



# OpenShift Container Platform 4.18

## Network Observability

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







# OpenShift Container Platform 4.18 Network Observability

---

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



## Legal Notice

Copyright © 2025 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

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

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> 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<sup>®</sup> 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.

## Abstract

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。



# Table of Contents

<b>第1章 NETWORK OBSERVABILITY OPERATOR リリースノート</b>	<b>5</b>
1.1. NETWORK OBSERVABILITY OPERATOR 1.10.1 アドバイザリー	5
1.2. NETWORK OBSERVABILITY OPERATOR 1.10.1 CVE	5
1.3. NETWORK OBSERVABILITY OPERATOR 1.10.1 の修正された問題	5
1.4. NETWORK OBSERVABILITY OPERATOR 1.10 アドバイザリー	6
1.5. NETWORK OBSERVABILITY OPERATOR 1.10 の新機能と機能拡張	6
1.6. NETWORK OBSERVABILITY OPERATOR 1.10 のテクノロジープレビュー機能	7
1.7. NETWORK OBSERVABILITY OPERATOR 1.10 で削除された機能	7
1.8. NETWORK OBSERVABILITY OPERATOR 1.10 の既知の問題	8
1.9. NETWORK OBSERVABILITY OPERATOR 1.10 で修正された問題	9
<b>第2章 NETWORK OBSERVABILITY OPERATOR リリースノートのアーカイブ</b>	<b>11</b>
2.1. NETWORK OBSERVABILITY OPERATOR リリースノートのアーカイブ	11
<b>第3章 NETWORK OBSERVABILITY について</b>	<b>41</b>
3.1. NETWORK OBSERVABILITY OPERATOR	41
3.2. NETWORK OBSERVABILITY OPERATOR のオプションの依存関係	41
3.3. OPENSIFT CONTAINER PLATFORM コンソールの統合	41
3.4. NETWORK OBSERVABILITY CLI	43
<b>第4章 NETWORK OBSERVABILITY OPERATOR のインストール</b>	<b>44</b>
4.1. LOKI を使用しない NETWORK OBSERVABILITY	44
4.2. LOKI OPERATOR のインストール	45
4.3. NETWORK OBSERVABILITY OPERATOR のインストール	50
4.4. NETWORK OBSERVABILITY でのマルチテナンシーの有効化	51
4.5. FLOW COLLECTOR 設定に関する重要な考慮事項	52
4.6. KAFKA のインストール (オプション)	54
4.7. NETWORK OBSERVABILITY OPERATOR のアンインストール	54
<b>第5章 OPENSIFT CONTAINER PLATFORM の NETWORK OBSERVABILITY OPERATOR</b>	<b>56</b>
5.1. 状況の表示	56
5.2. NETWORK OBSERVABILITY OPERATOR のアーキテクチャー	57
5.3. NETWORK OBSERVABILITY OPERATOR のステータスと設定の表示	59
<b>第6章 NETWORK OBSERVABILITY OPERATOR の設定</b>	<b>60</b>
6.1. FLOWCOLLECTOR リソースの表示	60
6.2. KAFKA を使用した FLOW COLLECTOR リソースの設定	62
6.3. エンリッチされたネットワークフローデータのエクスポート	63
6.4. FLOW COLLECTOR リソースの更新	65
6.5. ネットワークフロー取り込み時のフィルタリング	65
6.6. クイックフィルターの設定	67
6.7. リソース管理およびパフォーマンスに関する考慮事項	68
<b>第7章 ネットワークポリシー</b>	<b>72</b>
7.1. FLOWCOLLECTOR カスタムリソースを使用したネットワークポリシーの設定	72
<b>第8章 ネットワークトラフィックの観測</b>	<b>74</b>
8.1. OVERVIEW ビューからのネットワークトラフィックの観測	74
8.2. TRAFFIC FLOWS ビューからのネットワークトラフィックの観測	80
8.3. トポロジビューからのネットワークトラフィックの観察	95
8.4. ネットワークトラフィックのフィルタリング	96
<b>第9章 NETWORK OBSERVABILITY アラート</b>	<b>98</b>



9.1. NETWORK OBSERVABILITY アラートについて	98
9.2. NETWORK OBSERVABILITY のアラート (テクノロジープレビュー) の有効化	99
<b>第10章 ダッシュボードとアラートでのメトリクスの使用</b>	<b>106</b>
10.1. NETWORK OBSERVABILITY メトリクスのダッシュボードの表示	106
10.2. NETWORK OBSERVABILITY メトリクス	106
10.3. アラートの作成	108
10.4. カスタムメトリクス	109
10.5. FLOWMETRIC API を使用したカスタムメトリクスの設定	109
10.6. TRAFFIC FLOWS テーブルのネストされたフィールドまたは配列フィールドからメトリクスを作成する	111
10.7. FLOWMETRIC API を使用したカスタムグラフの設定	113
10.8. FLOWMETRIC API と TCP フラグを使用した SYN フラッドの検出	115
<b>第11章 NETWORK OBSERVABILITY OPERATOR の監視</b>	<b>119</b>
11.1. 健全性ダッシュボード	119
11.2. 健全性アラート	119
11.3. 健全性情報の表示	119
11.4. NETOBSERV ダッシュボードの LOKI レート制限アラートの作成	120
11.5. EBPf エージェントアラートの使用	121
<b>第12章 リソースのスケジューリング</b>	<b>123</b>
12.1. 特定のノードにおける NETWORK OBSERVABILITY デプロイメント	123
<b>第13章 セカンダリーネットワーク</b>	<b>125</b>
13.1. 前提条件	125
13.2. SR-IOV インターフェイストラフィックの監視の設定	125
13.3. 仮想マシン (VM) のセカンダリーネットワークインターフェイスを NETWORK OBSERVABILITY 用に設定する	126
<b>第14章 NETWORK OBSERVABILITY CLI</b>	<b>129</b>
14.1. NETWORK OBSERVABILITY CLI のインストール	129
14.2. NETWORK OBSERVABILITY CLI の使用	130
14.3. NETWORK OBSERVABILITY CLI (OC NETOBSERV) リファレンス	134
<b>第15章 FLOWCOLLECTOR API リファレンス</b>	<b>142</b>
15.1. FLOWCOLLECTOR API 仕様	142
<b>第16章 FLOWMETRIC 設定パラメーター</b>	<b>223</b>
16.1. FLOWMETRIC [FLOWS.NETOBSERV.IO/V1ALPHA1]	223
<b>第17章 ネットワークフロー形式のリファレンス</b>	<b>231</b>
17.1. ネットワークフロー形式のリファレンス	231
<b>第18章 NETWORK OBSERVABILITY のトラブルシューティング</b>	<b>238</b>
18.1. MUST-GATHER ツールの使用	238
18.2. 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	238
18.3. 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	240
18.4. 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	240
18.5. 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	241
18.6. LOKI へのカスタムクエリーの実行	241
18.7. LOKI RESOURCEEXHAUSTED エラーのトラブルシューティング	242
18.8. LOKI の EMPTY RING エラー	243
18.9. リソースのトラブルシューティング	243
18.10. LOKISTACK レート制限エラー	243



18.11. 大きなクエリーを実行すると LOKI エラーが発生する	244
------------------------------------	-----







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

## 1.1. NETWORK OBSERVABILITY OPERATOR 1.10.1 アドバイザリー

Network Observability Operator 1.10.1 リリースに関するアドバイザリーを確認できます。

- [RHEA-2025:22761 Network Observability Operator 1.10.1](#)

## 1.2. NETWORK OBSERVABILITY OPERATOR 1.10.1 CVE

Network Observability Operator 1.10.1 リリースの CVE を確認できます。

- [CVE-2025-47907](#)

## 1.3. NETWORK OBSERVABILITY OPERATOR 1.10.1 の修正された問題

Network Observability Operator 1.10.1 リリースには、パフォーマンスとユーザーエクスペリエンスを向上させるいくつかの修正された問題が含まれています。

### 15 ノードでのクラスターで直接生成される警告

この更新の前は、大規模なクラスターで **Direct** デプロイメントモデルを使用する場合の推奨事項は、ドキュメントでのみ利用できていました。

このリリースでは、クラスター上で Direct デプロイメントモードが使用された場合に、Network Observability Operator が警告を生成するようになりました。

[NETOBSERV-2460](#)

### OpenShiftSDN でネットワークポリシーのデプロイメントが無効化されている

この更新の前は、OpenShift SDN がクラスターネットワークプラグインであった場合、**FlowCollector** ネットワークポリシーを有効にすると、ネットワーク可観測性 Pod 間の通信が中断されていました。この問題は、サポートされているデフォルトのネットワークプラグインである OVN-Kubernetes では発生しません。

今回のリリースにより、Network Observability Operator は、OpenShift SDN の検出時にネットワークポリシーのデプロイを試みなくなりました。代わりに警告が表示されます。さらに、ネットワークポリシーを有効にするためのデフォルト値が変更され、OVN-Kubernetes がクラスターネットワークプラグインとして検出された場合にのみ、デフォルトで有効になりました。

[NETOBSERV-2450](#)

### サブネットラベル文字の検証の追加

今回の更新以前は、サブネットラベル "name" 設定で使用可能な文字に制限がなく、ユーザーはスペースまたは特殊文字を含むテキストを入力できました。これは、ユーザーがフィルターを適用しようとしたときに Web コンソールプラグインでエラーを生成し、サブネットラベルのフィルターアイコンをクリックしても失敗することがよくあります。



今回のリリースにより、設定されたサブネットラベル名は、**FlowCollector** カスタムリソースで設定された直後に検証されるようになりました。検証により、名前に英数字(:、\_、および)のみが含まれることが保証されます。その結果、Web コンソールプラグインからのサブネットラベルのフィルタリングが期待どおりに機能するようになりました。

[NETOBSERV-2438](#)

実行ごとに一意の一時ディレクトリーを使用します。

今回の更新以前は、Network Observability CLI が現在の作業ディレクトリーに1つの一時(tmp)ディレクトリーを作成または再利用していました。これにより、別の実行間で競合やデータの破損が発生する可能性があります。

このリリースでは、Network Observability CLI は実行ごとに一意の一時ディレクトリーを作成するようになり、競合の可能性を防ぎ、ファイル管理のハイジーンを改善できるようになりました。

[NETOBSERV-2481](#)

## 1.4. NETWORK OBSERVABILITY OPERATOR 1.10 アドバイザリー

Network Observability Operator 1.10 に関するアドバイザリーをご確認ください。

- [RHEA-2025:19153 Network Observability Operator 1.10](#)

## 1.5. NETWORK OBSERVABILITY OPERATOR 1.10 の新機能と機能拡張

Network Observability Operator 1.10 リリースでは、セキュリティが強化され、パフォーマンスが向上し、ネットワークフローの管理を改善するための新しい CLI UI ツールが導入されています。

### 1.5.1. ネットワークポリシーの更新

Network Observability Operator は、Pod トラフィックを制御するために、Ingress と Egress の両方のネットワークポリシーの設定をサポートするようになりました。この機能拡張によりセキュリティが向上します。

デフォルトで、**spec.NetworkPolicy.enable** 仕様が **true** に設定されるようになりました。そのため、Loki または Kafka を使用する場合は、Loki Operator と Kafka インスタンスを専用の namespace にデプロイすることを推奨します。これにより、ネットワークポリシーを正しく設定し、すべてのコンポーネント間の通信を許可することが可能になります。

### 1.5.2. Network Observability Operator CLI UI の更新

このリリースでは、Network Observability Operator CLI (**oc netobserv**) ユーザーインターフェイス(UI) に次の新機能と更新が追加されました。

#### テーブルビューの機能拡張

- カスタマイズ可能な列: **Manage Columns** をクリックして、表示する列を選択し、ニーズに合わせてテーブルをカスタマイズできます。
- スマートフィルタリング: ライブフィルターに自動提案機能が組み込まれ、適切なキーと値を選択しやすくなりました。
- パケットプレビュー: パケットをキャプチャーするときに、行をクリックして **pcap** の内容を直接調べることができます。



## ターミナルベースの折れ線グラフの機能拡張

- メトリクスの視覚化: リアルタイムグラフが CLI で直接レンダリングされます。
- パネルの選択: 事前定義済みのビューから選択するか、**Manage Panels** ポップアップメニューを使用してビューをカスタマイズし、特定のメトリクスのグラフを選択的に表示できます。

### 1.5.3. Network Observability コンソールの強化

Network Observability コンソールプラグインに、**FlowCollector** カスタムリソース (CR) を設定するための新しいビューが追加されています。このビューから、次のタスクを完了できます。

- **FlowCollector** CR を設定します。
- リソースフットプリントを計算します。
- 設定の警告やメトリクスのカーディナリティーが高いなどの問題に対する可視性を高めます。

### 1.5.4. パフォーマンスの向上

Network Observability Operator 1.10 では、特に大規模なクラスターで顕著に表れる Operator のパフォーマンスとメモリーフットプリントが改善されました。

## 1.6. NETWORK OBSERVABILITY OPERATOR 1.10 のテクノロジープレビュー機能

### 1.6.1. Network Observability Operator カスタムアラート (テクノロジープレビュー)

このリリースでは、新しいアラート機能とカスタムアラート設定が導入されています。これらの機能はテクノロジープレビュー機能であり、明示的に有効にする必要があります。

新しいアラートを表示するには、OpenShift Container Platform Web コンソールで、**Observe** → **Alerting** → **Alerting rules** をクリックします。

### 1.6.2. Network Observability Operator Network Health ダッシュボード (テクノロジープレビュー)

Network Observability Operator で、テクノロジープレビューのアラート機能を有効にする  
と、**Observe** をクリックして OpenShift Container Platform Web コンソールで新しい **Network Health** ダッシュボードを表示できます。

**Network Health** ダッシュボードは、トリガーされたアラートの概要を提供し、重大な問題、警告、および軽微な問題に分類します。また、保留中のアラートも表示します。

## 1.7. NETWORK OBSERVABILITY OPERATOR 1.10 で削除された機能

Network Observability Operator 1.10 リリースの使用に影響する可能性のある削除された機能をご確認ください。

### 1.7.1. FlowCollector API バージョン v1beta1 の削除

**FlowCollector** カスタムリソース (CR) API バージョン **v1beta1** が削除され、サポートされなくなりました。**v1beta2** バージョンを使用してください。



## 1.8. NETWORK OBSERVABILITY OPERATOR 1.10 の既知の問題

Network Observability Operator 1.10 リリースの使用に影響する可能性のある、次の既知の問題と推奨される回避策 (存在する場合) をご確認ください。

### 1.8.1. OpenShift Container Platform 4.14 以前で 1.10 へのアップグレードが失敗する

OpenShift Container Platform 4.14 以前で Network Observability Operator 1.10 にアップグレードすると、ソフトウェアカタログの **FlowCollector** カスタムリソース定義 (CRD) 検証エラーが原因で失敗する可能性があります。

この問題を回避するには、次の操作を行う必要があります。

1. OpenShift Container Platform Web コンソールのソフトウェアカタログから、両方のバージョンの Network Observability Operator をアンインストールします。
  - a. **FlowCollector** CRD は、フロー収集プロセスに中断が発生しないように、インストールしたままにしておきます。
2. 次のコマンドを実行して、**FlowCollector** CRD の現在の名前を確認します。

```
$ oc get crd flowcollectors.flows.netobserv.io -o jsonpath='{.spec.versions[0].name}'
```

想定される出力:

```
v1beta1
```

3. 次のコマンドを実行して、**FlowCollector** CRD の現在の提供ステータスを確認します。

```
$ oc get crd flowcollectors.flows.netobserv.io -o jsonpath='{.spec.versions[0].served}'
```

想定される出力:

```
true
```

4. 次のコマンドを実行して、**v1beta1** バージョンの **served** フラグを **false** に設定します。

```
$ oc patch crd flowcollectors.flows.netobserv.io --type='json' -p "[{'op': 'replace', 'path': '/spec/versions/0/served', 'value': false}]"
```

5. 次のコマンドを実行して、**served** フラグが **false** に設定されていることを確認します。

```
$ oc get crd flowcollectors.flows.netobserv.io -o jsonpath='{.spec.versions[0].served}'
```

想定される出力:

```
false
```

6. Network Observability Operator 1.10 をインストールします。

[OCPBUGS-63208](#)、[NETOBSERV-2451](#)



### 1.8.2. eBPF エージェントと OpenShift Container Platform の旧バージョンとの互換性

Network Observability コマンドラインインターフェイス (CLI) のパケットキャプチャー機能で使用する eBPF エージェントは、OpenShift Container Platform バージョン 4.16 以前と互換性がありません。

この制限により、eBPF ベースの Packet Capture Agent (PCA) が古いクラスター上で正しく機能しなくなります。

この問題を回避するには、互換性のある古い eBPF エージェントコンテナイメージを使用するように PCA を手動で設定する必要があります。詳細は、Red Hat ナレッジベースソリューション [eBPF agent compatibility with older Openshift versions in Network Observability CLI 1.10+](#) を参照してください。

NETOBSERV-2358

### 1.8.3. NetworkPolicy が有効な場合、OpenShiftSDN 環境で eBPF エージェントがフローを送信できない

**OpenShiftSDN** CNI プラグインを使用する OpenShift Container Platform 4.14 クラスターで Network Observability Operator 1.10 を実行すると、eBPF エージェントが **flowlogs-pipeline** コンポーネントにフローレコードを送信できません。これは、**NetworkPolicy** が有効な状態 (**spec.networkPolicy.enable: true**) で **FlowCollector** カスタムリソースが作成された場合に発生します。

その結果、フローデータが **flowlogs-pipeline** コンポーネントによって処理されず、**Network Traffic** ダッシュボードまたは設定されたストレージ (Loki) に表示されません。eBPF エージェント Pod のログには、コレクターへの接続を試みたときに **i/o timeout** エラーが表示されます。

```
time="2025-10-17T13:53:44Z" level=error msg="couldn't send flow records to collector"
collector="10.0.68.187:2055" component=exporter/GRPCProto error="rpc error: code = Unavailable
desc = connection error: desc = \"transport: Error while dialing: dial tcp 10.0.68.187:2055: i/o
timeout\""
```

この問題を回避するには、**spec.networkPolicy.enable** を **false** に設定して、Network Observability Operator 1.10 の **FlowCollector** リソースで **NetworkPolicy** を無効にします。

これにより、eBPF エージェントが、自動的にデプロイされたネットワークポリシーからの干渉を受けずに、**flowlogs-pipeline** コンポーネントと通信できるようになります。

NETOBSERV-2450

## 1.9. NETWORK OBSERVABILITY OPERATOR 1.10 で修正された問題

Network Observability Operator 1.10 リリースには、修正された問題がいくつか含まれています。これらの修正により、パフォーマンスとユーザーエクスペリエンスが向上します。

### 1.9.1. MetricName および Remap フィールドの検証

この更新前は、ユーザーが無効なメトリクス名を使用して **FlowMetric** カスタムリソース (CR) を作成することができました。**FlowMetric** CR は正常に作成されましたが、その元となるメトリクスはユーザーに何のエラーフィードバックを提供せず、サイレントに失敗していました。

このリリースでは、**FlowMetric**、**metricName**、および **remap** フィールドが作成前に検証されるようになりました。そのため、ユーザーが無効な名前を入力した場合、すぐに通知されます。



[NETOBSERV-2348](#)

### 1.9.2. html-to-image エクスポートのパフォーマンスの向上

この更新前は、基盤となるライブラリーのパフォーマンスの問題により、**html-to-image** エクスポート機能に時間がかかり、その結果ブラウザーがフリーズしていました。

このリリースでは、**html-to-image** ライブラリーのパフォーマンスが向上し、エクスポートの待機時間が短縮され、イメージ生成中にブラウザーがフリーズすることがなくなりました。

[NETOBSERV-2314](#)

### 1.9.3. eBPF privileged モードの警告の改善

この更新前は、**privileged** モードを必要とする **eBPF** 機能をユーザーが選択しても、**privileged** モードが設定されていないこと、または有効にする必要があることがユーザーに明確に通知されずに、機能が失敗することがよくありました。

このリリースでは、設定に矛盾がある場合、検証フックによってすぐにユーザーに警告が表示されます。これにより、ユーザーの理解が向上し、誤った設定を防ぐことができます。

[NETOBSERV-2268](#)

### 1.9.4. OpenTelemetry エクスポーターへのサブネットラベルの追加

この更新前は、**OpenTelemetry** メトリクスエクスポーターに、ネットワークフローラベル **SrcSubnetLabel** と **DstSubnetLabel** が欠落していたため、空のラベルが表示されていました。

このリリースでは、これらのラベルがエクスポーターによって正しく提供されるようになりました。また、明確さと **OpenTelemetry** 標準との整合性を向上させるために、**source.subnet.label** と **destination.subnet.label** に名前が変更されました。

[NETOBSERV-2405](#)

### 1.9.5. Network Observability コンポーネントのデフォルト toleration の削減

この更新前は、すべての Network Observability コンポーネントにデフォルトの toleration が設定されており、**NoSchedule** の taint が付与されているノードも含め、すべてのノードでコンポーネントをスケジュールすることが許可されていました。これにより、クラスターのアップグレードがブロックされることがありました。

このリリースでは、**Direct** モードで設定されている場合、**eBPF** エージェントと **Flowlogs-Pipeline** に対してのみ、デフォルトの toleration が維持されるようになりました。**Kafka** モードで設定されている場合、OpenShift Container Platform Web コンソールプラグインおよび **Flowlogs-Pipeline** から toleration が削除されるようになりました。

さらに、toleration は **FlowCollector** カスタムリソース (CR) で随時設定できますが、以前は toleration を空のリストに置き換えることは不可能でした。現在は、toleration を空のリストに置き換えることが可能です。

[NETOBSERV-2434](#)



## 第2章 NETWORK OBSERVABILITY OPERATOR リリースノートのアーカイブ

### 2.1. NETWORK OBSERVABILITY OPERATOR リリースノートのアーカイブ

これらのリリースノートは、OpenShift Container Platform の Network Observability Operator の開発履歴を記録したものです。これらはあくまで参考用に提供されています。

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。

#### 2.1.1. Network Observability Operator 1.9.3 アドバイザリー

Network Observability Operator 1.9.3 では、次のアドバイザリーを利用できます。

- [RHEA-2025:15780 Network Observability Operator 1.9.3](#)

#### 2.1.2. Network Observability Operator 1.9.2 アドバイザリー

Network Observability Operator 1.9.2 では、次のアドバイザリーを利用できます。

- [RHEA-2025:14150 Network Observability Operator 1.9.2](#)

#### 2.1.3. Network Observability 1.9.2 のバグ修正

- この更新前は、OpenShift Container Platform バージョン 4.15 以前では **TC\_ATTACH\_MODE** 設定はサポートされていませんでした。これにより、コマンドラインインターフェイス (CLI) エラーが発生し、パケットとフローの観測が妨げられました。このリリースでは、Traffic Control eXtension (TCX) のフックアタッチメントモードが、これらの古いバージョン向けに調整されました。これにより、**tcx** フックのエラーが解消され、フローとパケットの観測が可能になります。

#### 2.1.4. Network Observability Operator 1.8.0 のアドバイザリー

Network Observability Operator 1.8.0 リリースのアドバイザリーを確認できます。

Network Observability Operator 1.9.1 では、次のアドバイザリーを利用できます。

- [2025:12024 Network Observability Operator 1.9.1](#)

#### 2.1.5. Network Observability Operator 1.8.0 で修正された問題

Network Observability Operator 1.8.0 リリースで修正された問題を確認できます。

- この更新前は、アタッチモードの設定が間違っていたため、OpenShift Container Platform 4.15 でネットワークフローが観測されていませんでした。そのため、特に特定のカタログで、ユーザーがネットワークフローを正しく監視できていませんでした。このリリースでは、OpenShift Container Platform バージョン 4.16.0 より前のバージョンのデフォルトアタッチモードが **tc** に設定されているため、OpenShift Container Platform 4.15 でフローが観測されるようになりました。(NETOBSERV-2287)
- この更新前は、IPFIX コレクターが再起動すると、IPFIX エクスポーターの設定時に接続が失わ



れ、コレクターへのネットワークフローの送信が停止することがありました。このリリースでは、接続が復元され、ネットワークフローが引き続きコレクターに送信されます。  
([NETOBSERV-2287](#))

- この更新前は、IPFIX エクスポーターを設定すると、ポート情報のないフロー (ICMP トラフィックなど) が無視され、ログにエラーが記録されていました。TCP フラグと ICMP データも IPFIX エクスポートから欠落していました。このリリースでは、これらの詳細が含まれるようになりました。欠落しているフィールド (ポートなど) によってエラーが発生しなくなり、エクスポートされたデータに含まれるようになりました。(NETOBSERV-2287)
- この更新前は、OpenShift Container Platform 4.18 でユーザー定義ネットワーク (UDN) マッピング機能の設定上の問題と警告が表示されていました。これは、OpenShift のバージョンがコード内で誤って設定されていたことが原因でした。これはユーザーエクスペリエンスに影響を与えていました。このリリースでは、UDN マッピングが OpenShift Container Platform 4.18 をサポートするようになり、警告が表示されなくなったため、ユーザーエクスペリエンスがスムーズになりました。(NETOBSERV-2305)
- この更新前は、**Network Traffic** ページの拡張機能に、OpenShift Container Platform Console 4.19 との互換性の問題がありました。その結果、展開時に空のメニュースペースが表示され、ユーザーインターフェイスに一貫性がありませんでした。このリリースでは、**NetflowTraffic** 部分と **theme hook** の互換性の問題が解決されました。**Network Traffic** ビューのサイドメニューが適切に管理されるようになり、ユーザーインターフェイスの操作性が向上しました。  
(NETOBSERV-2287)

## 2.1.6. Network Observability Operator 1.9.0 のアドバイザー

Network Observability Operator 1.9.0 リリースのアドバイザーを確認できます。

- [Network Observability Operator 1.9](#)

## 2.1.7. Network Observability Operator 1.9.0 の新機能と機能拡張

Network Observability Operator 1.9.0 リリースの新機能と機能拡張を確認できます。

### 2.1.7.1. ユーザー定義ネットワークと Network Observability の連携

このリリースでは、Network Observability で [ユーザー定義ネットワーク \(UDN\)](#) 機能が一般提供になりました。Network Observability で **UDNMapping** 機能が有効になっている場合、**Traffic** フローテーブルに **UDN labels** 列が表示されます。**Source Network Name** と **Destination Network Name** の情報に基づいてログをフィルタリングできます。

### 2.1.7.2. フローログ取り込み時のフィルタリング

このリリースでは、生成されるネットワークフローの数と Network Observability コンポーネントのリソース使用量を削減するためのフィルターを作成できます。設定できるフィルターは次のとおりです。

- eBPF エージェントフィルター
- flowlogs-pipeline フィルター

### 2.1.7.3. IPsec のサポート

この更新により、OpenShift Container Platform で IPsec が有効な場合、Network Observability に次の機能拡張が導入されます。



- **IPsec Status** という新しい列が Network Observability の **Traffic** フロービューに表示され、フローが正常に IPsec で暗号化されたかどうか、または暗号化/復号化中にエラーが発生したかどうかが表示されます。
- 暗号化されたトラフィックの割合を示す新しいダッシュボードが生成されます。

#### 2.1.7.4. Network Observability CLI

パケット、フロー、メトリクスのキャプチャーで、次のフィルタリングオプションが利用できるようになりました。

- **--sampling** オプションを使用して、サンプリングされるパケットの比率を設定します。
- **--query** オプションを使用して、カスタムクエリーを使用してフローをフィルタリングします。
- **--interfaces** オプションを使用して、監視するインターフェイスを指定します。
- **--exclude\_interfaces** オプションを使用して、除外するインターフェイスを指定します。
- **--include\_list** オプションを使用して、生成するメトリクス名を指定します。

詳細は以下を参照してください。

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

#### 2.1.8. Network Observability Operator リリースノート 1.9.0 の主な技術上の変更点

Network Observability Operator 1.6.0 リリースの主な技術上の変更点を確認できます。

- Network Observability 1.9 では、**NetworkEvents** 機能が、OpenShift Container Platform 4.19 の新しい Linux カーネルで動作するように更新されました。この更新により、古いカーネルとの互換性が失われます。そのため、**NetworkEvents** 機能は OpenShift Container Platform 4.19 でのみ使用できます。Network Observability 1.8 および OpenShift Container Platform 4.18 でこの機能を使用している場合は、Network Observability のアップグレードを回避するか、Network Observability 1.9 にアップグレードし、OpenShift Container Platform を 4.19 にアップグレードすることを検討してください。
- **netobserv-reader** クラスターロールの名前が **netobserv-loki-reader** に変更されました。
- eBPF エージェントの CPU パフォーマンスが向上しました。

#### 2.1.9. Network Observability Operator 1.9.0 のテクノロジープレビュー機能

Network Observability Operator 1.9.0 リリースのテクノロジープレビュー機能を確認できます。

現在、こ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 있습니다。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](#)

##### 2.1.9.1. eBPF Manager Operator と Network Observability の連携

eBPF Manager Operator は、すべての eBPF プログラムを管理することで、攻撃対象領域を削減し、コ



ンプライアンス、セキュリティ、競合防止を実現します。Network Observability は、eBPF Manager Operator を使用してフックをロードできます。これにより、特権モードや、**CAP\_BPF** や **CAP\_PERFMON** などの追加の Linux ケイパビリティを eBPF エージェントに提供する必要がなくなります。eBPF Manager Operator と Network Observability の連携は、64 ビット AMD アーキテクチャーでのみサポートされています。

### 2.1.10. Network Observability Operator 1.9.0 の CVE

Network Observability Operator 1.9.0 リリースの CVE を確認できます。

- [CVE-2025-26791](#)

