



OpenShift Container Platform 4.16

Network Observability

OpenShift Container Platform での Network Observability Operator の設定と使用

OpenShift Container Platform 4.16 Network Observability

OpenShift Container Platform での Network Observability Operator の設定と使用

法律上の通知

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

Network Observability Operator を使用して、OpenShift Container Platform クラスターのネットワークトラフィックフローを監視および分析します。

目次

第1章 NETWORK OBSERVABILITY OPERATOR リリースノート	4
1.1. NETWORK OBSERVABILITY OPERATOR 1.6.0	4
1.2. NETWORK OBSERVABILITY OPERATOR 1.5.0	6
1.3. NETWORK OBSERVABILITY OPERATOR 1.4.2	9
1.4. NETWORK OBSERVABILITY OPERATOR 1.4.1	9
1.5. NETWORK OBSERVABILITY OPERATOR 1.4.0	10
1.6. NETWORK OBSERVABILITY OPERATOR 1.3.0	13
1.7. NETWORK OBSERVABILITY OPERATOR 1.2.0	15
1.8. NETWORK OBSERVABILITY OPERATOR 1.1.0	16
第2章 NETWORK OBSERVABILITY について	18
2.1. NETWORK OBSERVABILITY OPERATOR のオプションの依存関係	18
2.2. NETWORK OBSERVABILITY OPERATOR	18
2.3. OPENSIFT CONTAINER PLATFORM コンソール統合	18
2.4. NETWORK OBSERVABILITY CLI	19
第3章 NETWORK OBSERVABILITY OPERATOR のインストール	20
3.1. LOKI を使用しない NETWORK OBSERVABILITY	20
3.2. LOKI OPERATOR のインストール	21
3.3. NETWORK OBSERVABILITY OPERATOR のインストール	26
3.4. FLOW COLLECTOR 設定に関する重要な考慮事項	28
3.5. KAFKA のインストール (オプション)	28
3.6. NETWORK OBSERVABILITY OPERATOR のアンインストール	29
第4章 OPENSIFT CONTAINER PLATFORM の NETWORK OBSERVABILITY OPERATOR	30
4.1. 状況の表示	30
4.2. NETWORK OBSERVABILITY OPERATOR のアーキテクチャー	31
4.3. NETWORK OBSERVABILITY OPERATOR のステータスと設定の表示	32
第5章 NETWORK OBSERVABILITY OPERATOR の設定	34
5.1. FLOWCOLLECTOR リソースを表示する	34
5.2. KAFKA を使用した FLOW COLLECTOR リソースの設定	36
5.3. 強化されたネットワークフローデータをエクスポートする	37
5.4. FLOW COLLECTOR リソースの更新	38
5.5. クイックフィルターの設定	38
5.6. SR-IOV インターフェイストラフィックの監視の設定	39
5.7. リソース管理およびパフォーマンスに関する考慮事項	40
第6章 ネットワークポリシー	43
6.1. NETWORK OBSERVABILITY のためのネットワークポリシーの作成	43
6.2. ネットワークポリシーの例	44
第7章 ネットワークトラフィックの監視	45
7.1. OVERVIEW ビューからのネットワークトラフィックの監視	45
7.2. TRAFFIC FLOWS ビューからのネットワークトラフィックの監視	50
7.3. トポロジービューからのネットワークトラフィックの観察	58
7.4. ネットワークトラフィックのフィルタリング	59
第8章 ダッシュボードとアラートでのメトリクスの使用	61
8.1. NETWORK OBSERVABILITY メトリクスのダッシュボードの表示	61
8.2. 定義済みのメトリクス	61
8.3. NETWORK OBSERVABILITY メトリクス	61
8.4. アラートの作成	63

8.5. カスタムメトリクス	64
8.6. FLOWMETRIC API を使用したカスタムメトリクスの設定	64
8.7. FLOWMETRIC API を使用したカスタムグラフの設定	66
第9章 NETWORK OBSERVABILITY OPERATOR の監視	69
9.1. 健全性ダッシュボード	69
9.2. 健全性アラート	69
9.3. 健全性情報の表示	69
9.4. NETOBSERV ダッシュボードの LOKI レート制限アラートの作成	70
9.5. EBPf エージェントアラートの使用	71
第10章 スケジューリングリソース	73
10.1. 特定のノードにおける NETWORK OBSERVABILITY デプロイメント	73
第11章 NETWORK OBSERVABILITY CLI	75
11.1. NETWORK OBSERVABILITY CLI のインストール	75
11.2. NETWORK OBSERVABILITY CLI の使用	76
11.3. NETWORK OBSERVABILITY CLI (OC NETOBSERV) リファレンス	79
第12章 FLOWCOLLECTOR 設定パラメーター	83
12.1. FLOWCOLLECTOR API 仕様	83
第13章 FLOWMETRIC 設定パラメーター	145
13.1. FLOWMETRIC [FLOWS.NETOBSERV.IO/V1ALPHA1]	145
第14章 ネットワークフロー形式のリファレンス	153
14.1. ネットワークフロー形式のリファレンス	153
第15章 NETWORK OBSERVABILITY のトラブルシューティング	158
15.1. MUST-GATHER ツールの使用	158
15.2. OPENSIFT CONTAINER PLATFORM コンソールでのネットワークトラフィックメニューエントリーの設 定	158
15.3. KAFKA をインストールした後、FLOWLOGS-PIPELINE がネットワークフローを消費しない	160
15.4. BR-INT インターフェイスと BR-EX インターフェイスの両方からのネットワークフローが表示されない	160
15.5. NETWORK OBSERVABILITY コントローラマネージャー POD のメモリーが不足する	160
15.6. LOKI へのカスタムクエリーの実行	161
15.7. LOKI RESOURCEEXHAUSTED エラーのトラブルシューティング	162
15.8. LOKI の EMPTY RING エラー	163
15.9. リソースのトラブルシューティング	163
15.10. LOKISTACK レート制限エラー	163

第1章 NETWORK OBSERVABILITY OPERATOR リリースノート

Network Observability Operator を使用すると、管理者は OpenShift Container Platform クラスターのネットワークトラフィックフローを観察および分析できます。

これらのリリースノートは、OpenShift Container Platform での Network Observability Operator の開発を追跡します。

Network Observability Operator の概要は、[Network Observability Operator について](#) を参照してください。

1.1. NETWORK OBSERVABILITY OPERATOR 1.6.0

Network Observability Operator 1.6.0 では、次のアドバイザーを利用できます。

- [Network Observability Operator 1.6.0](#)

1.1.1. 新機能および機能拡張

1.1.1.1. Loki を使用しない場合の Network Observability Operator の使用の強化

Network Observability Operator を使用すると、Prometheus メトリクスを使用でき、ストレージのために Loki に依存する度合いが低下します。詳細は、[Loki を使用しない Network Observability](#) を参照してください。

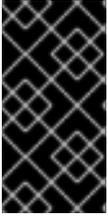
1.1.1.2. カスタムメトリクス API

FlowMetrics API を使用して、フローログデータからカスタムメトリクスを作成できます。フローログデータを Prometheus ラベルと組み合わせて使用することで、ダッシュボード上のクラスター情報をカスタマイズできます。識別する必要があるフローおよびメトリクスのサブネットに、カスタムラベルを追加できます。この機能拡張により、新しいラベル **SrcSubnetLabel** と **DstSubnetLabel** を使用して、フローログとメトリクスの両方に存在する外部トラフィックをより簡単に識別することもできます。外部トラフィックがある場合、これらのフィールドが空になるため、外部トラフィックを識別できません。詳細は、[カスタムメトリクス](#) と [FlowMetric API リファレンス](#) を参照してください。

1.1.1.3. eBPF のパフォーマンスの強化

次の更新により、CPU とメモリーの面で eBPF エージェントのパフォーマンスが向上しました。

- eBPF エージェントが、TC の代わりに TCX Webhook を使用するようになりました。
- **NetObserv/Health** ダッシュボードに、eBPF メトリクスを表示する新しいセクションがあります。
 - eBPF エージェントがフローをドロップしたときに、新しい eBPF メトリクスに基づいてアラートが通知されます。
- 重複したフローが削除されたため、Loki のストレージ需要が大幅に減少しました。ネットワークインターフェイス別の重複した複数のフローが、関連する一連のネットワークインターフェイスを含む重複排除された1つのフローになりました。



重要

重複したフローの更新により、Network Traffic テーブルの **Interface** および **Interface Direction** フィールドの名前が **Interfaces** および **Interface Directions** に変更されました。そのため、これらのフィールドを使用するブックマーク済みのクイックフィルターのクエリーを、**interfaces** および **ifdirections** に更新する必要があります。

詳細は、[eBPF エージェントアラートの使用](#) と [クイックフィルター](#) を参照してください。

1.1.1.4. eBPF 収集のルールベースのフィルタリング

ルールベースのフィルタリングを使用して、作成されるフローの量を削減できます。このオプションを有効にすると、eBPF エージェント統計の **Netobserv/Health** ダッシュボードに、**Filtered flows rate** ビューが表示されます。詳細は、[eBPF フローのルールフィルター](#) を参照してください。

1.1.2. テクノロジープレビュー機能

現在、今回のリリースに含まれる機能にはテクノロジープレビューのものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。これらの機能に関しては、Red Hat カスタマーポータル以下のサポート範囲を参照してください。

[テクノロジープレビュー機能のサポート範囲](#)

1.1.2.1. Network Observability CLI



警告

ネットワーク可観測性 CLI (**oc netobserv**)は一時的に利用できず、[OCPBUGS-36146](#) で解決することが期待されています。

Network Observability CLI を使用すると、Network Observability Operator をインストールしなくても、ネットワークトラフィックの問題をデバッグおよびトラブルシューティングできます。フローおよびパケットデータをリアルタイムでキャプチャーして可視化します。キャプチャー中に永続的なストレージは必要ありません。詳細は、[Network Observability CLI](#) および [Network Observability CLI 1.6.0](#) を参照してください。

1.1.3. バグ修正

- 以前は、**FlowMetrics** API 作成用の Operator Lifecycle Manager (OLM) フォームに、OpenShift Container Platform ドキュメントへの無効なリンクが表示されていました。このリンクの参照先が有効なページに更新されました。([NETOBSERV-1607](#))
- 以前は、Operator Hub の Network Observability Operator の説明に、ドキュメントへの無効なリンクが表示されていました。この修正により、このリンクは復元されます。([NETOBSERV-1544](#))
- 以前は、Loki が無効になっていて Loki **Mode** が **LokiStack** に設定されていても、または Loki の手動 TLS 設定が設定されていても、Network Observability Operator が Loki CA 証明書の読み取りを試行していました。この修正により、Loki が無効になっている場合、Loki 設定に設定

があっても Loki 証明書が読み取られなくなりました。(NETOBSERV-1647)

- 以前は、Network Observability Operator の **oc must-gather** プラグインが **amd64** アーキテクチャーでしか動作せず、他のすべてのアーキテクチャーでは失敗していました。これは、プラグインが **oc** バイナリーに **amd64** を使用していたためです。現在、Network Observability Operator **oc** の **must-gather** プラグインは、あらゆるアーキテクチャープラットフォームでログを収集します。
- 以前は、**not equal to** を使用して IP アドレスをフィルタリングすると、Network Observability Operator がリクエストエラーを返していました。現在は、IP アドレスと範囲が **equal** の場合でも **not equal to** の場合でも、IP フィルタリングが機能します。(NETOBSERV-1630)
- 以前は、ユーザーが管理者でなかった場合、エラーメッセージが Web コンソールの **Network Traffic** ビューで選択したタブと一致しませんでした。現在は、**user not admin** エラーがどのタブにも表示されるようになり、表示が改善されました。(NETOBSERV-1621)

1.1.4. 既知の問題

- eBPF エージェントの **PacketDrop** 機能が有効になっていて、サンプリングが 1 より大きい値に設定されている場合、ドロップされたバイト数とドロップされたパケット数の報告で、サンプリング設定が無視されます。これはドロップを見逃さないように意図的に行われていますが、副次的な影響として、報告されるドロップと非ドロップの比率が偏ったものになります。たとえば、**1:1000** などの非常に高いサンプリングレートでは、コンソールプラグインから見ると、ほぼすべてのトラフィックがドロップされているように見える可能性があります。(NETOBSERV-1676)
- **Overview** タブの **Manage panels** ポップアップウィンドウで、**total**、**bar**、**donut**、または **line** でフィルタリングしても、結果が表示されません。(NETOBSERV-1540)
- SR-IOV セカンダリーインターフェイスを作成してから eBPF エージェントをデプロイした場合、インターフェイスは検出されません。エージェントをデプロイしてから SR-IOV インターフェイスを作成した場合にのみ検出されます。(NETOBSERV-1697)
- Loki が無効になっている場合、OpenShift Web コンソールの **Topology** ビューで、関連機能が有効になっていない場合でも、ネットワークトポロジー図の横にあるスライダーに **Cluster** および **Zone** 集計オプションが常に表示されます。これらのスライダーオプションを無視する以外に、具体的な回避策はありません。(NETOBSERV-1705)
- Loki が無効になっているときに、OpenShift Web コンソールが初めて読み込まれると、**Request failed with status code 400 Loki is disabled** というエラーが表示される場合があります。回避策としては、**Topology** タブと **Overview** タブをクリックするなど、**Network Traffic** ページのコンテンツを何度か切り替える方法があります。エラーが表示されなくなるはずですが。(NETOBSERV-1706)

1.2. NETWORK OBSERVABILITY OPERATOR 1.5.0

Network Observability Operator 1.5.0 では、次のアドバイザーを利用できます。

- [Network Observability Operator 1.5.0](#)

1.2.1. 新機能および機能拡張

1.2.1.1. DNS 追跡の機能拡張

1.5 では、UDP に加えて TCP プロトコルもサポートされるようになりました。また、新しいダッシュボードが、Network Traffic ページの **Overview** ビューに追加されました。詳細は、[DNS 追跡の設定](#) および [DNS 追跡の使用](#) を参照してください。

1.2.1.2. ラウンドトリップタイム (RTT)

`fentry/tcp_rcv_published` Extended Berkeley Packet Filter (eBPF) フックポイントから取得した TCP ハンドシェイクのラウンドトリップタイム (RTT) を使用して、平滑化されたラウンドトリップタイム (SRTT) を読み取り、ネットワークフローを分析できます。Web コンソールの **Overview**、**Network Traffic**、および **Topology** ページで、ネットワークトラフィックを監視し、RTT メトリクス、フィルタリング、およびエッジラベルを使用してトラブルシューティングを行うことができます。詳細は、[RTT の概要](#) および [RTT の使用](#) を参照してください。

1.2.1.3. メトリクス、ダッシュボード、アラートの機能拡張

Observe → **Dashboards** → **NetObserv** の Network Observability メトリクスダッシュボードに、Prometheus アラートの作成に使用できる新しいメトリクスタイプがあります。利用可能なメトリクスを `includeList` 仕様で定義できるようになりました。以前のリリースでは、これらのメトリクスは `ignoreTags` 仕様で定義されていました。これらのメトリクスの完全なリストについては、[Network Observability メトリクス](#) を参照してください。

1.2.1.4. Loki を使用していない場合の Network Observability の向上

Loki を使用していない場合でも、DNS、パケットドロップ、および RTT メトリクスを使用して **Netobserv** ダッシュボードの Prometheus アラートを作成できます。旧バージョンの Network Observability 1.4 では、これらのメトリクスは、**Network Traffic**、**Overview**、および **Topology** ビューでのクエリーと分析にのみ使用できました。これらのビューを使用するには、Loki が必要でした。詳細は、[Network Observability メトリクス](#) を参照してください。

1.2.1.5. アベイラビリティゾーン

クラスターのアベイラビリティゾーンに関する情報を収集するように **FlowCollector** リソースを設定できます。この設定では、ノードに適用される `topology.kubernetes.io/zone` ラベル値を使用してネットワークフローデータをエンリッチします。詳細は、[アベイラビリティゾーンの使用](#) を参照してください。

1.2.1.6. 主な機能拡張

Network Observability Operator の 1.5 リリースでは、OpenShift Container Platform Web コンソールログインと Operator 設定が改良され、新機能が追加されています。

パフォーマンスの強化

- Kafka 使用時の eBPF のパフォーマンスを向上させるために、`spec.agent.ebpf.kafkaBatchSize` のデフォルトが **10MB** から **1MB** に変更されました。



重要

既存のインストールからアップグレードする場合、この新しい値は自動的に設定されません。アップグレード後に eBPF Agent のメモリー消費のパフォーマンスリグレッションが確認された場合は、`kafkaBatchSize` を減らして別の値にすることを検討してください。

Web コンソールの機能拡張:

- DNS と RTT の **Overview** ビューに新しいパネル (Min、Max、P90、P99) が追加されました。
- 新しいパネル表示オプションが追加されました。
 - 1つのパネルに焦点を当て、他のパネルの表示を小さくする。
 - グラフの種類を切り替える。
 - Top と Overall を表示する。
- **Custom time range** ポップアップウィンドウに収集遅延の警告が表示されます。
- **Manage panels** および **Manage columns** ポップアップウィンドウの内容の視認性が向上しました。
- Egress QoS の Differentiated Services Code Point (DSCP) フィールドを使用して、Web コンソールの **Network Traffic** ページの QoS DSCP をフィルタリングできます。

設定の機能拡張

- **spec.loki.mode** 仕様を **LokiStack** モードにすると、URL、TLS、クラスターロール、クラスターロールバインディング、および **authToken** 値を自動的に設定され、インストールが簡素化されます。**Manual** モードを使用すると、これらの設定をより詳細に制御できます。
- API バージョンが **flows.netobserv.io/v1beta1** から **flows.netobserv.io/v1beta2** に変更されます。

1.2.2. バグ修正

- 以前は、コンソールプラグインの自動登録が無効になっている場合、Web コンソールインターフェイスでコンソールプラグインを手動で登録することができませんでした。**FlowCollector** リソースの **spec.console.register** 値が **false** に設定されている場合、Operator がプラグインの登録をオーバーライドして消去します。この修正により、**spec.console.register** 値を **false** に設定しても、コンソールプラグインの登録または登録削除に影響しなくなりました。その結果、プラグインを手動で安全に登録できるようになりました。(NETOBSERV-1134)
- 以前は、デフォルトのメトリクス設定を使用すると、**NetObserv/Health** ダッシュボードに **Flows Overhead** という名前の空のグラフが表示されていました。このメトリクスを使用するには、**ignoreTags** リストから "namespaces-flows" と "namespaces" を削除する必要がありました。この修正により、デフォルトのメトリクス設定を使用する場合にこのメトリクスが表示されるようになります。(NETOBSERV-1351)
- 以前は、eBPF Agent を実行しているノードが、特定のクラスター設定で解決されませんでした。これにより連鎖的な影響が生じ、最終的にトラフィックメトリクスの一部を提供できなくなりました。この修正により、eBPF Agent のノード IP が、Pod のステータスから推測されて、Operator によって安全に提供されるようになりました。これにより、欠落していたメトリクスが復元されました。(NETOBSERV-1430)
- 以前は、Loki Operator の Loki エラー 'Input size too long' に、問題をトラブルシューティングするための追加情報が含まれていませんでした。この修正により、Web コンソールのエラーの隣にヘルプが直接表示され、詳細なガイダンスへの直接リンクが表示されるようになりました。(NETOBSERV-1464)
- 以前は、コンソールプラグインの読み取りタイムアウトが 30 秒に強制的に指定されていました。**FlowCollector v1beta2** API の更新により、この値を、Loki Operator の **queryTimeout** 制限に基づいて更新するように **spec.loki.readTimeout** 仕様を設定できるようになりました。(NETOBSERV-1443)

- 以前は、Operator バンドルが、CSV アノテーションによってサポートされている機能の一部 (features.operators.openshift.io/... など) を期待どおりに表示しませんでした。この修正により、これらのアノテーションが期待どおりに CSV に設定されるようになりました。(NETOBSERV-1305)
- 以前は、調整中に **FlowCollector** ステータスが **DeploymentInProgress** 状態と **Ready** 状態の間で変動することがありました。この修正により、基礎となるコンポーネントがすべて完全に準備完了した場合にのみ、ステータスが **Ready** になります。(NETOBSERV-1293)

1.2.3. 既知の問題

- Web コンソールにアクセスしようとする、OCP 4.14.10 のキャッシュの問題により、**Observe** ビューにアクセスできなくなります。Web コンソールに **Failed to get a valid plugin manifest from /api/plugins/monitoring-plugin/** というエラーメッセージが表示されま。推奨される回避策は、クラスターを最新のマイナーバージョンに更新することです。この回避策が機能しない場合は、こちらの [Red Hat ナレッジベースの記事 \(NETOBSERV-1493\)](#) で説明されている回避策を適用する必要があります。
- Network Observability Operator の 1.3.0 リリース以降、Operator をインストールすると、警告カーネルティントが表示されます。このエラーの理由は、Network Observability eBPF エージェントに、HashMap テーブル全体を事前割り当てするメモリ制約があることです。Operator eBPF エージェントは **BPF_F_NO_PREALLOC** フラグを設定し、HashMap がメモリを大幅に使用している際に事前割り当てが無効化されるようにします。

1.3. NETWORK OBSERVABILITY OPERATOR 1.4.2

Network Observability Operator 1.4.2 では、次のアドバイザリーを利用できます。

- [2023:6787 Network Observability Operator 1.4.2](#)

1.3.1. CVE

- [2023-39325](#)
- [2023-44487](#)

1.4. NETWORK OBSERVABILITY OPERATOR 1.4.1

Network Observability Operator 1.4.1 では、次のアドバイザリーを利用できます。

- [2023:5974 Network Observability Operator 1.4.1](#)

1.4.1. CVE

- [2023-44487](#)
- [2023-39325](#)
- [2023-29406](#)
- [2023-29409](#)
- [2023-39322](#)
- [2023-39318](#)

- [2023-39319](#)
- [2023-39321](#)

1.4.2. バグ修正

- 1.4 には、ネットワークフローデータを Kafka に送信するときに既知の問題がありました。Kafka メッセージキーが無視されたため、接続の追跡でエラーが発生していました。現在、キーはパーティショニングに使用されるため、同じ接続からの各フローが同じプロセッサに送信されます。(NETOBSERV-926)
- 1.4 で、同じノード上で実行されている Pod 間のフローを考慮するために、**Inner** 方向のフローが導入されました。**Inner** 方向のフローは、フローから派生して生成される Prometheus メトリクスでは考慮されなかったため、バイトレートとパケットレートが過小評価されていました。現在は派生メトリクスに **Inner** 方向のフローが含まれ、正しいバイトレートとパケットレートが提供されるようになりました。(NETOBSERV-1344)

1.5. NETWORK OBSERVABILITY OPERATOR 1.4.0

Network Observability Operator 1.4.0 では、次のアドバイザリーを利用できます。

- [RHSA-2023:5379 Network Observability Operator 1.4.0](#)

1.5.1. チャネルの削除

最新の Operator 更新を受信するには、チャネルを **v1.0.x** から **stable** に切り替える必要があります。**v1.0.x** チャネルは削除されました。

1.5.2. 新機能および機能拡張

1.5.2.1. 主な機能拡張

Network Observability Operator の 1.4 リリースでは、OpenShift Container Platform Web コンソールログインと Operator 設定が改良され、新機能が追加されています。

Web コンソールの機能拡張:

- **Query Options** に、重複したフローを表示するかどうかを選択するための **Duplicate flows** チェックボックスが追加されました。
- 送信元トラフィックおよび宛先トラフィックを、**↑ One-way**、**↑↓ Back-and-forth**、**Swap** のフィルターでフィルタリングできるようになりました。
- **Observe** → **Dashboards** → **NetObserv**、および **NetObserv / Health** の Network Observability メトリクスダッシュボードは次のように変更されます。
 - **NetObserv** ダッシュボードには、ノード、namespace、およびワークロードごとに、上位のバイト、送信パケット、受信パケットが表示されます。フローグラフはこのダッシュボードから削除されました。
 - **NetObserv/Health** ダッシュボードには、フローのオーバーヘッド以外にも、ノード、namespace、ワークロードごとの最大フローレートが表示されます。
 - インフラストラクチャーとアプリケーションのメトリクスは、namespace とワークロードの分割ビューで表示されます。

詳細は、[Network Observability メトリクス](#) と [クイックフィルター](#) を参照してください。

設定の機能拡張

- 証明書設定など、設定された ConfigMap または Secret 参照に対して異なる namespace を指定できるオプションが追加されました。
- **spec.processor.clusterName** パラメーターが追加されたため、クラスターの名前がフローデータに表示されるようになりました。これは、マルチクラスターコンテキストで役立ちます。OpenShift Container Platform を使用する場合は、自動的に決定されるように空のままにします。

詳細は、[Flow Collector のサンプルリソース](#) と [Flow Collector API リファレンス](#) を参照してください。

1.5.2.2. Loki を使用しない Network Observability

Network Observability Operator は、Loki なしでも機能し、使用できるようになりました。Loki がインストールされていない場合は、フローを KAFKA または IPFIX 形式にエクスポートし、Network Observability メトリクスダッシュボードに入力することのみ可能です。詳細は、[Loki を使用しない Network Observability](#) を参照してください。

1.5.2.3. DNS 追跡

1.4 では、Network Observability Operator は eBPF トレースポイントフックを使用して DNS 追跡を有効にします。Web コンソールの **Network Traffic** ページと **Overview** ページで、ネットワークの監視、セキュリティ分析の実施、DNS 問題のトラブルシューティングを行なえます。

詳細は、[DNS 追跡の設定](#) および [DNS 追跡の使用](#) を参照してください。

1.5.2.4. SR-IOV のサポート

Single Root I/O Virtualization (SR-IOV) デバイスを使用して、クラスターからトラフィックを収集できるようになりました。詳細は、[SR-IOV インターフェイストラフィックの監視の設定](#) を参照してください。

1.5.2.5. IPFIX エクスポートのサポート

eBPF が強化されたネットワークフローを IPFIX コレクターにエクスポートできるようになりました。詳細は、[強化されたネットワークフローデータのエクスポート](#) を参照してください。

1.5.2.6. パケットドロップ

Network Observability Operator の 1.4 リリースでは、eBPF トレースポイントフックを使用してパケットドロップの追跡を有効にできます。パケットドロップの原因を検出して分析し、ネットワークパフォーマンスを最適化するための決定を行えるようになりました。OpenShift Container Platform 4.14 以降では、ホストのドロップと OVS のドロップの両方が検出されます。OpenShift Container Platform 4.13 では、ホストのドロップのみが検出されます。詳細は、[パケットドロップ追跡の設定](#) および [パケットドロップの使用](#) を参照してください。

1.5.2.7. s390x アーキテクチャーのサポート

Network Observability Operator が、**s390x** アーキテクチャー上で実行できるようになりました。以前は、**amd64**、**ppc64le**、または **arm64** で実行されていました。

1.5.3. バグ修正

- これまで、Network Observability によってエクスポートされた Prometheus メトリクスは、重複する可能性のあるネットワークフローから計算されていました。その結果、関連するダッシュボード (**Observe** → **Dashboards**) でレートが2倍になる可能性がありました。ただし、**Network Traffic** ビューのダッシュボードは影響を受けていませんでした。現在は、メトリクスの計算前にネットワークフローがフィルタリングされて重複が排除されるため、ダッシュボードに正しいトラフィックレートが表示されます。([NETOBSERV-1131](#))
- 以前は、Network Observability Operator エージェントは、Multus または SR-IOV (デフォルト以外のネットワーク namespace) で設定されている場合、ネットワークインターフェイス上のトラフィックをキャプチャできませんでした。現在は、利用可能なすべてのネットワーク namespace が認識され、フローのキャプチャに使用されるため、SR-IOV のトラフィックをキャプチャできます。トラフィックを収集する場合は、**FlowCollector** および **SRIOVnetwork** カスタムリソースで [必要な設定](#) があります。([NETOBSERV-1283](#))
- 以前は、**Operators** → **Installed Operators** に表示される Network Observability Operator の詳細の **FlowCollector Status** フィールドで、デプロイメントの状態に関する誤った情報が報告されることがありました。ステータスフィールドには、改善されたメッセージと適切な状態が表示されるようになりました。イベントの履歴は、イベントの日付順に保存されます。([NETOBSERV-1224](#))
- 以前は、ネットワークトラフィックの負荷が増加すると、特定の eBPF Pod が OOM によって強制終了され、**CrashLoopBackOff** 状態になりました。現在は、**eBPF** agent のメモリーフットプリントが改善されたため、Pod が OOM によって強制終了されて **CrashLoopBackOff** 状態に遷移することはなくなりました。([NETOBSERV-975](#))
- 以前は、**processor.metrics.tls** が **PROVIDED** に設定されている場合、**insecureSkipVerify** オプションの値が強制的に **true** に設定されていました。現在は、**insecureSkipVerify** を **true** または **false** に設定し、必要に応じて CA 証明書を提供できるようになりました。([NETOBSERV-1087](#))

1.5.4. 既知の問題

- Network Observability Operator 1.2.0 リリース以降では、Loki Operator 5.6 を使用すると、Loki 証明書の変更が定期的に **flowlogs-pipeline** Pod に影響を及ぼすため、フローが Loki に書き込まれず、ドロップされます。この問題はしばらくすると自動的に修正されますが、Loki 証明書の移行中に一時的なフローデータの損失が発生します。この問題は、120 以上のノードを内包する大規模環境でのみ発生します。([NETOBSERV-980](#))
- 現在、**spec.agent.ebpf.features** に **DNSTracking** が含まれている場合、DNS パケットが大きいと、**eBPF** agent が最初のソケットバッファ (SKB) セグメント外で DNS ヘッダーを探す必要があります。これをサポートするには、**eBPF** agent の新しいヘルパー関数を実装する必要があります。現在、この問題に対する回避策はありません。([NETOBSERV-1304](#))
- 現在、**spec.agent.ebpf.features** に **DNSTracking** が含まれている場合、DNS over TCP パケットを扱うときに、**eBPF** agent が最初の SKB セグメント外で DNS ヘッダーを探す必要があります。これをサポートするには、**eBPF** agent の新しいヘルパー関数を実装する必要があります。現在、この問題に対する回避策はありません。([NETOBSERV-1245](#))
- 現在、**KAFKA** デプロイメントモデルを使用する場合、会話の追跡が設定されていると会話イベントが Kafka コンシューマー間で重複する可能性があり、その結果、会話の追跡に一貫性がなくなり、ボリュームデータが不正確になる可能性があります。そのため、**deploymentModel** が **KAFKA** に設定されている場合は、会話の追跡を設定することは推奨されません。([NETOBSERV-926](#))

