OpenShift Container Platform 4.17

Network Observability

Configuring and using the Network Observability Operator in OpenShift Container Platform

Last Updated: 2026-01-16

OpenShift Container Platform 4.17 Network Observability

Configuring and using the Network Observability Operator in OpenShift Container Platform

Legal Notice

Abstract

Use the Network Observability Operator to observe and analyze network traffic flows for OpenShift Container Platform clusters.

Table of Contents

CHAPTER 1. NETWORK OBSERVABILITY OPERATOR RELEASE NOTES

The Network Observability Operator enables administrators to observe and analyze network traffic flows for OpenShift Container Platform clusters.

These release notes track the development of the Network Observability Operator in the OpenShift Container Platform.

For an overview of the Network Observability Operator, see About network observability.

1.1. NETWORK OBSERVABILITY OPERATOR 1.10.1 ADVISORY

You can review the advisory for Network Observability Operator 1.10.1 release.

- RHEA-2025:22761 Network Observability Operator 1.10.1

1.2. NETWORK OBSERVABILITY OPERATOR 1.10.1 CVES

You can review the CVEs for the Network Observability Operator 1.10.1 release.

- CVE-2025-47907

1.3. NETWORK OBSERVABILITY OPERATOR 1.10.1 FIXED ISSUES

The Network Observability Operator 1.10.1 release contains several fixed issues that improve performance and the user experience.

Warning Generated for Direct Mode on Clusters Over 15 Nodes

Before this update, the recommendation against using the **Direct** deployment model on large clusters was only available in the documentation.
With this release, the Network Observability Operator now generates a warning when the Direct deployment mode is used on a cluster exceeding 15 nodes.

NETOBSERV-2460

Network policy deployment disabled on OpenShiftSDN

Before this update, when OpenShift SDN was the cluster network plugin, enabling the **FlowCollector** network policy would break communication between network observability pods. This issue does not occur with OVN-Kubernetes, which is the default supported network plugin.
With this release, the Network Observability Operator no longer attempts to deploy the network policy when OpenShift SDN is detected; a warning is displayed instead. Additionally, the default value for enabling the network policy is modified: it is now enabled by default only when OVN-Kubernetes is detected as the cluster network plugin.

NETOBSERV-2450

Validation added for subnet label characters

Before this update, there were no restrictions on characters allowed in the subnet labels "name" configuration, meaning users could enter text containing spaces or special characters. This generated errors in the web console plugin when users tried to apply filters, and clicking the filter icon for a subnet label often failed.

With this release, the configured subnet label name is validated immediately when configured in the **FlowCollector** custom resource. The validation ensures the name contains only alphanumeric characters, **:**, _, and **-**. As a result, filtering on subnet labels from the web console plugin now works as expected.

NETOBSERV-2438

Network Observability CLI uses unique temporary directory per run

Before this update, the Network Observability CLI created or reused a single temporary (**tmp**) directory in the current working directory. This could lead to conflicts or data corruption between separate runs.
With this release, the Network Observability CLI now creates a unique temporary directory for each run, preventing potential conflicts and improving file management hygiene.

NETOBSERV-2481

1.4. NETWORK OBSERVABILITY OPERATOR 1.10 ADVISORY

Review the advisory that is available for the Network Observability Operator 1.10:

- RHEA-2025:19153 Network Observability Operator 1.10

1.5. NETWORK OBSERVABILITY OPERATOR 1.10 NEW FEATURES AND ENHANCEMENTS

The Network Observability Operator 1.10 release enhances security, improves performance, and introduces new CLI UI tools for better network flow management.

1.5.1. Network policy updates

The Network Observability Operator now supports configuring both ingress and egress network policies to control pod traffic. This enhancement improves security.

By default, the **spec.NetworkPolicy.enable** specification is now set to **true**. This means that if you use Loki or Kafka, it is recommended that you deploy the Loki Operator and Kafka instances into dedicated namespaces. This ensures that the network policies can be configured correctly to allow communication between all components.

1.5.2. Network Observability Operator CLI UI updates

This release brings the following new features and updates to the Network Observability Operator CLI (**oc netobserv**) user interface (UI):

Table view enhancements

- Customizable columns: Click **Manage Columns** to select which columns to display, and tailor the table to your needs.
- Smart filtering: Live filters now include auto-suggestions, making it easier to select the right keys and values.
- Packet preview: When capturing packets, click a row to inspect the **pcap** content directly.

Terminal-based line charts enhancements

- Metrics visualization: Real-time graphs are rendered directly in the CLI.
- Panel selection: Choose from predefined views or customize views by using the **Manage Panels** pop-up menu to selectively view charts of specific metrics.

1.5.3. Network observability console improvements

The network observability console plugin includes a new view to configure the **FlowCollector** custom resource (CR). From this view, you can complete the following tasks:

- Configure the **FlowCollector** CR.
- Calculate your resource footprint.
- Gain increased of issues such as configuration warnings or high metrics cardinality.

1.5.4. Performance improvements

Network Observability Operator 1.10 has improved the performance and memory footprint of the Operator, especially visible on large clusters.

1.6. NETWORK OBSERVABILITY OPERATOR 1.10 TECHNOLOGY PREVIEW FEATURES

1.6.1. Network Observability Operator custom alerts (Technology Preview)

This release introduces new alert functionality, and custom alert configuration. These capabilities are Technology Preview features, and must be explicitly enabled.

To view the new alerts, in the OpenShift Container Platform web console, click **Observe → Alerting → Alerting rules**.

1.6.2. Network Observability Operator Network Health dashboard (Technology Preview)

When you enable the Technology Preview alerts functionality in the Network Observability Operator, you can view a a new **Network Health** dashboard in the OpenShift Container Platform web console by clicking **Observe**.

The **Network Health** dashboard provides a summary of triggered alerts, distinguishing between critical, warning, and minor issues, and also shows pending alerts.

1.7. NETWORK OBSERVABILITY OPERATOR 1.10 REMOVED FEATURES

Review the removed features that might affect your use of the Network Observability Operator 1.10 release.

1.7.1. FlowCollector API version v1beta1 has been removed

The **FlowCollector** custom resource (CR) API version **v1beta1** has been removed and is no longer supported. Use the **v1beta2** version.

1.8. NETWORK OBSERVABILITY OPERATOR 1.10 KNOWN ISSUES

Review the following known issues and their recommended workarounds (where available) that might affect your use of the Network Observability Operator 1.10 release.

1.8.1. Upgrading to 1.10 fails on OpenShift Container Platform 4.14 and earlier

Upgrading to the Network Observability Operator 1.10 on OpenShift Container Platform 4.14 and earlier can fail due to a **FlowCollector** custom resource definition (CRD) validation error in the software catalog.

To workaround this problem, you must:

1. Uninstall both versions of the Network Observability Operator from the software catalog in the OpenShift Container Platform web console.
    a. Keep the **FlowCollector** CRD installed so that it doesn't cause any disruption in the flow collection process.
2. Check the current name of the **FlowCollector** CRD by running the following command:

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

    Expected output:

    ```
    v1beta1
    ```

3. Check the current serving status of the **FlowCollector** CRD by running the following command:

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

    Expected output:

    ```
    true
    ```

4. Set the **served** flag for the **v1beta1** version to **false** by running the following command:

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

5. Verify that the **served** flag is set to **false** by running the following command:

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

    Expected output:

    ```
    false
    ```

6. Install Network Observability Operator 1.10.

OCPBUGS-63208, NETOBSERV-2451

1.8.2. eBPF agent compatibility with older OpenShift Container Platform versions

The eBPF agent used in the Network Observability Command Line Interface (CLI) packet capture feature is incompatible with OpenShift Container Platform versions 4.16 and older.

This limitation prevents the eBPF-based Packet Capture Agent (PCA) from functioning correctly on those older clusters.

To work around this problem, you must manually configure PCA to use an older, compatible eBPF agent container image. For more information, see the Red Hat Knowledgebase Solution eBPF agent compatibility with older Openshift versions in Network Observability CLI 1.10+.

NETOBSERV-2358

1.8.3. eBPF Agent fails to send flows with OpenShiftSDN when NetworkPolicy is enabled

When running Network Observability Operator 1.10 on OpenShift Container Platform 4.14 clusters that use the **OpenShiftSDN** CNI plugin, the eBPF agent is unable to send flow records to the **flowlogs-pipeline** component. This occurs when the **FlowCollector** custom resource is created with **NetworkPolicy** enabled (**spec.networkPolicy.enable: true**).

As a consequence, flow data is not processed by the **flowlogs-pipeline** component and does not appear in the **Network Traffic** dashboard or the configured storage (Loki). The eBPF agent pod logs show **i/o timeout** errors when attempting to connect to the collector:

```
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\""
```

To work around this problem, set **spec.networkPolicy.enable** to **false** to disable **NetworkPolicy** in the **FlowCollector** resource for Network Observability Operator 1.10.

This will allow the eBPF agent to communicate with the **flowlogs-pipeline** component without interference from the automatically deployed network policy.

NETOBSERV-2450

1.9. NETWORK OBSERVABILITY OPERATOR 1.10 FIXED ISSUES

The Network Observability Operator 1.10 release contains several fixed issues that improve performance and the user experience.

1.9.1. MetricName and Remap fields are validated

Before this update, users could create a **FlowMetric** custom resource (CR) with an invalid metric name. Although the **FlowMetric** CR was successfully created, the underlying metric would fail silently without providing any error feedback to the user.

With this release, the **FlowMetric**, **metricName**, and **remap** fields are now validated before creation, so users are immediately notified if they enter an invalid name.

NETOBSERV-2348

1.9.2. Improved html-to-image export performance

Before this update, performance issues in the underlying library caused the **html-to-image** export function to take a long time, leading to browser freezing.

With this release, the performance of the **html-to-image** library has been improved, reducing export wait times and eliminating browser freezing during image generation.

NETOBSERV-2314

1.9.3. Improved warnings for eBPF privileged mode

Before this update, when users selected **eBPF** features that require **privileged** mode, the features would often fail without clearly informing the user that **privileged** mode was missing or needed to be enabled.

With this release, a validation hook immediately warns the user if the configuration is inconsistent. This improves user understanding and prevents misconfiguration.

NETOBSERV-2268

1.9.4. Subnet labels added to OpenTelemetry exporter

Before this update, the **OpenTelemetry** metrics exporter was missing the network flow labels **SrcSubnetLabel** and **DstSubnetLabel**, causing them to show as empty.

With this release, these labels are now correctly provided by the exporter. They have also been renamed to **source.subnet.label** and **destination.subnet.label** for improved clarity and consistency with **OpenTelemetry** standards.

NETOBSERV-2405

1.9.5. Reduced default tolerations for network observability components

Before this update, a default toleration was set on all network observability components to allow them to be scheduled on any node, including those tainted with **NoSchedule**. This could potentially block cluster upgrades.

With this release, the default toleration is now only maintained for the **eBPF** agents and the **Flowlogs-Pipeline** when configured in **Direct** mode. The toleration has been removed from the OpenShift Container Platform web console plugin and the **Flowlogs-Pipeline** when configured in **Kafka** mode.

Additionally, while tolerations were always configurable in the **FlowCollector** custom resource (CR), it was previously impossible to replace the tolerations with an empty list. It is now possible to replace the tolerations with an empty list.