### 2.1.11. Network Observability Operator 1.9.0 で修正された問題

Network Observability Operator 1.9.0 リリースで修正された問題を確認できます。

- 以前は、コンソールプラグインから送信元または送信先 IP でフィルタリングするときに、**10.128.0.0/24** などの Classless Inter-Domain Routing (CIDR) 表記を使用すると機能せず、除外されるはずの結果が返されていました。この更新により、CIDR 表記を使用できるようになり、結果が期待どおりにフィルタリングされるようになりました。(NETOBSERV-2276)
- 以前は、ネットワークフローが使用中のネットワークインターフェイスを誤って識別することがあり、特に **eth0** と **ens5** が混同されるリスクがありました。この問題は、eBPF エージェントが **Privileged** として設定されている場合にのみ発生していました。この更新により、問題が部分的に修正され、ほぼすべてのネットワークインターフェイスが正しく識別されるようになりました。詳細は、以下の既知の問題を参照してください。(NETOBSERV-2257)
- 以前は、Operator が動作を適応させるために利用可能な Kubernetes API をチェックするときに、古い API がある場合、Operator の正常な起動を妨げるエラーが発生していました。この更新により、Operator は関連のない API のエラーを無視し、関連する API のエラーをログに記録して、正常に実行を続行するようになりました。(NETOBSERV-2240)
- 以前は、コンソールプラグインの **Traffic** フロービューで、フローを **Bytes** または **Packets** で並べ替えることができませんでした。この更新により、ユーザーがフローを **Bytes** と **Packets** で並べ替えられるようになりました。(NETOBSERV-2239)
- 以前は、IPFIX エクスポートを使用して **FlowCollector** リソースを設定すると、IPFIX フロー内の MAC アドレスが最初の 2 バイトに切り捨てられていました。この更新により、MAC アドレスが IPFIX フロー内で完全に表現されるようになりました。(NETOBSERV-2208)
- 以前は、Operator 検証 Webhook から送信される警告の一部に、実行する必要がある内容が明確に示されていないものがありました。この更新により、このようなメッセージの一部が見直され、より実用的なものに修正されました。(NETOBSERV-2178)
- 以前は、入力エラーなどが発生した場合、**FlowCollector** リソースから **LokiStack** を参照するときに問題が発生したのかが明確にわかりませんでした。この更新により、そのような場合に、参照された **LokiStack** が見つからないことが **FlowCollector** ステータスに明確に示されるようになりました。(NETOBSERV-2174)
- 以前は、コンソールプラグインの **Traffic flows** ビューで、テキストがオーバーフローすると、テキストの省略記号により、表示されるテキストの大部分が隠れてしまうことがありました。この更新により、可能な限り多くのテキストが表示されるようになりました。(NETOBSERV-2119)
- 以前は、Network Observability 1.8.1 以前のコンソールプラグインが OpenShift Container Platform 4.19 Web コンソールで動作しなかったため、**Network Traffic** ページにアクセスでき



ませんでした。この更新により、コンソールプラグインに互換性が追加され、Network Observability 1.9.0 で **Network Traffic** ページにアクセスできるようになりました。  
([NETOBSERV-2046](#))

- 以前は、会話トラッキング (**FlowCollector** リソースの **logTypes: Conversations** または **logTypes: All**) を使用すると、ダッシュボードに表示される **Traffic** レートのメトリクスに不具合が発生し、トラフィックの増加が制御不能であると誤って表示されていました。現在は、より正確なトラフィックレートがメトリクスに表示されます。ただし、**Conversations** および **EndedConversations** モードでは、長時間にわたる接続は対象外であるため、これらのメトリクスは依然として完全には正確でないことに注意してください。この情報はドキュメントに追加されました。このような不正確さを避けるために、デフォルトモードの **logTypes: Flows** が推奨されます。(NETOBSERV-1955)

### 2.1.12. Network Observability Operator 1.9.0 の既知の問題

Network Observability Operator 1.9.0 リリースの既知の問題を確認できます。

- ユーザー定義ネットワーク (UDN) 機能はサポートされていますが、OpenShift Container Platform 4.18 で使用すると、設定の問題と警告が表示されます。この警告は無視できます。  
([NETOBSERV-2305](#))
- まれに、eBPF エージェントが複数のネットワーク namespace がある環境で **privileged** モードで実行されている場合、フローと関連するインターフェイスを適切に相関させることができないことがあります。この問題の大部分は今回のリリースで特定され解決されました。しかし、特に **ens5** インターフェイスに関しては、いくつかの不整合が残っています。(NETOBSERV-2287)

### 2.1.13. Network Observability Operator 1.8.1 のアドバイザリー

Network Observability Operator 1.8.1 リリースのアドバイザリーを確認できます。

[Network Observability Operator 1.8.1](#)

### 2.1.14. Network Observability Operator 1.8.1 の CVE

Network Observability Operator 1.8.1 リリースの CVE を確認できます。

- [CVE-2024-56171](#)
- [CVE-2025-24928](#)

### 2.1.15. Network Observability Operator 1.8.0 で修正された問題

Network Observability Operator 1.8.1 リリースで修正された問題を確認できます。

- この修正により、OpenShift Container Platform の今後のバージョンでは、**Observe** メニューが1回だけ表示されるようになります。(NETOBSERV-2139)

### 2.1.16. Network Observability Operator 1.8.0 のアドバイザリー

Network Observability Operator 1.8.0 リリースのアドバイザリーを確認できます。

- [Network Observability Operator 1.8.0](#)



## 2.1.17. Network Observability Operator 1.8.0 の新機能と機能拡張

Network Observability Operator 1.8.0 リリースの新機能と機能拡張を確認できます。

### 2.1.17.1. パケット変換

変換されたエンドポイント情報を使用してネットワークフローをエンリッチできるようになりました。サービスだけでなく特定のバックエンド Pod も表示されるため、どの Pod がリクエストを処理したか確認できます。

詳細は以下を参照してください。

- [エンドポイント変換 \(xlat\)](#)
- [エンドポイント変換 \(xlat\) の操作](#)

### 2.1.17.2. OVN-Kubernetes ネットワークイベントの追跡



#### 重要

OVN-Kubernetes ネットワークイベントの追跡は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、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、以下のリンク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](#)

Network Observability のネットワークイベントトラッキングを使用して、ネットワークポリシー、管理ネットワークポリシー、Egress ファイアウォールなどの OVN-Kubernetes イベントに関する情報を取得できるようになりました。

詳細は以下を参照してください。

- [ネットワークイベントの表示](#)

### 2.1.17.3. 1.8 における eBPF パフォーマンスの改善

- Network Observability では、per-CPU マップの代わりにハッシュマップが使用されるようになりました。つまり、ネットワークフローデータがカーネル空間で追跡され、新しいパケットもそこに集約されます。ネットワークフローの重複排除がカーネル内で実行できるようになったため、カーネルとユーザー空間の間のデータ転送サイズによってパフォーマンスが向上します。これらの eBPF パフォーマンスの向上により、eBPF エージェントで CPU リソースが 40% から 57% 削減される可能性があります。

### 2.1.17.4. Network Observability CLI

このリリースでは、Network Observability CLI に次の新しい機能、オプション、フィルターが追加されました。



- **oc netobserv metrics** コマンドを実行して、フィルターを有効にしてメトリクスをキャプチャーします。
- フローおよびパケットキャプチャーで **--background** オプションを使用して CLI をバックグラウンドで実行し、**oc netobserv follow** を実行してバックグラウンド実行の進行状況を確認し、**oc netobserv copy** を実行して生成されたログをダウンロードします。
- **--get-subnets** オプションを使用して、マシン、Pod、およびサービスのサブネットでフローとメトリクスのキャプチャーを強化します。
- 以下は、パケット、フロー、メトリクスのキャプチャーで利用できる新しいフィルタリングオプションです。
  - IP、ポート、プロトコル、アクション、TCP フラグなどに基づく eBPF フィルター
  - **--node-selector** を使用するカスタムノード
  - **--drops** のみを使用するドロップ
  - **--regexes** を使用する任意のフィールド

詳細は以下を参照してください。

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

## 2.1.18. Network Observability Operator リリースノート 1.8.0 の修正された問題

Network Observability Operator 1.8.0 リリースで修正された問題を確認できます。

- 以前は、Network Observability Operator には、メトリクスサーバーの RBAC を管理するための "kube-rbac-proxy" コンテナが付属していました。この外部コンポーネントは非推奨であるため、削除する必要がありました。これは、サイドカープロキシを必要としない、Kubernetes コントローラーランタイムを介した TLS および RBAC の直接管理に置き換えられました。(NETOBSERV-1999)
- 以前の OpenShift Container Platform コンソールプラグインでは、複数の値と一致しないキーでフィルタリングするとフィルタリングされませんでした。この修正により、フィルタリングされた値が一切含まれないフローという期待どおりの結果が返されます。(NETOBSERV-1990)
- 以前は、Loki が無効になっている OpenShift Container Platform コンソールプラグインでは、互換性のないフィルターと集計のセットを選択することで "Can't build query" エラーが発生することが多くなっていました。現在は、ユーザーにフィルターの非互換性を認識させつつ、互換性のないフィルターを自動的に無効にすることで、このエラーを回避しています。(NETOBSERV-1977)
- 以前は、コンソールプラグインからフローの詳細を表示すると、ICMP 情報が常にサイドパネルに表示され、ICMP 以外のフローの場合は "undefined" の値が表示されていました。この修正により、ICMP 以外のフローでは ICMP 情報が表示されなくなります。(NETOBSERV-1969)
- 以前は、Traffic flows ビューの "Export data" リンクが意図したとおりに機能せず、空の CSV レポートが生成されていました。現在は、エクスポート機能が復元され、空ではない CSV データが生成されます。(NETOBSERV-1958)
- 以前は、会話ログは Loki が有効な場合にのみ役立つにもかかわらず、**loki.enable** を **false** に設定して、**processor.logTypes Conversations, EndedConversations**、または **All** を使用して **FlowCollector** を設定することが可能でした。その結果、リソースを無駄に使用してしま



た。現在、この設定は無効であり、検証 Webhook によって拒否されます。(NETOBSERV-1957)

- **FlowCollector** で、**processor.logTypes** を **All** に設定すると、他のオプションよりも CPU、メモリー、ネットワーク帯域幅などのリソースがはるかに多く消費されます。この点は、以前は文書化されていませんでした。これは現在文書化されており、検証 Webhook から警告がトリガーされます。(NETOBSERV-1956)
- 以前は、負荷が高い場合に、eBPF エージェントによって生成された一部のフローが誤って破棄され、トラフィック帯域幅が予測を下回っていました。現在、生成されたフローは破棄されません。(NETOBSERV-1954)
- 以前は、**FlowCollector** 設定でネットワークポリシーを有効にすると、Operator Webhook へのトラフィックがブロックされ、**FlowMetrics** API 検証が機能しなくなっていました。現在は、Webhook へのトラフィックは許可されます。(NETOBSERV-1934)
- 以前は、デフォルトのネットワークポリシーをデプロイすると、**additionalNamespaces** フィールドにデフォルトで **openshift-console** と **openshift-monitoring** の namespace が設定され、ルールが重複していました。現在は、デフォルトで追加の namespace が設定されなくなったため、ルールの重複を防止できます。(NETOBSERV-1933)
- 以前は、OpenShift Container Platform コンソールプラグインから TCP フラグでフィルタリングすると、目的のフラグのみを持つフローが一致していました。現在は、少なくとも目的のフラグを持つフローがフィルタリングされたフローに表示されるようになります。(NETOBSERV-1890)
- eBPF エージェントが特権モードで実行され、Pod が継続的に追加または削除されると、ファイル記述子 (FD) リークが発生します。この修正により、ネットワーク namespace の削除時に FD が適切に閉じられるようになります。(NETOBSERV-2063)
- 以前は、CLI エージェント **DaemonSet** はマスターノードにデプロイされませんでした。現在は、**taint** が設定された場合にすべてのノードでスケジュールするための **toleration** がエージェント **DaemonSet** に追加されています。CLI エージェント **DaemonSet** Pod はすべてのノードで実行されます。(NETOBSERV-2030)
- 以前は、Prometheus ストレージのみを使用する場合、**Source Resource** および **Source Destination** フィルターのオートコンプリートは機能しませんでした。現在、この問題は修正され、提案が期待どおりに表示されるようになりました。(NETOBSERV-1885)
- 以前は、複数の IP を使用するリソースは **Topology** ビューで個別に表示されていました。現在は、リソースはビュー内で単一のトポロジーノードとして表示されます。(NETOBSERV-1818)
- 以前は、マウスポインターを列の上に置くと、コンソールで **Network traffic** テーブルビューの内容が更新されていました。現在は表示が固定され、ポインターを置いても行の高さは一定のままです。(NETOBSERV-2049)

## 2.1.19. Network Observability Operator リリースノート 1.8.0 の既知の問題

Network Observability Operator 1.8.0 リリースの既知の問題を確認できます。

- クラスター内に重複するサブネットを使用するトラフィックがある場合、eBPF エージェントが重複した IP からのフローを混同するリスクがわずかにあります。これは、異なる接続がまったく同じ送信元 IP と宛先 IP を持ち、ポートとプロトコルが 5 秒の時間枠内にあり、同じノードで発生している場合に発生する可能性があります。セカンダリーネットワークまたは UDN を設定しない限り、これは不可能です。その場合でも、通常は送信元ポートが差別化要因となるため、通常のトラフィックで発生する可能性は非常に低くなります。(NETOBSERV-2115)



- OpenShift Container Platform Web コンソールのフォームビューから、**FlowCollector** リソースの **spec.exporters** セクションで設定するエクスポートのタイプを選択した後、そのタイプの詳細な設定がフォームに表示されません。回避策として、YAML を直接設定します。  
([NETOBSERV-1981](#))

## 2.1.20. Network Observability Operator 1.7.0 アドバイザリー

Network Observability Operator 1.7.0 リリースのアドバイザリーを確認できます。

- [Network Observability Operator 1.7.0](#)

## 2.1.21. Network Observability Operator 1.7.0 の新機能と機能拡張

Network Observability Operator 1.7.0 リリースの次の新機能と機能拡張を確認できます。

### 2.1.21.1. OpenTelemetry のサポート

エンリッチされたネットワークフローを、Red Hat build of OpenTelemetry などの互換性のある OpenTelemetry エンドポイントにエクスポートできるようになりました。

詳細は以下を参照してください。

- [エンリッチされたネットワークフローデータのエクスポート](#)

### 2.1.21.2. Network Observability Developer パースペクティブ

Developer パースペクティブで Network Observability を使用できるようになりました。

詳細は以下を参照してください。

- [OpenShift Container Platform コンソールの統合](#)

### 2.1.21.3. TCP フラグフィルタリング

**tcpFlags** フィルターを使用して、eBPF プログラムによって処理されるパケットの量を制限できるようになりました。

詳細は以下を参照してください。

- [フローフィルターの設定パラメーター](#)
- [eBPF フローのルールフィルター](#)
- [FlowMetric API と TCP フラグを使用した SYN フラッディングの検出](#)

### 2.1.21.4. OpenShift Virtualization の Network Observability

Open Virtual Network (OVN)-Kubernetesなどを介してセカンダリーネットワークに接続された仮想マシンから送信される eBPF エンリッチ化ネットワークフローを検出することで、OpenShift Virtualization 環境のネットワークパターンを観測できます。

詳細は以下を参照してください。

- 仮想マシン (VM) のセカンダリーネットワークインターフェイスを Network Observability 用に設定する



### 2.1.21.5. FlowCollector カスタムリソース (CR) でのネットワークポリシーのデプロイ

このリリースでは、**FlowCollector** カスタムリソース (CR) を設定して、ネットワーク可観測性のためのネットワークポリシーをデプロイできます。以前は、ネットワークポリシーが必要な場合は、手動で作成する必要がありました。ネットワークポリシーを手動で作成するオプションは引き続き利用可能です。

詳細は以下を参照してください。

- [FlowCollector カスタムリソースを使用した Ingress ネットワークポリシーの設定](#)

### 2.1.21.6. FIPS コンプライアンス

- FIPS モードで実行されている OpenShift Container Platform クラスターに Network Observability Operator をインストールして使用できます。



#### 重要

クラスターで FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピューターからインストールプログラムを実行する必要があります。RHEL で FIPS モードを設定する方法の詳細は、[RHEL から FIPS モードへの切り替え](#) を参照してください。

FIPS モードでブートされた Red Hat Enterprise Linux (RHEL) または Red Hat Enterprise Linux CoreOS (RHCOS) を実行する場合、OpenShift Container Platform コアコンポーネントは、x86\_64、ppc64le、および s390x アーキテクチャーのみで、FIPS 140-2/140-3 検証のために NIST に提出された RHEL 暗号化ライブラリーを使用します。

### 2.1.21.7. eBPF エージェントの機能拡張

eBPF エージェントで次の機能拡張を利用できます。

- DNS サービスが **53** 以外のポートにマッピングされている場合は、**spec.agent.ebpf.advanced.env.DNS\_TRACKING\_PORT** を使用して、この DNS 追跡ポートを指定できます。
- トランスポートプロトコル (TCP、UDP、または SCTP) のフィルタリングルールに 2 つのポートを使用できるようになりました。
- プロトコルフィールドを空のままにしておくことで、ワイルドカードプロトコルを使用してトランスポートポートをフィルタリングできるようになりました。

詳細は以下を参照してください。

- [FlowCollector API 仕様](#)

### 2.1.21.8. Network Observability CLI

Network Observability CLI (**oc netobserv**) が一般提供になりました。1.6 テクノロジープレビューリリース以降、次の機能強化が行われました。

- フローキャプチャーと同様に、パケットキャプチャー用の eBPF エンリッチメントフィルターが追加されました。



- フローキャプチャーとパケットキャプチャーの両方でフィルター **tcp\_flags** を使用できるようになりました。
- 最大バイト数または最大時間に達したときに、自動ティアダウンオプションを利用できます。

詳細は以下を参照してください。

- [Network Observability CLI について](#)
- [Network Observability CLI](#)

### 2.1.22. Network Observability Operator 1.7.0 で修正された問題

Network Observability Operator 1.7.0 リリースで修正された次の問題を確認できます。

- 以前は、RHEL 9.2 リアルタイムカーネルを使用すると、一部の Webhook が機能しませんでした。現在、この RHEL 9.2 リアルタイムカーネルが使用されているかどうかを確認するための修正が導入されています。このカーネルが使用されている場合、**s390x** アーキテクチャーの使用時にパケットドロップやラウンドトリップ時間などの機能が実行されないという内容の警告が表示されます。この修正は OpenShift 4.16 以降で適用されます。(NETOBSERV-1808)
- 以前は、**Overview** タブの **Manage panels** ダイアログで、**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でした。現在は、利用可能なパネルが正しくフィルタリングされます。(NETOBSERV-1540)
- 以前は、高ストレス下で eBPF エージェントが多数の小さなフローを生成する状態になり、フローがほとんど集約されないことがありました。この修正により、集約プロセスが高いストレス下でも維持され、作成されるフローが少なくなりました。この修正により、eBPF エージェントだけでなく、**flowlogs-pipeline** および **Loki** でもリソース消費が改善されます。(NETOBSERV-1564)
- 以前は、**namespace\_flows\_total** メトリクスではなく、**workload\_flows\_total** メトリクスが有効になっていると、健全性ダッシュボードに **By namespace** フローチャートが表示されませんでした。この修正により、**workload\_flows\_total** が有効な場合に健全性ダッシュボードにフローチャートが表示されるようになりました。(NETOBSERV-1746)
- 以前は、**FlowMetrics** API を使用してカスタムメトリクスを生成し、後で新しいラベルを追加するなどしてそのラベルを変更すると、メトリクスの入力が停止し、**flowlogs-pipeline** ログにエラーが表示されていました。この修正により、ラベルを変更しても、**flowlogs-pipeline** ログにエラーが表示されなくなりました。(NETOBSERV-1748)
- 以前は、デフォルトの **Loki** の **WriteBatchSize** 設定に不一致がありました。**FlowCollector** CRD のデフォルトでは 100 KB に設定されていましたが、OLM のサンプルまたはデフォルト設定では 10 MB に設定されていました。現在は、両方とも 10 MB になりました。これにより、一般的にパフォーマンスが向上し、リソースフットプリントが削減されました。(NETOBSERV-1766)
- 以前は、プロトコルを指定しなかった場合、ポート上の eBPF フローフィルターが無視されていました。この修正により、ポートやプロトコルごとに eBPF フローフィルターを個別に設定できるようになりました。(NETOBSERV-1779)
- 以前は、Pod からサービスへのトラフィックが **トポロジービュー** に表示されませんでした。サービスから Pod への戻りトラフィックのみが表示されていました。この修正により、そのトラフィックも正しく表示されるようになりました。(NETOBSERV-1788)
- 以前は、Network Observability にアクセスできるクラスター管理者以外のユーザーが、



namespace など、自動補完をトリガーする項目をフィルタリングしようとする、コンソールプラグインにエラーが表示されていました。この修正により、エラーが表示されなくなり、自動補完によって期待どおりの結果が返されるようになりました。(NETOBSERV-1798)

- セカンダリーインターフェイスのサポートが追加されたときに、インターフェイスの通知を確認するために、ネットワークごとの namespace を netlink に登録する作業を複数回繰り返す必要がありました。同時に、TC とは異なり、TCX フックではインターフェイスがダウンしたときにハンドラーを明示的に削除する必要があったため、失敗したハンドラーによってファイル記述子のリークが発生しました。さらに、ネットワーク namespace が削除されるときに、netlink goroutine ソケットを終了する Go クローズチャネルイベントが存在していなかったため、Go スレッドがリークしていました。現在は、Pod を作成または削除するときに、ファイル記述子や Go スレッドがリークしなくなりました。(NETOBSERV-1805)
- 以前は、フローの JSON で該当するデータが利用可能であっても、**Traffic flows** テーブルの ICMP のタイプと値に、'n/a' と表示されていました。この修正により、ICMP 列にフローテーブル内の該当する値が期待どおりに表示されるようになりました。(NETOBSERV-1806)
- 以前は、コンソールプラグインで、未設定の DNS レイテンシーなどの未設定のフィールドをフィルタリングできないことがありました。この修正により、未設定のフィールドでのフィルタリングが可能になりました。(NETOBSERV-1816)
- 以前は、OpenShift Web コンソールプラグインでフィルターをクリアして、別のページに移動してからフィルターがあったページに戻ると、フィルターが再表示される場合がありました。この修正により、クリアした後にフィルターが予期せず再表示されることがなくなりました。(NETOBSERV-1733)

### 2.1.23. Network Observability Operator 1.7.0 の既知の問題

Network Observability Operator 1.7.0 リリースの次の既知の問題を確認できます。

- Network Observability で must-gather ツールを使用する場合、クラスターで FIPS が有効になっているとログが収集されません。(NETOBSERV-1830)
- **FlowCollector** で **spec.networkPolicy** が有効になっている場合、**netobserv** namespace にネットワークポリシーがインストールされるため、**FlowMetrics** API を使用できません。ネットワークポリシーにより、検証 Webhook への呼び出しがブロックされます。回避策として、次のネットワークポリシーを使用してください。