- 現在、`processor.metrics.server.tls.type` が **PROVIDED** 証明書を使用するように設定されている場合、Operator の状態が不安定になり、パフォーマンスとリソース消費に影響を与える可能性があります。この問題が解決されるまでは **PROVIDED** 証明書を使用せず、代わりに自動生成された証明書を使用し、`processor.metrics.server.tls.type` を **AUTO** に設定することが推奨されます。(NETOBSERV-1293)
- Network Observability Operator の 1.3.0 リリース以降、Operator をインストールすると、警告カーネルティントが表示されます。このエラーの理由は、Network Observability eBPF エージェントに、HashMap テーブル全体を事前割り当てするメモリ制約があることです。Operator eBPF エージェントは **BPF_F_NO_PREALLOC** フラグを設定し、HashMap がメモリを大幅に使用している際に事前割り当てが無効化されるようにします。

1.6. NETWORK OBSERVABILITY OPERATOR 1.3.0

Network Observability Operator 1.3.0 では、次のアドバイザリーを利用できます。

- [RHSA-2023:3905 Network Observability Operator 1.3.0](#)

1.6.1. チャンネルの非推奨化

今後の Operator 更新を受信するには、チャンネルを **v1.0.x** から **stable** に切り替える必要があります。**v1.0.x** チャンネルは非推奨となり、次のリリースで削除される予定です。

1.6.2. 新機能および機能拡張

1.6.2.1. Network Observability におけるマルチテナンシー

- システム管理者は、Loki に保存されているフローへの個々のユーザーアクセスまたはグループアクセスを許可および制限できます。詳細は、[Network Observability におけるマルチテナンシー](#) を参照してください。

1.6.2.2. フローベースのメトリクスダッシュボード

- このリリースでは、OpenShift Container Platform クラスター内のネットワークフローの概要を表示する新しいダッシュボードが追加されています。詳細は、[Network Observability メトリクス](#) を参照してください。

1.6.2.3. must-gather ツールを使用したトラブルシューティング

- Network Observability Operator に関する情報を、トラブルシューティングで使用する `must-gather` データに追加できるようになりました。詳細は、[Network Observability の must-gather](#) を参照してください。

1.6.2.4. 複数のアーキテクチャーに対するサポートを開始

- Network Observability Operator は、**amd64**、**ppc64le**、または **arm64** アーキテクチャー上で実行できるようになりました。以前は、**amd64** 上でのみ動作しました。

1.6.3. 非推奨の機能

1.6.3.1. 非推奨の設定パラメーターの設定

Network Observability Operator 1.3 のリリースでは、**spec.Loki.authToken HOST** 設定が非推奨になりました。Loki Operator を使用する場合、**FORWARD** 設定のみを使用する必要があります。

1.6.4. バグ修正

- 以前は、Operator が CLI からインストールされた場合、Cluster Monitoring Operator がメトリクスを読み取るために必要な **Role** と **RoleBinding** が期待どおりにインストールされませんでした。この問題は、Operator が Web コンソールからインストールされた場合には発生しませんでした。現在は、どちらの方法で Operator をインストールしても、必要な **Role** と **RoleBinding** がインストールされます。(NETOBSERV-1003)
- バージョン 1.2 以降、Network Observability Operator は、フローの収集で問題が発生した場合にアラートを生成できます。以前は、バグのため、アラートを無効にするための関連設定である **spec.processor.metrics.disableAlerts** が期待どおりに動作せず、効果がない場合があります。現在、この設定は修正され、アラートを無効にできるようになりました。(NETOBSERV-976)
- 以前は、Network Observability の **spec.loki.authToken** が **DISABLED** に設定されている場合、**kubeadmin** クラスター管理者のみがネットワークフローを表示できました。他のタイプのクラスター管理者は認可エラーを受け取りました。これで、クラスター管理者は誰でもネットワークフローを表示できるようになりました。(NETOBSERV-972)
- 以前は、バグが原因でユーザーは **spec.consolePlugin.portNaming.enable** を **false** に設定できませんでした。現在は、これを **false** に設定すると、ポートからサービスへの名前変換を無効にできます。(NETOBSERV-971)
- 以前は、設定が間違っていたため、コンソールプラグインが公開するメトリクスは、Cluster Monitoring Operator (Prometheus) によって収集されませんでした。現在は設定が修正され、コンソールプラグインメトリクスが正しく収集され、OpenShift Container Platform Web コンソールからアクセスできるようになりました。(NETOBSERV-765)
- 以前は、**FlowCollector** で **processor.metrics.tls** が **AUTO** に設定されている場合、**flowlogs-pipeline servicemonitor** は適切な TLS スキームを許可せず、メトリクスは Web コンソールに表示されませんでした。この問題は AUTO モードで修正されました。(NETOBSERV-1070)
- 以前は、Kafka や Loki に使用されるような証明書設定では、namespace フィールドを指定できず、Network Observability がデプロイされているのと同じ namespace に証明書が存在する必要があります。さらに、TLS/mTLS で Kafka を使用する場合、ユーザーは **eBPF** agent Pod がデプロイされている特権付き namespace に証明書を手動でコピーし、証明書のローテーションを行う場合などに手動で証明書の更新を管理する必要があります。現在は、**FlowCollector** リソースに証明書の namespace フィールドを追加することで、Network Observability のセットアップが簡素化されています。その結果、ユーザーは Network Observability namespace に証明書を手動でコピーすることなく、Loki または Kafka を別の namespace にインストールできるようになりました。元の証明書は監視されているため、必要に応じてコピーが自動的に更新されます。(NETOBSERV-773)
- 以前は、SCTP、ICMPv4、および ICMPv6 プロトコルは Network Observability エージェントのカバレッジに含まれていなかったため、ネットワークフローのカバレッジもあまり包括的ではありませんでした。これらのプロトコルを使用することで、フローカバレッジが向上することが確認されています。(NETOBSERV-934)

1.6.5. 既知の問題

- **FlowCollector** で **processor.metrics.tls** が **PROVIDED** に設定されている場合、**flowlogs-pipelineservicemonitor** は TLS スキームに適用されません。(NETOBSERV-1087)

- Network Observability Operator 1.2.0 リリース以降では、Loki Operator 5.6 を使用すると、Loki 証明書の変更が定期的に **flowlogs-pipeline** Pod に影響を及ぼすため、フローが Loki に書き込まれず、ドロップされます。この問題はしばらくすると自動的に修正されますが、Loki 証明書の移行中に一時的なフローデータの損失が発生します。この問題は、120 以上のノードを内包する大規模環境でのみ発生します。(NETOBSERV-980)
- Operator のインストール時に、警告のカーネルテイントが表示される場合があります。このエラーの理由は、Network Observability eBPF エージェントに、HashMap テーブル全体を事前割り当てするメモリ制約があることです。Operator eBPF エージェントは **BPF_F_NO_PREALLOC** フラグを設定し、HashMap がメモリを大幅に使用している際に事前割り当てが無効化されるようにします。

1.7. NETWORK OBSERVABILITY OPERATOR 1.2.0

Network Observability Operator 1.2.0 では、次のアドバイザリーを利用できます。

- [RHSA-2023:1817 Network Observability Operator 1.2.0](#)

1.7.1. 次の更新の準備

インストールされた Operator のサブスクリプションは、Operator の更新を追跡および受信する更新チャンネルを指定します。Network Observability Operator の 1.2 リリースまでは、利用可能なチャンネルは **v1.0.x** だけでした。Network Observability Operator の 1.2 リリースでは、更新の追跡および受信用に **stable** 更新チャンネルが導入されました。今後の Operator 更新を受信するには、チャンネルを **v1.0.x** から **stable** に切り替える必要があります。**v1.0.x** チャンネルは非推奨となり、次のリリースで削除される予定です。

1.7.2. 新機能および機能拡張

1.7.2.1. Traffic Flows ビューのヒストグラム

- 経時的なフローのヒストグラムバーグラフを表示するように選択できるようになりました。ヒストグラムを使用すると、Loki クエリー制限に達することなくフロー履歴を可視化できます。詳細は、[ヒストグラムの使用](#) を参照してください。

1.7.2.2. 会話の追跡

- ログタイプでフローをクエリーできるようになりました。これにより、同じ会話に含まれるネットワークフローをグループ化できるようになりました。詳細は、[会話の使用](#) を参照してください。

1.7.2.3. Network Observability のヘルスアラート

- Network Observability Operator は、書き込み段階でのエラーが原因で **flowlogs-pipeline** がフローをドロップする場合、または Loki 取り込みレート制限に達した場合、自動アラートを作成するようになりました。詳細は、[健全性ダッシュボード](#) を参照してください。

1.7.3. バグ修正

- これまでは、FlowCollector 仕様の **namespace** の値を変更すると、以前の namespace で実行されている **eBPF** agent Pod が適切に削除されませんでした。今は、以前の namespace で実行されている Pod も適切に削除されるようになりました。(NETOBSERV-774)
- これまでは、FlowCollector 仕様 (Loki セクションなど) の **caCert.name** 値を変更しても、

FlowLogs-Pipeline Pod および Console プラグイン Pod が再起動されないため、設定の変更が認識されませんでした。今は、Pod が再起動されるため、設定の変更が適用されるようになりました。(NETOBSERV-772)

- これまでは、異なるノードで実行されている Pod 間のネットワークフローは、異なるネットワークインターフェイスでキャプチャーされるため、重複が正しく認識されないことがありました。その結果、コンソールプラグインに表示されるメトリクスが過大に見積もられていました。現在は、フローが重複として正しく識別され、コンソールプラグインで正確なメトリクスが表示されます。(NETOBSERV-755)
- コンソールプラグインのレポーターオプションは、送信元ノードまたは宛先ノードのいずれかの観測点に基づいてフローをフィルタリングするために使用されます。以前は、このオプションはノードの観測点に関係なくフローを混合していました。これは、ネットワークフローがノードレベルで Ingress または Egress として誤って報告されることが原因でした。これで、ネットワークフロー方向のレポートが正しくなりました。レポーターオプションは、期待どおり、ソース観測点または宛先観測点をフィルターします。(NETOBSERV-696)
- 以前は、フローを gRPC+protobuf リクエストとしてプロセッサに直接送信するように設定されたエージェントの場合、送信されたペイロードが大きすぎる可能性があり、プロセッサの gRPC サーバーによって拒否されました。これは、非常に高負荷のシナリオで、エージェントの一部の設定でのみ発生しました。エージェントは、次のようなエラーメッセージをログに記録しました: `grpc: max より大きいメッセージを受信しました`。その結果、それらのフローに関する情報が損失しました。現在、gRPC ペイロードは、サイズがしきい値を超えると、いくつかのメッセージに分割されます。その結果、サーバーは接続を維持します。(NETOBSERV-617)

1.7.4. 既知の問題

- Loki Operator 5.6 を使用する Network Observability Operator の 1.2.0 リリースでは、Loki 証明書の移行が定期的に `flowlogs-pipeline` Pod に影響を及ぼし、その結果、Loki に書き込まれるフローではなくフローがドロップされます。この問題はしばらくすると自動的に修正されますが、依然として Loki 証明書の移行中に一時的なフローデータの損失が発生します。(NETOBSERV-980)

1.7.5. 主な技術上の変更点

- 以前は、カスタム namespace を使用して Network Observability Operator をインストールできました。このリリースでは、`ClusterServiceVersion` を変更する `conversion webhook` が導入されています。この変更により、使用可能なすべての namespace がリストされなくなりました。さらに、Operator メトリクス収集を有効にするには、`openshift-operators` namespace など、他の Operator と共有される namespace は使用できません。ここで、Operator を `openshift-netobserv-operator` namespace にインストールする必要があります。以前にカスタム namespace を使用して Network Observability Operator をインストールした場合、新しい Operator バージョンに自動的にアップグレードすることはできません。以前にカスタム namespace を使用して Operator をインストールした場合は、インストールされた Operator のインスタンスを削除し、`openshift-netobserv-operator` namespace に Operator を再インストールする必要があります。一般的に使用される `netobserv` namespace などのカスタム namespace は、`FlowCollector`、Loki、Kafka、およびその他のプラグインでも引き続き使用できることに注意することが重要です。(NETOBSERV-907)(NETOBSERV-956)

1.8. NETWORK OBSERVABILITY OPERATOR 1.1.0

Network Observability Operator 1.1.0 については、次のアドバイザリーを利用できます。

- [RHSA-2023:0786 Network Observability Operator セキュリティアドバイザリーの更新](#)

Network Observability Operator は現在安定しており、リリースチャンネルは **v1.1.0** にアップグレードされています。

1.8.1. バグ修正

- 以前は、Loki の **authToken** 設定が **FORWARD** モードに設定されていない限り、認証が適用されず、OpenShift Container Platform クラスター内の OpenShift Container Platform コンソールに接続できるすべてのユーザーが認証なしでフローを取得できました。現在は、Loki の **authToken** モードに関係なく、クラスター管理者のみがフローを取得できます。
([BZ#2169468](#))

第2章 NETWORK OBSERVABILITY について

Red Hat は、OpenShift Container Platform クラスターのネットワークトラフィックを監視する Network Observability Operator をクラスター管理者に提供します。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 の情報でエンリッチされます。ネットワークフローは、Prometheus メトリクスとして、または Loki のログとして利用できます。保存されたネットワークフロー情報は 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。

2.1. NETWORK OBSERVABILITY OPERATOR のオプションの依存関係

- **Loki Operator:** Loki は、収集されたすべてのフローを最大の詳細度で保存するのに使用できるバックエンドです。[Loki を使用せずに Network Observability](#) を使用することも選択できますが、その場合はリンク先のセクションで説明されているいくつかの事項を考慮する必要があります。Loki をインストールする場合は、Red Hat がサポートする Loki Operator を使用することを推奨します。
- **AMQ Streams Operator:** Kafka は、大規模なデプロイメント向けに OpenShift Container Platform クラスターにスケラビリティ、復元力、高可用性を提供します。Kafka を使用することを選択する場合は、Red Hat がサポートする AMQ Streams Operator を使用することが推奨されます。

2.2. NETWORK OBSERVABILITY OPERATOR

Network Observability Operator は Flow Collector API カスタムリソース定義を提供します。Flow Collector インスタンスは、ネットワークフロー収集の設定を可能にするクラスタースコープのリソースです。Flow Collector インスタンスは、モニタリングパイプラインを形成する Pod とサービスをデプロイします。モニタリングパイプラインでは、ネットワークフローが収集され、Loki への保存前または Prometheus メトリクスの生成前に、Kubernetes メタデータでエンリッチされます。**daemonset** オブジェクトとしてデプロイされる eBPF エージェントが、ネットワークフローを作成します。

2.3. OPENSIFT CONTAINER PLATFORM コンソール統合

OpenShift Container Platform コンソール統合は、概要、トポロジービュー、およびトラフィックフローテーブルを提供します。

2.3.1. Network Observability メトリクスのダッシュボード

OpenShift Container Platform コンソールの **Overview** タブでは、クラスター上のネットワークトラフィックフローの集約された全体的なメトリクスを表示できます。ゾーン、ノード、namespace、所有者、Pod、サービスごとに情報を表示できます。フィルターと表示オプションにより、メトリクスをさらに絞り込むことができます。詳細は、[Overview ビューからのネットワークトラフィックの監視](#) を参照してください。

Observe → Dashboards の **Netobserv** ダッシュボードには、OpenShift Container Platform クラスター内のネットワークフローの簡易的な概要が表示されます。**Netobserv/Health** ダッシュボードは、Operator の健全性に関するメトリクスを提供します。詳細は、[Network Observability メトリクス](#) および [健全性情報の表示](#) を参照してください。

2.3.2. Network Observability トポロジービュー

OpenShift Container Platform コンソールは、ネットワークフローとトラフィック量をグラフィカルに

表示する **Topology** タブを提供します。トポロジービューは、OpenShift Container Platform コンポーネント間のトラフィックをネットワークグラフとして表します。フィルターと表示オプションを使用して、グラフを絞り込むことができます。ゾーン、ノード、namespace、所有者、Pod、サービスの情報にアクセスできます。

2.3.3. トラフィックフローテーブル

トラフィックフローテーブルビューは、生のフロー、集約されていないフィルタリングオプション、および設定可能な列のビューを提供します。OpenShift Container Platform コンソールは、ネットワークフローのデータとトラフィック量を表示する **Traffic flows** タブを提供します。

2.4. NETWORK OBSERVABILITY CLI

Network Observability CLI (**oc netobserv**) を使用すると、Network Observability に関するネットワークの問題を迅速にデバッグおよびトラブルシューティングできます。Network Observability CLI は、eBPF エージェントを利用して収集したデータを一時的なコレクター Pod にストリーミングするフローおよびパケット可視化ツールです。キャプチャー中に永続的なストレージは必要ありません。実行後、出力がローカルマシンに転送されます。そのため、Network Observability Operator をインストールしなくても、パケットとフローデータをすばやくライブで把握できます。

第3章 NETWORK OBSERVABILITY OPERATOR のインストール

Network Observability Operator を使用する場合、前提条件として Loki のインストールが推奨されます。[Loki を使用せずに Network Observability](#) を使用することも選択できますが、その場合はリンクした前述のセクションで説明されているいくつかの事項を考慮する必要があります。

Loki Operator は、マルチテナンシーと認証を実装するゲートウェイを Loki と統合して、データフローストレージを実現します。**LokiStack** リソースは、スケーラブルで高可用性のマルチテナントログ集約システムである Loki と、OpenShift Container Platform 認証を備えた Web プロキシを管理します。**LokiStack** プロキシは、OpenShift Container Platform 認証を使用してマルチテナンシーを適用し、Loki ログストアでのデータの保存とインデックス作成を容易にします。



注記

Loki Operator は、[LokiStack ログストアの設定](#) にも使用できます。Network Observability Operator には、ログインとは別の専用の LokiStack が必要です。

3.1. LOKI を使用しない NETWORK OBSERVABILITY

Loki のインストール手順を実行せず、直接「Network Observability Operator のインストール」を実行することで、Loki なしで Network Observability を使用できます。フローを Kafka コンシューマーまたは IPFIX コレクターのみにエクスポートする場合、またはダッシュボードメトリクスのみ必要な場合は、Loki をインストールしたり、Loki 用のストレージを提供したりする必要はありません。次の表は、Loki を使用した場合と使用しない場合の利用可能な機能を比較しています。

表3.1 Loki を使用した場合と使用しない場合の使用可能な機能の比較

	Loki を使用する場合	Loki を使用しない場合
エクスポーター	✓	✓
マルチテナンシー	✓	✗
完全なフィルタリングと集計機能 ^[1]	✓	✗
部分的なフィルタリングと集計機能 ^[2]	✓	✓
フローベースのメトリクスとダッシュボード	✓	✓
Traffic flows ビューの概要 ^[3]	✓	✓
Traffic flows ビューテーブル	✓	✗
Topology ビュー	✓	✓

	Loki を使用する場合	Loki を使用しない場合
OpenShift Container Platform コンソールの Network Traffic タブの統合	✓	✓

1. Pod ごとなど。
2. ワークロードまたは namespace ごとなど。
3. パケットドロップの統計情報は Loki でのみ利用可能です。

関連情報

- [強化されたネットワークフローデータのエクスポート](#)

3.2. LOKI OPERATOR のインストール

Network Observability でサポートされている Loki Operator のバージョンは、[Loki Operator バージョン 5.7 以降](#) です。これらのバージョンでは、**openshift-network** テナント設定モードを使用して **LokiStack** インスタンスを作成する機能が提供されており、Network Observability に対する完全に自動化されたクラスター内認証および認可がサポートされています。Loki をインストールするにはいくつかの方法があります。そのうちの1つが、OpenShift Container Platform Web コンソールの Operator Hub を使用する方法です。

前提条件

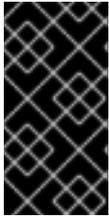
- 対応ログストア (AWS S3、Google Cloud Storage、Azure、Swift、Minio、OpenShift Data Foundation)
- OpenShift Container Platform 4.10 以上
- Linux カーネル 4.18 以降

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
2. 使用可能な Operator のリストから **Loki Operator** を選択し、**Install** をクリックします。
3. **Installation Mode** で、**All namespaces on the cluster** を選択します。

検証

1. Loki Operator がインストールされていることを確認します。**Operators → Installed Operators** ページにアクセスして、**Loki Operator** を探します。
2. **Loki Operator** がすべてのプロジェクトで **Succeeded** の **Status** でリストされていることを確認します。



重要

Loki をアンインストールするには、Loki のインストールに使用した方法に対応するアンインストールプロセスを参照してください。**ClusterRole** と **ClusterRoleBindings**、オブジェクトストアに格納されたデータ、および削除する必要のある永続ボリュームが残っている可能性があります。

3.2.1. Loki ストレージのシークレットの作成

Loki Operator は、AWS S3、Google Cloud Storage、Azure、Swift、Minio、OpenShift Data Foundation など、いくつかのログストレージオプションをサポートしています。次の例は、AWS S3 ストレージのシークレットを作成する方法を示しています。この例で作成されたシークレット **loki-s3** は、「LokiStack リソースの作成」で参照されています。このシークレットは、Web コンソールまたは CLI で作成できます。

1. Web コンソールを使用して、**Project** → **All Projects** ドロップダウンに移動し、**Create Project** を選択します。プロジェクトに **netobserv** という名前を付けて、**Create** をクリックします。
2. 右上隅にあるインポートアイコン + に移動します。YAML ファイルをエディターにペーストします。
以下は、S3 ストレージのシークレット YAML ファイルの例です。

```

apiVersion: v1
kind: Secret
metadata:
  name: loki-s3
  namespace: netobserv ①
stringData:
  access_key_id: QUtJQUIPU0ZPRE5ON0VYQU1QTEUK
  access_key_secret:
d0phbHJYVXRuRkVNSS9LN01ERU5HL2JQeFJmaUNZRvHBTvBMRUtFWQo=
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1

```

- ① このドキュメントに記載されているインストール例では、すべてのコンポーネントで同じ namespace である **netobserv** を使用しています。オプションで、異なるコンポーネントで異なる namespace を使用できます。

検証

- シークレットを作成すると、Web コンソールの **Workloads** → **Secrets** の下に一覧表示されます。

関連情報

- [Flow Collector API リファレンス](#)
- [Flow Collector のサンプルリソース](#)
- [Loki オブジェクトストレージ](#)

3.2.2. LokiStack カスタムリソースの作成

Web コンソールまたは OpenShift CLI (**oc**) を使用して namespace または新しいプロジェクトを作成し、**LokiStack** カスタムリソース (CR) をデプロイできます。

手順

1. **Operators** → **Installed Operators** に移動し、**Project** ドロップダウンから **All projects** を表示します。
2. **Loki Operator** を探します。詳細の **Provided APIs** で、**LokiStack** を選択します。
3. **Create LokiStack** をクリックします。
4. **Form View** または **YAML view** で次のフィールドが指定されていることを確認します。

```

apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: loki
  namespace: netobserv ①
spec:
  size: 1x.small ②
  storage:
    schemas:
      - version: v12
        effectiveDate: '2022-06-01'
    secret:
      name: loki-s3
      type: s3
  storageClassName: gp3 ③
  tenants:
    mode: openshift-network

```

- ① このドキュメントに記載されているインストール例では、すべてのコンポーネントで同じ namespace である **netobserv** を使用しています。必要に応じて、別の namespace を使用できます。
- ② デプロイメントサイズを指定します。Loki Operator 5.8 以降のバージョンでは、Loki の実稼働インスタンスでサポートされているサイズオプションは **1x.extra-small**、**1x.small**、または **1x.medium** です。



重要

デプロイメントサイズの **1x** の数は変更できません。

- ③ **ReadWriteOnce** アクセスモードのクラスターで使用可能なストレージクラス名を使用します。**oc get storageclasses** を使用して、クラスターで利用できるものを確認できます。



重要

ログ記録に使用したのと同じ **LokiStack** CR を再利用しないでください。

5. **Create** をクリックします。

3.2.3. cluster-admin ユーザーロールの新規グループの作成



重要

cluster-admin ユーザーとして複数の namespace のアプリケーションログをクエリーすると、クラスター内のすべての namespace の文字数の合計が 5120 を超え、**Parse error: input size too long (XXXX > 5120)** エラーが発生します。LokiStack のログへのアクセスをより適切に制御するには、**cluster-admin** ユーザーを **cluster-admin** グループのメンバーにします。**cluster-admin** グループが存在しない場合は、作成して必要なユーザーを追加します。

次の手順を使用して、**cluster-admin** 権限のあるユーザー用に、新しいグループを作成します。

手順

1. 以下のコマンドを入力して新規グループを作成します。

```
$ oc adm groups new cluster-admin
```

2. 以下のコマンドを実行して、必要なユーザーを **cluster-admin** グループに追加します。

```
$ oc adm groups add-users cluster-admin <username>
```

3. 以下のコマンドを実行して **cluster-admin** ユーザーロールをグループに追加します。

```
$ oc adm policy add-cluster-role-to-group cluster-admin cluster-admin
```

3.2.4. カスタム管理者グループのアクセス権

多数のユーザーが広範な権限を必要とする大規模デプロイメントの場合は、**adminGroup** フィールドを使用してカスタムグループを作成できます。**LokiStack** CR の **adminGroups** フィールドで指定されたグループのメンバーであるユーザーは、管理者とみなされます。**cluster-logging-application-view** ロールも割り当てられている管理者ユーザーは、すべての namespace のすべてのアプリケーションログにアクセスできます。

LokiStack CR の例

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  tenants:
    mode: openshift-network ①
  openshift:
    adminGroups: ②
    - cluster-admin
    - custom-admin-group ③
```

- ① カスタム管理者グループは、このモードでのみ使用できます。

- 2 このフィールドに空のリスト値 [] を入力すると、管理者グループが無効になります。
- 3 デフォルトのグループ (**system:cluster-admins**、**cluster-admin**、**dedicated-admin**) をオーバーライドします。

3.2.5. Loki デプロイメントのサイズ

Loki のサイズは **1x.<size>** の形式に従います。この場合の **1x** はインスタンスの数を、**<size>** は性能を指定します。



重要

デプロイメントサイズの **1x** の数は変更できません。

表3.2 Loki のサイズ

	1x.demo	1x.extra-small	1x.small	1x.medium
データ転送	デモ使用のみ	100 GB/日	500 GB/日	2 TB/日
1秒あたりのクエリー数 (QPS)	デモ使用のみ	200 ミリ秒で 1-25 QPS	200 ミリ秒で 25-50 QPS	200 ミリ秒で 25-75 QPS
レプリケーション係数	なし	2	2	2
合計 CPU 要求	なし	仮想 CPU 14 個	仮想 CPU 34 個	仮想 CPU 54 個
合計メモリー要求	なし	31 Gi	67 Gi	139 Gi
合計ディスク要求	40Gi	430 Gi	430 Gi	590 Gi

3.2.6. LokiStack の取り込み制限とヘルスアラート

LokiStack インスタンスには、設定されたサイズに応じたデフォルト設定が付属しています。取り込みやクエリーの制限など、これらの設定の一部を上書きすることができます。コンソールプラグインまたは **flowlogs-pipeline** ログに Loki エラーが表示される場合は、それらを更新することを推奨します。これらの制限に達すると、Web コンソールの自動アラートで通知されます。

設定された制限の例を次に示します。

```
spec:
  limits:
    global:
      ingestion:
        ingestionBurstSize: 40
        ingestionRate: 20
        maxGlobalStreamsPerTenant: 25000
    queries:
```

```
maxChunksPerQuery: 2000000
maxEntriesLimitPerQuery: 10000
maxQuerySeries: 3000
```

これらの設定の詳細は、[LokiStack API リファレンス](#) を参照してください。

3.2.7. Network Observability でのマルチテナンシーの有効化

Network Observability Operator のマルチテナンシーにより、Loki に保存されているフローへのユーザーアクセスまたはグループアクセスが個別に許可および制限されます。プロジェクト管理者のアクセスが有効になっています。一部の namespace へのアクセスが制限されているプロジェクト管理者は、それらの namespace のフローのみにアクセスできます。

前提条件

- [Loki Operator バージョン 5.7](#) 以降をインストールしている。
- 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、**user1** に読み取り権限を付与します。

