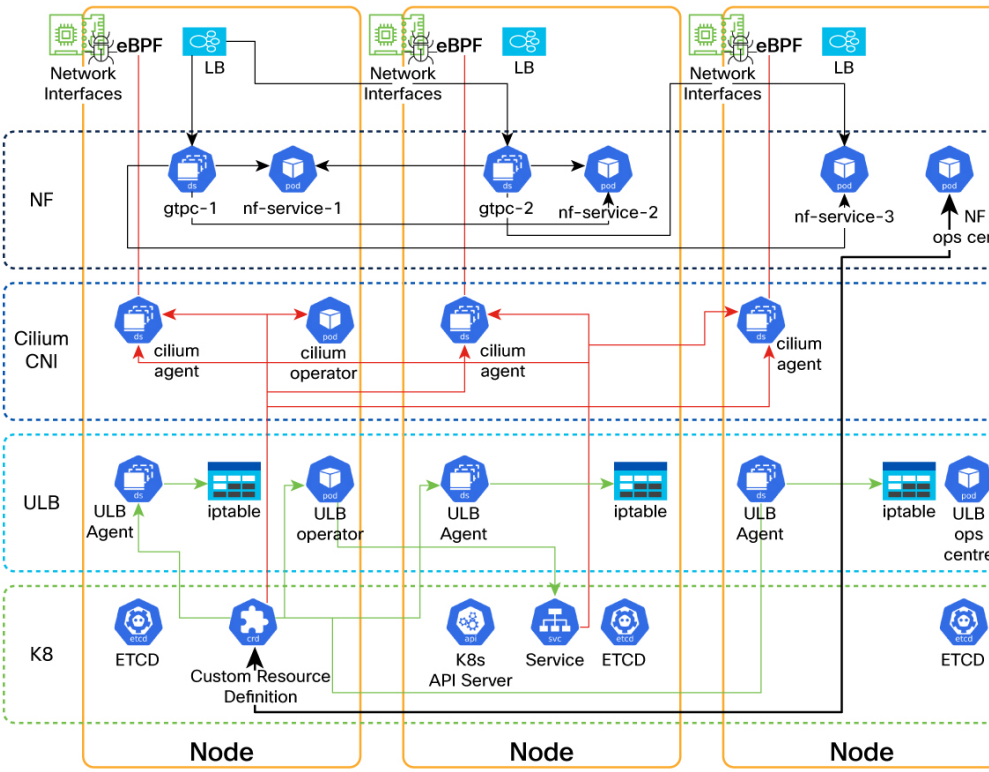
:

- ULB □□□□□□□□□□□

このトピックでは、ULB デプロイメント アーキテクチャについて説明します。これには、技術的な実装と運用の主要コンポーネントに関するワークフロー、要件、および期待される結果などが含まれます。

次の図は、ULB の一般的なデプロイメントモデルを示しています。

1 : Unified Load Balancer



ULB はオンプレミスとクラウドプラットフォームの両方でデプロイでき、アプリケーションの使用と統合を簡素化する統合されたロードバランシングの抽象化を提供します。

- オンプレミス：
  - このサービスは、リーフスパイン型ネットワークアーキテクチャを採用したオンプレミス環境と統合し、BGP および BFD プロトコルを使用して、確実なルートアドバタイズメントと迅速なパス障害検出を実現します。
- クラウド：
  - パブリッククラウドデプロイの場合、サービスは優先順位が等しいルートを公開して、障害イベント時のコンバージェンス高速化を促進し、ワーカーノード間でバランスの取れた通信分散を維持します。

> **注**
>
> クラウドプラットフォームでの ULB のデプロイは、このリリースでは認定されていません。詳細については、シスコのアカウント担当者にお問い合わせください。

ULB

ULB の主要なコンポーネントについて説明し、ロール、要件、運用上の考慮事項に焦点を当てています。Kubernetes クラスターで Pod として実行される主要な ULB コンポーネントについて説明します。

ULB は、Kubernetes クラスターで Pod として実行される次のコンポーネントで構成されます。

- オペレータ：これはコントローラ Pod です。いずれの K8s ノードでも実行または生成できます。オペレータは、ロードバランサおよびイグレス管理ポリシーのカスタムリソース（CR）に関連する、必要な Kubernetes および Cilium リソースを管理します。オペレータは、管理するリソースのデプロイ、スケーリング、および運用タスクを自動化し、システムの状態が CR によって示される望ましい状態と一致するようにします。
- エージェント：ULB エージェントは、K8s クラスター内の各ノードで実行される DaemonSet です。ULB エージェントは、エグレス管理ポリシーの対象となるノードの対応する Pod に対して、各ノードの NAT テーブルと RAW テーブルのエントリを管理します。これにより、通信ルーティングとポリシー適用がノードレベルで正しく適用されます。
- Ops センター：ULB オペレーションズ（Ops）センターは、クラスター内の ULB コンポーネントの構成とアップグレードの中心点として機能します。これは、デプロイメント、モニタリング、およびメンテナンスアクティビティを含む、ULB コンポーネントのライフサイクルを管理するのに役立ちます。
- Load Balancer API：Kubernetes Custom Resource Definition（CRD）として実装され、Kubernetes クラスター内のロードバランサインスタンスの作成、構成、および管理が容易になります。
- Egress Management Policy API：このAPI も CRD として実装され、Pod のエグレスポリシーの定義を適用を有効にし、アウトバウンド通信が制御され、具体的なルールを順守していることを確実にします。

ULB オペレータとエージェントには、Helm チャートと Docker ファイル（イメージ）が含まれています。これにより、Kubernetes 内で製品をインストールおよび管理するための業界標準に準拠できます。Ops Center は、オペレータが製品を総体的に管理するために、安定した CLI/API を提供します。ULB Ops Center は、Helm v3 を使用して Kubernetes APIサーバーと直接通信し、Kubernetes リソースのインストール、アップグレード、クエリ、および削除を行います。

第 3 章

:

- □ □
- □□□□□□□□□□□□□□□□□□□□

オンプレミスデプロイメントの要件と初期構成など、技術的な実装と運用の開始方法について説明します。

ULBデプロイメントの要件と主要なパラメータおよび参照情報についてまとめます。最適なパフォーマンスと互換性を確保するためのシステムの前提条件について説明します。

最適なパフォーマンスと互換性を確保するには、ULBデプロイメントの次のシステム要件を満たす必要があります。

- ULB は、Kubernetes に基づくコンテナ管理プラットフォームにインストールする必要があります。
- Linux カーネルバージョンは、5.4.x 以降である必要があります。ULB eBPF ベースの高速パスをサポートするには、すべての参加ノードがLinux カーネルバージョン 6.8.x 以降で実行されていることを確認します。
- AMD64 または AArch64 アーキテクチャをホストします。
- カーネルでパケット転送を有効にし、各 K8s ノードでリバースパスフィルタリングを無効（つまり、rp_filter - 0）にする必要があります。
- Kubernetes バージョン：v1.28.x 以降。
- Cilium バージョン：1.17.9 以降。
- Ubuntu/Linux kernel Netfilter モジュールで、エグレス関連ソース NAT IP パケットフィルタルールを構成するには、「iptables」という名前のユーザースペース ユーティリティ プログラムをインストールする必要があります。
- Kubernetes がシステムにインストール、構成されていることを確認します。

Cilium CNI のインストール、Isovalent BGP および BFD CRD の有効化、Isovalent egress-gateway CRD の有効化、ロードバランサアルゴリズムの構成など、オンプレミスデプロイメントに必要な初期構成を提供します。

Cilium CNI のインストール、Isovalent BGP および BFD CRD の有効化、Isovalent egress-gateway CRD の有効化、ロードバランサアルゴリズムの構成など、オンプレミスデプロイメントに必要な初期構成を提供します。

SMI Cluster Deployer がセットアップされ、SMI クラスターがすでにデプロイされていることを確認します。SMI クラスターのデプロイメントにより、スタンドアロンの Cilium CNI のインストールが容易になり、Cilium のさまざまな特徴と機能がサポートされます。クラスターのデプロイの詳細については、SMI のドキュメントを参照してください。

Cilium

このトピックでは、SMI Cluster Deployer でサポートされる Cilium 構成の概要について説明します。これらの設定は eBPF ベースのロードバランシングではオプションですが、iptables ベースのデプロイメントで使用できます。また、特定の Cilium 機能が必要な場合に使用できます。

SMI Cluster Deployer では、以下の構成がサポートされています。これらの構成は eBPF ベースのロードバランシングには必要ありませんが、iptables ベースのロードバランシングのデプロイメントに向けて構成できます。また、対応する Cilium 機能が必要な場合に構成できます。

Cilium CNI

cilium CNI のインストールについて説明し、主要なパラメータ、要件、および参照情報についてまとめます。SMI は、Cilium CNI をサポートしています。これは、Cluster Deployer 経由でインストールできます。

SMI は、Cilium CNI をサポートしています。これは、次の CLI コマンドを使用して、Cluster Deployer 経由でインストールできます。

configuration cni type cilium

Isovalent BGP　BFD CRD

Isovalent BGP と BFD CRD の有効化について説明し、主要なパラメータ、要件、および参照情報についてまとめます。BGP および BFD CRD を有効にするには、次の CLI を使用して Cluster Deployer を構成します。

BGP および BFD CRD を有効にするには、次の CLI コマンドを使用して Cluster Deployer を構成します。

configuration cilium enable-bgp true

When IsovalentBgpClusterConfig、IsovalentBgpPeerConfig、IsovalentBgpAdvertisement CR が作成されると、Cilium は、BGP ルータをデプロイし、構成済みピアとの BGP セッションを確立し、サービスの作成または削除に応じて応答でサービス IP をアドバタイズするかアドバタイズを解除します。

Isovalent egress-gateway CRD

Isovalent エグレスゲートウェイ CRD について説明し、主要なパラメータ、要件、および参照情報についてまとめます。次の CLI コマンドを使用したステートフル Pod およびステートレス Pod のエグレスゲートウェイ CRD の有効化について説明します。

Cluster Deployer で次の CLI コマンドを使用して、ステートフル Pod およびステートレス Pod のエグレスゲートウェイ CRD を有効にします。

configuration cilium enable-egress-gateway true

ロードバランサアルゴリズムの構成について説明し、主要なパラメータ、要件、および参考情報についてまとめます。「次の CLI コマンドを使用して Cilium のロードバランサアルゴリズムを構成する」について説明しています。

Cilium のロードバランサアルゴリズムを構成するには、次の CLI コマンドを使用します。

configuration cilium lb-algorithm { maglev | random }

> **注**
>
> ロードバランサアルゴリズムとして「maglev」は使用できますが、現在「random」アルゴリズムの使用が推奨されています。

従来のホストルーティングについて説明し、主要なパラメータ、要件、および参照情報についてまとめます。次のコマンドを使用したデフォルトでは無効になっている従来のホストルーティングの有効化について説明します。

次のコマンドを使用してデフォルトでは無効になっている従来のホストルーティングを有効にします。

configuration cilium enable-legacy-host-routing true

eBPF

ロードバランサアルゴリズムの構成要件について詳述します。

eBPF ベースのロードバランシングの場合は、次の設定を構成します。

- 必要に応じて、クラスターに対して IPv4 と IPv6 の両方のネットワーキングを有効にします。

```
configuration ipv6-mode dual-stack
```

- Cilium をクラスター CNI プラグインとして構成します。

```
configuration cni type cilium
```

- セカンダリ ネットワーク インターフェイスを Pod に接続するために Multus サポートを有効にします。

```
configuration enable-multus true
```

Multus Whereabouts IPAM

ワークロードがプライマリ Cilium Pod ネットワーク外の IP 割り当てとの追加のネットワーク接続を必要とする場合は、次のコマンドを使用します。

```
configuration whereabouts enabled
```

Whereabouts

セカンダリネットワークの IP 割り当てをノード間でスケーリングする必要がある場合、またはノードスライスベースの割り当て動作が必要な場合は、次のコマンドを使用します。

```
configuration whereabouts enable-node-slice-controller true
```

Cilium

複数のアップリンク、ボンド、またはブリッジインターフェイスを持つノード、またはインターフェイス選択が明示的かつ予測可能である必要がある場合は、次のコマンドを使用します。

<device-name1> <device-name2> という名前のデバイスを明示的に含めるには、次の構成を使用します。

```
configuration cilium devices [ <device-name1> <device-name2>  ]
```

Cilium

Cilium が明示的に構成済みインターフェイスのみを使用する際は、次の構成を使用します。

```
configuration cilium force-device-detection false
```

第

4 Ops SMP

章

:

- ULB Ops □□□□
- ULB □□□□□□□□□□□□
- ULB □□□□□□□□
- ULB Ops □□□□□□□□□□□□

Ops Center 経由で ULB をデプロイおよび構成する方法について学習します。この手順では、オンプレミス環境での ULB の前提条件、デプロイメント手順、および構成について説明します。

インストールプロセスには、オンプレミス環境での ULB の設定、必要なコンポーネントの構成、およびインストールの検証が含まれます。

ULB は、次のイメージを使用して CN アプリケーションとしてデプロイされます。

イメージ	説明
ulb.<release>.SPA.tgz	導入環境でソフトウェアをオンボードするために使用される ULB ソフトウェアイメージ。

ULB のデプロイメントと構成の手順には、次の段階が含まれます。

1. SMI Cluster Deployer 経由で ULB をデプロイします。
2. ULB Ops センター経由で設定やカスタマイズを構成します。

   Ops-Center は ConfD CLI に基づいています。ULB 構成には、非常に効率的なロードバランシング機能を提供する機能固有の構成があります。

ULB Ops

ULB Opsセンターについて説明し、その主要なパラメータ、要件、および参照情報についてまとめます。デプロイメント、構成、および管理機能を提供するシステムレベルのインフラストラクチャとしての Ops センターについて説明します。