NETOBSERV-2434

CHAPTER 2. NETWORK OBSERVABILITY OPERATOR RELEASE NOTES ARCHIVE

2.1. NETWORK OBSERVABILITY OPERATOR RELEASE NOTES ARCHIVE

These release notes track past developments of the Network Observability Operator in the OpenShift Container Platform. They are for reference purposes only.

The Network Observability Operator enables administrators to observe and analyze network traffic flows for OpenShift Container Platform clusters.

2.1.1. Network Observability Operator 1.9.3 advisory

The following advisory is available for the Network Observability Operator 1.9.3:

- RHEA-2025:15780 Network Observability Operator 1.9.3

2.1.2. Network Observability Operator 1.9.2 advisory

The following advisory is available for the Network Observability Operator 1.9.2:

- RHEA-2025:14150 Network Observability Operator 1.9.2

2.1.3. Network observability 1.9.2 bug fixes

- Before this update, OpenShift Container Platform versions 4.15 and earlier did not support the **TC_ATTACH_MODE** configuration. This led to command-line interface (CLI) errors and prevented the observation of packets and flows. With this release, the Traffic Control eXtension (TCX) hook attachment mode has been adjusted for these older versions. This eliminates **tcx** hook errors and enables flow and packet observation.

2.1.4. Network Observability Operator 1.9.1 advisory

You can review the advisory for the Network Observability Operator 1.9.1 release.

The following advisory is available for the Network Observability Operator 1.9.1:

- 2025:12024 Network Observability Operator 1.9.1

2.1.5. Network Observability Operator 1.9.1 fixed issues

You can review the fixed issues for the Network Observability Operator 1.9.1 release.

- Before this update, network flows were not observed on OpenShift Container Platform 4.15 due to an incorrect attach mode setting. This stopped users from monitoring network flows correctly, especially with certain catalogs. With this release, the default attach mode for OpenShift Container Platform versions older than 4.16.0 is set to **tc**, so flows are now observed on OpenShift Container Platform 4.15. (NETOBSERV-2333)

- Before this update, if an IPFIX collector restarted, configuring an IPFIX exporter could lose its connection and stop sending network flows to the collector. With this release, the connection is restored, and network flows continue to be sent to the collector. (NETOBSERV-2315)

- Before this update, when you configured an IPFIX exporter, flows without port information (such as ICMP traffic) were ignored, which caused errors in logs. TCP flags and ICMP data were also missing from IPFIX exports. With this release, these details are now included. Missing fields (like ports) no longer cause errors and are part of the exported data. (NETOBSERV-2307)

- Before this update, the User Defined Networks (UDN) Mapping feature showed a configuration issue and warning on OpenShift Container Platform 4.18 because the OpenShift version was incorrectly set in the code. This impacted the user experience. With this release, UDN Mapping now supports OpenShift Container Platform 4.18 without warnings, making the user experience smooth. (NETOBSERV-2305)

- Before this update, the expand function on the **Network Traffic** page had compatibility problems with OpenShift Container Platform Console 4.19. This resulted in empty menu space when expanding and an inconsistent user interface. With this release, the compatibility problem in the **NetflowTraffic** part and **theme hook** is resolved. The side menu in the **Network Traffic** view is now properly managed, which improves how you interact with the user interface. (NETOBSERV-2304)

2.1.6. Network Observability Operator 1.9.0 advisory

You can review the advisory for the Network Observability Operator 1.9.0 release.

- Network Observability Operator 1.9

2.1.7. Network Observability Operator 1.9.0 new features and enhancements

You can review the new features and enhancements for the Network Observability Operator 1.9.0 release.

2.1.7.1. User-defined networks with network observability

With this release, user-defined networks (UDN) feature is generally available with network observability. When the **UDNMapping** feature is enabled in network observability, the **Traffic** flow table has a **UDN labels** column. You can filter logs on **Source Network Name** and **Destination Network Name** information.

2.1.7.2. Filter flowlogs at ingestion

With this release, you can create filters to reduce the number of generated network flows and the resource usage of network observability components. The following filters can be configured:

- eBPF Agent filters

- Flowlogs-pipeline filters

2.1.7.3. IPsec support

This update brings the following enhancements to network observability when IPsec is enabled on OpenShift Container Platform:

- A new column named **IPsec Status** is displayed in the network observability **Traffic** flows view to show whether a flow was successfully IPsec-encrypted or if there was an error during encryption/decryption.

- A new dashboard showing the percentage of encrypted traffic is generated.

2.1.7.4. Network Observability CLI

The following filtering options are now available for packets, flows, and metrics capture:

- Configure the ratio of packets being sampled by using the **--sampling** option.
- Filter flows using a custom query by using the **--query** option.
- Specify interfaces to monitor by using the **--interfaces** option.
- Specify interfaces to exclude by using the **--exclude_interfaces** option.
- Specify metric names to generate by using the **--include_list** option.

For more information, see:

- Network Observability CLI reference

2.1.8. Network Observability Operator release notes 1.9.0 notable technical changes

You can review the notable technical changes for the Network Observability Operator 1.6.0 release.

- The **NetworkEvents** feature in network observability 1.9 has been updated to work with the newer Linux kernel of OpenShift Container Platform 4.19. This update breaks compatibility with older kernels. As a result, the **NetworkEvents** feature can only be used with OpenShift Container Platform 4.19. If you are using this feature with network observability 1.8 and OpenShift Container Platform 4.18, consider avoiding a network observability upgrade or upgrade to network observability 1.9 and OpenShift Container Platform to 4.19.
- The **netobserv-reader** cluster role has been renamed to **netobserv-loki-reader**.
- Improved CPU performance of the eBPF agents.

2.1.9. Network Observability Operator 1.9.0 Technology Preview features

You can review the Technology Preview features for the Network Observability Operator 1.9.0 release.

Some features in this release are currently in Technology Preview. These experimental features are not intended for production use. Note the following scope of support on the Red Hat Customer Portal for these features:

Technology Preview Features Support Scope

2.1.9.1. eBPF Manager Operator with network observability

The eBPF Manager Operator reduces the attack surface and ensures compliance, security, and conflict prevention by managing all eBPF programs. Network observability can use the eBPF Manager Operator to load hooks. This eliminates the need to provide the eBPF Agent with privileged mode or additional Linux capabilities like **CAP_BPF** and **CAP_PERFMON**. The eBPF Manager Operator with network observability is only supported on 64-bit AMD architecture.

2.1.10. Network Observability Operator 1.9.0 CVEs

You can review the CVEs for the Network Observability Operator 1.9.0 release.

- CVE-2025-26791

2.1.11. Network Observability Operator 1.9.0 fixed issues

You can review the fixed issues for the Network Observability Operator 1.9.0 release.

- Previously, when filtering by source or destination IP from the console plugin, using a Classless Inter-Domain Routing (CIDR) notation such as **10.128.0.0/24** did not work, returning results that should be filtered out. With this update, it is now possible to use a CIDR notation, with the results being filtered as expected. (NETOBSERV-2276)
- Previously, network flows might have incorrectly identified the network interfaces in use, especially with a risk of mixing up **eth0** and **ens5**. This issue only occurred when the eBPF agents were configured as **Privileged**. With this update, it has been fixed partially, and almost all network interfaces are correctly identified. Refer to the known issues below for more details. (NETOBSERV-2257)
- Previously, when the Operator checked for available Kubernetes APIs in order to adapt its behavior, if there was a stale API, this resulted in an error that prevented the Operator from starting normally. With this update, the Operator ignores error on unrelated APIs, logs errors on related APIs, and continues to run normally. (NETOBSERV-2240)
- Previously, users could not sort flows by **Bytes** or **Packets** in the **Traffic** flows view of the Console plugin. With this update, users can sort flows by **Bytes** and **Packets**. (NETOBSERV-2239)
- Previously, when configuring the **FlowCollector** resource with an IPFIX exporter, MAC addresses in the IPFIX flows were truncated to their 2 first bytes. With this update, MAC addresses are fully represented in the IPFIX flows. (NETOBSERV-2208)
- Previously, some of the warnings sent from the Operator validation webhook could lack clarity on what needed to be done. With this update, some of these messages have been reviewed and amended to make them more actionable. (NETOBSERV-2178)
- Previously, it was not obvious to figure out there was an issue when referencing a **LokiStack** from the **FlowCollector** resource, such as in case of typing error. With this update, the **FlowCollector** status clearly states that the referenced **LokiStack** is not found in that case. (NETOBSERV-2174)
- Previously, in the console plugin **Traffic flows** view, in case of text overflow, text ellipses sometimes hid much of the text to be displayed. With this update, it displays as much text as possible. (NETOBSERV-2119)
- Previously, the console plugin for network observability 1.8.1 and earlier did not work with the OpenShift Container Platform 4.19 web console, making the **Network Traffic** page inaccessible. With this update, the console plugin is compatible and the **Network Traffic** page is accessible in network observability 1.9.0. (NETOBSERV-2046)
- Previously, when using conversation tracking (**logTypes: Conversations** or **logTypes: All** in the **FlowCollector** resource), the **Traffic** rates metrics visible in the dashboards were flawed, wrongly showing an out-of-control increase in traffic. Now, the metrics show more accurate traffic rates. However, note that in **Conversations** and **EndedConversations** modes, these metrics are still not completely accurate as they do not include long-standing connections. This information has been added to the documentation. The default mode **logTypes: Flows** is recommended to avoid these inaccuracy. (NETOBSERV-1955)

2.1.12. Network Observability Operator 1.9.0 known issues

You can review the known issues for the Network Observability Operator 1.9.0 release.

- The user-defined network (UDN) feature displays a configuration issue and a warning when used with OpenShift Container Platform 4.18, even though it is supported. This warning can be ignored. (NETOBSERV-2305)
- In some rare cases, the eBPF agent is unable to appropriately correlate flows with the involved interfaces when running in **privileged** modes with several network namespaces. A large part of these issues have been identified and resolved in this release, but some inconsistencies remain, especially with the **ens5** interface. (NETOBSERV-2287)

2.1.13. Network Observability Operator 1.8.1 advisory

You can review the advisory for the Network Observability Operator 1.8.1 release.

Network Observability Operator 1.8.1

2.1.14. Network Observability Operator 1.8.1 CVEs

You can review the CVEs for the Network Observability Operator 1.8.1 release.

- CVE-2024-56171
- CVE-2025-24928

2.1.15. Network Observability Operator 1.8.1 fixed issues

You can review the fixed issues for the Network Observability Operator 1.8.1 release.

- This fix ensures that the **Observe** menu appears only once in future versions of OpenShift Container Platform. (**NETOBSERV-2139**)

2.1.16. Network Observability Operator 1.8.0 advisory

You can review the advisory for the Network Observability Operator 1.8.0 release.

- Network Observability Operator 1.8.0

2.1.17. Network Observability Operator 1.8.0 new features and enhancements

You can review the new features and enhancements for the Network Observability Operator 1.8.0 release.

2.1.17.1. Packet translation

You can now enrich network flows with translated endpoint information, showing not only the service but also the specific backend pod, so you can see which pod served a request.

For more information, see:

- Endpoint translation (xlat)
- Working with endpoint translation (xlat)

2.1.17.2. OVN-Kubernetes networking events tracking

> **IMPORTANT**
>
> OVN-Kubernetes networking events tracking is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can now use network event tracking in network observability to gain insight into OVN-Kubernetes events, including network policies, admin network policies, and egress firewalls.