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-from-hostnetwork
  namespace: netobserv
spec:
  podSelector:
    matchLabels:
      app: netobserv-operator
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
              policy-group.network.openshift.io/host-network: "
```

(NETOBSERV-193)



### 2.1.24. Network Observability Operator リリースノート 1.6.2 のアドバイザリー

Network Observability Operator 1.6.2 リリースのアドバイザリーを確認できます。

- [2024:7074 Network Observability Operator 1.6.2](#)

### 2.1.25. Network Observability Operator リリースノート 1.6.2 の CVE

Network Observability Operator 1.6.2 リリースの CVE を確認できます。

- [CVE-2024-24791](#)

### 2.1.26. Network Observability Operator リリースノート 1.6.2 の修正された問題

Network Observability Operator 1.6.2 リリースで修正された問題を確認できます。

- セカンダリーインターフェイスのサポートが追加されたときに、インターフェイスの通知を確認するために、ネットワークごとの namespace を netlink に登録する作業を複数回繰り返す必要がありました。同時に、TC とは異なり、TCX フックではインターフェイスがダウンしたときにハンドラーを明示的に削除する必要があったため、失敗したハンドラーによってファイル記述子のリークが発生しました。これで、Pod の作成および削除時にファイル記述子がリークしなくなりました。(NETOBSERV-1805)

### 2.1.27. Network Observability Operator リリースノート 1.6.2 の既知の問題

Network Observability Operator 1.6.2 リリースの既知の問題を確認できます。

- コンソールプラグインとの互換性の問題があり、OpenShift Container Platform クラスターの将来のバージョンに Network Observability をインストールできない可能性がありました。1.6.2 にアップグレードすると、互換性の問題が解決され、Network Observability を期待どおりにインストールできるようになります。(NETOBSERV-1737)

### 2.1.28. Network Observability Operator リリースノート 1.6.1 のアドバイザリー

Network Observability Operator 1.6.1 リリースのアドバイザリーを確認できます。

- [2024:4785 Network Observability Operator 1.6.1](#)

### 2.1.29. Network Observability Operator リリースノート 1.6.1 の CVE

Network Observability Operator 1.6.1 リリースの CVE を確認できます。

- [RHSA-2024:4237](#)
- [RHSA-2024:4212](#)

### 2.1.30. Network Observability Operator リリースノート 1.6.1 の修正された問題

Network Observability Operator 1.6.1 リリースで修正された問題を確認できます。

- 以前は、原因や TCP 状態などのパケットドロップに関する情報は、Loki データストアでのみ入手でき、Prometheus では入手できませんでした。そのため、OpenShift Web コンソールプラグインの **概要** のドロップ統計は、Loki でのみ利用可能でした。この修正により、パケットドロップに関する情報もメトリクスに追加されるため、Loki が無効になっているときにドロップ統計を表示できるようになります。(NETOBSERV-1649)



- 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と、報告されたドロップされたバイトとドロップされたパケットでは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たものですが、副作用として、ドロップなしの報告率と比較したドロップの報告率が偏ってしまいました。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。この修正により、ドロップされたバイトとパケットでサンプリング設定が尊重されるようになりました。(NETOBSERV-1676)
- 以前は、最初にインターフェイスが作成されてから eBPF エージェントがデプロイされると、SR-IOV セカンダリーインターフェイスが検出されませんでした。これは、最初にエージェントがデプロイされ、その後 SR-IOV インターフェイスが作成された場合にのみ検出されました。この修正により、デプロイメントの順序に関係なく SR-IOV セカンダリーインターフェイスが検出されるようになりました。(NETOBSERV-1697)
- 以前は、Loki が無効になっていると、関連機能が有効になっていない場合でも、OpenShift Web コンソールの **Topology** ビューで、ネットワークトポロジダイアグラムの横にあるスライダーに **Cluster** と **Zone** の集約オプションが表示されていました。この修正により、スライダーには有効な機能に応じたオプションのみが表示されるようになりました。(NETOBSERV-1705)
- 以前は、Loki が無効になっていて、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** エラーが発生していました。この修正により、エラーは発生しなくなりました。(NETOBSERV-1706)
- 以前は、OpenShift Web コンソールの **トポロジ** ビューで、任意のグラフノードの横にある **Step into** アイコンをクリックすると、選択したグラフノードにフォーカスを設定するために必要なフィルターが適用されず、OpenShift Web コンソールに **Topology** ビューの広いビューが表示されていました。この修正により、フィルターが正しく設定され、**トポロジ** が効果的に絞り込まれます。この変更の一環として、**ノード** 上の **Step into** アイコンをクリックすると、**Namespaces** スcopeではなく **Resource** スcopeに移動するようになりました。(NETOBSERV-1720)
- 以前は、Loki が無効になっていると、**Scope** が **Owner** に設定されている OpenShift Web コンソールの **Topology** ビューで、任意のグラフノードの横にある **Step into** アイコンをクリックすると、**Scope** が **Resource** に移動しましたが、これは Loki なしでは利用できないため、エラーメッセージが表示されていました。この修正により、Loki が無効になっていると、**Owner** スcopeで **Step into** アイコンが非表示になるため、このシナリオは発生しなくなります。(NETOBSERV-1721)
- 以前は、Loki が無効になっている場合に、グループを設定すると OpenShift Web コンソールの **Topology** ビューにエラーが表示されましたが、その後スcopeが変更されたため、グループが無効になりました。この修正により、無効なグループが削除され、エラーが防止されます。(NETOBSERV-1722)
- **YAML** ビューではなく、OpenShift Web コンソールの **Form view** から **FlowCollector** リソースを作成すると、**agent.ebpf.metrics.enable** および **processor.subnetLabels.openShiftAutoDetect** の設定が Web コンソールによって誤って管理されていました。これらの設定は、**Form view** ではなく、**YAML view** でのみ無効にできます。混乱を避けるため、これらの設定は **Form view** から削除されました。これらは引き続き **YAML view** でアクセスできます。(NETOBSERV-1731)
- 以前は、eBPF エージェントは、SIGTERM 信号によるクラッシュなど、予期しないクラッシュの前にインストールされたトラフィック制御フローをクリーンアップできませんでした。これにより、古いものが削除されなかったため、同じ名前のトラフィック制御フローフィルターが



複数作成されました。この修正により、エージェントの起動時に、新しいトラフィック制御フローがインストールされる前に、以前にインストールされたトラフィック制御フローがすべてクリーンアップされるようになります。(NETOBSERV-1732)

- 以前は、カスタムサブネットラベルを設定し、OpenShift サブネットの自動検出を有効にしたままにすると、OpenShift サブネットがカスタムサブネットよりも優先され、クラスターサブネット内のカスタムラベルの定義が妨げられていました。この修正により、カスタム定義されたサブネットが優先され、クラスター内のサブネットにカスタムラベルを定義できるようになります。(NETOBSERV-1734)

### 2.1.31. Network Observability Operator リリースノート 1.6.0 のアドバイザリー

Network Observability Operator 1.6.0 リリースのアドバイザリーを確認できます。

- [Network Observability Operator 1.6.0](#)

### 2.1.32. Network Observability Operator 1.6.0 の新機能と機能拡張

Network Observability Operator 1.6.0 の次の新機能と機能拡張を確認できます。

#### 2.1.32.1. 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。

詳細は以下を参照してください。

- [Loki を使用しない Network Observability](#)

#### 2.1.32.2. 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。

詳細は以下を参照してください。

- [カスタムメトリクス](#)
- [FlowMetric API リファレンス](#)

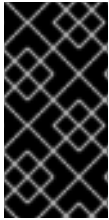
#### 2.1.32.3. eBPF のパフォーマンスの強化

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。

詳細は以下を参照してください。

- [eBPF エージェントアラートの使用](#)
- [Network Observability メトリクスのダッシュボード](#)
- [ネットワークトラフィックのフィルタリング](#)

### 2.1.32.4. 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。

詳細は以下を参照してください。

- [eBPF フローのルールフィルター](#)

### 2.1.33. Network Observability Operator 1.6.0 で修正された問題

Network Observability Operator 1.6.0 で修正された次の問題を確認できます。

- 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ていませんでした。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)

### 2.1.34. Network Observability Operator 1.6.0 の既知の問題

Network Observability Operator 1.6.0 の次の既知の問題を確認できます。

- 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ますが、副作用として、ドロップが報告された割合と非ドロップが報告された割合が偏ってしまいます。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** 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、ネットワークポロジ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)

### 2.1.35. Network Observability Operator 1.5.0 アドバイザリー

Network Observability Operator 1.5 リリースの次のアドバイザリーを参照できます。

[Network Observability Operator 1.5.0](#)

### 2.1.36. Network Observability Operator 1.5.0 の新機能と機能拡張

Network Observability Operator 1.5 リリースの次の新機能と機能拡張を確認できます。

#### 2.1.36.1. 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。

詳細は以下を参照してください。



- [DNS 追跡の設定](#)
- [DNS 追跡の使用](#)

### 2.1.36.2. ラウンドトリップタイム (RTT)

**fentry/tcp\_rcv\_established** 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。

詳細は以下を参照してください。

- [RTT の概要](#)
- [RTT の操作](#)

### 2.1.36.3. 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は、以下を参照してください。

- [Network Observability メトリクス](#)

### 2.1.36.4. 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。

詳細は以下を参照してください。

- [Network Observability メトリクス](#)

### 2.1.36.5. アベイラビリティーゾーン

クラスターのアベイラビリティー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](https://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。

詳細は以下を参照してください。

- [アベイラビリティーゾーンの使用](#)

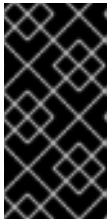
### 2.1.36.6. 主な機能拡張

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。



### 2.1.36.7. 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 のメモリ消費のパフォーマンスリグレー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。

### 2.1.36.8. 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 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。

### 2.1.36.9. 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。

### 2.1.37. Network Observability Operator 1.5.0 で修正された問題

Network Observability Operator 1.5 リリースで修正された次の問題を確認できます。

- 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す。



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)

### 2.1.38. Network Observability Operator 1.5.0 の既知の問題

Network Observability Operator 1.5 リリースの次の既知の問題を確認できます。

- 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す。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 taint 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ー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 がメモリー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。

### 2.1.39. Network Observability Operator 1.4.2 のアドバイザー

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

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

### 2.1.40. Network Observability Operator 1.4.2 の CVE

Network Observability Operator 1.4.2 リリースでは、次の CVE を確認できます。



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

### 2.1.41. Network Observability Operator 1.4.1 のアドバイザー

Network Observability Operator 1.4.1 の次のアドバイザーを確認できます。

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

### 2.1.42. Network Observability Operator リリース 1.4.1 の CVE

Network Observability Operator 1.4.1 リリースでは、次の CVE を確認できます。

- [2023-44487](#)
- [2023-39325](#)
- [2023-29406](#)
- [2023-29409](#)
- [2023-39322](#)
- [2023-39318](#)
- [2023-39319](#)
- [2023-39321](#)

### 2.1.43. Network Observability Operator リリースノート 1.4.1 の修正された問題

Network Observability Operator 1.4.1 リリースで修正された次の問題を確認できます。

- 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)

### 2.1.44. ネットワーク可観測性リリースノート 1.4.0 のアドバイザー

Network Observability Operator 1.4.0 リリースの次のアドバイザーを確認できます。

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

### 2.1.45. ネットワーク可観測性リリースノート 1.4.0 の新機能と機能拡張

Network Observability Operator 1.4.0 リリースでは、次の新機能と機能拡張を確認できます。



### 2.1.45.1. 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。

### 2.1.45.2. 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。

詳細は以下を参照してください。

- [Network Observability メトリクスのダッシュボード](#)
- [クイックフィルター](#)

### 2.1.45.3. 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。

詳細は以下を参照してください。

- [Flow Collector のサンプルリソース](#)
- [Flow Collector API リファレンス](#)

### 2.1.45.4. 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。

詳細は以下を参照してください。



- [Loki を使用しない Network Observability](#)

#### 2.1.45.5. 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。

詳細は以下を参照してください。

- [DNS 追跡の設定](#)
- [DNS 追跡の使用](#)

#### 2.1.45.6. 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。

詳細は以下を参照してください。

- [SR-IOV インターフェイストラフィックのモニタリングの設定](#)

#### 2.1.45.7. IPFIX エクスポートのサポート

eBPF エンリッチ化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。

詳細は以下を参照してください。

- [エンリッチされたネットワークフローデータのエクスポート](#)

#### 2.1.45.8. 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。

詳細は以下を参照してください。

- [パケットドロップ追跡の設定](#)
- [パケットドロップの使用](#)

#### 2.1.45.9. 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。

### 2.1.46. ネットワーク可観測性リリースノート 1.4.0 で削除された機能

Network Observability Operator 1.4.0 リリースでは、次の削除された機能を確認できます。



### 2.1.46.1. 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。

### 2.1.47. ネットワーク可観測性リリースノート 1.4.0 の修正された問題

Network Observability Operator 1.4.0 リリースで修正された次の問題を確認できます。

- 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す。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](#))

### 2.1.48. ネットワーク可観測性リリースノート 1.4.0 の既知の問題

Network Observability Operator 1.4.0 リリースでは、次の既知の問題を確認できます。

- 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 taint 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ー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 がメモリー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。

#### 2.1.49. Network Observability Operator 1.3.0 のアドバイザリー

Network Observability Operator 1.3.0 リリースでは、次のアドバイザリーを確認できます。

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

#### 2.1.50. Network Observability Operator 1.3.0 の新機能と機能拡張

Network Observability Operator 1.3.0 リリースでは、次の新機能と機能拡張を確認できます。

##### 2.1.50.1. 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。

##### 2.1.50.2. 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 メトリクスのダッシュボード」を参照してください。

##### 2.1.50.3. 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。

##### 2.1.50.4. 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。



## 2.1.51. Network Observability Operator 1.3.0 の非推奨機能

Network Observability Operator 1.3.0 リリースでは、次の非推奨機能を確認できます。

### 2.1.51.1. チャネルの非推奨化

今後の 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** チャネルは非推奨となり、次のリリースで削除される予定です。

### 2.1.51.2. 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。

## 2.1.52. Network Observability Operator 1.3.0 で修正された問題

Network Observability Operator 1.3.0 リリースで修正された次の問題を確認できます。

- 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した。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した。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しました。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)

### 2.1.53. Network Observability Operator 1.3.0 の既知の問題

Network Observability Operator 1.3.0 リリースの問題をトラブルシューティングするために、次の問題とその回避策 (存在する場合) を確認できます。

- **FlowCollector** で **processor.metrics.tls** が **PROVIDED** に設定されている場合、**flowlogs-pipeline servicemonitor** 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 taint 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。

### 2.1.54. ネットワーク可観測性リリースノート 1.2.0 における次回更新に向けての準備

今後のリリースと更新を引き続き受け取るには、Network Observability Operator の更新チャンネルを非推奨の **v1.0.x** から **stable** チャンネルに切り替えます。

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。

### 2.1.55. Network Observability Operator 1.2.0 のアドバイザー

Network Observability Operator 1.2.0 リリースの次のアドバイザーを参照できます。

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

### 2.1.56. Network Observability Operator 1.2.0 の新機能と機能拡張

Network Observability Operator 1.2.0 リリースの次の新機能と機能拡張を確認できます。

#### 2.1.56.1. Traffic Flows ビューのヒストグラム

このドキュメントは、Network Observability Operator 1.2.0 リリースの新しい機能と機能拡張に関するものです。



経時的なフローのヒストグラムを表示することを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。

### 2.1.56.2. 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。

### 2.1.56.3. 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。

## 2.1.57. Network Observability Operator 1.2.0 のバグ修正

Network Observability Operator 1.2.0 リリースで修正された次の問題を確認できます。

- 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)

## 2.1.58. Network Observability Operator 1.2.0 の既知の問題

Network Observability Operator 1.2.0 リリースの問題をトラブルシューティングするために、次の問題とその回避策 (存在する場合) を確認してください。



- 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。[\(NETOBSERV-980\)](#)

### 2.1.59. Network Observability Operator 1.2.0 の主な技術上の変更点

新しい技術変更により、Network Observability Operator 1.2.0 リリースでは、**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 を使用していたユーザーは、古いインスタンスを削除して Operator を再インストールする必要があります。

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](#)

### 2.1.60. Network Observability Operator 1.1.0 の機能拡張

Network Observability Operator 1.1.0 の次のアドバイザリーを参照できます。

- [RHSA-2023:0786 Network Observability Operator Security Advisory Update](#)

Network Observability Operator は安定版になり、リリースチャネルが **v1.1.0** にアップグレードされました。

### 2.1.61. Network Observability Operator 1.10 で修正された問題

Network Observability Operator 1.1.0 リリースで修正された次の問題を確認できます。

- 以前は、Loki の **authToken** 設定が **FORWARD** モードに設定されていない限り、認証が強制されず、権限のないユーザーが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。[\(BZ#2169468\)](#)

### 2.1.62. 関連情報

- [Network Observability におけるマルチテナンシー](#)
- [Network Observability メトリクスのダッシュボード](#)



- [ネットワーク可観測性の must-gather](#)
- [ヒストグラムの使用](#)
- [会話の使用](#)
- [健全性ダッシュボード](#)



## 第3章 NETWORK OBSERVABILITY について

Network Observability Operator を使用し、**eBPF** テクノロジーを利用してネットワークトラフィックを観測することで、Prometheus メトリクスと Loki ログを通じてトラブルシューティング用の詳細情報を入手できます。

OpenShift Container Platform コンソールでこの保存された情報を表示および分析して、さらに詳細な分析やトラブルシューティングを行うことができます。

### 3.1. NETWORK OBSERVABILITY OPERATOR

Network Observability Operator は、クラスタースコープの **FlowCollector** API カスタムリソースを提供します。これは、ネットワークフローを Loki または Prometheus に収集、強化、および保存する eBPF エージェントとサービスのパイプラインを管理します。

**FlowCollector** インスタンスは、監視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。

**eBPF** エージェントは **daemonset** オブジェクトとしてデプロイされ、ネットワークフローを作成します。このパイプラインは、ネットワークフローを収集し、Kubernetes メタデータでエンリッチしてから、Loki への保存や Prometheus メトリクスの生成を行います。

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

Network Observability Operator をフローストレージ用の Loki Operator や AMQ Streams (Kafka)などのオプションの依存関係と統合して、回復力のある大規模なデータ処理やスケーラビリティを確保します。

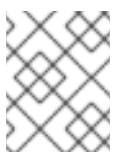
サポートされているオプションの依存関係には、フローストレージ用の Loki Operator や、Kafka を使用した大規模データ処理用の AMQ Streams などがあります。

#### Loki Operator

収集されたすべてのフローを最大限の詳細度で保存するために、Loki をバックエンドとして使用できます。Loki をインストールするには、Red Hat がサポートする Loki Operator を使用することを推奨します。Loki を使用せずに Network Observability を使用するように選択することもできますが、いくつかの要素を考慮する必要があります。詳細は、「[Loki を使用しない Network Observability](#)」を参照してください。

#### AMQ Streams Operator

Kafka は、大規模なデプロイメント向けに OpenShift Container Platform クラスターにスケーラビリティ、復元力、高可用性を提供します。



#### 注記

Kafka を使用する場合は、Red Hat がサポートする AMQ Streams Operator を使用することを推奨します。

#### 関連情報

- [Loki を使用しない Network Observability](#)

### 3.3. OPENSIFT CONTAINER PLATFORM コンソールの統合



Network Observability Operator は OpenShift Container Platform コンソールと統合され、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。

**Observe → Dashboards** の Network Observability メトリクスダッシュボードは、管理者アクセス権を持つユーザーのみが利用できます。



#### 注記

開発者アクセスと namespace へのアクセスが制限されている管理者に対してマルチテナンシーを有効にするには、ロールを定義して権限を指定する必要があります。詳細は、「Network Observability でのマルチテナンシーの有効化」を参照してください。

#### 関連情報

- [Network Observability でのマルチテナンシーの有効化](#)

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

OpenShift Container Platform コンソールでネットワーク可観測性メトリクスダッシュボードを確認します。これは、全体的なトラフィックフローの集約、フィルタリングオプション、および Operator の正常性を監視するための専用ダッシュボードを提供します。

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全体的な集計メトリクスを表示できます。クラスター、ノード、namespace、所有者、Pod、サービスごとに情報を表示するように選択できます。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観測」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。

#### 関連情報

- [Overview ビューからのネットワークトラフィックの観測](#)
- [Network Observability メトリクス](#)
- [健全性ダッシュボード](#)

### 3.3.2. Network Observability トポロジービュー

OpenShift Container Platform コンソールのネットワーク可観測性トポロジービューには、コンポーネント間のトラフィックフローがグラフィカル表示されます。これは、さまざまなフィルターおよび表示オプションを使用して絞り込むことができます。

OpenShift Container Platform コンソールは、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す **Topology** タブを提供します。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。クラスター、ゾーン、udn、ノード、namespace、所有者、Pod、サービスの情報にアクセスできます。

### 3.3.3. トラフィックフローテーブル



OpenShift Container Platform Web コンソールの **Traffic flow** テーブルには、生のネットワークフローの詳細ビューが表示され、詳細な分析のための強力なフィルターオプションと設定可能な列が含まれています。

OpenShift Container Platform Web コンソールの **Traffic flows** タブには、ネットワークフローのデータとトラフィック量が表示されます。

### 3.4. NETWORK OBSERVABILITY CLI

Network Observability CLI (**oc netobserv**)は、Network Observability Operator の完全なインストールを必要とせずに、フローおよびパケットデータを素早く、ネットワークの問題に素早くストリーミングする軽量ツールです。

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4章 NETWORK OBSERVABILITY OPERATOR のインストール

Network Observability Operator を使用する前に、Loki Operator をインストールすることを推奨します。Loki なしでも Network Observability を使用できますが、メトリクスまたは外部エクスポート者のみが必要な場合には、特別な考慮事項が適用されます。

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。

### 4.1. LOKI を使用しない NETWORK OBSERVABILITY

Loki Operator のインストールの有無にかかわらず、利用可能な機能をネットワーク可観測性と比較します。

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4.1 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
エクスポート	X	X
マルチテナンシー	X	X
完全なフィルタリングと集計機能 <sup>[1]</sup>	X	
部分的なフィルタリングと集計機能 <sup>[2]</sup>	X	X
フローベースのメトリクスとダッシュボード	X	X
Traffic flows ビューの概要 <sup>[3]</sup>	X	X
Traffic flows ビューテーブル	X	
トポロジービュー	X	X
OpenShift Container Platform コンソールの Network Traffic タブの統合	X	X

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

- [エンリッチされたネットワークフローデータのエクスポート](#)

## 4.2. LOKI OPERATOR のインストール

ソフトウェアカタログからサポートされる Loki Operator バージョンをインストールして、ネットワーク可観測性の自動クラスター内の認証および承認を提供するセキュアな **LokiStack** インスタンスを有効にします。

[Loki Operator バージョン 6.0+](#) は、ネットワーク可観測性でサポートされる Loki Operator バージョンです。これらのバージョンは、**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し、ネットワーク可観測性に対して完全にクラスター内の認証および認可サポートを提供する機能を提供します。

## 前提条件

- 管理者権限がある。
- OpenShift Container Platform Web コンソールにアクセスできる。
- サポートされているオブジェクトストアにアクセスできる。例: AWS S3、Google Cloud Storage、Azure、Swift、Minio、OpenShift Data Foundation。

## 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。削除する必要がある **ClusterRoles** と **ClusterRoleBindings**、オブジェクトストアに保存されたデータ、および永続ボリュームが残っている可能性があります。

### 4.2.1. Loki ストレージのシークレットの作成



Amazon Web Services (AWS)などのクラウドストレージ認証情報を使用してシークレットを作成し、Loki Operator がログ永続化に必要なオブジェクトストアにアクセスできるようにします。

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 カスタムリソースの作成」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。

## 手順

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。
2. 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。
3. 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** にリストされたシークレットが表示されます。

## 関連情報

- [LokiStack カスタムリソースの作成](#)
- [Flow Collector API リファレンス](#)
- [Flow Collector のサンプルリソース](#)

### 4.2.2. 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 **LokiStack** カスタムリソースをデプロイし、Loki オブジェクトストレージの正しい namespace、デプロイメントサイズ、およびシークレット名を設定するようにしてください。

**LokiStack** カスタムリソース(CR)をデプロイして、namespace または新しいプロジェクトを作成できます。

## 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 ❶
spec:
  size: 1x.small ❷
  storage:
    schemas:
      - version: v12
        effectiveDate: '2022-06-01'
    secret:
      name: loki-s3
      type: s3
  storageClassName: gp3 ❸
  tenants:
    mode: openshift-network
```

- ❶ 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。
- ❷ 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。

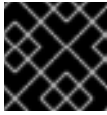


### 重要

デプロイメントサイズの **1x** の数は変更できません。

- ❸ **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。

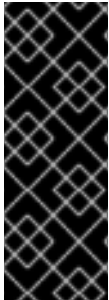


**重要**

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。

#### 4.2.3. 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照会すると、クラスター内の全 namespace の合計文字数が 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
```

#### 4.2.4. カスタム管理者グループのアクセス権

必ずしも管理者でなくてもクラスター全体のログを確認する必要がある場合、またはここで使用したいグループがすでに定義されている場合は、**adminGroup** フィールドを使用してカスタムグループを指定できます。**LokiStack** カスタムリソース (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には、管理者と同じログの読み取りアクセス権があります。

**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。

管理者ユーザーは、クラスター全体のすべてのネットワークログにアクセスできます。

#### LokiStack CR の例

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: loki
```



```

namespace: netobserv
spec:
  tenants:
    mode: openshift-network ❶
    openshift:
      adminGroups: ❷
      - cluster-admin
      - custom-admin-group ❸

```

- ❶ 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。
- ❷ 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。
- ❸ 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。

#### 4.2.5. 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4.2 Loki のサイズ

	1x.demo	1x.extra-small	1x.small	1x.medium
Data transfer	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

#### 4.2.6. 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、パフォーマンスを管理し、システムアラートやエラーを防止するために管理者が上書きできるデフォルトの取り込みとクエリー制限が含まれています。





### 注記

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、取り込みとクエリーの制限を更新することを推奨します。

設定された制限の例を次に示します。

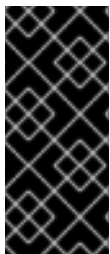
```
spec:
  limits:
    global:
      ingestion:
        ingestionBurstSize: 40
        ingestionRate: 20
        maxGlobalStreamsPerTenant: 25000
      queries:
        maxChunksPerQuery: 2000000
        maxEntriesLimitPerQuery: 10000
        maxQuerySeries: 3000
```

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。

## 4.3. NETWORK OBSERVABILITY OPERATOR のインストール

Network Observability Operator をインストールし、setup ウィザードを使用して **FlowCollector** カスタムリソース定義(CRD)を作成し、初期設定を完了します。

**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 の実際のメモリー消費量は、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、メモリー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 コントローラーマネージャー Pod のメモリーが不足する」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。
- メインネットワークプラグインとして OVN-Kubernetes を使用して設定する必要があり、オプションで 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する必要があります。



### 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. **Network Observability FlowCollector setup** ウィザードに従います。
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。

## 4.4. NETWORK OBSERVABILITY でのマルチテナンシーの有効化

クラスターロールと namespace ロールを設定して、ネットワーク可観測性でのマルチテナンシーを有効にします。これにより、プロジェクト管理者および開発者は Loki および Prometheus のフローとメトリクスへの細かいアクセスを許可します。

アクセスはプロジェクト管理者に対して有効になります。一部の namespace だけに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 のフローにのみアクセスできます。

開発者の場合、Loki と Prometheus の両方でマルチテナンシー機能を利用できますが、必要なアクセス権が異なります。

### 前提条件

- Loki を使用している場合は、少なくとも **Loki Operator バージョン 5.7** がインストールされている。
- プロジェクト管理者としてログインしている。

## 手順

- テナントごとのアクセスの場合、Developer パースペクティブを使用するために、**netobserv-loki-reader** クラスターロールと **netobserv-metrics-reader** namespace ロールを付与する必要があります。このレベルのアクセスを提供するために、次のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user netobserv-loki-reader <user_group_or_name>
```

```
$ oc adm policy add-role-to-user netobserv-metrics-reader <user_group_or_name> -n <namespace>
```



- クラスター全体のアクセスの場合、クラスター管理者以外のユーザーに、**netobserv-loki-reader**、**cluster-monitoring-view**、および **netobserv-metrics-reader** クラスターロールを付与する必要があります。この場合、Administrator パースペクティブまたは Developer パースペクティブのいずれかを使用できます。このレベルのアクセスを提供するために、次のコマンドを実行します。

```
$ oc adm policy add-cluster-role-to-user netobserv-loki-reader <user_group_or_name>
```

```
$ oc adm policy add-cluster-role-to-user cluster-monitoring-view <user_group_or_name>
```

```
$ oc adm policy add-cluster-role-to-user netobserv-metrics-reader <user_group_or_name>
```

## 4.5. 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](#)
- [エンリッチされたネットワークフローデータを Kafka または IPFIX にエクスポートする](#)
- [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](#)

### 4.5.1. FlowCollector CRD の削除された保存バージョンの移行

非推奨の **v1alpha1** バージョンを **FlowCollector** カスタムリソース定義(CRD) **storedVersion** リストから手動で削除し、アップグレードエラーを回避し、Network Observability Operator 1.6 に正常に移行します。

保存されたバージョンを削除するには、次の 2 つのオプションがあります。



1. Storage Version Migrator Operator を使用します。
2. Network Observability Operator をアンインストールして再インストールし、インストールがクリーンな状態であることを確認します。

### 前提条件

- 古いバージョンの Operator がインストールされており、最新バージョンの Operator をインストールするようにクラスターを準備する必要がある。または、Network Observability Operator 1.6 をインストールしようとして、**Failed risk of data loss updating "flowcollectors.flows.netobserv.io": new CRD removes version v1alpha1 that is listed as a stored version on the existing CRD** エラーが発生している。

### 手順

1. 古い **FlowCollector** CRD バージョンが、**storedVersion** で引き続き参照されていることを確認します。

```
$ oc get crd flowcollectors.flows.netobserv.io -ojsonpath='{.status.storedVersions}'
```

2. 結果のリストに **v1alpha1** が表示される場合は、**手順 a** に進んで Kubernetes Storage Version Migrator を使用するか、**手順 b** に進んで CRD と Operator をアンインストールして再インストールします。

- a. **オプション 1: Kubernetes Storage Version Migrator StorageVersionMigration** オブジェクトを定義する YAML を作成します (例: **migrate-flowcollector-v1alpha1.yaml**)。

```
apiVersion: migration.k8s.io/v1alpha1
kind: StorageVersionMigration
metadata:
  name: migrate-flowcollector-v1alpha1
spec:
  resource:
    group: flows.netobserv.io
    resource: flowcollectors
    version: v1alpha1
```

- i. ファイルを保存します。
- ii. 次のコマンドを実行して、**StorageVersionMigration** を適用します。

```
$ oc apply -f migrate-flowcollector-v1alpha1.yaml
```

- iii. **FlowCollector** CRD を更新して、**storedVersion** から **v1alpha1** を手動で削除します。

```
$ oc edit crd flowcollectors.flows.netobserv.io
```

- b. **オプション 2: 再インストール**: Network Observability Operator 1.5 バージョンの **FlowCollector** CR をファイル (例: **flowcollector-1.5.yaml**) に保存します。

```
$ oc get flowcollector cluster -o yaml > flowcollector-1.5.yaml
```



- i. 「Network Observability Operator のアンインストール」の手順に従って、Operator をアンインストールし、既存の **FlowCollector** CRD を削除します。
- ii. Network Observability Operator の最新バージョン 1.6.0 をインストールします。
- iii. 手順 b で保存したバックアップを使用して **FlowCollector** を作成します。

## 検証

- 以下のコマンドを実行します。

```
$ oc get crd flowcollectors.flows.netobserv.io -ojsonpath='{.status.storedVersions}'
```

結果のリストには **v1alpha1** が表示されなくなり、最新バージョンの **v1beta1** のみが表示されます。

## 関連情報

- [Kubernetes Storage Version Migrator Operator](#)

## 4.6. 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](#)

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

**Ecosystem → 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、Network Observability Operator をアンインストールします。

## 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 Options 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 Options 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、Options 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5章 OPENSIFT CONTAINER PLATFORM の NETWORK OBSERVABILITY OPERATOR

OpenShift Container Platform の Network Observability Operator は、モニタリングパイプラインをデプロイします。このパイプラインは、**eBPF agent** によって生成されたネットワークトラフィックフローを収集および拡充します。

### 5.1. 状況の表示

**oc get** コマンドを使用して **FlowCollector** リソースのステータスと、**eBPF エージェント**、**flowlogs-pipeline**、およびコンソールプラグイン Pod のステータスをチェックして、Network Observability Operator の運用ステータスを表示します。

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、収集したフローをエンリッチさせてから、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。

3. 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。

4. Loki Operator を使用している場合は、次のコマンドを入力して、**netobserv** namespace にある **LokiStack** カスタムリソースの **component** 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
lokistack-compactor-0	1/1	Running	0	18h
lokistack-distributor-654f87c5bc-qhkhv	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

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

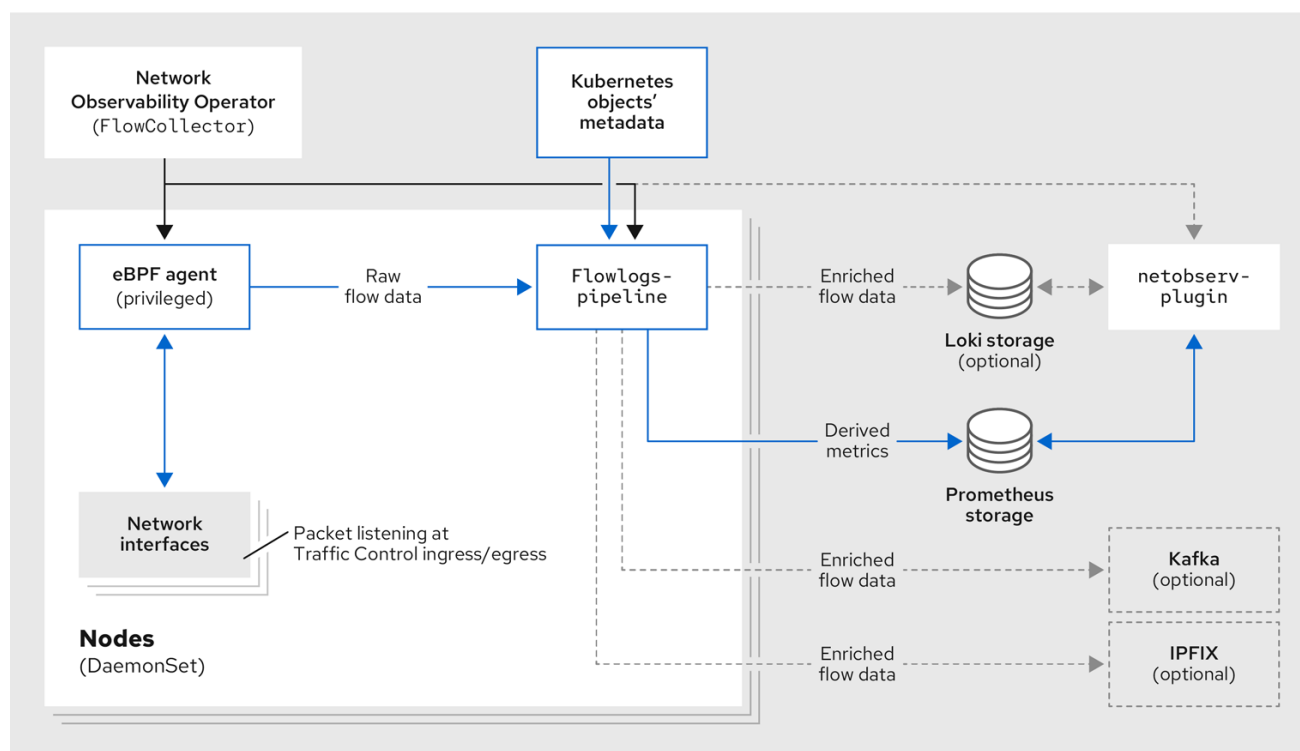
Network Observability Operator アーキテクチャーを確認してください。**FlowCollector** リソースが **eBPF エージェントを管理する方法について詳しく説明します**。このエージェントは、フローを収集して強化し、ストレージまたは Prometheus にデータを送信してメトリクスがないかの詳細を示します。

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 を使用することを選択した場合、**flowlogs-pipeline** はフローログデータを Loki に送信し、保存およびインデックス作成を行います。**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。

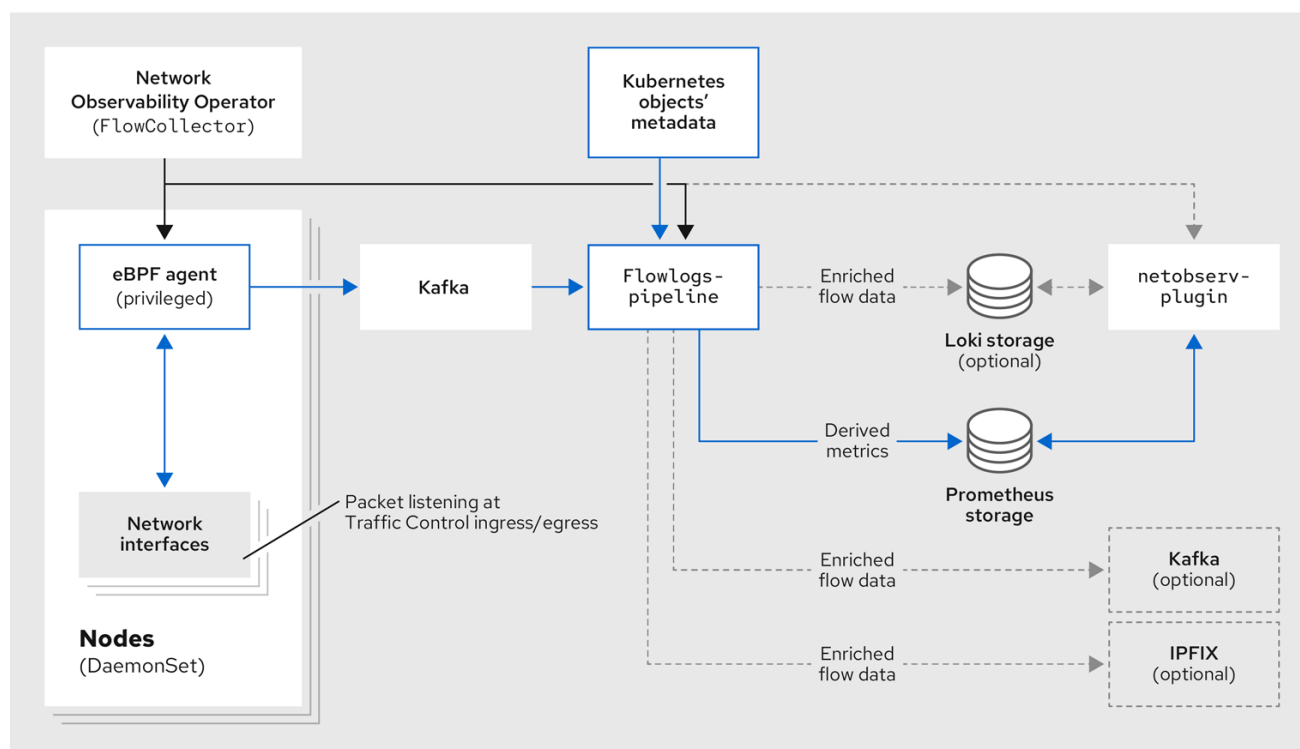


Loki を使用しない場合は、Prometheus を使用してメトリクスを生成できます。これらのメトリクスと関連するダッシュボードには、Web コンソールからアクセスできます。詳細は、"Loki を使用しない Network Observability" を参照してください。



461\_OpenShift\_0824

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。



461\_OpenShift\_0824



## 関連情報

- [Loki を使用しない Network Observability](#)

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

**oc describe flowcollector/cluster** コマンドを使用して、Network Observability Operator の現在のステータス、設定の詳細、および生成されたリソースを検査します。

### 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6章 NETWORK OBSERVABILITY OPERATOR の設定

Network Observability Operator を設定するには、クラスター全体の **FlowCollector** API リソース (クラスター) を更新して、コンポーネント設定とフロー収集設定を管理します。

**FlowCollector** 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。詳細は、[FlowCollector API リファレンス](#) を参照してください。

### 6.1. FLOWCOLLECTOR リソースの表示

**FlowCollector** リソースは、統合セットアップ、詳細フォーム、または YAML を直接編集することで、OpenShift Container Platform Web コンソールで表示および変更できます。

#### 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** である必要があります。eBPF は、OpenShift Container Platform でサポートされてい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。デフォルトでは、eBPF サンプリングは **50** に設定されているため、フローがサンプリングされる確率は 50 分の 1 になります。サンプリング間隔の値が小さいほど、より多くの計算、メモリー、およびストレージリソースが必要になります。値が **0** または **1** の場合、すべてのフローがサンプリングされます。デフォルト値から始めて、実験結果を基に調整し、クラスターに最適な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、会話の追跡が有効になります。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



る、`netobsrv` namespace のリソースとして登録する必要があります。`netobsrv` 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

- [FlowCollector API リファレンス](#)
- [会話追跡の使用](#)

## 6.2. 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

2

3

4



- 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、必要に応じてポートを指定できます。
- 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。

### 6.3. エンリッチされたネットワークフローデータのエクスポート

ネットワークフローを、Kafka、IPFIX、Red Hat build of OpenTelemetry、またはこれら 3 つすべてに同時に送信できます。Kafka または IPFIX の場合、Splunk、Elasticsearch、Fluentd など、Kafka または IPFIX の入力をサポートするプロセッサまたはストレージで、エンリッチされたネットワークフローデータを利用できます。OpenTelemetry の場合、ネットワークフローデータとメトリクスを、Red Hat build of OpenTelemetry、Prometheus など、互換性のある OpenTelemetry エンドポイントにエクスポートできます。

#### 前提条件

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

#### 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
      kafka:
        address: "kafka-cluster-kafka-bootstrap.netobserv"
        topic: netobserv-flows-export
        tls:
```