```
$ oc adm policy add-cluster-role-to-user netobserv-reader user1
```

現在、データは許可されたユーザー namespace のみに制限されています。たとえば、単一の namespace にアクセスできるユーザーは、この namespace 内部のフローすべてと、この namespace から出入りするフローを表示できます。プロジェクト管理者は、OpenShift Container Platform コンソールの Administrator パースペクティブにアクセスして、Network Flows Traffic ページにアクセスできます。

3.3. NETWORK OBSERVABILITY OPERATOR のインストール

OpenShift Container Platform Web コンソール Operator Hub を使用して Network Observability Operator をインストールできます。Operator をインストールすると、**FlowCollector** カスタムリソース定義 (CRD) が提供されます。**FlowCollector** を作成するときに、Web コンソールで仕様を設定できます。



重要

Operator の実際のメモリ消費量は、クラスターのサイズとデプロイされたリソースの数によって異なります。それに応じて、メモリ消費量を調整する必要がある場合があります。詳細は、「Flow Collector 設定に関する重要な考慮事項」セクションの「Network Observability コントローラマネージャー Pod のメモリが不足する」を参照してください。

前提条件

- Loki を使用する場合は、[Loki Operator バージョン 5.7](#) 以降 をインストールしている。
- **cluster-admin** 権限を持っている必要があります。
- サポートされているアーキテクチャーである **amd64**、**ppc64le**、**arm64**、**s390x** のいずれか。

- Red Hat Enterprise Linux (RHEL) 9 でサポートされる任意の CPU。
- OVN-Kubernetes または OpenShift SDN をメインネットワークプラグインとして設定し、必要に応じて Multus および SR-IOV を使用したセカンダリーインターフェイスを使用している。



注記

さらに、このインストール例では、すべてのコンポーネントで使用される **netobserv** namespace を使用します。必要に応じて、別の namespace を使用できます。

手順

1. OpenShift Container Platform Web コンソールで、**Operators** → **OperatorHub** をクリックします。
2. **OperatorHub** で使用可能な Operator のリストから **Network Observability Operator** を選択し、**Install** をクリックします。
3. **Enable Operator recommended cluster monitoring on this Namespace** チェックボックスを選択します。
4. **Operators** → **Installed Operators** に移動します。Network Observability の Provided APIs で、**Flow Collector** リンクを選択します。
5. **Flow Collector** タブに移動し、**Create FlowCollector** をクリックします。フォームビューで次の選択を行います。
 - a. **spec.agent.ebpf.Sampling**: フローのサンプリングサイズを指定します。サンプリングサイズが小さいほど、リソース使用率への影響が大きくなります。詳細は、「FlowCollector API リファレンス」の **spec.agent.ebpf** を参照してください。
 - b. Loki を使用している場合は、次の仕様を設定します。
 - i. **spec.loki.mode**: これを **LokiStack** モードに設定すると、URL、TLS、クラスターロール、クラスターロールバインディング、および **authToken** 値が自動的に設定されます。または、**Manual** モードを使用すると、これらの設定をより詳細に制御できます。
 - ii. **spec.loki.lokistack.name**: これは **LokiStack** リソースの名前に設定します。このドキュメントでは、**loki** を使用します。
 - c. オプション: 使用している環境が大規模な場合は、回復性とスケーラビリティが高い方法でデータを転送するために、Kafka を使用して **FlowCollector** を設定することを検討してください。「Flow Collector 設定に関する重要な考慮事項」セクションの「Kafka ストレージを使用した Flow Collector リソースの設定」を参照してください。
 - d. オプション: 次の **FlowCollector** 作成手順に進む前に、他のオプションを設定します。たとえば、Loki を使用しないことを選択した場合は、Kafka または IPFIX へのフローのエクスポートを設定できます。「Flow Collector 設定に関する重要な考慮事項」セクションの「強化されたネットワークフローデータを Kafka および IPFIX にエクスポートする」などを参照してください。
6. **Create** をクリックします。

検証

これが成功したことを確認するには、**Observe** に移動すると、オプションに **Network Traffic** が表示されます。

OpenShift Container Platform クラスター内にアプリケーショントラフィックがない場合は、デフォルトのフィルターが "No results" と表示され、視覚的なフローが発生しないことがあります。フィルター選択の横にある **Clear all filters** を選択して、フローを表示します。

3.4. FLOW COLLECTOR 設定に関する重要な考慮事項

FlowCollector インスタンスを作成すると、それを再設定することはできますが、Pod が終了して再作成されるため、中断が生じる可能性があります。そのため、初めて **FlowCollector** を作成する際には、以下のオプションを設定することを検討してください。

- [Kafka を使用した Flow Collector リソースの設定](#)
- [強化されたネットワークフローデータを Kafka または IPFIX にエクスポート](#)
- [SR-IOV インターフェイストラフィックの監視の設定](#)
- [会話追跡の使用](#)
- [DNS 追跡の使用](#)
- [パケットドロップの使用](#)

関連情報

Flow Collector の仕様や、Network Observability Operator のアーキテクチャーとリソースの使用に関する全般的な情報については、次のリソースを参照してください。

- [Flow Collector API リファレンス](#)
- [Flow Collector のサンプルリソース](#)
- [リソースの留意事項](#)
- [Network Observability コントローラマネージャー Pod のメモリ不足のトラブルシューティング](#)
- [Network Observability アーキテクチャー](#)

3.5. KAFKA のインストール (オプション)

Kafka Operator は、大規模な環境でサポートされています。Kafka は、回復性とスケーラビリティの高い方法でネットワークフローデータを転送するために、高スループットかつ低遅延のデータフィードを提供します。Loki Operator および Network Observability Operator がインストールされたのと同じように、Kafka Operator を Operator Hub から [Red Hat AMQ Streams](#) としてインストールできます。Kafka をストレージオプションとして設定する場合は、「[Kafka を使用した FlowCollector リソースの設定](#)」を参照してください。



注記

Kafka をアンインストールするには、インストールに使用した方法に対応するアンインストールプロセスを参照してください。

関連情報

[Kafka を使用した FlowCollector リソースの設定](#)

3.6. NETWORK OBSERVABILITY OPERATOR のアンインストール

Network Observability Operator は、**Operators → Installed Operators** エリアで作業する OpenShift Container Platform Web コンソール Operator Hub を使用してアンインストールできます。

手順

1. **FlowCollector** カスタムリソースを削除します。
 - a. **Provided APIs** 列の **Network Observability Operator** の横にある **Flow Collector** をクリックします。
 - b. **cluster** のオプションメニュー  をクリックし、**Delete FlowCollector** を選択します。
2. Network Observability Operator をアンインストールします。
 - a. **Operators → Installed Operators** エリアに戻ります。
 - b. **Network Observability Operator** の隣にあるオプションメニュー  をクリックし、**Uninstall Operator** を選択します。
 - c. **Home → Projects** を選択し、**openshift-netobserv-operator** を選択します。
 - d. **Actions** に移動し、**Delete Project** を選択します。
3. **FlowCollector** カスタムリソース定義 (CRD) を削除します。
 - a. **Administration → CustomResourceDefinitions** に移動します。
 - b. **FlowCollector** を探し、オプションメニュー  をクリックします。
 - c. **Delete CustomResourceDefinition** を選択します。



重要

Loki Operator と Kafka は、インストールされていた場合、残っているため、個別に削除する必要があります。さらに、オブジェクトストアに保存された残りのデータ、および削除する必要がある永続ボリュームがある場合があります。

第4章 OPENSIFT CONTAINER PLATFORM の NETWORK OBSERVABILITY OPERATOR

Network Observability は、Network Observability eBPF agent によって生成されるネットワークトラフィックフローを収集および強化するためにモニタリングパイプラインをデプロイする OpenShift Operator です。

4.1. 状況の表示

Network Observability Operator は Flow Collector API を提供します。Flow Collector リソースが作成されると、Pod とサービスをデプロイしてネットワークフローを作成して Loki ログストアに保存し、ダッシュボード、メトリクス、およびフローを OpenShift Container Platform Web コンソールに表示します。

手順

1. 次のコマンドを実行して、**FlowCollector** の状態を表示します。

```
$ oc get flowcollector/cluster
```

出力例

```
NAME      AGENT  SAMPLING (EBPF)  DEPLOYMENT MODEL  STATUS
cluster  EBPF   50                DIRECT             Ready
```

2. 次のコマンドを実行して、**netobserv** namespace で実行している Pod のステータスを確認します。

```
$ oc get pods -n netobserv
```

出力例

```
NAME                                READY  STATUS  RESTARTS  AGE
flowlogs-pipeline-56hbp             1/1   Running  0          147m
flowlogs-pipeline-9plvv             1/1   Running  0          147m
flowlogs-pipeline-h5gkb             1/1   Running  0          147m
flowlogs-pipeline-hh6kf             1/1   Running  0          147m
flowlogs-pipeline-w7vv5             1/1   Running  0          147m
netobserv-plugin-cdd7dc6c-j8ggp    1/1   Running  0          147m
```

flowlogs-pipeline Pod はフローを収集し、収集したフローを充実させてから、フローを Loki ストレージに送信します。**netobserv-plugin** Pod は、OpenShift Container Platform コンソール用の視覚化プラグインを作成します。

1. 次のコマンドを入力して、namespace **netobserv-privileged** で実行している Pod のステータスを確認します。

```
$ oc get pods -n netobserv-privileged
```

出力例

```
NAME                                READY  STATUS  RESTARTS  AGE
```

```
netobserv-ebpf-agent-4lpp6 1/1 Running 0 151m
netobserv-ebpf-agent-6gbrk 1/1 Running 0 151m
netobserv-ebpf-agent-klpl9 1/1 Running 0 151m
netobserv-ebpf-agent-vrcnf 1/1 Running 0 151m
netobserv-ebpf-agent-xf5jh 1/1 Running 0 151m
```

netobserv-ebpf-agent Pod は、ノードのネットワークインターフェイスを監視してフローを取得し、それを **flowlogs-pipeline** Pod に送信します。

1. Loki Operator を使用している場合は、次のコマンドを実行して、**openshift-operators-redhat** namespace で実行している Pod のステータスを確認します。

```
$ oc get pods -n openshift-operators-redhat
```

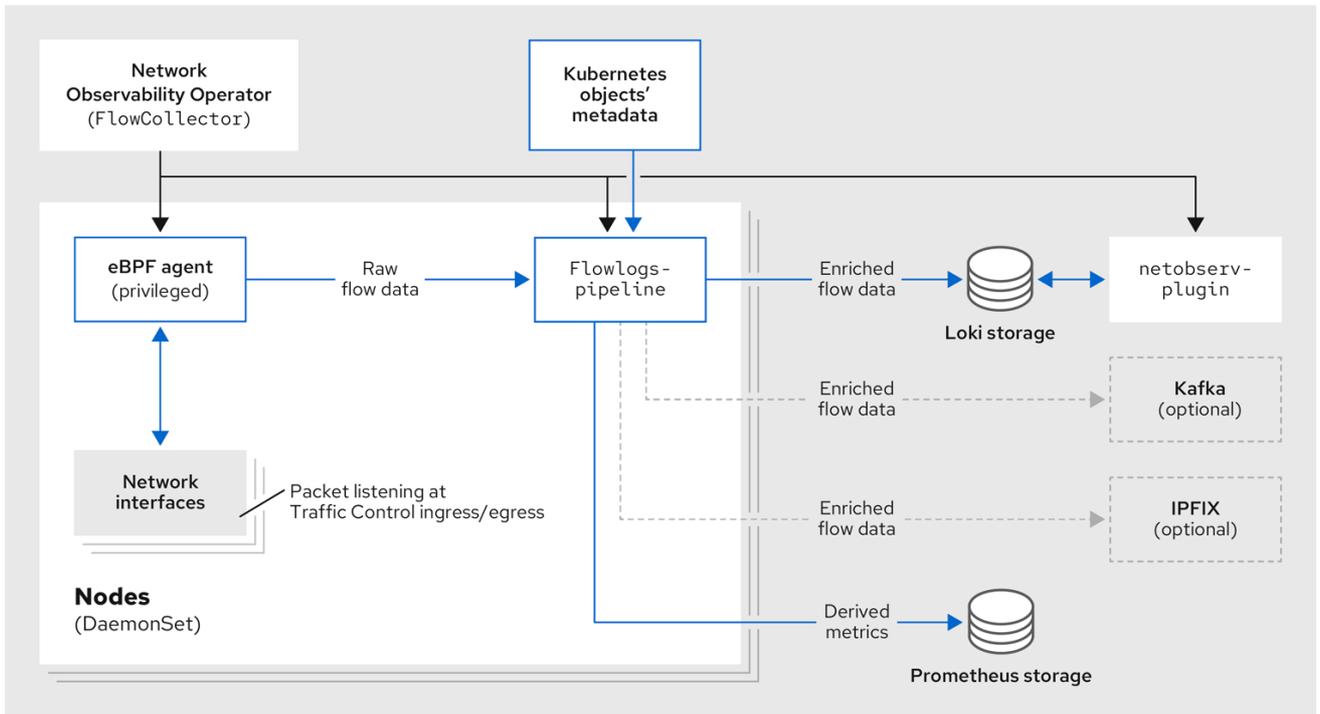
出力例

```
NAME                                READY STATUS RESTARTS AGE
loki-operator-controller-manager-5f6cff4f9d-jq25h 2/2 Running 0 18h
lokistack-compact-0                  1/1 Running 0 18h
lokistack-distributor-654f87c5bc-qhkvh 1/1 Running 0 18h
lokistack-distributor-654f87c5bc-skxgm 1/1 Running 0 18h
lokistack-gateway-796dc6ff7-c54gz 2/2 Running 0 18h
lokistack-index-gateway-0            1/1 Running 0 18h
lokistack-index-gateway-1            1/1 Running 0 18h
lokistack-ingester-0                 1/1 Running 0 18h
lokistack-ingester-1                 1/1 Running 0 18h
lokistack-ingester-2                 1/1 Running 0 18h
lokistack-querier-66747dc666-6vh5x 1/1 Running 0 18h
lokistack-querier-66747dc666-cjr45 1/1 Running 0 18h
lokistack-querier-66747dc666-xh8rq 1/1 Running 0 18h
lokistack-query-frontend-85c6db4fbd-b2xfb 1/1 Running 0 18h
lokistack-query-frontend-85c6db4fbd-jm94f 1/1 Running 0 18h
```

4.2. NETWORK OBSERVABILITY OPERATOR のアーキテクチャー

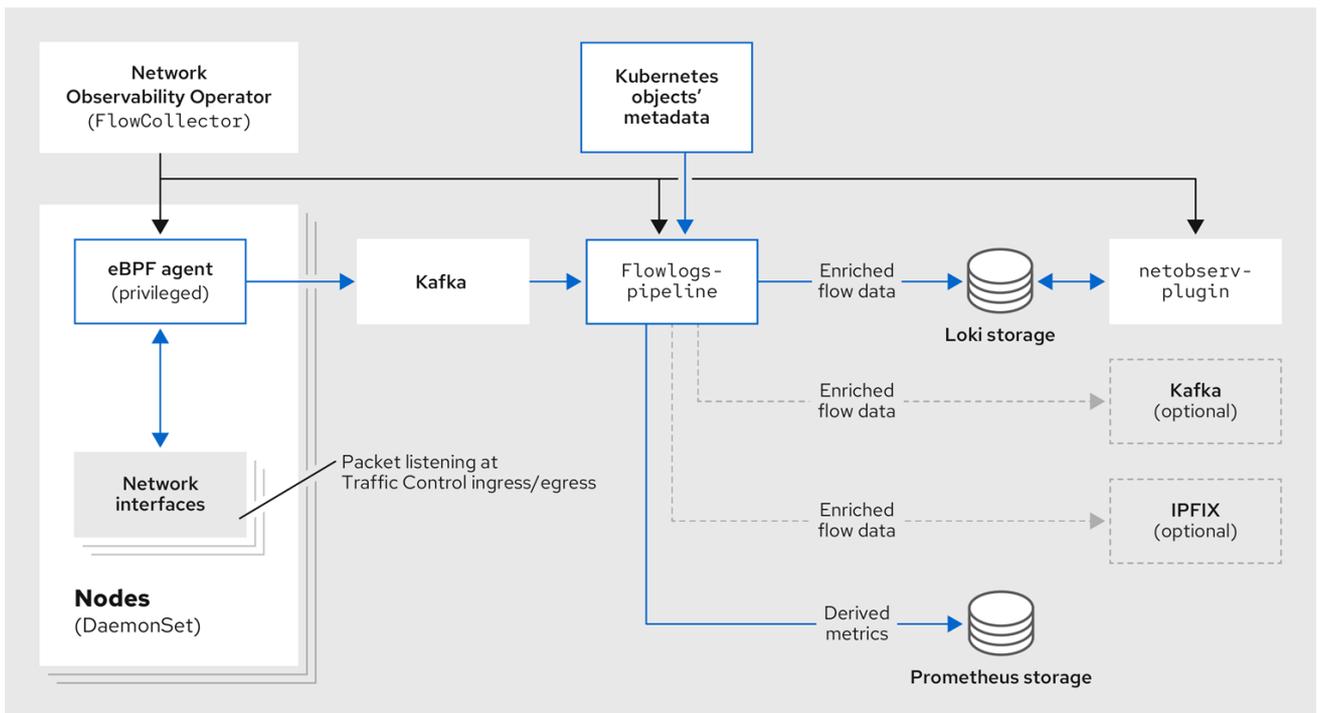
Network Observability Operator は、**FlowCollector** API を提供します。これは、インストール時にインストール化され、**eBPF agent**、**flowlogs-pipeline**、**netobserv-plugin** コンポーネントを調整するように設定されています。**FlowCollector** は、クラスターごとに1つだけサポートされます。

eBPF agent は、各クラスター上で実行され、ネットワークフローを収集するためのいくつかの権限を持っています。**flowlogs-pipeline** はネットワークフローデータを受信し、データに Kubernetes 識別子を追加します。Loki を使用している場合、**flowlogs-pipeline** はフローログデータを Loki に送信して、保存およびインデックス化を行います。**netobserv-plugin** は、動的 OpenShift Container Platform Web コンソールプラグインであり、Loki にクエリーを実行してネットワークフローデータを取得します。クラスター管理者は、Web コンソールでデータを表示できます。



351_OpenShift_0823

次の図に示すように、Kafka オプションを使用している場合、eBPF agent はネットワークフローデータを Kafka に送信し、**flowlogs-pipeline** は Loki に送信する前に Kafka トピックから読み取ります。



351_OpenShift_0823

4.3. NETWORK OBSERVABILITY OPERATOR のステータスと設定の表示

`oc describe` コマンドを使用して、**FlowCollector** のステータスを確認し、詳細を表示できます。

手順

1. 次のコマンドを実行して、Network Observability Operator のステータスと設定を表示します。

```
$ oc describe flowcollector/cluster
```

第5章 NETWORK OBSERVABILITY OPERATOR の設定

Flow Collector API リソースを更新して、Network Observability Operator とそのマネージドコンポーネントを設定できます。Flow Collector は、インストール中に明示的に作成されます。このリソースはクラスター全体で動作するため、単一の **FlowCollector** のみが許可され、**cluster** という名前を付ける必要があります。

5.1. FLOWCOLLECTOR リソースを表示する

OpenShift Container Platform Web コンソールで YAML を直接表示および編集できます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。そこで、**FlowCollector** リソースを変更して Network Observability Operator を設定できます。

以下の例は、OpenShift Container Platform Network Observability Operator のサンプル **FlowCollector** リソースを示しています。

FlowCollector リソースのサンプル

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF 1
    ebpf:
      sampling: 50 2
      logLevel: info
      privileged: false
      resources:
        requests:
          memory: 50Mi
          cpu: 100m
        limits:
          memory: 800Mi
  processor: 3
    logLevel: info
    resources:
      requests:
        memory: 100Mi
        cpu: 100m
      limits:
        memory: 800Mi
  logTypes: Flows
  advanced:
```

```

conversationEndTimeout: 10s
conversationHeartbeatInterval: 30s
loki:
  mode: LokiStack
consolePlugin:
  register: true
  logLevel: info
  portNaming:
    enable: true
    portNames:
      "3100": loki
quickFilters:
- name: Applications
  filter:
    src_namespace!: 'openshift-,netobserv'
    dst_namespace!: 'openshift-,netobserv'
  default: true
- name: Infrastructure
  filter:
    src_namespace: 'openshift-,netobserv'
    dst_namespace: 'openshift-,netobserv'
- name: Pods network
  filter:
    src_kind: 'Pod'
    dst_kind: 'Pod'
  default: true
- name: Services network
  filter:
    dst_kind: 'Service'

```

- 1 エージェント仕様 **spec.agent.type** は **EBPF** でなければなりません。eBPF は、OpenShift Container Platform でサポートされる唯一のオプションです。
- 2 サンプリング仕様 **spec.agent.ebpf.sampling** を設定して、リソースを管理できます。サンプリング値が低いと、大量の計算、メモリー、およびストレージリソースが消費される可能性があります。これは、サンプリング比の値を指定することで軽減できます。値 100 は、100 ごとに1つのフローがサンプリングされることを意味します。0 または 1 の値は、すべてのフローがキャプチャーされることを意味します。値が低いほど、返されるフローが増加し、派生メトリクスの精度が向上します。デフォルトでは、eBPF サンプリングは値 50 に設定されているため、50 ごとに1つのフローがサンプリングされます。より多くのサンプルフローは、より多くのストレージが必要になることにも注意してください。デフォルト値から始めて経験的に調整し、クラスターが管理できる設定を決定することを推奨します。
- 3 プロセッサ仕様 **spec.processor** を設定すると、会話追跡を有効にできます。有効にすると、Web コンソールで会話イベントをクエリーできるようになります。**spec.processor.logTypes** の値は **Flows** です。**spec.processor.advanced** の値は、**Conversations**、**EndedConversations**、または **ALL** です。ストレージ要件は **All** で最も高く、**EndedConversations** で最も低くなります。
- 4 Loki 仕様である **spec.loki** は、Loki クライアントを指定します。デフォルト値は、Loki Operator のインストールセクションに記載されている Loki インストールパスと一致します。Loki の別のインストール方法を使用した場合は、インストールに適切なクライアント情報を指定します。
- 5 **LokiStack** モードは、いくつかの設定 (**querierUrl**、**ingesterUrl**、**statusUrl**、**tenantID**、および対応する TLS 設定) を自動的に設定します。クラスターロールとクラスターロールバインディングが、Loki へのログの読み取りと書き込みのために作成されます。**authToken** は **Forward** に設定さ

れます。**Manual** モードを使用すると、これらを手動で設定できます。

- 6 **spec.quickFilters** 仕様は、Web コンソールに表示されるフィルターを定義します。**Application** フィルターキー、**src_namespace** および **dst_namespace** は否定 (!) されているため、**Application** フィルターは、**openshift-** または **netobserv** namespace から発信されていない、または宛先がないすべてのトラフィックを表示します。詳細は、以下のクイックフィルターの設定を参照してください。

関連情報

会話追跡の詳細は、[Working with conversations](#) を参照してください。

5.2. KAFKA を使用した FLOW COLLECTOR リソースの設定

Kafka を高スループットかつ低遅延のデータフィードのために使用するように、**FlowCollector** リソースを設定できます。Kafka インスタンスを実行する必要があり、そのインスタンスで OpenShift Container Platform Network Observability 専用の Kafka トピックを作成する必要があります。詳細は、[AMQ Streams を使用した Kafka ドキュメント](#) を参照してください。

前提条件

- Kafka がインストールされている。Red Hat は、AMQ Streams Operator を使用する Kafka をサポートします。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. Network Observability Operator の **Provided APIs** という見出しの下で、**Flow Collector** を選択します。
3. クラスターを選択し、**YAML** タブをクリックします。
4. 次のサンプル YAML に示すように、Kafka を使用するように OpenShift Container Platform Network Observability Operator の **FlowCollector** リソースを変更します。

FlowCollector リソースの Kafka 設定のサンプル

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  deploymentModel: Kafka
  kafka:
    address: "kafka-cluster-kafka-bootstrap.netobserv"
    topic: network-flows
    tls:
      enable: false
```

- 1 Kafka デプロイメントモデルを有効にするには、**spec.deploymentModel** を **Direct** ではなく **Kafka** に設定します。

- 2 **spec.kafka.address** は、Kafka ブートストラップサーバーのアドレスを参照します。ポート 9093 で TLS を使用するため、**kafka-cluster-kafka-bootstrap.netobserv:9093** など、必要に応じて
- 3 **spec.kafka.topic** は、Kafka で作成されたトピックの名前と一致する必要があります。
- 4 **spec.kafka.tls** を使用して、Kafka との間のすべての通信を TLS または mTLS で暗号化できます。有効にした場合、Kafka CA 証明書は、**flowlogs-pipeline** プロセッサコンポーネントがデプロイされている namespace (デフォルト: **netobserv**) と eBPF エージェントがデプロイされている namespace (デフォルト: **netobserv-privileged**) の両方で ConfigMap または Secret として使用できる必要があります。**spec.kafka.tls.caCert** で参照する必要があります。mTLS を使用する場合、クライアントシークレットはこれらの namespace でも利用でき (たとえば、AMQ Streams User Operator を使用して生成できます)、**spec.kafka.tls.userCert** で参照される必要があります。

5.3. 強化されたネットワークフローデータをエクスポートする

ネットワークフローを Kafka、IPFIX、またはその両方に同時に送信できます。Splunk、Elasticsearch、Fluentd などをはじめとする、Kafka または IPFIX 入力をサポートするプロセッサまたはストレージは、補完されたネットワークフローデータを使用できます。

前提条件

- Network Observability の **flowlogs-pipeline** Pod から Kafka または IPFIX コレクターエンドポイントを使用できる。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. **FlowCollector** を編集して、**spec.exporters** を次のように設定します。

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  exporters:
    - type: Kafka 1
      kafka:
        address: "kafka-cluster-kafka-bootstrap.netobserv"
        topic: netobserv-flows-export 2
        tls:
          enable: false 3
    - type: IPFIX 4
      ipfix:
        targetHost: "ipfix-collector.ipfix.svc.cluster.local"
        targetPort: 4739
        transport: tcp or udp 5

```

- 2 Network Observability Operator は、すべてのフローを設定された Kafka トピックにエクスポートします。
 - 3 Kafka との間のすべての通信を SSL/TLS または mTLS で暗号化できます。有効にした場合、Kafka CA 証明書は、**flowlogs-pipeline** プロセッサコンポーネントがデプロイされている namespace (デフォルト: netobserv) で、ConfigMap または Secret として使用する必要があります。これは **spec.exporters.tls.caCert** で参照する必要があります。mTLS を使用する場合、クライアントシークレットはこれらの namespace でも利用可能であり (たとえば、AMQ Streams User Operator を使用して生成できます)、**spec.exporters.tls.userCert** で参照される必要があります。
 - 1 4 Kafka にフローをエクスポートする代わりに、またはそれを併せて、フローを IPFIX にエクスポートできます。
 - 5 オプションでトランスポートを指定できます。デフォルト値は **tcp** ですが、**udp** を指定することもできます。
5. 設定後、ネットワークフローデータを JSON 形式で利用可能な出力に送信できます。詳細は、ネットワークフロー形式のリファレンスを参照してください。

関連情報

フロー形式の指定の詳細は、[ネットワークフロー形式リファレンス](#) を参照してください。

5.4. FLOW COLLECTOR リソースの更新

OpenShift Container Platform Web コンソールで YAML を編集する代わりに、**flowcollector** カスタムリソース (CR) にパッチを適用することで、eBPF サンプルングなどの仕様を設定できます。

手順

1. 次のコマンドを実行して、**flowcollector** CR にパッチを適用し、**spec.agent.ebpf.sampling** 値を更新します。

```
$ oc patch flowcollector cluster --type=json -p [{"op": "replace", "path":
"/spec/agent/ebpf/sampling", "value": <new value>}] -n netobserv"
```

5.5. クイックフィルターの設定

FlowCollector リソースでフィルターを変更できます。値を二重引用符で囲むと、完全一致が可能になります。それ以外の場合、テキスト値には部分一致が使用されます。キーの最後にあるバング (!) 文字は、否定を意味します。YAML の変更に関する詳細なコンテキストは、サンプルの **FlowCollector** リソースを参照してください。



注記

フィルターマッチングタイプ "all of" または "any of" は、ユーザーがクエリーオプションから変更できる UI 設定です。これは、このリソース設定の一部ではありません。

使用可能なすべてのフィルターキーのリストを次に示します。

表5.1 フィルターキー

Universe*	ソース	送信先	説明
namespace	src_namespace	dst_namespace	特定の namespace に関連するトラフィックをフィルタリングします。
name	src_name	dst_name	特定の Pod、サービス、またはノード (ホストネットワークトラフィックの場合) など、特定のリーフリソース名に関連するトラフィックをフィルター処理します。
kind	src_kind	dst_kind	特定のリソースの種類に関連するトラフィックをフィルタリングします。リソースの種類には、リーフリソース (Pod、Service、または Node)、または所有者リソース (Deployment および StatefulSet) が含まれます。
owner_name	src_owner_name	dst_owner_name	特定のリソース所有者に関連するトラフィックをフィルタリングします。つまり、ワークロードまたは Pod のセットです。たとえば、Deployment 名、StatefulSet 名などです。
resource	src_resource	dst_resource	一意に識別する正規名で示される特定のリソースに関連するトラフィックをフィルタリングします。正規の表記法は、namespace の種類の場合は kind.namespace.name 、ノードの場合は node.name です。たとえば、 Deployment.my-namespace.my-web-server です。
address	src_address	dst_address	IP アドレスに関連するトラフィックをフィルタリングします。IPv4 と IPv6 がサポートされています。CIDR 範囲もサポートされています。
mac	src_mac	dst_mac	MAC アドレスに関連するトラフィックをフィルタリングします。
port	src_port	dst_port	特定のポートに関連するトラフィックをフィルタリングします。
host_addresses	src_host_address	dst_host_address	Pod が実行しているホスト IP アドレスに関連するトラフィックをフィルタリングします。
protocol	該当なし	該当なし	TCP や UDP などのプロトコルに関連するトラフィックをフィルタリングします。

- ソースまたは宛先のいずれかのユニバーサルキーフィルター。たとえば、フィルタリング **name: 'my-pod'** は、使用される一致タイプ (Match all または Match any) に関係なく、**my-pod** からのすべてのトラフィックと **my-pod** へのすべてのトラフィックを意味します。

5.6. SR-IOV インターフェイストラフィックの監視の設定

Single Root I/O Virtualization (SR-IOV) デバイスを使用してクラスターからトラフィックを収集するに

は、**FlowCollector spec.agent.ebpf.privileged** フィールドを **true** に設定する必要があります。次に、eBPF agent は、デフォルトで監視されるホストネットワーク namespace に加え、他のネットワーク namespace も監視します。仮想機能 (VF) インターフェイスを持つ Pod が作成されると、新しいネットワーク namespace が作成されます。**SRIOVNetwork** ポリシーの **IPAM** 設定を指定すると、VF インターフェイスがホストネットワーク namespace から Pod ネットワーク namespace に移行されます。

前提条件

- SR-IOV デバイスを使用して OpenShift Container Platform クラスタにアクセスできる。
- **SRIOVNetwork** カスタムリソース (CR) の **spec.ipam** 設定は、インターフェイスのリストにある範囲または他のプラグインからの IP アドレスを使用して設定する必要があります。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. **FlowCollector** カスタムリソースを設定します。設定例は次のとおりです。