Ops センターは、次の機能を提供するシステムレベルのインフラストラクチャです：

- 展開に関連付けられた Kubernetes オブジェクト（展開、ポッド、サービスなど）のスケールとプロパティを制御する変数 helm チャート パラメータを提供できる柔軟性を備えた、マイクロ サービスの展開をトリガーするユーザー インターフェイス。
- Kubernetes の構成マップを介して、アプリケーション固有の構成を 1 つ以上のマイクロサービスにプッシュするユーザー インターフェイスです。
- アプリケーション固有の実行コマンド（show コマンドや clear コマンドなど）を発行するユーザー インターフェイスです。これらのコマンド：
  - アプリケーション固有のポッドでいくつかの API を呼び出
  - ユーザー インターフェイス アプリケーションから返された情報を表示する

ULB Ops センターは、クラスター内の ULB コンポーネント（ULB オペレータとエージェント）の構成とデプロイメントの中心点として機能します。ULB オペレータとエージェントには、Helm チャートと Docker ファイル（イメージ）が含まれています。これにより、Kubernetes 内で ULB をインストールおよび管理する方法に関する業界標準に準拠できます。ULB Ops Center は、オペレータが製品を総体的に管理するために、安定した CLI/API を提供します。ULB OpsCenter は、Helm v3 を使用して Kubernetes API サーバーと直接通信し、Kubernetes リソースのインストール、アップグレード、クエリ、および削除を行います。

ULB Ops Center を使用すると、ネイバー探索、レジリエンスと通信コンバージェンスなどの機能を構成できます。

ULB

ULB デプロイメントの前提条件について説明し、SMI 層に ULB をデプロイする前の主要なパラメータ、要件、および参照情報についてまとめます。

SMI 層に UBL をデプロイする前に：

- すべての仮想ネットワーク機能（VNF）が展開されていることを確認します。
- ULB Ops センターと Cloud Native Common Execution Environment （CN-CEE）に対して SMI 同期を実行します。
- 推奨されたポッド展開レイアウトに従って、ノード ラベルが構成されていることを確認します。
- UBL デプロイメント要件に従って外部 VIP を構成します。
- Pod 間通信ロードバランサ向け Istio を有効にします。

ULB

必要な手順や検証アクションなど、ULB のデプロイ手順を示します。

この手順を使用して、準備、デプロイメント、および構成手順を含む、オンプレミス環境で ULB をデプロイします。

この手順では、イメージの処理、Helmチャートの設定、およびオペレータとエージェントの両方のコンポーネントのログ管理を含む、ULB のエンドツーエンドのデプロイメントについて説明します。

Before you begin

オンプレミス環境で ULB のデプロイメントを開始する前に、次の前提条件が満たされていることを確認します。

- Kubernetes クラスターがインストールされ、構成されていることを確認します。
- Linux カーネルバージョンは、5.4.x 以降である必要があります。ULB eBPF ベースの高速パスをサポートするには、すべての参加ノードがLinux カーネルバージョン 6.8.x 以降で実行されていることを確認します。
- カーネルでパケット転送を有効にします。

> **注**
>
> ULB は、デフォルトで有効になっている eBPF データパスを使用してデプロイされ、`lbs-agent` Pod が実行されている各ノードで ULB CNI プラグインを自動的にインストールおよび管理するため、個別のインストールや構成は必要ありません。バックエンド アプリケーション Pod が ULB で生成された NAD を参照すると、Multus はプラグインを呼び出して ULB 管理対象 Pod インターフェイスを作成し、構成された設定を適用します。

> **注**
>
> ULB はアクティブ/スタンバイ冗長性モデルを使用しません。対応するスタンバイ ULB ピアを持つ専用のアクティブな ULB インスタンスがありません。
>
> ULB は、Cilium や Multus に似た、クラスターレベルのネットワーキング製品としてデプロイされます。各 Kubernetes クラスターには、1 つの ULBデプロイメントが必要です。

- リバースパスフィルタリングを無効にします（つまり、「rp_filter = 0」）。
- iptables：ユーザースペース ユーティリティ プログラムをインストールする必要があります。
- Cilium は、SMI/CNDP を使用してインストールされます。

---

1. 専用ソース（https://software.cisco.com/）から ULB image tar ファイルをダウンロードします。

   tar ファイルを抽出して、Docker イメージを解凍します。

2. ULB イメージをの導入準備をして、デプロイします。
   a) Docker イメージをロードし、必要に応じてタグ付けします。
   b) Docker Hub にログインし、イメージをプッシュします。
   c) ULB オペレータとエージェントの Helm チャートがあることを確認します。Docker イメージを使用して、Helm チャートの「values.yaml」ファイルを更新します。
   d) Cluster Deployer から ULB Ops-center を使用して ULB（ULB オペレータとエージェント）をデプロイします。

e) 要件に従って ULB を構成します。
f) パケット転送が有効になっていて、リバースパスフィルタリングが無効になっていることを確認します。
g) ULB オペレータとエージェントのログタグ構成を更新します。

オペレータに次の構成コマンドを使用します。

```
config
logging name lbs_operator.app.app level { debug | error | info | off | trace | warn }
logging name lbs_operator.app.empcrd level { debug | error | info | off | trace | warn }
logging name lbs_operator.app.lbcrd level { debug | error | info | off | trace | warn }
logging name lbs_operator.app.service level { debug | error | info | off | trace | warn }
end
```

エージェントに対して次の構成コマンドを使用します。

```
config
logging name lbs-agent.egress-mgr.app level { debug | error | info | off | trace | warn }
logging name lbs-agent.egress-mgr.iptables level { debug | error | info | off | trace | warn
}
logging name lbs-agent.egress-mgr.iptables-oper level { debug | error | info | off | trace
| warn }
logging name lbs-agent.egress-mgr.lbs-iptables level { debug | error | info | off | trace |
warn }
logging name lbs-agent.egress-mgr.egressmgr level { debug | error | info | off | trace |
warn }
logging name lbs-agent.egress-mgr.listener level { debug | error | info | off | trace | warn
}
end
```

このログタグ構成により、ログ出力を正確に制御でき、効果的なモニタリング、障害対応、パフォーマンス分析、セキュリティ監査、および運用状況の把握が可能になります。管理者は、さまざまなコンポーネントに対して異なるログレベルを指定することで、具体的なニーズに合わせてロギングを調整し、ロードバランシングサービスのスムーズな動作を保証できます。

3. 必要に応じて、LoadBalancer とエグレス管理ポリシー CR を定義して適用します。

Kubernetes のカスタムリソースを含む YAML ファイルのサンプルについては、付録「ロードバランサとエグレス管理ポリシーのカスタムリソース定義の例（68ページ）」を参照してください。

ULB Ops

ULB Ops センターへのアクセス方法について説明し、SSH 経由で接続するための主要なパラメータ、要件、および参照情報についてまとめます。

SSH 経由で SMF Ops センターに接続するには、次のコマンドを使用します。

ssh admin*@ *ops_center_pod_ip* **-p 2024**

SMI Ops センターで設定したものと同じユーザー名とパスワードを使用します。アクセスコントロールのユーザー管理の詳細については、『*CEE* 構成とアドミニストレーション ガイド』を参照してください。

デフォルトでは、Day 0 の構成が ULB にロードされます。

第
5 Unified Load Balancer
章

:

- ULB □□□□
- ULB Pod □□□□□□□□□□

ネットワークの信頼性を確保するために、Unified Load Balancer（ULB）がワーカーノードとバックエンド Pod 間で通信分散を管理する方法について概説します。

Unified Load Balancer（ULB）は、すべてのワーカーノードに受信ネットワーク通信を均等に分散するシスコ独自のネットワーク機能です。入力通信と出力通信に対して一貫したサービス IP アドレスを維持することで、ネットワーク インフラストラクチャ全体で高可用性と信頼性を確保します。

- 通信管理：複数のサーバー間で通信のバランスを取ることで、ボトルネックを防ぎます。
- 拡張性：通信負荷の増加を処理するための水平スケーリングをサポートします。
- パフォーマンス：Direct Server Response（DSR）によって遅延を低減します。

ULB

ULB が着信要求をバックエンドサーバーに分散するために使用するランダム ロード バランサ アルゴリズムの段階について説明します。

ULB は、ランダム ロード バランサ アルゴリズムを使用して、着信要求をバックエンドサーバーに分散します。このアプローチは、バックエンドサーバープールが変更された場合の中断を最小限に抑えるように設計されています。

このアルゴリズムによって、効率的な要求分散と、サーバープールの変更時の影響を最小限に抑えられます。

Summary

このプロセスに関連する主要なコンポーネントには、ULB、バックエンドサーバー、ランダム ロード バランサ アルゴリズムなどが挙げられます。

- ULB：クライアント要求を受信し、バックエンドサーバーへの分散を管理します。
- バックエンドサーバー：クライアントリクエストの実際の処理を取り扱います。
- ランダム ロード バランサ アルゴリズム：ハッシュベースのルックアップテーブルを使用して着信要求をバックエンドサーバーにマッピングします。

このプロセスにより、着信要求がバックエンドサーバーに効率的にマッピングされ、サーバープールに変更があっても既存のリクエストマッピングの中断を最小限に抑えられます。

Workflow

次の段階では、ランダム ロード バランサ アルゴリズムを使用して ULB が要求を分散する方法について説明します。

1. クライアントリクエストが到着すると、ULB はランダムアルゴリズムを使用して適切なバックエンドサーバーを決定します。
   - クライアントの IP アドレス（または別の識別子）を使用して、ハッシュ値が生成されます。
   - ハッシュ値は、事前に計算されたルックアップテーブルを使用してバックエンドサーバーにマッピングされます。

	サーバープールが以下の場合：	次の操作	結合できるフィールド
2.	変更された場合（追加や削除）	アルゴリズムがルックアップテーブルを再計算します	影響を受けるのはごく一部のマッピングのみです。
	引き続き安定している場合	ルックアップテーブルは引き続き一定性を維持します	通信分散は中断することなく続行されます。

ULB Pod

このトピックでは、ULB Pod が使用できない期間中に、システムが通信の継続性を維持し、構成の更新を管理する方法について説明します。データパスがアクティブな間、既存の通信はどのように影響を受けず、ULB Pod がサービスに戻ったときに構成の変更が適用されるようにする調整プロセスについて説明します。

ULB Pod が停止するか一時的に使用できなくなった場合、既存のライブ通信は影響を受けません。通信は、ULB Pod が停止する前にプログラムされていたデータパス状態を使用して転送され続けます。

ただし、ULB Pod が使用できない間に行われた変更変更はすぐには適用されません。これには、新しいアプリケーション Pod の開始、LoadBalancer サービスの追加、更新、または削除、および ULB CLI 構成の追加、更新、削除が含まれます。これらの変更は、ULB Pod が再度実行された後に調整され、適用されます。

第

6 ULB

章

:

- BGP □□□ BFD □□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□
- □□□□□□□□□
- □□□□ □□□□□□□□ □□□□SOR□
- eBPF □□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□
- ULB □□□□□□□□□□□UCP□
- ULB □□□□□□□□□□□□UNP□
- □□□□□□□□□□
- ULB □ SCTP □□□□□□□□□□□□□□□□□□□
- ULB □ GTP-C □□□□□□□□□□□□□□□□□□□
- GTP-C □□□□□□□□□□□□□□□□□□□□□□□
- ULB □ PFCP □□□□□□□□□□□□□□□□□□□

技術的な実装と運用に向けた UBL の機能と構成について説明します。BGP とBFD 経由のレジリエンスと通信コンバージェンス、ネイバー探索、コアダンプの生成、およびサービスアウトバウンドルート（SOR）について説明します。

ULB は、ネットワークのレジリエンスを強化し、迅速な通信コンバージェンスを確保し、ネットワークリソースを効果的に管理するように設計されています。

> **注**
>
> 現在、これらの機能はオンプレミスデプロイにのみ適用されます。

BGP　BFD

ノード障害時のネットワークの安定性と迅速な再ルーティングに焦点を当て、BGP および BFD プロトコルを使用してレジリエンスと通信コンバージェンスをどのように達成するかについて説明します。

ULB は、BGP および BFD プロトコルを採用して、ネットワークの安定性を維持し、ノード障害時に通信を迅速に再ルーティングできるようにします。

技術的な実装と運用に向けたレジリエンスと通信コンバージェンスの仕組みについて詳述します。また、BGP 統合、迅速な障害検出のための BFD 、および BGP の実装に伴う制限についても説明します。

BGP

BGP 統合について説明し、主要なパラメータ、要件、および参考情報についてまとめています。BGP を使用して ULB サービスをルーティングサーバーと統合し、リアルタイムのルートアドバタイズメントを容易にする方法について説明します。

BGP を使用して ULB サービスをルーティングサーバーと統合し、リアルタイムのルートアドバタイズメントと通信の再配布を容易にします。これは、Cilium のサポートにより実装され、バージョン 1.17 以降が必要です。特定の CR が作成されると、Cilium は、BGP ルータをデプロイし、BGP セッションを管理します。

BFD

BFD によりルーター間のパス障害を迅速に検出し、迅速なフェイルオーバーを実現して、通信の中断を最小限に抑える方法について説明します。

BFD は、ルータ間のパス障害を迅速に検出し、迅速なフェールオーバーと最小限の通信中断を実現するために使用されます。

BGP

一貫した実装をサポートするための BGP 実装に関する推奨事項と制約について説明します。

- タイマーなどの構成を更新すると、BGP セッションが再起動します。
- Cilium の制限により、アプリケーションは通信を分離することができず、代わりにすべての通信に BGP インターフェイスを使用します。
- BGP サーバーモードが構成されていても、Cilium の問題によってサーバーとして機能することはできません。