1

2



```

    enable: false
- type: IPFIX
  ipfix:
    targetHost: "ipfix-collector.ipfix.svc.cluster.local"
    targetPort: 4739
    transport: tcp or udp
- type: OpenTelemetry
  openTelemetry:
    targetHost: my-otelcol-collector-headless.otlp.svc
    targetPort: 4317
    type: grpc
    logs:
      enable: true
    metrics:
      enable: true
      prefix: netobserv
      pushTimeInterval: 20s
      expiryTime: 2m
# fieldsMapping:
#   input: SrcAddr
#   output: source.address

```

- 1 4 6 フローを IPFIX、OpenTelemetry、Kafka に個別または同時にエクスポートできます。
- 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 として使用できる必要があります。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。
- 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。
- 7 OpenTelemetry 接続のプロトコル。使用可能なオプションは **http** と **grpc** です。
- 8 Loki 用に作成されたログと同じログをエクスポートするための OpenTelemetry 設定。
- 9 Prometheus 用に作成されたメトリクスと同じメトリクスをエクスポートするための OpenTelemetry 設定。これらの設定を、**FlowMetrics** カスタムリソースを使用して定義したカスタムメトリクスとともに、**FlowCollector** カスタムリソースの **spec.processor.metrics.includeList** パラメーターで指定します。
- 10 メトリクスを OpenTelemetry コレクターに送信する時間間隔。
- 11 オプション: Network Observability のネットワークフローの形式は、OpenTelemetry 準拠の形式に合わせて、自動的に名前が変更されます。**fieldsMapping** 仕様を使用すると、OpenTelemetry 形式の出力をカスタマイズできます。たとえば、YAML サンプルでは、**SrcAddr** が Network Observability の入力フィールドです。これは、OpenTelemetry の出力で **source.address** に名前が変更されます。「ネットワークフローの形式リファレンス」で、Network Observability の形式と OpenTelemetry の形式の両方を確認できます。



す。

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

- [ネットワークフロー形式のリファレンス](#)

## 6.4. 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 サンプリ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"
```

## 6.5. ネットワークフロー取り込み時のフィルタリング

フィルターを作成すると、生成されるネットワークフローの数を減らすことができます。ネットワークフローをフィルタリングすると、Network Observability コンポーネントのリソース使用量を削減できます。

次の 2 種類のフィルターを設定できます。

- eBPF エージェントフィルター
- flowlogs-pipeline フィルター

### 6.5.1. eBPF エージェントフィルター

eBPF エージェントフィルターはパフォーマンスを最大化します。このフィルターは、ネットワークフロー収集プロセスの最も早い段階で有効になるためです。

Network Observability Operator を使用して eBPF エージェントフィルターを設定するには、「複数のルールを使用した eBPF フローデータのフィルタリング」を参照してください。

### 6.5.2. flowlogs-pipeline フィルター

flowlogs-pipeline フィルターでは、トラフィックの選択をより細かく制御できます。このフィルターは、ネットワークフロー収集プロセスの遅い段階で有効になるためです。これは主にデータの保存を改善するために使用されます。

flowlogs-pipeline フィルターは、次の例に示すように、単純なクエリー言語を使用してネットワークフローをフィルタリングします。

```
(srcnamespace="netobserv" OR (srcnamespace="ingress" AND dstnamespace="netobserv")) AND
srckind!="service"
```



クエリ言語では次の構文を使用します。

表6.1 クエリ言語の構文

カテゴリー	演算子
論理ブール演算子 (大文字と小文字の区別なし)	<b>and</b> 、 <b>or</b>
比較演算子	<b>=</b> (等しい)、 <b>!=</b> (等しくない)、 <b>=~</b> (正規表現にマッチ)、 <b>!~</b> (正規表現にマッチしない)、 <b>&lt;/ &lt;=</b> (以下)、 <b>&gt;/ &gt;=</b> (以上)
単項演算子	<b>with(field)</b> (フィールドが存在する)、 <b>without(field)</b> (フィールドが存在しない)

flowlogs-pipeline フィルターは、**FlowCollector** リソースの **spec.processor.filters** セクションで設定できます。以下に例を示します。

flowlogs-pipeline フィルターの YAML の例

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  agent:
    processor:
      filters:
        - query: |
            (SrcK8S_Namespace="netobserv" OR (SrcK8S_Namespace="openshift-ingress" AND
            DstK8S_Namespace="netobserv"))
            outputTarget: Loki ❶
            sampling: 10 ❷
```

- ❶ マッチしたフローを特定の出力 (Loki、Prometheus、外部システムなど) に送信します。省略した場合は、設定されているすべての出力に送信します。
- ❷ 任意。サンプリング間隔を適用して、保存またはエクスポートするマッチしたフローの数を制限します。たとえば、**sampling: 10** は、10 分の 1 の確率でフローが保存されることを意味します。

関連情報

- 複数のルールを使用した eBPF フローデータのフィルタリング



## 6.6. 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6.2 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 <b>kind.namespace.name</b> 、ノードの場合は <b>node.name</b> です。たとえば、 <b>Deployment.my-namespace.my-web-server</b> 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。



Universal*	ソース	送信先	説明
host_addresses	src_host_addresses	dst_host_addresses	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。

## 6.7. リソース管理およびパフォーマンスに関する考慮事項

Network Observability に必要なリソースの量は、クラスターのサイズと、クラスター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、クラスター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。デフォルトでは、eBPF サンプリングは **50** に設定されているため、フローがサンプリングされる確率は 50 分の 1 になります。サンプリング間隔の値が小さいほど、より多くの計算、メモリー、およびストレージリソースが必要になります。値が **0** または **1** の場合、すべてのフローがサンプリングされます。デフォルト値から始めて、実験結果を基に調整し、クラスターに最適な設定を決定することを推奨します。

### eBPF の機能

有効にされた機能が増えるほど、CPU とメモリーへの影響が大きくなります。該当する機能の完全なリストは、"ネットワークトラフィックのモニタリング" を参照してください。

### Loki を使用しない場合

Loki ではなく Prometheus を代わりに使用することで、Network Observability に必要なリソースの量を削減できます。たとえば、Network Observability を Loki なしで設定すると、サンプリング間隔の値に応じて、メモリー使用量が合計で 20 - 65% 削減され、CPU 使用率が 10 - 30% 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。

### 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。

### パフォーマンスのファインチューニング

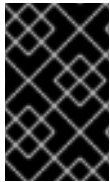


Network Observability をしばらく実行した後、次の設定を使用してパフォーマンスを微調整できます。

- **リソース要件と制限:** `spec.agent.ebpf.resources` および `spec.processor.resources` 仕様を使用して、クラスターで予想される負荷とメモリー使用量に合わせてリソース要件と制限を調整します。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 負荷の低下と関連します。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。

### 6.7.1. 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6.3 リソースの推奨事項

	極小規模 (10 ノード)	小規模 (25 ノード)	大規模 (250 ノード)[2]
ワーカーノードの vCPU とメモリー	4 仮想 CPU   16 GiB メモリー [1]	16 仮想 CPU   64 GiB メモリー [1]	16 仮想 CPU   64 GiB メモリー [1]
LokiStack サイズ	<b>1x.extra-small</b>	<b>1x.small</b>	<b>1x.medium</b>
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)
eBPF サンプリング間隔	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)	1600 Mi
cacheMaxSize	50,000	100,000 (デフォルト)	100,000 (デフォルト)
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)
FLP Kafka パーティション	–	48	48



	極小規模 (10 ノード)	小規模 (25 ノード)	大規模 (250 ノード)[2]
Kafka コンシューマーレプリカ	-	6	18
Kafka ブローカー	-	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。

### 6.7.2. メモリーと CPU の合計平均使用量

次の表は、2つの異なるテスト (**Test 1**、**Test 2**) について、サンプリング値が **1** および **50** であるクラスターの合計リソース使用量の平均を示しています。テストは次の点で異なります。

- **Test 1** は、OpenShift Container Platform クラスター内の namespace、Pod、およびサービスの合計数に加え、大量の Ingress トラフィックを考慮しており、eBPF エージェントに負荷をかけた、特定のクラスターサイズに対して多数のワークロードが発生するユースケースを表しています。たとえば、**Test 1** は、76 個の namespace、5153 個の Pod、および 2305 個のサービスで構成され、ネットワークトラフィックの規模は ~350 MB/秒です。
- **Test 2** は、OpenShift Container Platform クラスター内の namespace、Pod、およびサービスの合計数に加え、大量の Ingress トラフィックを考慮しており、特定のクラスターサイズに対して多数のワークロードが発生するユースケースを表しています。たとえば、**Test 2** は、553 個の namespace、6998 個の Pod、および 2508 個のサービスで構成され、ネットワークトラフィックの規模は ~950 MB/秒です。

さまざまなテストでさまざまなタイプのクラスターユースケースが例示されているため、この表の数値は並べて比較しても直線的に増加しません。代わりに、これらは個人のクラスター使用状況を評価するためのベンチマークとして使用することを目的としています。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。



#### 注記

Prometheus にエクスポートされたメトリクスは、リソースの使用状況に影響を与える可能性があります。メトリクスのカーディナリティー値は、リソースがどの程度影響を受けるかを判断するのに役立ちます。詳細は、関連情報セクションの「ネットワークフローの形式」を参照してください。

表6.4 リソース合計平均使用量

サンプリング値	使用されるリソース	テスト 1 (25 ノード)	テスト 2 (250 ノード)
Sampling = 50	NetObserv の CPU 合計使用量	1.35	5.39
	NetObserv RSS (メモリー) の合計使用量	16 GB	63 GB



サンプリング値	使用されるリソース	テスト 1 (25 ノード)	テスト 2 (250 ノード)
Sampling = 1	NetObserv の CPU 合計 使用量	1.82	11.99
	NetObserv RSS (メモ リー) の合計使用量	22 GB	87 GB

概要: この表は、すべての機能が有効になっているエージェント、FLP、Kafka、Loki を含む Network Observability の平均合計リソース使用量を示しています。有効な機能の詳細は、「ネットワークトラフィックの観測」で説明されている機能を参照してください。このテストで有効になっているすべての機能が記載されています。

#### 関連情報

- [Traffic flows ビューからのネットワークトラフィックの観測](#)
- [Loki を使用しない Network Observability](#)
- [ネットワークフロー形式のリファレンス](#)



## 第7章 ネットワークポリシー

管理者は、**netobserv** namespace 用のネットワークポリシーを作成できます。このポリシーにより、Network Observability Operator への受信および送信アクセスを保護します。

### 7.1. FLOWCOLLECTOR カスタムリソースを使用したネットワークポリシーの設定

Pod のトラフィックを制御するために、Ingress および Egress ネットワークポリシーを設定できます。これにより、セキュリティが強化され、必要なネットワークフローデータだけが収集されます。これにより、ノイズが削減され、コンプライアンスがサポートされるとともに、ネットワーク通信に対する可視性が向上します。

**FlowCollector** カスタムリソース (CR) を設定することで、Network Observability 用の Egress および Ingress ネットワークポリシーをデプロイできます。デフォルトでは、**spec.NetworkPolicy.enable** 仕様は **true** に設定されています。

ネットワークポリシーを持つ別の namespace に Loki、Kafka、または任意のエクスポーターをインストールした場合は、Network Observability コンポーネントがそれらと通信できることを確認する必要があります。セットアップについて、次の点を考慮してください。

- Loki への接続 (**FlowCollector** CR の **spec.loki** パラメーターで定義)
- Kafka への接続 (**FlowCollector** CR の **spec.kafka** パラメーターで定義)
- 任意のエクスポーターへの接続 (FlowCollector CR の **spec.exporters** パラメーターで定義)
- Loki を使用していて、Loki をポリシーターゲットに含める場合は、外部オブジェクトストレージへの接続 (**LokiStack** 関連のシークレットで定義)

#### 手順

1. Web コンソールで、**Operators → Installed Operators** ページに移動します。
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** CR を設定します。設定例は次のとおりです。

#### ネットワークポリシー用の FlowCollector CR の例

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  networkPolicy:
    enable: true ①
    additionalNamespaces: ["openshift-console", "openshift-monitoring"] ②
# ...
```



- ① デフォルトでは、**enable** 値は **true** です。
- ② デフォルト値は **["openshift-console", "openshift-monitoring"]** です。

## 関連情報

[CLI を使用したネットワークポリシーの作成](#)



## 第8章 ネットワークトラフィックの観測

管理者は、OpenShift Container Platform Web コンソールでネットワークトラフィックを観測し、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。

### 8.1. OVERVIEW ビューからのネットワークトラフィックの観測

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。

#### 8.1.1. 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。

#### 8.1.2. 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。

##### 8.1.2.1. 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。

### 8.1.3. 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す。

- 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

#### 8.1.3.1. 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](#)

### 8.1.4. 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](#)

### 8.1.5. 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](#)

### 8.1.6. 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取得するようにフィルターを指定できます。フィルターに一致するパケットのみがキャプチャーされ、残りはドロップされます。

複数のフィルタールールを適用できます。

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

Classless Inter-Domain Routing (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。

#### 8.1.6.2. 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観測できます。

### 8.1.6.3. 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8.1 必須設定パラメーター

パラメーター	説明
<b>enable</b>	eBPF フローのフィルタリング機能を有効にするには、 <b>enable</b> を <b>true</b> に設定します。
<b>cidr</b>	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 <b>0.0.0.0/0</b> 、IPv6 の場合は <b>::/0</b> を使用できます。
<b>action</b>	<p>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 <b>Accept</b> または <b>Reject</b> です。</p> <ul style="list-style-type: none"> <li>● <b>Accept</b> 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 <b>FlowFilterAcceptCounter</b> で更新されます。</li> <li>● <b>Reject</b> 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 <b>FlowFilterRejectCounter</b> を使用して更新されます。</li> <li>● 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 <b>FlowFilterNoMatchCounter</b> で更新されます。</li> </ul>

表8.2 オプションの設定パラメーター

パラメーター	説明
<b>direction</b>	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 <b>Ingress</b> または <b>Egress</b> です。
<b>protocol</b>	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、 <b>TCP</b> 、 <b>UDP</b> 、 <b>SCTP</b> 、 <b>ICMP</b> 、および <b>ICMPv6</b> です。
<b>tcpFlags</b>	フローをフィルタリングするための TCP フラグを定義します。可能な値は、 <b>SYN</b> 、 <b>SYN-ACK</b> 、 <b>ACK</b> 、 <b>FIN</b> 、 <b>RST</b> 、 <b>PSH</b> 、 <b>URG</b> 、 <b>ECE</b> 、 <b>CWR</b> 、 <b>FIN-ACK</b> 、および <b>RST-ACK</b> です。



パラメーター	説明
<b>ports</b>	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、 <b>ports: 80</b> 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、 <b>ports: "80-100"</b> です。
<b>sourcePorts</b>	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: <b>sourcePorts: 80</b> )。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: <b>sourcePorts: "80-100"</b> )。
<b>destPorts</b>	<b>destPorts</b> 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: <b>destPorts: 80</b> )。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: <b>destPorts: "80-100"</b> )。
<b>icmpType</b>	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。
<b>icmpCode</b>	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。
<b>peerIP</b>	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: <b>10.10.10.10</b> )。

## 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](#)

### 8.1.7. ユーザー定義ネットワーク

ユーザー定義ネットワーク (UDN) は、カスタムの Layer 2 および Layer 3 ネットワークセグメントを有効にすることで、Kubernetes Pod ネットワークのデフォルトの Layer 3 トポロジが持つ柔軟性とセグメンテーション機能を強化するものです。これらのセグメントはすべてデフォルトで分離されています。これらのセグメントは、デフォルトの OVN-Kubernetes CNI プラグインを使用するコンテナ Pod および仮想マシンのプライマリーネットワークまたはセカンダリーネットワークとして機能します。

UDN を使用することで、幅広いネットワークアーキテクチャーとトポロジが可能になり、ネットワークの柔軟性、セキュリティ、パフォーマンスが向上します。

Network Observability で **UDNMapping** 機能が有効になっている場合、**Traffic** フローテーブルに **UDN labels** 列が表示されます。**Source Network Name** と **Destination Network Name** でフィルタリングできます。

## 関連情報



- [ユーザー定義ネットワークについて](#)
- [CLI を使用した UserDefinedNetwork の作成](#)
- [Web コンソールを使用した UserDefinedNetwork の作成](#)
- [ユーザー定義ネットワークの操作](#)

### 8.1.8. OVN Kubernetes ネットワークイベント



#### 重要

OVN-Kubernetes ネットワークイベントの追跡は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、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、以下のリンク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](#)

Network Observability のネットワークイベントトラッキングを使用して、ネットワークポリシー、管理ネットワークポリシー、Egress ファイアウォールなどの OVN-Kubernetes イベントに関する情報を取得できます。ネットワークイベントの追跡から得られる情報は、次のタスクに役立ちます。

- ネットワークモニタリング: 許可されたトラフィックとブロックされたトラフィックを監視し、ネットワークポリシーと管理ネットワークポリシーに基づきパケットが許可されているか、あるいはブロックされているかを検出します。
- ネットワークセキュリティ: 送信トラフィックを追跡し、Egress ファイアウォールルールに準拠しているか確認できます。許可されていない送信接続を検出し、Egress ルールに違反する送信トラフィックにフラグを立てます。

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

- [ネットワークイベントの表示](#)

## 8.2. TRAFFIC FLOWS ビューからのネットワークトラフィックの観測

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。

### 8.2.1. 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。

### 8.2.2. 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。

#### 8.2.2.1. 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。

#### 8.2.2.2. 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。

### 8.2.3. FlowCollector カスタムリソースを使用した IPsec の設定

OpenShift Container Platform では、IPsec はデフォルトで無効になっています。「IPsec 暗号化の設定」の手順に従って IPsec を有効にできます。

#### 前提条件

- OpenShift Container Platform で IPsec 暗号化を有効にした。

#### 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. IPsec の **FlowCollector** カスタムリソースを設定します。

#### IPsec の FlowCollector の設定例

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



```

metadata:
  name: cluster
spec:
  namespace: netobserv
agent:
  type: eBPF
  ebpf:
    features:
      - "IPSec"

```

## 検証

IPsec が有効な場合:

- **IPsec Status** という新しい列が Network Observability の **Traffic** フロービューに表示され、フローが正常に IPsec で暗号化されたかどうか、または暗号化/復号化中にエラーが発生したかどうかが表示されます。
- 生成された暗号化トラフィックの割合を示す新しいダッシュボード。

## 関連情報

- [IPsec 暗号化の設定](#)

### 8.2.4. 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。

### 8.2.5. 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。

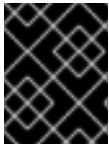
## 検証

- **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。

### 8.2.6. 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ponseコード** と **DNS ID** を含むフローを観測できますが、**DNS レイテンシー** を観測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。

## 8.2.7. 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。

### 8.2.8. eBPF Manager Operator の操作

eBPF Manager Operator は、すべての eBPF プログラムを管理することで、攻撃対象領域を削減し、コンプライアンス、セキュリティ、競合防止を実現します。Network Observability は、eBPF Manager Operator を使用してフックをロードできます。そのため、特権モードや、**CAP\_BPF** や **CAP\_PERFMON** などの追加の Linux ケイパビリティを eBPF エージェントに提供する必要がなくなります。eBPF Manager Operator と Network Observability の連携は、64 ビット AMD アーキテクチャーでのみサポートされています。

#### 重要

eBPF Manager Operator と Network Observability の連携は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、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、以下のリンク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](#)

#### 手順

1. Web コンソールで、**Operator** → **Operator Hub** に移動します。
2. **eBPF Manager** をインストールします。
3. **bpfman** namespace の **Workloads** → **Pod** をチェックして、すべてが稼働していることを確認します。
4. eBPF Manager Operator を使用するよう **FlowCollector** カスタムリソースを設定します。

#### FlowCollector の設定例

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
```



```
ebpf:
  features:
    - EbpfManager
```

## 検証

1. Web コンソールで、**Operators → Installed Operators** に移動します。
2. **eBPF Manager Operator → All instances** タブをクリックします。  
各ノードについて、**netobserv** という名前の **BpfApplication** と、**BpfProgram** オブジェクトのペア (Traffic Control (TCx) Ingress 用と TCx Egress 用のもの) が存在することを確認します。他の eBPF エージェント機能を有効にすると、オブジェクトが増える可能性があります。

## 関連情報

- [eBPF Manager Operator のインストール](#)

### 8.2.8.1. 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。

### 8.2.9. アベイラビリティーゾーンの使用

クラスターのアベイラビリティー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](https://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。

### アベイラビリティーゾーン収集用に FlowCollector を設定する

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

## 検証

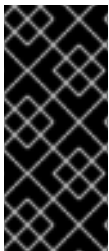


Network Traffic ページを更新すると、Overview、Traffic Flow、Topology ビューにアベイラビリティゾーンに関する新しい情報が表示されます。

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。

### 8.2.10. 複数のルールを使用した eBPF フローデータのフィルタリング

**FlowCollector** カスタムリソースを設定して、複数のルールを使用して 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。



#### 重要

- フィルタールールでは重複する Classless Inter-Domain Routing (CIDR) を使用することはできません。
- IP アドレスが複数のフィルタールールにマッチする場合、最も具体的な CIDR 接頭辞 (最も長い接頭辞) を持つルールが優先されます。

#### 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. 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。

### すべての North-South トラフィックと 1:50 の East-West トラフィックをサンプリングする YAML の例

デフォルトでは、他のすべてのフローが拒否されます。

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF
  ebpf:
    flowFilter:
      enable: true ❶
      rules:
        - action: Accept ❷
          cidr: 0.0.0.0/0 ❸
          sampling: 1 ❹
```



```
- action: Accept
  cidr: 10.128.0.0/14
  peerCIDR: 10.128.0.0/14 ⑤
- action: Accept
  cidr: 172.30.0.0/16
  peerCIDR: 10.128.0.0/14
  sampling: 50
```

- ① eBPF フローフィルタリングを有効にするには、**spec.agent.ebpf.flowFilter.enable** を **true** に設定します。
- ② フローフィルタールールのアクションを定義するには、必要な **action** パラメーターを設定します。有効な値は **Accept** または **Reject** です。
- ③ フローフィルタールールの IP アドレスと CIDR マスクを定義するには、必要な **cidr** パラメーターを設定します。このパラメーターは 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 アドレスにマッチさせるには、IPv4 の場合は **0.0.0.0/0**、IPv6 の場合は **::/0** を使用します。
- ④ マッチさせるフローのサンプリング間隔を定義し、グローバルサンプリング設定 **spec.agent.ebpf.sampling** をオーバーライドするには、**sampling** パラメーターを設定します。
- ⑤ Peer IP CIDR でフローをフィルタリングするには、**peerCIDR** パラメーターを設定します。

## パケットドロップでフローをフィルタリングする YAML の例

デフォルトでは、他のすべてのフローが拒否されます。

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
agent:
  type: eBPF
  ebpf:
    privileged: true ①
    features:
      - PacketDrop ②
    flowFilter:
      enable: true ③
      rules:
        - action: Accept ④
          cidr: 172.30.0.0/16
          pktDrops: true ⑤
```

- ① パケットドロップを有効にするには、**spec.agent.ebpf.privileged** を **true** に設定します。
- ② 各ネットワークフローのパケットドロップを報告するには、**spec.agent.ebpf.features** リストに **PacketDrop** 値を追加します。
- ③ eBPF フローフィルタリングを有効にするには、**spec.agent.ebpf.flowFilter.enable** を **true** に設



- 4 フローフィルタールールのアクションを定義するには、必要な **action** パラメーターを設定します。有効な値は **Accept** または **Reject** です。
- 5 ドロップを含むフローをフィルタリングするには、**pktDrops** を **true** に設定します。

### 8.2.11. エンドポイント変換 (xlat)

Network Observability と拡張 Berkeley Packet Filter (eBPF) を使用して、統合ビューでトラフィックを処理するエンドポイントを可視化できます。通常、トラフィックがサービス、egressIP、またはロードバランサーを通過する場合、トラフィックフロー情報は、利用可能な Pod の1つにルーティングされる時に抽象化されます。トラフィックに関する情報を取得しようとする、サービス IP やポートなどのサービス関連情報のみが表示され、リクエストを処理している特定の Pod に関する情報は表示されません。多くの場合、サービストラフィックと仮想サービスエンドポイントの両方の情報が2つの別々のフローとしてキャプチャーされるため、トラブルシューティングが複雑になります。

この問題の解決において、エンドポイント xlat は次のように役立ちます。

- カーネルレベルでネットワークフローをキャプチャーします。この場合のパフォーマンスへの影響は、最小限に抑えられます。
- 変換されたエンドポイント情報を使用してネットワークフローを拡充し、サービスだけでなく特定のバックエンド Pod も表示することで、どの Pod がリクエストを処理したか確認できます。

ネットワークパケットが処理されると、eBPF フックは、変換されたエンドポイントに関するメタデータでフローログをエンリッチします。これには、**Network Traffic** ページに1行で表示できる次の情報が含まれます。

- ソース Pod IP
- 送信元ポート
- 宛先 Pod IP
- 宛先ポート
- [Conntrack Zone ID](#)

### 8.2.12. エンドポイント変換 (xlat) の操作

Network Observability と eBPF を使用すると、Kubernetes サービスからのネットワークフローを変換されたエンドポイント情報でエンリッチして、トラフィックを処理するエンドポイントに関する詳細情報を得ることができます。

#### 手順

1. Web コンソールで、**Operators** → **Installed Operators** に移動します。
2. **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. **PacketTranslation** の **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:
        - PacketTranslation ❶

```

- ❶ **spec.agent.ebpf.features** 仕様リストに **PacketTranslation** パラメーターをリストすることで、変換されたパケット情報を使用してネットワークフローを充実させることができます。

## フィルタリングの例

Network Traffic ページを更新すると、変換されたパケットに関する情報をフィルタリングできます。

1. **Destination kind: Service** に基づき、ネットワークフローデータをフィルタリングします。
2. 変換された情報が表示される場所を区別する **xlat** 列と、次のデフォルト列が表示されます。
  - Xlat Zone ID
  - Xlat Src Kubernetes Object
  - Xlat Dst Kubernetes Object

追加の **xlat** 列の表示は、**Manage columns** で管理できます。

### 8.2.13. ユーザー定義ネットワークの操作

Network Observability リソースでユーザー定義ネットワーク (UDN) を有効にできます。次の例は、**FlowCollector** リソースの設定を示しています。

#### 前提条件

- Red Hat OpenShift Networking で UDN を設定した。詳細は、「CLI を使用した UserDefinedNetwork の作成」または「Web コンソールを使用した UserDefinedNetwork の作成」を参照してください。

#### 手順

1. 次のコマンドを実行して、Network Observability の **FlowCollector** リソースを編集します。

```
$ oc edit flowcollector
```

2. **FlowCollector** リソースの **ebpf** セクションを設定します。