SR-IOV モニタリング用に FlowCollector を設定する

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
agent:
  type: eBPF
  ebpf:
    privileged: true ①
```

- ① SR-IOV モニタリングを有効にするには、**spec.agent.ebpf.privileged** フィールドの値を **true** に設定する必要があります。

関連情報

SriovNetwork カスタムリソースの作成の詳細は、[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](#) を参照してください。

5.7. リソース管理およびパフォーマンスに関する考慮事項

ネットワーク監視に必要なリソースの量は、クラスタのサイズと、クラスタが可観測データを取り込んで保存するための要件によって異なります。リソースを管理し、クラスタのパフォーマンス基準を設定するには、次の設定を設定することを検討してください。これらの設定を設定すると、最適なセットアップと可観測性のニーズを満たす可能性があります。

次の設定は、最初からリソースとパフォーマンスを管理するのに役立ちます。

eBPF サンプリング

サンプリング仕様 **spec.agent.ebpf.sampling** を設定して、リソースを管理できます。サンプリング値が低いと、大量の計算、メモリー、およびストレージリソースが消費される可能性があります。これは、サンプリング比の値を指定することで軽減できます。値 **100** は、100 ごとに1つのフローがサンプリングされることを意味します。**0** または **1** の値は、すべてのフローがキャプチャーされることを意味します。値が小さいほど、返されるフローが増加し、派生メトリクスの精度が向上します。デフォルトでは、eBPF サンプリングは値 50 に設定されているため、50 ごとに1つのフローがサンプリングされます。より多くのサンプルフローは、より多くのストレージが必要になることにも注意してください。クラスターがどの設定を管理できるかを判断するには、デフォルト値から始めて実験的に調整することを検討してください。

インターフェイスの制限または除外

spec.agent.ebpf.interfaces および **spec.agent.ebpf.excludeInterfaces** の値を設定して、観測されるトラフィック全体を削減します。デフォルトでは、エージェントは、**excludeInterfaces** および **lo** (ローカルインターフェイス) にリストされているインターフェイスを除く、システム内のすべてのインターフェイスを取得します。インターフェイス名は、使用される Container Network Interface (CNI) によって異なる場合があることに注意してください。

Network Observability をしばらく実行した後、次の設定を使用してパフォーマンスを微調整できます。

リソース要件および制限

spec.agent.ebpf.resources および **spec.processor.resources** 仕様を使用して、リソース要件と制限をクラスターで予想される負荷とメモリー使用量に適応させます。多くの中規模のクラスターには、デフォルトの制限の 800MB で十分な場合があります。

キャッシュの最大フロータイムアウト

eBPF エージェントの **spec.agent.ebpf.cacheMaxFlows** および **spec.agent.ebpf.cacheActiveTimeout** 仕様を使用して、エージェントによってフローが報告される頻度を制御します。値が大きいほど、エージェントで生成されるトラフィックが少なくなり、これは CPU 負荷の低下と相関します。ただし、値を大きくするとメモリー消費量がわずかに増加し、フロー収集でより多くの遅延が発生する可能性があります。

5.7.1. リソースの留意事項

次の表は、特定のワークロードサイズのクラスターのリソースに関する考慮事項の例を示しています。



重要

表に概要を示した例は、特定のワークロードに合わせて調整されたシナリオを示しています。各例は、ワークロードのニーズに合わせて調整を行うためのベースラインとしてのみ考慮してください。

表5.2 リソースの推奨事項

	極小規模 (10 ノード)	小規模 (25 ノード)	中規模 (65 ノード) [2]	大規模 (120 ノード) [2]
ワーカーノードの vCPU とメモリー	4 vCPU 16 GiB メモリー [1]	16 vCPU 64 GiB メモリー [1]	16 vCPU 64 GiB メモリー [1]	16 vCPU 64 GiB メモリー [1]
LokiStack サイズ	1x.extra-small	1x.small	1x.small	1x.medium

	極小規模 (10 ノード)	小規模 (25 ノード)	中規模 (65 ノード) [2]	大規模 (120 ノード) [2]
Network Observability コントローラーのメモリー制限	400 Mi (デフォルト)	400 Mi (デフォルト)	400 Mi (デフォルト)	400 Mi (デフォルト)
eBPF サンプリングレート	50 (デフォルト)	50 (デフォルト)	50 (デフォルト)	50 (デフォルト)
eBPF メモリー制限	800 Mi (デフォルト)	800 Mi (デフォルト)	800 Mi (デフォルト)	1600 Mi
FLP メモリー制限	800 Mi (デフォルト)	800 Mi (デフォルト)	800 Mi (デフォルト)	800 Mi (デフォルト)
FLP Kafka パーティション	該当なし	48	48	48
Kafka コンシューマーレプリカ	該当なし	24	24	24
Kafka ブローカー	該当なし	3 (デフォルト)	3 (デフォルト)	3 (デフォルト)

1. AWS M6i インスタンスでテスト済み。
2. このワーカーとそのコントローラーに加えて、3つのインフラノード (サイズ **M6i.12xlarge**) と1つのワークロードノード (サイズ **M6i.8xlarge**) がテストされました。

第6章 ネットワークポリシー

admin ロールを持つユーザーは、**netobserv** namespace のネットワークポリシーを作成して、Network Observability Operator への受信アクセスを保護できます。

6.1. NETWORK OBSERVABILITY のためのネットワークポリシーの作成

netobserv namespace への ingress トラフィックを保護するために、ネットワークポリシーを作成する必要がある場合があります。Web コンソールでは、フォームビューを使用してネットワークポリシーを作成できます。

手順

1. **Networking** → **NetworkPolicies** に移動します。
2. **Project** ドロップダウンメニューから **netobserv** プロジェクトを選択します。
3. ポリシーに名前を付けます。この例では、ポリシー名は **allowed-ingress** です。
4. **Add ingress rule** を 3 回クリックして、3 つのイングレスルールを作成します。
5. フォームで以下を指定します。
 - a. 最初の **Ingress rule** に対して以下の仕様を作成します。
 - i. **Add allowed source** ドロップダウンメニューから、**Allow pods from the same namespace** を選択します。
 - b. 2 番目の **Ingress rule** に対して次の仕様を作成します。
 - i. **Add allowed source** ドロップダウンメニューから、**Allow pods from inside the cluster** を選択します。
 - ii. **+ Add namespace selector** をクリックします。
 - iii. ラベル **kubernetes.io/metadata.name** とセレクター **openshift-console** を追加します。
 - c. 3 番目の **Ingress rule** に対して次の仕様を作成します。
 - i. **Add allowed source** ドロップダウンメニューから、**Allow pods from inside the cluster** を選択します。
 - ii. **+ Add namespace selector** をクリックします。
 - iii. ラベル **kubernetes.io/metadata.name** とセレクター **openshift-monitoring** を追加します。

検証

1. **Observe** → **Network Traffic** に移動します。
2. **Traffic Flows** タブまたは任意のタブを表示して、データが表示されていることを確認します。
3. **Observe** → **Dashboards** に移動します。NetObserv/Health の選択で、フローが取り込まれて Loki に送信されていることを確認します (最初のグラフに示されています)。

6.2. ネットワークポリシーの例

以下は、**netobserv** namespace の **NetworkPolicy** オブジェクトの例にアノテーションを付けています。

サンプルネットワークポリシー

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-ingress
  namespace: netobserv
spec:
  podSelector: {} ①
  ingress:
    - from:
      - podSelector: {} ②
        namespaceSelector: ③
          matchLabels:
            kubernetes.io/metadata.name: openshift-console
      - podSelector: {}
        namespaceSelector:
          matchLabels:
            kubernetes.io/metadata.name: openshift-monitoring
  policyTypes:
    - Ingress
status: {}
```

- ① 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。このドキュメントでは、Netobservability Operator がインストールされているプロジェクト、つまり **netobserv** プロジェクトになります。
- ② 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。デフォルトでは、セレクターは **NetworkPolicy** と同じ namespace の Pod と一致します。
- ③ **namespaceSelector** が指定されている場合、セレクターは指定された namespace 内の Pod と一致します。

関連情報

[CLI を使用したネットワークポリシーの作成](#)

第7章 ネットワークトラフィックの監視

管理者は、OpenShift Container Platform コンソールでネットワークトラフィックを観察して、詳細なトラブルシューティングと分析を行うことができます。この機能は、トラフィックフローのさまざまなグラフィカル表現から洞察を得るのに役立ちます。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ビューがいくつかあります。

7.1. OVERVIEW ビューからのネットワークトラフィックの監視

Overview ビューには、クラスター上のネットワークトラフィックフローの集約された全体的なメトリクスが表示されます。管理者は、使用可能な表示オプションを使用して統計を監視できます。

7.1.1. 概要ビューの操作

管理者は、**Overview** ビューに移動して、フローレートの統計をグラフィカルに表示できます。

手順

1. **Observe** → **Network Traffic** に移動します。
2. ネットワークトラフィック ページで、**Overview** タブをクリックします。

メニューアイコンをクリックすると、各流量データの範囲を設定できます。

7.1.2. 概要ビューの詳細オプションの設定

詳細オプションを使用して、グラフィカルビューをカスタマイズできます。詳細オプションにアクセスするには、**Show advanced options** をクリックします。**Display options** ドロップダウンメニューを使用して、グラフの詳細を設定できます。利用可能なオプションは次のとおりです。

- **Scope**: ネットワークトラフィックが流れるコンポーネントを表示する場合に選択します。スコープは、**Node**、**Namespace**、**Owner**、**Zones**、**Cluster**、または **Resource** に設定できます。**Owner** はリソースの集合体です。**Resource** は、ホストネットワークトラフィックの場合は Pod、サービス、ノード、または不明な IP アドレスです。デフォルト値は **Namespace** です。
- **Truncate labels**: ドロップダウンリストから必要なラベルの幅を選択します。デフォルト値は **M** です。

7.1.2.1. パネルとディスプレイの管理

表示する必要なパネルを選択したり、並べ替えたり、特定のパネルに焦点を当てたりすることができます。パネルを追加または削除するには、**Manage panels** をクリックします。

デフォルトでは、次のパネルが表示されます。

- 上位 X の平均バイトレート
- 上位 X のバイトレートと合計の積み上げ値

他のパネルは **Manage panels** で追加できます。

- 上位 X の平均パケットレート
- 上位 X のパケットレートと合計の積み上げ値

Query options を使用すると、**Top 5**、**Top 10**、または **Top 15** のレートを表示するかどうかを選択できます。

7.1.3. パケットドロップの追跡

Overview ビューで、パケットロスが発生したネットワークフローレコードのグラフィック表示を設定できます。eBPF トレースポイントフックを採用すると、TCP、UDP、SCTP、ICMPv4、ICMPv6 プロトコルのパケットドロップに関する貴重な知見を得ることができ、その結果、以下のアクションにつながる可能性があります。

- **識別:** パケットドロップが発生している正確な場所とネットワークパスを特定します。ドロップが発生しやすい特定のデバイス、インターフェイス、またはルートがあるか判断します。
- **根本原因分析:** eBPF プログラムによって収集されたデータを調査し、パケットドロップの原因を把握します。たとえば、輻輳、バッファの問題、特定のネットワークイベントなどの原因です。
- **パフォーマンスの最適化:** パケットドロップをより明確に把握し、バッファサイズの調整、ルーティングパスの再設定、Quality of Service (QoS) 対策の実装など、ネットワークパフォーマンスを最適化するための手順を実行できます。

パケットドロップの追跡が有効になっている場合、デフォルトで **Overview** に次のパネルが表示されません。

- 上位 X のパケットドロップの状態と合計の積み上げ値
- 上位 X のパケットドロップの原因と合計の積み上げ値
- 上位 X の平均パケットドロップレート
- 上位 X のパケットドロップレートと合計の積み上げ値

他のパケットドロップパネルは **Manage panels** で追加できます。

- 上位 X の平均ドロップバイトレート
- 上位 X の平均ドロップバイトレートと合計の積み上げ値

7.1.3.1. パケットドロップの種類

Network Observability では、ホストドロップと OVS ドロップという 2 種類のパケットドロップが検出されます。ホストドロップには **SKB_DROP** という接頭辞が付き、OVS ドロップには **OVS_DROP** という接頭辞が付きます。ドロップされたフローは、各ドロップタイプの説明へのリンクとともに、**Traffic flows** テーブルのサイドパネルに表示されます。ホストドロップの理由の例は次のとおりです。

- **SKB_DROP_REASON_NO_SOCKET:** ソケットが検出されないため、パケットがドロップされました。
- **SKB_DROP_REASON_TCP_CSUM:** TCP チェックサムエラーによりパケットがドロップされました。

OVS ドロップの理由の例は次のとおりです。

- **OVS_DROP_LAST_ACTION:** 暗黙的なドロップアクション (設定されたネットワークポリシーなど) によりドロップされた OVS パケット。

- **OVS_DROP_IP_TTL**: IP TTL の期限切れにより OVS パケットがドロップされました。

パケットドロップの追跡を有効化および使用方法の詳細は、このセクションの **関連情報** を参照してください。

関連情報

- [パケットドロップの使用](#)
- [Network Observability メトリクス](#)

7.1.4. DNS 追跡

Overview ビューで、ネットワークフローの Domain Name System (DNS) 追跡のグラフィカル表示を設定できます。拡張 Berkeley Packet Filter (eBPF) トレースポイントフックを使用する DNS 追跡は、さまざまな目的に使用できます。

- **ネットワーク監視**: DNS クエリーと応答に関する知見を得ることで、ネットワーク管理者は異常パターン、潜在的なボトルネック、またはパフォーマンスの問題を特定できます。
- **セキュリティ分析**: マルウェアによって使用されるドメイン名生成アルゴリズム (DGA) などの不審な DNS アクティビティを検出したり、セキュリティを侵害する可能性のある不正な DNS 解決を特定したりします。
- **トラブルシューティング**: DNS 解決手順を追跡し、遅延を追跡し、設定ミスを特定することにより、DNS 関連の問題をデバッグします。

デフォルトでは、DNS 追跡が有効になっている場合、**Overview** に、次の空でないメトリクスがドーナツグラフまたは折れ線グラフで表示されます。

- 上位 X の DNS レスポンスコード
- 上位 X の平均 DNS 遅延と合計
- 上位 X の 90 パーセンタイルの DNS 遅延

他の DNS 追跡パネルは **Manage panels** で追加できます。

- 下位 X の最小 DNS 遅延
- 上位 X の最大 DNS 遅延
- 上位 X の 99 パーセンタイルの DNS 遅延

この機能は、IPv4 および IPv6 の UDP および TCP プロトコルでサポートされています。

このビューの有効化と使用の詳細は、このセクションの **関連情報** を参照してください。

関連情報

- [DNS 追跡の使用](#)
- [Network Observability メトリクス](#)

7.1.5. ラウンドトリップタイム

TCP の平滑化されたラウンドトリップタイム (sRTT) を使用して、ネットワークフローの遅延を分析できます。**fentry/tcp_rcv_established** eBPF フックポイントから取得した RTT を使用して TCP ソケットから sRTT を読み取ると、次のことに役立てることができます。

- ネットワーク監視: TCP の遅延に関する知見を得ることで、ネットワーク管理者は、異常なパターン、潜在的なボトルネック、またはパフォーマンスの問題を特定できます。
- トラブルシューティング: 遅延を追跡し、設定ミスを特定することにより、TCP 関連の問題をデバッグします。

デフォルトでは、RTT が有効になっている場合、**Overview** に次の TCP RTT メトリクスが表示されます。

- 上位 X の 90 パーセンタイルの TCP ラウンドトリップタイムと合計
- 上位 X の平均 TCP ラウンドトリップタイムと合計
- 下位 X の最小 TCP ラウンドトリップタイムと合計

他の RTT パネルは **Manage panels** で追加できます。

- 上位 X の最大 TCP ラウンドトリップタイムと合計
- 上位 X の 99 パーセンタイルの TCP ラウンドトリップタイムと合計

このビューの有効化と使用の詳細は、このセクションの **関連情報** を参照してください。

関連情報

- [RTT トレーシングの使用](#)

7.1.6. eBPF フローのルールフィルター

ルールベースのフィルタリングを使用して、eBPF フローテーブルにキャッシュされるパケットの量を制御できます。たとえば、ポート 100 から送信されるパケットのみを記録するようにフィルターを指定できます。その場合、フィルターに一致するパケットのみがキャッシュされ、残りのパケットはキャッシュされません。

7.1.6.1. Ingress および Egress トラフィックのフィルタリング

CIDR 表記は、ベース IP アドレスとプレフィックス長を組み合わせることにより、IP アドレス範囲を効率的に表すものです。Ingress と Egress トラフィックの両方において、送信元 IP アドレスが、CIDR 表記で設定されたフィルタールールを照合するために最初に使用されます。一致するものがあれば、フィルタリングが続行されます。一致するものがない場合、宛先 IP を使用して、CIDR 表記で設定されたフィルタールールを照合します。

送信元 IP または宛先 IP の CIDR のいずれかを照合した後、**peerIP** を使用して特定のエンドポイントを特定し、パケットの宛先 IP アドレスを識別できます。定められたアクションに基づいて、フローデータが eBPF フローテーブルにキャッシュされるかされないかが決まります。

7.1.6.2. ダッシュボードとメトリクスの統合

このオプションを有効にすると、**eBPF agent statistics** の **Netobserv/Health** ダッシュボードに、**Filtered flows rate** ビューが表示されるようになります。さらに、**Observe → Metrics** で、**netobserv_agent_filtered_flows_total** をクエリーし

て、**FlowFilterAcceptCounter**、**FlowFilterNoMatchCounter**、または **FlowFilterRejectCounter** の理由を含むメトリクスを監視できます。

7.1.6.3. フローフィルターの設定パラメーター

フローフィルタールールは、必須のパラメーターと任意のパラメーターで構成されます。

表7.1 必須設定パラメーター

パラメーター	説明
enable	eBPF フローのフィルタリング機能を有効にするには、 enable を true に設定します。
cidr	フローフィルタールールの IP アドレスと CIDR マスクを指定します。IPv4 と IPv6 の両方のアドレス形式をサポートしています。すべての IP と照合する場合、IPv4 の場合は 0.0.0.0/0 、IPv6 の場合は ::/0 を使用できます。
action	フローフィルタールールに対して実行されるアクションを示します。可能な値は Accept または Reject です。 <ul style="list-style-type: none"> ● Accept アクションに一致するルールの場合、フローデータが eBPF テーブルにキャッシュされ、グローバルメトリクス FlowFilterAcceptCounter で更新されます。 ● Reject アクションに一致するルールの場合、フローデータがドロップされ、eBPF テーブルにキャッシュされません。フローデータは、グローバルメトリクス FlowFilterRejectCounter を使用して更新されます。 ● ルールが一致しない場合、フローは eBPF テーブルにキャッシュされ、グローバルメトリクス FlowFilterNoMatchCounter で更新されます。

表7.2 オプションの設定パラメーター

パラメーター	説明
direction	フローフィルタールールの方向を定義します。可能な値は Ingress または Egress です。
protocol	フローフィルタールールのプロトコルを定義します。可能な値は、 TCP 、 UDP 、 SCTP 、 ICMP 、および ICMPv6 です。
ports	フローのフィルタリングに使用するポートを定義します。送信元ポートまたは宛先ポートのどちらにも使用できます。単一のポートをフィルタリングするには、単一のポートを整数値として設定します。たとえば、 ports: 80 です。ポートの範囲をフィルタリングするには、文字列形式の "開始 - 終了" 範囲を使用します。たとえば、 ports: "80-100" です。

パラメーター	説明
sourcePorts	フローのフィルタリングに使用する送信元ポートを定義します。単一のポートをフィルタリングするには、単一のポートを整数値として設定します (例: sourcePorts: 80)。ポートの範囲をフィルタリングするには、文字列形式の "開始 - 終了" 範囲を使用します (例: sourcePorts: "80-100")。
destPorts	destPorts は、フローのフィルタリングに使用する宛先ポートを定義します。単一のポートをフィルタリングするには、単一のポートを整数値として設定します (例: destPorts: 80)。ポートの範囲をフィルタリングするには、文字列形式の "開始 - 終了" 範囲を使用します (例: destPorts: "80-100")。
icmpType	フローのフィルタリングに使用する ICMP タイプを定義します。
icmpCode	フローのフィルタリングに使用する ICMP コードを定義します。
peerIP	フローのフィルタリングに使用する IP アドレスを定義します (例: 10.10.10.10)。

関連情報

- [ルールによる eBPF フローデータのフィルタリング](#)
- [Network Observability メトリクス](#)
- [健全性ダッシュボード](#)

7.2. TRAFFIC FLOWS ビューからのネットワークトラフィックの監視

Traffic flows ビューには、ネットワークフローのデータとトラフィックの量がテーブルに表示されます。管理者は、トラフィックフローテーブルを使用して、アプリケーション全体のトラフィック量を監視できます。

7.2.1. Traffic flows ビューの操作

管理者は、Traffic flows テーブルに移動して、ネットワークフロー情報を確認できます。

手順

1. **Observe** → **Network Traffic** に移動します。
2. **Network Traffic** ページで、**Traffic flows** タブをクリックします。

各行をクリックして、対応するフロー情報を取得できます。

7.2.2. Traffic flows ビューの詳細オプションの設定

Show advanced options を使用して、ビューをカスタマイズおよびエクスポートできます。**Display options** ドロップダウンメニューを使用して、行サイズを設定できます。デフォルト値は **Normal** です。

7.2.2.1. 列の管理

表示する必要のある列を選択し、並べ替えることができます。列を管理するには、**Manage columns** をクリックします。

7.2.2.2. トラフィックフローデータのエクスポート

Traffic flows ビューからデータをエクスポートできます。

手順

1. **Export data** をクリックします。
2. ポップアップウィンドウで、**Export all data** チェックボックスを選択してすべてのデータをエクスポートし、チェックボックスをオフにしてエクスポートする必要のあるフィールドを選択できます。
3. **Export** をクリックします。

7.2.3. 会話追跡の使用

管理者は、同じ会話の一部であるネットワークフローをグループ化できます。会話は、IP アドレス、ポート、プロトコルによって識別されるピアのグループとして定義され、その結果、一意の **Conversation ID** が得られます。Web コンソールで対話イベントをクエリーできます。これらのイベントは、Web コンソールでは次のように表示されます。

- **Conversation start**: このイベントは、接続が開始されているか、TCP フラグがインターセプトされたときに発生します。
- **Conversation tick**: このイベントは、接続がアクティブである間、**FlowCollector spec.processor.conversationHeartbeatInterval** パラメーターで定義された指定間隔ごとに発生します。
- **Conversation end**: このイベントは、**FlowCollector spec.processor.conversationEndTimeout** パラメーターに達するか、TCP フラグがインターセプトされたときに発生します。
- **Flow**: これは、指定された間隔内に発生するネットワークトラフィックフローです。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. **spec.processor.logTypes**、**conversationEndTimeout**、および **conversationHeartbeatInterval** パラメーターが観察のニーズに応じて設定されるように、**FlowCollector** カスタムリソースを設定します。設定例は次のとおりです。

会話追跡用に FlowCollector を設定する

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
```

```

metadata:
  name: cluster
spec:
  processor:
    logTypes: Flows
  advanced:
    conversationEndTimeout: 10s
    conversationHeartbeatInterval: 30s

```

- 1 **logTypes** を **Flows** に設定すると、**Flow** イベントのみがエクスポートされます。値を **All** に設定すると、会話イベントとフローイベントの両方がエクスポートされ、**Network Traffic** ページに表示されます。会話イベントのみに焦点を当てるには、**Conversations** を指定します。これを指定すると、**Conversation start**、**Conversation tick**、および **Conversation end** イベントがエクスポートされます。**EndedConversations** を指定すると、**Conversation end** イベントのみがエクスポートされます。ストレージ要件は **All** で最も高く、**EndedConversations** で最も低くなります。
- 2 **Conversation end** イベントは、**conversationEndTimeout** に達するか、TCP フラグがインターセプトされた時点を表します。
- 3 **Conversation tick** イベントは、ネットワーク接続がアクティブである間の、**FlowCollector** の **conversationHeartbeatInterval** パラメーターで定義された各指定間隔を表します。



注記

logType オプションを更新しても、以前の選択によるフローはコンソールプラグインから消去されません。たとえば、午前10時まで **logType** を **Conversations** に設定し、その後 **EndedConversations** に移行すると、コンソールプラグインは、午前10時まではすべての会話イベントを表示し、午前10時以降は終了した会話のみを表示します。

5. **Traffic flows** タブの **Network Traffic** ページを更新します。**Event/Type** と **Conversation Id** という2つの新しい列があることに注意してください。クエリーオプションとして **Flow** が選択されている場合、すべての **Event/Type** フィールドは **Flow** になります。
6. **Query Options** を選択し、**Log Type** として **Conversation** を選択します。**Event/Type** は、必要なすべての会話イベントを表示するようになりました。
7. 次に、特定の会話 ID でフィルタリングするか、サイドパネルから **Conversation** と **Flow** ログタイプのオプションを切り替えることができます。

7.2.4. パケットドロップの使用

パケットロス、ネットワークフローデータの1つ以上のパケットが宛先に到達できない場合に発生します。パケットのドロップは、次に示す YAML の例の仕様に合わせて **FlowCollector** を編集することで追跡できます。



重要

この機能を有効にすると、CPU とメモリーの使用量が増加します。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. パケットドロップ用に **FlowCollector** カスタムリソースを設定します。以下はその例です。

FlowCollector の設定例

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  agent:
    type: eBPF
    ebpf:
      features:
        - PacketDrop
      privileged: true

```

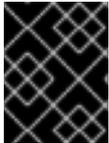
- ① **spec.agent.ebpf.features** 仕様リストに **PacketDrop** パラメーターをリストすることで、各ネットワークフローにおけるパケットドロップの報告を開始できます。
- ② パケットドロップを追跡するには、**spec.agent.ebpf.privileged** の仕様値が **true** である必要があります。

検証

- **Network Traffic** ページを更新すると、**Overview**、**Traffic Flow**、**Topology** ビューにパケットドロップに関する新しい情報が表示されます。
 - a. **Manage panels** で、**Overview** に表示するパケットドロップのグラフィカル表示を新しく選択します。
 - b. **Manage columns** で、**Traffic flows** テーブルに表示するパケットドロップ情報を選択します。
 - i. **Traffic Flows** ビューでは、サイドパネルを展開してパケットドロップの詳細情報を表示することもできます。ホストドロップには **SKB_DROP** という接頭辞が付き、OVS ドロップには **OVS_DROP** という接頭辞が付きます。
 - c. **Topology** ビューでは、ドロップが発生した場所が赤線が表示されます。

7.2.5. DNS 追跡の使用

DNS 追跡を使用すると、ネットワークの監視、セキュリティ分析の実施、DNS 問題のトラブルシューティングを実行できます。次に示す YAML の例の仕様に合わせて **FlowCollector** を編集することで、DNS を追跡できます。



重要

この機能を有効にすると、eBPF agent で CPU とメモリーの使用量の増加が観察されます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **Network Observability** の **Provided APIs** という見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. **FlowCollector** カスタムリソースを設定します。設定例は次のとおりです。

DNS 追跡用に FlowCollector を設定する

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  agent:
    type: eBPF
    ebpf:
      features:
        - DNSTracking
      sampling: 1

```

- 1 **spec.agent.ebpf.features** パラメーターリストを設定すると、Web コンソールで各ネットワークフローの DNS 追跡を有効にできます。
 - 2 より正確なメトリクスと DNS レイテンシー をキャプチャーするために、**sampling** 値を 1 に設定できます。**sampling** 値が 1 より大きい場合、DNS レスポンスコードと DNS ID を含むフローを監視できますが、DNS レイテンシー を監視できる可能性は低くなります。
5. **Network Traffic** ページを更新すると、**Overview** ビューと **Traffic Flow** ビューで表示する新しい DNS 表示と適用可能な新しいフィルターが表示されます。
 - a. **Manage panels** で新しい DNS の選択肢を選択すると、**Overview** にグラフィカルな表現と DNS メトリクスが表示されます。
 - b. **Manage columns** で新しい選択肢を選択すると、DNS 列が **Traffic Flows** ビューに追加されます。
 - c. **DNS Id**、**DNS Error**、**DNS Latency**、**DNS Response Code** などの特定の DNS メトリクスでフィルタリングして、サイドパネルから詳細情報を確認します。**DNS Latency** 列と **DNS Response Code** 列がデフォルトで表示されます。



注記

TCP ハンドシェイクパケットには DNS ヘッダーがありません。DNS ヘッダーのない TCP プロトコルフローの場合、トラフィックフローデータに表示される **DNS Latency**、**ID**、および **Response code** の値が "n/a" になります。"DNSError" が "0" の **Common** フィルターを使用すると、フローデータをフィルタリングして、DNS ヘッダーを持つフローのみを表示できます。

7.2.6. RTT トレーシングの使用

次に示す YAML の例の仕様に合わせて **FlowCollector** を編集することで、RTT を追跡できます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** という見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. RTT トレーシング用に **FlowCollector** カスタムリソースを設定します。次に例を示します。

FlowCollector の設定例

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  agent:
    type: eBPF
    eBPF:
      features:
        - FlowRTT ①
```

- ① **spec.agent.eBPF.features** 仕様リストに **FlowRTT** パラメーターをリストすることで、RTT ネットワークフローのトレースを開始できます。

検証

Network Traffic ページを更新すると、**Overview**、**Traffic Flow**、**Topology** ビューに RTT に関する新しい情報が表示されます。

- a. **Overview** で、**Manage panels** の新しい選択肢を選択して、表示する RTT のグラフィカル表示を選択します。
- b. **Traffic flows** テーブルに **Flow RTT** 列が表示されます。**Manage columns** で表示を管理できます。
- c. **Traffic Flows** ビューでは、サイドパネルを展開して RTT の詳細情報を表示することもできます。

フィルタリングの例

- i. **Common** フィルター → **Protocol** をクリックします。
 - ii. **TCP**、**Ingress** の方向に基づいてネットワークフローデータをフィルタリングし、10,000,000 ナノ秒 (10 ms) を超える **FlowRTT** 値を探します。
 - iii. **Protocol** フィルターを削除します。
 - iv. **Common** フィルターで 0 より大きい **Flow RTT** 値をフィルタリングします。
- d. **Topology** ビューで、Display option ドロップダウンをクリックします。次に、**edge labels** のドロップダウンリストで **RTT** をクリックします。

7.2.6.1. ヒストグラムの使用

Show histogram をクリックすると、フローの履歴を棒グラフとして視覚化するためのツールバービューが表示されます。ヒストグラムは、時間の経過に伴うログの数を示します。ヒストグラムの一部を選択して、ツールバーに続く表でネットワークフローデータをフィルタリングできます。

7.2.7. アベイラビリティゾーンの使用

クラスターのアベイラビリティゾーンに関する情報を収集するように **FlowCollector** を設定できます。この設定により、ノードに適用される topology.kubernetes.io/zone ラベル値を使用してネットワークフローデータをエンリッチできます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. **FlowCollector** カスタムリソースを設定し、**spec.processor.addZone** パラメーターを **true** に設定します。設定例は次のとおりです。

アベイラビリティゾーン収集用に FlowCollector を設定する

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  # ...
  processor:
    addZone: true
  # ...
```

検証

Network Traffic ページを更新すると、**Overview**、**Traffic Flow**、**Topology** ビューにアベイラビリティゾーンに関する新しい情報が表示されます。

1. **Overview** タブに、使用可能な **Scope** として **Zones** が表示されます。
2. **Network Traffic** → **Traffic flows** の **SrcK8S_Zone** フィールドと **DstK8S_Zone** フィールドに **Zones** が表示されます。

3. **Topology** ビューで、**Scope** または **Group** として **Zones** を設定できます。

7.2.8. グローバルルールを使用した eBPF フローデータのフィルタリング

FlowCollector を設定して、グローバルルールを使用して eBPF フローをフィルタリングし、eBPF フローテーブルにキャッシュされるパケットのフローを制御できます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **Network Observability** の **Provided APIs** という見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. 次のサンプル設定と同じように **FlowCollector** カスタムリソースを設定します。

例7.1 特定の Pod IP エンドポイントへの Kubernetes サービストラフィックをフィルタリングする

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF
  ebpf:
    flowFilter:
      action: Accept ①
      cidr: 172.210.150.1/24 ②
      protocol: SCTP
      direction: Ingress
      destPortRange: 80-100
      peerIP: 10.10.10.10
      enable: true ③
```