BGP

BGP の構成方法について説明します。

ネットワークでレジリエンスと通信 コンバージェンスをサポートするには、BGP 機能を有効にします。

1. BFD ピアプロファイルを定義する
2. BGP ピアプロファイルを設定する
3. BGP ルータを構成する

BFD

必要な手順と検証アクションを含む、BFD ピアプロファイルの定義方法について説明します。この手順では、制御およびエコーパケットのタイマー、最大再試行設定などの BFD 機能の構成について説明します。

BFD ピアプロファイルは、制御パケットのタイマー、エコーパケットのタイマー、BFD メッセージの最大再試行などの BFD 機能の構成を提供します。

次の手順では、ULB で BFD ピアプロファイルと関連パラメータを定義する方法について説明します。

---

1. グローバル構成モードで BFD ピアプロファイルを構成します。

   Example

   ```
   lbs(config)# bfd-peer-profile bfd-prof1
   ```

2. BFD 制御パケットの送受信用のタイマーを秒単位で指定します。

   BFD 制御パケットの rx-timer の有効な範囲は、10 ～ 60000 ミリ秒です。デフォルト値は、10 ミリ秒です。

   BFD 制御パケットの tx-timer の有効な範囲は 10 ～ 60000 ミリ秒です。デフォルト値は、250 ミリ秒です。

   Example

   ```
   lbs(config-bfd-peer-profile-bfd-prof1)# control-msg rx-timer 10 tx-timer 250
   ```

3. BFD エコーパケットを送受信するためのタイマーを秒単位で指定します。

   エコーパケットの rx-timer および tx-timer の有効な範囲は 10 ～ 60000 ミリ秒です。デフォルト値は 90 ミリ秒です。

   Example

   ```
   lbs(config-bfd-peer-profile-bfd-prof1)# echo-msg rx-timer 90 tx-timer 90
   ```

4. BFD セッションダウンをマーク付けする BFD メッセージの最大再試行回数を指定します。

   有効な範囲は 1 ～ 255 です。デフォルト値は 3 です。

   Example

   ```
   lbs(config-bfd-peer-profile-bfd-prof1)# max-retry 3
   ```

5. 構成を保存してコミットします。

   Example

   ```
   lbs(config-bfd-peer-profile-bfd-prof1)# end
   ```

   このコマンドを使用すると、構成をコミットまたは無視することができます。**[はい（Yes）]** と入力すると、構成を保存または変更できます。

BGP

必要な手順と検証アクションを含む、BGP ピアプロファイルの設定方法について説明します。この手順を使用して、BGP セッションパラメータを、タイマー、BGP アドバタイズメントリソース関連機能を構成します。

BGP ピアプロファイルは、再試行タイマー、キープアライブタイマーなどの BGP セッション関連機能の構成オプションを提供します。このプロファイルは、グレースフルリスタート、EBGP マルチホップをサポートし、BGP アドバタイズメントリソースに使用するラベル機能をアドバタイズします。このプロファイル構成には、BFD 機能を有効にするためのサポートも含まれています。

この手順を実行すると、ULB で BGP ピアプロファイルと関連するパラメータを定義できます。

---

1. グローバル構成モードで BGP プアプロファイルを構成します。

   Example

   ```
   lbs(config)# bgp-peer-profile bgp-prof1
   ```

2. BFD ピアプロファイルを構成する

   Example

   ```
   lbs(config)# bfd-peer-profile bfd-prof1
   ```

3. 必要に応じて、BGP セッションに関連するタイマーを定義します。

   1 : BGP

CLI コマンド	説明	範囲	デフォルト値
connection-retry-timer	ピアのダウンまたは Pod の再起動を検出した場合に接続を再試行するためのタイマー	1 ～ 300	12
graceful-restart restart-timer	BGP ピアのグレースフルリスタート機能を有効化します	1 ～ 4095	無効化
hold-timer	ピアのダウンを検出した場合に、アド払い図されたルートを保持するために BGP ピアとネゴシエートするタイマー	3 ～ 300	3
keepalive-timer	BGP ピアにキープアライブメッセージを送信するタイマー期間	1 ～ 300	1

> **注**
>
> タイマーは無効にできません。デプロイメントのニーズに応じてのみ調整できます。デフォルトでは、グレースフル リスタートはディセーブルになっています。

4. BGP ピアプロファイルの一致ラベルキー/値構成を定義します。このラベルは、BGP アドバタイズメント リソースに使用されます。

   Example

   ```
   lbs(config-bgp-peer-profile-bgp-prof1)# advertise-label key advertise value
   bgp
   ```

5. BGP メッセージの EBGP マルチホップカウントを構成します。

   Example

   ```
   lbs(config-bgp-peer-profile-bgp-prof1)# ebgp-multihop 5
   ```

6. 構成を保存してコミットします。

BGP

必要な手順や検証アクションなど、BGP ルータの構成方法について説明します。この手順では、仮想ルータ、BGP ピアリング、およびサービス IP アドバタイズメントの構成について説明します。

ルータは、仮想ルータ、BGP ピアリング、およびサービス IP アドバタイズメントの構成をサポートします。また、ノードを BGP ルータとして構成することもできます。

この手順を使用して、BGP ピアプロファイル、ノードセレクタ、およびサービス IP など BGP ルータとしてルータを構成します。

---

1. グローバル構成モードを開始します。

   Example

   ```
   [cluster1/data] lbs# config
   ```

2. BGP ネイバーの ASN（自律システム番号）を指定して、BGP ルータを構成します。

   Example

   ```
   lbs(config)# router bgp 65000
   ```

3. プロファイル名を指定して、BGP ピアプロファイルを構成します。

   Example

   ```
   lbs(config-router-bgp-65000)# bgp-peer-profile bgp-prof1
   ```

4. キーと値のペアを使用してノードセレクタを指定します。

Example

```
lbs(config-bgp-peer-profile-bgp-prof1)# node-selector key feature value bgp
```

5. サービス名と IP アドレスを構成します。

Example

```
lbs(config-bgp-peer-profile-bgp-prof1)# service name web-service ip 192.168.1.1
```

6. ピア名、AS 番号、およびピア IP を指定します。

Example

```
lbs(config-bgp-peer-profile-bgp-prof1)# peer-name peer_name
lbs(config-bgp-peer-profile-bgp-prof1)# peer-as AS peer-ip IP_address
```

7. 構成を保存してコミットします。

Example

```
lbs(config-bgp-peer-profile-bgp-prof1)# end
```

次の構成例を使用して、BGP および BFD 設定を構成します。

```
config
        bfd-peer-profile bfd-prof1
        control-msg rx-timer 10 tx-timer 250
        echo-msg rx-timer 90 tx-timer 90
        max-retry 3
        exit
        bgp-peer-profile bgp-prof1
       connection-retry-timer 12 hold-timer 3 keepalive-timer 1 ebgp-multihop
 255
               QAA
        bfd-peer-profile bfd-prof1
        exit
        router bgp 65000
        bgp-peer-profile bgp-prof1
        node-selector key feature value bgp
        service name pfcp ip 7.7.7.7
        service name s11 ip 8.8.8.8
        peer-name leaf1
        peer-as 63141 peer-ip 120.20.20.120
        exit
        peer-name leaf1-v6
        peer-as 63141 peer-ip 2006::98
        exit
        peer-name leaf2
        peer-as 63141 peer-ip 130.30.30.130
        exit
        peer-name leaf2-v6
        peer-as 63141 peer-ip 3006::98
```

```
        exit
        exit
```

技術的な実装と運用に向けたネイバー探索について詳述します。ネイバー探索の仕組み、更新間隔、インターフェイス、およびノードラベルの構成、およびネイバーエントリのモニタリングについて説明します。

ネイバー探索は、ネイバー IP の ARP エントリが断続的に削除される問題に対処することで、安定したネットワーク接続を保証します。この機能は、これらのエントリを維持することで、通信のドロップを防ぐためにサポートされています。

シーケンス、依存関係、予想される結果など、ネイバー探索の仕組みを示すワークフローについて概説します。ネイバー探索により、管理者が CLI を使用してネイバー IP とインターフェイスをどのように構成できるかについて説明します。

ネイバー探索を使用すると、管理者は CLI を使用してネイバー IP とインターフェイスを構成できます。ラベルを使用して、プロセスを具体的なノードに合わせて調整できます。

- 管理者は、それぞれに独自の IP アドレスを持つ複数のインターフェイスを構成できます。
- 検出は、指定されたラベルに一致するノードか、ラベルが設定されていない場合はすべてのノードに限定されます。

Summary

ネイバー探索には、次の主要なアクターとコンポーネントが含まれます。

- 管理者：CLI を使用してネイバー IP とインターフェイスを構成します。
- CLI：ネイバー探索パラメータを設定するコマンドを提供します。
- ノード：構成されたラベルに基づいてネイバー探索に参加します。

管理者は、CLI を使用してネイバー探索を構成します。この構成は、指定されたラベルに一致するノードに動的に適用されます。ラベルが設定されていない場合、すべてのノードが検出プロセスに含まれます。

Workflow

次の段階では、ネイバー探索をノード間でどのように構成し、適用するかについて説明します。

1. 管理者は、CLI を使用してネイバー IP とインターフェイスを構成し、必要に応じてラベルを指定します。
   - CLI にアクセスし、ネイバー探索構成モードを開始します。
   - インターフェイスを指定し、各ネイバーに IP アドレスを割り当てます。
   - 必要に応じて、特定のノードに検出を制限するラベルを構成します。

   ネイバー探索は、指定されたラベルに一致するノードに制限されます。ラベルが構成されていない場合、探索はすべてのノードで実行されます。

   構成の変更は、ノード全体で動的に適用され、必要に応じて、ネイバーエントリが更新されます。

必要な手順や検証アクションなど、更新間隔、インターフェイスおよびノードラベルを構成する手順を示します。

次の手順では、更新間隔と、関連付けられた IP を使用して複数のインターフェイスを構成する方法について説明します。

1. グローバル構成モードを開始します。

   Example

   ```
   [cluster1/data] lbs# config
   ```

2. ネイバー探索の詳細を構成します。

   Example

   ```
   lbs(config)# neighbor-discovery
   ```

3. ネイバーの更新間隔を秒単位で設定します。

   更新間隔は、1 ～ 3600 秒の整数で設定します。デフォルト値は 30 秒です。

   Example

   ```
   lbs(config)# neighbor-discovery refresh-interval 45
   ```

4. BGP ネイバーが到達可能なインターフェイスを構成します。次に、関連する IPv4 および IPv6 ネクストホップアドレスを指定します。

   > **注**
   >
   > それぞれが複数の IP アドレスを持つ複数のインターフェイスを設定できます。

   Example

   ```
   lbs(config)# neighbor-discovery interface v500
                lbs(config*)# neighbor-discovery interface v500 nexthop 7.7.7.3
               lbs(config*)# neighbor-discovery interface v500 nexthop 7:7:7::3
   ```

5. キーと値のペアを使用してノードセレクタを指定します。

   Example

   ```
   lbs(config)# neighbor-discovery node-selector key feature value bgp
   ```

   ネイバー探索は、構成されたノードラベルのいずれかに一致する各ノードで実行されます。ラベルが構成されていない場合、ネイバー探索はすべてのノードで実行されます。

6. 構成を保存してコミットします。

Example

```
lbs(config)# end
```

次のサンプル構成を使用して、ネイバー探索サービスを構成します。

```
config
        neighbor-discovery
        refresh-interval 45
        interface v500
        nexthop 7.7.7.3
        nexthop 7:7:7::3
        exit
        interface v501
        nexthop 6.6.6.3
        nexthop 6:6:6::3
        exit
        interface v502
        nexthop 5.5.5.3
        nexthop 5:5:5::3
        exit
        node-selector
        key frontend value protocol
        key test1 value value1
        exit
        exit
```

ネイバーエントリをモニターする方法について説明し、主要なパラメータ、要件、および参照情報についてまとめます。

次の **show** コマンドを使用してエントリを確認します。

```
lbs# show running-config neighbor-discovery
neighbor-discovery
refresh-interval 10
node-selector
key key1 value value1
key key2 value value2
exit
interface v500
nexthop 7.7.7.3
nexthop 7:7:7::3
exit
interface v501
nexthop 6.6.6.3
nexthop 6:6:6::3
exit
exit
```

ULB を使用して、**ULB** エージェントおよびオペレータプロセスのコアダンプファイルの生成を可能にする方法について説明します。これには、デバッグのための保存場所や収集手順などが含まれます。

ULB を使用すると、予期しないクラッシュまたは再起動が発生した場合に、ULB エージェントおよびオペレータプロセスのコアダンプファイルを生成できます。これらのコアダンプは、障害発生時のプロセスのメモリ状態をキャプチャし、デバッグのための貴重な情報を提供します。

コアダンプファイルは、プロセスが実行されていたノードの /var/lib/systemd/coredump/ directory に保存されます。

ネットワーク管理者は、各 K8s ノードから手動でコアダンプファイルを収集し、デバッグツールを使用してファイルを分析できます。

SOR

サービス アウトバウンド ルート（SOR）機能について説明しています。これには、技術的な実装、運用上の利点、制限事項、および Kubernetes クラスター内での SOR の仕組みなどが挙げられます。

サービス アウトバウンド ルート（SOR）機能は、Kubernetes クラスター内のアプリケーション向けアウトバウンド通信を分離するポリシーベースのメカニズムを提供します。これにより、管理者はカスタムリソース（CR）を使用してカスタムルーティングポリシーを定義できるため、静的ルートの手動設定が不要になります。SOR CR は、Direct Server Return（DSR）のロードバランサ（LB）CR および出力通信のエグレス管理ポリシー（EMP）CR によって参照されます。

SOR

SOR を実装する利点について説明します。ここでは、ネットワーク構成の簡素化、強化されたセキュリティ、コンプライアンスの改善、およびアウトバウンドルーティングのリソース最適化に焦点を当てます。