```

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

```



```

name: cluster
spec:
  agent:
    ebpf:
      sampling: 1
      privileged: true
      features:
        - UDNMapping

```

- ① すべてのフローを観測できるように、この設定が推奨されます。

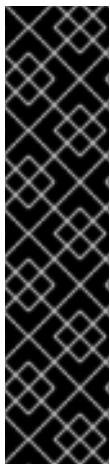
## 検証

- **Network Traffic** ページを更新して、**Traffic Flow** と **Topology** ビューで更新された UDN の情報を表示します。
  - **Network Traffic** > **Traffic flows** では、**SrcK8S\_NetworkName** フィールドと **DstK8S\_NetworkName** フィールドで UDN を確認できます。
  - **Topology** ビューでは、**Network** を **Scope** または **Group** に設定できます。

## 関連情報

- [CLI を使用した UserDefinedNetwork の作成](#)
- [Web コンソールを使用した UserDefinedNetwork の作成](#)

### 8.2.14. ネットワークイベントの表示



#### 重要

OVN-Kubernetes ネットワークイベントの追跡は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、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、以下のリンク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](#)

**FlowCollector** を編集して、次のリソースによってドロップまたは許可されたネットワークフローなどのネットワークトラフィックイベントに関する情報を表示できます。

- **NetworkPolicy**
- **AdminNetworkPolicy**
- **BaselineNetworkPolicy**
- **EgressFirewall**



- **UserDefinedNetwork** 分離
- マルチキャスト ACL

## 前提条件

- **cluster** という名前の **FeatureGate** カスタムリソース (CR) で **TechPreviewNoUpgrade** 機能セットを設定することで、**OVNObservability** を有効にした。詳細は、「CLI を使用した機能セットの有効化」および「CLI を使用して OVS サンプルングで OVN-Kubernetes ネットワークトラフィックを確認する」を参照してください。
- **NetworkPolicy**、**AdminNetworkPolicy**、**BaselineNetworkPolicy**、**UserDefinedNetwork** の分離、マルチキャスト、または **EgressFirewall** のいずれかのネットワーク API を 1 つ以上作成した。

## 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. **NetworkEvents** の表示を有効にするには、**FlowCollector** CR を設定します。例:

### FlowCollector の設定例

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
    type: eBPF
    ebpf:
      # sampling: 1
      privileged: true
      features:
        - "NetworkEvents"
```

- 1 オプション: すべてのネットワークイベントをキャプチャするために、**sampling** パラメーターの値は 1 に設定されます。サンプリング 1 ではリソースが多すぎる場合、サンプリングをニーズに合わせてより適切な値に設定します。
- 2 **OVN observability** ライブラリーはローカルの Open vSwitch (OVS) ソケットと OpenShift Virtual Network (OVN) データベースにアクセスするため、**privileged** パラメーターは **true** に設定されています。

## 検証

1. **Network Traffic** ビューに移動し、**Traffic flows** テーブルを選択します。
2. **Network Events** という新しい列が表示されます。ここでは、有効にしたネットワーク API



(**NetworkPolicy**、**AdminNetworkPolicy**、**BaselineNetworkPolicy**、**UserDefinedNetwork** の分離、マルチキャスト、Egress ファイアウォール) のいずれかがもたらす影響に関する情報を表示できます。

この列で確認できるイベントの kind の例は次のとおりです。

## Network Events の出力例

```
<Dropped_or_Allowed> by <network_event_and_event_name>, direction <Ingress_or_Egress>
```

## 関連情報

- [CLI を使用した機能セットの有効化](#)
- [CLI を使用して OVS サンプルングで OVN-Kubernetes ネットワークトラフィックを確認する](#)

## 8.3. 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。

### 8.3.1. 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。

### 8.3.2. 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** を展開または折りたたみます。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。

### 8.3.2.1. 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、**Export topology view** 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。

## 8.4. 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

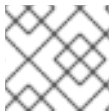
**Quick filters** ドロップダウンメニューのデフォルト値は、**FlowCollector** 設定で定義されます。コンソールからオプションを変更できます。

## 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 defaults** をクリックすると、既存のフィルターが削除され、**FlowCollector** 設定で定義されたフィルターが適用されます。



### 注記

テキスト値の指定規則を理解するには、**Learn More** をクリックしてください。

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

- [クイックフィルターの設定](#)
- [Flow Collector のサンプルリソース](#)



## 第9章 NETWORK OBSERVABILITY アラート

Network Observability Operator は、組み込みのメトリクスと OpenShift Container Platform モニタリングスタックを使用してアラートを提供し、クラスターのネットワークの健全性を迅速に示します。

### 重要

Network Observability アラートは、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、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、以下のリンク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](#)

### 9.1. NETWORK OBSERVABILITY アラートについて

Network Observability には、事前定義済みのアラートがあります。これらのアラートを使用して、OpenShift Container Platform アプリケーションとインフラストラクチャーの健全性とパフォーマンスに関する詳細情報を得ることができます。

事前定義済みのアラートにより、**Network Health** ダッシュボードに、クラスターのネットワークの健全性状態がすぐに表示されます。Prometheus Query Language (PromQL) クエリーを使用してアラートをカスタマイズすることもできます。

デフォルトでは、Network Observability により、有効にした機能に応じたアラートが作成されます。

たとえば、パケットドロップ関連のアラートは、**FlowCollector** カスタムリソース (CR) で **PacketDrop** エージェント機能が有効になっている場合にのみ作成されます。アラートはメトリクスに基づいて作成されるため、有効なアラートに必要なメトリクスが不足している場合、設定に関する警告が表示されることがあります。

これらのメトリクスは、**FlowCollector** CR の **spec.processor.metrics.includeList** オブジェクトで設定できます。

#### 9.1.1. デフォルトのアラートテンプレートのリスト

次のアラートテンプレートはデフォルトでインストールされます。

##### PacketDropsByDevice

デバイス (**/proc/net/dev**) からのパケットドロップ率が高い場合にトリガーされます。

##### PacketDropsByKernel

カーネルによるパケットドロップ率が高い場合にトリガーされます。**PacketDrop** エージェント機能が必要です。

##### IPsecErrors

Network Observability によって IPsec 暗号化エラーが検出されるとトリガーされます。**IPSec** エージェント機能が必要です。

##### NetpolDenied



ネットワークポリシーによって拒否されたトラフィックが Network Observability によって検出されるとトリガーされます。**NetworkEvents** エージェント機能が必要です。

### LatencyHighTrend

Network Observability によって TCP レイテンシーの増加が検出されるとトリガーされます。**FlowRTT** エージェント機能が必要です。

### DNSErrors

Network Observability によって DNS エラーが検出されるとトリガーされます。**DNSTracking** エージェント機能が必要です。

以下は、Network Observability 自体の健全性に関連する運用アラートです。

### NetObservNoFlows

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。

### NetObservLokiError

Loki エラーによりフローがドロップされたときにトリガーされます。

Network Observability のアラートは、設定、拡張、または無効にできます。次のコマンドを実行すると、デフォルトの **netobserv** namespace に生成された **PrometheusRule** リソースを表示できます。

```
$ oc get prometheusrules -n netobserv -oyaml
```

## 9.1.2. Network Health ダッシュボード

Network Observability Operator でアラートが有効な場合、次の 2 つのものが表示されます。

- 新しいアラートが、OpenShift Container Platform Web コンソールの **Observe → Alerting → Alerting rules** タブに表示されます。
- 新しい **Network Health** ダッシュボードが、OpenShift Container Platform Web コンソール → **Observe** に表示されます。

**Network Health** ダッシュボードは、トリガーされたアラートと保留中のアラートの概要を提供し、重大な問題、警告、および軽微な問題に分類します。ルール違反に関するアラートは次のタブに表示されます。

- **Global**: クラスター全体に対するアラートを表示します。
- **Nodes**: ノードごとにルール違反に関するアラートを表示します。
- **Namespaces**: namespace ごとにルール違反に関するアラートを表示します。

リソースカードをクリックすると詳細情報が表示されます。各アラートの横に、省略記号メニューが表示されます。このメニューから、**Network Traffic → Traffic flows** に移動して、選択したリソースの詳細情報を表示できます。

## 9.2. NETWORK OBSERVABILITY のアラート (テクノロジープレビュー) の有効化

Network Observability Operator アラートはテクノロジープレビュー機能です。この機能を使用するには、**FlowCollector** カスタムリソース (CR) でこの機能を有効にしてから、お客様のニーズに合わせてアラートを設定する必要があります。



## 手順

1. **FlowCollector** CR を編集して、実験的なアラートのフラグを **true** に設定します。

```
apiVersion: flows.netobserv.io/v1beta1
kind: FlowCollector
metadata:
  name: flow-collector
spec:
  processor:
    advanced:
      env:
        EXPERIMENTAL_ALERTS_HEALTH: "true"
```

アラートの作成に関しては、従来の方法を引き続き使用できます。詳細は、「アラートの作成」を参照してください。

### 9.2.1. 事前定義済みアラートの設定

Network Observability Operator のアラートは、**FlowCollector** カスタムリソース (CR) の **spec.processor.metrics.alerts** オブジェクト内のアラートテンプレートおよびバリエーションを使用して定義されます。デフォルトのテンプレートとバリエーションをカスタマイズして、柔軟できめ細かなアラートを作成できます。

アラートを有効にすると、OpenShift Container Platform Web コンソールの **Observe** セクションに **Network Health** ダッシュボードが表示されます。

テンプレートごとに、それぞれ固有のしきい値とグループ化設定を持つバリエーションのリストを定義できます。詳細は、「デフォルトのアラートテンプレートのリスト」を参照してください。

以下に例を示します。

```
apiVersion: flows.netobserv.io/v1beta1
kind: FlowCollector
metadata:
  name: flow-collector
spec:
  processor:
    metrics:
      alerts:
        - template: PacketDropsByKernel
          variants:
            # triggered when the whole cluster traffic (no grouping) reaches 10% of drops
            - thresholds:
                critical: "10"
            # triggered when per-node traffic reaches 5% of drops, with gradual severity
            - thresholds:
                critical: "15"
                warning: "10"
                info: "5"
          groupBy: Node
```





## 注記

アラートをカスタマイズすると、そのテンプレートのデフォルト設定が置き換えられます。デフォルト設定を維持する場合は、手動で複製する必要があります。

### 9.2.2. アラートの PromQL 式について

Prometheus Query Language (**PromQL**) のバースクエリーと、それをカスタマイズして特定のニーズに合わせて Network Observability アラートを設定する方法を説明します。

Network Observability の **FlowCollector** カスタムリソース (CR) 内のアラート API は、Prometheus Operator API にマッピングされており、**PrometheusRule** を生成します。次のコマンドを実行すると、デフォルトの **netobserv** namespace 内の **PrometheusRule** を確認できます。

```
$ oc get prometheusrules -n netobserv -oyaml
```

#### 9.2.2.1. 受信トラフィックの急増に関するアラートのクエリー例

この例では、受信トラフィックの急増に関するアラートの **PromQL** バースクエリーパターンを示します。

```
sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}[30m]))
by (DstK8S_Namespace)
```

このクエリーは、過去 30 分間に **openshift-ingress** namespace からいずれかのワークロードの namespace に送信されたバイトレートを計算します。

このクエリーをカスタマイズして、一部のレートのみを保持したり、特定の期間にクエリーを実行したり、最終的なしきい値を設定したりできます。

#### ノイズのフィルタリング

このクエリーに **> 1000** を追加すると、**1 KB/s** を超える観測レートのみが保持され、通信量が少ないコンシューマーからのノイズが除去されます。

```
(sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}[30m])) by (DstK8S_Namespace) > 1000)
```

バイトレートは、**FlowCollector** カスタムリソース (CR) 設定で定義されたサンプリング間隔と相対的な関係にあります。サンプリング間隔が **1:100** の場合、実際のトラフィックは報告されたメトリクスの約 100 倍であると考えられます。

#### 期間の比較

**offset** 修飾子を使用すると、特定の期間に同じクエリーを実行できます。たとえば、**offset 1d** 使用すると 1 日前にクエリーを実行でき、**offset 5h** を使用すると 5 時間前にクエリーを実行できます。

```
sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}[30m] offset 1d)) by (DstK8S_Namespace))
```

**100 \* (<query now> - <query from the previous day>) / <query from the previous day>** という数式を使用すると、前日と比較した増加率を計算できます。今日のバイトレートが前日より低い場合、この値は負になることがあります。

#### 最終的なしきい値



最終的なしきい値を適用すると、目的のパーセンテージに満たない増加分を除外できます。たとえば、**> 100** は 100% 未満の増加分を除外します。

まとめると、**PrometheusRule** の完全な式は次のようになります。

```
...
  expr: |-
    (100 *
      (
        (sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}
[30m])) by (DstK8S_Namespace) > 1000)
        - sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-
ingress"}[30m] offset 1d)) by (DstK8S_Namespace)
      )
      / sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}
[30m] offset 1d)) by (DstK8S_Namespace))
    > 100
```

#### 9.2.2.2. アラートのメタデータフィールド

Network Observability Operator は、モニタリングスタックなど、他の OpenShift Container Platform 機能のコンポーネントを使用して、ネットワークトラフィックの可視性を強化します。詳細は、「モニタリングスタックアーキテクチャー」を参照してください。

アラートを定義するには、いくつかのメタデータを設定する必要があります。このメタデータは、モニタリングスタックの Prometheus および **Alertmanager** サービス、または **Network Health** ダッシュボードによって使用されます。

次の例は、メタデータが設定された **AlertingRule** リソースを示しています。

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: netobserv-alerts
  namespace: openshift-monitoring
spec:
  groups:
  - name: NetObservAlerts
    rules:
    - alert: NetObservIncomingBandwidth
      annotations:
        netobserv_io_network_health: '{"namespaceLabels":
["DstK8S_Namespace"],"threshold":"100","unit":"%", "upperBound":"500"}'
      message: |-
        NetObserv is detecting a surge of incoming traffic: current traffic to {{
$labels.DstK8S_Namespace }} has increased by more than 100% since yesterday.
      summary: "Surge in incoming traffic"
      expr: |-
        (100 *
          (
            (sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}
[30m])) by (DstK8S_Namespace) > 1000)
            - sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-
ingress"}[30m] offset 1d)) by (DstK8S_Namespace)
          )
```



```

    / sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}
[30m] offset 1d)) by (DstK8S_Namespace))
    > 100
    for: 1m
    labels:
      app: netobserv
      netobserv: "true"
      severity: warning

```

ここでは、以下のようになります。

#### spec.groups.rules.alert.labels.netobserv

**true** に設定すると、このアラートが、Network Health ダッシュボードによって検出されるべきアラートとして指定されます。

#### spec.groups.rules.alert.labels.severity

アラートの重大度を指定します。有効な値は、**critical**、**warning**、または **info** です。

**message** アノテーションで定義されている **PromQL** 式からの出力ラベルを活用できます。この例では、結果が **DstK8S\_Namespace** ごとにグループ化されるため、メッセージテキストで **{{ \$labels.DstK8S\_Namespace }}** という式が使用されています。

**netobserv\_io\_network\_health** アノテーションは任意であり、Network Health ページでアラートをどのようにレンダリングするかを制御します。

**netobserv\_io\_network\_health** アノテーションは、次のフィールドで構成される JSON 文字列です。

表9.1 netobserv\_io\_network\_health アノテーションのフィールド

フィールド	型	説明
<b>namespaceLabels</b>	文字列のリスト	namespace を保持する1つ以上のラベル。指定すると、アラートが <b>Namespaces</b> タブに表示されます。
<b>nodeLabels</b>	文字列のリスト	ノード名を保持する1つ以上のラベル。指定すると、アラートが <b>Nodes</b> タブに表示されます。
<b>threshold</b>	文字列	アラートしきい値。 <b>PromQL</b> 式で定義したしきい値と同じ値にする必要があります。
<b>unit</b>	文字列	表示目的でのみ使用されるデータ単位。
<b>upperBound</b>	文字列	閉じた尺度でスコアを計算するために使用される上限値。この上限を超えるメトリクス値は丸められます。
<b>links</b>	オブジェクトのリスト	アラートに応じて表示されるリンクのリスト。各リンクには、 <b>name</b> (表示名) と <b>url</b> が必要です。
<b>trafficLinkFilter</b>	文字列	<b>Network Traffic</b> ページの URL に注入する追加のフィルター。

**namespaceLabels** と **nodeLabels** は相互に排他的です。どちらも指定されていない場合は、**Global** タブにアラートが表示されます。



### 9.2.3. カスタムアラートルールの作成

Prometheus Query Language (**PromQL**) を使用して、特定のネットワークメトリクス (トラフィックの急増など) に基づいてアラートをトリガーするカスタム **AlertingRule** リソースを定義します。

#### 前提条件

- **PromQL** に関する知識。
- OpenShift Container Platform 4.14 以降がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- Network Observability Operator がインストールされている。

#### 手順

1. **AlertingRule** リソースを含む、**custom-alert.yaml** という名前の YAML ファイルを作成します。
2. 次のコマンドを実行して、カスタムアラートルールを適用します。

```
$ oc apply -f custom-alert.yaml
```

#### 検証

1. 次のコマンドを実行して、**PrometheusRule** リソースが **netobserv** namespace に作成されたことを確認します。

```
$ oc get prometheusrules -n netobserv -oyaml
```

出力に、作成した **netobserv-alerts** ルールが含まれているはずです。これにより、リソースが正しく生成されたことを確認できます。

2. OpenShift Container Platform Web コンソールの **Network Health** ダッシュボード → **Observe** をチェックして、ルールがアクティブであることを確認します。

### 9.2.4. 事前定義済みアラートの無効化

**FlowCollector** カスタムリソース (CR) の **spec.processor.metrics.disableAlerts** フィールドで、アラートテンプレートを無効にできます。この設定は、アラートテンプレート名のリストを受け付けます。アラートテンプレート名のリストは、「デフォルトのアラートのリスト」を参照してください。

テンプレートが無効化され、かつ **spec.processor.metrics.alerts** フィールドでオーバーライドされている場合、無効化の設定が優先され、アラートルールは作成されません。

#### 関連情報

- [デフォルトのアラートのリスト](#)
- [Network Observability メトリクスのダッシュボードの表示](#)
- [アラートの作成](#)



- モニタリングスタックアーキテクチャー



## 第10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** コンポーネントを使用して、フローログからメトリクスを生成します。カスタムアラートを設定し、ネットワークアクティビティ分析用のダッシュボードを表示するには、これらのメトリクスを使用します。

### 10.1. 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** タブを使用してネットワーク可観測性メトリクスダッシュボードを表示し、全体的なトラフィックフローとシステムの正常性を監視し、ノード、namespace、所有者、Pod、およびサービスごとにメトリクスをフィルタリングするオプションを指定します。

#### 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. 次のカテゴリーのネットワークトラフィックメトリクスを表示します。各カテゴリーには、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ーのメトリクスを表示します。各カテゴリーには、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
  - プロセッサー
  - 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。

### 10.2. NETWORK OBSERVABILITY メトリクス

**netobserv\_** という接頭辞が付いたネットワーク可観測性メトリックの包括的なリストを確認します。これは、**FlowCollector** リソースで設定でき、トラフィックを監視し、Prometheus アラートを作成するために使用できます。



**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。

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**

### ネットワークイベントメトリクス名

**NetworkEvents** 機能が有効になっている場合、このメトリクスはデフォルトで利用できます。

- **namespace\_network\_policy\_events\_total**

## 10.3. アラートの作成

**Netobserv** ダッシュボードメトリクスに基づいてカスタム **AlertingRule** リソースを作成し、OpenShift Container Platform コンソールでアラートをトリガーする条件を定義します。

### 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。

## 10.4. カスタムメトリクス

**FlowMetric** API を使用して flowlog データからカスタムメトリクスを定義し、ログフィールドを Prometheus ラベルとして使用し、ダッシュボード情報をカスタマイズし、特定のクラスターデータを監視します。

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。

## 10.5. FLOWMETRIC API を使用したカスタムメトリクスの設定

特定のモニタリングニーズを満たすために、フローログフィールドをラベルとしてマッピングしてカスタム Prometheus メトリクスを作成するように **FlowMetric** API を設定します。

### 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。



## 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
spec:
  metricName: cluster_external_ingress_bytes_total
  type: Counter
  valueField: Bytes
  direction: Ingress
  labels:
  [DstK8S_HostName,DstK8S_Namespace,DstK8S_OwnerName,DstK8S_OwnerType]
  filters:
  - field: SrcSubnetLabel
    matchType: Absence

```

- 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。

## 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 ❶
spec:
  metricName: cluster_external_ingress_rtt_seconds
  type: Histogram ❷
  valueField: TimeFlowRttNs
  direction: Ingress
  labels:
  [DstK8S_HostName,DstK8S_Namespace,DstK8S_OwnerName,DstK8S_OwnerType]
  filters:
    - field: SrcSubnetLabel
      matchType: Absence
    - field: TimeFlowRttNs
      matchType: Presence
  divider: "1000000000" ❸
  buckets: [".001", ".005", ".01", ".02", ".03", ".04", ".05", ".075", ".1", ".25", "1"] ❹

```

- 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。
- ❷ **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。
- ❸ 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。
- ❹ 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。

## 10.6. TRAFFIC FLOWS テーブルのネストされたフィールドまたは配列フィールドからメトリクスを作成する

**FlowMetric** カスタムリソースを作成し、**ネットワークイベント** やインターフェイスなどの **Traffic flows** テーブルでネストされたフィールドまたは配列フィールドのメトリクスを生成します。





## 重要

OVN Observability/**NetworkEvents** の表示はテクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、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、以下のリンク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](#)



## 重要

OVN Observability とネットワークイベントの表示および追跡機能は、OpenShift Container Platform 4.17 および 4.18 でのみ利用できます。

次の例は、ネットワークポリシーイベントの **Network events** フィールドからメトリクスを生成する方法を示しています。

### 前提条件

- **NetworkEvents feature** を有効にする。これを行う方法は、関連情報を参照してください。
- ネットワークポリシーが指定されている。

### 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. **FlowMetric** リソースを作成して、次の設定を追加します。

#### ポリシー名と namespace ごとにネットワークポリシーイベントをカウントする設定

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: network-policy-events
  namespace: netobserv
spec:
  metricName: network_policy_events_total
  type: Counter
  labels: [NetworkEvents>Type, NetworkEvents>Namespace, NetworkEvents>Name,
NetworkEvents>Action, NetworkEvents>Direction]
  filters:
    - field: NetworkEvents>Feature
```

1



```

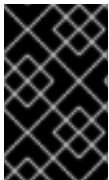
value: acl
flatten: [NetworkEvents]
remap:
  "NetworkEvents>Type": type
  "NetworkEvents>Namespace": namespace
  "NetworkEvents>Name": name
  "NetworkEvents>Direction": direction

```

- ① これらのラベルは、**Traffic flows** テーブルの **Network Events** のネストされたフィールドを表します。各ネットワークイベントには、特定のタイプ、namespace、名前、アクション、方向があります。使用している OpenShift Container Platform バージョンで **NetworkEvents** が利用できない場合は、代わりに **Interfaces** を指定することもできます。
- ② オプション: アイテムのリストを含むフィールドを、個別のアイテムとして表すことを選択できます。
- ③ オプション: Prometheus のフィールドの名前を変更できます。

## 検証

1. Web コンソールで、**Observe** → **Dashboards** に移動し、下にスクロールして **Network Policy** タブを表示します。
2. 作成したメトリクスとネットワークポリシー仕様に基つき、メトリクスのフィルタリングが適用されているはずです。



## 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。

## 10.7. FLOWMETRIC API を使用したカスタムグラフの設定

**FlowMetric** カスタムリソースの charts セクションを定義して、OpenShift Container Platform Web コンソールダッシュボードのカスタムチャートを生成します。

Dashboard メニューで、管理者としてカスタムチャートを表示できます。

## 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。

クラスタ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。

## 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。

②③④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。

## 10.8. FLOWMETRIC API と TCP フラグを使用した SYN フラッディングの検出



カスタム **AlertingRule** および **FlowMetric** 設定をデプロイして TCP フラグを監視し、クラスター上の SYN フラッディング攻撃のリアルタイム検出およびアラートを有効にします。

## 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. **FlowMetric** リソースを作成して、次の設定を追加します。

### TCP フラグを使用して、宛先ホストとリソースごとにフローをカウントする設定

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flows-with-flags-per-destination
spec:
  metricName: flows_with_flags_per_destination_total
  type: Counter
  labels:
    [SrcSubnetLabel,DstSubnetLabel,DstK8S_Name,DstK8S_Type,DstK8S_HostName,DstK8S_Namespace,Flags]
```

### TCP フラグを使用して、ソースホストとリソースごとにフローをカウントする設定

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flows-with-flags-per-source
spec:
  metricName: flows_with_flags_per_source_total
  type: Counter
  labels:
    [DstSubnetLabel,SrcSubnetLabel,SrcK8S_Name,SrcK8S_Type,SrcK8S_HostName,SrcK8S_Namespace,Flags]
```

6. SYN フラッディングを警告するための次の **AlertingRule** リソースをデプロイします。

### SYN フラッディングの AlertingRule

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: netobserv-syn-alerts
  namespace: openshift-monitoring
# ...
spec:
  groups:
    - name: NetObservSYNAAlerts
```



```

rules:
- alert: NetObserv-SYNFlood-in
  annotations:
    message: |-
      {{ $labels.job }}: incoming SYN-flood attack suspected to Host={{
      $labels.DstK8S_HostName}}, Namespace={{ $labels.DstK8S_Namespace }}, Resource={{
      $labels.DstK8S_Name }}. This is characterized by a high volume of SYN-only flows with
      different source IPs and/or ports.
      summary: "Incoming SYN-flood"
      expr: sum(rate(netobserv_flows_with_flags_per_destination_total{Flags="2"}[1m])) by
      (job, DstK8S_HostName, DstK8S_Namespace, DstK8S_Name) > 300 1
      for: 15s
      labels:
        severity: warning
        app: netobserv
- alert: NetObserv-SYNFlood-out
  annotations:
    message: |-
      {{ $labels.job }}: outgoing SYN-flood attack suspected from Host={{
      $labels.SrcK8S_HostName}}, Namespace={{ $labels.SrcK8S_Namespace }}, Resource={{
      $labels.SrcK8S_Name }}. This is characterized by a high volume of SYN-only flows with
      different source IPs and/or ports.
      summary: "Outgoing SYN-flood"
      expr: sum(rate(netobserv_flows_with_flags_per_source_total{Flags="2"}[1m])) by (job,
      SrcK8S_HostName, SrcK8S_Namespace, SrcK8S_Name) > 300 2
      for: 15s
      labels:
        severity: warning
        app: netobserv
# ...

```

- 1 2 この例では、アラートのしきい値は **300** ですが、この値は環境に応じて調整できます。しきい値が低すぎると誤検出が発生する可能性があります。また、しきい値が高すぎると実際の攻撃を見逃す可能性があります。

## 検証

1. Web コンソールで、**Network Traffic** テーブルビューの **Manage Columns** をクリックし、**TCP flags** をクリックします。
2. **Network Traffic** テーブルビューで、**TCP protocol SYN TCPFlag** でフィルタリングします。同じ **byteSize** を持つフローが多数ある場合は、SYN フラッドが発生しています。
3. **Observe** → **Alerting** に移動し、**Alerting Rules** タブを選択します。
4. **netobserv-synflood-in alert** でフィルタリングします。SYN フラッキングが発生すると、アラートが起動するはずですが。

## 関連情報

- [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](#)
- [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11章 NETWORK OBSERVABILITY OPERATOR の監視

OpenShift Container Platform 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します。これにより、システムの安定性を維持し、運用上の問題を迅速に検出できます。

### 11.1. 健全性ダッシュボード

OpenShift Container Platform Web コンソールで Network Observability Operator ヘルスダッシュボードを表示し、Operator およびそのコンポーネントのヘルスステータス、リソースの使用状況、および内部統計を監視します。

メトリクスは、OpenShift Container Platform Web コンソールの **Observe → Dashboards** ページにあります。Network Observability Operator の健全性に関するメトリクスは、次のカテゴリで確認できます。

- 1秒あたりのフロー数
- Sampling
- 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

### 11.2. 健全性アラート

Network Observability Operator によって生成されるヘルスアラートを理解します。これは、Loki の取り込みエラー、ゼロフローの取り込み、またはドロップされた eBPF フローなどの条件が発生した場合にバナーをトリガーします。

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 エージェントが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。

### 11.3. 健全性情報の表示



OpenShift Container Platform Web コンソール内の **Netobserv/Health** ダッシュボードを表示して、Network Observability Operator とそのコンポーネントの正常性ステータスとリソースの使用状況を監視します。

### 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。

#### 11.3.1. ヘルスアラートの無効化

**FlowCollector** リソースを編集し、**spec.processor.metrics.disableAlerts** 仕様を使用して、**NetObservLokiError** や **NetObservNoFlows** などの特定の正常性アラートを無効にします。

### 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. 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] ❶
```

- ❶ 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。

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

Loki メトリクスに基づいてカスタム **AlertingRule** リソースを作成し、Loki の取り込みレート制限に達したときにアラートを監視およびトリガーします。これは、HTTP 429 エラーで示されます。



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

- 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。
- 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
```

- 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。

## 11.5. EBPF エージェントアラートの使用

FlowCollector カスタムリソースの **spec.agent.ebpf.cacheMaxFlows** 値を増やすことにより、eBPF エージェントハッシュマップがいっぱいになったときに発生する **NetObservAgentFlowsDropped** アラートを解決します。

容量リミッターがトリガーされると、アラート **NetObservAgentFlowsDropped** もトリガーされます。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 ❶
```

- ❶ **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12章 リソースのスケジューリング

taint と toleration は、どのノードに特定の Pod を配置するかを制御するのに役立ちます。Network Observability コンポーネントの配置を制御するには、これらの手段をノードセクターとともに使用します。

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。

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

**NodeSelector**、**Tolerations**、**Affinity** を含むスケジューリング仕様を使用して **FlowCollector** リソースを設定し、特定のノードでのネットワーク可観測性コンポーネントのデプロイメントを制御します。

**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

- [taint および 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13章 セカンダリーネットワーク

Network Observability Operator を設定して、**SR-IOV** や **OVN-Kubernetes** などのセカンダリーネットワークからネットワークフローデータを収集し、データをエンリッチすることができます。

### 13.1. 前提条件

- セカンダリーインターフェイスや L2 ネットワークなど、追加のネットワークインターフェイスを備えた OpenShift Container Platform クラスターへのアクセス。

### 13.2. SR-IOV インターフェイストラフィックの監視の設定

**spec.agent.ebpf.privileged** フィールドを **true** に設定して、Single Root I/O Virtualization (SR-IOV) デバイス上のトラフィックを監視するように **FlowCollector** リソースを設定します。これにより、eBPF エージェントは他のネットワーク namespace を監視できるようになります。

eBPF エージェントは、デフォルトで監視されるホストネットワークの namespace に加えて、他のネットワーク名前空間を監視します。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 クラスターにアクセスできる。
- 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

- Web コンソールで、**Operators → Installed Operators** に移動します。
- NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
- cluster** を選択し、**YAML タブ**を選択します。
- 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

- [SR-IOV ネットワークデバイスの設定](#)

## 13.3. 仮想マシン (VM) のセカンダリーネットワークインターフェイスを NETWORK OBSERVABILITY 用に設定する

eBPF エージェントを **特権** モードに設定し、セカンダリーネットワークのインデックスを定義することにより、VM セカンダリーネットワークトラフィックを監視するように **FlowCollector** を設定し、OpenShift Virtualization からのフローのキャプチャーと強化を有効にします。