- ① 必須の **action** パラメーターは、フローフィルタールールに対して実行されるアクションを示します。可能な値は **Accept** または **Reject** です。
- ② 必須の **cidr** パラメーターは、フローフィルタールールの IP アドレスと CIDR マスクを指定します。IPv4 および IPv6 アドレス形式をサポートしています。すべての IP アドレスと照合する場合、IPv4 の場合は **0.0.0.0/0**、IPv6 の場合は **:::0** を使用できます。
- ③ この機能を有効にするには、**spec.agent.ebpf.flowFilter.enable** を **true** に設定する必要があります。

例7.2 クラスタ外へのアドレスへのフローを確認する

```
apiVersion: flows.netobserv.io/v1beta2
```

```

kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
agent:
  type: eBPF
  ebpf:
    flowFilter:
      action: Accept ①
      cidr: 0.0.0.0/0 ②
      protocol: TCP
      direction: Egress
      sourcePort: 100
      peerIP: 192.168.127.12 ③
      enable: true ④

```

- ① **flowFilter** 仕様の基準に基づいてフローを **Accept** (受け入れる) ことができます。
- ② **cidr** 値 **0.0.0.0/0** は、すべての IP アドレスと一致します。
- ③ **peerIP** を **192.168.127.12** に設定した上でフローを確認します。
- ④ この機能を有効にするには、**spec.agent.ebpf.flowFilter.enable** を **true** に設定する必要があります。

7.3. トポロジービューからのネットワークトラフィックの観察

Topology ビューには、ネットワークフローとトラフィック量がグラフィカルに表示されます。管理者は、Topology ビューを使用して、アプリケーション全体のトラフィックデータを監視できます。

7.3.1. トポロジービューの操作

管理者は、Topology ビューに移動して、コンポーネントの詳細とメトリクスを確認できます。

手順

1. **Observe** → **Network Traffic** に移動します。
2. **Network Traffic** ページで、**Topology** タブをクリックします。

Topology 内の各コンポーネントをクリックして、コンポーネントの詳細とメトリクスを表示できます。

7.3.2. トポロジービューの詳細オプションの設定

Show advanced options を使用して、ビューをカスタマイズおよびエクスポートできます。詳細オプションビューには、次の機能があります。

- **Find in view** で必要なコンポーネントを検索します。

- **Display options:** 次のオプションを設定するには:
 - **Edge labels:** 指定した測定値をエッジラベルとして表示します。デフォルトでは、**Average rate** が **Bytes** 単位で表示されます。
 - **Scope:** ネットワークトラフィックが流れるコンポーネントのスコープを選択します。デフォルト値は **Namespace** です。
 - **Groups:** コンポーネントをグループ化することにより、所有権をわかりやすくします。デフォルト値は **None** です。
 - **Layout:** グラフィック表示のレイアウトを選択します。デフォルト値は **ColaNoForce** です。
 - **表示:** 表示する必要がある詳細を選択します。デフォルトでは、すべてのオプションがチェックされています。使用可能なオプションは、**Edges**、**Edges label**、および **Badges** です。
 - **Truncate labels:** ドロップダウンリストから必要なラベルの幅を選択します。デフォルト値は **M** です。
 - グループを **Collapse groups** をデプロイメントまたは折りたたむ。グループはデフォルトで展開されています。**Groups** の値が **None** の場合、このオプションは無効になります。

7.3.2.1. トポロジービューのエクスポート

ビューをエクスポートするには、トポロジービューのエクスポート をクリックします。ビューは PNG 形式でダウンロードされます。

7.4. ネットワークトラフィックのフィルタリング

デフォルトでは、ネットワークトラフィックページには、**FlowCollector** インスタンスで設定されたデフォルトフィルターに基づいて、クラスター内のトラフィックフローデータが表示されます。フィルターオプションを使用して、プリセットフィルターを変更することにより、必要なデータを観察できます。

クエリーオプション

以下に示すように、**Query Options** を使用して検索結果を最適化できます。

- **Log Type:** 利用可能なオプション **Conversation** と **Flows** では、フローログ、新しい会話、完了した会話、および長い会話の更新を含む定期的なレコードであるハートビートなどのログタイプ別にフローをクエリーする機能が提供されます。会話は、同じピア間のフローの集合体です。
- **Match filters:** 高度なフィルターで選択されたさまざまなフィルターパラメーター間の関係を決定できます。利用可能なオプションは、**Match all** と **Match any** です。**Match all** はすべての値に一致する結果を提供し、**Match any** は入力された値のいずれかに一致する結果を提供します。デフォルト値は **Match all** です。
- **Datasource:** クエリーに使用するデータソース (**Loki**、**Prometheus**、**Auto**) を選択できます。Loki ではなく Prometheus をデータソースとして使用すると、パフォーマンスが大幅に向上します。ただし、Prometheus がサポートするフィルターと集計は限られています。デフォルトのデータソースは **Auto** です。Auto の場合、Prometheus をサポートしているクエリーでは Prometheus を使用し、サポートしていないクエリーでは Loki を使用します。

- **Drops filter:** 次のクエリーオプションを使用して、各レベルのドロップパケットを表示できます。
 - **Fully dropped** の場合、パケットが完全にドロップされたフローレコードが表示されます。
 - **Containing drops** の場合、ドロップが発生したが送信可能なフローレコードが表示されます。
 - **Without drops** の場合、送信されたパケットを含むレコードが表示されます。
 - **All** の場合、上記のレコードがすべて表示されます。
- **Limit:** 内部バックエンドクエリーのデータ制限。マッチングやフィルターの設定に応じて、トラフィックフローデータの数に指定した制限内で表示されます。

クイックフィルター

クイックフィルター ドロップダウンメニューのデフォルト値は、**FlowCollector** 設定で定義されません。コンソールからオプションを変更できます。

高度なフィルター

ドロップダウンリストからフィルタリングするパラメーターを選択することで、詳細フィルター (**Common**、**Source**、**Destination**) を設定できます。フローデータは選択に基づいてフィルタリングされます。適用されたフィルターを有効または無効にするには、フィルターオプションの下にリストされている適用されたフィルターをクリックします。

↑ **One way** と ↑↓ **Back and forth** のフィルタリングを切り替えることができます。↑ **One way** フィルターを使用すると、選択したフィルターに基づき **Source** および **Destination** トラフィックのみが表示されます。**Swap** を使用すると、**Source** および **Destination** トラフィックの方向ビューを変更できます。↑↓ **Back and forth** フィルターには、**Source** フィルターと **Destination** フィルターによる戻りトラフィックが含まれます。ネットワークトラフィックの方向性があるフローは、Traffic flows テーブルの **Direction** 列に、ノード間トラフィックの場合は **Ingress** or **Egress** として、シングルノード内のトラフィックの場合は **Inner** として表示されます。

Reset default をクリックして既存のフィルターを削除し、**FlowCollector** 設定で定義したフィルターを適用できます。



注記

テキスト値を指定する規則を理解するには、**詳細** をクリックします。

または、**Namespaces**、**Services**、**Routes**、**Nodes**、および **Workloads** ページの **Network Traffic** タブでトラフィックフローデータにアクセスして、対応する集約のフィルタリングされたデータを提供します。

関連情報

FlowCollector でのクイックフィルターの設定の詳細は、[クイックフィルターの設定](#) および [Flow Collector のサンプルリソース](#) を参照してください。

第8章 ダッシュボードとアラートでのメトリクスの使用

Network Observability Operator は、**flowlogs-pipeline** を使用してフローログからメトリクスを生成します。これらのメトリクスは、カスタムアラートを設定し、ダッシュボードを表示することで利用できます。

8.1. NETWORK OBSERVABILITY メトリクスのダッシュボードの表示

OpenShift Container Platform コンソールの **Overview** タブでは、クラスター上のネットワークトラフィックフローの集約された全体的なメトリクスを表示できます。ノード、namespace、所有者、Pod、サービスごとに情報を表示することを選択できます。フィルターと表示オプションを使用して、メトリクスをさらに絞り込むこともできます。

手順

1. Web コンソールの **Observe** → **Dashboards** で、**Netobserv** ダッシュボードを選択します。
2. 次のカテゴリのネットワークトラフィックメトリクスを表示します。各カテゴリには、ノード、namespace、送信元、宛先ごとのサブセットがあります。
 - バイトレート
 - パケットドロップ
 - DNS
 - RTT
3. **Netobserv/Health** ダッシュボードを選択します。
4. Operator の健全性に関する次のカテゴリのメトリクスを表示します。各カテゴリには、ノード、namespace、送信元、宛先ごとのサブセットがあります。
 - フロー
 - フローのオーバーヘッド
 - フローレート
 - エージェント
 - プロセッサ
 - Operator

Infrastructure および Application メトリクスは、namespace とワークロードの分割ビューで表示されます。

8.2. 定義済みのメトリクス

flowlogs-pipeline によって生成されるメトリクスは、**FlowCollector** カスタムリソースの **spec.processor.metrics.includeList** で設定して追加または削除できます。

8.3. NETWORK OBSERVABILITY メトリクス

Prometheus ルールの **includeList** メトリクスを使用してアラートを作成することもできます。「アラートの作成」の例を参照してください。

コンソールで **Observe** → **Metrics** を選択するなどして Prometheus でこれらのメトリクスを探す場合、またはアラートを定義する場合、すべてのメトリクス名に **netobserv_** という接頭辞が付きます。たとえば、**netobserv_namespace_flows_total** です。利用可能なメトリクス名は以下のとおりです。

includeList のメトリクス名

名前の後にアスタリスク * が付いているものは、デフォルトで有効です。

- **namespace_egress_bytes_total**
- **namespace_egress_packets_total**
- **namespace_ingress_bytes_total**
- **namespace_ingress_packets_total**
- **namespace_flows_total ***
- **node_egress_bytes_total**
- **node_egress_packets_total**
- **node_ingress_bytes_total ***
- **node_ingress_packets_total**
- **node_flows_total**
- **workload_egress_bytes_total**
- **workload_egress_packets_total**
- **workload_ingress_bytes_total ***
- **workload_ingress_packets_total**
- **workload_flows_total**

PacketDrop のメトリクス名

PacketDrop 機能が (**privileged** モードにより) **spec.agent.ebpf.features** で有効になっている場合、次の追加のメトリクスを使用できます。

- **namespace_drop_bytes_total**
- **namespace_drop_packets_total ***
- **node_drop_bytes_total**
- **node_drop_packets_total**
- **workload_drop_bytes_total**
- **workload_drop_packets_total**

DNS のメトリクス名

DNSTracking 機能が **spec.agent.ebpf.features** で有効になっている場合、次の追加のメトリクスを使用できます。

- **namespace_dns_latency_seconds** *
- **node_dns_latency_seconds**
- **workload_dns_latency_seconds**

FlowRTT のメトリクス名

FlowRTT 機能が **spec.agent.ebpf.features** で有効になっている場合、次の追加のメトリクスを使用できます。

- **namespace_rtt_seconds** *
- **node_rtt_seconds**
- **workload_rtt_seconds**

8.4. アラートの作成

Netobserv ダッシュボードメトリクスのカスタム警告ルールを作成すると、定義した条件が満たされたときにアラートをトリガーできます。

前提条件

- cluster-admin ロールを持つユーザー、またはすべてのプロジェクトの表示権限を持つユーザーとしてクラスターにアクセスできる。
- Network Observability Operator がインストールされています。

手順

1. インポートアイコン + をクリックして、YAML ファイルを作成します。
2. アラートルール設定を YAML ファイルに追加します。次の YAML サンプルでは、クラスターの Ingress トラフィックが宛先ワークロードごとの指定しきい値 (10 MBps) に達したときに、アラートが作成されます。

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: netobserv-alerts
  namespace: openshift-monitoring
spec:
  groups:
    - name: NetObservAlerts
      rules:
        - alert: NetObservIncomingBandwidth
          annotations:
            message: |-
              {{ $labels.job }}: incoming traffic exceeding 10 MBps for 30s on {{
              $labels.DstK8S_OwnerType }} {{ $labels.DstK8S_OwnerName }} ({{
              $labels.DstK8S_Namespace }}).
```

```
summary: "High incoming traffic."
expr: sum(rate(netobserv_workload_ingress_bytes_total
{SrcK8S_Namespace="openshift-ingress"}[1m])) by (job, DstK8S_Namespace,
DstK8S_OwnerName, DstK8S_OwnerType) > 10000000 ❶
for: 30s
labels:
severity: warning
```

- ❶ **netobserv_workload_ingress_bytes_total** メトリクスは、**spec.processor.metrics.includeList** でデフォルトで有効です。

3. **Create** をクリックして設定ファイルをクラスターに適用します。

8.5. カスタムメトリクス

FlowMetric API を使用して、フローログデータからカスタムメトリクスを作成できます。収集されるすべてのフローログデータには、送信元名や宛先名など、ログごとのラベルが付けられたフィールドがいくつかあります。これらのフィールドを Prometheus ラベルとして活用して、ダッシュボード上のクラスター情報をカスタマイズできます。

8.6. FLOWMETRIC API を使用したカスタムメトリクスの設定

FlowMetric API を設定して、フローログデータフィールドを Prometheus ラベルとして使用してカスタムメトリクスを作成できます。複数の **FlowMetric** リソースをプロジェクトに追加して、複数のダッシュボードビューを表示できます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しで、**FlowMetric** を選択します。
3. **Project**: ドロップダウンリストで、Network Observability Operator インスタンスのプロジェクトを選択します。
4. **Create FlowMetric** をクリックします。
5. 次のサンプル設定と同じように **FlowMetric** リソースを設定します。

例8.1 クラスターの外部ソースから受信した Ingress バイト数を追跡するメトリクスを生成する

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-traffic
  namespace: netobserv ❶
spec:
  metricName: cluster_external_ingress_bytes_total ❷
  type: Counter ❸
  valueField: Bytes
  direction: Ingress ❹
  labels:
  [DstK8S_HostName,DstK8S_Namespace,DstK8S_OwnerName,DstK8S_OwnerType] ❺
```

```
filters:
- field: SrcSubnetLabel
  matchType: Absence
```

6

- 1 **FlowMetric** リソースは、**FlowCollector spec.namespace** で定義された namespace (デフォルトでは **netobserv**) に作成する必要があります。
- 2 Prometheus メトリクスの名前。Web コンソールでは接頭辞 **netobserv-
<metricName>** とともに表示されます。
- 3 **type** はメトリクスのタイプを指定します。**Counter type** は、バイト数またはパケット数をカウントするのに役立ちます。
- 4 キャプチャーするトラフィックの方向。指定しない場合は、Ingress と Egress の両方がキャプチャーされ、重複したカウントが発生する可能性があります。
- 5 ラベルは、メトリクスの外観とさまざまなエンティティー間の関係を定義します。また、メトリクスのカーディナリティーも定義します。たとえば、**SrcK8S_Name** はカーディナリティーが高いメトリクスです。
- 6 リストされた基準に基づいて結果を絞り込みます。この例では、**SrcSubnetLabel** が存在しないフローのみを照合することによって、クラスターの外部トラフィックのみを選択します。これは、(**spec.processor.subnetLabels** により) サブネットラベル機能が有効になっていることを前提としています。この機能はデフォルトで有効になっています。

検証

1. Pod が更新されたら、**Observe** → **Metrics** に移動します。
2. **Expression** フィールドにメトリクス名を入力して、対応する結果を表示します。 **topk(5, sum(rate(netobserv_cluster_external_ingress_bytes_total{DstK8S_Namespace="my-namespace"}[2m])) by (DstK8S_HostName, DstK8S_OwnerName, DstK8S_OwnerType))** などの式を入力することもできます。

例8.2 クラスター外部 Ingress トラフィックの RTT 遅延を表示する

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-rtt
  namespace: netobserv 1
spec:
  metricName: cluster_external_ingress_rtt_seconds
  type: Histogram 2
  valueField: TimeFlowRttNs
  direction: Ingress
  labels:
  [DstK8S_HostName,DstK8S_Namespace,DstK8S_OwnerName,DstK8S_OwnerType]
  filters:
  - field: SrcSubnetLabel
    matchType: Absence
  - field: TimeFlowRttNs
```

```

matchType: Presence
divider: "1000000000" ③
buckets: [".001", ".005", ".01", ".02", ".03", ".04", ".05", ".075", ".1", ".25", "1"] ④

```

- ① **FlowMetric** リソースは、**FlowCollector spec.namespace** で定義された namespace (デフォルトでは **netobserv**) に作成する必要があります。
- ② **type** はメトリクスのタイプを指定します。**Histogram type** は、遅延値 (**TimeFlowRttNs**) に役立ちます。
- ③ ラウンドトリップタイム (RTT) はフロー内でナノ秒単位で提供されるため、秒単位に変換するには除数として 10 億を使用します。これは Prometheus ガイドラインの標準です。
- ④ カスタムバケットは RTT の精度を指定します。最適な精度は 5 ミリ秒から 250 ミリ秒の範囲です。

検証

1. Pod が更新されたら、**Observe** → **Metrics** に移動します。
2. **Expression** フィールドにメトリクス名を入力して、対応する結果を表示できます。



重要

カーディナリティーが高いと、Prometheus のメモリー使用量に影響する可能性があります。特定のラベルのカーディナリティーが高いかどうかは、[ネットワークフロー形式のリファレンス](#) で確認できます。

8.7. FLOWMETRIC API を使用したカスタムグラフの設定

OpenShift Container Platform Web コンソールでダッシュボードのグラフを生成できます。グラフは、**FlowMetric** リソースの **charts** セクションを定義することで、管理者の **Dashboard** メニューで表示できます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しで、**FlowMetric** を選択します。
3. **Project**: ドロップダウンリストで、Network Observability Operator インスタンスのプロジェクトを選択します。
4. **Create FlowMetric** をクリックします。
5. 次のサンプル設定と同じように **FlowMetric** リソースを設定します。

例8.3 クラスターの外部ソースから受信した **Ingress** バイト数を追跡するためのチャート

```

apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:

```

```

name: flowmetric-cluster-external-ingress-traffic
namespace: netobserv ❶
# ...
charts:
- dashboardName: Main ❷
  title: External ingress traffic
  unit: Bps
  type: SingleStat
  queries:
  - promQL: "sum(rate($METRIC[2m]))"
    legend: ""
- dashboardName: Main ❸
  sectionName: External
  title: Top external ingress traffic per workload
  unit: Bps
  type: StackArea
  queries:
  - promQL: "sum(rate($METRIC{DstK8S_Namespace!=\"\"}[2m])) by (DstK8S_Namespace,
DstK8S_OwnerName)"
    legend: "{{DstK8S_Namespace}} / {{DstK8S_OwnerName}}"
# ...

```

- ❶ **FlowMetric** リソースは、**FlowCollector spec.namespace** で定義された namespace (デフォルトでは **netobserv**) に作成する必要があります。

検証

- Pod が更新されたら、**Observe** → **Dashboards** に移動します。
- NetObserv / Main** ダッシュボードを検索します。**NetObserv / Main** ダッシュボードの下、または必要に応じて作成したダッシュボード名の下にある次の2つのパネルを確認します。
 - すべてのディメンションにわたりグローバルな外部 Ingress レートを合計したテキスト形式の単一の統計
 - 上記と同じメトリクスを示す、宛先ワークロードごとの時系列グラフ

クエリ言語の詳細は、[Prometheus のドキュメント](#) を参照してください。

例8.4 クラスター外部 Ingress トラフィックの RTT 遅延のグラフ

```

apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-traffic
  namespace: netobserv ❶
# ...
charts:
- dashboardName: Main ❷
  title: External ingress TCP latency
  unit: seconds
  type: SingleStat
  queries:

```

```

- promQL: "histogram_quantile(0.99, sum(rate($METRIC_bucket[2m])) by (le)) > 0"
  legend: "p99"
- dashboardName: Main 3
  sectionName: External
  title: "Top external ingress sRTT per workload, p50 (ms)"
  unit: seconds
  type: Line
  queries:
  - promQL: "histogram_quantile(0.5, sum(rate($METRIC_bucket{DstK8S_Namespace!=\"\"}[2m])) by (le,DstK8S_Namespace,DstK8S_OwnerName))*1000 > 0"
    legend: "{{DstK8S_Namespace}} / {{DstK8S_OwnerName}}"
- dashboardName: Main 4
  sectionName: External
  title: "Top external ingress sRTT per workload, p99 (ms)"
  unit: seconds
  type: Line
  queries:
  - promQL: "histogram_quantile(0.99, sum(rate($METRIC_bucket{DstK8S_Namespace!=\"\"}[2m])) by (le,DstK8S_Namespace,DstK8S_OwnerName))*1000 > 0"
    legend: "{{DstK8S_Namespace}} / {{DstK8S_OwnerName}}"
# ...

```

1 **FlowMetric** リソースは、**FlowCollector spec.namespace** で定義された namespace (デフォルトでは **netobserv**) に作成する必要があります。

2 **3** **4** 異なる **dashboardName** を使用すると、接頭辞が **Netobserv** である新しいダッシュボードが作成されます。たとえば、**Netobserv / <dashboard_name>** です。

この例では、**histogram_quantile** 関数を使用して **p50** と **p99** を表示します。

ヒストグラムを作成するときに自動的に生成されるメトリクス **\$METRIC_sum** をメトリクス **\$METRIC_count** で割ることで、ヒストグラムの平均を表示できます。前述の例では、これを実行するための Prometheus クエリーは次のとおりです。

```

promQL: "(sum(rate($METRIC_sum{DstK8S_Namespace!=\"\"}[2m])) by (DstK8S_Namespace,DstK8S_OwnerName) / sum(rate($METRIC_count{DstK8S_Namespace!=\"\"}[2m])) by (DstK8S_Namespace,DstK8S_OwnerName))*1000"

```

検証

1. Pod が更新されたら、**Observe** → **Dashboards** に移動します。
2. **NetObserv / Main** ダッシュボードを検索します。**NetObserv / Main** ダッシュボードの下、または必要に応じて作成したダッシュボード名の下にある新しいパネルを表示します。

クエリー言語の詳細は、[Prometheus のドキュメント](#) を参照してください。

関連情報

- [ユーザー定義プロジェクトのアラートルールの作成](#)
- [高カーディナリティーメトリクスのトラブルシューティング - Prometheus が大量のディスク領域を消費している理由の特定](#)

第9章 NETWORK OBSERVABILITY OPERATOR の監視

Web コンソールを使用して、Network Observability Operator の健全性に関連するアラートを監視できます。

9.1. 健全性ダッシュボード

Network Observability Operator の健全性とリソース使用状況に関するメトリクスは、Web コンソールの **Observe → Dashboards** ページにあります。Operator の健全性に関する次のカテゴリのメトリクスを表示できます。

- 1秒あたりのフロー数
- サンプリング
- 過去1分間のエラー数
- 1秒あたりのドロップされたフロー数
- flowlogs-pipeline 統計
- flowlogs-pipeline 統計ビュー
- eBPF エージェント統計ビュー
- Operator 統計
- リソースの使用状況

9.2. 健全性アラート

アラートがトリガーされると、ダッシュボードに誘導するヘルスアラートバナーが **Network Traffic** ページと **Home** ページに表示されることがあります。アラートは次の場合に生成されます。

- **NetObservLokiError** アラートは、Loki 取り込みレート制限に達した場合など、Loki エラーが原因で **flowlogs-pipeline** ワークロードがフローをドロップすると発生します。
- **NetObservNoFlows** アラートは、一定時間フローが取り込まれない場合に発生します。
- **NetObservFlowsDropped** アラートは、Network Observability eBPF エージェントの HashMap テーブルがいっぱいで、eBPF エージェントのパフォーマンスが低下した状態でフローを処理している場合、または容量制限がトリガーされた場合に発生します。

9.3. 健全性情報の表示

Web コンソールの **Dashboards** ページから、Network Observability Operator の健全性とリソースの使用状況に関するメトリクスにアクセスできます。

前提条件

- Network Observability Operator がインストールされています。
- **cluster-admin** ロールまたはすべての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にアクセスできる。

手順

1. Web コンソールの **Administrator** パースペクティブから、**Observe** → **Dashboards** に移動します。
2. **Dashboards** ドロップダウンメニューから、**Netobserv/Health** を選択します。
3. ページに表示された Operator の健全性に関するメトリクスを確認します。

9.3.1. ヘルスアラートの無効化

FlowCollector リソースを編集して、ヘルスアラートをオプトアウトできます。

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. 次の YAML サンプルのように、**spec.processor.metrics.disableAlerts** を追加してヘルスアラートを無効にします。

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  processor:
    metrics:
      disableAlerts: [NetObservLokiError, NetObservNoFlows] 1
```

- 1** 無効にするアラートの1つまたは両方のタイプを含むリストを指定できます。

9.4. NETOBSERV ダッシュボードの LOKI レート制限アラートの作成

Netobserv ダッシュボードメトリクスのカスタム警告ルールを作成して、Loki のレート制限に達した場合にアラートをトリガーできます。

前提条件

- cluster-admin ロールを持つユーザー、またはすべてのプロジェクトの表示権限を持つユーザーとしてクラスターにアクセスできる。
- Network Observability Operator がインストールされています。

手順

1. インポートアイコン + をクリックして、YAML ファイルを作成します。
2. アラートルール設定を YAML ファイルに追加します。次の YAML サンプルでは、Loki のレート制限に達した場合にアラートが作成されます。

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
```

```

metadata:
  name: loki-alerts
  namespace: openshift-monitoring
spec:
  groups:
  - name: LokiRateLimitAlerts
    rules:
    - alert: LokiTenantRateLimit
      annotations:
        message: |-
          {{ $labels.job }} {{ $labels.route }} is experiencing 429 errors.
          summary: "At any number of requests are responded with the rate limit error code."
          expr: sum(irate(loki_request_duration_seconds_count{status_code="429"}[1m])) by (job,
          namespace, route) / sum(irate(loki_request_duration_seconds_count[1m])) by (job,
          namespace, route) * 100 > 0
          for: 10s
          labels:
            severity: warning

```

3. **Create** をクリックして設定ファイルをクラスターに適用します。

9.5. EBPF エージェントアラートの使用

Network Observability eBPF エージェントの HashMap テーブルがいっぱいになるか、容量制限がトリガーされると、アラート **NetObservAgentFlowsDropped** がトリガーされます。このアラートが表示された場合は、次の例に示すように、**FlowCollector** の **cacheMaxFlows** を増やすことを検討してください。



注記

cacheMaxFlows を増やすと、eBPF エージェントのメモリー使用量が増加する可能性があります。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **Network Observability Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
3. **cluster** を選択し、**YAML** タブを選択します。
4. 次の YAML サンプルに示すように、**spec.agent.ebpf.cacheMaxFlows** の値を増やします。

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF
    ebpf:
      cacheMaxFlows: 200000 1

```

■

- 1 **NetObservAgentFlowsDropped** アラート発生時の値から **cacheMaxFlows** 値を増やします。

関連情報

- ダッシュボードに表示できるアラート作成の詳細は、[ユーザー定義プロジェクトのアラートルールの作成](#) を参照してください。

第10章 スケジューリングリソース

テイントおよび容認 (Toleration) により、ノードはノード上でスケジュールする必要のある (またはスケジュールすべきでない) Pod を制御できます。

ノードセクターは、ノードのカスタムラベルと Pod で指定されたセクターを使用して定義されるキー/値のペアのマップを指定します。

Pod がノードで実行する要件を満たすには、Pod にはノードのラベルと同じキー/値のペアがなければなりません。

10.1. 特定のノードにおける NETWORK OBSERVABILITY デプロイメント

FlowCollector を設定して、特定のノードにおける Network Observability コンポーネントのデプロイメントを制御できます。 **spec.agent.ebpf.advanced.scheduling**、 **spec.processor.advanced.scheduling**、 および **spec.consolePlugin.advanced.scheduling** 仕様で、次の設定が可能です。

- **NodeSelector**
- **Tolerations**
- **Affinity**
- **PriorityClassName**

spec.<component>.advanced.scheduling のサンプル **FlowCollector** リソース

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
# ...
advanced:
  scheduling:
    tolerations:
      - key: "<taint key>"
        operator: "Equal"
        value: "<taint value>"
        effect: "<taint effect>"
        nodeSelector:
          <key>: <value>
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
            - matchExpressions:
                - key: name
                  operator: In
                  values:
                    - app-worker-node
          priorityClassName: ""
# ...

```

関連情報

- [テイントおよび容認 \(Toleration\) について](#)
- [Assign Pods to Nodes](#) (Kubernetes ドキュメント)
- [Pod Priority and Preemption](#) (Kubernetes ドキュメント)

第11章 NETWORK OBSERVABILITY CLI

11.1. NETWORK OBSERVABILITY CLI のインストール



警告

ネットワーク可観測性 CLI (**oc netobserv**)は一時的に利用できず、[OCPBUGS-36146](#) で解決することが期待されています。

Network Observability CLI (**oc netobserv**) は、Network Observability Operator とは別にデプロイされます。CLI は、OpenShift CLI (**oc**) プラグインとして利用できます。CLI は、ネットワーク可観測性を活用して、迅速にデバッグおよびトラブルシューティングを行う軽量な方法です。



重要

Network Observability CLI (**oc netobserv**) 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機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

11.1.1. Network Observability CLI について

Network Observability CLI (**oc netobserv**) を使用すると、ネットワークの問題を迅速にデバッグおよびトラブルシューティングできます。Network Observability CLI は、eBPF エージェントを利用して収集したデータを一時的なコレクター Pod にストリーミングするフローおよびパケット可視化ツールです。キャプチャー中に永続的なストレージは必要ありません。実行後、出力がローカルマシンに転送されます。そのため、Network Observability Operator をインストールしなくても、パケットとフローデータをすばやくライブで把握できます。



重要

CLI のキャプチャーは、8 - 10 分などの短い期間のみ実行されるように設計されています。実行時間が長すぎると、実行中のプロセスを削除するのが難しくなる可能性があります。

11.1.2. Network Observability CLI のインストール

Network Observability CLI (**oc netobserv**) のインストールは、Network Observability Operator のインストールとは別の手順です。つまり、OperatorHub から Operator をインストールした場合でも、CLI を別途インストールする必要があります。



注記

必要に応じて、Krew を使用して **netobserv** CLI プラグインをインストールできます。詳細は、「Krew を使用した CLI プラグインのインストール」を参照してください。

前提条件

- OpenShift CLI (**oc**) をインストールしておく。
- macOS または Linux オペレーティングシステムがある。

手順

1. **oc netobserv CLI tar ファイル** をダウンロードします。
2. アーカイブを展開します。