- ネットワーク構成の簡素化：アウトバウンドルーティングの構成を自動化し、手動作業とエラーの可能性を削減します。
- 強化されたセキュリティ：アウトバウンド通信を分離し、よりきめ細かい制御とセキュリティポリシーを有効にします。
- コンプライアンスの改善：アウトバウンド通信が定義されたパスをたどることを保証することで、コンプライアンス要件の遵守を促進します。
- リソース最適化：静的ルートの管理に関連する運用オーバーヘッドを削減します。

SOR

SOR の実装時にこれらのガイドラインを使用することで、バックエンド アプリケーション Pod が、異なるアプリケーションやインターフェイスに子熱のターゲットポートを使用できるようになり、昨日の適用性やプロトコルのサポートについて理解できるようになります。

- バックエンド アプリケーション Pod は、それぞれの LB または EMP CR で異なるアプリケーションまたはインターフェイスに個別のターゲットポートを使用する必要があります。
- この機能は、ターゲットポートが定義されている LB/EMP CR にのみ適用されます。
- 現在、SOR は UDP 通信にのみ適用されます。

SOR

ULB の SOR 機能がルーティングテーブルとルールを使用してアウトバウンド通信を分離する方法について説明し、アウトバウンド通信管理の攻撃者、ワークフロー、および期待される結果について詳しく説明します。

ULB の SOR 機能は、ルーティングテーブルとルールを使用してアウトバウンド通信を分離します。アウトバウンド通信の分離が必要なアプリケーションの場合、ネットワーク管理者は、ネクストホップ（ゲート

ウェイ IP アドレス）とデバイス（ネットワーク インターフェイス）を指定する SOR CR を作成します。ロードバランサとエンドポイントのカスタムリソースは、この SOR CR を参照してルーティングを管理します。

- ネットワーク管理者は、ネクストホップとデバイスを指定する SOR CR を作成します。
- ULB がルーティング テーブル ID を割り当て、ルートを構成します。
- ロードバランサとエンドポイントのカスタムリソースは、SOR CR を参照します。

Summary

このプロセスに関与する主要なコンポーネントは次のとおりです。

- ネットワーク管理者：アウトバウンド通信分離のために SOR CR を作成および管理します。
- ULB（Unified Load Balancer）：ルーティングテーブル ID を割り当て、SOR CR に基づいてルートを構成します。
- ロードバランサとエンドポイントのカスタムリソース：SOR CR を参照して、関連付けられた Pod のルーティングを管理します。

このプロセスにより、Pod からのアウトバウンド通信が指定されたゲートウェイ経由でルーティングされるようになり、アウトバウンド要求の制御された効率的なルーティングが可能になります。

Workflow

次の段階では、SOR 機能が SOR CR とルーティングテーブルを使用してアウトバウンド通信ルーティングを管理する方法について説明します。

1. ネットワーク管理者は、ネクストホップ（ゲートウェイ IP アドレス）とデバイス（ネットワーク インターフェイス）を指定する SOR CR を作成します。
2. ULB は一意のルーティング テーブル ID を割り当て、指定されたネクストホップとデバイスを使用してルートを構成します。
3. ロードバランサとエンドポイントのカスタムリソースは、SOR CR を参照して、関連付けられた Pod のルーティングを管理します。
4. IP ルールは、Pod 通信が対応するルーティングテーブルを使用するように設定されます。Pod の更新または削除により、これらの IP ルールの変更または削除がトリガーれます。Iptables は、ロードバランサのイベント用に調整されています。
5. Pod がアウトバウンド通信を生成すると、SOR CR にリンクされたルーティングテーブルにより、通信が構成されたゲートウェイ経由で送信されるようになります。

eBPF

ULB は、デュアル eBPF ベースのメトリック機能をサポートし、異なる構成方法でサービス通信と運用データパスイベントをきめ細かく可視化します。

ULB は、2 つの独立した eBPF ベースのメトリック機能を提供します。サービスのパケットおよびバイトのメトリックは、各サービスとバックエンド Pod のパケット数とバイト数をカウントします。ULB 運用データパスメトリクスでは、インストゥルメント化された受信、セッション開始、フォワーディングパス、および eBPF の障害イベントが、`ulb_dp_*` メトリックファミリを通じて公開されます。各機能には個別の構成と有効化メソッドがあります。

eBPF

この機能は、eBPF テクノロジーを使用して、ULB サービス内のイングレスおよびエグレス パケットとバイトメトリックを追跡することによりオブザーバビリティを強化します。

ULB は、Extended Berkeley Packet Filter（eBPF）テクノロジーを使用して、ULB サービスでデータパスを詳細に可視化します。これは、入力通信と出力通信関連のメトリックを収集し、管理者は、パフォーマンスの問題と障害対応の問題を監視できるようになります。

> **注**
>
> この機能は、CGROUP（Control Group）v2 をサポートする Ubuntu カーネルバージョンのみをサポートします。

データパスメトリクスを使用することの利点について説明します。これには、デバッグの強化、パフォーマンス監視、ネットワーク運用における可視性の向上が含まれます。

データパスメトリクスを使用することで、ULB は次のような利点を提供します。

- デバッグの強化：詳細なデータパスメトリクスを提供し、ネットワーク問題の迅速な特定と解決を容易にします。
- パフォーマンス監視：通信スループットをリアルタイムで監視できるため、管理者はボトルネックを特定し、パフォーマンスを最適化できます。
- 可視性の向上：サービスの通信パターンに関する分析情報を提供し、キャパシティプランニングおよびリソース割り当てに役立ちます。

データパスメトリックのサポートに関する制限事項について説明します。これには、システムリソースでの eBPF モニタリングの影響、モニタリングオーバーヘッドの考慮事項などが挙げられます。

eBPF モニタリングは、オーバーヘッドを引き起こす可能性があるため、システムリソースの使用状況をモニタリングすることが重要です。

シーケンス、依存関係、期待される結果など、データパスメトリックの収集とレポートの仕組みについて説明します。

この機能は、ロードバランササービス内のネットワーク通信フローを理解するのに役立ちます。

Summary

このプロセスには、ネットワーク通信メトリックを収集し、報告するために連携する複数のアクターとコンポーネントが含まれます。

- eBPF：通信特性を受動的に監視し、各サービスと Pod のパケットとバイトをカウントします。
- ULB エージェント：eBPF からカウントを定期的に収集し、メトリックとして使用できるようにします。

これらのメトリックは、デバッグおよびパフォーマンス分析に使用できます。

Workflow

これらの段階では、ロードバランササービス内のデータパスメトリックの収集とレポートについて説明します。

1. この CLI コマンドを使用して、メトリック収集機能を有効にします。

   metrics service-stats

有効にすると、モニタリングプログラム（eBPF）が通信特性の観察を開始します。

機能がアクティブ化され、メトリック収集の準備が整います。

2. eBPF は通信を観察し、各サービスと Pod のパケットとバイトをカウントします。

   eBPF はバックグラウンドで動作し、パフォーマンスに影響を与えることなくデータを収集します。

   通信メトリックは、サービスおよび Pod ごとに累積されます。

3. ULB エージェントは、eBPF からカウントを収集し、メトリックとして使用できるようにします。

   メトリックは定期的に収集され、デバッグおよびパフォーマンス分析のためにアクセス可能になります。

eBPF

必要な手順と検証アクションを含む、データパスメトリック収集の有効化方法について説明します。この手順では、metrics service-stats コマンドについて説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

---

1. グローバル構成モードを開始します。
2. このコマンドを実行して、データパスメトリックの収集機能を有効にします。

   metrics service-stats

   > **注**
   >
   > デフォルトでは、この CLI は無効になっています。

3. 変更を確定します。

---

eBPF ULB

eBPF に基づいて ULB 運用データパスメトリックを導入し、その主な対象範囲について説明します。ULB データパスの運用統計の有効化または無効化、ULB クラスターポリシーの構成、構成の確認、および IP プールを使用した ULB クラスターポリシーの構成について説明します。

ULB 動作データパスメトリックにより、通信が ULB eBPF データパスでどのように処理されるかを可視化できます。受信した通信が構成済みの ULB サービスに到達しているかどうかを検出し、セッション確立アクティビティを観察し、ULB が選択した転送パスを理解し、eBPF 処理の失敗を特定します。

メトリックは、累積的な観測内容を提供し、現在アクティブなセッションまたはアソシエーションの実際の数を表すものではありません。デフォルトの 300 秒のエクスポータ更新間隔では、新しく記録されたアクティビティがエクスポートされるまでに最大 5 分かかることがあります。統計マップは、16,384 のサービス/インターフェイス エントリをサポートします。

公開されるメトリックファミリは次のとおりです。

- ulb_dp_rx_total

構成された ULB サービスのパケットが、構成されたノードインターフェイス上の ULB eBPF 処理パスに到達していることを確認します。ulb_dp_rx-total メトリックは、サービスおよびレポートインターフェイスを識別し、通信が外部ピアから到着したか、別の参加ノードから到着したかを示します。

- ulb_dp_session_total

ulb_dp_session_total 経由で、新しいセッション、接続、フローを識別します。たとえば、新しいアソシエーションに対して処理された SCTP INIT が、このカウンターに加算されます。このメトリックは、開始イベントを報告します。

- ulb_dp_forward_total

ULB が ulb_dp_forward-total 経由でサービス通信を転送する方法を理解します。管理者は、ローカル バックエンド アプリケーション Pod 、別の参加ノード上のバックエンド、または別のサポートされている転送パスへの転送アクティビティを区別できます。この情報は、通信が、予想される ULB パスに従っているかどうかを確認するのに役立ちます。

- ulb_dp_ebpf_failures

ulb_dp_ebpf_failures 経由で eBPF データパス処理エラーを特定します。エラーは処理理由によってグループ化されるため、パケット検証、サービスルックアップ、バックエンド解決、転送、またはその他の監視対象の ULB 操作中に問題が発生したかどうかを判断するのに役立ちます。これらのカウンタは、ULB データパスによって検出された障害を報告しますが、バックエンド Pod 内のアプリケーションレベルのエラーは報告しません。

ULB

必要な手順や検証アクションなど、ULBデータパス運用統計を有効または無効にする手順を示します。運用データパスのカウントを制御するクラスター範囲の UlbStats リソースについて説明します。

クラスター範囲の UlbStats リソースは、運用データパスのカウントを制御します。これらの運用データパスの統計は、デフォルトでは無効になっています。リソースが適用されると、ULB は参加している ULB エージェント Pod に構成を伝播します。構成された eBPF プログラムは、データパス統計を更新し、エクスポータはPrometheus と Grafana の結果のメトリックシリーズを定期的に公開します。

---

1. ulb-stats という名前のサポートされているシングルトン UlbStats リソースを構成します。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: UlbStats
metadata:
  name: ulb-stats
spec:
  dataPathStats:
    enable: true
```

2. 機能を無効にするには、spec.dataPathStats.enable を false に設定し、リソースを再度適用します。リソースが [適用済み（Applied）] フェーズを報告することを確認します。

```
kubectl get ulbstats ulb-stats
```

> **注**
>
> 既存のメトリック service-stats CLI は、別のサービスのパケットおよびバイトのコレクタを制御するもので、UlbStats を置き換えません。

技術的な実装および運用に関するその他の構成について説明します。これには、Cilium BPF マップの構成方法や、Cilium Hubble 統計の有効化方法が含まれます

cilium BPF

CLI コマンドを使用してマップサイズを編集して Cilium BPF マップを構成する方法について説明します。

この手順を使用して、適切な CLI コマンドを入力して、Cilium の BPF マップサイズを編集します。

すべてのマップサイズを構成する必要はありません。セットアップに必要な数のみを調整します。デフォルトでは、この CLI は値を設定しないため、Cilium BPF マップサイズは、Cilium の標準に従います。この CLI で変更をコミットすると、Cilium Pod が再起動します。値が、定義された制限内にあることを確認します。そうしないと、コミットは失敗します。

構成モードを開始し、次のコマンドを使用して、目的の BPF マップサイズを設定します。

Example