デフォルトの内部 Pod ネットワークに接続された VM から送信されるネットワークフローは、ネットワークの可観測性によって自動的にキャプチャーされます。

## 手順

1. 次のコマンドを実行して、仮想マシンのランチャー Pod に関する情報を取得します。この情報はステップ 5 で使用します。

```
$ oc get pod virt-launcher-<vm_name>-<suffix> -n <namespace> -o yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    k8s.v1.cni.cncf.io/network-status: |-
      [{
        "name": "ovn-kubernetes",
        "interface": "eth0",
        "ips": [
          "10.129.2.39"
        ],
        "mac": "0a:58:0a:81:02:27",
        "default": true,
        "dns": {}
      },
      {
        "name": "my-vms/l2-network",
        "interface": "podc0f69e19ba2",
        "ips": [
          "10.10.10.15"
        ],
        "mac": "02:fb:f8:00:00:12",
        "dns": {}
      }
    ]
  name: virt-launcher-fedora-aqua-fowl-13-zr2x9
  namespace: my-vms
spec:
  # ...
status:
```



# ...

- ① セカンダリーネットワークの名前。
- ② セカンダリーネットワークのネットワークインターフェイス名。
- ③ セカンダリーネットワークで使用する IP のリスト。
- ④ セカンダリーネットワークに使用される MAC アドレス。

2. Web コンソールで、**Operators** → **Installed Operators** に移動します。
3. **NetObserv Operator** の **Provided APIs** 見出しの下で、**Flow Collector** を選択します。
4. **cluster** を選択し、**YAML** タブを選択します。
5. 追加のネットワーク調査で調べた情報に基づいて **FlowCollector** を設定します。

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
    ebpf:
      privileged: true \ ①
  processor:
    advanced:
      secondaryNetworks:
        - index: ②
          - MAC ③
            name: my-vms/l2-network ④
# ...

```

- ① セカンダリーインターフェイスのフローが収集されるように、eBPF エージェントが **privileged** モードになっていることを確認します。
- ② 仮想マシンランチャー Pod のインデックス作成に使用するフィールドを定義します。セカンダリーインターフェイスのネットワークフローのエンリッチメントを取得するには、**MAC** アドレスをインデックスフィールドとして使用することを推奨します。Pod 間で MAC アドレスが重複している場合は、**IP** や **Interface** などの追加のインデックスフィールドを追加すると、正確なエンリッチメントを取得できます。
- ③ 追加のネットワーク情報に MAC アドレスがある場合は、フィールドリストに **MAC** を追加します。
- ④ **k8s.v1.cni.cncf.io/network-status** アノテーションで見つかったネットワークの名前を指定します。通常は <namespace>/<network\_attachment\_definition\_name> です。

6. 仮想マシンのトラフィックを観測します。
  - a. **Network Traffic** ページに移動します。



- b. **k8s.v1.cni.cncf.io/network-status** アノテーションで見つかった仮想マシンの IP を使用して、**Source** IP でフィルタリングします。
- c. エンリッチする必要がある **Source** フィールドと **Destination** フィールドの両方を表示し、仮想マシンランチャー Pod と仮想マシンインスタンスを所有者として指定します。



## 第14章 NETWORK OBSERVABILITY CLI

### 14.1. NETWORK OBSERVABILITY CLI のインストール

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、Network Observability 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。

#### 14.1.1. Network Observability CLI について

ネットワーク可観測性 CLI (**oc netobserv**)を使用して、ネットワークの問題を迅速にデバッグし、トラブルシューティングします。このツールは、Network Observability Operator をインストールすることなく、即時、ライブの洞察をフローおよびパケットに提供します。

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。



#### 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。

#### 14.1.2. Network Observability CLI のインストール

Network Observability CLI は、ネットワーク可観測性のデバッグおよびトラブルシューティングを迅速に行う軽量な方法です。これは、別途インストールする必要があります。

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、Operator がソフトウェアカタログからインストールされている場合でも、**CLI** を別途インストールする必要があります。



#### 注記

ユーザーは、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。
- **docker** または **podman** をインストールする必要があります。



#### 注記

インストールコマンドの実行には、**podman** または **docker** を使用できます。この手順では **podman** を使用します。



## 手順

1. 次のコマンドを実行して、**Red Hat レジストリー**にログインします。

```
$ podman login registry.redhat.io
```

2. 次のコマンドを実行して、イメージから **oc-netobserv** ファイルを抽出します。

```
$ podman create --name netobserv-cli registry.redhat.io/network-observability/network-observability-cli-rhel9:1.10  
$ podman cp netobserv-cli:/oc-netobserv .  
$ podman rm netobserv-cli
```

3. 次のコマンドを実行して、抽出したファイルをシステムの **PATH** にあるディレクトリー (**/usr/local/bin/** など) に移動します。

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

## 検証

1. **oc netobserv** が利用可能であることを確認します。

```
$ oc netobserv version
```

このコマンドを実行すると、次の例のような結果が生成されるはずです。

```
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](#)
- [CLI Manager Operator のインストール](#)

## 14.2. 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。

### 14.2.1. フローのキャプチャー

ネットワークフローをキャプチャーし、リソースまたはゾーンに基づいて CLI で直接フィルターを適用します。これは、2つの異なるゾーン間の RTT (Round-Trip Time) を視覚的に視覚化するなど、複雑なユースケースを解決するのに役立ちます。

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. **PageUp** キーと **PageDown** キーを使用して、**None**、**Resource**、**Zone**、**Host**、**Owner**、および **all of the above** を切り替えます。
4. 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。
5. 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
}
```



6. 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
```

## 14.2.2. パケットのキャプチャー

Network Observability CLI を使用して、ネットワークパケットを取得します。ターミナルでフィルターを適用し、それらを改良して、正確なリアルタイムのデバッグを行うことができます。

### 前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 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 --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. **PageUp** キーと **PageDown** キーを使用して、**None**、**Resource**、**Zone**、**Host**、**Owner**、および **all of the above** を切り替えます。
4. キャプチャーを停止するには、**Ctrl+C** を押します。



5. 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。

### 14.2.3. メトリクスの取得

サービスモニターを使用して、Prometheus でオンデマンドネットワーク可観測性ダッシュボードを生成します。これにより、ネットワークメトリクスをすばやく表示および分析できます。

#### 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. 次のコマンドを実行して、フィルターを有効にしてメトリクスをキャプチャーします。

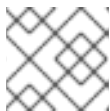
#### 出力例

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

2. ターミナルで提供されているリンクを開いて **NetObserv / On-Demand** ダッシュボードを表示します。

#### URL の例

```
https://console-openshift-console.apps.rosa...openshiftapps.com/monitoring/dashboards/netobserv-cli
```



#### 注記

有効になっていない機能は空のグラフとして表示されます。

### 14.2.4. Network Observability CLI のクリーンアップ

**oc netobserv cleanup** を使用して、Network Observability CLI によってインストールされたすべてのコンポーネントをクラスターから手動で削除します。クライアントはキャプチャ後にこのコマンドを自動的に実行しますが、接続の問題が発生した場合は手動で実行する必要があります。

#### 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](#)



## 14.3. 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。

### 14.3.1. Network Observability CLI の使用

Network Observability CLI (**oc netobserv**) を使用して、コマンドライン引数を渡してさらに分析するためのフローデータ、パケットデータ、メトリクスをキャプチャーしたり、Network Observability Operator がサポートする機能を有効にしたりできます。

#### 14.3.1.1. 構文

**oc netobserv** コマンドの基本構文:

**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。

#### 14.3.1.2. 基本コマンド

表14.1 基本コマンド

コマンド	説明
flows	フロー情報をキャプチャーします。サブコマンドは、「フローキャプチャーのオプション」表を参照してください。
packets	パケットデータをキャプチャーします。サブコマンドは、「パケットキャプチャーのオプション」表を参照してください。
metrics	メトリクスデータをキャプチャーします。サブコマンドは、「メトリクスキャプチャーのオプション」の表を参照してください。
follow	バックグラウンドで実行しているときはコレクターログに従います。
stop	エージェントデーモンセットを削除して収集を停止します。
copy	コレクターが生成したファイルをローカルにコピーします。
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。



### 14.3.1.3. 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>]
```

オプション	説明	デフォルト
--enable_all	すべての eBPF 機能の有効化	false
--enable_dns	DNS 追跡の有効化	false
--enable_ipsec	IPsec トラッキングの有効化	false
--enable_network_events	ネットワークイベントモニタリングの有効化	false
--enable_pkt_translation	パケット変換の有効化	false
--enable_pkt_drop	パケットドロップの有効化	false
--enable_rtt	RTT 追跡の有効化	false
--enable_udn_mapping	ユーザー定義ネットワークマッピングの有効化	false
--get-subnets	サブネット情報の取得	false
--privileged	eBPF エージェント特権モードの強制	auto
--sampling	パケットサンプリング間隔	1
--background	バックグラウンドでの実行	false
--copy	出力ファイルをローカルにコピー	prompt
--log-level	コンポーネントログ	info
--max-time	最大キャプチャー時間	5m
--max-bytes	最大キャプチャーバイト数	50000000 = 50 MB
--action	アクションのフィルタリング	Accept



オプション	説明	デフォルト
--cidr	CIDR のフィルタリング	0.0.0.0/0
--direction	方向のフィルタリング	-
--dport	送信先ポートのフィルタリング	-
--dport_range	送信先ポート範囲のフィルタリング	-
--dports	2つの送信先ポートのどちらかでフィルタリング	-
--drops	ドロップされたパケットのみでフローをフィルタリング	false
--icmp_code	ICMP コードのフィルタリング	-
--icmp_type	ICMP タイプのフィルタリング	-
--node-selector	特定のノードでのキャプチャー	-
--peer_ip	ピア IP のフィルタリング	-
--peer_cidr	ピア CIDR のフィルタリング	-
--port_range	ポート範囲のフィルタリング	-
--port	ポートのフィルタリング	-
--ports	2つのポートのどちらかでフィルタリング	-
--protocol	プロトコルのフィルタリング	-
--query	カスタムクエリーを使用したフローのフィルタリング	-
--sport_range	送信元ポート範囲のフィルタリング	-
--sport	送信元ポートのフィルタリング	-
--sports	2つの送信元ポートのどちらかでフィルタリング	-



オプション	説明	デフォルト
--tcp_flags	TCP フラグのフィルタリング	-
--interfaces	監視するインターフェイスのリスト (コンマ区切り)	-
--exclude_interfaces	除外するインターフェイスのリスト (コンマ区切り)	lo

**PacketDrop および RTT 機能を有効にして TCP プロトコルとポート 49051 でフローキャプチャーを実行する例:**

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

#### 14.3.1.4. パケットキャプチャーのオプション

フィルターを使用すると、フローのキャプチャーと同じようにパケットのキャプチャーデータをフィルタリングできます。パケットドロップ、DNS、RTT、ネットワークイベントなどの特定の機能は、フローおよびメトリクスのキャプチャーでのみ使用できます。

#### oc netobserv packets の構文

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

オプション	説明	デフォルト
--background	バックグラウンドでの実行	false
--copy	出力ファイルをローカルにコピー	prompt
--log-level	コンポーネントログ	info
--max-time	最大キャプチャー時間	5m
--max-bytes	最大キャプチャーバイト数	50000000 = 50 MB
--action	アクションのフィルタリング	Accept
--cidr	CIDR のフィルタリング	0.0.0.0/0
--direction	方向のフィルタリング	-
--dport	送信先ポートのフィルタリング	-



オプション	説明	デフォルト
--dport_range	送信先ポート範囲のフィルタリング	-
--dports	2つの送信先ポートのどちらかでフィルタリング	-
--drops	ドロップされたパケットのみでフローをフィルタリング	false
--icmp_code	ICMP コードのフィルタリング	-
--icmp_type	ICMP タイプのフィルタリング	-
--node-selector	特定のノードでのキャプチャー	-
--peer_ip	ピア IP のフィルタリング	-
--peer_cidr	ピア CIDR のフィルタリング	-
--port_range	ポート範囲のフィルタリング	-
--port	ポートのフィルタリング	-
--ports	2つのポートのどちらかでフィルタリング	-
--protocol	プロトコルのフィルタリング	-
--query	カスタムクエリーを使用したフローのフィルタリング	-
--sport_range	送信元ポート範囲のフィルタリング	-
--sport	送信元ポートのフィルタリング	-
--sports	2つの送信元ポートのどちらかでフィルタリング	-
--tcp_flags	TCP フラグのフィルタリング	-

**TCP プロトコルとポート 49051 でパケットキャプチャーを実行する例:**

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



### 14.3.1.5. メトリクスキャプチャーのオプション

フローキャプチャーと同じように、メトリクスキャプチャーでも機能を有効にしてフィルターを使用できます。生成されたグラフはダッシュボードに適宜表示されます。

#### oc netobserv metrics 構文

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

オプション	説明	デフォルト
--enable_all	すべての eBPF 機能の有効化	false
--enable_dns	DNS 追跡の有効化	false
--enable_ipsec	IPsec トラッキングの有効化	false
--enable_network_events	ネットワークイベントモニタリングの有効化	false
--enable_pkt_translation	パケット変換の有効化	false
--enable_pkt_drop	パケットドロップの有効化	false
--enable_rtt	RTT 追跡の有効化	false
--enable_udn_mapping	ユーザー定義ネットワークマッピングの有効化	false
--get-subnets	サブネット情報の取得	false
--privileged	eBPF エージェント特権モードの強制	auto
--sampling	パケットサンプリング間隔	1
--background	バックグラウンドでの実行	false
--log-level	コンポーネントログ	info
--max-time	最大キャプチャー時間	1h
--action	アクションのフィルタリング	Accept
--cidr	CIDR のフィルタリング	0.0.0.0/0
--direction	方向のフィルタリング	-



オプション	説明	デフォルト
--dport	送信先ポートのフィルタリング	-
--dport_range	送信先ポート範囲のフィルタリング	-
--dports	2つの送信先ポートのどちらかでフィルタリング	-
--drops	ドロップされたパケットのみでフローをフィルタリング	false
--icmp_code	ICMP コードのフィルタリング	-
--icmp_type	ICMP タイプのフィルタリング	-
--node-selector	特定のノードでのキャプチャー	-
--peer_ip	ピア IP のフィルタリング	-
--peer_cidr	ピア CIDR のフィルタリング	-
--port_range	ポート範囲のフィルタリング	-
--port	ポートのフィルタリング	-
--ports	2つのポートのどちらかでフィルタリング	-
--protocol	プロトコルのフィルタリング	-
--query	カスタムクエリーを使用したフローのフィルタリング	-
--sport_range	送信元ポート範囲のフィルタリング	-
--sport	送信元ポートのフィルタリング	-
--sports	2つの送信元ポートのどちらかでフィルタリング	-
--tcp_flags	TCP フラグのフィルタリング	-



オプション	説明	デフォルト
--include_list	生成するメトリクス名のリスト (コンマ区切り)	namespace_flows_total,node_ingress_bytes_total,node_egress_bytes_total,workload_ingress_bytes_total
--interfaces	監視するインターフェイスのリスト (コンマ区切り)	-
--exclude_interfaces	除外するインターフェイスのリスト (コンマ区切り)	lo

### TCP ドロップのメトリクスキャプチャーの実行例