```
$ tar xvf netobserv-cli.tar.gz
```

3. ファイルを実行可能にします。

```
$ chmod +x ./build/oc-netobserv
```

4. 展開した **netobserv-cli** バイナリーを、**/usr/local/bin/** などの **PATH** 上のディレクトリーに移動します。

```
$ sudo mv ./build/oc-netobserv /usr/local/bin/
```

検証

- **oc netobserv** が利用可能であることを確認します。

```
$ oc netobserv version
```

出力例

```
Netobserv CLI version <version>
```

関連情報

- [CLI プラグインのインストールおよび使用](#)
- [Krew を使用した CLI プラグインのインストール](#)

11.2. NETWORK OBSERVABILITY CLI の使用

フローやパケットデータをターミナル内で直接可視化およびフィルタリングすると、特定のポートを使用しているユーザーの特定など、詳細な使用状況を確認できます。Network Observability CLI は、フローを JSON およびデータベースファイルとして、パケットを PCAP ファイルとして収集します。これらのファイルはサードパーティーツールで使用できます。

11.2.1. フローのキャプチャー

フローをキャプチャーし、データ内の任意のリソースまたはゾーンでフィルタリングすると、2つのゾーン間のラウンドトリップタイム (RTT) を表示するなどのユースケースに対応できます。CLIでのテーブル視覚化により、表示機能とフロー検索機能が提供されます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- Network Observability CLI (**oc netobserv**) プラグイン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、フィルターを有効にしてフローをキャプチャーします。

```
$ oc netobserv flows --enable_filter=true --action=Accept --cidr=0.0.0.0/0 --protocol=TCP --port=49051
```

2. ターミナルの **live table filter** プロンプトでフィルターを追加して、受信するフローをさらに絞り込みます。以下に例を示します。

```
live table filter: [SrcK8S_Zone:us-west-1b] press enter to match multiple regular expressions at once
```

3. キャプチャーを停止するには、**Ctrl+C** を押します。キャプチャーされたデータは、CLI のインストールに使用したのと同じパスにある **./output** ディレクトリー内の2つの異なるファイルに書き込まれます。
4. キャプチャーされたデータを、JSON ファイル **./output/flow/<capture_date_time>.json** で確認します。このファイルには、キャプチャーされたデータの JSON 配列が含まれています。

JSON ファイルの例:

```
{
  "AgentIP": "10.0.1.76",
  "Bytes": 561,
  "DnsErrno": 0,
  "Dscp": 20,
  "DstAddr": "f904:ece9:ba63:6ac7:8018:1e5:7130:0",
  "DstMac": "0A:58:0A:80:00:37",
  "DstPort": 9999,
  "Duplicate": false,
  "Etype": 2048,
  "Flags": 16,
  "FlowDirection": 0,
  "IfDirection": 0,
  "Interface": "ens5",
  "K8S_FlowLayer": "infra",
  "Packets": 1,
  "Proto": 6,
  "SrcAddr": "3e06:6c10:6440:2:a80:37:b756:270f",
  "SrcMac": "0A:58:0A:80:00:01",
  "SrcPort": 46934,
  "TimeFlowEndMs": 1709741962111,
  "TimeFlowRttNs": 121000,
```

```
"TimeFlowStartMs": 1709741962111,
"TimeReceived": 1709741964
}
```

5. SQLite を使用して、`./output/flow/<capture_date_time>.db` データベースファイルを検査できます。以下に例を示します。

- a. 次のコマンドを実行してファイルを開きます。

```
$ sqlite3 ./output/flow/<capture_date_time>.db
```

- b. SQLite **SELECT** ステートメントを実行してデータをクエリーします。次に例を示します。

```
sqlite> SELECT DnsLatencyMs, DnsFlagsResponseCode, DnsId, DstAddr, DstPort,
Interface, Proto, SrcAddr, SrcPort, Bytes, Packets FROM flow WHERE DnsLatencyMs
>10 LIMIT 10;
```

出力例

```
12|NoError|58747|10.128.0.63|57856||17|172.30.0.10|53|284|1
11|NoError|20486|10.128.0.52|56575||17|169.254.169.254|53|225|1
11|NoError|59544|10.128.0.103|51089||17|172.30.0.10|53|307|1
13|NoError|32519|10.128.0.52|55241||17|169.254.169.254|53|254|1
12|NoError|32519|10.0.0.3|55241||17|169.254.169.254|53|254|1
15|NoError|57673|10.128.0.19|59051||17|172.30.0.10|53|313|1
13|NoError|35652|10.0.0.3|46532||17|169.254.169.254|53|183|1
32|NoError|37326|10.0.0.3|52718||17|169.254.169.254|53|169|1
14|NoError|14530|10.0.0.3|58203||17|169.254.169.254|53|246|1
15|NoError|40548|10.0.0.3|45933||17|169.254.169.254|53|174|1
```

11.2.2. パケットのキャプチャー

Network Observability CLI を使用してパケットをキャプチャーできます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- Network Observability CLI (**oc netobserv**) プラグインがインストールされている。

手順

1. フィルターを有効にしてパケットキャプチャーを実行します。

```
$ oc netobserv packets tcp,80
```

2. ターミナルの **live table filter** プロンプトでフィルターを追加して、受信するパケットを絞り込みます。フィルターの例は次のとおりです。

```
live table filter: [SrcK8S_Zone:us-west-1b] press enter to match multiple regular expressions
at once
```

3. キャプチャーを停止するには、**Ctrl+C** を押します。

4. キャプチャーされたデータを確認します。このデータは、CLI のインストールに使用したのと同じパスにある `./output/pcap` ディレクトリー内の1つのファイルに書き込まれます。
 - a. `./output/pcap/<capture_date_time>.pcap` ファイルは Wireshark で開くことができます。

11.2.3. Network Observability CLI のクリーンアップ

`oc netobserv cleanup` を実行して、CLI ワークロードを手動でクリーンアップできます。このコマンドは、クラスターからすべての CLI コンポーネントを削除します。

キャプチャーを終了すると、このコマンドがクライアントによって自動的に実行されます。接続の問題が発生した場合は、手動で実行する必要がある場合があります。

手順

- 以下のコマンドを実行します。

```
$ oc netobserv cleanup
```

関連情報

- [Network Observability CLI リファレンス](#)

11.3. NETWORK OBSERVABILITY CLI (OC NETOBSERV) リファレンス

Network Observability CLI (`oc netobserv`) は、Network Observability Operator で使用できるほとんどの機能とフィルタリングオプションを備えています。コマンドライン引数を渡すことで、機能やフィルタリングオプションを有効にできます。

11.3.1. oc netobserv CLI リファレンス

Network Observability CLI (`oc netobserv`) は、詳細な分析のために、フローデータとパケットデータをキャプチャーするための CLI ツールです。

oc netobserv の構文

```
$ oc netobserv [<command>] [<feature_option>] [<command_options>] 1
```

- 1** 機能オプションは、`oc netobserv flows` コマンドでのみ使用できます。`oc netobserv packets` コマンドでは使用できません。

表11.1 基本コマンド

コマンド	説明
<code>flows</code>	フロー情報をキャプチャーします。サブコマンドについては、「フローキャプチャーのサブコマンド」表を参照してください。
<code>packets</code>	特定のプロトコルまたはポートのペアからパケットをキャプチャーします (<code>netobserv packets --filter=tcp,80</code> など)。パケットキャプチャーの詳細は、「パケットキャプチャーのサブコマンド」表を参照してください。

コマンド	説明
cleanup	Network Observability CLI コンポーネントを削除します。
version	ソフトウェアのバージョンを出力します。
help	ヘルプを表示します。

11.3.1.1. Network Observability のエンリッチメント

Network Observability のエンリッチメントは、ゾーン、ノード、所有者、リソース名 (パケットドロップ、DNS 遅延、ラウンドトリップタイムに関するオプション機能を含む) を表示するためのものです。これは、フローをキャプチャーする場合にのみ有効にできます。これらの情報は、パケットキャプチャーの pcap 出力ファイルには表示されません。

Network Observability のエンリッチメントの構文

```
$ oc netobserv flows [<enrichment_options>] [<subcommands>]
```

表11.2 Network Observability のエンリッチメントのオプション

オプション	説明	使用できる値	デフォルト
--enable_pktdrop	パケットドロップを有効にします。	true、false	false
--enable_rtt	ラウンドトリップタイムを有効にします。	true、false	false
--enable_dns	DNS 追跡を有効にします。	true、false	false
--help	ヘルプを表示します。	-	-
--interfaces	フローに対して照合するインターフェイス。たとえば、 "eth0,eth1" です。	"<interface>"	-

11.3.1.2. フローキャプチャーのオプション

フローキャプチャーには、必須コマンドのほか、パケットドロップ、DNS 遅延、ラウンドトリップタイム、フィルタリングに関する追加機能の有効化などの追加オプションがあります。

oc netobserv flows の構文

```
$ oc netobserv flows [<feature_option>] [<command_options>]
```

表11.3 フローキャプチャーのフィルターオプション

オプション	説明	使用できる値	必須	デフォルト
--enable_filter	フローフィルターを有効にします。	true、false	はい	false
--action	フローに適用するアクション。	Accept、Reject	はい	Accept
--cidr	フローに対して照合する CIDR。	1.1.1.0/24、1::100/64、または 0.0.0.0/0	はい	0.0.0.0/0
--protocol	フローに対して照合するプロトコル。	TCP、UDP、SMTP、ICMP、または ICMPv6	いいえ	-
--direction	フローに対して照合する方向。	Ingress、Egress	いいえ	-
--dport	フローに対して照合する宛先ポート。	80、443、または 49051	いいえ	-
--sport	フローに対して照合する送信元ポート。	80、443、または 49051	いいえ	-
--port	フローに対して照合するポート。	80、443、または 49051	いいえ	-
--sport_range	フローに対して照合する送信元ポートの範囲。	80-100 または 443-445	いいえ	-
--dport_range	フローに対して照合する宛先ポートの範囲。	80-100	いいえ	-
--port_range	フローに対して照合するポート範囲。	80-100 または 443-445	いいえ	-
--icmp_type	フローに対して照合する ICMP タイプ。	8 または 13	いいえ	-
--icmp_code	フローに対して照合する ICMP コード。	0 または 1	いいえ	-

オプション	説明	使用できる値	必須	デフォルト
<code>--peer_ip</code>	フローに対して照合するピア IP。	1.1.1.1 または 1::1	いいえ	-

11.3.1.3. パケットキャプチャーのオプション

パケットキャプチャーデータのポートとプロトコルをフィルタリングできます。

oc netobserv packets の構文