For more information, see:

- Viewing network events

2.1.17.3. eBPF performance improvements in 1.8

- Network observability now uses hash maps instead of per-CPU maps. This means that network flows data is now tracked in the kernel space and new packets are also aggregated there. The de-duplication of network flows can now occur in the kernel, so the size of data transfer between the kernel and the user spaces yields better performance. With these eBPF performance improvements, there is potential to observe a CPU resource reduction between 40% and 57% in the eBPF Agent.

2.1.17.4. Network Observability CLI

The following new features, options, and filters are added to the Network Observability CLI for this release:

- Capture metrics with filters enabled by running the **oc netobserv metrics** command.
- Run the CLI in the background by using the **--background** option with flows and packets capture and running **oc netobserv follow** to see the progress of the background run and **oc netobserv copy** to download the generated logs.
- Enrich flows and metrics capture with Machines, Pods, and Services subnets by using the **--get-subnets** option.
- New filtering options available with packets, flows, and metrics capture:
  - eBPF filters on IPs, Ports, Protocol, Action, TCP Flags and more
  - Custom nodes using **--node-selector**
  - Drops only using **--drops**
  - Any field using **--regexes**

For more information, see:

- Network Observability CLI reference

2.1.18. Network Observability Operator release notes 1.8.0 fixed issues

You can review the fixed issues for the Network Observability Operator 1.8.0 release.

- Previously, the Network Observability Operator came with a "kube-rbac-proxy" container to manage RBAC for its metrics server. Since this external component is deprecated, it was necessary to remove it. It is now replaced with direct TLS and RBAC management through Kubernetes controller-runtime, without the need for a side-car proxy. (NETOBSERV-1999)
- Previously in the OpenShift Container Platform console plugin, filtering on a key that was not equal to multiple values would not filter anything. With this fix, the expected results are returned, which is all flows not having any of the filtered values. (NETOBSERV-1990)
- Previously in the OpenShift Container Platform console plugin with disabled Loki, it was very likely to generate a "Can't build query" error due to selecting an incompatible set of filters and aggregations. Now this error is avoided avoid by automatically disabling incompatible filters while still making the user aware of the filter incompatibility. (NETOBSERV-1977)
- Previously, when viewing flow details from the console plugin, the ICMP info was always displayed in the side panel, showing "undefined" values for non-ICMP flows. With this fix, ICMP info is not displayed for non-ICMP flows. (NETOBSERV-1969)
- Previously, the "Export data" link from the **Traffic flows** view did not work as intended, generating empty CSV reports. Now, the export feature is restored, generating non-empty CSV data. (NETOBSERV-1958)
- Previously, it was possible to configure the **FlowCollector** with **processor.logTypes** **Conversations**, **EndedConversations** or **All** with **loki.enable** set to **false**, despite the conversation logs being only useful when Loki is enabled. This resulted in resource usage waste. Now, this configuration is invalid and is rejected by the validation webhook. (NETOBSERV-1957)
- Configuring the **FlowCollector** with **processor.logTypes** set to **All** consumes much more resources, such as CPU, memory and network bandwidth, than the other options. This was previously not documented. It is now documented, and triggers a warning from the validation webhook. (NETOBSERV-1956)
- Previously, under high stress, some flows generated by the eBPF agent were mistakenly dismissed, resulting in traffic bandwidth under-estimation. Now, those generated flows are not dismissed. (NETOBSERV-1954)
- Previously, when enabling the network policy in the **FlowCollector** configuration, the traffic to the Operator webhooks was blocked, breaking the **FlowMetrics** API validation. Now traffic to the webhooks is allowed. (NETOBSERV-1934)
- Previously, when deploying the default network policy, namespaces **openshift-console** and **openshift-monitoring** were set by default in the **additionalNamespaces** field, resulting in duplicated rules. Now there is no additional namespace set by default, which helps avoid getting duplicated rules.(NETOBSERV-1933)
- Previously from the OpenShift Container Platform console plugin, filtering on TCP flags would match flows having only the exact desired flag. Now, any flow having at least the desired flag appears in filtered flows. (NETOBSERV-1890)

- When the eBPF agent runs in privileged mode and pods are continuously added or deleted, a file descriptor (FD) leak occurs. The fix ensures proper closure of the FD when a network namespace is deleted. (NETOBSERV-2063)
- Previously, the CLI agent **DaemonSet** did not deploy on master nodes. Now, a toleration is added on the agent **DaemonSet** to schedule on every node when taints are set. Now, CLI agent **DaemonSet** pods run on all nodes. (NETOBSERV-2030)
- Previously, the **Source Resource** and **Source Destination** filters autocomplete were not working when using Prometheus storage only. Now this issue is fixed and suggestions displays as expected. (NETOBSERV-1885)
- Previously, a resource using multiple IPs was displayed separately in the **Topology** view. Now, the resource shows as a single topology node in the view. (NETOBSERV-1818)
- Previously, the console refreshed the **Network traffic** table view contents when the mouse pointer hovered over the columns. Now, the display is fixed, so row height remains constant with a mouse hover. (NETOBSERV-2049)

2.1.19. Network Observability Operator release notes 1.8.0 known issues

You can review the known issues for the Network Observability Operator 1.8.0 release.

- If there is traffic that uses overlapping subnets in your cluster, there is a small risk that the eBPF Agent mixes up the flows from overlapped IPs. This can happen if different connections happen to have the exact same source and destination IPs and if ports and protocol are within a 5 seconds time frame and happening on the same node. This should not be possible unless you configured secondary networks or UDN. Even in that case, it is still very unlikely in usual traffic, as source ports are usually a good differentiator. (NETOBSERV-2115)
- After selecting a type of exporter to configure in the **FlowCollector** resource **spec.exporters** section from the OpenShift Container Platform web console form view, the detailed configuration for that type does not show up in the form. The workaround is to configure directly the YAML. (NETOBSERV-1981)

2.1.20. Network Observability Operator 1.7.0 advisory

You can review the advisory for the Network Observability Operator 1.7.0 release.

- Network Observability Operator 1.7.0

2.1.21. Network Observability Operator 1.7.0 new features and enhancements

You can review the following new features and enhancements for the Network Observability Operator 1.7.0 release.

2.1.21.1. OpenTelemetry support

You can now export enriched network flows to a compatible OpenTelemetry endpoint, such as the Red Hat build of OpenTelemetry.

For more information, see:

- Export enriched network flow data

2.1.21.2. Network observability Developer perspective

You can now use network observability in the **Developer** perspective.

For more information, see:

- OpenShift Container Platform console integration

2.1.21.3. TCP flags filtering

You can now use the **tcpFlags** filter to limit the volume of packets processed by the eBPF program.

For more information, see:

- Flow filter configuration parameters
- eBPF flow rule filter
- Detecting SYN flooding using the FlowMetric API and TCP flags

2.1.21.4. Network observability for OpenShift Virtualization

You can observe networking patterns on an OpenShift Virtualization setup by identifying eBPF-enriched network flows coming from VMs that are connected to secondary networks, such as through Open Virtual Network (OVN)-Kubernetes.

For more information, see:

- Configuring virtual machine (VM) secondary network interfaces for network observability

2.1.21.5. Network policy deploys in the FlowCollector custom resource (CR)

With this release, you can configure the **FlowCollector** custom resource (CR) to deploy a network policy for network observability. Previously, if you wanted a network policy, you had to manually create one. The option to manually create a network policy is still available.

For more information, see:

- Configuring an ingress network policy by using the FlowCollector custom resource

2.1.21.6. FIPS compliance

- You can install and use the Network Observability Operator in an OpenShift Container Platform cluster running in FIPS mode.

IMPORTANT

To enable FIPS mode for your cluster, you must run the installation program from a Red Hat Enterprise Linux (RHEL) computer configured to operate in FIPS mode. For more information about configuring FIPS mode on RHEL, see Switching RHEL to FIPS mode .

When running Red Hat Enterprise Linux (RHEL) or Red Hat Enterprise Linux CoreOS (RHCOS) booted in FIPS mode, OpenShift Container Platform core components use the RHEL cryptographic libraries that have been submitted to NIST for FIPS 140-2/140-3 Validation on only the x86_64, ppc64le, and s390x architectures.

2.1.21.7. eBPF agent enhancements

The following enhancements are available for the eBPF agent:

- If the DNS service maps to a different port than **53**, you can specify this DNS tracking port using **spec.agent.ebpf.advanced.env.DNS_TRACKING_PORT**.
- You can now use two ports for transport protocols (TCP, UDP, or SCTP) filtering rules.
- You can now filter on transport ports with a wildcard protocol by leaving the protocol field empty.

For more information, see:

- FlowCollector API specifications

2.1.21.8. Network Observability CLI

The Network Observability CLI (**oc netobserv**), is now generally available. The following enhancements have been made since the 1.6 Technology Preview release:

- There are now eBPF enrichment filters for packet capture similar to flow capture.
- You can now use filter **tcp_flags** with both flow and packets capture.
- The auto-teardown option is available when max-bytes or max-time is reached.

For more information, see:

- About the Network Observability CLI
- Network Observability CLI 1.7.0

2.1.22. Network Observability Operator 1.7.0 fixed issues

You can review the following fixed issues for the Network Observability Operator 1.7.0 release.

- Previously, when using a RHEL 9.2 real-time kernel, some of the webhooks did not work. Now, a fix is in place to check whether this RHEL 9.2 real-time kernel is being used. If the kernel is being used, a warning is displayed about the features that do not work, such as packet drop and neither Round-trip Time when using **s390x** architecture. The fix is in OpenShift 4.16 and later. (**NETOBSERV-1808**)

- Previously, in the **Manage panels** dialog in the **Overview** tab, filtering on **total**, **bar**, **donut**, or **line** did not show a result. Now the available panels are correctly filtered. (NETOBSERV-1540)
- Previously, under high stress, the eBPF agents were susceptible to enter into a state where they generated a high number of small flows, almost not aggregated. With this fix, the aggregation process is still maintained under high stress, resulting in less flows being created. This fix improves the resource consumption not only in the eBPF agent but also in **flowlogs-pipeline** and Loki. (NETOBSERV-1564)
- Previously, when the **workload_flows_total** metric was enabled instead of the **namespace_flows_total** metric, the health dashboard stopped showing **By namespace** flow charts. With this fix, the health dashboard now shows the flow charts when the **workload_flows_total** is enabled. (NETOBSERV-1746)
- Previously, when you used the **FlowMetrics** API to generate a custom metric and later modified its labels, such as by adding a new label, the metric stopped populating and an error was shown in the **flowlogs-pipeline** logs. With this fix, you can modify the labels, and the error is no longer raised in the **flowlogs-pipeline** logs. (NETOBSERV-1748)
- Previously, there was an inconsistency with the default Loki **WriteBatchSize** configuration: it was set to 100 KB in the **FlowCollector** CRD default, and 10 MB in the OLM sample or default configuration. Both are now aligned to 10 MB, which generally provides better performances and less resource footprint. (NETOBSERV-1766)
- Previously, the eBPF flow filter on ports was ignored if you did not specify a protocol. With this fix, you can set eBPF flow filters independently on ports and or protocols. (NETOBSERV-1779)
- Previously, traffic from Pods to Services was hidden from the **Topology view**. Only the return traffic from Services to Pods was visible. With this fix, that traffic is correctly displayed. (NETOBSERV-1788)
- Previously, non-cluster administrator users that had access to Network Observability saw an error in the console plugin when they tried to filter for something that triggered auto-completion, such as a namespace. With this fix, no error is displayed, and the auto-completion returns the expected results. (NETOBSERV-1798)
- When the secondary interface support was added, you had to iterate multiple times to register the per network namespace with the netlink to learn about interface notifications. At the same time, unsuccessful handlers caused a leaking file descriptor because with TCX hook, unlike TC, handlers needed to be explicitly removed when the interface went down. Furthermore, when the network namespace was deleted, there was no Go close channel event to terminate the netlink goroutine socket, which caused go threads to leak. Now, there are no longer leaking file descriptors or go threads when you create or delete pods. (NETOBSERV-1805)
- Previously, the ICMP type and value were displaying 'n/a' in the **Traffic flows** table even when related data was available in the flow JSON. With this fix, ICMP columns display related values as expected in the flow table. (NETOBSERV-1806)
- Previously in the console plugin, it wasn't always possible to filter for unset fields, such as unset DNS latency. With this fix, filtering on unset fields is now possible. (NETOBSERV-1816)
- Previously, when you cleared filters in the OpenShift web console plugin, sometimes the filters reappeared after you navigated to another page and returned to the page with filters. With this fix, filters do not unexpectedly reappear after they are cleared. (NETOBSERV-1733)

2.1.23. Network Observability Operator 1.7.0 known issues

You can review the following known issues for the Network Observability Operator 1.7.0 release.