```
$ oc netobserv metrics --enable_pkt_drop --protocol=TCP
```



## 第15章 FLOWCOLLECTOR API リファレンス

**FlowCollector** API は、ネットワークフローを収集するためのデプロイメントを制御および設定するために使用される基礎となるスキーマです。このリファレンスガイドは、このような重要な設定を管理するのに役立ちます。

### 15.1. 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
<b>apiVersion</b>	<b>string</b>	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、 <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources</a> を参照してください。
<b>kind</b>	<b>string</b>	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。CamelCase を使用します。詳細: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds</a>
<b>metadata</b>	<b>object</b>	標準オブジェクトのメタデータ。詳細: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata</a>



プロパティ	型	説明
<b>spec</b>	<b>object</b>	<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>

### 15.1.1. .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**

### 15.1.2. .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
<b>agent</b>	<b>object</b>	フローを抽出するためのエージェント設定。
<b>consolePlugin</b>	<b>object</b>	<b>consolePlugin</b> 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
<b>deploymentModel</b>	<b>string</b>	<p><b>deploymentModel</b> 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"> <li>- <b>フロー</b> プロセッサがエージェントから直接リッスンするようにする <b>direct</b>（デフォルト）。小規模なクラスターで(15 ノード以下の)推奨されます。</li> <li>- <b>Kafka</b> 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。</li> </ul> <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、<a href="https://www.redhat.com/en/topics/integration/what-is-apache-kafka">https://www.redhat.com/en/topics/integration/what-is-apache-kafka</a> を参照してください)。</p>
<b>exporters</b>	<b>array</b>	<p><b>exporters</b> 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。</p>
<b>kafka</b>	<b>object</b>	<p>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、<b>spec.deploymentModel</b> が <b>Kafka</b> の場合です。</p>
<b>loki</b>	<b>object</b>	<p><b>loki</b> (フローストア) のクライアント設定。</p>
<b>namespace</b>	<b>string</b>	<p>Network Observability Pod がデプロイされる namespace。</p>
<b>networkPolicy</b>	<b>object</b>	<p><b>networkPolicy</b> は、Network Observability のコンポーネントを分離するためのネットワークポリシー設定を定義します。</p>



プロパティ	型	説明
<b>processor</b>	<b>object</b>	<b>processor</b> は、エージェントからフローを受信してエンリッチし、メトリクスを生成して Loki 永続化レイヤーや利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。
<b>prometheus</b>	<b>object</b>	<b>prometheus</b> 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。

### 15.1.3. .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
<b>ebpf</b>	<b>object</b>	<b>ebpf</b> は、 <b>spec.agent.type</b> が <b>eBPF</b> 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。
<b>type</b>	<b>string</b>	<b>type</b> [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 <b>eBPF</b> または <b>IPFIX</b> を選択できました。現在、 <b>eBPF</b> 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。

### 15.1.4. .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
<b>advanced</b>	<b>object</b>	<b>advanced</b> 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、 <b>GOGC</b> や <b>GOMAXPROCS</b> 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。このセクションでは、デフォルトの Linux ケイパビリティをオーバーライドすることもできます。
<b>cacheActiveTimeout</b>	<b>string</b>	<b>cacheActiveTimeout</b> 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。 <b>cacheMaxFlows</b> と <b>cacheActiveTimeout</b> 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。
<b>cacheMaxFlows</b>	<b>integer</b>	<b>cacheMaxFlows</b> 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。 <b>cacheMaxFlows</b> と <b>cacheActiveTimeout</b> 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。
<b>excludeInterfaces</b>	<b>array (string)</b>	<b>excludeInterfaces</b> 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。 <b>/br-/</b> 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。
<b>features</b>	<b>array (string)</b>	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



プロパティ	型	説明
		<p>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"> <li>- <b>PacketDrop</b>: 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には、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 エージェント Pod を <b>spec.agent.ebpf.privileged</b> を介して特権付き Pod として実行する必要があります。</li> <li>- <b>DNSTracking</b>: DNS 追跡機能を有効にします。</li> <li>- <b>FlowRTT</b>: 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。</li> <li>- <b>NetworkEvents</b>: フローやネットワークポリシーの相関付けなどのネットワークイベント監視機能を有効にします。この機能を使用するには、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 エージェント Pod を <b>spec.agent.ebpf.privileged</b> を介して特権付き Pod として実行する必要があります。Observability 機能を備えた OVN-Kubernetes ネットワークプラグインを使用する必要があります。重要: この機能はテクノロジープレビューとして提供されています。</li> <li>- <b>PacketTranslation</b>: Service NAT などのパケット変換情報を使用してフローをエンリッチできるようにします。</li> <li>- <b>EbpfManager</b>: [サポートされていません (*)]。eBPF Manager を使用して、Network Observability eBPF プログラムを管理します。前提条件として、eBPF Manager Operator (またはアップストリームの bpfman Operator) がインストールされている必要があります。</li> <li>- <b>UDNMapping</b>: ユーザー定義ネットワーク (UDN) へのインターフェイスのマッピングを有効</li> </ul>



プロパティ	型	説明
		<p>にします。</p> <p>この機能を使用するには、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 エージェント Pod を <b>spec.agent.ebpf.privileged</b> を介して特権付き Pod として実行する必要があります。Observability 機能を備えた OVN-Kubernetes ネットワークプラグインを使用する必要があります。</p>
<b>flowFilter</b>	<b>object</b>	<p>- <b>IPSec</b>: IPsec 暗号化を使用して <b>flowFilter</b> はフローを追跡し、リングに関する eBPF エージェント設定を定義します。</p>
<b>imagePullPolicy</b>	<b>string</b>	<p><b>imagePullPolicy</b> は、上で定義したイメージの Kubernetes プルポリシーです。</p>
<b>interfaces</b>	<b>array (string)</b>	<p><b>interfaces</b> 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 <b>excludeInterfaces</b> 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。<b>/br-/</b> など、スラッシュで囲まれたエントリー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。</p>
<b>kafkaBatchSize</b>	<b>integer</b>	<p><b>kafkaBatchSize</b> 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。</p>
<b>logLevel</b>	<b>string</b>	<p><b>logLevel</b> は、Network Observability eBPF Agent のログレベルを定義します。</p>
<b>metrics</b>	<b>object</b>	<p><b>metrics</b> 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。</p>



プロパティ	型	説明
<b>privileged</b>	<b>boolean</b>	eBPF Agent コンテナの特権モード。 <b>true</b> に設定すると、エージェントが、セカンダリーインターフェイスからのトラフィックも含め、より多くのトラフィックをキャプチャーできるようになります。無視されるか <b>false</b> に設定されていると、Operator がコンテナに詳細なケイパビリティ (BPF、PERFMON、NET_ADMIN) を設定します。パケットドロップの追跡 ( <b>features</b> 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。
<b>resources</b>	<b>object</b>	<b>resources</b> 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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。
<b>sampling</b>	<b>integer</b>	eBPF プローブのサンプリング間隔。100 を指定すると、100 分の 1 のパケットが送信されます。0 または 1 を指定すると、すべてのパケットがサンプリングされます。

### 15.1.5. .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。このセクションでは、デフォルトの Linux ケイパビリティをオーバーライドすることもできます。

#### 型

**object**

プロパティ	型	説明
<b>capOverride</b>	<b>array (string)</b>	特権モードで実行されていない場合に、Linux ケイパビリティをオーバーライドします。デフォルトのケイパビリティは、BPF、PERFMON、NET_ADMIN です。



プロパティ	型	説明
<b>env</b>	<b>object (string)</b>	<b>env</b> 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。 <b>GOGC</b> や <b>GOMAXPROCS</b> 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。
<b>scheduling</b>	<b>object</b>	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。

### 15.1.6. .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
<b>affinity</b>	<b>object</b>	指定した場合、Pod のスケジューリング制約。ドキュメントは、 <a href="https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling">https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling</a> を参照してください。
<b>nodeSelector</b>	<b>object (string)</b>	<b>nodeSelector</b> 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。ドキュメントは、 <a href="https://kubernetes.io/docs/concepts/configuration/assign-pod-node/">https://kubernetes.io/docs/concepts/configuration/assign-pod-node/</a> を参照してください。



プロパティ	型	説明
<b>priorityClassName</b>	<b>string</b>	指定した場合、Pod の優先度を示します。ドキュメントは、 <a href="https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption">https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption</a> 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。
<b>tolerations</b>	<b>array</b>	<b>tolerations</b> は、一致する taint を持つノードに Pod がスケジューリングできるようにする toleration のリストです。ドキュメントは、 <a href="https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling">https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling</a> を参照してください。

### 15.1.7. .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。ドキュメントは、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

#### 型

**object**

### 15.1.8. .spec.agent.ebpf.advanced.scheduling.tolerations

#### 説明

**tolerations** は、一致する taint を持つノードに Pod がスケジューリングできるようにする toleration のリストです。ドキュメントは、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

#### 型

**array**

### 15.1.9. .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
<b>action</b>	<b>string</b>	<b>action</b> 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。使用可能なオプションは、 <b>Accept</b> (デフォルト) と <b>Reject</b> です。
<b>cidr</b>	<b>string</b>	<b>cidr</b> 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、 <b>10.10.10.0/24</b> または <b>100:100:100:100::/64</b> です。
<b>destPorts</b>	<b>integer-or-string</b>	<b>destPorts</b> 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 (任意)。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、 <b>destPorts: 80</b> 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、 <b>destPorts: "80-100"</b> です。2つのポートをフィルタリングするには、文字列形式の "port1,port2" を使用します。たとえば、 <b>ports: "80,100"</b> です。
<b>direction</b>	<b>string</b>	<b>direction</b> 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 (任意)。使用可能なオプションは <b>Ingress</b> と <b>Egress</b> です。
<b>enable</b>	<b>boolean</b>	eBPF フローのフィルタリング機能を有効にするには、 <b>enable</b> を <b>true</b> に設定します。
<b>icmpCode</b>	<b>integer</b>	<b>icmpCode</b> 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 (任意)。
<b>icmpType</b>	<b>integer</b>	<b>icmpType</b> 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 (任意)。



プロパティ	型	説明
<b>peerCIDR</b>	<b>string</b>	<b>peerCIDR</b> は、フローをフィルタリングする Peer IP CIDR を定義します。たとえば、 <b>10.10.10.0/24</b> または <b>100:100:100:100::/64</b> です。
<b>peerIP</b>	<b>string</b>	<b>peerIP</b> は、フローのフィルタリングに使用するリモート IP アドレスを定義します (任意)。たとえば、 <b>10.10.10.10</b> です。
<b>pktDrops</b>	<b>boolean</b>	<b>pktDrops</b> は、パケットドロップを含むフローのみをフィルタリングします (任意)。
<b>ports</b>	<b>integer-or-string</b>	<b>ports</b> 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 (任意)。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、 <b>ports: 80</b> 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、 <b>ports: "80-100"</b> です。2つのポートをフィルタリングするには、文字列形式の "port1,port2" を使用します。たとえば、 <b>ports: "80,100"</b> です。
<b>protocol</b>	<b>string</b>	<b>protocol</b> 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 (任意)。使用可能なオプションは、 <b>TCP</b> 、 <b>UDP</b> 、 <b>ICMP</b> 、 <b>ICMP v6</b> 、 <b>SCTP</b> です。
<b>rules</b>	<b>array</b>	<b>rules</b> は、eBPF エージェントのフィルタリングルールのリストを定義します。フィルタリングが有効になっている場合、どのルールにも一致しないフローはデフォルトで拒否されます。デフォルトを変更するには、すべてを受け入れるルール ( <b>{ action: "Accept", cidr: "0.0.0.0/0" }</b> ) を定義してから拒否ルールで調整します。



プロパティ	型	説明
<b>sampling</b>	<b>integer</b>	<b>sampling</b> は、マッチしたパケットのサンプリング間隔であり、 <b>spec.agent.ebpf.sampling</b> で定義されたグローバルサンプリングをオーバーライドします。
<b>sourcePorts</b>	<b>integer-or-string</b>	<b>sourcePorts</b> 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 (任意)。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、 <b>sourcePorts: 80</b> 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、 <b>sourcePorts: "80-100"</b> です。2つのポートをフィルタリングするには、文字列形式の "port1,port2" を使用します。たとえば、 <b>ports: "80,100"</b> です。
<b>tcpFlags</b>	<b>string</b>	<b>tcpFlags</b> は、フローのフィルタリングに使用するための TCP フラグを定義します (任意)。標準のフラグ (RFC-9293) に加えて、 <b>SYN-ACK</b> 、 <b>FIN-ACK</b> 、 <b>RST-ACK</b> の3つの組み合わせのいずれかを使用してフィルタリングすることもできます。

### 15.1.10. .spec.agent.ebpf.flowFilter.rules

#### 説明

**rules** は、eBPF エージェントのフィルタリングルールをリストを定義します。フィルタリングが有効になっている場合、どのルールにも一致しないフローはデフォルトで拒否されます。デフォルトを変更するには、すべてを受け入れるルール (**{ action: "Accept", cidr: "0.0.0.0/0" }**) を定義してから拒否ルールで調整します。

#### 型

**array**

### 15.1.11. .spec.agent.ebpf.flowFilter.rules[]

#### 説明

**EBPFFlowFilterRule** は、フローフィルタリングルールに関する 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
<b>action</b>	<b>string</b>	<b>action</b> 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。使用可能なオプションは、 <b>Accept</b> (デフォルト) と <b>Reject</b> です。
<b>cidr</b>	<b>string</b>	<b>cidr</b> 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、 <b>10.10.10.0/24</b> または <b>100:100:100:100::/64</b> です。
<b>destPorts</b>	<b>integer-or-string</b>	<b>destPorts</b> 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 (任意)。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、 <b>destPorts: 80</b> 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、 <b>destPorts: "80-100"</b> です。2つのポートをフィルタリングするには、文字列形式の "port1,port2" を使用します。たとえば、 <b>ports: "80,100"</b> です。
<b>direction</b>	<b>string</b>	<b>direction</b> 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 (任意)。使用可能なオプションは <b>Ingress</b> と <b>Egress</b> です。
<b>icmpCode</b>	<b>integer</b>	<b>icmpCode</b> 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 (任意)。
<b>icmpType</b>	<b>integer</b>	<b>icmpType</b> 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 (任意)。



プロパティ	型	説明
<b>peerCIDR</b>	<b>string</b>	<b>peerCIDR</b> は、フローをフィルタリングする Peer IP CIDR を定義します。たとえば、 <b>10.10.10.0/24</b> または <b>100:100:100:100::/64</b> です。
<b>peerIP</b>	<b>string</b>	<b>peerIP</b> は、フローのフィルタリングに使用するリモート IP アドレスを定義します (任意)。たとえば、 <b>10.10.10.10</b> です。
<b>pktDrops</b>	<b>boolean</b>	<b>pktDrops</b> は、パケットドロップを含むフローのみをフィルタリングします (任意)。
<b>ports</b>	<b>integer-or-string</b>	<b>ports</b> 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 (任意)。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、 <b>ports: 80</b> 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、 <b>ports: "80-100"</b> です。2つのポートをフィルタリングするには、文字列形式の "port1,port2" を使用します。たとえば、 <b>ports: "80,100"</b> です。
<b>protocol</b>	<b>string</b>	<b>protocol</b> 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 (任意)。使用可能なオプションは、 <b>TCP</b> 、 <b>UDP</b> 、 <b>ICMP</b> 、 <b>ICMP v6</b> 、 <b>SCTP</b> です。
<b>sampling</b>	<b>integer</b>	<b>sampling</b> は、マッチしたパケットのサンプリング間隔であり、 <b>spec.agent.ebpf.sampling</b> で定義されたグローバルサンプリングをオーバーライドします。



プロパティ	型	説明
<b>sourcePorts</b>	<b>integer-or-string</b>	<p><b>sourcePorts</b> 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 (任意)。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、<b>sourcePorts: 80</b> です。</p> <p>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、<b>sourcePorts: "80-100"</b> です。2つのポートをフィルタリングするには、文字列形式の "port1,port2" を使用します。たとえば、<b>ports: "80,100"</b> です。</p>
<b>tcpFlags</b>	<b>string</b>	<p><b>tcpFlags</b> は、フローのフィルタリングに使用するための TCP フラグを定義します (任意)。標準のフラグ (RFC-9293) に加えて、<b>SYN-ACK</b>、<b>FIN-ACK</b>、<b>RST-ACK</b> の3つの組み合わせのいずれかを使用してフィルタリングすることもできます。</p>

### 15.1.12. .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
<b>disableAlerts</b>	<b>array (string)</b>	<p><b>disableAlerts</b> 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><b>NetObservDroppedFlows</b> は、eBPF エージェントでパケットまたはフローが欠落している場合にトリガーされます。たとえば、eBPF の HashMap がビジー状態または満杯の場合や、容量制限がトリガーされている場合などです。</p>



プロパティ	型	説明
<b>enable</b>	<b>boolean</b>	eBPF エージェントのメトリクス収集を無効にするには、 <b>enable</b> を <b>false</b> に設定します。これは、デフォルトで有効になっています。
<b>server</b>	<b>object</b>	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。

### 15.1.13. .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
<b>port</b>	<b>integer</b>	メトリクスサーバーの HTTP ポート。
<b>tls</b>	<b>object</b>	TLS 設定。

### 15.1.14. .spec.agent.ebpf.metrics.server.tls

#### 説明

TLS 設定。

#### 型

**object**

#### 必須

- **type**

プロパティ	型	説明
-------	---	----



プロパティ	型	説明
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>providedCaFile</b> フィールドが無視されます。
<b>provided</b>	<b>object</b>	<b>type</b> が <b>Provided</b> に設定されている場合の TLS 設定。
<b>providedCaFile</b>	<b>object</b>	<b>type</b> が <b>Provided</b> に設定されている場合の CA ファイルへの参照。
<b>type</b>	<b>string</b>	TLS 設定のタイプを選択します。  - <b>Disabled</b> (デフォルト) は、エンドポイントに TLS を設定しません。- <b>Provided</b> 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 (*)]。- <b>Auto</b> 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。

### 15.1.15. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.16. .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
<b>file</b>	<b>string</b>	config map またはシークレット内のファイル名。
<b>name</b>	<b>string</b>	ファイルを含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	ファイル参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.17. .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
<b>limits</b>	<b>integer-or-string</b>	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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。
<b>requests</b>	<b>integer-or-string</b>	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。コンテナで 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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。

## 15.1.18. .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
<b>advanced</b>	<b>object</b>	<b>advanced</b> 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、 <b>GOGC</b> や <b>GOMAXPROCS</b> 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
<b>autoscaler</b>	<b>object</b>	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 <b>autoscaler</b> 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。
<b>enable</b>	<b>boolean</b>	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。
<b>imagePullPolicy</b>	<b>string</b>	<b>imagePullPolicy</b> は、上で定義したイメージの Kubernetes プルポリシーです。
<b>logLevel</b>	<b>string</b>	コンソールプラグインバックエンドの <b>logLevel</b> 。
<b>portNaming</b>	<b>object</b>	<b>portNaming</b> は、ポートからサービス名への変換の設定を定義します。
<b>quickFilters</b>	<b>array</b>	<b>quickFilters</b> 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。
<b>replicas</b>	<b>integer</b>	<b>replicas</b> は、開始するレプリカ (Pod) の数を定義します。
<b>resources</b>	<b>object</b>	<b>resources</b> (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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。

### 15.1.19. .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
<b>args</b>	<b>array (string)</b>	<b>args</b> 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。
<b>env</b>	<b>object (string)</b>	<b>env</b> 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。 <b>GOGC</b> や <b>GOMAXPROCS</b> 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。
<b>port</b>	<b>integer</b>	<b>port</b> 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。
<b>register</b>	<b>boolean</b>	<b>register</b> を <b>true</b> 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。 <b>false</b> に設定した場合でも、 <b>oc patch console.operator.openshift.io cluster --type='json' -p '{"op": "add", "path": "/spec/plugins/-", "value": "netobserv-plugin"}'</b> コマンドで <code>console.operator.openshift.io/cluster</code> を編集することにより、手動で登録できます。
<b>scheduling</b>	<b>object</b>	<b>scheduling</b> 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。



## 15.1.20. .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
<b>affinity</b>	<b>object</b>	指定した場合、Pod のスケジューリング制約。ドキュメントは、 <a href="https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling">https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling</a> を参照してください。
<b>nodeSelector</b>	<b>object (string)</b>	<b>nodeSelector</b> 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。ドキュメントは、 <a href="https://kubernetes.io/docs/concepts/configuration/assign-pod-node/">https://kubernetes.io/docs/concepts/configuration/assign-pod-node/</a> を参照してください。
<b>priorityClassName</b>	<b>string</b>	指定した場合、Pod の優先度を示します。ドキュメントは、 <a href="https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption">https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption</a> 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。
<b>tolerations</b>	<b>array</b>	<b>tolerations</b> は、一致する taint を持つノードに Pod がスケジュールできるようにする toleration のリストです。ドキュメントは、 <a href="https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling">https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling</a> を参照してください。

## 15.1.21. .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。ドキュメントは、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

**object**

### 15.1.22. .spec.consolePlugin.advanced.scheduling.tolerations

説明

**tolerations** は、一致する taint を持つノードに Pod がスケジューリングできるようにする toleration のリストです。ドキュメントは、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

型

**array**

### 15.1.23. .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**

### 15.1.24. .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
<b>enable</b>	<b>boolean</b>	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。
<b>portNames</b>	<b>object (string)</b>	<b>portNames</b> 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: <b>portNames: {"3100": "loki"}</b> )。

### 15.1.25. .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**



## 15.1.26. .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
<b>default</b>	<b>boolean</b>	<b>default</b> 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。
<b>filter</b>	<b>object (string)</b>	<b>filter</b> 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: <b>filter: {"src_namespace": "namespace1,namespace2"}</b> )。
<b>name</b>	<b>string</b>	コンソールに表示されるフィルターの名前

## 15.1.27. .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
<b>limits</b>	<b>integer-or-string</b>	<b>limits</b> 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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。



プロパティ	型	説明
<b>requests</b>	<b>integer-or-string</b>	requests は、必要なコンピュータリソースの最小量を示します。コンテナで 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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。

### 15.1.28. .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**

### 15.1.29. .spec.exporters[]

#### 説明

**FlowCollectorExporter** は、エンリッチ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
<b>ipfix</b>	<b>object</b>	エンリッチされた IPFIX フローの送信先の IP アドレスやポートなどの IPFIX 設定。
<b>kafka</b>	<b>object</b>	エンリッチされたフローの送信先のアドレスやトピックなどの Kafka 設定。
<b>openTelemetry</b>	<b>object</b>	エンリッチされたログやメトリクスの送信先の IP アドレスやポートなどの OpenTelemetry 設定。



プロパティ	型	説明
<b>type</b>	<b>string</b>	<b>type</b> 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、 <b>Kafka</b> 、 <b>IPFIX</b> 、 <b>OpenTelemetry</b> です。

### 15.1.30. .spec.exporters[].ipfix

#### 説明

エンリッチされた IPFIX フローの送信先の 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
<b>targetHost</b>	<b>string</b>	IPFIX 外部レシーバーのアドレス。
<b>targetPort</b>	<b>integer</b>	IPFIX 外部レシーバー用のポート。
<b>transport</b>	<b>string</b>	IPFIX 接続に使用されるトランスポートプロトコル ( <b>TCP</b> または <b>UDP</b> )。デフォルトは <b>TCP</b> です。

### 15.1.31. .spec.exporters[].kafka

#### 説明

エンリッチされたフローの送信先の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
<b>address</b>	<b>string</b>	Kafka サーバーのアドレス
<b>sasl</b>	<b>object</b>	SASL 認証の設定。[サポート対象外 (*)]。
<b>tls</b>	<b>object</b>	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。
<b>topic</b>	<b>string</b>	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。

### 15.1.32. .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
<b>clientIDReference</b>	<b>object</b>	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
<b>clientSecretReference</b>	<b>object</b>	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
<b>type</b>	<b>string</b>	使用する SASL 認証のタイプ。SASL を使用しない場合は <b>Disabled</b>

### 15.1.33. .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
<b>file</b>	<b>string</b>	config map またはシークレット内のファイル名。
<b>name</b>	<b>string</b>	ファイルを含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	ファイル参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.34. .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
<b>file</b>	<b>string</b>	config map またはシークレット内のファイル名。
<b>name</b>	<b>string</b>	ファイルを含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。



プロパティ	型	説明
<b>type</b>	<b>string</b>	ファイル参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.35. .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
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

### 15.1.36. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.37. .spec.exporters[].kafka.tls.userCert

#### 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

#### 型

**object**

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.38. .spec.exporters[].openTelemetry

#### 説明

エンリッチされたログやメトリクスを送信先の IP アドレスやポートなどの OpenTelemetry 設定。

#### 型

**object**

#### 必須

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

プロパティ	型	説明
<b>fieldsMapping</b>	<b>array</b>	OpenTelemetry 準拠の形式にマッピングされるカスタムフィールド。デフォルトでは、Network Observability の形式の提案 ( <a href="https://github.com/rhobs/observability-data-model/blob/main/network-observability.md#format-proposal">https://github.com/rhobs/observability-data-model/blob/main/network-observability.md#format-proposal</a> ) が使用されます。L3 または L4 エンリッチ化ネットワークログは、現在、受け入れられている標準が存在しないため、デフォルトを独自の形式で自由にオーバーライドできます。
<b>headers</b>	<b>object (string)</b>	メッセージに追加するヘッダー (任意)。
<b>logs</b>	<b>object</b>	ログの OpenTelemetry 設定。



プロパティ	型	説明
<b>metrics</b>	<b>object</b>	メトリクスの OpenTelemetry 設定。
<b>protocol</b>	<b>string</b>	OpenTelemetry 接続のプロトコル。使用可能なオプションは <b>http</b> と <b>grpc</b> です。
<b>targetHost</b>	<b>string</b>	OpenTelemetry レシーバーのアドレス。
<b>targetPort</b>	<b>integer</b>	OpenTelemetry レシーバーのポート。
<b>tls</b>	<b>object</b>	TLS クライアント設定。

### 15.1.39. .spec.exporters[].openTelemetry.fieldsMapping

#### 説明

OpenTelemetry 準拠の形式にマッピングされるカスタムフィールド。デフォルトでは、Network Observability の形式の提案 (<https://github.com/rhobs/observability-data-model/blob/main/network-observability.md#format-proposal>) が使用されます。L3 または L4 エンリッチ化ネットワークログは、現在、受け入れられている標準が存在しないため、デフォルトを独自の形式で自由にオーバーライドできます。

#### 型

**array**

### 15.1.40. .spec.exporters[].openTelemetry.fieldsMapping[]

#### 説明

#### 型

**object**

プロパティ	型	説明
<b>input</b>	<b>string</b>	
<b>multiplier</b>	<b>integer</b>	
<b>output</b>	<b>string</b>	

### 15.1.41. .spec.exporters[].openTelemetry.logs

#### 説明

ログの OpenTelemetry 設定。



## 型

object

プロパティ	型	説明
<b>enable</b>	<b>boolean</b>	ログを OpenTelemetry レシーバーに送信するには、 <b>enable</b> を <b>true</b> に設定します。

## 15.1.42. .spec.exporters[].openTelemetry.metrics

## 説明

メトリクスの OpenTelemetry 設定。

## 型

object

プロパティ	型	説明
<b>enable</b>	<b>boolean</b>	メトリクスを OpenTelemetry レシーバーに送信するには、 <b>enable</b> を <b>true</b> に設定します。
<b>pushTimeInterval</b>	<b>string</b>	メトリクスをコレクターに送信する頻度を指定します。

## 15.1.43. .spec.exporters[].openTelemetry.tls

## 説明

TLS クライアント設定。

## 型

object

プロパティ	型	説明
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。



プロパティ	型	説明
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

#### 15.1.44. .spec.exporters[].openTelemetry.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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

#### 15.1.45. .spec.exporters[].openTelemetry.tls.userCert

##### 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。



## 型

## object

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

## 15.1.46. .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
<b>address</b>	<b>string</b>	Kafka サーバーのアドレス



プロパティ	型	説明
<b>sasl</b>	<b>object</b>	SASL 認証の設定。[サポート対象外 (*)]。
<b>tls</b>	<b>object</b>	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。
<b>topic</b>	<b>string</b>	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。

### 15.1.47. .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
<b>clientIDReference</b>	<b>object</b>	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
<b>clientSecretReference</b>	<b>object</b>	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
<b>type</b>	<b>string</b>	使用する SASL 認証のタイプ。SASL を使用しない場合は <b>Disabled</b>

### 15.1.48. .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
-------	---	----



プロパティ	型	説明
<b>file</b>	<b>string</b>	config map またはシークレット内のファイル名。
<b>name</b>	<b>string</b>	ファイルを含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	ファイル参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.49. .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
<b>file</b>	<b>string</b>	config map またはシークレット内のファイル名。
<b>name</b>	<b>string</b>	ファイルを含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。



プロパティ	型	説明
<b>type</b>	<b>string</b>	ファイル参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.50. .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
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

### 15.1.51. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.52. .spec.kafka.tls.userCert

#### 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

#### 型

**object**

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.53. .spec.loki

#### 説明

**loki** (フローストア) のクライアント設定。

#### 型

**object**

#### 必須

- **mode**

プロパティ	型	説明
<b>advanced</b>	<b>object</b>	<b>advanced</b> 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
<b>enable</b>	<b>boolean</b>	Loki にフローを保存するには、 <b>enable</b> を <b>true</b> 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 ( <b>spec.prometheus.queries</b> 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。
<b>lokiStack</b>	<b>object</b>	<b>LokiStack</b> 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。
<b>manual</b>	<b>object</b>	<b>Manual</b> 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。
<b>microservices</b>	<b>object</b>	<b>Microservices</b> 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 ( <a href="https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode">https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode</a> ) 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
<b>mode</b>	<b>string</b>	<p><b>mode</b> 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"> <li>- Loki が Loki Operator を使用して管理されている場合は、<b>LokiStack</b> を使用します。</li> <li>- 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、<b>Monolithic</b> を使用します。</li> <li>- 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、<b>Microservices</b> を使用します。</li> <li>- 上記のオプションが、いずれもお使いのセットアップに合わない場合は、<b>Manual</b> を使用します。</li> </ul>
<b>monolithic</b>	<b>object</b>	<p><b>Monolithic</b> 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 (<a href="https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode">https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode</a>) 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>
<b>readTimeout</b>	<b>string</b>	<p><b>readTimeout</b> 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>
<b>writeBatchSize</b>	<b>integer</b>	<p><b>writeBatchSize</b> 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>
<b>writeBatchWait</b>	<b>string</b>	<p><b>writeBatchWait</b> は、Loki バッチを送信するまでに待機する最大時間です。</p>
<b>writeTimeout</b>	<b>string</b>	<p><b>writeTimeout</b> 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。</p>



プロパティ	型	説明
-------	---	----

### 15.1.54. .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
<b>excludeLabels</b>	<b>array (string)</b>	<b>excludeLabels</b> は、Loki ラベルのリストから除外するフィールドのリストです [サポートされていません (*)]。
<b>staticLabels</b>	<b>object (string)</b>	<b>staticLabels</b> 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プです。
<b>writeMaxBackoff</b>	<b>string</b>	<b>writeMaxBackoff</b> 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。
<b>writeMaxRetries</b>	<b>integer</b>	<b>writeMaxRetries</b> は、Loki クライアント接続の最大再試行回数です。
<b>writeMinBackoff</b>	<b>string</b>	<b>writeMinBackoff</b> 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。

### 15.1.55. .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**

#### 必須

- **name**



プロパティ	型	説明
<b>name</b>	<b>string</b>	使用する既存の LokiStack リソースの名前。
<b>namespace</b>	<b>string</b>	この <b>LokiStack</b> リソースが配置される namespace。省略した場合は、 <b>spec.namespace</b> と同じであるとみなされます。

### 15.1.56. .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
<b>authToken</b>	<b>string</b>	<p><b>authToken</b> 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"> <li>- <b>Disabled</b> の場合、リクエストとともにトークンが送信されません。</li> <li>- <b>Forward</b> の場合、認可のためにユーザートークンを転送します。</li> <li>- <b>Host</b> [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。</li> </ul> <p>Loki Operator を使用する場合は、<b>Forward</b> に設定する必要があります。</p>
<b>ingesterUrl</b>	<b>string</b>	<p><b>ingesterUrl</b> 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 <b>network</b> 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: <a href="https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network">https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network</a>)。</p>



プロパティ	型	説明
<b>querierUrl</b>	<b>string</b>	<b>querierUrl</b> 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 <b>network</b> 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: <a href="https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network">https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network</a> )。
<b>statusTls</b>	<b>object</b>	Loki ステータス URL の TLS クライアント設定。
<b>statusUrl</b>	<b>string</b>	<b>statusUrl</b> は、Loki クエリアー URL と異なる場合に備えて、Loki <b>/ready</b> 、 <b>/metrics</b> 、 <b>/config</b> エンドポイントのアドレスを指定します。空の場合、 <b>querierUrl</b> 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: <a href="https://loki-query-frontend-http.netobserv.svc:3100/">https://loki-query-frontend-http.netobserv.svc:3100/</a> ) に設定します。 <b>statusTls</b> 設定は、 <b>statusUrl</b> が設定されている場合に使用されます。
<b>tenantID</b>	<b>string</b>	<b>tenantID</b> は、各リクエストのテナントを識別する Loki <b>X-Scope-OrgID</b> 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 <b>network</b> に設定します。
<b>tls</b>	<b>object</b>	Loki URL の TLS クライアント設定。

### 15.1.57. .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
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

### 15.1.58. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.59. .spec.loki.manual.statusTls.userCert

#### 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方 TLS を使用する場合は、このプロパティを無視できます。

#### 型

**object**

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.60. .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
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

### 15.1.61. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

## 15.1.62. .spec.loki.manual.tls.userCert

## 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

## 型

## object

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.63. .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
<b>ingesterUrl</b>	<b>string</b>	<b>ingesterUrl</b> 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。
<b>querierUrl</b>	<b>string</b>	<b>querierURL</b> は、Loki クエリアーサービスのアドレスを指定します。



プロパティ	型	説明
<b>tenantID</b>	<b>string</b>	<b>tenantID</b> は、各リクエストのテナントを識別する <b>Loki X-Scope-OrgID</b> ヘッダーです。
<b>tls</b>	<b>object</b>	Loki URL の TLS クライアント設定。

#### 15.1.64. .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
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方方向 TLS を使用する場合は、このプロパティを無視できます。

#### 15.1.65. .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
-------	---	----



プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.66. .spec.loki.microservices.tls.userCert

#### 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方 TLS を使用する場合は、このプロパティを無視できます。

#### 型

**object**

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.67. .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
<b>tenantID</b>	<b>string</b>	<b>tenantID</b> は、各リクエストのテナントを識別する Loki <b>X-Scope-OrgID</b> ヘッダーです。
<b>tls</b>	<b>object</b>	Loki URL の TLS クライアント設定。
<b>url</b>	<b>string</b>	<b>url</b> 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。

### 15.1.68. .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
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

### 15.1.69. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.70. .spec.loki.monolithic.tls.userCert

#### 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方 TLS を使用する場合は、このプロパティを無視できます。

#### 型

**object**

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.71. .spec.networkPolicy

#### 説明

**networkPolicy** は、Network Observability のコンポーネントを分離するためのネットワークポリシー設定を定義します。

#### 型

**object**

プロパティ	型	説明
<b>additionalNamespaces</b>	<b>array (string)</b>	<b>additionalNamespaces</b> には、Network Observability namespace への接続を許可する追加の namespace を含めます。これにより、ネットワークポリシー設定の柔軟性が向上しますが、より詳細な設定が必要な場合は、これを無効にして独自の設定をインストールできます。



プロパティ	型	説明
<b>enable</b>	<b>boolean</b>	Network Observability (main および privileged)が使用する namespace にネットワークポリシーをデプロイします。これらのネットワークポリシーは、Network Observability コンポーネントをより適切に分離し、望ましくない接続を防ぎます。このオプションは、OVNKubernetes で使用する際にデフォルトで有効にされ、それ以外の場合は無効にされます（他の CNI でテストされていません）。無効にすると、Network Observability コンポーネントのネットワークポリシーを手動で作成できます。

### 15.1.72. .spec.processor

#### 説明

**processor** は、エージェントからフローを受信してエンリッチし、メトリクスを生成して Loki 永続化レイヤーや利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
<b>addZone</b>	<b>boolean</b>	<b>addZone</b> 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。
<b>advanced</b>	<b>object</b>	<b>advanced</b> 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、 <b>GOGC</b> や <b>GOMAXPROCS</b> 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
<b>clusterName</b>	<b>string</b>	<b>clusterName</b> 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。
<b>deduper</b>	<b>object</b>	<b>deduper</b> を使用すると、重複として識別されたフローをサンプリングまたはドロップして、リソース使用量を節約できます。
<b>filters</b>	<b>array</b>	<b>filters</b> を使用すると、生成されるフローの量を制限するカスタムフィルターを定義できます。これらのフィルターは、Kubernetes namespace によるフィルター処理などを含め、eBPF エージェントフィルター ( <b>spec.agent.ebpf.flowFilter</b> 内) よりも柔軟性が高くなりますが、パフォーマンスの向上は少なくなります。
<b>imagePullPolicy</b>	<b>string</b>	<b>imagePullPolicy</b> は、上で定義したイメージの Kubernetes プルポリシーです。
<b>kafkaConsumerAutoscaler</b>	<b>object</b>	<b>kafkaConsumerAutoscaler</b> は、Kafka メッセージを消費する <b>flowlogs-pipeline-transformer</b> 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。
<b>kafkaConsumerBatchSize</b>	<b>integer</b>	<b>kafkaConsumerBatchSize</b> 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
<b>kafkaConsumerQueueCapacity</b>	<b>integer</b>	<b>kafkaConsumerQueueCapacity</b> 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。
<b>kafkaConsumerReplicas</b>	<b>integer</b>	<b>kafkaConsumerReplicas</b> は、Kafka メッセージを消費する <b>flowlogs-pipeline-transformer</b> 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。
<b>logLevel</b>	<b>string</b>	プロセッサランタイムの <b>logLevel</b>
<b>logTypes</b>	<b>string</b>	<p><b>logTypes</b> 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"> <li>- 通常のネットワークフローをエクスポートする場合は <b>Flows</b>。これはデフォルトです。</li> <li>- <b>Conversations</b> 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。このモードでは、長時間にわたる会話では Prometheus メトリクスが不正確になることに注意してください。</li> <li>- <b>EndedConversations</b> は、終了した会話イベントのみを生成します。このモードでは、長時間にわたる会話では Prometheus メトリクスが不正確になることに注意してください。</li> <li>- <b>All</b> 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。リソースフットプリントへの影響があるため、推奨されません。</li> </ul>
<b>metrics</b>	<b>object</b>	<b>Metrics</b> は、メトリクスに関するプロセッサ設定を定義します。



プロパティ	型	説明
<b>multiClusterDeployment</b>	<b>boolean</b>	マルチクラスター機能を有効にするには、 <b>multiClusterDeployment</b> を <b>true</b> に設定します。これにより、 <b>clusterName</b> ラベルがフローデータに追加されます。
<b>resources</b>	<b>object</b>	<b>resources</b> 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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。
<b>subnetLabels</b>	<b>object</b>	<b>subnetLabels</b> 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 <b>SrcSubnetLabel</b> または <b>DstSubnetLabel</b> が追加されます。

### 15.1.73. .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
-------	---	----



プロパティ	型	説明
<b>conversationEndTimeout</b>	<b>string</b>	<b>conversationEndTimeout</b> 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、 <b>conversationTerminatingTimeout</b> を使用します)。
<b>conversationHeartbeatInterval</b>	<b>string</b>	<b>conversationHeartbeatInterval</b> は、対話の "tick" イベント間の待機時間です。
<b>conversationTerminatingTimeout</b>	<b>string</b>	<b>conversationTerminatingTimeout</b> 、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。
<b>dropUnusedFields</b>	<b>boolean</b>	<b>dropUnusedFields</b> [非推奨 (*)] この設定は、現在使用されていません。
<b>enableKubeProbes</b>	<b>boolean</b>	<b>enableKubeProbes</b> 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。
<b>env</b>	<b>object (string)</b>	<b>env</b> 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。 <b>GOGC</b> や <b>GOMAXPROCS</b> 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。
<b>healthPort</b>	<b>integer</b>	<b>healthPort</b> 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。



プロパティ	型	説明
<b>port</b>	<b>integer</b>	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。
<b>profilePort</b>	<b>integer</b>	<b>profilePort</b> 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
<b>scheduling</b>	<b>object</b>	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。
<b>secondaryNetworks</b>	<b>array</b>	リソース識別のためにチェックするセカンダリーネットワークを定義します。正確な識別を確実に行うために、インデックス値からクラスター全体で一意的識別子が形成されるようにする必要があります。同じインデックスが複数のリソースで使用されている場合、それらのリソースに誤ったラベルが付けられる可能性があります。

#### 15.1.74. .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
<b>affinity</b>	<b>object</b>	指定した場合、Pod のスケジューリング制約。ドキュメントは、 <a href="https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling">https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling</a> を参照してください。



プロパティ	型	説明
<b>nodeSelector</b>	<b>object (string)</b>	<b>nodeSelector</b> 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。ドキュメントは、 <a href="https://kubernetes.io/docs/concepts/configuration/assign-pod-node/">https://kubernetes.io/docs/concepts/configuration/assign-pod-node/</a> を参照してください。
<b>priorityClassName</b>	<b>string</b>	指定した場合、Pod の優先度を示します。ドキュメントは、 <a href="https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption">https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption</a> 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。
<b>tolerations</b>	<b>array</b>	<b>tolerations</b> は、一致する taint を持つノードに Pod がスケジュールできるようにする toleration のリストです。ドキュメントは、 <a href="https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling">https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling</a> を参照してください。

### 15.1.75. .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。ドキュメントは、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

#### 型

**object**

### 15.1.76. .spec.processor.advanced.scheduling.tolerations

#### 説明

**tolerations** は、一致する taint を持つノードに Pod がスケジュールできるようにする toleration のリストです。ドキュメントは、<https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling> を参照してください。

#### 型

**array**



## 15.1.77. .spec.processor.advanced.secondaryNetworks

### 説明

リソース識別のためにチェックするセカンダリーネットワークを定義します。正確な識別を確実に  
行うために、インデックス値からクラスター全体で一意的識別子が形成されるようにする必要があります。  
同じインデックスが複数のリソースで使用されている場合、それらのリソースに誤ったラ  
ベルが付けられる可能性があります。

### 型

array

## 15.1.78. .spec.processor.advanced.secondaryNetworks[]

### 説明

### 型

object

### 必須

- index
- name

プロパティ	型	説明
index	array (string)	<b>index</b> は、Pod のインデックス作成に使用するフィールドのリストです。これらのフィールドから、クラスター全体で一意的 Pod 識別子が形成されるようにする必要があります。 <b>MAC</b> 、 <b>IP</b> 、 <b>Interface</b> のいずれかを使用できます。 'k8s.v1.cni.cncf.io/network-status' アノテーションに存在しないフィールドは、インデックスに追加しないでください。
name	string	<b>name</b> は、Pod のアノテーション 'k8s.v1.cni.cncf.io/network-status' に表示されるネットワーク名と一致する必要があります。

## 15.1.79. .spec.processor.deduper

### 説明

**deduper** を使用すると、重複として識別されたフローをサンプリングまたはドロップして、リソース使用量を節約できます。

### 型

object



プロパティ	型	説明
<b>mode</b>	<b>string</b>	<p>プロセッサの重複排除モードを設定します。エージェントは別々のノードから報告された同じフローを重複排除できないため、これはエージェントベースの重複排除に加えて設定されます。</p> <p>- <b>Drop</b> を使用して重複と見なされるすべてのフローをドロップすると、リソース使用量をさらに節約できますが、ピアから使用されるネットワークインターフェイスやネットワークイベントなどの一部の情報が失われる可能性があります。</p> <p>- 重複と見なされる 50 (デフォルト) のフローのうち1つだけランダムに保持するには、<b>Sample</b> を使用します。これは、すべての重複を排除する場合と、すべての重複を保持する場合の中間です。このサンプリングアクションは、エージェントベースのサンプリングに加えて実行されます。エージェントとプロセッサの両方のサンプリング値が <b>50</b> の場合、結合されたサンプリングは 1:2500 になります。</p> <p>- プロセッサベースの重複排除をオフにするには、<b>Disabled</b> を使用します。</p>
<b>sampling</b>	<b>integer</b>	<p><b>sampling</b> は、deduper の <b>mode</b> が <b>Sample</b> の場合のサンプリング間隔です。たとえば、値が <b>50</b> の場合、50 個中1個のフローがサンプリングされます。</p>

### 15.1.80. .spec.processor.filters

#### 説明

**filters** を使用すると、生成されるフローの量を制限するカスタムフィルターを定義できます。これらのフィルターは、Kubernetes namespace によるフィルター処理などを含め、eBPF エージェントフィルター (**spec.agent.ebpf.flowFilter** 内) よりも柔軟性が高くなりますが、パフォーマンスの向上は少なくなります。

#### 型

**array**



## 15.1.81. .spec.processor.filters[]

## 説明

**FLPFilterSet** は、すべての条件を満たす FLP ベースのフィルタリングに必要な設定を定義します。

## 型

object

プロパティ	型	説明
<b>outputTarget</b>	<b>string</b>	指定されている場合、これらのフィルターのターゲットが、1つの出力 ( <b>Loki</b> 、 <b>Metrics</b> 、または <b>Exporters</b> ) に設定されます。デフォルトでは、すべての出力がターゲットになります。
<b>query</b>	<b>string</b>	保存するネットワークフローを選択するクエリー。このクエリー言語の詳細は、 <a href="https://github.com/netobserv/flowlogs-pipeline/blob/main/docs/filtering.md">https://github.com/netobserv/flowlogs-pipeline/blob/main/docs/filtering.md</a> を参照してください。
<b>sampling</b>	<b>integer</b>	<b>sampling</b> は、このフィルターに適用するオプションのサンプリング間隔です。たとえば、値が <b>50</b> の場合、50 個中 1 個のマッチしたフローがサンプリングされることを意味します。

## 15.1.82. .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

## 15.1.83. .spec.processor.metrics

## 説明

**Metrics** 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
<b>alerts</b>	<b>array</b>	<p><b>alerts</b> は、Prometheus AlertManager 用に作成されるアラートのリストであり、テンプレートとバリエーションによって構成されています [サポート対象外 (*)]。これは現在、フィーチャゲートの制御下にある実験的な機能です。有効にするには、<b>spec.processor.advanced.env</b> を編集し、<b>EXPERIMENTAL_ALERTS_HEALTH</b> を <b>true</b> に設定して追加します。アラートの詳細情報: <a href="https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md">https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md</a></p>
<b>disableAlerts</b>	<b>array (string)</b>	<p><b>disableAlerts</b> は、デフォルトのアラートセットから無効にするアラートグループのリストです。指定可能な値は、<b>NetObservNoFlows</b>、<b>NetObservLokiError</b>、<b>PacketDropsByKernel</b>、<b>PacketDropsByDevice</b>、<b>IPsecErrors</b>、<b>NetpolDenied</b>、<b>LatencyHighTrend</b>、<b>DNSErrors</b>、<b>ExternalEgressHighTrend</b>、<b>ExternalIngressHighTrend</b>、<b>CrossAZ</b> です。アラートの詳細情報: <a href="https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md">https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md</a></p>



プロパティ	型	説明
<b>includeList</b>	<b>array (string)</b>	<p><b>includeList</b> 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、<b>namespace_egress_packets_total</b> は、Prometheus では <b>netobserv_namespace_egress_packets_total</b> 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。デフォルトで有効になっているメトリクスは、<b>namespace_flows_total</b>、<b>node_ingress_bytes_total</b>、<b>node_egress_bytes_total</b>、<b>workload_ingress_bytes_total</b>、<b>workload_egress_bytes_total</b>、<b>namespace_drop_packets_total</b> (<b>PacketDrop</b> 機能が有効な場合)、<b>namespace_rtt_seconds</b> (<b>FlowRTT</b> 機能が有効な場合)、<b>namespace_dns_latency_seconds</b> (<b>DNSTracking</b> 機能が有効な場合)、<b>namespace_network_policy_events_total</b> (<b>NetworkEvents</b> 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、<a href="https://github.com/netobserv/network-observability-operator/blob/main/docs/Metrics.md">https://github.com/netobserv/network-observability-operator/blob/main/docs/Metrics.md</a> を参照してください。</p>
<b>server</b>	<b>object</b>	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

## 15.1.84. .spec.processor.metrics.alerts

### 説明

**alerts** は、Prometheus AlertManager 用に作成されるアラートのリストであり、テンプレートとバリエーションによって構成されています [サポート対象外 (\*)]。これは現在、フィーチャーゲートの制御下にある実験的な機能です。有効にするには、**spec.processor.advanced.env** を編集し、**EXPERIMENTAL\_ALERTS\_HEALTH** を **true** に設定して追加します。アラートの詳細情報: <https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md>



型

array

## 15.1.85. .spec.processor.metrics.alerts[]

説明

型

object

必須

- **template**
- **variants**

プロパティ	型	説明
<b>template</b>	<b>string</b>	アラートテンプレート名。指定可能な値は、 <b>PacketDropsByKernel</b> 、 <b>PacketDropsByDevice</b> 、 <b>IPsec Errors</b> 、 <b>NetpolDenied</b> 、 <b>LatencyHighTrend</b> 、 <b>DNSErrors</b> 、 <b>ExternalEgressHighTrend</b> 、 <b>ExternalIngressHighTrend</b> 、 <b>CrossAZ</b> です。アラートの詳細情報: <a href="https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md">https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md</a>
<b>variants</b>	<b>array</b>	このテンプレートのバリエーションとのリスト

## 15.1.86. .spec.processor.metrics.alerts[].variants

説明

このテンプレートのバリエーションのリスト

型

array

## 15.1.87. .spec.processor.metrics.alerts[].variants[]

説明

型

object

必須



- **thresholds**

プロパティ	型	説明
<b>groupBy</b>	<b>string</b>	オプションのグループ化基準。指定可能な値は、 <b>Node</b> 、 <b>Namespace</b> 、 <b>Workload</b> です。
<b>lowVolumeThreshold</b>	<b>string</b>	低ボリュームしきい値を使用すると、S/N 比を改善するために、トラフィック量が少なすぎるメトリクスを無視できます。これは絶対レート (コンテキストに応じて、1 秒あたりのバイト数または 1 秒あたりのパケット数) の形で指定します。指定した場合、浮動小数点数として解析可能である必要があります。
<b>thresholds</b>	<b>object</b>	重大度別のアラートのしきい値。これらの値は、アラートがトリガーされる基準となるエラーのパーセンテージとして表されます。浮動小数点数として解析可能である必要があります。
<b>trendDuration</b>	<b>string</b>	トレンドアラートで、ベースライン比較に使用される期間。たとえば、"2h" は 2 時間の平均と比較することを意味します。デフォルトは 2h です。
<b>trendOffset</b>	<b>string</b>	トレンドアラートで、ベースライン比較に使用されるオフセット時間。たとえば、"1d" は昨日と比較することを意味します。デフォルトは 1d です。

15.1.88. `.spec.processor.metrics.alerts[].variants[].thresholds`

## 説明

重大度別のアラートのしきい値。これらの値は、アラートがトリガーされる基準となるエラーのパーセンテージとして表されます。浮動小数点数として解析可能である必要があります。

## 型

**object**



プロパティ	型	説明
重大	string	重大度 <b>critical</b> のしきい値。重大なアラートを生成しない場合は空のままにします。
info	string	重大度 <b>info</b> のしきい値。情報提供アラートを生成しない場合は空白のままにします。
警告	string	重大度 <b>warning</b> のしきい値。警告アラートを生成しない場合は空のままにします。

### 15.1.89. .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。
tls	object	TLS 設定。

### 15.1.90. .spec.processor.metrics.server.tls

#### 説明

TLS 設定。

#### 型

**object**

#### 必須

- type

プロパティ	型	説明
-------	---	----



プロパティ	型	説明
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>providedCaFile</b> フィールドが無視されます。
<b>provided</b>	<b>object</b>	<b>type</b> が <b>Provided</b> に設定されている場合の TLS 設定。
<b>providedCaFile</b>	<b>object</b>	<b>type</b> が <b>Provided</b> に設定されている場合の CA ファイルへの参照。
<b>type</b>	<b>string</b>	TLS 設定のタイプを選択します。  - <b>Disabled</b> (デフォルト) は、エンドポイントに TLS を設定しません。- <b>Provided</b> 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 (*)]。- <b>Auto</b> 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。

### 15.1.91. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	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
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.92. .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
<b>file</b>	<b>string</b>	config map またはシークレット内のファイル名。
<b>name</b>	<b>string</b>	ファイルを含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	ファイル参照のタイプ ( <b>configmap</b> または <b>secret</b> )。

### 15.1.93. .spec.processor.resources

#### 説明



**resources** は、このコンテナー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
<b>limits</b>	<b>integer-or-string</b>	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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。
<b>requests</b>	<b>integer-or-string</b>	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。コンテナーで 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、 <a href="https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/">https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/</a> を参照してください。

## 15.1.94. .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
-------	---	----



プロパティ	型	説明
<b>customLabels</b>	<b>array</b>	<b>customLabels</b> 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。 <b>openShiftAutoDetect</b> 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、 <b>customLabels</b> がそのサブネットをオーバーライドできます。
<b>openShiftAutoDetect</b>	<b>boolean</b>	<b>openShiftAutoDetect</b> を <b>true</b> 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。

### 15.1.95. .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**

### 15.1.96. .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**

#### 必須

- **cidrs**



- **name**

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

### 15.1.97. .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
<b>querier</b>	<b>object</b>	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。

### 15.1.98. .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**

#### 必須

- **mode**

プロパティ	型	説明
-------	---	----



プロパティ	型	説明
<b>enable</b>	<b>boolean</b>	<b>enable</b> が <b>true</b> 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 <b>false</b> 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 ( <b>spec.loki</b> 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。
<b>manual</b>	<b>object</b>	<b>Manual</b> モードの Prometheus 設定。
<b>mode</b>	<b>string</b>	<p><b>mode</b> 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。</p> <ul style="list-style-type: none"> <li>- 自動設定を試行するには、<b>Auto</b> を使用します。OpenShift Container Platform では、OpenShift Container Platform クラスターモニタリングの Thanos クエリーを使用します。</li> <li>- 手動設定の場合は、<b>Manual</b> を使用します。</li> </ul>
<b>timeout</b>	<b>string</b>	<b>timeout</b> 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。



プロパティ	型	説明
-------	---	----

### 15.1.99. .spec.prometheus.querier.manual

#### 説明

**Manual** モードの Prometheus 設定。

#### 型

**object**

プロパティ	型	説明
<b>forwardUserToken</b>	<b>boolean</b>	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、 <b>true</b> に設定します。
<b>tls</b>	<b>object</b>	Prometheus URL の TLS クライアント設定。
<b>url</b>	<b>string</b>	<b>url</b> 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。

### 15.1.100. .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
<b>caCert</b>	<b>object</b>	<b>caCert</b> は、認証局の証明書の参照を定義します。
<b>enable</b>	<b>boolean</b>	TLS を有効にします。



プロパティ	型	説明
<b>insecureSkipVerify</b>	<b>boolean</b>	<b>insecureSkipVerify</b> 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。 <b>true</b> に設定すると、 <b>caCert</b> フィールドが無視されます。
<b>userCert</b>	<b>object</b>	<b>userCert</b> は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方 TLS を使用する場合は、このプロパティを無視できます。

### 15.1.101. .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
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。



## 15.1.102. .spec.prometheus.queries.manual.tls.userCert

### 説明

**userCert** は、ユーザー証明書の参照を定義し、mTLS に使用されます。一方向 TLS を使用する場合は、このプロパティを無視できます。

### 型

**object**

プロパティ	型	説明
<b>certFile</b>	<b>string</b>	<b>certFile</b> 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。
<b>certKey</b>	<b>string</b>	<b>certKey</b> 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。
<b>name</b>	<b>string</b>	証明書を含む config map またはシークレットの名前。
<b>namespace</b>	<b>string</b>	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。
<b>type</b>	<b>string</b>	証明書参照のタイプ ( <b>configmap</b> または <b>secret</b> )。



## 第16章 FLOWMETRIC 設定パラメーター

**FlowMetric** API は、収集されたネットワークフローログからカスタムの可観測性メトリクスを生成するために使用されます。

### 16.1. 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
<b>apiVersion</b>	<b>string</b>	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、 <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources</a> を参照してください。
<b>kind</b>	<b>string</b>	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。CamelCase を使用します。詳細: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds</a>
<b>metadata</b>	<b>object</b>	標準オブジェクトのメタデータ。詳細: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata</a>



プロパティ	型	説明
<b>spec</b>	<b>object</b>	<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。<a href="https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric">https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric</a> 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、<b>promql: count({name=~"netobserv.*"}) by (name)</b> を実行します。</p>

### 16.1.1. .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**

### 16.1.2. .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

- type

プロパティ	型	説明
<b>buckets</b>	<b>array (string)</b>	<b>type</b> 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。
<b>charts</b>	<b>array</b>	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。
<b>direction</b>	<b>string</b>	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。 <b>Ingress</b> に設定すると、 <b>FlowDirection</b> に正規表現フィルター <b>0 2</b> を追加した場合と同じになります。 <b>Egress</b> に設定すると、 <b>FlowDirection</b> に正規表現フィルター <b>1 2</b> を追加した場合と同じになります。
<b>divider</b>	<b>string</b>	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。
<b>filters</b>	<b>array</b>	<b>filters</b> 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。使用可能なフィールドのリストは、ドキュメント <a href="https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference">https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference</a> を参照してください。



プロパティ	型	説明
<b>flatten</b>	<b>array (string)</b>	<b>flatten</b> は、Interfaces や NetworkEvents など、フラット化する必要がある配列型フィールドのリストです。フラット化されたフィールドでは、そのフィールド内の項目ごとに1つのメトリクスが生成されます。たとえば、バイトカウンターで <b>Interfaces</b> をフラット化すると、Interfaces [br-ex, ens5] を持つフローでは <b>br-ex</b> のカウンターと <b>ens5</b> のカウンターが1つずつ増加します。
<b>labels</b>	<b>array (string)</b>	<b>labels</b> 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 (例: <b>SrcK8S_Namespace</b> )。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 ( <a href="https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric">https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric</a> 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、 <b>"SrcK8S_OwnerName"</b> または <b>"DstK8S_OwnerName"</b> を、 <b>"SrcK8S_Name"</b> または <b>"DstK8S_Name"</b> 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は、ドキュメント <a href="https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference">https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference</a> を参照してください。
<b>metricName</b>	<b>string</b>	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。 <b>FlowMetric</b> リソース名に基づいて名前を生成するには、空のままにします。



プロパティ	型	説明
<b>remap</b>	<b>object (string)</b>	生成されるメトリクスラベルにフローフィールドとは異なる名前を使用するには、 <b>remap</b> プロパティを設定します。元のフローフィールドをキーとして使用し、目的のラベル名を値として使用します。
<b>type</b>	<b>string</b>	メトリクスタイプ: "Counter"、"Histogram"、または"Gauge"。 "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。 "Gauge" は、経時的な正確さが必要がないその他の値に使用します (ゲージは、Prometheus がメトリクスを取得するときに N 秒ごとにのみサンプリングされます)。
<b>valueField</b>	<b>string</b>	<b>valueField</b> 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 (例: <b>Bytes</b> )。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は、ドキュメント <a href="https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference">https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference</a> を参照してください。

### 16.1.3. .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**

### 16.1.4. .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
<b>dashboardName</b>	<b>string</b>	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。
<b>queries</b>	<b>array</b>	このグラフに表示するクエリーのリスト。 <b>type</b> が <b>SingleStat</b> 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。
<b>sectionName</b>	<b>string</b>	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。 <b>sectionName</b> 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。
<b>title</b>	<b>string</b>	グラフのタイトル。
<b>type</b>	<b>string</b>	グラフの種類。
<b>unit</b>	<b>string</b>	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。

### 16.1.5. .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

## 16.1.6. .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 <code>{{ Label }}</code> という形式で設定できます。たとえば、 <b>promQL</b> でラベルごとに時系列をグループ化する場合 (例: <b>sum(rate(\$METRIC[2m])) by (Label1, Label2)</b> )、凡例として <b>Label1={{ Label1 }}</b> , <b>Label2={{ Label2 }}</b> と記述します。
promQL	string	Prometheus に対して実行する <b>promQL</b> クエリー。グラフの <b>type</b> が <b>SingleStat</b> 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、 <b>\$METRIC</b> を使用できます。たとえば、 <b>sum(rate(\$METRIC[2m]))</b> です。 <b>promQL</b> の詳細は、Prometheus のドキュメント <a href="https://prometheus.io/docs/prometheus/latest/querying/basics/">https://prometheus.io/docs/prometheus/latest/querying/basics/</a> を参照してください。



プロパティ	型	説明
<b>top</b>	<b>integer</b>	タイムスタンプごとに表示する上位 N 個の系列。 <b>SingleStat</b> グラフタイプには適用されません。

### 16.1.7. .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。使用可能なフィールドのリストは、ドキュメント [https://docs.redhat.com/en/documentation/openshift\\_container\\_platform/latest/html/network\\_observation\\_flows-format-reference](https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observation_flows-format-reference) を参照してください。

#### 型

**array**

### 16.1.8. .spec.filters[]

#### 説明

#### 型

**object**

#### 必須

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

プロパティ	型	説明
<b>field</b>	<b>string</b>	フィルタリングするフィールドの名前 (例: <b>SrcK8S_Namespace</b> )。
<b>matchType</b>	<b>string</b>	適用するマッチングのタイプ
<b>value</b>	<b>string</b>	フィルタリングする値。 <b>matchType</b> が <b>Equal</b> または <b>NotEqual</b> の場合、 <b>\$(SomeField)</b> 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17章 ネットワークフロー形式のリファレンス

ネットワークフロー形式の仕様をご確認ください。この仕様は、フローデータを Kafka にエクスポートするために内部的に使用されます。

### 17.1. 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 **FlowMetrics** 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。この API の使用に関する詳細は、**FlowMetrics** のドキュメントを参照してください。

名前	型	説明	フィルター ID	Loki ラベル	カーディナリティー	OpenTelemetry
<b>Bytes</b>	number	バイト数	該当なし	いいえ	avoid	bytes
<b>DnsErrno</b>	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	<b>dns_errno</b>	いいえ	fine	dns.errno
<b>DnsFlags</b>	number	DNS レコードの DNS フラグ	該当なし	いいえ	fine	dns.flags
<b>DnsFlagsResponseCode</b>	string	解析された DNS ヘッダーの RCODEs 名	<b>dns_flag_response_code</b>	いいえ	fine	dns.responsecode
<b>DnsId</b>	number	DNS レコード id	<b>dns_id</b>	いいえ	avoid	dns.id
<b>DnsLatencyMs</b>	number	DNS リクエストとレスポンスの間の時間 (ミリ秒単位)	<b>dns_latency</b>	いいえ	avoid	dns.latency
<b>Dscp</b>	number	Differentiated Services Code Point (DSCP) の値	<b>dscp</b>	いいえ	fine	dscp
<b>DstAddress</b>	string	宛先 IP アドレス (ipv4 または ipv6)	<b>dst_address</b>	いいえ	avoid	destination.address



名前	型	説明	フィルター ID	Loki ラベル	カーディナリティー	OpenTelemetry
<b>DstK8S_HostIP</b>	string	送信先ノード IP	<b>dst_host_address</b>	いいえ	fine	destination.k8s.host.address
<b>DstK8S_HostName</b>	string	送信先ノード名	<b>dst_host_name</b>	いいえ	fine	destination.k8s.host.name
<b>DstK8S_Name</b>	string	宛先 Kubernetes オブジェクトの名前 (Pod 名、Service 名、Node 名など)。	<b>dst_name</b>	いいえ	careful	destination.k8s.name
<b>DstK8S_NameSpace</b>	string	宛先 namespace	<b>dst_namespace</b>	はい	fine	destination.k8s.namespace.name
<b>DstK8S_NetworkName</b>	string	宛先ネットワーク名	<b>dst_network</b>	いいえ	fine	該当なし
<b>DstK8S_OwnerName</b>	string	宛先所有者の名前 (Deployment 名、StatefulSet 名など)。	<b>dst_owner_name</b>	はい	fine	destination.k8s.owner.name
<b>DstK8S_OwnerType</b>	string	宛先所有者の種類 (Deployment、StatefulSet など)。	<b>dst_kind</b>	いいえ	fine	destination.k8s.owner.kind
<b>DstK8S_Type</b>	string	宛先 Kubernetes オブジェクトの種類 (Pod、Service、Node など)。	<b>dst_kind</b>	はい	fine	destination.k8s.kind
<b>DstK8S_Zone</b>	string	宛先アベイラビリティゾーン	<b>dst_zone</b>	はい	fine	destination.zone
<b>DstMac</b>	string	宛先 MAC アドレス	<b>dst_mac</b>	いいえ	avoid	destination.mac
<b>DstPort</b>	number	送信先ポート	<b>dst_port</b>	いいえ	careful	destination.port



名前	型	説明	フィルター ID	Loki ラベル	カーディナリティー	OpenTelemetry
<b>DstSubnetLabel</b>	string	宛先サブネットラベル	<b>dst_subnet_label</b>	いいえ	fine	destination.subnet.label
<b>フラグ</b>	string[]	RFC-9293 に基づく、フローに含まれる TCP フラグのリスト。パケットごとの組み合わせ ( <ul style="list-style-type: none"> <li>- SYN_ACK</li> <li>- FIN_ACK</li> <li>- RST_ACK</li> </ul> ) を表す追加のカスタムフラグが含まれます。	<b>tcp_flags</b>	いいえ	careful	tcp.flags
<b>FlowDirection</b>	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。 <ul style="list-style-type: none"> <li>- 0: Ingress (ノード観測点からの受信トラフィック)</li> <li>- 1: Egress (ノード観測点からの送信トラフィック)</li> <li>- 2: Inner (送信元ノードと宛先ノードが同じ)</li> </ul>	<b>node_direction</b>	はい	fine	host.direction
<b>IPSecStatus</b>	string	IPsec 暗号化のステータス (Egress 時、カーネルの xfrm_output 関数によって指定) または復号化のステータス (Ingress 時、xfrm_input 経由)	<b>ipsec_status</b>	いいえ	fine	該当なし
<b>IcmpCode</b>	number	ICMP コード	<b>icmp_code</b>	いいえ	fine	icmp.code
<b>IcmpType</b>	number	ICMP のタイプ	<b>icmp_type</b>	いいえ	fine	icmp.type
<b>IfDirections</b>	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。 <ul style="list-style-type: none"> <li>- 0: Ingress (インターフェイスの受信トラフィック)</li> <li>- 1: Egress (インターフェイスの送信トラフィック)</li> </ul>	<b>ifdirections</b>	いいえ	fine	interface.directions
<b>Interfaces</b>	string[]	ネットワークインターフェイス	<b>interfaces</b>	いいえ	careful	interface.names



名前	型	説明	フィルター ID	Loki ラベル	カーディナリティー	OpenTelemetry
<b>K8S_ClusterName</b>	string	クラスター名またはクラスター識別子	<b>cluster_name</b>	はい	fine	k8s.cluster.name
<b>K8S_FlowLayer</b>	string	フローのレイヤー: 'app' または 'infra'	<b>flow_layer</b>	はい	fine	k8s.layer
<b>NetworkEvents</b>	object[]	ネストされたフィールドで構成されるネットワークポリシーアクションなどのネットワークイベント: - 機能 (ネットワークポリシーの "acl" など) - タイプ ("AdminNetworkPolicy" など) - namespace (存在する場合はイベントが適用される namespace) - 名前 (イベントをトリガーしたリソースの名前) - アクション ("allow" や "drop" など) - 方向 (Ingress または Egress)	<b>network_events</b>	いいえ	avoid	該当なし
<b>Packets</b>	number	パケット数	該当なし	いいえ	avoid	packets
<b>PktDropBytes</b>	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	drops.bytes
<b>PktDropLatestDropCause</b>	string	最新のドロップの原因	<b>pkt_drop_cause</b>	いいえ	fine	drops.latestcause
<b>PktDropLatestFlags</b>	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	drops.latestflags
<b>PktDropLatestState</b>	string	最後にドロップされたパケットの TCP 状態	<b>pkt_drop_state</b>	いいえ	fine	drops.lateststate



名前	型	説明	フィルター ID	Loki ラベル	カーディナリティー	OpenTelemetry
<b>PktDropPackets</b>	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	drops.packets
<b>Proto</b>	number	L4 プロトコル	<b>protocol</b>	いいえ	fine	protocol
<b>Sampling</b>	number	このフローで使用するサンプリング間隔	該当なし	いいえ	fine	該当なし
<b>SrcAddr</b>	string	送信元 IP アドレス (ipv4 または ipv6)	<b>src_address</b>	いいえ	avoid	source.address
<b>SrcK8S_HostIP</b>	string	送信元ノード IP	<b>src_host_address</b>	いいえ	fine	source.k8s.host.address
<b>SrcK8S_HostName</b>	string	送信元ノード名	<b>src_host_name</b>	いいえ	fine	source.k8s.host.name
<b>SrcK8S_Name</b>	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)	<b>src_name</b>	いいえ	careful	source.k8s.name
<b>SrcK8S_NameSpace</b>	string	送信元 namespace	<b>src_namespace</b>	はい	fine	source.k8s.namespace.name
<b>SrcK8S_NetworkName</b>	string	送信元ネットワーク名	<b>src_network</b>	いいえ	fine	該当なし
<b>SrcK8S_OwnerName</b>	string	送信元所有者の名前 (Deployment 名、StatefulSet 名など)。	<b>src_owner_name</b>	はい	fine	source.k8s.owner.name
<b>SrcK8S_OwnerType</b>	string	送信元所有者の種類 (Deployment、StatefulSet など)。	<b>src_kind</b>	いいえ	fine	source.k8s.owner.kind



名前	型	説明	フィルター ID	Loki ラベル	カーディナリティー	OpenTelemetry
<b>SrcK8S_Type</b>	string	送信元 Kubernetes オブジェクトの種類 (Pod、Service、Node など)。	<b>src_kind</b>	はい	fine	source.k8s.kind
<b>SrcK8S_Zone</b>	string	送信元アベイラビリティゾーン	<b>src_zone</b>	はい	fine	source.zone
<b>SrcMac</b>	string	送信元 MAC アドレス	<b>src_mac</b>	いいえ	avoid	source.mac
<b>SrcPort</b>	number	送信元ポート	<b>src_port</b>	いいえ	careful	source.port
<b>SrcSubnetLabel</b>	string	送信元サブネットラベル	<b>src_subnet_label</b>	いいえ	fine	source.subnet.label
<b>TimeFlowEndMs</b>	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	timeflow.end
<b>TimeFlowRttNs</b>	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)	<b>time_flow_rtt</b>	いいえ	avoid	tcp.rtt
<b>TimeFlowStartMs</b>	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	timeflow.start
<b>TimeReceived</b>	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	timereceived
<b>Udns</b>	string[]	ユーザー定義ネットワークのリスト	<b>udns</b>	いいえ	careful	該当なし
<b>XlatDstAddr</b>	string	パケット変換の送信先アドレス	<b>xlat_dst_address</b>	いいえ	avoid	該当なし
<b>XlatDstPort</b>	number	パケット変換の送信先ポート	<b>xlat_dst_port</b>	いいえ	careful	該当なし



名前	型	説明	フィルター ID	Loki ラベル	カーディナリティー	OpenTelemetry
<b>XlatSrcAddr</b>	string	パケット変換の送信元アドレス	<b>xlat_src_addr</b>	いいえ	avoid	該当なし
<b>XlatSrcPort</b>	number	パケット変換の送信元ポート	<b>xlat_src_port</b>	いいえ	careful	該当なし
<b>ZoneId</b>	number	パケット変換のゾーン ID	<b>xlat_zone_id</b>	いいえ	avoid	該当なし
<b>_HashId</b>	string	会話追跡では、会話識別子	<b>id</b>	いいえ	avoid	該当なし
<b>_RecordType</b>	string	レコードの種類: 通常のフローログの場合は <b>flowLog</b> 、会話追跡の場合は <b>newConnection</b> 、 <b>heartbeat</b> 、 <b>endConnection</b>	<b>type</b>	はい	fine	該当なし



## 第18章 NETWORK OBSERVABILITY のトラブルシューティング

Network Observability Operator とそのコンポーネントに関連する一般的な問題をトラブルシューティングするための診断アクションを実行します。

### 18.1. MUST-GATHER ツールの使用

must-gather ツールを使用すると、Network Observability Operator リソースとクラスター全体のリソース (Pod ログ、**FlowCollector**、**webhook** 設定など) 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
```