```
$ oc netobserv packets [<option>]
```

表11.4 パケットキャプチャーのフィルターオプション

オプション	説明	必須	デフォルト
<code><protocol>,<port></code>	特定のプロトコルおよびポートペアからパケットをキャプチャーします。コンマを区切り文字として使用します。たとえば、 tcp,80 は tcp プロトコルとポート 80 を指定します。	はい	-

第12章 FLOWCOLLECTOR 設定パラメーター

FlowCollector は、基盤となるデプロイメントを操作および設定するネットワークフロー収集 API のスキーマです。

12.1. FLOWCOLLECTOR API 仕様

説明

FlowCollector は、基盤となるデプロイメントを操作および設定するネットワークフロー収集 API のスキーマです。

型

object

プロパティ	型	説明
apiVersion	string	APIVersion はオブジェクトのこの表現のバージョンスキーマを定義します。サーバーは認識されたスキーマを最新の内部値に変換し、認識されない値は拒否することがあります。詳細は、 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources を参照してください。
kind	string	kind はこのオブジェクトが表す REST リソースを表す文字列の値です。サーバーは、クライアントが要求を送信するエンドポイントからこれを推測できることがあります。これを更新することはできません。キャメルケースを使用します。詳細は、 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds を参照してください。
metadata	object	標準オブジェクトのメタデータ。詳細は、 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata を参照してください。

プロパティ	型	説明
spec	object	<p>FlowCollector リソースの望ましい状態を定義します。</p> <p>*: このドキュメントで "サポート対象外" または "非推奨" と記載されている場合、Red Hat はその機能を公式にサポートしていません。たとえば、コミュニティによって提供され、メンテナンスに関する正式な合意なしに受け入れられた可能性があります。製品のメンテナーは、ベストエフォートに限定してこれらの機能に対するサポートを提供する場合があります。</p>

12.1.1. .metadata

説明

標準オブジェクトのメタデータ。詳細は、<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata> を参照してください。

型

object

12.1.2. .spec

説明

FlowCollector リソースの望ましい状態を定義します。

*: このドキュメントで "サポート対象外" または "非推奨" と記載されている場合、Red Hat はその機能を公式にサポートしていません。たとえば、コミュニティによって提供され、メンテナンスに関する正式な合意なしに受け入れられた可能性があります。製品のメンテナーは、ベストエフォートに限定してこれらの機能に対するサポートを提供する場合があります。

型

object

プロパティ	型	説明
agent	object	フローを抽出するためのエージェント設定。
consolePlugin	object	consolePlugin は、利用可能な場合、OpenShift Container Platform コンソールプラグインに関連する設定を定義します。

プロパティ	型	説明
deploymentModel	string	<p>deploymentModel は、フロー処理に必要なデプロイメントのタイプを定義します。可能な値は次のとおりです。</p> <ul style="list-style-type: none"> - Direct (デフォルト) の場合、フロープロセッサがエージェントから直接リスンします。 - Kafka の場合、プロセッサによって消費される前にフローを Kafka パイプラインに送信します。 <p>Kafka は、より優れたスケーラビリティ、回復性、および高可用性を提供できます (詳細は、https://www.redhat.com/en/topics/integration/what-is-apache-kafka を参照してください)。</p>
exporters	array	exporters は、カスタム消費またはストレージ用の追加のオプションのエクスポートを定義します。
kafka	object	Kafka 設定。Kafka をフローコレクションパイプラインの一部としてブローカーとして使用できます。この設定を利用できるのは、 spec.deploymentModel が Kafka の場合です。
loki	object	loki (フローストア) のクライアント設定。
namespace	string	Network Observability Pod がデプロイされる namespace。
processor	object	processor は、エージェントからフローを受信し、それを強化し、メトリクスを生成し、Loki 永続化レイヤーや使用可能なエクスポーターに転送するコンポーネントの設定を定義します。

プロパティ	型	説明
prometheus	object	prometheus は、コンソールプラグインからメトリクスを取得するために使用されるクエリー設定などの Prometheus 設定を定義します。

12.1.3. .spec.agent

説明

フローを抽出するためのエージェント設定。

型

object

プロパティ	型	説明
ebpf	object	ebpf は、 spec.agent.type が eBPF に設定されている場合、eBPF ベースのフローレポーターに関連する設定を示します。
type	string	type [非推奨 (*)] では、フロートレースエージェントを選択します。以前は、このフィールドでは eBPF または IPFIX を選択できました。現在、 eBPF のみが許可されているため、このフィールドは非推奨であり、API の今後のバージョンで削除される予定です。

12.1.4. .spec.agent.ebpf

説明

ebpf は、**spec.agent.type** が **eBPF** に設定されている場合、eBPF ベースのフローレポーターに関連する設定を示します。

型

object

プロパティ	型	説明
-------	---	----

プロパティ	型	説明
advanced	object	advanced を使用すると、eBPF エージェントの内部設定のいくつかの側面を設定できます。このセクションは、 GOGC や GOMAXPROCS 環境変数などのデバッグと詳細なパフォーマンスの最適化を主な目的としています。これらの値はお客様の責任のもと設定してください。
cacheActiveTimeout	string	cacheActiveTimeout は、レポーターがフローを集約して送信するまでの最大期間です。 cacheMaxFlows と cacheActiveTimeout を増やすと、ネットワークトラフィックのオーバーヘッドと CPU 負荷を減らすことができますが、メモリー消費量が増え、フローコレクションのレイテンシーが増加することが予想されます。
cacheMaxFlows	integer	cacheMaxFlows は、集約内のフローの最大数です。到達すると、レポーターはフローを送信します。 cacheMaxFlows と cacheActiveTimeout を増やすと、ネットワークトラフィックのオーバーヘッドと CPU 負荷を減らすことができますが、メモリー消費量が増え、フローコレクションのレイテンシーが増加することが予想されます。
excludeInterfaces	array (string)	excludeInterfaces には、フロートレースから除外するインターフェイス名を含めます。 /br- など、スラッシュで囲まれたエントリは正規表現として照合されます。それ以外は、大文字と小文字を区別する文字列として照合されます。

プロパティ	型	説明
features	array (string)	<p>有効にする追加機能のリスト。これらはデフォルトですべて無効になっています。追加機能を有効にすると、パフォーマンスに影響が出る可能性があります。可能な値は次のとおりです。</p> <ul style="list-style-type: none"> - PacketDrop: パケットドロップフローのロギング機能を有効にします。この機能を使用する場合はカーネルデバッグファイルシステムをマウントする必要があるため、eBPF Pod は特権付きとして実行する必要があります。spec.agent.ebpf.privileged パラメーターが設定されていない場合は、エラーが報告されません。 - DNSTracking: DNS 追跡機能を有効にします。 - FlowRTT: eBPF エージェントで TCP トラフィックからのフロー遅延 (sRTT) 抽出を有効にします。
flowFilter	object	flowFilter は、フローフィルタリングに関する eBPF エージェント設定を定義します。
imagePullPolicy	string	imagePullPolicy は、上で定義したイメージの Kubernetes プルポリシーです。
interfaces	array (string)	interfaces には、フローの収集元であるインターフェイスの名前を含めます。空の場合、エージェントは excludeInterfaces にリストされているものを除いて、システム内のすべてのインターフェイスを取得します。 /br- など、スラッシュで囲まれたエントリは正規表現として照合されます。それ以外は、大文字と小文字を区別する文字列として照合されます。

プロパティ	型	説明
kafkaBatchSize	integer	kafkaBatchSize は、パーティションに送信される前のリクエストの最大サイズをバイト単位で制限します。Kafka を使用しない場合は無視されます。デフォルト:1 MB。
logLevel	string	logLevel は、Network Observability eBPF Agent のログレベルを定義します。
metrics	object	metrics は、メトリクスに関する eBPF エージェント設定を定義します。
privileged	boolean	eBPF Agent コンテナの特権モード。無視されるか false に設定されていると、Operator がコンテナに詳細なキャパビリティ (BPF、PERFMON、NET_ADMIN、SYS_RESOURCE) を設定します。CAP_BPF を認識しない古いカーネルバージョンが使用されている場合など、何らかの理由でこれらの機能を設定できない場合は、このモードをオンにして、より多くのグローバル権限を取得できます。パケットドロップの追跡 (機能を参照) や SR-IOV サポートなど、一部のエージェント機能には特権モードが必要です。
resources	object	resources は、このコンテナが必要とするコンピューティングリソースです。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。
sampling	integer	フローレポーターのサンプリングレート。100 は、100 の1つのフローが送信されることを意味します。0 または 1 は、すべてのフローがサンプリングされることを意味します。

12.1.5. .spec.agent.ebpf.advanced

説明

advanced を使用すると、eBPF エージェントの内部設定のいくつかの側面を設定できます。このセクションは、**GOGC** や **GOMAXPROCS** 環境変数などのデバッグと詳細なパフォーマンスの最適化を主な目的としています。これらの値はお客様の責任のもと設定してください。

型

object

プロパティ	型	説明
env	object (string)	env を使用すると、カスタム環境変数を基礎となるコンポーネントに渡すことができます。 GOGC や GOMAXPROCS など、非常に具体的なパフォーマンスチューニングオプションを渡すのに役立ちます。これらのオプションは、エッジのデバッグ時かサポートを受ける場合にのみ有用なものであるため、FlowCollector 記述子の一部として公開しないでください。
scheduling	object	scheduling は、Pod がノードでどのようにスケジュールされるかを制御します。

12.1.6. .spec.agent.ebpf.advanced.scheduling

説明

scheduling は、Pod がノードでどのようにスケジュールされるかを制御します。

型

object

プロパティ	型	説明
affinity	object	指定した場合、Pod のスケジューリング制約。ドキュメントについては、 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling を参照してください。

プロパティ	型	説明
nodeSelector	object (string)	nodeSelector を使用すると、指定した各ラベルを持つノードにのみ Pod をスケジュールできます。ドキュメントについては、 https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ を参照してください。
priorityClassName	string	指定した場合、Pod の優先度を示します。ドキュメントについては、 https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption を参照してください。指定されていない場合はデフォルトの優先度が使用され、デフォルトの優先度がない場合は 0 が使用されます。
tolerations	array	tolerations は、一致するテイントを持つノードに Pod がスケジュールできるようにする容認のリストです。ドキュメントについては、 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling を参照してください。

12.1.7. .spec.agent.ebpf.advanced.scheduling.affinity

説明

指定した場合、Pod のスケジューリング制約。ドキュメントについては、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

object

12.1.8. .spec.agent.ebpf.advanced.scheduling.tolerations

説明

tolerations は、一致するテイントを持つノードに Pod がスケジュールできるようにする容認のリストです。ドキュメントについては、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

array

12.1.9. .spec.agent.ebpf.flowFilter

説明

flowFilter は、フローフィルタリングに関する eBPF エージェント設定を定義します。

型

object

プロパティ	型	説明
action	string	action は、フィルターに一致するフローに対して実行するアクションを定義します。
cidr	string	cidr は、フローのフィルタリングに使用する IP CIDR を定義します。たとえば、 10.10.10.0/24 または 100:100:100:100::/64 です。
destPorts	integer-or-string	destPorts は、フローのフィルタリングに使用する宛先ポートを定義します。単一のポートをフィルタリングするには、単一のポートを整数値として設定します。たとえば、 destPorts: 80 です。ポートの範囲をフィルタリングするには、文字列形式の "開始 - 終了" 範囲を使用します。たとえば、 destPorts: "80-100" です。
direction	string	direction は、フローのフィルタリングに使用する方向を定義します。
enable	boolean	eBPF フローのフィルタリング機能を有効にするには、 enable を true に設定します。
icmpCode	integer	icmpCode は、Internet Control Message Protocol (ICMP) トラフィックの場合に、フローのフィルタリングに使用する ICMP コードを定義します。

プロパティ	型	説明
icmpType	integer	icmpType は、ICMP トラフィックの場合に、フローのフィルタリングに使用する ICMP タイプを定義します。
peerIP	string	peerIP は、フローのフィルタリングに使用する IP アドレスを定義します。たとえば、 10.10.10.10 です。
ports	integer-or-string	ports は、フローのフィルタリングに使用するポートを定義します。これは送信元ポートと宛先ポートの両方に使用されます。単一のポートをフィルタリングするには、単一のポートを整数値として設定します。たとえば、 ports: 80 です。ポートの範囲をフィルタリングするには、文字列形式の "開始 - 終了" 範囲を使用します。たとえば、 ports: "80-100" です。
protocol	string	protocol は、フローのフィルタリングに使用するプロトコルを定義します。
sourcePorts	integer-or-string	sourcePorts は、フローのフィルタリングに使用する送信元ポートを定義します。単一のポートをフィルタリングするには、単一のポートを整数値として設定します。たとえば、 sourcePorts: 80 です。ポートの範囲をフィルタリングするには、文字列形式の "開始 - 終了" 範囲を使用します。たとえば、 sourcePorts: "80-100" です。

12.1.10. .spec.agent.ebpf.metrics

説明

metrics は、メトリクスに関する eBPF エージェント設定を定義します。

型

object

プロパティ	型	説明
disableAlerts	array (string)	disableAlerts は、無効にする必要があるアラートのリストです。可能な値は次のとおりです。 NetObservDroppedFlows は、BPF の HashMap がいっぱいになったときや容量制限がトリガーされたときなど、eBPF エージェントがフローをドロップしているときにトリガーされます。
enable	boolean	eBPF エージェントのメトリクス収集を無効にするには、 enable を false に設定します。これは、デフォルトで有効になっています。
server	object	Prometheus スクレイパーのメトリクスサーバーエンドポイント設定。

12.1.11. .spec.agent.ebpf.metrics.server

説明

Prometheus スクレイパーのメトリクスサーバーエンドポイント設定。

型

object

プロパティ	型	説明
port	integer	メトリクスサーバーの HTTP ポート。
tls	object	TLS 設定。

12.1.12. .spec.agent.ebpf.metrics.server.tls

説明

TLS 設定。

型

object

プロパティ	型	説明
<code>insecureSkipVerify</code>	<code>boolean</code>	<code>insecureSkipVerify</code> を使用すると、提供された証明書に対するクライアント側の検証をスキップできます。 <code>true</code> に設定すると、 <code>providedCaFile</code> フィールドが無視されます。
<code>provided</code>	<code>object</code>	<code>type</code> が <code>Provided</code> に設定されている場合の TLS 設定。
<code>providedCaFile</code>	<code>object</code>	<code>type</code> が <code>Provided</code> に設定されている場合の CA ファイルへの参照。
<code>type</code>	<code>string</code>	TLS 設定のタイプを選択します。 - Disabled (デフォルト) は、エンドポイントに TLS を設定しません。- Provided は、証明書ファイルとキーファイルを手動で指定します [サポート対象外 (*)]。- Auto は、アノテーションを使用して OpenShift Container Platform の自動生成証明書を使用します。

12.1.13. .spec.agent.ebpf.metrics.server.tls.provided

説明

`type` が `Provided` に設定されている場合の TLS 設定。

型

`object`

プロパティ	型	説明
<code>certFile</code>	<code>string</code>	<code>certFile</code> は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
<code>certKey</code>	<code>string</code>	<code>certKey</code> は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。

プロパティ	型	説明
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.14. .spec.agent.ebpf.metrics.server.tls.providedCaFile

説明

type が **Provided** に設定されている場合の CA ファイルへの参照。

型

object

プロパティ	型	説明
file	string	config map またはシークレット内のファイル名。
name	string	ファイルを含む config map またはシークレットの名前。
namespace	string	ファイルを含む config map またはシークレットの namespace。省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	ファイル参照のタイプ ("configmap" または "secret")。

12.1.15. .spec.agent.ebpf.resources

説明

resources は、このコンテナが必要とするコンピューティングリソースです。詳細は、<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/> を参照してください。

型

object

プロパティ	型	説明
limits	integer-or-string	limits は、許可されるコンピュートリソースの最大量を示します。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。
requests	integer-or-string	requests は、必要なコンピュートリソースの最小量を示します。コンテナについて Requests が省略される場合、明示的に指定される場合にデフォルトで Limits に設定されます。指定しない場合は、実装定義の値に設定されます。リクエストは制限を超えることはできません。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。

12.1.16. .spec.consolePlugin

説明

consolePlugin は、利用可能な場合、OpenShift Container Platform コンソールプラグインに関連する設定を定義します。

型

object

プロパティ	型	説明
advanced	object	advanced を使用すると、コンソールプラグインの内部設定のいくつかの側面を設定できます。このセクションは、 GOGC や GOMAXPROCS 環境変数などのデバッグと詳細なパフォーマンスの最適化を主な目的としています。これらの値はお客様の責任のもと設定してください。

プロパティ	型	説明
autoscaler	object	プラグインのデプロイメント用に設定する水平 Pod オートスケーラーの autoscaler 仕様。 HorizontalPodAutoscaler のドキュメント (自動スケーリング/v2) を参照してください。
enable	boolean	コンソールプラグインのデプロイメントを有効にします。
imagePullPolicy	string	imagePullPolicy は、上で定義したイメージの Kubernetes プルポリシーです。
logLevel	string	コンソールプラグインバックエンドの logLevel 。
portNaming	object	portNaming は、ポートからサービス名への変換の設定を定義します。
quickFilters	array	quickFilters は、コンソールプラグインのクイックフィルタープリセットを設定します。
replicas	integer	replicas は、開始するレプリカ (Pod) の数を定義します。
resources	object	resources (コンピューティングリソースから見た場合にコンテナーに必要)。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。

12.1.17. .spec.consolePlugin.advanced

説明

advanced を使用すると、コンソールプラグインの内部設定のいくつかの側面を設定できます。このセクションは、**GOGC** や **GOMAXPROCS** 環境変数などのデバッグと詳細なパフォーマンスの最適化を主な目的としています。これらの値はお客様の責任のもと設定してください。

型

object

プロパティ	型	説明
args	array (string)	args を使用すると、カスタム引数を基礎となるコンポーネントに渡すことができます。URL や設定パスなど、一部のパラメーターをオーバーライドする場合に有用です。これらのパラメーターは、エッジのデバッグ時かサポートを受ける場合にのみ有用なものであるため、FlowCollector 記述子の一部として公開しないでください。
env	object (string)	env を使用すると、カスタム環境変数を基礎となるコンポーネントに渡すことができます。 GOGC や GOMAXPROCS など、非常に具体的なパフォーマンスチューニングオプションを渡すのに役立ちます。これらのオプションは、エッジのデバッグ時かサポートを受ける場合にのみ有用なものであるため、FlowCollector 記述子の一部として公開しないでください。
port	integer	port はプラグインサービスポートです。メトリクス用に予約されている 9002 は使用しないでください。
register	boolean	register を true に設定すると、提供されたコンソールプラグインを OpenShift Container Platform Console Operator に自動的に登録できます。 false に設定した場合でも、 oc patch console.operator.openshift.io cluster --type=json -p '{"op": "add", "path": "/spec/plugins/-", "value": "netobserv-plugin"}' コマンドで console.operator.openshift.io/cluster を編集することにより、手動で登録できます。
scheduling	object	scheduling は、Pod がノードでどのようにスケジューラれるかを制御します。

12.1.18. .spec.consolePlugin.advanced.scheduling

説明

scheduling は、Pod がノードでどのようにスケジュールされるかを制御します。

型

object

プロパティ	型	説明
affinity	object	指定した場合、Pod のスケジューリング制約。ドキュメントについては、 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling を参照してください。
nodeSelector	object (string)	nodeSelector を使用すると、指定した各ラベルを持つノードにのみ Pod をスケジュールできます。ドキュメントについては、 https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ を参照してください。
priorityClassName	string	指定した場合、Pod の優先度を示します。ドキュメントについては、 https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption を参照してください。指定されていない場合はデフォルトの優先度が使用され、デフォルトの優先度がない場合は 0 が使用されます。
tolerations	array	tolerations は、一致するティントを持つノードに Pod がスケジュールできるようにする容認のリストです。ドキュメントについては、 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling を参照してください。

12.1.19. .spec.consolePlugin.advanced.scheduling.affinity

説明

指定した場合、Pod のスケジューリング制約。ドキュメントについては、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

object

12.1.20. .spec.consolePlugin.advanced.scheduling.tolerations

説明

tolerations は、一致するテイントを持つノードに Pod がスケジュールできるようにする容認のリストです。ドキュメントについては、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

array

12.1.21. .spec.consolePlugin.autoscaler

説明

プラグインのデプロイメント用に設定する水平 Pod オートスケーラーの **autoscaler** 仕様。HorizontalPodAutoscaler のドキュメント (自動スケーリング/v2) を参照してください。

型

object

12.1.22. .spec.consolePlugin.portNaming

説明

portNaming は、ポートからサービス名への変換の設定を定義します。

型

object

プロパティ	型	説明
enable	boolean	コンソールプラグインのポートからサービス名への変換を有効にします。
portNames	object (string)	portNames は、コンソールで使用する追加のポート名を定義します (例: portNames: {"3100": "loki"})。

12.1.23. .spec.consolePlugin.quickFilters

説明

quickFilters は、コンソールプラグインのクイックフィルタープリセットを設定します。

型

array

12.1.24. .spec.consolePlugin.quickFilters[]

説明

QuickFilter は、コンソールのクイックフィルターのプリセット設定を定義します。

型

object

必須

- **filter**
- **name**

プロパティ	型	説明
default	boolean	default は、このフィルターをデフォルトで有効にするかどうかを定義します。
filter	object (string)	filter は、このフィルターが選択されたときに設定されるキーと値のセットです。各キーは、コンマ区切りの文字列を使用して値のリストに関連付けることができます (例: filter: {"src_namespace": "namespace1,namespace2"})。
name	string	コンソールに表示されるフィルターの名前

12.1.25. .spec.consolePlugin.resources

説明

resources (コンピューティングリソースから見た場合にコンテナに必要)。詳細は、<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/> を参照してください。

型

object

プロパティ	型	説明
limits	integer-or-string	limits は、許可されるコンピューティングリソースの最大量を示します。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。

プロパティ	型	説明
requests	integer-or-string	requests は、必要なコンピューティングリソースの最小量を示します。コンテナについて Requests が省略される場合、明示的に指定される場合にデフォルトで Limits に設定されます。指定しない場合は、実装定義の値に設定されます。リクエストは制限を超えることはできません。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。

12.1.26. .spec.exporters

説明

exporters は、カスタム消費またはストレージ用の追加のオプションのエクスポートを定義します。

型

array

12.1.27. .spec.exporters[]

説明

FlowCollectorExporter は、強化されたフローを送信する追加のエクスポートを定義します。

型

object

必須

- **type**

プロパティ	型	説明
ipfix	object	強化された IPFIX フローの送信先となる、IP アドレスやポートなどの IPFIX 設定。
kafka	object	強化されたフローの送信先となる、アドレスやトピックなどの Kafka 設定。
type	string	type は、エクスポートのタイプを選択します。使用可能なオプションは Kafka と IPFIX です。

12.1.28. .spec.exporters[].ipfix

説明

強化された IPFIX フローの送信先となる、IP アドレスやポートなどの IPFIX 設定。

型

object

必須

- **targetHost**
- **targetPort**

プロパティ	型	説明
targetHost	string	IPFIX 外部レシーバーのアドレス
targetPort	integer	IPFIX 外部レシーバー用のポート
transport	string	IPFIX 接続に使用されるトランスポートプロトコル (TCP または UDP)。デフォルトは TCP です。

12.1.29. .spec.exporters[].kafka

説明

強化されたフローの送信先となる、アドレスやトピックなどの Kafka 設定。

型

object

必須

- **address**
- **topic**

プロパティ	型	説明
address	string	Kafka サーバーのアドレス
sasl	object	SASL 認証の設定。[サポート対象外 (*)]。
tls	object	TLS クライアント設定。TLS を使用する場合は、アドレスが TLS に使用される Kafka ポート (通常は 9093) と一致することを確認します。

プロパティ	型	説明
topic	string	使用する Kafka トピック。これは必ず存在する必要があります。Network Observability はこれを作成しません。

12.1.30. .spec.exporters[].kafka.sasl

説明

SASL 認証の設定。[サポート対象外 (*)]。

型

object

プロパティ	型	説明
clientIDReference	object	クライアント ID を含むシークレットまたは config map への参照
clientSecretReference	object	クライアントシークレットを含むシークレットまたは config map への参照
type	string	使用する SASL 認証のタイプ。SASL を使用しない場合は Disabled

12.1.31. .spec.exporters[].kafka.sasl.clientIDReference

説明

クライアント ID を含むシークレットまたは config map への参照

型

object

プロパティ	型	説明
file	string	config map またはシークレット内のファイル名。
name	string	ファイルを含む config map またはシークレットの名前。

プロパティ	型	説明
namespace	string	ファイルを含む config map またはシークレットの namespace。省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	ファイル参照のタイプ ("configmap" または "secret")。

12.1.32. .spec.exporters[].kafka.sasl.clientSecretReference

説明

クライアントシークレットを含むシークレットまたは config map への参照

型

object

プロパティ	型	説明
file	string	config map またはシークレット内のファイル名。
name	string	ファイルを含む config map またはシークレットの名前。
namespace	string	ファイルを含む config map またはシークレットの namespace。省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	ファイル参照のタイプ ("configmap" または "secret")。

12.1.33. .spec.exporters[].kafka.tls

説明

TLS クライアント設定。TLS を使用する場合は、アドレスが TLS に使用される Kafka ポート (通常は 9093) と一致することを確認します。

型

object

プロパティ	型	説明
caCert	object	caCert は、認証局の証明書の参照を定義します。
enable	boolean	TLS を有効にします。
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、サーバー証明書のクライアント側の検証をスキップできます。 true に設定すると、 caCert フィールドが無視されます。
userCert	object	userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

12.1.34. .spec.exporters[].kafka.tls.caCert

説明

caCert は、認証局の証明書の参照を定義します。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。

プロパティ	型	説明
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.35. .spec.exporters[.].kafka.tls.userCert

説明

userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。

プロパティ	型	説明
type	string	証明書参照のタイプ (configmap または secret)。

12.1.36. .spec.kafka

説明

Kafka 設定。Kafka をフローコレクションパイプラインの一部としてブローカーとして使用できます。この設定を利用できるのは、**spec.deploymentModel** が **Kafka** の場合です。

型

object

必須

- **address**
- **topic**

プロパティ	型	説明
address	string	Kafka サーバーのアドレス
sasl	object	SASL 認証の設定。[サポート対象外 (*)]。
tls	object	TLS クライアント設定。TLS を使用する場合は、アドレスが TLS に使用される Kafka ポート (通常は 9093) と一致することを確認します。
topic	string	使用する Kafka トピック。これは必ず存在する必要があります。Network Observability はこれを作成しません。

12.1.37. .spec.kafka.sasl

説明

SASL 認証の設定。[サポート対象外 (*)]。

型

object

プロパティ	型	説明
clientIDReference	object	クライアント ID を含むシークレットまたは config map への参照
clientSecretReference	object	クライアントシークレットを含むシークレットまたは config map への参照
type	string	使用する SASL 認証のタイプ。 SASL を使用しない場合は Disabled

12.1.38. .spec.kafka.sasl.clientIDReference

説明

クライアント ID を含むシークレットまたは config map への参照

型

object

プロパティ	型	説明
file	string	config map またはシークレット内のファイル名。
name	string	ファイルを含む config map またはシークレットの名前。
namespace	string	ファイルを含む config map またはシークレットの namespace。 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	ファイル参照のタイプ ("configmap" または "secret")。

12.1.39. .spec.kafka.sasl.clientSecretReference

説明

クライアントシークレットを含むシークレットまたは config map への参照

型

object

プロパティ	型	説明
file	string	config map またはシークレット内のファイル名。
name	string	ファイルを含む config map またはシークレットの名前。
namespace	string	ファイルを含む config map またはシークレットの namespace。省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	ファイル参照のタイプ ("configmap" または "secret")。

12.1.40. .spec.kafka.tls

説明

TLS クライアント設定。TLS を使用する場合は、アドレスが TLS に使用される Kafka ポート (通常は 9093) と一致することを確認します。

型

object

プロパティ	型	説明
caCert	object	caCert は、認証局の証明書の参照を定義します。
enable	boolean	TLS を有効にします。
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、サーバー証明書のクライアント側の検証をスキップできます。 true に設定すると、 caCert フィールドが無視されます。

プロパティ	型	説明
userCert	object	userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

12.1.41. .spec.kafka.tls.caCert

説明

caCert は、認証局の証明書の参照を定義します。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.42. .spec.kafka.tls.userCert

説明

userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.43. .spec.loki

説明

loki (フローストア) のクライアント設定。

型

object

プロパティ	型	説明
advanced	object	advanced を使用すると、Loki クライアントの内部設定のいくつかの側面を設定できます。このセクションは、デバッグと詳細なパフォーマンスの最適化を主な目的としています。

プロパティ	型	説明
enable	boolean	Loki にフローを保存するには、 enable を true に設定します。コンソールプラグインは、メトリクスのデータソースとして Loki または Prometheus、またはその両方を使用できます (spec.prometheus.queries も参照してください)。すべてのクエリーを Loki から Prometheus に転送できるわけではありません。したがって、Loki が無効になっている場合、Pod ごとの情報の取得や raw フローの表示など、プラグインの一部の機能も無効になります。Prometheus と Loki の両方が有効になっている場合は、Prometheus が優先され、Prometheus が処理できないクエリーのフォールバックとして Loki が使用されます。両方とも無効になっている場合、コンソールプラグインはデプロイされません。
lokiStack	object	LokiStack モードの Loki 設定。これは、Loki Operator を簡単に設定するのに役立ちます。他のモードでは無視されます。
manual	object	Manual モードの Loki 設定。これは最も柔軟な設定です。他のモードでは無視されます。
microservices	object	Microservices モードの Loki 設定。このオプションは、Loki がマイクロサービスデプロイメントモード (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode) を使用してインストールされている場合に使用します。他のモードでは無視されます。

プロパティ	型	説明
mode	string	<p>mode は、Loki のインストールモードに応じて設定する必要があります。</p> <ul style="list-style-type: none"> - Loki が Loki Operator を使用して管理されている場合は、LokiStack を使用します。 - Loki がモノリシックなワークロードとしてインストールされている場合は、Monolithic を使用します。 - Loki がマイクロサービスとしてインストールされているが、Loki Operator がない場合は、Microservices を使用します。 - 上記のオプションが、いずれもお使いの構成に合わない場合は、Manual を使用します。
monolithic	object	<p>Monolithic モードの Loki 設定。このオプションは、Loki がモノリシックデプロイメントモード (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode) を使用してインストールされている場合に使用します。他のモードでは無視されます。</p>
readTimeout	string	<p>readTimeout は、コンソールプラグインの loki クエリーの合計時間上限です。タイムアウトがゼロの場合は、タイムアウトしません。</p>
writeBatchSize	integer	<p>writeBatchSize は、送信前に蓄積する Loki ログの最大バッチサイズ (バイト単位) です。</p>
writeBatchWait	string	<p>writeBatchWait は、Loki バッチを送信するまでに待機する最大時間です。</p>

プロパティ	型	説明
writeTimeout	string	writeTimeout は、Loki の接続/リクエスト時間の上限です。タイムアウトがゼロの場合は、タイムアウトしません。

12.1.44. .spec.loki.advanced

説明

advanced を使用すると、Loki クライアントの内部設定のいくつかの側面を設定できます。このセクションは、デバッグと詳細なパフォーマンスの最適化を主な目的としています。

型

object

プロパティ	型	説明
staticLabels	object (string)	staticLabels は、Loki ストレージ内の各フローに設定する共通ラベルのマッピングです。
writeMaxBackoff	string	writeMaxBackoff は、Loki クライアント接続の再試行間の最大バックオフ時間です。
writeMaxRetries	integer	writeMaxRetries は、Loki クライアント接続の最大再試行回数です。
writeMinBackoff	string	writeMinBackoff は、Loki クライアント接続の再試行間の初期バックオフ時間です。

12.1.45. .spec.loki.lokiStack

説明

LokiStack モードの Loki 設定。これは、Loki Operator を簡単に設定するのに役立ちます。他のモードでは無視されます。

型

object

プロパティ	型	説明
name	string	使用する既存の LokiStack リソースの名前。

プロパティ	型	説明
namespace	string	この LokiStack リソースが配置される namespace。省略した場合は、 spec.namespace と同じであるとみなされます。

12.1.46. .spec.loki.manual

説明

Manual モードの Loki 設定。これは最も柔軟な設定です。他のモードでは無視されます。

型

object

プロパティ	型	説明
authToken	string	<p>authToken は、Loki に対して認証するためのトークンを取得する方法を示します。</p> <ul style="list-style-type: none"> - Disabled の場合、リクエストとともにトークンが送信されません。 - Forward の場合、認可のためにユーザートークンを転送します。 - Host [非推奨 (*)] - ローカル Pod サービスアカウントを使用して Loki に認証します。 <p>Loki Operator を使用する場合、Forward に設定する必要があります。</p>
ingesterUrl	string	<p>ingesterUrl は、フローのプッシュ先となる既存の Loki インジェスターサービスのアドレスです。Loki Operator を使用する場合は、パスに network テナントが設定された Loki ゲートウェイサービスに設定します (例: https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network)。</p>

プロパティ	型	説明
querierUrl	string	querierUrl は、Loki クエリアーサービスのアドレスを指定します。Loki Operator を使用する場合は、パスに network テナントが設定された Loki ゲートウェイサービスに設定します (例: https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network)。
statusTls	object	Loki ステータス URL の TLS クライアント設定。
statusUrl	string	statusUrl は、Loki クエリアー URL と異なる場合に備えて、Loki /ready 、 /metrics 、 /config エンドポイントのアドレスを指定します。空の場合、 querierUrl の値が使用されます。これは、フロントエンドでエラーメッセージやコンテキストを表示するのに便利です。Loki Operator を使用する場合は、Loki HTTP クエリーフロントエンドサービス (例: https://loki-query-frontend-http.netobserv.svc:3100/) に設定します。 statusTls 設定は、 statusUrl が設定されている場合に使用されます。
tenantID	string	tenantID は、各リクエストのテナントを識別する Loki X-Scope-OrgID です。Loki Operator を使用する場合は、特別なテナントモードに対応する network に設定します。
tls	object	Loki URL の TLS クライアント設定。

12.1.47. .spec.loki.manual.statusTls

説明

Loki ステータス URL の TLS クライアント設定。

型

object

プロパティ	型	説明
caCert	object	caCert は、認証局の証明書の参照を定義します。
enable	boolean	TLS を有効にします。
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、サーバー証明書のクライアント側の検証をスキップできます。 true に設定すると、 caCert フィールドが無視されます。
userCert	object	userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

12.1.48. .spec.loki.manual.statusTls.caCert

説明

caCert は、認証局の証明書の参照を定義します。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。

プロパティ	型	説明
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.49. .spec.loki.manual.statusTls.userCert

説明

userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。

プロパティ	型	説明
type	string	証明書参照のタイプ (configmap または secret)。

12.1.50. .spec.loki.manual.tls

説明

Loki URL の TLS クライアント設定。

型

object

プロパティ	型	説明
caCert	object	caCert は、認証局の証明書の参照を定義します。
enable	boolean	TLS を有効にします。
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、サーバー証明書のクライアント側の検証をスキップできます。 true に設定すると、 caCert フィールドが無視されます。
userCert	object	userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

12.1.51. .spec.loki.manual.tls.caCert

説明

caCert は、認証局の証明書の参照を定義します。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。

プロパティ	型	説明
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.52. .spec.loki.manual.tls.userCert

説明

userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。

プロパティ	型	説明
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.53. .spec.loki.microservices

説明

Microservices モードの Loki 設定。このオプションは、Loki がマイクロサービスデプロイメントモード (<https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode>) を使用してインストールされている場合に使用します。他のモードでは無視されます。

型

object

プロパティ	型	説明
ingesterUrl	string	ingesterUrl は、フローのプッシュ先となる既存の Loki インジェスターサービスのアドレスです。
querierUrl	string	querierURL は、Loki クエリアーサービスのアドレスを指定します。
tenantID	string	tenantID は、各リクエストのテナントを識別する Loki X-Scope-OrgID ヘッダーです。
tls	object	Loki URL の TLS クライアント設定。

12.1.54. .spec.loki.microservices.tls

説明

Loki URL の TLS クライアント設定。

型

object

プロパティ	型	説明
caCert	object	caCert は、認証局の証明書の参照を定義します。
enable	boolean	TLS を有効にします。
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、サーバー証明書のクライアント側の検証をスキップできます。 true に設定すると、 caCert フィールドが無視されます。
userCert	object	userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

12.1.55. .spec.loki.microservices.tls.caCert

説明

caCert は、認証局の証明書の参照を定義します。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。

プロパティ	型	説明
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.56. .spec.loki.microservices.tls.userCert

説明

userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。

プロパティ	型	説明
type	string	証明書参照のタイプ (configmap または secret)。

12.1.57. .spec.loki.monolithic

説明

Monolithic モードの Loki 設定。このオプションは、Loki がモノリシックデプロイメントモード (<https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode>) を使用してインストールされている場合に使用します。他のモードでは無視されます。

型

object

プロパティ	型	説明
tenantID	string	tenantID は、各リクエストのテナントを識別する Loki X-Scope-OrgID ヘッダーです。
tls	object	Loki URL の TLS クライアント設定。
url	string	url は、インジェスターとクエリアーの両方を参照する既存の Loki サービスの一意のアドレスです。

12.1.58. .spec.loki.monolithic.tls

説明

Loki URL の TLS クライアント設定。

型

object

プロパティ	型	説明
caCert	object	caCert は、認証局の証明書の参照を定義します。
enable	boolean	TLS を有効にします。

プロパティ	型	説明
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、サーバー証明書のクライアント側の検証をスキップできます。 true に設定すると、 caCert フィールドが無視されます。
userCert	object	userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

12.1.59. .spec.loki.monolithic.tls.caCert

説明

caCert は、認証局の証明書の参照を定義します。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.60. .spec.loki.monolithic.tls.userCert

説明

userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace。省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.61. .spec.processor

説明

processor は、エージェントからフローを受信し、それを強化し、メトリクスを生成し、Loki 永続化レイヤーや使用可能なエクスポーターに転送するコンポーネントの設定を定義します。

型

object

プロパティ	型	説明
-------	---	----

プロパティ	型	説明
addZone	boolean	addZone は、フローに送信元ゾーンと宛先ゾーンのラベルを付けることで、アベイラビリティゾーンを認識できるようにします。この機能を使用するには、ノードに "topology.kubernetes.io/zone" ラベルを設定する必要があります。
advanced	object	advanced を使用すると、フロープロセッサの内部設定のいくつかの側面を設定できます。このセクションは、 GOGC や GOMAXPROCS 環境変数などのデバッグと詳細なパフォーマンスの最適化を主な目的としています。これらの値はお客様の責任のもと設定してください。
clusterName	string	clusterName は、フローデータに表示されるクラスターの名前です。これは、マルチクラスターコンテキストで役立ちます。OpenShift Container Platform を使用する場合は、自動的に決定されるように空のままにします。
imagePullPolicy	string	imagePullPolicy は、上で定義したイメージの Kubernetes プルポリシーです。
kafkaConsumerAutoscaler	object	kafkaConsumerAutoscaler は、Kafka メッセージを消費する flowlogs-pipeline-transformer を設定する水平 Pod オートスケーラーの仕様です。Kafka が無効になっている場合、この設定は無視されます。HorizontalPodAutoscaler のドキュメント (自動スケーリング/v2) を参照してください。
kafkaConsumerBatchSize	integer	kafkaConsumerBatchSize は、コンシューマーが受け入れる最大バッチサイズ (バイト単位) をブローカーに示します。Kafka を使用しない場合は無視されます。デフォルト: 10MB。

プロパティ	型	説明
kafkaConsumerQueueCapacity	integer	kafkaConsumerQueueCapacity は、Kafka コンシューマークライアントで使用される内部メッセージキューの容量を定義します。Kafka を使用しない場合は無視されます。
kafkaConsumerReplicas	integer	kafkaConsumerReplicas は、Kafka メッセージを消費する flowlogs-pipeline-transformer に対して開始するレプリカ (Pod) の数を定義します。Kafka が無効になっている場合、この設定は無視されます。
logLevel	string	プロセッサランタイムの logLevel
logTypes	string	<p>logTypes は、生成するレコードタイプを定義します。可能な値は次のとおりです。</p> <ul style="list-style-type: none"> - Flows (デフォルト) は、通常のネットワークフローをエクスポートします。 - Conversations は、開始した会話、終了した会話、および定期的な "ティック" 更新のイベントを生成します。 - EndedConversations は、終了した会話イベントのみを生成します。 - All は、ネットワークフローとすべての会話イベントの両方を生成します。
metrics	object	Metric は、メトリクスに関するプロセッサ設定を定義します。
multiClusterDeployment	boolean	マルチクラスター機能を有効にするには、 multiClusterDeployment を true に設定します。これにより、 clusterName ラベルがフローデータに追加されます。

プロパティ	型	説明
resources	object	resources は、このコンテナが必要とするコンピューティングリソースです。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。
subnetLabels	object	subnetLabels を使用すると、サブネットと IP にカスタムラベルを定義したり、OpenShift Container Platform で認識されているサブネットの自動ラベル付けを有効にしたりできます。自動ラベル付けは、クラスタの外部トラフィックを識別するために使用されます。サブネットがフローの送信元 IP または宛先 IP と一致する場合、対応するフィールド SrcSubnetLabel または DstSubnetLabel が追加されます。

12.1.62. .spec.processor.advanced

説明

advanced を使用すると、フロープロセッサの内部設定のいくつかの側面を設定できます。このセクションは、**GOGC** や **GOMAXPROCS** 環境変数などのデバッグと詳細なパフォーマンスの最適化を主な目的としています。これらの値はお客様の責任のもと設定してください。

型

object

プロパティ	型	説明
conversationEndTimeout	string	conversationEndTimeout は、ネットワークフローを受信した後、対話が終了したとみなされるまでの待機時間です。TCP フローの FIN パケットが収集される場合、この遅延は無視されます (代わりに、 conversationTerminatingTimeout を使用します)。
conversationHeartbeatInterval	string	conversationHeartbeatInterval は、対話の "tick" イベント間の待機時間です。

プロパティ	型	説明
conversationTerminatingTimeout	string	conversationTerminatingTimeout 、FIN フラグが検知されてから対話が終了するまでの待機時間です。TCP フローにのみ関連します。
dropUnusedFields	boolean	dropUnusedFields [非推奨 (*)] この設定は、現在使用されていません。
enableKubeProbes	boolean	enableKubeProbes は、Kubernetes の liveness および readiness プローブを有効または無効にするフラグです。
env	object (string)	env を使用すると、カスタム環境変数を基礎となるコンポーネントに渡すことができます。 GOGC や GOMAXPROCS など、非常に具体的なパフォーマンスチューニングオプションを渡すのに役立ちます。これらのオプションは、エッジのデバッグ時かサポートを受ける場合にのみ有用なものであるため、FlowCollector 記述子の一部として公開しないでください。
healthPort	integer	healthPort は、ヘルスチェック API を公開する Pod のコレクター HTTP ポートです。
port	integer	フローコレクターのポート (ホストポート)。慣例により、一部の値は禁止されています。1024 より大きい値とし、4500、4789、6081 は使用できません。
profilePort	integer	profilePort を使用すると、このポートをリスンする Go pprof プロファイラーを設定できます
scheduling	object	scheduling は、Pod がノードでどのようにスケジュールされるかを制御します。

12.1.63. .spec.processor.advanced.scheduling

説明

scheduling は、Pod がノードでどのようにスケジュールされるかを制御します。

型

object

プロパティ	型	説明
affinity	object	指定した場合、Pod のスケジューリング制約。ドキュメントについては、 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling を参照してください。
nodeSelector	object (string)	nodeSelector を使用すると、指定した各ラベルを持つノードのみ Pod をスケジュールできます。ドキュメントについては、 https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ を参照してください。
priorityClassName	string	指定した場合、Pod の優先度を示します。ドキュメントについては、 https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption を参照してください。指定されていない場合はデフォルトの優先度が使用され、デフォルトの優先度がない場合は 0 が使用されます。
tolerations	array	tolerations は、一致するテイントを持つノードに Pod がスケジュールできるようにする容認のリストです。ドキュメントについては、 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling を参照してください。

12.1.64. .spec.processor.advanced.scheduling.affinity

説明

指定した場合、Pod のスケジューリング制約。ドキュメントについては、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

object

12.1.65. .spec.processor.advanced.scheduling.tolerations

説明

tolerations は、一致するテイントを持つノードに Pod がスケジュールできるようにする容認のリストです。ドキュメントについては、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

array

12.1.66. .spec.processor.kafkaConsumerAutoscaler

説明

kafkaConsumerAutoscaler は、Kafka メッセージを消費する **flowlogs-pipeline-transformer** を設定する水平 Pod オートスケーラーの仕様です。Kafka が無効になっている場合、この設定は無視されます。HorizontalPodAutoscaler のドキュメント (自動スケーリング/v2) を参照してください。

型

object

12.1.67. .spec.processor.metrics

説明

Metric は、メトリクスに関するプロセッサ設定を定義します。

型

object

プロパティ	型	説明
disableAlerts	array (string)	<p>disableAlerts は、無効にする必要があるアラートのリストです。可能な値は次のとおりです。</p> <p>NetObservNoFlows は、一定期間フローが観測されない場合にトリガーされます。</p> <p>NetObservLokiError: Loki エラーが原因でフローがドロップされるとトリガーされます。</p>

プロパティ	型	説明
includeList	array (string)	<p>includeList は、生成するメトリクスを指定するためのメトリクス名のリストです。名前は、接頭辞を除いた Prometheus の名前に対応します。たとえば、namespace_egress_packets_total は、Prometheus では netobserv_namespace_egress_packets_total と表示されます。メトリクスを追加するほど、Prometheus ワークロードリソースへの影響が大きくなることに注意してください。デフォルトで有効なメトリクスは、namespace_flows_total、node_ingress_bytes_total、workload_ingress_bytes_total、namespace_drop_packets_total (PacketDrop 機能が有効な場合)、namespace_rtt_seconds (FlowRTT 機能が有効な場合)、namespace_dns_latency_seconds (DNSTracking 機能が有効な場合) です。利用可能なメトリクスの完全なリストを含む詳細情報は、https://github.com/netobserv/network-observability-operator/blob/main/docs/Metrics.md を参照してください。</p>
server	object	Prometheus スクレイパーのメトリクスサーバーエンドポイント設定

12.1.68. .spec.processor.metrics.server

説明

Prometheus スクレイパーのメトリクスサーバーエンドポイント設定

型

object

プロパティ	型	説明
port	integer	メトリクスサーバーの HTTP ポート。

プロパティ	型	説明
tls	object	TLS 設定。

12.1.69. .spec.processor.metrics.server.tls

説明

TLS 設定。

型

object

プロパティ	型	説明
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、提供された証明書に対するクライアント側の検証をスキップできます。 true に設定すると、 providedCaFile フィールドが無視されます。
provided	object	type が Provided に設定されている場合の TLS 設定。
providedCaFile	object	type が Provided に設定されている場合の CA ファイルへの参照。
type	string	TLS 設定のタイプを選択します。 <ul style="list-style-type: none"> - Disabled (デフォルト) は、エンドポイントに TLS を設定しません。 - Provided は、証明書ファイルとキーファイルを手動で指定します [サポート対象外 (*)]。 - Auto は、アノテーションを使用して OpenShift Container Platform の自動生成証明書を使用します。

12.1.70. .spec.processor.metrics.server.tls.provided

説明

type が **Provided** に設定されている場合の TLS 設定。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.71. .spec.processor.metrics.server.tls.providedCaFile

説明

type が **Provided** に設定されている場合の CA ファイルへの参照。

型

object

プロパティ	型	説明
file	string	config map またはシークレット内のファイル名。
name	string	ファイルを含む config map またはシークレットの名前。

プロパティ	型	説明
namespace	string	ファイルを含む config map またはシークレットの namespace。省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	ファイル参照のタイプ ("configmap" または "secret")。

12.1.72. .spec.processor.resources

説明

resources は、このコンテナが必要とするコンピューティングリソースです。詳細は、<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/> を参照してください。

型

object

プロパティ	型	説明
limits	integer-or-string	limits は、許可されるコンピューティングリソースの最大量を示します。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。
requests	integer-or-string	requests は、必要なコンピューティングリソースの最小量を示します。コンテナについて Requests が省略される場合、明示的に指定される場合にデフォルトで Limits に設定されます。指定しない場合は、実装定義の値に設定されます。リクエストは制限を超えることはできません。詳細は、 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/ を参照してください。

12.1.73. .spec.processor.subnetLabels

説明

subnetLabels を使用すると、サブネットと IP にカスタムラベルを定義したり、OpenShift Container Platform で認識されているサブネットの自動ラベル付けを有効にしたりできます。自動ラベル付けは、クラスターの外部トラフィックを識別するために使用されます。サブネットがフローの送信元 IP または宛先 IP と一致する場合、対応するフィールド **SrcSubnetLabel** または **DstSubnetLabel** が追加されます。

型

object

プロパティ	型	説明
customLabels	array	customLabels を使用すると、クラスター外部のワークロードや Web サービスの識別などのために、サブネットと IP のラベル付けをカスタマイズできます。 openShiftAutoDetect を有効にすると、検出されたサブネットがオーバーラップしている場合に、 customLabels がそのサブネットをオーバーライドできます。
openShiftAutoDetect	boolean	openShiftAutoDetect を true に設定すると、OpenShift Container Platform のインストール設定と Cluster Network Operator の設定に基づいて、マシン、Pod、およびサービスのサブネットを自動的に検出できます。これは間接的に外部トラフィックを正確に検出する方法です。つまり、これらのサブネットのラベルが付いていないフローは、クラスターの外部のもので、OpenShift Container Platform ではデフォルトで有効になっています。

12.1.74. .spec.processor.subnetLabels.customLabels

説明

customLabels を使用すると、クラスター外部のワークロードや Web サービスの識別などのために、サブネットと IP のラベル付けをカスタマイズできます。**openShiftAutoDetect** を有効にすると、検出されたサブネットがオーバーラップしている場合に、**customLabels** がそのサブネットをオーバーライドできます。

型

array

12.1.75. .spec.processor.subnetLabels.customLabels[]

説明

SubnetLabel を使用すると、クラスター外部のワークロードや Web サービスの識別などのために、サブネットと IP にラベルを付けることができます。

型

object

プロパティ	型	説明
cidrs	array (string)	["1.2.3.4/32"] などの CIDR のリスト。
name	string	一致するフローにフラグを設定するために使用するラベル名。

12.1.76. .spec.prometheus

説明

prometheus は、コンソールプラグインからメトリクスを取得するために使用されるクエリー設定などの Prometheus 設定を定義します。

型

object

プロパティ	型	説明
querier	object	コンソールプラグインで使用される、クライアント設定などの Prometheus クエリー設定。

12.1.77. .spec.prometheus.querier

説明

コンソールプラグインで使用される、クライアント設定などの Prometheus クエリー設定。

型

object

プロパティ	型	説明
-------	---	----

プロパティ	型	説明
enable	boolean	enable が true の場合、コンソールプラグインは、可能な場合は常に、Loki ではなく Prometheus からフローメトリクスをクエリーします。デフォルトでは有効になっています。この機能を無効にするには false に設定します。コンソールプラグインは、メトリクスのデータソースとして Loki または Prometheus、またはその両方を使用できます (spec.loki も参照してください)。すべてのクエリーを Loki から Prometheus に転送できるわけではありません。したがって、Loki が無効になっている場合、Pod ごとの情報の取得や raw フローの表示など、プラグインの一部の機能も無効になります。Prometheus と Loki の両方が有効になっている場合は、Prometheus が優先され、Prometheus が処理できないクエリーのフォールバックとして Loki が使用されます。両方とも無効になっている場合、コンソールプラグインはデプロイされません。
manual	object	Manual モードの Prometheus 設定。
mode	string	mode は、Network Observability メトリクスを保存する Prometheus インストールのタイプに応じて設定する必要があります。 - 自動設定を試行するには、 Auto を使用します。OpenShift Container Platform では、OpenShift Container Platform クラスタモニタリングの Thanos クエリーを使用します。 - 手動設定の場合は、 Manual を使用します。
timeout	string	timeout は、Prometheus へのコンソールプラグインクエリーの読み取りタイムアウトです。タイムアウトがゼロの場合は、タイムアウトしません。

12.1.78. .spec.prometheus.querier.manual

説明

Manual モードの Prometheus 設定。

型

object

プロパティ	型	説明
forwardUserToken	boolean	ログインしたユーザートークンをクエリーで Prometheus に転送するには、 true に設定します。
tls	object	Prometheus URL の TLS クライアント設定。
url	string	url は、メトリクスのクエリーに使用する既存の Prometheus サービスのアドレスです。

12.1.79. .spec.prometheus.querier.manual.tls

説明

Prometheus URL の TLS クライアント設定。

型

object

プロパティ	型	説明
caCert	object	caCert は、認証局の証明書の参照を定義します。
enable	boolean	TLS を有効にします。
insecureSkipVerify	boolean	insecureSkipVerify を使用すると、サーバー証明書のクライアント側の検証をスキップできます。 true に設定すると、 caCert フィールドが無視されます。
userCert	object	userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

12.1.80. .spec.prometheus.querier.manual.tls.caCert

説明

caCert は、認証局の証明書の参照を定義します。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

12.1.81. .spec.prometheus.querier.manual.tls.userCert

説明

userCert は、mTLS に使用されるユーザー証明書参照を定義します (一方向 TLS を使用する場合は無視できます)。

型

object

プロパティ	型	説明
certFile	string	certFile は、config map またはシークレット内の証明書ファイル名へのパスを定義します。

プロパティ	型	説明
certKey	string	certKey は、config map またはシークレット内の証明書秘密鍵ファイル名へのパスを定義します。キーが不要な場合は省略します。
name	string	証明書を含む config map またはシークレットの名前。
namespace	string	証明書を含む config map またはシークレットの namespace 省略した場合、デフォルトでは、Network Observability がデプロイされているのと同じ namespace が使用されます。namespace が異なる場合は、必要に応じてマウントできるように、config map またはシークレットがコピーされます。
type	string	証明書参照のタイプ (configmap または secret)。

第13章 FLOWMETRIC 設定パラメーター

FlowMetric は、収集されたフローログからカスタムメトリクスを作成することを可能にする API です。

13.1. FLOWMETRIC [FLOWS.NETOBSERV.IO/V1ALPHA1]

説明

FlowMetric は、収集されたフローログからカスタムメトリクスを作成することを可能にする API です。

型

object

プロパティ	型	説明
apiVersion	string	APIVersion はオブジェクトのこの表現のバージョンスキーマを定義します。サーバーは認識されたスキーマを最新の内部値に変換し、認識されない値は拒否することがあります。詳細は、 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources を参照してください。
kind	string	kind はこのオブジェクトが表す REST リソースを表す文字列の値です。サーバーは、クライアントが要求を送信するエンドポイントからこれを推測できることがあります。これを更新することはできません。キャメルケースを使用します。詳細は、 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds を参照してください。
metadata	object	標準オブジェクトのメタデータ。詳細は、 https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata を参照してください。

プロパティ	型	説明
spec	object	<p>FlowMetricSpec は、FlowMetric の目的の状態を定義します。提供されている API を使用すると、ニーズに応じてこれらのメトリクスをカスタマイズできます。</p> <p>新しいメトリクスを追加したり、既存のラベルを変更したりする場合は、大きな影響を与える可能性があります。そのため、Prometheus ワークロードのメモリ使用量を注意深く監視する必要があります。 https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric を参照してください。</p> <p>すべての Network Observability メトリクスのカーディナリティーを確認するには、promql: count({name=~"netobserv.*"}) by (name) を実行します。</p>

13.1.1. .metadata

説明

標準オブジェクトのメタデータ。詳細は、 <https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata> を参照してください。

型

object

13.1.2. .spec

説明

FlowMetricSpec は、FlowMetric の目的の状態を定義します。提供されている API を使用すると、ニーズに応じてこれらのメトリクスをカスタマイズできます。

新しいメトリクスを追加したり、既存のラベルを変更したりする場合は、大きな影響を与える可能性があります。そのため、Prometheus ワークロードのメモリ使用量を注意深く監視する必要があります。 <https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric> を参照してください。

すべての Network Observability メトリクスのカーディナリティーを確認するには、**promql: count({name=~"netobserv.*"}) by (name)** を実行します。

型

object

必須

- **metricName**
- **type**

プロパティ	型	説明
buckets	array (string)	type が "Histogram" の場合に使用するバケットのリスト。このリストは、浮動小数点数として解析可能である必要があります。設定されていない場合は、Prometheus のデフォルトのバケットが使用されます。
charts	array	管理者ビューの Dashboards メニューにある OpenShift Container Platform コンソールのチャート設定。
direction	string	Ingress、Egress、または任意の方向のフローをフィルタリングします。 Ingress に設定すると、 FlowDirection に正規表現フィルター 0 2 を追加した場合と同じになります。 Egress に設定すると、 FlowDirection に正規表現フィルター 1 2 を追加した場合と同じになります。
divider	string	ゼロ以外の場合、値の換算係数 (除数)。メトリクス値 = フロー値 / 除数。
filters	array	filters は、考慮されるフローを制限するために使用するフィールドと値のリストです。多くの場合、重複を排除するために、 Duplicate != "true" および FlowDirection = "0" というフィルターを使用する必要があります。使用可能なフィールドのリストについては、ドキュメント https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html を参照してください。

プロパティ	型	説明
labels	array (string)	<p>labels は、Prometheus ラベル (ディメンションとも呼ばれます) として使用するフィールドのリストです。ラベルを選択すると、このメトリクスの粒度レベルと、クエリー時に使用可能な集計が決定されます。これは、メトリクスのカーディナリティーに影響するため、慎重に行う必要があります (https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric を参照)。一般に、IP アドレスや MAC アドレスなど、カーディナリティーが非常に高いラベルを設定することは避けてください。可能な限り、"SrcK8S_OwnerName" または "DstK8S_OwnerName" を、"SrcK8S_Name" または "DstK8S_Name" よりも優先してください。使用可能なフィールドのリストについては、ドキュメント https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html を参照してください。</p>
metricName	string	<p>メトリクスの名前。Prometheus では、自動的に "netobserv_" という接頭辞が付けられます。</p>
type	string	<p>メトリクスタイプ ("Counter" または "Histogram")。"Counter" は、バイト数やパケット数など、時間の経過とともに増加し、レートを計算できる値に使用します。"Histogram" は、遅延など、個別にサンプリングする必要がある値に使用します。</p>

プロパティ	型	説明
valueField	string	valueField は、このメトリクスの値として使用する必要のあるフローフィールドです。このフィールドには数値を入力する必要があります。フロー数をカウントするには、フローごとに特定の値を指定するのではなく、空のままにします。使用可能なフィールドのリストについては、ドキュメント https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html を参照してください。

13.1.3. .spec.charts

説明

管理者ビューの Dashboards メニューにある OpenShift Container Platform コンソールのチャート設定。

型

array

13.1.4. .spec.charts[]

説明

メトリクスに関連するグラフ/ダッシュボード生成を設定します。

型

object

必須

- **dashboardName**
- **queries**
- **title**
- **type**

プロパティ	型	説明
dashboardName	string	配置先のダッシュボードの名前。この名前が既存のダッシュボードを示すものでない場合は、新しいダッシュボードが作成されます。

プロパティ	型	説明
queries	array	このグラフに表示するクエリーのリスト。 type が SingleStat で、複数のクエリーが指定されている場合、このグラフは複数のパネル (クエリーごとに1つ) に自動的に展開されます。
sectionName	string	配置先のダッシュボードセクションの名前。この名前が既存のセクションを示すものでない場合は、新しいセクションが作成されます。 sectionName が省略されているか空の場合、グラフはグローバルトップセクションに配置されます。
title	string	グラフのタイトル。
type	string	グラフの種類。
unit	string	このグラフの単位。現在サポートされている単位はごく少数です。一般的な数字を使用する場合は空白のままにします。

13.1.5. .spec.charts[].queries

説明

このグラフに表示するクエリーのリスト。**type** が **SingleStat** で、複数のクエリーが指定されている場合、このグラフは複数のパネル (クエリーごとに1つ) に自動的に展開されます。

型

array

13.1.6. .spec.charts[].queries[]

説明

PromQL クエリーを設定します。

型

object

必須

- **legend**
- **promQL**
- **top**

プロパティ	型	説明
legend	string	このグラフに表示する各時系列に適用するクエリーの凡例。複数の時系列を表示する場合は、それぞれを区別する凡例を設定する必要があります。これは {{ Label }} という形式で設定できます。たとえば、 promQL でラベルごとに時系列をグループ化する場合 (例: sum(rate(\$METRIC[2m])) by (Label1, Label2))、凡例として Label1={{ Label1 }} 、 Label2={{ Label2 }} と記述します。
promQL	string	Prometheus に対して実行する promQL クエリー。グラフの type が SingleStat の場合、このクエリーは単一の時系列のみを返します。その他のタイプの場合、上位7つが表示されます。このリソースで定義されたメトリクスを参照するには、 \$METRIC を使用できます。たとえば、 sum(rate(\$METRIC[2m])) です。 promQL の詳細は、Prometheus のドキュメント https://prometheus.io/docs/prometheus/latest/querying/basics/ を参照してください。
top	integer	タイムスタンプごとに表示する上位 N 個の系列。 SingleStat グラフタイプには適用されません。

13.1.7. .spec.filters

説明

filters は、考慮されるフローを制限するために使用するフィールドと値のリストです。多くの場合、重複を排除するために、**Duplicate != "true"** および **FlowDirection = "0"** というフィルターを使用する必要があります。使用可能なフィールドのリストについては、ドキュメント https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html を参照してください。

型

array

13.1.8. .spec.filters[]

説明

型

object

必須

- **field**
- **matchType**

プロパティ	型	説明
field	string	フィルタリングするフィールドの名前
matchType	string	適用するマッチングのタイプ
value	string	フィルタリングする値。 matchType が Equal または NotEqual の場合、 \$(SomeField) を使用したフィールドインジェクションを使用して、フロー内の他のフィールドを参照できます。

第14章 ネットワークフロー形式のリファレンス

これらはネットワークフロー形式の仕様であり、内部で使用され、フローを Kafka にエクスポートする場合にも使用されます。

14.1. ネットワークフロー形式のリファレンス

これはネットワークフロー形式の仕様です。この形式は、Prometheus メトリクスラベルに、および内部で Loki ストアに Kafka エクスポーターが設定されているときに使用されます。

"フィルター ID" 列は、クイックフィルターを定義するときに使用する関連名を示します (**FlowCollector** 仕様の **spec.consolePlugin.quickFilters** を参照)。

"Loki ラベル" 列は、Loki に直接クエリーを実行する場合に役立ちます。ラベルフィールドは、[stream selectors](#) を使用して選択する必要があります。

"カーディナリティー" 列は、このフィールドが **FlowMetric** API で Prometheus ラベルとして使用される場合の暗黙のメトリクスカーディナリティーに関する情報を示します。詳細は、「FlowMetric API リファレンス」を参照してください。

Name	型	説明	フィルター ID	Loki ラベル	カーディナリティー
Bytes	number	バイト数	該当なし	いいえ	avoid
DnsErrno	number	DNS トラッカーの ebpf フック関数から返されたエラー番号	dns_errno	いいえ	fine
DnsFlags	number	DNS レコードの DNS フラグ	該当なし	いいえ	fine
DnsFlagsResponseCode	string	解析された DNS ヘッダーの RCODEs 名	dns_flag_response_code	いいえ	fine
DnsId	number	DNS レコード id	dns_id	いいえ	avoid
DnsLatencyMs	number	DNS リクエストとレスポンスの間の時間 (ミリ秒単位)	dns_latency	いいえ	avoid
Dscp	number	Differentiated Services Code Point (DSCP) の値	dscp	いいえ	fine
DstAddr	string	宛先 IP アドレス (ipv4 または ipv6)	dst_address	いいえ	avoid
DstK8S_HostIP	string	送信先ノード IP	dst_host_address	いいえ	fine

Name	型	説明	フィルター ID	Loki ラベル	カーディナリティー
DstK8S_HostName	string	送信先ノード名	dst_host_name	いいえ	fine
DstK8S_Name	string	宛先 Kubernetes オブジェクトの名前 (Pod 名、Service 名、Node 名など)。	dst_name	いいえ	careful
DstK8S_Namespace	string	宛先 namespace	dst_namespace	はい	fine
DstK8S_OwnerName	string	宛先所有者の名前 (Deployment 名、StatefulSet 名など)。	dst_owner_name	はい	fine
DstK8S_OwnerType	string	宛先所有者の種類 (Deployment、StatefulSet など)。	dst_kind	いいえ	fine
DstK8S_Type	string	宛先 Kubernetes オブジェクトの種類 (Pod、Service、Node など)。	dst_kind	はい	fine
DstK8S_Zone	string	宛先アベイラビリティゾーン	dst_zone	はい	fine
DstMac	string	宛先 MAC アドレス	dst_mac	いいえ	avoid
DstPort	number	送信先ポート	dst_port	いいえ	careful
DstSubnetLabel	string	宛先サブネットラベル	dst_subnet_label	いいえ	fine
Duplicate	boolean	このフローが同じホスト上の別のインターフェイスからもキャプチャーされたかどうかを示します。	該当なし	はい	fine

Name	型	説明	フィルター ID	Loki ラベル	カーディナリティー
Flags	number	RFC-9293 に基づく、フローに含まれる一意の TCP フラグと追加カスタムフラグの論理和の組み合わせ。次のパケット別の組み合わせを表します。 - SYN+ACK (0x100) - FIN+ACK (0x200) - RST+ACK (0x400)	該当なし	いいえ	fine
FlowDirection	number	ノード観測点から解釈されたフローの方向。次のいずれかになります。 - 0: Ingress (ノード観測点からの受信トラフィック) - 1: Egress (ノード観測点からの送信トラフィック) - 2: Inner (送信元ノードと宛先ノードが同じ)	node_direction	はい	fine
IcmpCode	number	ICMP コード	icmp_code	いいえ	fine
IcmpType	number	ICMP のタイプ	icmp_type	いいえ	fine
IfDirections	number	ネットワークインターフェイス観測点からのフローの方向。次のいずれかになります。 - 0: Ingress (インターフェイスの受信トラフィック) - 1: Egress (インターフェイスの送信トラフィック)	ifdirections	いいえ	fine
インターフェイス	string	ネットワークインターフェイス	interfaces	いいえ	careful
K8S_ClusterName	string	クラスター名またはクラスター識別子	cluster_name	はい	fine
K8S_FlowLayer	string	フローのレイヤー: 'app' または 'infra'	flow_layer	いいえ	fine
パケット	number	パケット数	該当なし	いいえ	avoid
PktDropBytes	number	カーネルによってドロップされたバイト数	該当なし	いいえ	avoid

Name	型	説明	フィルター ID	Loki ラベル	カーディナリティー
PktDrop LatestDropCause	string	最新のドロップの原因	pkt_drop_cause	いいえ	fine
PktDrop LatestFlags	number	最後にドロップされたパケットの TCP フラグ	該当なし	いいえ	fine
PktDrop LatestState	string	最後にドロップされたパケットの TCP 状態	pkt_drop_state	いいえ	fine
PktDrop Packets	number	カーネルによってドロップされたパケットの数	該当なし	いいえ	avoid
Proto	number	L4 プロトコル	protocol	いいえ	fine
SrcAddr	string	送信元 IP アドレス (ipv4 または ipv6)	src_address	いいえ	avoid
SrcK8S_HostIP	string	送信元ノード IP	src_host_address	いいえ	fine
SrcK8S_HostName	string	送信元ノード名	src_host_name	いいえ	fine
SrcK8S_Name	string	送信元 Kubernetes オブジェクトの名前 (Pod 名、サービス名、ノード名など)。	src_name	いいえ	careful
SrcK8S_Namespace	string	リソースの namespace。	src_namespace	はい	fine
SrcK8S_OwnerName	string	送信元所有者の名前 (Deployment 名、StatefulSet 名など)。	src_owner_name	はい	fine
SrcK8S_OwnerType	string	送信元所有者の種類 (Deployment、StatefulSet など)。	src_kind	いいえ	fine

Name	型	説明	フィルター ID	Loki ラベル	カーディナリティー
SrcK8S_Type	string	送信元 Kubernetes オブジェクトの種類 (Pod、Service、Node など)。	src_kind	はい	fine
SrcK8S_Zone	string	送信元アベイラビリティゾーン	src_zone	はい	fine
SrcMac	string	送信元 MAC アドレス	src_mac	いいえ	avoid
SrcPort	number	送信元ポート	src_port	いいえ	careful
SrcSubnetLabel	string	送信元サブネットラベル	src_subnet_label	いいえ	fine
TimeFlowEndMs	number	このフローの終了タイムスタンプ (ミリ秒単位)	該当なし	いいえ	avoid
TimeFlowRttNs	number	TCP の平滑化されたラウンドトリップタイム (SRTT) (ナノ秒単位)	time_flow_rtt	いいえ	avoid
TimeFlowStartMs	number	このフローの開始タイムスタンプ (ミリ秒単位)	該当なし	いいえ	avoid
TimeReceived	number	このフローがフローコレクターによって受信および処理されたときのタイムスタンプ (秒単位)	該当なし	いいえ	avoid
_HashId	string	会話追跡では、会話識別子	id	いいえ	avoid
_RecordType	string	レコードの種類: 通常のフローログの場合は 'flowLog'、会話追跡の場合は 'newConnection'、'heartbeat'、'endConnection'	type	はい	fine

第15章 NETWORK OBSERVABILITY のトラブルシューティング

Network Observability の問題のトラブルシューティングを支援するために、いくつかのトラブルシューティングアクションを実行できます。

15.1. MUST-GATHER ツールの使用

must-gather ツールを使用すると、Pod ログ、**FlowCollector**、**Webhook** 設定などの、Network Observability Operator リソースおよびクラスター全体のリソースに関する情報を収集できます。

手順

1. must-gather データを保存するディレクトリーに移動します。
2. 次のコマンドを実行して、クラスター全体の must-gather リソースを収集します。