- When you use the must-gather tool with network observability, logs are not collected when the cluster has FIPS enabled. (NETOBSERV-1830)

- When the **spec.networkPolicy** is enabled in the **FlowCollector**, which installs a network policy on the **netobserv** namespace, it is impossible to use the **FlowMetrics** API. The network policy blocks calls to the validation webhook. As a workaround, use the following network policy:

```
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: ''
  policyTypes:
    - Ingress
```

(NETOBSERV-193)

2.1.24. Network Observability Operator release notes 1.6.2 advisory

You can review the advisory for the Network Observability Operator 1.6.2 release.

- 2024:7074 Network Observability Operator 1.6.2

2.1.25. Network Observability Operator release notes 1.6.2 CVEs

You can review the CVEs for the Network Observability Operator 1.6.2 release.

- CVE-2024-24791

2.1.26. Network Observability Operator release notes 1.6.2 fixed issues

You can review the fixed issues for the Network Observability Operator 1.6.2 release.

- When the secondary interface support was added, there was a need to iterate multiple times to register the per network namespace with the netlink to learn about interface notifications. At the same time, unsuccessful handlers caused a leaking file descriptor because with TCX hook, unlike TC, handlers needed to be explicitly removed when the interface went down. Now, there are no longer leaking file descriptors when creating and deleting pods. (NETOBSERV-1805)

2.1.27. Network Observability Operator release notes 1.6.2 known issues

You can review the known issues for the Network Observability Operator 1.6.2 release.

- There was a compatibility issue with console plugins that would have prevented network observability from being installed on future versions of an OpenShift Container Platform cluster. By upgrading to 1.6.2, the compatibility issue is resolved and network observability can be installed as expected. (**NETOBSERV-1737**)

2.1.28. Network Observability Operator release notes 1.6.1 advisory

You can review the advisory for the Network Observability Operator 1.6.1 release.

- 2024:4785 Network Observability Operator 1.6.1

2.1.29. Network Observability Operator release notes 1.6.1 CVEs

You can review the CVEs for the Network Observability Operator 1.6.1 release.

- RHSA-2024:4237
- RHSA-2024:4212

2.1.30. Network Observability Operator release notes 1.6.1 fixed issues

You can review the fixed issues for the Network Observability Operator 1.6.1 release.

- Previously, information about packet drops, such as the cause and TCP state, was only available in the Loki datastore and not in Prometheus. For that reason, the drop statistics in the OpenShift web console plugin **Overview** was only available with Loki. With this fix, information about packet drops is also added to metrics, so you can view drops statistics when Loki is disabled. (**NETOBSERV-1649**)
- When the eBPF agent **PacketDrop** feature was enabled, and sampling was configured to a value greater than **1**, reported dropped bytes and dropped packets ignored the sampling configuration. While this was done on purpose, so as not to miss any drops, a side effect was that the reported proportion of drops compared with non-drops became biased. For example, at a very high sampling rate, such as **1:1000**, it was likely that almost all the traffic appears to be dropped when observed from the console plugin. With this fix, the sampling configuration is honored with dropped bytes and packets. (**NETOBSERV-1676**)
- Previously, the SR-IOV secondary interface was not detected if the interface was created first and then the eBPF agent was deployed. It was only detected if the agent was deployed first and then the SR-IOV interface was created. With this fix, the SR-IOV secondary interface is detected no matter the sequence of the deployments. (**NETOBSERV-1697**)
- Previously, when Loki was disabled, the **Topology** view in the OpenShift web console displayed the **Cluster** and **Zone** aggregation options in the slider beside the network topology diagram, even when the related features were not enabled. With this fix, the slider now only displays options according to the enabled features. (**NETOBSERV-1705**)
- Previously, when Loki was disabled, and the OpenShift web console was first loading, an error would occur: **Request failed with status code 400 Loki is disabled**. With this fix, the errors no longer occur. (**NETOBSERV-1706**)
- Previously, in the **Topology** view of the OpenShift web console, when clicking on the **Step into** icon next to any graph node, the filters were not applied as required in order to set the focus to the selected graph node, resulting in showing a wide view of the **Topology** view in the OpenShift

web console. With this fix, the filters are correctly set, effectively narrowing down the **Topology**. As part of this change, clicking the **Step into** icon on a **Node** now brings you to the **Resource** scope instead of the **Namespaces** scope. (NETOBSERV-1720)

- Previously, when Loki was disabled, in the **Topology** view of the OpenShift web console with the **Scope** set to **Owner**, clicking on the **Step into** icon next to any graph node would bring the **Scope** to **Resource**, which is not available without Loki, so an error message was shown. With this fix, the **Step into** icon is hidden in the **Owner** scope when Loki is disabled, so this scenario no longer occurs. (NETOBSERV-1721)

- Previously, when Loki was disabled, an error was displayed in the **Topology** view of the OpenShift web console when a group was set, but then the scope was changed so that the group becomes invalid. With this fix, the invalid group is removed, preventing the error. (NETOBSERV-1722)

- When creating a **FlowCollector** resource from the OpenShift web console **Form view**, as opposed to the **YAML view**, the following settings were incorrectly managed by the web console: **agent.ebpf.metrics.enable** and **processor.subnetLabels.openShiftAutoDetect**. These settings can only be disabled in the **YAML view**, not in the **Form view**. To avoid any confusion, these settings have been removed from the **Form view**. They are still accessible in the **YAML view**. (NETOBSERV-1731)

- Previously, the eBPF agent was unable to clean up traffic control flows installed before an ungraceful crash, for example a crash due to a SIGTERM signal. This led to the creation of multiple traffic control flow filters with the same name, since the older ones were not removed. With this fix, all previously installed traffic control flows are cleaned up when the agent starts, before installing new ones. (NETOBSERV-1732)

- Previously, when configuring custom subnet labels and keeping the OpenShift subnets auto-detection enabled, OpenShift subnets would take precedence over the custom ones, preventing the definition of custom labels for in cluster subnets. With this fix, custom defined subnets take precedence, allowing the definition of custom labels for in cluster subnets. (NETOBSERV-1734)

2.1.31. Network Observability Operator release notes 1.6.0 advisory

You can review the advisory for the Network Observability Operator 1.6.0 release.

- Network Observability Operator 1.6.0

2.1.32. Network Observability Operator 1.6.0 new features and enhancements

You can review the following new features and enhancements for the Network Observability Operator 1.6.0.

2.1.32.1. Enhanced use of Network Observability Operator without Loki

You can now use Prometheus metrics and rely less on Loki for storage when using the Network Observability Operator.

For more information, see:

- Network observability without Loki

2.1.32.2. Custom metrics API

You can create custom metrics out of flowlogs data by using the **FlowMetrics** API. Flowlogs data can be used with Prometheus labels to customize cluster information on your dashboards. You can add custom labels for any subnet that you want to identify in your flows and metrics. This enhancement can also be used to more easily identify external traffic by using the new labels **SrcSubnetLabel** and **DstSubnetLabel**, which exists both in flow logs and in metrics. Those fields are empty when there is external traffic, which gives a way to identify it.

For more information, see:

- Custom metrics
- FlowMetric API reference

2.1.32.3. eBPF performance enhancements

Experience improved performances of the eBPF agent, in terms of CPU and memory, with the following updates:

- The eBPF agent now uses TCX webhooks instead of TC.
- The **NetObserv / Health** dashboard has a new section that shows eBPF metrics.
  - Based on the new eBPF metrics, an alert notifies you when the eBPF agent is dropping flows.
- Loki storage demand decreases significantly now that duplicated flows are removed. Instead of having multiple, individual duplicated flows per network interface, there is one de-duplicated flow with a list of related network interfaces.

> **IMPORTANT**
>
> With the duplicated flows update, the **Interface** and **Interface Direction** fields in the **Network Traffic** table are renamed to **Interfaces** and **Interface Directions**, so any bookmarked **Quick filter** queries using these fields need to be updated to **interfaces** and **ifdirections**.

For more information, see:

- Using the eBPF agent alert
- Network observability metrics dashboards
- Filtering the network traffic

2.1.32.4. eBPF collection rule-based filtering

You can use rule-based filtering to reduce the volume of created flows. When this option is enabled, the **Netobserv / Health** dashboard for eBPF agent statistics has the **Filtered flows rate** view.

For more information, see:

- eBPF flow rule filter

2.1.33. Network Observability Operator 1.6.0 fixed issues

You can review the following fixed issues for the Network Observability Operator 1.6.0.

- Previously, a dead link to the OpenShift Container Platform documentation was displayed in the Operator Lifecycle Manager (OLM) form for the **FlowMetrics** API creation. Now the link has been updated to point to a valid page. (NETOBSERV-1607)
- Previously, the Network Observability Operator description in the Operator Hub displayed a broken link to the documentation. With this fix, this link is restored. (NETOBSERV-1544)
- Previously, if Loki was disabled and the Loki **Mode** was set to **LokiStack**, or if Loki manual TLS configuration was configured, the Network Observability Operator still tried to read the Loki CA certificates. With this fix, when Loki is disabled, the Loki certificates are not read, even if there are settings in the Loki configuration. (NETOBSERV-1647)
- Previously, the **oc must-gather** plugin for the Network Observability Operator was only working on the **amd64** architecture and failing on all others because the plugin was using **amd64** for the **oc** binary. Now, the Network Observability Operator **oc must-gather** plugin collects logs on any architecture platform.
- Previously, when filtering on IP addresses using **not equal to**, the Network Observability Operator would return a request error. Now, the IP filtering works in both **equal** and **not equal to** cases for IP addresses and ranges. (NETOBSERV-1630)
- Previously, when a user was not an admin, the error messages were not consistent with the selected tab of the **Network Traffic** view in the web console. Now, the **user not admin** error displays on any tab with improved display.(NETOBSERV-1621)

2.1.34. Network Observability Operator 1.6.0 known issues

You can review the following known issues for the Network Observability Operator 1.6.0.