### 18.2. 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 **Observe** 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
```



### 18.3. 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
```

### 18.4. 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。



## 18.5. 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

**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 **800Mi** 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。

## 18.6. 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](#)

## 18.7. 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。

## 18.8. 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。

## 18.9. リソースのトラブルシューティング

### 18.10. 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 ❶
perStreamRateLimitBurst: 30 ❷
tenants:
  mode: openshift-network
managementState: Managed

```

- ❶ **perStreamRateLimit** のデフォルト値は **3** です。
- ❷ **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。

## 18.11. 大きなクエリーを実行すると LOKI エラーが発生する

大規模なクエリーを長時間実行すると、**timeout** や **too many outstanding requests** などの Loki エラーが発生する可能性があります。この問題を完全に修正する方法はありませんが、軽減する方法はいくつかあります。

### クエリーを調整してインデックス付きフィルターを追加する

Loki クエリーを使用すると、インデックスが付けられたフィールドまたはラベルと、インデックスが付けられていないフィールドまたはラベルの両方に対してクエリーを実行できます。ラベルにフィルターを含むクエリーのパフォーマンスが向上します。たとえば、インデックス付きフィールドではない特定の Pod をクエリーする場合は、その namespace をクエリーに追加できます。インデックス付きフィールドのリストは、"Network flows format reference" の **Loki label** 列にあります。

### Loki ではなく Prometheus にクエリーすることを検討する

長い時間範囲でクエリーを実行するには、Loki よりも Prometheus の方が適しています。ただし、Loki の代わりに Prometheus を使用できるかどうかは、ユースケースによって異なります。たとえば、Prometheus のクエリーは Loki よりもはるかに高速であり、時間範囲が長くてもパフォーマンスに影響はありません。しかし、Prometheus メトリクスには、Loki のフローログほど多くの情報は含まれていません。Network Observability OpenShift Web コンソールは、クエリーに互換性がある場合は、自動的に Loki よりも Prometheus を優先します。互換性がない場合は、デフォルトで Loki が使用されます。クエリーが Prometheus に対して実行されない場合は、いくつかのフィルターまたは集計を変更して切り替えることができます。OpenShift Web コンソールでは、Prometheus の使用を強制できます。互換性のないクエリーが失敗するとエラーメッセージが表示され、クエリーを互換性のあるものにするためにどのラベルを変更すればよいかを判断するのに役立ちます。たとえば、フィルターまたは集計を **Resource** または **Pods** から **Owner** に変更します。

### FlowMetrics API を使用して独自のメトリクスを作成することを検討する

必要なデータが Prometheus メトリクスとして利用できない場合は、FlowMetrics API を使用して独自のメトリクスを作成できます。詳細は、「FlowMetrics API リファレンス」および「FlowMetric API を使用したカスタムメトリクスの設定」を参照してください。

### クエリーパフォーマンスを向上させるために Loki を設定する

問題が解決しない場合は、クエリーのパフォーマンスを向上させるために Loki を設定することを検討してください。一部のオプションは、Operator と **LokiStack** の使用、**Monolithic** モード、**Microservices** モードなど、Loki に使用したインストールモードによって異なります。



- **LokiStack** または **Microservices** モードでは、[クエリーレプリカの数を増やして](#) みてください。
- [クエリーのタイムアウト](#) を増やします。また、**FlowCollector spec.loki.readTimeout** で、Loki への Network Observability の読み取りタイムアウトを増やす必要があります。

#### 関連情報

- [ネットワークフロー形式のリファレンス](#)
- [FlowMetric API リファレンス](#)
- [FlowMetric API を使用したカスタムメトリクスの設定](#)