```
$ oc adm must-gather
--image-stream=openshift/must-gather \
--image=quay.io/netobserv/must-gather
```

15.2. OPENSIFT CONTAINER PLATFORM コンソールでのネットワークトラフィックメニューエントリーの設定

OpenShift Container Platform コンソールの 監視 メニューにネットワークトラフィックのメニューエントリーがリストされていない場合は、OpenShift Container Platform コンソールでネットワークトラフィックのメニューエントリーを手動で設定します。

前提条件

- OpenShift Container Platform バージョン 4.10 以降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、**spec.consolePlugin.register** フィールドが **true** に設定されているかどうかを確認します。

```
$ oc -n netobserv get flowcollector cluster -o yaml
```

出力例

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  consolePlugin:
    register: false
```

2. オプション: Console Operator 設定を手動で編集して、**netobserv-plugin** プラグインを追加します。

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

出力例

```
...
spec:
  plugins:
  - netobserv-plugin
...
```

3. オプション: 次のコマンドを実行して、**spec.consolePlugin.register** フィールドを **true** に設定します。

```
$ oc -n netobserv edit flowcollector cluster -o yaml
```

出力例

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  consolePlugin:
    register: true
```

4. 次のコマンドを実行して、コンソール Pod のステータスが **running** であることを確認します。

```
$ oc get pods -n openshift-console -l app=console
```

5. 次のコマンドを実行して、コンソール Pod を再起動します。

```
$ oc delete pods -n openshift-console -l app=console
```

6. ブラウザーのキャッシュと履歴をクリアします。

7. 次のコマンドを実行して、Network Observability プラグイン Pod のステータスを確認します。

```
$ oc get pods -n netobserv -l app=netobserv-plugin
```

出力例

```
NAME                                READY STATUS RESTARTS AGE
netobserv-plugin-68c7bbb9bb-b69q6  1/1   Running  0      21s
```

8. 次のコマンドを実行して、Network Observability プラグイン Pod のログを確認します。

```
$ oc logs -n netobserv -l app=netobserv-plugin
```

出力例

```
time="2022-12-13T12:06:49Z" level=info msg="Starting netobserv-console-plugin [build
version: , build date: 2022-10-21 15:15] at log level info" module=main
time="2022-12-13T12:06:49Z" level=info msg="listening on https://:9001" module=server
```

15.3. KAFKA をインストールした後、FLOWLOGS-PIPELINE がネットワークフローを消費しない

最初に **deploymentModel: KAFKA** を使用してフローコレクターをデプロイし、次に Kafka をデプロイした場合、フローコレクターが Kafka に正しく接続されない可能性があります。Flowlogs-pipeline が Kafka からのネットワークフローを消費しないフローパイプライン Pod を手動で再起動します。

手順

1. 次のコマンドを実行して、flow-pipeline Pod を削除して再起動します。

```
$ oc delete pods -n netobserv -l app=flowlogs-pipeline-transformer
```

15.4. BR-INT インターフェイスと BR-EX インターフェイスの両方からのネットワークフローが表示されない

br-ex` と **br-int** は、OSI レイヤー 2 で動作する仮想ブリッジデバイスです。eBPF エージェントは、IP レベルと TCP レベル、それぞれレイヤー 3 と 4 で動作します。ネットワークトラフィックが物理ホストや仮想 Pod インターフェイスなどの他のインターフェイスによって処理される場合、eBPF エージェントは **br-ex** および **br-int** を通過するネットワークトラフィックをキャプチャすることが期待できます。eBPF エージェントのネットワークインターフェイスを **br-ex** および **br-int** のみに接続するように制限すると、ネットワークフローは表示されません。

ネットワークインターフェイスを **br-int** および **br-ex** に制限する **interfaces** または **excludeInterfaces** の部分を手動で削除します。

手順

1. **interfaces: ['br-int', 'br-ex']** フィールド。これにより、エージェントはすべてのインターフェイスから情報を取得できます。または、レイヤー 3 インターフェイス (例: **eth0**) を指定することもできます。以下のコマンドを実行します。

```
$ oc edit -n netobserv flowcollector.yaml -o yaml
```

出力例

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
    type: EBPF
    ebpf:
      interfaces: [ 'br-int', 'br-ex' ] ❶
```

- ❶ ネットワークインターフェイスを指定します。

15.5. NETWORK OBSERVABILITY コントローラーマネージャー POD のメモリーが不足する

Subscription オブジェクトの **spec.config.resources.limits.memory** 仕様を編集することで、Network Observability Operator のメモリー制限を引き上げることができます。

手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **Network Observability** をクリックし、**Subscription** を選択します。
3. **Actions** メニューから、**Edit Subscription** をクリックします。
 - a. または、CLI を使用して次のコマンドを実行して、**Subscription** オブジェクトの YAML 設定を開くこともできます。

```
$ oc edit subscription netobserv-operator -n openshift-netobserv-operator
```

4. **Subscription** オブジェクトを編集して **config.resources.limits.memory** 仕様を追加し、メモリー要件を考慮して値を設定します。リソースに関する考慮事項の詳細は、関連情報を参照してください。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: netobserv-operator
  namespace: openshift-netobserv-operator
spec:
  channel: stable
  config:
    resources:
      limits:
        memory: 800Mi ①
      requests:
        cpu: 100m
        memory: 100Mi
  installPlanApproval: Automatic
  name: netobserv-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: <network_observability_operator_latest_version> ②
```

- ① たとえば、メモリー制限を **800 Mi** に引き上げることができます。
- ② この値は編集しないでください。この値は Operator の最新リリースによって異なります。

15.6. LOKI へのカスタムクエリーの実行

トラブルシューティングのために、Loki に対してカスタムクエリーを実行できます。これを行う方法の例が 2 つあり、<api_token> を独自のものに置き換えることで、ニーズに合わせて調整できます。



注記

これらの例では、Network Observability Operator および Loki デプロイメントに **netobserv** namespace を使用します。さらに、例では、LokiStack の名前が **loki** であると想定しています。オプションで、例 (具体的には **-n netobserv** または **loki-gateway** URL) を調整して、異なる namespace と命名を使用することもできます。

前提条件

- Network Observability Operator で使用するために Loki Operator をインストールしている

手順

- 利用可能なすべてのラベルを取得するには、次のコマンドを実行します。

```
$ oc exec deployment/netobserv-plugin -n netobserv -- curl -G -s -H 'X-Scope-
OrgID:network' -H 'Authorization: Bearer <api_token>' -k https://loki-gateway-
http.netobserv.svc:8080/api/logs/v1/network/loki/api/v1/labels | jq
```

- ソース namespace **my-namespace** からすべてのフローを取得するには、次のコマンドを実行します。

```
$ oc exec deployment/netobserv-plugin -n netobserv -- curl -G -s -H 'X-Scope-
OrgID:network' -H 'Authorization: Bearer <api_token>' -k https://loki-gateway-
http.netobserv.svc:8080/api/logs/v1/network/loki/api/v1/query --data-urlencode 'query=
{SrcK8S_Namespace="my-namespace"}' | jq
```

関連情報

- [リソースの留意事項](#)

15.7. LOKI RESOURCEEXHAUSTED エラーのトラブルシューティング

Network Observability によって送信されたネットワークフローデータが、設定された最大メッセージサイズを超えると、Loki は **ResourceExhausted** エラーを返すことがあります。Red Hat Loki Operator を使用している場合、この最大メッセージサイズは 100 MiB に設定されています。

手順

1. **Operators** → **Installed Operators** に移動し、**Project** ドロップダウンメニューから **All projects** を表示します。
2. **Provided APIs** リストで、Network Observability Operator を選択します。
3. **Flow Collector** をクリックし、**YAML view** タブをクリックします。
 - a. Loki Operator を使用している場合は、**spec.loki.batchSize** 値が 98 MiB を超えていないことを確認してください。
 - b. Red Hat Loki Operator とは異なる Loki インストール方法 (Grafana Loki など) を使用している場合は、[Grafana Loki サーバー設定](#) の **grpc_server_max_recv_msg_size** が、**FlowCollector** リソースの **spec.loki.batchSize** 値より大きいことを確認してください。大きくない場合は、**grpc_server_max_recv_msg_size** 値を増やすか、**spec.loki.batchSize** 値を制限値よりも小さくなるように減らす必要があります。

4. FlowCollector を編集した場合は、Save をクリックします。

15.8. LOKI の EMPTY RING エラー

Loki の "empty ring" エラーにより、フローが Loki に保存されず、Web コンソールに表示されなくなります。このエラーはさまざまな状況で発生する可能性があります。これらすべてに対処できる1つの回避策はありません。Loki Pod 内のログを調査し、**LokiStack** が健全な状態で準備が整っていることを確認するために、いくつかのアクションを実行できます。

このエラーが発生する状況には次のようなものがあります。

- **LokiStack** をアンインストールし、同じ namespace に再インストールすると、古い PVC が削除されないため、このエラーが発生する可能性があります。
 - アクション: **LokiStack** を再度削除し、PVC を削除してから、**LokiStack** の再インストールをお試しください。
- 証明書のローテーション後、このエラーにより、**flowlogs-pipeline** Pod および **console-plugin** Pod との通信が妨げられる可能性があります。
 - アクション: Pod を再起動すると、接続を復元できます。

15.9. リソースのトラブルシューティング

15.10. LOKISTACK レート制限エラー

Loki テナントにレート制限が設定されていると、データが一時的に失われ、429 エラー (**Per stream rate limit exceeded (limit:xMB/sec) while attempting to ingest for stream**) が発生する可能性があります。このエラーを通知するようにアラートを設定することを検討してください。詳細は、このセクションの関連情報として記載されている「NetObserv ダッシュボードの Loki レート制限アラートの作成」を参照してください。

次に示す手順を実行して、**perStreamRateLimit** および **perStreamRateLimitBurst** 仕様で LokiStack CRD を更新できます。

手順

1. Operators → Installed Operators に移動し、Project ドロップダウンから All projects を表示します。
2. Loki Operator を見つけて、LokiStack タブを選択します。
3. YAML view を使用して LokiStack インスタンスを作成するか既存のものを編集し、**perStreamRateLimit** および **perStreamRateLimitBurst** 仕様を追加します。

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: loki
  namespace: netobserv
spec:
  limits:
    global:
      ingestion:
        perStreamRateLimit: 6
```

1

```
perStreamRateLimitBurst: 30 ②
tenants:
  mode: openshift-network
  managementState: Managed
```

- ① **perStreamRateLimit** のデフォルト値は **3** です。
- ② **perStreamRateLimitBurst** のデフォルト値は **15** です。

4. **Save** をクリックします。

検証

perStreamRateLimit および **perStreamRateLimitBurst** 仕様を更新すると、クラスター内の Pod が再起動し、429 レート制限エラーが発生しなくなります。