- When the eBPF agent **PacketDrop** feature is enabled, and sampling is configured to a value greater than **1**, reported dropped bytes and dropped packets ignore the sampling configuration. While this is done on purpose to not miss any drops, a side effect is that the reported proportion of drops compared to non-drops becomes biased. For example, at a very high sampling rate, such as **1:1000**, it is likely that almost all the traffic appears to be dropped when observed from the console plugin. (NETOBSERV-1676)
- In the **Manage panels** window in the **Overview** tab, filtering on **total**, **bar**, **donut**, or **line** does not show any result. (NETOBSERV-1540)
- The SR-IOV secondary interface is not detected if the interface was created first and then the eBPF agent was deployed. It is only detected if the agent was deployed first and then the SR-IOV interface is created. (NETOBSERV-1697)
- When Loki is disabled, the **Topology** view in the OpenShift web console always shows the **Cluster** and **Zone** aggregation options in the slider beside the network topology diagram, even when the related features are not enabled. There is no specific workaround, besides ignoring these slider options. (NETOBSERV-1705)
- When Loki is disabled, and the OpenShift web console first loads, it might display an error: **Request failed with status code 400 Loki is disabled**. As a workaround, you can continue switching content on the **Network Traffic** page, such as clicking between the **Topology** and the **Overview** tabs. The error should disappear. (NETOBSERV-1706)

2.1.35. Network Observability Operator 1.5.0 advisory

You can view the following advisory for the Network Observability Operator 1.5 release.

Network Observability Operator 1.5.0

2.1.36. Network Observability Operator 1.5.0 new features and enhancements

You can view the following new features and enhancements for the Network Observability Operator 1.5 release.

2.1.36.1. DNS tracking enhancements

In 1.5, the TCP protocol is now supported in addition to UDP. New dashboards are also added to the **Overview** view of the Network Traffic page.

For more information, see:

- Configuring DNS tracking
- Working with DNS tracking

2.1.36.2. Round-trip time (RTT)

You can use TCP handshake Round-Trip Time (RTT) captured from the **fentry/tcp_rcv_established** Extended Berkeley Packet Filter (eBPF) hookpoint to read smoothed round-trip time (SRTT) and analyze network flows. In the **Overview**, **Network Traffic**, and **Topology** pages in web console, you can monitor network traffic and troubleshoot with RTT metrics, filtering, and edge labeling.

For more information, see:

- RTT Overview
- Working with RTT

2.1.36.3. Metrics, dashboards, and alerts enhancements

The network observability metrics dashboards in **Observe → Dashboards → NetObserv** have new metrics types you can use to create Prometheus alerts. You can now define available metrics in the **includeList** specification. In previous releases, these metrics were defined in the **ignoreTags** specification.

For a complete list of these metrics, see:

- Network observability metrics

2.1.36.4. Improvements for network observability without Loki

You can create Prometheus alerts for the **Netobserv** dashboard using DNS, Packet drop, and RTT metrics, even if you don’t use Loki. In the previous version of network observability, 1.4, these metrics were only available for querying and analysis in the **Network Traffic**, **Overview**, and **Topology** views, which are not available without Loki.

For more information, see:

- Network observability metrics

2.1.36.5. Availability zones

You can configure the **FlowCollector** resource to collect information about the cluster availability zones. This configuration enriches the network flow data with the **topology.kubernetes.io/zone** label value applied to the nodes.

For more information, see:

- Working with availability zones

2.1.36.6. Notable enhancements

The 1.5 release of the Network Observability Operator adds improvements and new capabilities to the OpenShift Container Platform web console plugin and the Operator configuration.

2.1.36.7. Performance enhancements

- The **spec.agent.ebpf.kafkaBatchSize** default is changed from **10MB** to **1MB** to enhance eBPF performance when using Kafka.

  IMPORTANT

  When upgrading from an existing installation, this new value is not set automatically in the configuration. If you monitor a performance regression with the eBPF Agent memory consumption after upgrading, you might consider reducing the **kafkaBatchSize** to the new value.

2.1.36.8. Web console enhancements:

- There are new panels added to the **Overview** view for DNS and RTT: Min, Max, P90, P99.
- There are new panel display options added:
  - Focus on one panel while keeping others viewable but with smaller focus.
  - Switch graph type.
  - Show **Top** and **Overall**.
- A collection latency warning is shown in the **Custom time range** window.
- There is enhanced visibility for the contents of the **Manage panels** and **Manage columns** pop-up windows.
- The Differentiated Services Code Point (DSCP) field for egress QoS is available for filtering QoS DSCP in the web console **Network Traffic** page.

2.1.36.9. Configuration enhancements:

- The **LokiStack** mode in the **spec.loki.mode** specification simplifies installation by automatically setting URLs, TLS, cluster roles and a cluster role binding, as well as the **authToken** value. The **Manual** mode allows more control over configuration of these settings.

- The API version changes from **flows.netobserv.io/v1beta1** to **flows.netobserv.io/v1beta2**.

2.1.37. Network Observability Operator 1.5.0 fixed issues

You can view the following fixed issues for the Network Observability Operator 1.5 release.

- Previously, it was not possible to register the console plugin manually in the web console interface if the automatic registration of the console plugin was disabled. If the **spec.console.register** value was set to **false** in the **FlowCollector** resource, the Operator would override and erase the plugin registration. With this fix, setting the **spec.console.register** value to **false** does not impact the console plugin registration or registration removal. As a result, the plugin can be safely registered manually. (NETOBSERV-1134)
- Previously, using the default metrics settings, the **NetObserv/Health** dashboard was showing an empty graph named **Flows Overhead**. This metric was only available by removing "namespaces-flows" and "namespaces" from the **ignoreTags** list. With this fix, this metric is visible when you use the default metrics setting. (NETOBSERV-1351)
- Previously, the node on which the eBPF Agent was running would not resolve with a specific cluster configuration. This resulted in cascading consequences that culminated in a failure to provide some of the traffic metrics. With this fix, the eBPF agent's node IP is safely provided by the Operator, inferred from the pod status. Now, the missing metrics are restored. (NETOBSERV-1430)
- Previously, the Loki error 'Input size too long' error for the Loki Operator did not include additional information to troubleshoot the problem. With this fix, help is directly displayed in the web console next to the error with a direct link for more guidance. (NETOBSERV-1464)
- Previously, the console plugin read timeout was forced to 30s. With the **FlowCollector v1beta2** API update, you can configure the **spec.loki.readTimeout** specification to update this value according to the Loki Operator **queryTimeout** limit. (NETOBSERV-1443)
- Previously, the Operator bundle did not display some of the supported features by CSV annotations as expected, such as **features.operators.openshift.io/...** With this fix, these annotations are set in the CSV as expected. (NETOBSERV-1305)
- Previously, the **FlowCollector** status sometimes oscillated between **DeploymentInProgress** and **Ready** states during reconciliation. With this fix, the status only becomes **Ready** when all of the underlying components are fully ready. (NETOBSERV-1293)

2.1.38. Network Observability Operator 1.5.0 known issues

You can view the following known issues for the Network Observability Operator 1.5 release.

- When trying to access the web console, cache issues on OCP 4.14.10 prevent access to the **Observe** view. The web console shows the error message: **Failed to get a valid plugin manifest from /api/plugins/monitoring-plugin/**. The recommended workaround is to update the cluster to the latest minor version. If this does not work, you need to apply the workarounds described in this Red Hat Knowledgebase article .(NETOBSERV-1493)
- Since the 1.3.0 release of the Network Observability Operator, installing the Operator causes a warning kernel taint to appear. The reason for this error is that the network observability eBPF agent has memory constraints that prevent preallocating the entire hashmap table. The Operator eBPF agent sets the **BPF_F_NO_PREALLOC** flag so that pre-allocation is disabled when the hashmap is too memory expansive.

2.1.39. Network Observability Operator 1.4.2 advisory

The following advisory is available for the Network Observability Operator 1.4.2:

- 2023:6787 Network Observability Operator 1.4.2

2.1.40. Network Observability Operator 1.4.2 CVEs

You can review the following CVEs in the Network Observability Operator 1.4.2 release.

- 2023-39325
- 2023-44487

2.1.41. Network Observability Operator 1.4.1 advisory

You can review the following advisory for the Network Observability Operator 1.4.1.

- 2023:5974 Network Observability Operator 1.4.1

2.1.42. Network Observability Operator release 1.4.1 CVEs

You can review the following CVEs in the Network Observability Operator 1.4.1 release.

- 2023-44487
- 2023-39325
- 2023-29406
- 2023-29409
- 2023-39322
- 2023-39318
- 2023-39319
- 2023-39321

2.1.43. Network Observability Operator release notes 1.4.1 fixed issues

You can review the following fixed issues in the Network Observability Operator 1.4.1 release.

- In 1.4, there was a known issue when sending network flow data to Kafka. The Kafka message key was ignored, causing an error with connection tracking. Now the key is used for partitioning, so each flow from the same connection is sent to the same processor. (**NETOBSERV-926**)
- In 1.4, the **Inner** flow direction was introduced to account for flows between pods running on the same node. Flows with the **Inner** direction were not taken into account in the generated Prometheus metrics derived from flows, resulting in under-evaluated bytes and packets rates. Now, derived metrics are including flows with the **Inner** direction, providing correct bytes and packets rates. (**NETOBSERV-1344**)

2.1.44. Network observability release notes 1.4.0 advisory

You can review the following advisory for the Network Observability Operator 1.4.0 release.

- RHSA-2023:5379 Network Observability Operator 1.4.0

2.1.45. Network observability release notes 1.4.0 new features and enhancements

You can review the following new features and enhancements in the Network Observability Operator 1.4.0 release.

2.1.45.1. Notable enhancements

The 1.4 release of the Network Observability Operator adds improvements and new capabilities to the OpenShift Container Platform web console plugin and the Operator configuration.

2.1.45.2. Web console enhancements:

- In the **Query Options**, the **Duplicate flows** checkbox is added to choose whether or not to show duplicated flows.
- You can now filter source and destination traffic with ↑ **One-way**, ↑↓ **Back-and-forth**, and **Swap** filters.
- The network observability metrics dashboards in **Observe** → **Dashboards** → **NetObserv** and **NetObserv / Health** are modified as follows:
  - The **NetObserv** dashboard shows top bytes, packets sent, packets received per nodes, namespaces, and workloads. Flow graphs are removed from this dashboard.
  - The **NetObserv / Health** dashboard shows flows overhead as well as top flow rates per nodes, namespaces, and workloads.
  - Infrastructure and Application metrics are shown in a split-view for namespaces and workloads.

For more information, see:

- Network observability metrics dashboards
- Quick filters

2.1.45.3. Configuration enhancements:

- You now have the option to specify different namespaces for any configured ConfigMap or Secret reference, such as in certificates configuration.
- The **spec.processor.clusterName** parameter is added so that the name of the cluster appears in the flows data. This is useful in a multi-cluster context. When using OpenShift Container Platform, leave empty to make it automatically determined.

For more information, see:

- Flow Collector sample resource
- Flow Collector API Reference

2.1.45.4. Network observability without Loki

The Network Observability Operator is now functional and usable without Loki. If Loki is not installed, it can only export flows to KAFKA or IPFIX format and provide metrics in the network observability metrics dashboards.