```
config
             [no] cilium bpf auth-map-max <value>
             [no] cilium bpf fragments-map-max <value>
             [no] cilium bpf lb-affinity-map-max <value>
             [no] cilium bpf lb-maglev-map-max <value>
             [no] cilium bpf lb-maglev-table-size <value>
             [no] cilium bpf lb-map-max <value>
             [no] cilium bpf lb-rev-nat-map-max <value>
             [no] cilium bpf lb-service-backend-map-max <value>
             [no] cilium bpf lb-service-map-max <value>
             [no] cilium bpf lb-source-range-map-max <value>
             [no] cilium bpf map-dynamic-size-ratio <value>
             [no] cilium bpf neigh-global-max <value>
             [no] cilium bpf node-map-max <value>
             [no] cilium bpf policy-map-max <value>
             exit
```

- すべてのマップサイズを構成する必要はありません。セットアップに必要な数のみを調整します。
- デフォルトでは、この CLI は値を設定しないため、Cilium BPF マップサイズは、Cilium の標準に従います。
- この CLI で変更をコミットすると、Cilium Pod が再起動します。
- 値が、定義された制限内にあることを確認します。そうでない場合、コミットは失敗します。

これらのコマンドに関連する最大サイズの参考資料として、次の表を使用します。

コマンド	説明	範囲
cilium bpf auth-map-max	BPF 認証マップの最大サイズ	65536 ～ 4194304
cilium bpf fragments-map-max	BPF フラグメントマップの最大サイズ	4096 ～ 65536
cilium bpf lb-affinity-map-max	BPF ロードバランサのアフィニティマップの最大サイズ	65536 ～ 4194304
cilium bpf lb-maglev-map-max	BPF ロードバランサ Maglev マップの最大サイズ	65536 ～ 4194304
cilium bpf lb-maglev-table-size	BPF ロードバランサ Maglev ルックアップテーブルのサイズ。	MAGLEV_TABLE_SIZE_251 MAGLEV_TABLE_SIZE_509 MAGLEV_TABLE_SIZE_1021 MAGLEV_TABLE_SIZE_2039 MAGLEV_TABLE_SIZE_4093 MAGLEV_TABLE_SIZE_8191 MAGLEV_TABLE_SIZE_16381 MAGLEV_TABLE_SIZE_32749 MAGLEV_TABLE_SIZE_65521 MAGLEV_TABLE_SIZE_131071
cilium bpf lb-map-max	BPF ロードバランサマップの最大サイズ	65536 ～ 4194304
cilium bpf lb-rev-nat-map-max	BPF ロードバランサのリバース NAT マップの最大サイズ。	65536 ～ 4194304
cilium bpf lb-service-backend-map-max	BPF ロードバランササービスのバックエンドマップの最大サイズ。	65536 ～ 4194304
cilium bpf lb-service-map-max	BPF ロードバランサ サービスマップの最大サイズ。	65536 ～ 4194304
cilium bpf lb-source-range-map-max	BPF ロードバランサソース範囲の最大サイズ	65536 ～ 4194304
cilium bpf map-dynamic-size-ratio	使用可能なメモリの量に対する BPF マップの動的なサイジングに使用される比率	0.0 ～ 1.0
cilium bpf neigh-global-max	BPF ネイバーマップのグローバル最大サイズ。	65536 ～ 4194304
cilium bpf node-map-max	BPF ノードマップの最大サイズ	16384 ～ 4194304

コマンド	説明	範囲
cilium bpf policy-map-max	BPF ポリシーマップの最大サイズ。	8192 ～ 16384

Cilium Hubible

必要な構成手順と、デフォルトの動作と Pod の再起動に関する重要な注意事項を含め、Cilium Hubbble 統計を有効にする方法について説明します。

構成モードに入り、データ収集のために Hubbble 統計を有効にします。

Example

```
config
metrics collection
exit
```

> **注**
>
> デフォルトでは、この CLI は無効になっています。Hubbble 統計を有効にすると、Cilium Pod が再起動します。

ULB

必要な手順や検証アクションなど、ULBデータパスロギングを有効または無効にする手順を示します。これは、個別のクラスター全体の障害対応制御です。

データパスロギングは、個別のクラスター全体の障害対応制御です。ULBデータパスロギングは、デフォルトでは無効に設定されています。ulb-logging で spec.dataPathLogging.enable を true に設定すると有効になりき、false に設定すると無効になります。この設定は、eBPF データパス診断ロギングにのみ影響します。通常のコンポーネントのロギングは引き続き使用できます。

1. ulb-logging という名前のサポートされているシングルトン UlbLogging リソースを構成します。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: UlbLogging
metadata:
  name: ulb-logging
spec:
```

```
  dataPathLogging:
    enable: true
```

2. リソースが [適用済み（Applied）] フェーズを報告することを確認します。

```
kubectl get ulbloggings ulb-logging
```

> **注**
>
> データパスロギングは、簡潔で制御された障害対応のみを目的としています。有効にすると、大量の診断出力が生成され、特に通信が多い場合、CPU 使用率が大幅にスパイクする可能性があります。通常の実稼働操作中には、データパスロギングを有効のままにしないでください。問題を再現して必要な情報を収集するために必要な時間だけ有効にし、その後すぐに無効にします。

ULB UCP

ULB クラスターポリシー（UCP）について説明し、主要な概念、要件、および運用上の考慮事項に焦点を当てます。ULB クラスターポリシーは、ULB がサービス通信を受信するノードインターフェイスを識別します。

ULB クラスターポリシーは、ULB がサービス通信を受信するノードインターフェイスを識別します。サービス側ネットワークで使用される物理インターフェイスまたは VLAN サブインターフェイスの名前を構成します。ULB は、クラスター全体にこの構成を適用し、使用可能で起動している構成済みの各インターフェイスでイングレス eBPF 処理をアクティブにします。

ポリシーはインターフェイスを識別しまが、作成されません。構成済みのインターフェイスが現在既存しない場合、または、1 つ以上のモードが停止している場合でもポリシーをコミットできます。ポリシーは引き続き有効ですが、影響を受けるノードはそのインターフェイス経由で ULB サービス通信を処理できません。インターフェイスが使用可能になり、起動すると、ULB が変更を検出し、自動的に使用を開始します。管理者はポリシーを再コミットする必要はありません。

サービスの可用性については、構成済みのインターフェイスが存在し、ULB サービス通信を受信することが予想されるすべてのノードで稼働していることを確認します。既存しないインターフェイスまたは使用できないインターフェイスは、そのノードの処理に影響しますが、クラスター全体のポリシーを無効にすることはありません。

CLI の構成がコミットされると、ULB は自動的にクラススコープのポリシーを作成または更新します。管理者は、Kubernetes リソースを作成したり、生成された名前を指定したりしません。CLI の構成からインターフェイスを削除すると、ULB はそのインターフェイスを使用しなくなります。すべてのインターフェイスエントリを削除すると、生成されたポリシーも削除されます。

ULB

必要な手順や検証アクションなど、ULB クラスターポリシーを構成する手順を示します。「ULB がサービス通信に使用するノードインターフェイスを構成する」について説明します。

ULB がサービス通信に使用するノードインターフェイスを構成します。

```
[cluster1/data] lbs# config
[cluster1/data] lbs(config)# cluster-policy
[cluster1/data] lbs(config-cluster-policy)# node-interface name ens192.1001
[cluster1/data] lbs(config-cluster-policy)# exit
[cluster1/data] lbs(config)# end
```

必要な手順や検証アクションなど、構成の検証手順を示します。ULB ノードインターフェイスの動作状態について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

1. ULB が目的のインターフェイス名でポリシーを作成したことを確認します。

```
kubectl get ulbclusterpolicy global-ulb-cluster-policy -o yaml
Sample O/p:
kubectl get ulbclusterpolicy global-ulb-cluster-policy -o yaml
apiVersion: lbs.cisco.com/v1alpha1
kind: UlbClusterPolicy
metadata:
  name: global-ulb-cluster-policy
spec:
  nodeInterfaces:
  - name: ens192.1001
```

2. 各ノードについて報告されたインターフェイスの状態を確認します（UlbNodeInfo は ULB によって作成されます）。

```
kubectl get ulbnodeinfos -o yaml
 Sample O/p:
apiVersion: v1
items:
- apiVersion: lbs.cisco.com/v1alpha1
  kind: UlbNodeInfo
  metadata:
    name: <ulbnodeinfo-node-name-1>
  spec:
    interfaces:
    - name: ens192.1001
      primaryIpv4: <ipv4-address>
      primaryIpv6: <ipv6-address>
      state: up
    nodeName: <node-name-1>

- apiVersion: lbs.cisco.com/v1alpha1
  kind: UlbNodeInfo
  metadata:
    name: <ulbnodeinfo-node-name-2>
  spec:
    interfaces:
    - name: ens192.1001
      primaryIpv4: <ipv4-address>
```

```
      primaryIpv6: <ipv6-address>
      state: up
    nodeName: <node-nmae-2>

kind: List
```

ULB

ULB ノードインターフェイスの運用状態について説明し、主要なパラメータ、要件、および参照情報についてまとめます。主要なワークフロー、要件、および期待される結果に焦点を当てます。

レポートの状態	意味と管理者のアクション
up	インターフェイスは、ULB サービス処理に使用できます。インターフェイスレベルでのアクションは不要です。
down	インターフェイスは存在しますが動作していません。サービス通信をそのノードにリダイレクトする前に、インターフェイスを起動します。
unknown	インターフェイスが見つかりません。構成された名前を確認するか、そのノードに必要なインターフェイスを作成します。

注

ULB クラスターポリシーが存在する場合は、構成が受け入れられたことを確認します。構成済みのインターフェイスがすべてのノードで現在使用可能であるとは限りません。ノードごとのインターフェイスの状態を使用して、運用準備状況を確認します。

IP ULB

必要な手順や検証アクションなど、IP プール を使用して ULB クラスターポリシーを構成する手順を示します。ULB がサービス通信に使用するノードインターフェイスを構成します。

ULB がサービス通信に使用するノードインターフェイスと Pod のセカンダリインターフェイスの IP プールを構成します。

```
[cluster1/data] lbs# config
[cluster1/data] lbs(config)# cluster-policy
[cluster1/data] lbs(config-cluster-policy)# node-interface name <interface-name>
[cluster1/data] lbs(config-cluster-policy)# pod-ip-pool name <pool-name>
ip-version v4 cidr <ipv4-cidr> node-subnet-prefix <ipv4-prefix>
[cluster1/data] lbs(config-cluster-policy)# pod-ip-pool name <pool-name>
ip-version v6 cidr <ipv6-cidr> node-subnet-prefix <ipv6-prefix>
```

```
[cluster1/data] lbs(config-cluster-policy)# exit
[cluster1/data] lbs(config)# end
```

ULB UNP

ULB ネットワークポリシー（UNP）について説明し、主要なパラメータ、要件、および参照情報についてまとめます。ULB ネットワークポリシーの構成について説明します。

ULB ネットワークポリシーは、アプリケーション Pod の追加のネットワーク接続を定義する、名前空間対応 ULB 機能です。構成がコミットされると、ULB が、名前空間対応 UlbNetworkPolicy リソースを作成するか更新し、ULB ネットワークポリシーと同じ名前と名前空間の Multus NetworkAttachmentDefinition（NAD）を生成します。生成された NAD には、要求されたワークロードネットワーク添付ファイルを作成するために Multus と ULB CNI が必要とする情報が含まれています。バックエンド アプリケーション Pod テンプレートは、k8s.v1.cni.cncf.io/networks 経由で生成された NAD を参照します。

必要なネットワークは、各 Pod レプリカに接続できます。UNP は、バックエンド アプリケーション ネットワークで、複数の IPv4 および IPv6 ダイレクトサービスのアドレスをサポートします。

ULB

ULB ネットワークポリシー（UNP）の構成について説明し、主要なパラメータ、要件、および参照情報についてまとめます。ダイレクトサービス IP の構成について説明します。

ULB ネットワークポリシーを構成するには、次のタスクを使用します。

- ダイレクトサービス IP を構成する

IP

必要な手順や検証アクションなど、ダイレクトサービス IP の構成手順を示します。[cluster1/data] lbs# config [cluster1/data] lbs(config)# network-policy name [network-policy-name] namespace [application-namespace] direct-service-ip ip j についても説明します。

ダイレクトサービス IP を構成する

```
[cluster1/data] lbs# config
[cluster1/data] lbs(config)# network-policy name <network-policy-name> namespace
 <application-namespace> direct-service-ip ip <ipv4/6-ip-address>
[cluster1/data] lbs(config)# network-policy name <network-policy-name> namespace
 <application-namespace> direct-service-ip ip <ipv4/6-ip-address> interface-mtu
 <1280..9000>[1500]
[cluster1/data] lbs(config)# end
```

> **注**
>
> a）オプションの interface-mtu 設定は、ポリシーから作成されるすべてのワークロードインターフェイスに適用されます。サポート範囲は 1280 ～ 9000 で、デフォルトは 1500 です。b）ULB ネットワークポリシーを更新した後、影響を受けるアプリケーション Pod を再作成します。更新されたネットワーク構成は、Pod が作成された場合にのみ CNI ワークフローによって適用されます。実行中の Pod のネットワークインターフェイスに遡って適用されることはありません。新しく作成された Pod は、更新されたダイレクトサービス IP アドレス、インターフェイス MTU、およびその他の該当する NAD 設定を受信します。

ULB

必要な手順や検証アクションなど、ULB ネットワークポリシーを構成する手順を示します。kubectl get ulbnetworkpolicy [network-policy-name] -n [application-namespace] -o yaml について説明します。

ULB ネットワークポリシーを検証する

```
kubectl get ulbnetworkpolicy <network-policy-name> -n <application-namespace>
 -o yaml
```

サンプル出力：

```
apiVersion: lbs.cisco.com/v1alpha1
kind: UlbNetworkPolicy
metadata:
  labels:
    lbs.cisco.com/managed-by: ulb-operator
    lbs.cisco.com/source: configmap
  name: <network-policy-name>
  namespace: <application-namespace>
spec:
  features:
    directServiceIPs:
    - ip: <ipv4-address>
    - ip:<ipv6-address>
    - ip: <ipv6-address>
    interface:
      mtu: <mtu-size>
```

ULB

必要な手順や検証アクションなど、ULB セカンダリ ネットワーク ポリシーを構成する手順を示します。[cluster1/data] lbs# config について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

ULB セカンダリ ネットワーク ポリシーを構成する

```
[cluster1/data] lbs# config
[cluster1/data] lbs(config)# network-policy name <network-policy-name> namespace
 <k8s-namespace> pod-ip-pool name <ip-pool-name>
[cluster1/data] lbs(config)# network-policy name <network-policy-name> namespace
 <k8s-namespace> interface-mtu <1280..9000>[1500]
[cluster1/data] lbs(config)# end
```

注

a. サポートされる範囲は 1280 ～ 9000 で、デフォルト値は 1500 です。

b. ULB ネットワークポリシーを更新した後、必要なバックエンド アプリケーション Pod を再作成/再起動します。更新されたネットワーク構成は、Pod が作成された場合にのみ CNI ワークフローによって適用されます。実行中の Pod のネットワーク インターフェイスに遡って適用されることはありません。新しく作成された Pod は、更新された IP アドレス、インターフェイス MTU、およびその他の該当する NAD 設定を受信します。

ダイレクトサービスについて説明し、主要なパラメータ、要件、および参照情報についてまとめます。対象となるバックエンドにサービス通信を配信する ULB 機能であるダイレクトサービスについて説明します。