For more information, see:

- Network observability without Loki

2.1.45.5. DNS tracking

In 1.4, the Network Observability Operator makes use of eBPF tracepoint hooks to enable DNS tracking. You can monitor your network, conduct security analysis, and troubleshoot DNS issues in the **Network Traffic** and **Overview** pages in the web console.

For more information, see:

- Configuring DNS tracking
- Working with DNS tracking

2.1.45.6. SR-IOV support

You can now collect traffic from a cluster with Single Root I/O Virtualization (SR-IOV) device.

For more information, see:

- Configuring the monitoring of SR-IOV interface traffic

2.1.45.7. IPFIX exporter support

You can now export eBPF-enriched network flows to the IPFIX collector.

For more information, see:

- Export enriched network flow data

2.1.45.8. Packet drops

In the 1.4 release of the Network Observability Operator, eBPF tracepoint hooks are used to enable packet drop tracking. You can now detect and analyze the cause for packet drops and make decisions to optimize network performance. In OpenShift Container Platform 4.14 and later, both host drops and OVS drops are detected. In OpenShift Container Platform 4.13, only host drops are detected.

For more information, see:

- Configuring packet drop tracking
- Working with packet drops

2.1.45.9. s390x architecture support

Network Observability Operator can now run on **s390x** architecture. Previously it ran on **amd64**, **ppc64le**, or **arm64**.

2.1.46. Network observability release notes 1.4.0 removed features

You can review the following removed features from the Network Observability Operator 1.4.0 release.

2.1.46.1. Channel removal

You must switch your channel from **v1.0.x** to **stable** to receive the latest Operator updates. The **v1.0.x** channel is now removed.

2.1.47. Network observability release notes 1.4.0 fixed issues

You can review the following fixed issues in the Network Observability Operator 1.4.0 release.

- Previously, the Prometheus metrics exported by network observability were computed out of potentially duplicated network flows. In the related dashboards, from **Observe → Dashboards**, this could result in potentially doubled rates. Note that dashboards from the **Network Traffic** view were not affected. Now, network flows are filtered to eliminate duplicates before metrics calculation, which results in correct traffic rates displayed in the dashboards. (NETOBSERV-1131)

- Previously, the Network Observability Operator agents were not able to capture traffic on network interfaces when configured with Multus or SR-IOV, non-default network namespaces. Now, all available network namespaces are recognized and used for capturing flows, allowing capturing traffic for SR-IOV. There are configurations needed for the **FlowCollector** and **SRIOVnetwork** custom resource to collect traffic. (NETOBSERV-1283)

- Previously, in the Network Observability Operator details from **Operators → Installed Operators**, the **FlowCollector Status** field might have reported incorrect information about the state of the deployment. The status field now shows the proper conditions with improved messages. The history of events is kept, ordered by event date. (NETOBSERV-1224)

- Previously, during spikes of network traffic load, certain eBPF pods were OOM-killed and went into a **CrashLoopBackOff** state. Now, the **eBPF** agent memory footprint is improved, so pods are not OOM-killed and entering a **CrashLoopBackOff** state. (NETOBSERV-975)

- Previously when **processor.metrics.tls** was set to **PROVIDED** the **insecureSkipVerify** option value was forced to be **true**. Now you can set **insecureSkipVerify** to **true** or **false**, and provide a CA certificate if needed. (NETOBSERV-1087)

2.1.48. Network observability release notes 1.4.0 known issues

You can review the following known issues in the Network Observability Operator 1.4.0 release.

- Since the 1.2.0 release of the Network Observability Operator, using Loki Operator 5.6, a Loki certificate change periodically affects the **flowlogs-pipeline** pods and results in dropped flows rather than flows written to Loki. The problem self-corrects after some time, but it still causes temporary flow data loss during the Loki certificate change. This issue has only been observed in large-scale environments of 120 nodes or greater. (NETOBSERV-980)

- Currently, when **spec.agent.ebpf.features** includes DNSTracking, larger DNS packets require the **eBPF** agent to look for DNS header outside of the 1st socket buffer (SKB) segment. A new **eBPF** agent helper function needs to be implemented to support it. Currently, there is no workaround for this issue. (NETOBSERV-1304)

- Currently, when **spec.agent.ebpf.features** includes DNSTracking, DNS over TCP packets requires the **eBPF** agent to look for DNS header outside of the 1st SKB segment. A new **eBPF** agent helper function needs to be implemented to support it. Currently, there is no workaround for this issue. (NETOBSERV-1245)

- Currently, when using a **KAFKA** deployment model, if conversation tracking is configured, conversation events might be duplicated across Kafka consumers, resulting in inconsistent tracking of conversations, and incorrect volumetric data. For that reason, it is not recommended to configure conversation tracking when **deploymentModel** is set to **KAFKA**. (NETOBSERV-926)

- Currently, when the **processor.metrics.server.tls.type** is configured to use a **PROVIDED** certificate, the operator enters an unsteady state that might affect its performance and resource consumption. It is recommended to not use a **PROVIDED** certificate until this issue is resolved, and instead using an auto-generated certificate, setting **processor.metrics.server.tls.type** to **AUTO**. (NETOBSERV-1293

- Since the 1.3.0 release of the Network Observability Operator, installing the Operator causes a warning kernel taint to appear. The reason for this error is that the network observability eBPF agent has memory constraints that prevent preallocating the entire hashmap table. The Operator eBPF agent sets the **BPF_F_NO_PREALLOC** flag so that pre-allocation is disabled when the hashmap is too memory expansive.

2.1.49. Network Observability Operator 1.3.0 advisory

You can review the following advisory in the Network Observability Operator 1.3.0 release.

- RHSA-2023:3905 Network Observability Operator 1.3.0

2.1.50. Network Observability Operator 1.3.0 new features and enhancements

You can review the following new features and enhancements in the Network Observability Operator 1.3.0 release.

2.1.50.1. Multi-tenancy in network observability

- System administrators can allow and restrict individual user access, or group access, to the flows stored in Loki. For more information, see "Multi-tenancy in network observability".

2.1.50.2. Flow-based metrics dashboard

- This release adds a new dashboard, which provides an overview of the network flows in your OpenShift Container Platform cluster. For more information, see "Network observability metrics dashboards".

2.1.50.3. Troubleshooting with the must-gather tool

- Information about the Network Observability Operator can now be included in the must-gather data for troubleshooting. For more information, see "Network observability must-gather".

2.1.50.4. Multiple architectures now supported

- Network Observability Operator can now run on an **amd64**, **ppc64le**, or **arm64** architectures. Previously, it only ran on **amd64**.

2.1.51. Network Observability Operator 1.3.0 deprecated features

You can review the following deprecated features in the Network Observability Operator 1.3.0 release.

2.1.51.1. Channel deprecation

You must switch your channel from **v1.0.x** to **stable** to receive future Operator updates. The **v1.0.x** channel is deprecated and planned for removal in the next release.

2.1.51.2. Deprecated configuration parameter setting

The release of Network Observability Operator 1.3 deprecates the **spec.Loki.authToken HOST** setting. When using the Loki Operator, you must now only use the **FORWARD** setting.

2.1.52. Network Observability Operator 1.3.0 fixed issues

You can review the following fixed issues in the Network Observability Operator 1.3.0 release.

- Previously, when the Operator was installed from the CLI, the **Role** and **RoleBinding** that are necessary for the Cluster Monitoring Operator to read the metrics were not installed as expected. The issue did not occur when the operator was installed from the web console. Now, either way of installing the Operator installs the required **Role** and **RoleBinding**. (NETOBSERV-1003)

- Since version 1.2, the Network Observability Operator can raise alerts when a problem occurs with the flows collection. Previously, due to a bug, the related configuration to disable alerts, **spec.processor.metrics.disableAlerts** was not working as expected and sometimes ineffectual. Now, this configuration is fixed so that it is possible to disable the alerts. (NETOBSERV-976)

- Previously, when network observability was configured with **spec.loki.authToken** set to **DISABLED**, only a **kubeadmin** cluster administrator was able to view network flows. Other types of cluster administrators received authorization failure. Now, any cluster administrator is able to view network flows. (NETOBSERV-972)

- Previously, a bug prevented users from setting **spec.consolePlugin.portNaming.enable** to **false**. Now, this setting can be set to **false** to disable port-to-service name translation. (NETOBSERV-971)

- Previously, the metrics exposed by the console plugin were not collected by the Cluster Monitoring Operator (Prometheus), due to an incorrect configuration. Now the configuration has been fixed so that the console plugin metrics are correctly collected and accessible from the OpenShift Container Platform web console. (NETOBSERV-765)

- Previously, when **processor.metrics.tls** was set to **AUTO** in the **FlowCollector**, the **flowlogs-pipeline servicemonitor** did not adapt the appropriate TLS scheme, and metrics were not visible in the web console. Now the issue is fixed for AUTO mode. (NETOBSERV-1070)

- Previously, certificate configuration, such as used for Kafka and Loki, did not allow specifying a namespace field, implying that the certificates had to be in the same namespace where network observability is deployed. Moreover, when using Kafka with TLS/mTLS, the user had to manually copy the certificate(s) to the privileged namespace where the **eBPF** agent pods are deployed and manually manage certificate updates, such as in the case of certificate rotation. Now, network observability setup is simplified by adding a namespace field for certificates in the **FlowCollector** resource. As a result, users can now install Loki or Kafka in different namespaces without needing to manually copy their certificates in the network observability namespace. The original certificates are watched so that the copies are automatically updated when needed. (NETOBSERV-773)

- Previously, the SCTP, ICMPv4 and ICMPv6 protocols were not covered by the network observability agents, resulting in a less comprehensive network flows coverage. These protocols are now recognized to improve the flows coverage. (NETOBSERV-934)

2.1.53. Network Observability Operator 1.3.0 known issues

You can review the following issues and their workarounds, if available, to troubleshoot issues with the Network Observability Operator 1.3.0 release.

- When **processor.metrics.tls** is set to **PROVIDED** in the **FlowCollector**, the **flowlogs-pipeline servicemonitor** is not adapted to the TLS scheme. (NETOBSERV-1087)
- Since the 1.2.0 release of the Network Observability Operator, using Loki Operator 5.6, a Loki certificate change periodically affects the **flowlogs-pipeline** pods and results in dropped flows rather than flows written to Loki. The problem self-corrects after some time, but it still causes temporary flow data loss during the Loki certificate change. This issue has only been observed in large-scale environments of 120 nodes or greater.(NETOBSERV-980)
- When you install the Operator, a warning kernel taint can appear. The reason for this error is that the network observability eBPF agent has memory constraints that prevent preallocating the entire hashmap table. The Operator eBPF agent sets the **BPF_F_NO_PREALLOC** flag so that pre-allocation is disabled when the hashmap is too memory expansive.

2.1.54. Network observability release notes 1.2.0 preparing for the next update

Switch the Network Observability Operator's update channel from the deprecated **v1.0.x** to the **stable** channel to continue receiving future releases and updates.

The subscription of an installed Operator specifies an update channel that tracks and receives updates for the Operator. Until the 1.2 release of the Network Observability Operator, the only channel available was **v1.0.x**. The 1.2 release of the Network Observability Operator introduces the **stable** update channel for tracking and receiving updates. You must switch your channel from **v1.0.x** to **stable** to receive future Operator updates. The **v1.0.x** channel is deprecated and planned for removal in a following release.

2.1.55. Network Observability Operator 1.2.0 advisory

You can view the following advisory for the Network Observability Operator 1.2.0 release.

- RHSA-2023:1817 Network Observability Operator 1.2.0

2.1.56. Network Observability Operator 1.2.0 new features and enhancements

You can view the following new features and enhancements for the Network Observability Operator 1.2.0 release.

2.1.56.1. Histogram in Traffic Flows view

You can now choose to show a histogram of flows over time. The histogram enables you to visualize the history of flows without hitting the Loki query limit. For more information, see "Using the histogram".

2.1.56.2. Conversation tracking

You can now query flows by **Log Type**, which enables grouping network flows that are part of the same conversation. For more information, see "Working with conversations".

2.1.56.3. Network observability health alerts

The Network Observability Operator now creates automatic alerts if the **flowlogs-pipeline** is dropping flows because of errors at the write stage or if the Loki ingestion rate limit has been reached. For more information, see "Health dashboards".

2.1.57. Network Observability Operator 1.2.0 bug fixes

You can view the following fixed issues for the Network Observability Operator 1.2.0 release.

- Previously, after changing the **namespace** value in the FlowCollector spec, **eBPF** agent pods running in the previous namespace were not appropriately deleted. Now, the pods running in the previous namespace are appropriately deleted. (NETOBSERV-774)
- Previously, after changing the **caCert.name** value in the FlowCollector spec (such as in Loki section), FlowLogs-Pipeline pods and Console plug-in pods were not restarted, therefore they were unaware of the configuration change. Now, the pods are restarted, so they get the configuration change. (NETOBSERV-772)
- Previously, network flows between pods running on different nodes were sometimes not correctly identified as being duplicates because they are captured by different network interfaces. This resulted in over-estimated metrics displayed in the console plug-in. Now, flows are correctly identified as duplicates, and the console plug-in displays accurate metrics. (NETOBSERV-755)
- The "reporter" option in the console plug-in is used to filter flows based on the observation point of either source node or destination node. Previously, this option mixed the flows regardless of the node observation point. This was due to network flows being incorrectly reported as Ingress or Egress at the node level. Now, the network flow direction reporting is correct. The "reporter" option filters for source observation point, or destination observation point, as expected. (NETOBSERV-696)
- Previously, for agents configured to send flows directly to the processor as gRPC+protobuf requests, the submitted payload could be too large and is rejected by the processors' GRPC server. This occurred under very-high-load scenarios and with only some configurations of the agent. The agent logged an error message, such as: *grpc: received message larger than max* . As a consequence, there was information loss about those flows. Now, the gRPC payload is split into several messages when the size exceeds a threshold. As a result, the server maintains connectivity. (NETOBSERV-617)

2.1.58. Network Observability Operator 1.2.0 known issues

You can review the following issues and their workarounds, if available, to troubleshoot issues with the Network Observability Operator 1.2.0 release.

- In the 1.2.0 release of the Network Observability Operator, using Loki Operator 5.6, a Loki certificate transition periodically affects the **flowlogs-pipeline** pods and results in dropped flows rather than flows written to Loki. The problem self-corrects after some time, but it still causes temporary flow data loss during the Loki certificate transition. (NETOBSERV-980)

2.1.59. Network Observability Operator 1.2.0 notable technical changes

The Network Observability Operator 1.2.0 release requires installation in the **openshift-netobserv-operator** namespace due to new technical changes. Users who previously used a custom namespace must delete the old instance and reinstall the Operator.

Previously, you could install the Network Observability Operator using a custom namespace. This release introduces the **conversion webhook** which changes the **ClusterServiceVersion**. Because of this change, all the available namespaces are no longer listed. Additionally, to enable Operator metrics collection, namespaces that are shared with other Operators, like the **openshift-operators** namespace, cannot be used.

Now, the Operator must be installed in the **openshift-netobserv-operator** namespace.

You cannot automatically upgrade to the new Operator version if you previously installed the Network Observability Operator using a custom namespace. If you previously installed the Operator using a custom namespace, you must delete the instance of the Operator that was installed and re-install your operator in the **openshift-netobserv-operator** namespace. It is important to note that custom namespaces, such as the commonly used **netobserv** namespace, are still possible for the **FlowCollector**, Loki, Kafka, and other plug-ins.

- NETOBSERV-907
- NETOBSERV-956

2.1.60. Network Observability Operator 1.1.0 enhancements

You can view the following advisory for the Network Observability Operator 1.1.0:

- RHSA-2023:0786 Network Observability Operator Security Advisory Update

The Network Observability Operator is now stable and the release channel is upgraded to **v1.1.0**.

2.1.61. Network Observability Operator 1.1.0 fixed issues

You can view the following fixed issues for the Network Observability Operator 1.1.0 release.

- Previously, unless the Loki **authToken** configuration was set to **FORWARD** mode, authentication was not enforced, allowing unauthorized users to retrieve flows. Now, regardless of the Loki **authToken** mode, only cluster administrators can retrieve flows. (BZ#2169468)

2.1.62. Additional resources

- Multi-tenancy in network observability
- Network observability metrics dashboards
- Network observability must-gather
- Using the histogram
- Working with conversations
- Health dashboards

CHAPTER 3. ABOUT NETWORK OBSERVABILITY

Use the Network Observability Operator to observe network traffic via **eBPF** technology, providing troubleshooting insights through Prometheus metrics and Loki logs.

You can view and analyze this stored information in the OpenShift Container Platform console for further insight and troubleshooting.

3.1. NETWORK OBSERVABILITY OPERATOR

The Network Observability Operator provides the cluster-scoped **FlowCollector** API custom resource, which manages a pipeline of eBPF agents and services that collect, enrich, and store network flows in Loki or Prometheus.

A **FlowCollector** instance deploys pods and services that form a monitoring pipeline.

The **eBPF** agent is deployed as a **daemonset** object and creates the network flows. The pipeline collects and enriches network flows with Kubernetes metadata before storing them in Loki or generating Prometheus metrics.

3.2. OPTIONAL DEPENDENCIES OF THE NETWORK OBSERVABILITY OPERATOR

Integrate the Network Observability Operator with optional dependencies, such as the Loki Operator for flow storage and AMQ Streams (Kafka) for resilient, large-scale data handling and scalability.

Supported optional dependencies include the Loki Operator for flow storage, and AMQ Streams for large-scale data handling with Kafka.

Loki Operator

You can use Loki as the backend to store all collected flows with a maximal level of details. It is recommended to use the Red Hat supported Loki Operator to install Loki. You can also choose to use network observability without Loki, but you need to consider some factors. For more information, see "Network observability without Loki".

AMQ Streams Operator

Kafka provides scalability, resiliency and high availability in the OpenShift Container Platform cluster for large scale deployments.

> **NOTE**
>
> If you choose to use Kafka, it is recommended to use Red Hat supported AMQ Streams Operator.

Additional resources

- Network observability without Loki

3.3. OPENSHIFT CONTAINER PLATFORM CONSOLE INTEGRATION

The Network Observability Operator integrates with the OpenShift Container Platform console, providing an overview, topology view, and traffic flow tables.

The Network observability metrics dashboards in **Observe → Dashboards** are available only to users with administrator access.

> **NOTE**
>
> To enable multi-tenancy for developer access and for administrators with limited access to namespaces, you must specify permissions by defining roles. For more information, see "Enabling multi-tenancy in network observability".

Additional resources

- Enabling multi-tenancy in network observability

3.3.1. Network observability metrics dashboards

Review the network observability metrics dashboards in the OpenShift Container Platform console, which provide overall traffic flow aggregation, filtering options, and dedicated dashboards for monitoring operator health.

In the OpenShift Container Platform console on the **Overview** tab, you can view the overall aggregated metrics of the network traffic flow on the cluster. You can choose to display the information by cluster, node, namespace, owner, pod, and service. Filters and display options can further refine the metrics. For more information, see "Observing the network traffic from the Overview view".

In **Observe → Dashboards**, the **Netobserv** dashboards provide a quick overview of the network flows in your OpenShift Container Platform cluster. The **Netobserv/Health** dashboard provides metrics about the health of the Operator. For more information, see "Network observability metrics" and "Viewing health information".

Additional resources

- Observing the network traffic from the Overview view
- Network observability metrics
- Health dashboards

3.3.2. Network observability topology views

The network observability topology view in the OpenShift Container Platform console displays a graphical representation of traffic flow between components, which you can refine using various filters and display options.

The OpenShift Container Platform console offers the **Topology** tab which represents traffic between the OpenShift Container Platform components as a network graph. You can refine the graph by using the filters and display options. You can access the information for cluster, zone, udn, node, namespace, owner, pod, and service.

3.3.3. Traffic flow tables

The **Traffic flow** tables in the OpenShift Container Platform web console provide a detailed view of raw network flows, offering powerful filtering options and configurable columns for in-depth analysis.

The **Traffic flows** tab in the OpenShift Container Platform web console displays the data of the network flows and the amount of traffic.

3.4. NETWORK OBSERVABILITY CLI

The Network Observability CLI (**oc netobserv**) is a lightweight tool that streams flow and packet data for quick, live insight into networking issues without requiring the full Network Observability Operator installation.

The Network Observability CLI is a flow and packet visualization tool that relies on eBPF agents to stream collected data to an ephemeral collector pod. It requires no persistent storage during the capture. After the run, the output is transferred to your local machine. This enables quick, live insight into packets and flow data without installing the Network Observability Operator.

CHAPTER 4. INSTALLING THE NETWORK OBSERVABILITY OPERATOR

Installing the Loki Operator is recommended before using the Network Observability Operator. You can use network observability without Loki, but special considerations apply if you only need metrics or external exporters.

The Loki Operator integrates a gateway that implements multi-tenancy and authentication with Loki for data flow storage. The **LokiStack** resource manages Loki, which is a scalable, highly-available, multi-tenant log aggregation system, and a web proxy with OpenShift Container Platform authentication. The **LokiStack** proxy uses OpenShift Container Platform authentication to enforce multi-tenancy and facilitate the saving and indexing of data in Loki log stores.

4.1. NETWORK OBSERVABILITY WITHOUT LOKI

Compare the features available with network observability with and without installing the Loki Operator.

If you only want to export flows to a Kafka consumer or IPFIX collector, or you only need dashboard metrics, then you do not need to install Loki or provide storage for Loki. The following table compares available features with and without Loki.

Table 4.1. Comparison of feature availability with and without Loki

	With Loki	Without Loki
Exporters	X	X
Multi-tenancy	X	X
Complete filtering and aggregations capabilities [1]	X	
Partial filtering and aggregations capabilities [2]	X	X
Flow-based metrics and dashboards	X	X
Traffic flows view overview [3]	X	X
Traffic flows view table	X	
Topology view	X	X
OpenShift Container Platform console Network Traffic tab integration	X	X

1. Such as per pod.

2. Such as per workload or namespace.

3. Statistics on packet drops are only available with Loki.

Additional resources

- Export enriched network flow data

4.2. INSTALLING THE LOKI OPERATOR

Install the supported Loki Operator version from the software catalog to enable the secure **LokiStack** instance, which provides automatic in-cluster authentication and authorization for network observability.

The Loki Operator versions 6.0+ are the supported Loki Operator versions for network observability; these versions provide the ability to create a **LokiStack** instance using the **openshift-network** tenant configuration mode and provide fully-automatic, in-cluster authentication and authorization support for network observability.