ダイレクトサービスは、構成済みサービス IP アドレスおよびポートを保持しながら対象となるバックエンド アプリケーション POD にサービス通信を配信する ULB 機能です。これは、各項で説明されているように、ULB クラスタポリシーと ULB ネットワークポリシー経由で構成されたノードイングレスおよび Pod ネットワークリソースを使用します。backend pod テンプレートは、生成された Multus NAD を参照します。Multus が Pod ネットワークを作成すると、構成された各ダイレクトサービスアドレスが、接続されている各 Pod 内の個別の ULB 管理対象インターフェイスに割り当てられます。これにより、選択したバックエンドがエントリーノード上にある場合でも、別の参加ノード上にある場合でも、一貫したサービス提供が可能になります。

通信が動作中のノードインターフェイスに到着すると、ULB イングレス eBPF プログラムは、接続先サービスの IP アドレス、ポート、およびプロトコルを Direct Service LoadBalancer と照合します。適格なバックエンドを選択するか、既存の有効な配置を再利用し、その後、変更されていないパケットをローカルバックエンドに配信するか、選択したバックエンドをホストする参加ノードに転送します。応答通信は、発信元クライアントまたはピアへの ULB 管理対象リターンパスをたどります。

ULB SCTP

ULB での SCTP ロードバランシングのサポートについて詳述し、主要なパラメータ、要件、および参照情報についてまとめます。SCTP サービス配布の管理について説明します。

ULB は、ダイレクトサービス経由で SCTP ロードバランシングを提供します。すべての SCTP LoadBalancer カスタムリソースは、プロトコル：SCTP および ulbDirectServiceEnabled：true に設定する必要があります。ULB は、SCTP INIT パケットを処理する際に、アソシエーションに適したバックエンド アプリケーション

Pod を選択します。後続のパケットでは、同じバックエンドが使用されますが、記録されたアソシエーションの状態は有効なままです。マルチホーム アソシエーションの場合、別の学習済みピアアドレス経由で受信した通信も、同じバックエンドにリダイレクトされます。

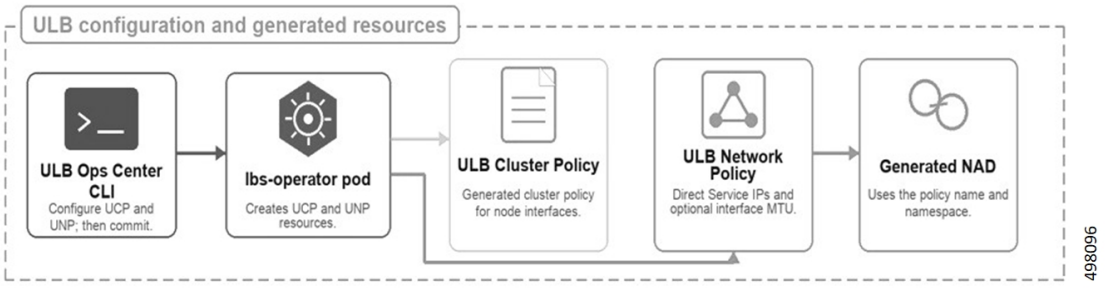
SCTP ロード バランシング構成の要件

2 : SCTP

エリア	管理者の要件
ノードイングレスの可用性	ULB クラスターポリシーは、構成されたインターフェイスがないか、停止している場合でも有効です。ノードは、[状態：起動（state: up）] と報告している間にのみ、そのインターフェイス経由で ULB サービス通信を受信できます。ULB は、インターフェイスが動作可能になると、自動的にそのインターフェイスの使用を開始します。
バックエンドネットワーク接続	SCTP ダイレクト サービス 通信を受信するための各バックエンド Pod は、生成された NAD を参照する Pod テンプレートから作成する必要があります。これにより、Pod 内に ULB 管理対象インターフェイスとサービスアドレスが作成されます。
プロトコルと機能	LoadBalancer で、プロトコル：SCTP and ulbDirectServiceEnabled：true と設定します。
サービスアドレス	LoadBalancer serviceIP と ULB ネットワークポリシーで設定されたものと同じアドレスを使用します。
ポートおよびアプリケーション	servicePort と target.port を同じ値に設定します。バックエンドアプリケーションは、構成されたサービスアドレスとポートで SCTP 通信を受け入れる必要があります。
バックエンドの選択	LoadBalancer 名前空間で対象のバックエンド アプリケーション Pod と一致するように、target.selectors.matchLabels を構成します。
インターフェイス MTU	必要に応じて、1280 ～ 9000 の MTU を設定します。デフォルトは 1500 です。

SCTP ロードバランシングに向けた ULB 構成および生成リソース

2 : SCTP ULB



SCTP デプロイメントの場合、それぞれのセクションの説明に従い、ULB クラスターポリシーを使用して目的のノードインターフェイスを構成し、ULBネットポリシーを使用してサービスアドレスを構成します。

ULB は、コミットされた構成を処理し、対応する Kubernetes リソースを自動的に作成または更新します。管理者は、これらのリソースを手動で作成しません。

ULB クラスターポリシーには、ULB が着信サービス通信を処理するノードインターフェイスが含まれます。ULB ネットワークポリシーには、ダイレクトサービス IP アドレスと、必要に応じてインターフェイス MTU が含まれています。ULB ネットワークポリシーの場合、ULB は、ポリシーと同じ名前空間にポリシーと同じ名前で Multus NetworkAttachmentDefinition も作成します。Backend application pod テンプレートは、k8s.v1.cni.cncf.io/networks Pod 注釈によってこの名前を参照します。Multus が Pod ネットワークを作成すると、構成された各サービスアドレスが Pod 内の個別の ULB 管理対象インターフェイスに割り当てられます。

生成されたリソースを確認するには、次を使用します。

```
kubectl get ulbclusterpolicy global-ulb-cluster-policy
```

```
kubectl get ulbnetworkpolicy -n <application-namespace> <network-policy-name>
```

```
kubectl get network-attachment-definitions -n <application-namespace>
<network-policy-name>
```

SCTP

必要な手順や検証アクションなど、SCTP ダイレクトサービス LoadBalancer の構成手順を示します。次の例では、デュアルスタック SCTP LoadBalancer を作成します。

次の例では、デュアルスタック SCTP LoadBalancer を作成します。ULB ネットワークポリシー、生成された NAD、バックエンド アプリケーション Pod の注釈、およびバックエンド アプリケーション Pod の準備ができた後に適用します。

SCTP ダイレクトサービスのロードバランサを構成する

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  labels:
    app: <sample-application>
    component: <sample-component>
    release: <sample-release>
  name: <loadbalancer-name>
  namespace: <application-namespace>
spec:
  protocol: SCTP
  serviceIPs:
    - <ipv6-address>
    - <ipv6-address>
    - <ipv4-address>
    - <ipv4-address>
  servicePort: 9002
  target:
    port: 9002
    selectors:
      matchLabels:
        <label-Key1>: <label-Value1>
        <label-Key2>: <label-Value2>
        <label-Key3>: <label-Value3>
```

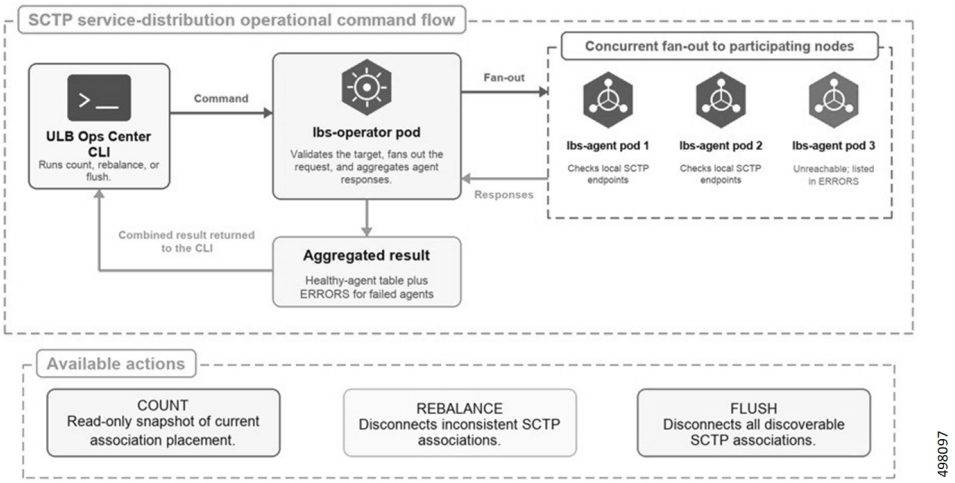
```
ulbDirectServiceEnabled: true
```

SCTP

SCTP サービス分散の管理について説明し、主要なパラメータ、要件、および参照情報についてまとめます。ULB Ops Center service-distribution コマンドを使用して、SCTP アソシエーションを調査するか、意図的にリセットする方法について説明します。

ULB Ops Center service-distribution コマンドを使用して、SCTP アソシエーションを調査するか、意図的にリセットします。コマンドは、ダイレクトサービスが有効になっている SCTP LoadBalancer にのみ適用されます。

3 : SCTP service-distribution



3 : SCTP service-distribution

動作	動作と使用
count	読み取り専用。SCTPアソシエーションの配置を報告し、アクティブな配信と不整合なアソシエーションを識別します。
rebalance	不整合なSCTPアソシエーションのみを切断します。クライアントは再接続し、現在の分散に従って配置されます。
flush	LoadBalancer に対してE メール検出可能なすべてのSCTPアソシエーションを解除します。計画中断を伴うサービス操作中にのみ使用します。

現在の SCTP アソシエーション分散をカウントします。

loadbalancer service distribution count lbName *loadbalancer-name* **namespace** *application-namespace*

不整合な SCTP アソシエーションのみを切断します。

loadbalancer service distribution rebalance lbName *loadbalancer-name* **namespace** *application-namespace*

検出可能なすべての SCTP アソシエーションを解除します。

loadbalancer service distribution flush lbName *loadbalancer-name* **namespace** *application-namespace*

> **注**
>
> 再調整とフラッシュはすぐに有効になります。最初に count を実行し、LoadBalancer の名前と名前空間を確認してから、中断を伴う操作としてフラッシュをスケジュールします。

lbs-operator Pod は、実行中の lbs-agent Pod からの結果を集約し、1 つの結合テーブルを ULB Ops Center CLI に返します。正常な ULB エージェント Pod の行は表示されたままになり、失敗したエージェント Pod のステータスは、顧客が判読できる理由とともに [エラー] の下に表示されます。

ULB GTP-C

ULB で GTP-C ロードバランサを導入し、主要な概念、要件、および運用上の考慮事項に焦点を当てます。ULB は、バックエンド Pod 全体に GTPv2-C 通信を分散する eBPF ベースの GTP-C ロードバランサを提供します。

ULB は、セッションアフィニティを維持しながら、バックエンド Pod 全体に GTPv2-C 通信を分散する eBPF ベースの GTP-C ロードバランサを提供します。ノードレベルのメッセージ、新しいセッションメッセージ、および既存のセッションメッセージを個別に処理します。アプリケーションは、セッションレベルとノードレベルの GTPv2-C 通信に個別のバックエンド Pod グループを使用できます。

GTP セッション管理：

新しいサブスクライバセッションが開始されると、ULB は構成されたロードバランシングアルゴリズムを使用して GTP-C サービスインスタンスを選択します。セッションが確立されたら、ULB はセッションアフィニティを維持し、そのセッションに関連付けられている後続のすべてのメッセージが、同じサービスインスタンスによって処理されるようにします。

GTP-C サービス Pod によって生成されたエグレス GTPv2-C セッションメッセージは、ULB 経由で適切な外部ピアにルーティングされます。これにより、ULB は一貫したセッション処理を保証しながら、GTPv2-C 出入力通信の両方をルーティングできます。

GTP ピア管理：

ULB は、ピアに関連するシグナリングメッセージを適切な GTP-C ノード管理 Pod にルーティングすることで、GTPv2-C ピア管理をサポートします。現在、ULB は、ランダム選択アルゴリズムを使用して、使用可能なノード管理バックエンド Pod 全体にピア管理メッセージを分散します。

GTP- C

技術的な実装と運用に向けた GTP-C ロードバランサのカスタムリソースについて詳述します。主要なワークフロー、要件、および期待される結果に焦点を当てます。

この項では、GTP-C ロードバランサのカスタムリソースの構成例を示します。

セッションレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  name: <gtpc-sess-lb-name>
spec:
  lbMode: bpf
```

```
  protocol: GTPC
  serviceIPs:
  - <ipv4-address>
  servicePort: 2123
  target:
    port: <target-port>
    selectors:
      matchLabels:
        <label-key1>: <label-value1>
        <label-key2>: <label-value2>
  trafficManagementPolicyRef: <gtpc-sess-tmp-name>
  ulbDirectServiceEnabled: false
```

ノードレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  name: <gtpc-node-lb-name>
spec:
  lbMode: bpf
  protocol: GTPC
  serviceIPs:
  - <ipv4-address>
  servicePort: 2123
  target:
    port: <target-port>
    selectors:
      matchLabels:
        <label-key1>: <label-value1>
        <label-key2>: <label-value2>
  trafficManagementPolicyRef: <gtpc-node-tmp-name>
  ulbDirectServiceEnabled: false
```

GTP-C

技術的な実装と運用に向けたGTP-C通信管理ポリシーのカスタムリソースについて詳述します。主要なワークフロー、要件、および期待される結果に焦点を当てます。

セッションレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: TrafficManagementPolicy
metadata:
  name: <gtpc-sess-tmp-name>
spec:
  # GTP-C v2 session message handling (Message Type >= 32)
  gtpcSessionMessages:

    # Load balancing algorithm
    algorithm: RANDOM_LOCAL

    # Affinity policy
    sessionAffinity:
```

```
      # Node identifier extraction
      nodeIdentifier:
        target:
          header: GTPC          # GTP-C Header
          offset: 4             # TEID field offset
          length: 4             # TEID is 4 bytes
        operation:
          type: BITWISE_AND
          mask: "0x3C000000"    # Extract bits 26-29 (4 bits)

      # Session identifier extraction
      sessionIdentifier:
        target:
          header: GTPC          # GTP-C Header
          offset: 4             # TEID field offset
          length: 4             # TEID is 4 bytes
        operation:
          type: BITWISE_AND
          mask: "0x03FFFFFF"    # Extract bits 0-25 (26 bits)

      # Session Map provisioning strategy
      sessionMap:
        provisioningMode: APPLICATION_PUSH_NOTIFICATION

      # Affinity Notification
      affinityNotification:
        # Currently single rule is allowed
        rules:
          - condition: KEY_NOT_FOUND

            # Action to take by marking spare bit in GTP-C Header
            # SET_GTPC_SPARE_BIT1: Set spare bit 1 in GTPC Header
            # SET_GTPC_SPARE_BIT2: Set spare bit 2 in GTPC Header
            action: SET_GTPC_SPARE_BIT1
```

ノードレベルのメッセージを紹介し、プライマリ範囲の概要を示します。**apiVersion** について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: TrafficManagementPolicy
metadata:
  name: <gtpc-node-tmp-name>
spec:
  # GTP-C v2 node message handling (Message Type < 32)
  gtpcNodeMessages:

    # Load balancing algorithm (RANDOM)
    algorithm: RANDOM
```

GTP-C

技術的な実装と運用に向けた **GTP-C** エグレス管理ポリシーのカスタムリソースについて詳述します。主要なワークフロー、要件、および期待される結果に焦点を当てます。

セッションレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: <gtpc-sess-emp-name>
spec:
  egressIPs:
  - <gtpc-service-IP>
  egressMode: bpf
  egressPort:
    stateful:
    - podName: <gtpc-sess-0-name>
      portRanges:
      - <port-range11...port-range1n>
    - podName: <gtpc-sess-1-name>
      portRanges:
      - <port-range21...port-range2n>
    - podName: <gtpc-sess-2-name>
      portRanges:
      - <port-range31...port-range3n>
    - podName: <gtpc-sess-3-name>
      portRanges:
      - <port-range41...port-range4n>
    - podName: <gtpc-sess-4-name>
      portRanges:
      - <port-range51...port-range5n>
  protocol: GTPC
  target:
    selectors:
      matchLabels:
        <label-key1>: <label-value1>
        <label-key2>: <label-value2>
```

技術的な実装と運用に向けたノードレベルメッセージについて詳述します。PFCP セッション管理と PFCP ピアの管理について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

GTP-C Echo Request および Response メッセージの場合、送信元ポートと宛て先ポートの両方が通常 2123 に設定されます。アプリケーションがこれらのメッセージに異なる送信元ポート範囲を使用する場合、前述のセッションレベルの EMP CR と同じ方法で、ノードレベルの通信に対して EMP CR を構成できます。

制限事項について、一貫した実装をサポートするための推奨事項と制約について説明します。ULB がフラグメント化された IPv4 パケットまたは IPv6 拡張ヘッダーチェーンを明示的に処理しないことについて説明します。

1. ULB はフラグメント化された IPv4 パケットまたは IPv6 拡張ヘッダーチェーンを明示的に処理しません。
2. このリリースでは、ローリングアップグレードとダウングレードはサポートされていません。
3. このリリースでは、GTPv1-C メッセージ処理はサポートされません。
4. Pod やノードの再起動などのレジリエンシイベントからのリカバリは、このリリースではサポートされていません。

ULB PFCP

ULB での PFCP ロードバランシングのサポートについて説明し、主要なパラメータ、要件、および参照情報についてまとめます。ULB は、バックエンド Pod 全体に PFCP 通信を分散する eBPF ベースの PFCP ロードバランサを提供します。

ULB は、セッションアフィニティを維持しながらバックエンド Pod 全体に PFCP 通信を分散する、eBPF ベースの PFCP ロードバランサを提供します。ノードレベルのメッセージ、新しいセッションメッセージ、および既存のセッションメッセージを個別に処理します。アプリケーションは、セッションレベルとノードレベルの PFCP 通信に個別のバックエンド Pod グループを使用できます。

PFCP

技術的な実装と運用に向けた PFCP ロードバランサのカスタムリソースについて詳述します。主要なワークフロー、要件、および期待される結果に焦点を当てます。

セッションレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  name: <pfcp-sess-lb-name>
spec:
  lbMode: bpf
  protocol: PFCP
  serviceIPs:
  - <ipv4_address>
  servicePort: 8805
  target:
    port: <target-port>
    selectors:
      matchLabels:
        <label-key1>: <label-value1>
        <label-key2>: <label-value2>
  trafficManagementPolicyRef: <pfcp-sess-tmp-name>
  ulbDirectServiceEnabled: false
```

ノードレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  name: <pfcp-node-lb-name>
spec:
  lbMode: bpf
  protocol: PFCP
  serviceIPs:
  - <pfcp-service-ip>
  servicePort: 8805
  target:
    port: <target-port>
    selectors:
```

```
      matchLabels:
        <label-key1>: <label-value1>
        <label-key2>: <label-value2>

  trafficManagementPolicyRef: <pfcp-node-tmp-name>
  ulbDirectServiceEnabled: false
```

PFCP

技術的な実装と運用に向けたPFCP通信管理ポリシーのカスタムリソースについて詳述します。主要なワークフロー、要件、および期待される結果に焦点を当てます。

セッションレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: TrafficManagementPolicy
metadata:
  name: <pfcp-sess-lb-name>
spec:
  # PFCP session message handling (Message Type >= 50)
  pfcpSessionMessages:

    # Load balancing algorithm
    algorithm: RANDOM_LOCAL

    # Affinity policy
    sessionAffinity:
      # Node identifier extraction
      nodeIdentifier:
        target:
          header: PFCP          # PFCP Header
          offset: 4             # SEID field offset
          length: 8             # SEID is 8 bytes
        operation:
          type: BITWISE_AND
          mask: "0x3C00000000000000"    # Extract bits 58-61 (4 bits)

      # Session identifier extraction
      sessionIdentifier:
        target:
          header: PFCP          # PFCP Header
          offset: 4             # SEID field offset
          length: 8             # SEID is 8 bytes
        operation:
          type: BITWISE_AND
          mask: "0x03FFFFFFFFFFFFFF"    # Extract bits 0-57 (58 bits)

      # Session Map provisioning strategy
      sessionMap:
        provisioningMode: APPLICATION_PUSH_NOTIFICATION

      # Affinity Notification
      affinityNotification:
        # Currently single rule is allowed
        rules:
          - condition: KEY_NOT_FOUND

            # Action to take by marking spare bit in PFCP Header
```

```
            # SET_PFCP_SPARE_BIT4: Set spare bit 4 in PFCP Header

            # SET_PCFP_SPARE_BIT5: Set spare bit 5 in PFCP Header
            action: SET_PFCP_SPARE_BIT4
```

ノードレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: TrafficManagementPolicy
metadata:
  name: <pfcp-node-tmp-name>
spec:
  # PFCP node message handling (Message Type < 50)
  pfcpNodeMessages:

    # Load balancing algorithm (RANDOM)
    algorithm: RANDOM
```

PFCP

技術的な実装と運用に向けたPFCPエグレス管理ポリシーのカスタムリソースについて詳述します。主要なワークフロー、要件、および期待される結果に焦点を当てます。

セッションレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: <pfcp-sess-emp-name>
spec:
  egressIPs:
  - <pfcp-service-ip>
  egressMode: bpf
  egressPort:
    stateful:
    - podName: <pfcp-sess-0-name>
      portRanges:
      - <port-range11...port-range1n>
    - podName: <pfcp-sess-1-name>
      portRanges:
      - <port-range21...port-range2n>
    - podName: <pfcp-sess-2-name>
      portRanges:
      - <port-range31...port-range3n>
    - podName: <pfcp-sess-3-name>
      portRanges:
      - <port-range31...port-range4n>
    - podName: <pfcp-sess-4-name>
      portRanges:
      - <port-range41...port-range5n>
  protocol: PFCP
  target:
    selectors:
      matchLabels:
```

```
        <label-key1>: <label-value1>
        <label-key2>: <label-value2>
```

ノードレベルのメッセージを紹介し、プライマリ範囲の概要を示します。apiVersion について説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: <pfcp-node-emp-name>
spec:
  egressIPs:
  - <pfcp-service-ip>
  egressMode: bpf
  egressPort:
    stateful:
    - podName: <pfcp-node-0-name>
      portRanges:
      - <port-range11...port-range1n>
    - podName: <pfcp-node-1-name>
      portRanges:
      - <port-range21...port-range2n>
  protocol: PFCP
  target:
    selectors:
      matchLabels:
        <label-key1>: <label-value1>
        <label-key2>: <label-value2>
```

制限事項について、一貫した実装をサポートするための推奨事項と制約について説明します。ULB がフラグメント化された IPv4 パケットまたは IPv6 拡張ヘッダーチェーンを明示的に処理しないことについて説明します。

1. ULB はフラグメント化された IPv4 パケットまたは IPv6 拡張ヘッダーチェーンを明示的に処理しません。
2. このリリースでは、ローリングアップグレードとダウングレードはサポートされていません。
3. Pod やノードの再起動などのレジリエンシイベントからのリカバリは、このリリースではサポートされていません。

第 7 章 KPI

:

- ULB □□□□□□□□□□□
- ULB □□□□□□□□□□□□

技術的な実装と運用の評価指標と KPI について説明します。ULB オペラ-タメトリックおよび ULB E メールメトリックについて説明し、主要なワークフロー、要件、および期待される結果に焦点を当てます。

この項では、ULB のパフォーマンスにアクセスできる主要なメトリックについて概説します。

ULB

ULB オペレータメトリックについて説明し、主要なパラメータ、要件、および参考情報についてまとめます。

ULB オペラ-タメトリックは次のとおりです。

名前	データ タイプ	メトリックタイプ	説明
loadbalancer_custom_resource_add_success_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信した正常な loadbalancer カスタムリソース追加要求の総数。
loadbalancer_custom_resource_add_fail_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信したエラーとなった loadbalancer カスタムリソース追加要求の総数。
loadbalancer_custom_resource_modify_success_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信した正常な loadbalancer カスタムリソース変更要求の総数。
loadbalancer_custom_resource_modify_fail_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信したエラーとなった loadbalancer カスタム リソース変更要求の総数。
loadbalancer_custom_resource_delete_success_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信した正常な loadbalancer カスタムリソース削除要求の総数。
loadbalancer_custom_resource_delete_fail_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信したエラーとなった loadbalancer カスタムリソース削除要求の総数。
egressmanagementpolicies_custom_resource_add_success_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信した正常な egressmanagementpolicies カスタムリソース追加要求の総数。
egressmanagementpolicies_custom_resource_add_fail_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信したエラーとなった egressmanagementpolicies カスタムリソース追加要求の総数。

名前	データ タイプ	メトリックタイプ	説明
egressmanagementpolicies_custom_resource_modify_success_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信した正常な egressmanagementpolicies カスタムリソース変更要求の総数。
egressmanagementpolicies_custom_resource_modify_failed_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信したエラーとなった egressmanagementpolicies カスタムリソース変更要求の総数。
egressmanagementpolicies_custom_resource_delete_success_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信した正常な egressmanagementpolicies カスタムリソース削除要求の総数。
egressmanagementpolicies_custom_resource_delete_failed_total ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	受信したエラーとなった egressmanagementpolicies カスタムリソース削除要求の総数。
service_modify_total_not_loadbalancer_owner ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	ロードバランサーが所有していない サービスが原因でエラーとなった サービス変更要求の総数。
service_delete_total_not_loadbalancer_owner ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	ロードバランサーが所有していない サービスが原因でエラーとなった サービス削除要求の総数。
service_modify_total_not_egressmanagementpolicies_owner ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	egressmanagementpolicies が所有していない サービスが原因で エラーとなった サービス変更要求の総数。
service_delete_total_not_egressmanagementpolicies_owner ラベル：[NameSpace, Kind, CrName, SvcName, PodName, Error]	INT64	カウンタ	egressmanagementpolicies が所有していない サービスが原因でエラーとなった サービス変更要求の 総数。

次の表に、オペレータメトリックに関連付けられたラベルの説明を示します。

ラベル名	ラベル タイプ	説明
NameSpace	文字列	リソースが作成される名前空間。
種類	文字列	作成されたリソースの種類が表示されます。
CrName	文字列	カスタムリソースの名前。
SvcName	文字列	作成されたサービスの名前。
PodName	文字列	Pod の名前。
エラー（Error）	文字列	障害発生時のエラー文字列。

ULB

ULBEメールメトリックについて説明し、主要なパラメータ、要件、および参考情報についてまとめています。

ULB エージェントメトリックは次のとおりです。

名前	データ タイプ	メトリックタイプ	説明
egressmanagementpolicy_customresource_add_event_total	INT64	カウンタ	受信したエグレス管理ポリシーのカスタムリソース追加イベントの総数。
egressmanagementpolicy_customresource_modify_event_total	INT64	カウンタ	受信したエグレス管理ポリシーのカスタムリソース変更イベントの総数。
egressmanagementpolicy_customresource_delete_event_total	INT64	カウンタ	受信したエグレス管理ポリシーのカスタムリソース削除イベントの総数。
pod_add_event_total	INT64	カウンタ	受信した Pod 追加イベントの総数
pod_add_event_ignore_total	INT64	カウンタ	無視された Pod 追加イベントの総数。
pod_modify_event_total	INT64	カウンタ	受信した Pod 変更イベントの総数
pod_modify_event_ignore_total	INT64	カウンタ	無視された Pod 変更イベントの総数。
pod_delete_event_total	INT64	カウンタ	受信した Pod 削除イベントの総数
node_add_event_total ラベル：component	INT64	カウンタ	受信したノード 追加イベントの総数。
node_add_event_ignored_total ラベル：component（値：neigbor）	INT64	カウンタ	無視されたノード 追加イベントの総数。

名前	データ タイプ	メトリックタイプ	説明
node_modify_event_total ラベル：component（値：neigbor）	INT64	カウンタ	受信したノード 変更イベントの総数。
node_modify_event_ignored_total ラベル：component（値：neigbor）	INT64	カウンタ	無視されたノード 変更イベントの総数。
node_delete_event_total ラベル：component（値：neigbor）	INT64	カウンタ	受信したノード 削除イベントの総数。
loadbalancer_customresource _add_event_total	INT64	カウンタ	受信したロードバランサ カスタムリソース 追加イベントの 総数。
loadbalancer_customresource _delete_event_total	INT64	カウンタ	受信したロードバランサ カスタムの 削除イベントの 総数。
loadbalancer_customresource _modify_event_total	INT64	カウンタ	受信したロードバランサ カスタムの 変更イベントの 総数。
loadbalancer_pod_add_event_total	INT64	カウンタ	受信したロードバランサ。 Pod 追加イベントの 総数。
loadbalancer_pod_modify_event_total	INT64	カウンタ	受信したロードバランサ Pod 変更イベントの 総数。
loadbalancer_pod_delete_event_total	INT64	カウンタ	受信したロードバランサ Pod 削除イベントの 総数。
loadbalancer_pod_add_event _ignored_total	INT64	カウンタ	無視されたロードバランサ Pod 追加イベントの 総数。
loadbalancer_pod_modify_event _ignored_total	INT64	カウンタ	無視されたロードバランサ Pod 変更イベントの 総数。

名前	データ タイプ	メトリックタイプ	説明
loadbalancer_pod_delete_event_ignored_total	INT64	カウンタ	無視されたロードバランサ Pod 削除イベントの総数。
serviceoutboundroute_customresource_add_event_total	INT64	カウンタ	受信したサービスアウトバウンドルート カスタム リソース追加イベントの総数。
serviceoutboundroute_customresource_modify_event_total	INT64	カウンタ	受信したサービスアウトバウンドルート カスタム リソース変更イベントの総数。
serviceoutboundroute_customresource_delete_event_total	INT64	カウンタ	受信したサービスアウトバウンドルート カスタム リソース削除イベントの総数。総数。
serviceoutboundroute_customresource_add_event_ignored_total	INT64	カウンタ	無視されたサービスアウトバウンド ルート カスタム リソース追加イベントの総数。
serviceoutboundroute_customresource_modify_event_ignored_total	INT64	カウンタ	無視されたサービスアウトバウンド ルート カスタム リソース変更イベントの総数。
serviceoutboundroute_customresource_delete_event_ignored_total	INT64	カウンタ	無視されたサービスアウトバウンド ルート カスタム リソース削除イベントの総数。
svc_packet_count ラベル：[svc_ip, svc_name, pod_ip, pod_port, pod_name, protocol, direction]	INT64	カウンタ	サービス IP ごとのパケットの総数。

名前	データ タイプ	メトリックタイプ	説明
svc_byte_count ラベル：[svc_ip, svc_name, pod_ip, pod_port, pod_name, protocol, direction]	INT64	カウンタ	サービス IP ごとのバイトの総数。

次の表に、エージェントメトリックの一部に関連付けられたラベルの説明を示します。

ラベル名	ラベル タイプ	説明
svc_ip	文字列	サービスの IP アドレス。
svc-name	文字列	EMP サービスの名前
pod_ip	文字列	バックエンド Pod の IP アドレス
pod_port	文字列	バックエンド Pod で使用されるポート
pod_name	文字列	バックエンド Pod の名前。
protocol	文字列	L4 プロトコル：TCP または UDP
direction	文字列	出入力パケットの方法。

ULBConfig CR 経由でデータパス統計を有効にすると、各 ULB E メールは、プロトコルニュートラル eBPF データパス操作統計を Prometheus に公開します。

名前	データ種別	メトリックタイプ	説明
ulb_dp_rx_total	INT64	ゲージ	構成済みの ULB サービスに帰属する累積パケット観測数を、レポート対象のインターフェイスおよびパケットソース別に分類したもの。
ulb_dp_session_total	INT64	ゲージ	プロトコル定義セッション、接続、またはフロー開始の累積観測数。
ulb_dp_forward_total	INT64	ゲージ	インストゥルメント化 ULB フォワーディングパスイベントの累積数。1 つのパケットを複数のパス値に指定できます。
ulb_dp_ebpf_failures	INT64	ゲージ	理由別にグループ化された、インストゥルメント化された eBPF 障害イベントの累積数。

次の表では、前述の eBPF データパス運用 ulb_dp_* 統計に関連付けられたラベルについて説明します。

ラベル名	ラベルの種類	説明
app_namespace	文字列	ULB サービスに関連付けられた Kubernetes 名前空間。

resource_name	文字列	サービスに関連付けられた ULB リソースの名前。
service_ip	文字列	構成された ULB サービスの IPv4 または IPv6 アドレス。
service_port	文字列	設定済みの ULB サービスに関連付けられたポート。
protocol	文字列	サービスに関連付けられたプロトコル。udp、tcp、sctp、gtpc、gtpu、pfcp などの値は小文字でエクスポートされます。
ifname	文字列	報告されたイベントに関連付けられているネットワーク インターフェイスの名前。
source	文字列	パケット送信元の分類。このラベルは、ulb_dp_rx_total に固有です。
path	文字列	フォワーダーパスの分類。このラベルは、ulb_dp_forward_total に固有です。
reason	文字列	インストゥルメント化された eBPF 障害に関連付けられた理由。このラベルは、ulb_dp_ebpf_failures に固有です。

注

エクスポータは、デフォルトでは、これらの累積スナップショットを 5 分ごとに更新します。統計を有効にしたら、コントローラとエージェントの伝達、エクスポータの更新、および Prometheus によるスクレイプが行われるまで時間を置きます。

付録

A

:

- □□□□□□□ CR□IPv4□
- □□□□□□□ CR□IPv6□
- □□□□□□ Pod □□□□□□□□□□□□ CR□IPv4□
- □□□□□□ Pod □□□□□□□□□□□□ CR□IPv6□
- □□□□□□ Pod □□□□□□□□□□□□ CR□IPv4□
- □□□□□□ Pod □□□□□□□□□□□□ CR□IPv6□
- □□□□□□□□□□□□□□□□ Pod □□□ □□□□□□□□ CR□IPv4□
- □□□□ □□□□□□□ □□□□SOR□ IPv4
- □□□□ □□□□□□□ □□□□SOR□ IPv6

ロードバランサとエグレス管理ポリシーのカスタムリソース定義の例について説明します。これには、ロードバランサ CR（IPv4）、ロードバランサ CR（IPv6）、ステートフル Pod（IPv4）のエグレス管理ポリシー CR、ステートフル Pod（IPv6）のエグレス管理ポリシー CR が含まれます。

この項では、ロードバランサ（LB）とエグレス管理ポリシー（EMP）カスタムリソース（CR）を焦点に置いた YAML 構成ファイルの例について説明します。この例を使用して、デプロイメントのニーズに基づいてLB と EMP CR を定義して適用します。

CR IPv4

ロードバランサ CR （IPv4）について説明し、主要なパラメータ、要件、および参考情報についてまとめます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  name: udp-lb
spec:
  protocol: UDP
  servicePort: 8765
  serviceIPs:
    - 2.2.2.2
  target:
    port: 8807
    selectors:
      matchLabels:
        release: udp-nf
```

CR IPv6

ロードバランサ CR （IPv6）について説明し、主要なパラメータ、要件、および参考情報についてまとめます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: LoadBalancer
metadata:
  name: udp-lb-v6
spec:
  protocol: UDP
  servicePort: 8766
  serviceIPs:
    - 2201::110B
  target:
    port: 8808
    selectors:
      matchLabels:
        release: udp-nf
```

Pod CR IPv4

IPv4 を使用するステートフル Pod のエグレス管理ポリシーカスタムリソース（CR）について説明します。これには、主要なパラメータ、要件、参照情報が含まれます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp
spec:
  protocol: UDP
  egressPort:
    stateful:
      - podName: pod-1
        portRanges:
```

```
          - "15001…15003"
          - 16001
      - podName: pod-2
        portRanges:
          - "17001…17004"
  target:
    selectors:
      matchLabels:
        release: udp-nf
  egressIPs:
    - 2.2.2.2
```

Pod CR IPv6

IPv6 を使用するステートフル Pod のエグレス管理ポリシーカスタムリソース（CR）について説明します。これには、主要なパラメータ、要件、参照情報が含まれます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp-v6
spec:
  protocol: UDP
  egressPort:
    stateful:
      - podName: pod-1
        portRanges:
          - "25001…25003"
      - podName: pod-2
        portRanges:
          - "27001…27003"
  target:
    selectors:
      matchLabels:
        release: udp-nf
  egressIPs:
    - 2201::110B
```

Pod CR IPv4

IPv4 を使用するステートレス Pod のエグレス管理ポリシーカスタムリソース（CR）について説明します。これには、主要なパラメータ、要件、参照情報が含まれます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp-sl
spec:
  protocol: UDP
  egressPort:
    stateless:
        portRanges:
          - "35001…35003"
  target:
    selectors:
      matchLabels:
```

```
        release: udp-nf
  egressIPs:
    - 4.4.4.4
```

Pod CR IPv6

IPv4 と比較した主要なパラメータや構成の違いなど、ステートレス Pod（IPv6）のエグレス管理ポリシー CR について説明します。

LoadBalancer と EgressManagementPolicy の構成は、前のセクションで IPv4 について説明したものと似ていますが、唯一の違いは、LoadBalancer の [serviceIPs] フィールドと EgressManagementPolicy の [egressIPs] フィールドでは IPv6 アドレスが使用されることです。

Pod CR IPv4

ポート範囲のないステートフル Pod（IPv4）のエグレス管理ポリシー CR について説明します。これには、主要なパラメータ、参照情報が含まれます。

```
apiVersion: lbs.cisco.com/v1alpha1
kind: EgressManagementPolicy
metadata:
  name: udp-emp-st-wo-pr
spec:
  protocol: UDP
  egressPort:
    stateful:
      - podName: pod-1
      - podName: pod-2
  target:
    selectors:
      matchLabels:
        release: udp-nf
  node:
    selectors:
      matchLabels:
        smi.cisco.com/egress: enable
  egressIPs:
    - 7.7.7.7
```

SOR IPv4

サービス アウトバウンド ルート（SOR）IPv4 について説明し、主要なパラメータ、要件、および出力で期待される結果についてまとめます。

```
Name:         sor4
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  lbs.cisco.com/v1alpha1
Kind:         ServiceOutboundRoute
Metadata:
  Creation Timestamp:  2025-02-25T05:23:57Z
  Generation:          1
  Resource Version:    465878
```

```
  UID:                 8d3a629c-b9cc-4195-9bb3-a8cdae948613
Spec:
  Device:    ens224.502
  Next Hop:  209.165.201.1
Status:
  Node Info:
    Default Route:  {Ifindex: 64 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2
Flags: [] Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-worker2
    Table Id:       1002
    Default Route:  {Ifindex: 94 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2
Flags: [] Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-master2
    Table Id:       1002
    Default Route:  {Ifindex: 78 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2
Flags: [] Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-master3
    Table Id:       1002
    Default Route:  {Ifindex: 92 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2
Flags: [] Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-master1
    Table Id:       1002
    Default Route:  {Ifindex: 72 Dst: 0.0.0.0/0 Src: <nil> Gw: 209.165.201.2
Flags: [] Table: 1002 Realm: 0}
    Node Name:      rid7656453-user-1-worker1
    Table Id:       1002
```

出力：

```
sudo ip route show table all | grep default
local default dev lo table 2004 proto kernel scope host
default via 209.165.201.2 dev ens224.502 table 1002
```

SOR IPv6

サービス アウトバウンド ルート（SOR）IPv6 について説明し、主要なパラメータ、要件、および出力で期待される結果についてまとめます。

```
Name:         sor4-v6
Namespace:    default
Labels:       <none>
Annotations:  <none>
API Version:  lbs.cisco.com/v1alpha1
Kind:         ServiceOutboundRoute
Metadata:
  Creation Timestamp:  2025-02-25T07:35:33Z
  Generation:          1
  Resource Version:    489696
  UID:                 8d43362a-8662-435e-a1dd-32600cc0d0d3
Spec:
  Device:    ens224.502
  Next Hop:  4444::100
Status:
  Node Info:
    Default Route:  {Ifindex: 78 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: []
 Table: 1003 Realm: 0}
    Node Name:      rid7656453-user-1-master3
    Table Id:       1003
    Default Route:  {Ifindex: 72 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: []
```

```
 Table: 1003 Realm: 0}
    Node Name:      rid7656453-user-1-worker1
    Table Id:       1003
    Default Route:  {Ifindex: 64 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: []
 Table: 1003 Realm: 0}
    Node Name:      rid7656453-user-1-worker2
    Table Id:       1003
    Default Route:  {Ifindex: 94 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: []
 Table: 1003 Realm: 0}
    Node Name:      rid7656453-user-1-master2
    Table Id:       1003
    Default Route:  {Ifindex: 92 Dst: ::/0 Src: <nil> Gw: 4444::100 Flags: []
 Table: 1003 Realm: 0}
    Node Name:      rid7656453-user-1-master1
    Table Id:       1003
Events:             <none>
```

出力：

```
sudo ip route show table all | grep default
default via 4444::100 dev ens224.502 table 1003 metric 1024 pref medium
```

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。

翻訳について
このドキュメントは、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。