Prerequisites

- You have administrator permissions.
- You have access to the OpenShift Container Platform web console.
- You have access to a supported object store. For example: AWS S3, Google Cloud Storage, Azure, Swift, Minio, or OpenShift Data Foundation.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Choose **Loki Operator** from the list of available Operators, and click **Install**.
3. Under **Installation Mode**, select **All namespaces on the cluster**.

Verification

1. Verify that you installed the Loki Operator. Visit the **Operators → Installed Operators** page and look for **Loki Operator**.
2. Verify that **Loki Operator** is listed with **Status** as **Succeeded** in all the projects.

IMPORTANT

To uninstall Loki, refer to the uninstallation process that corresponds with the method you used to install Loki. You might have remaining **ClusterRoles** and **ClusterRoleBindings**, data stored in object store, and persistent volume that must be removed.

4.2.1. Creating a secret for Loki storage

Create a secret with cloud storage credentials, such as for Amazon Web Services (AWS), to allow the Loki Operator to access the necessary object store for log persistence.

The Loki Operator supports a few log storage options, such as AWS S3, Google Cloud Storage, Azure, Swift, Minio, OpenShift Data Foundation. The following example shows how to create a secret for AWS S3 storage. The secret created in this example, **loki-s3**, is referenced in "Creating a LokiStack custom resource". You can create this secret in the web console or CLI.

Procedure

1. Using the web console, navigate to the **Project → All Projects** dropdown and select **Create Project**.

2. Name the project **netobserv** and click **Create**.

3. Navigate to the Import icon, **+**, in the top right corner. Paste your YAML file into the editor. The following shows an example secret YAML file for S3 storage:

```
apiVersion: v1
kind: Secret
metadata:
  name: loki-s3
  namespace: netobserv (1)
stringData:
  access_key_id: QUtJQUlPU0ZPRE5ON0VYQU1QTEUK
  access_key_secret:
d0phbHJYVXRuRkVNSS9LN01ERU5HL2JQeFJmaUNZRVhBTVBMRUtFWQo=
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1
```

(1) The installation examples in this documentation use the same namespace, **netobserv**, across all components. You can optionally use a different namespace for the different components

Verification

- After you create the secret, you view the secret listed under **Workloads → Secrets** in the web console.

Additional resources

- Creating a LokiStack custom resource
- Flow Collector API Reference
- Flow Collector sample resource

4.2.2. Creating a LokiStack custom resource

Deploy the **LokiStack** custom resource using the web console or OpenShift CLI (**oc**), ensuring you configure the correct namespace, deployment size, and secret name for Loki object storage.

You can deploy a **LokiStack** custom resource (CR) to create a namespace or new project.

Procedure

1. Navigate to **Operators** → **Installed Operators**, viewing **All projects** from the **Project** dropdown.

2. Look for **Loki Operator**. In the details, under **Provided APIs**, select **LokiStack**.

3. Click **Create LokiStack**.

4. Ensure the following fields are specified in either **Form View** or **YAML view**:

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

1. The installation examples in this documentation use the same namespace, **netobserv**, across all components. You can optionally use a different namespace.

2. Specify the deployment size. In the Loki Operator 5.8 and later versions, the supported size options for production instances of Loki are **1x.extra-small**, **1x.small**, or **1x.medium**.

   IMPORTANT

   It is not possible to change the number **1x** for the deployment size.

3. Use a storage class name that is available on the cluster for **ReadWriteOnce** access mode. For best performance, specify a storage class that allocates block storage. You can use **oc get storageclasses** to see what is available on your cluster.

   IMPORTANT

   You must not reuse the same **LokiStack** CR that is used for logging.

5. Click **Create**.

4.2.3. Creating a new group for the cluster-admin user role

IMPORTANT

Querying application logs for multiple namespaces as a **cluster-admin** user, where the sum total of characters of all of the namespaces in the cluster is greater than 5120, results in the error **Parse error: input size too long (XXXX > 5120)**. For better control over access to logs in LokiStack, make the **cluster-admin** user a member of the **cluster-admin** group. If the **cluster-admin** group does not exist, create it and add the desired users to it.

Use the following procedure to create a new group for users with **cluster-admin** permissions.

Procedure

1. Enter the following command to create a new group:

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

2. Enter the following command to add the desired user to the **cluster-admin** group:

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

3. Enter the following command to add **cluster-admin** user role to the group:

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

4.2.4. Custom admin group access

If you need to see cluster-wide logs without necessarily being an administrator, or if you already have any group defined that you want to use here, you can specify a custom group using the **adminGroup** field. Users who are members of any group specified in the **adminGroups** field of the **LokiStack** custom resource (CR) have the same read access to logs as administrators.

Administrator users have access to all network logs across the cluster.

Example LokiStack CR

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

(1) Custom admin groups are only available in this mode.

(2) Entering an empty list **[]** value for this field disables admin groups.

3 Overrides the default groups (**system:cluster-admins**, **cluster-admin**, **dedicated-admin**)

4.2.5. Loki deployment sizing

Sizing for Loki follows the format of **1x.<size>** where the value **1x** is number of instances and **<size>** specifies performance capabilities.

IMPORTANT

It is not possible to change the number **1x** for the deployment size.

Table 4.2. Loki sizing

	1x.demo	1x.extra-small	1x.small	1x.medium
Data transfer	Demo use only	100GB/day	500GB/day	2TB/day
Queries per second (QPS)	Demo use only	1-25 QPS at 200ms	25-50 QPS at 200ms	25-75 QPS at 200ms
Replication factor	None	2	2	2
Total CPU requests	None	14 vCPUs	34 vCPUs	54 vCPUs
Total memory requests	None	31Gi	67Gi	139Gi
Total disk requests	40Gi	430Gi	430Gi	590Gi

4.2.6. LokiStack ingestion limits and health alerts

The **LokiStack** instance includes default ingestion and query limits that can be overridden by administrators to manage performance and prevent system alerts or errors.

NOTE

You might want to update the ingestion and query limits if you get Loki errors showing up in the Console plugin, or in **flowlogs-pipeline** logs.

Here is an example of configured limits:

```
spec:
  limits:
    global:
      ingestion:
        ingestionBurstSize: 40
        ingestionRate: 20
```

```
    maxGlobalStreamsPerTenant: 25000
  queries:
    maxChunksPerQuery: 2000000
    maxEntriesLimitPerQuery: 10000
    maxQuerySeries: 3000
```

For more information about these settings, see the LokiStack API reference.

4.3. INSTALLING THE NETWORK OBSERVABILITY OPERATOR

Install the Network Observability Operator and use the setup wizard to create the **FlowCollector** custom resource definition (CRD) to complete the initial configuration.

You can set specifications in the web console when you create the **FlowCollector**.

> **IMPORTANT**
>
> The actual memory consumption of the Operator depends on your cluster size and the number of resources deployed. Memory consumption might need to be adjusted accordingly. For more information refer to "Network Observability controller manager pod runs out of memory" in the "Important Flow Collector configuration considerations" section.

Prerequisites

- If you choose to use Loki, install the Loki Operator version 5.7+.
- You must have **cluster-admin** privileges.
- One of the following supported architectures is required: **amd64**, **ppc64le**, **arm64**, or **s390x**.
- Any CPU supported by Red Hat Enterprise Linux (RHEL) 9.
- Must be configured with OVN-Kubernetes as the main network plugin, and optionally using secondary interfaces with Multus and SR-IOV.

> **NOTE**
>
> Additionally, this installation example uses the **netobserv** namespace, which is used across all components. You can optionally use a different namespace.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Choose **Network Observability Operator** from the list of available Operators in the **OperatorHub**, and click **Install**.
3. Select the checkbox **Enable Operator recommended cluster monitoring on this Namespace**.
4. Navigate to **Operators → Installed Operators**. Under Provided APIs for Network Observability, select the **Flow Collector** link.
5. Follow the **Network Observability FlowCollector setup** wizard.

6. Click **Create**.

Verification

To confirm this was successful, when you navigate to **Observe** you should see **Network Traffic** listed in the options.

In the absence of **Application Traffic** within the OpenShift Container Platform cluster, default filters might show that there are "No results", which results in no visual flow. Beside the filter selections, select **Clear all filters** to see the flow.

4.4. ENABLING MULTI-TENANCY IN NETWORK OBSERVABILITY

Enable multi-tenancy in network observability by configuring cluster roles and namespace roles to grant project administrators and developers granular, restricted access to flows and metrics in Loki and Prometheus.

Access is enabled for project administrators. Project administrators who have limited access to some namespaces can access flows for only those namespaces.

For Developers, multi-tenancy is available for both Loki and Prometheus but requires different access rights.

Prerequisite

- If you are using Loki, you have installed at least Loki Operator version 5.7.
- You must be logged in as a project administrator.

Procedure

- For per-tenant access, you must have the **netobserv-loki-reader** cluster role and the **netobserv-metrics-reader** namespace role to use the developer perspective. Run the following commands for this level of access:

  ```
  $ 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>
  ```

- For cluster-wide access, non-cluster-administrators must have the **netobserv-loki-reader**, **cluster-monitoring-view**, and **netobserv-metrics-reader** cluster roles. In this scenario, you can use either the admin perspective or the developer perspective. Run the following commands for this level of access:

  ```
  $ 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. IMPORTANT FLOW COLLECTOR CONFIGURATION CONSIDERATIONS

Once you create the **FlowCollector** instance, you can reconfigure it, but the pods are terminated and recreated again, which can be disruptive. Therefore, you can consider configuring the following options when creating the **FlowCollector** for the first time:

- Configuring the Flow Collector resource with Kafka
- Export enriched network flow data to Kafka or IPFIX
- Configuring monitoring for SR-IOV interface traffic
- Working with conversation tracking
- Working with DNS tracking
- Working with packet drops

Additional resources

- Flow Collector API Reference
- Flow Collector sample resource
- Resource considerations
- Troubleshooting network observability controller manager pod runs out of memory
- Network observability architecture

4.5.1. Migrating removed stored versions of the FlowCollector CRD

Manually remove the deprecated **v1alpha1** version from the **FlowCollector** custom resource definition (CRD) **storedVersion** list to prevent upgrade errors and successfully migrate to Network Observability Operator 1.6.

There are two options to remove stored versions:

1. Use the Storage Version Migrator Operator.
2. Uninstall and reinstall the Network Observability Operator, ensuring that the installation is in a clean state.

Prerequisites

- You have an older version of the Operator installed, and you want to prepare your cluster to install the latest version of the Operator. Or you have attempted to install the Network Observability Operator 1.6 and run into the error: **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**.

Procedure

1. Verify that the old **FlowCollector** CRD version is still referenced in the **storedVersion**:

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

2. If **v1alpha1** appears in the list of results, proceed with **Step a** to use the Kubernetes Storage Version Migrator or **Step b** to uninstall and reinstall the CRD and the Operator.

   a. **Option 1: Kubernetes Storage Version Migrator** Create a YAML to define the **StorageVersionMigration** object, for example **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. Save the file.

      ii. Apply the **StorageVersionMigration** by running the following command:

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

      iii. Update the **FlowCollector** CRD to manually remove **v1alpha1** from the **storedVersion**:

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

   b. **Option 2: Reinstall**: Save the Network Observability Operator 1.5 version of the **FlowCollector** CR to a file, for example **flowcollector-1.5.yaml**.

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

      i. Follow the steps in "Uninstalling the Network Observability Operator", which uninstalls the Operator and removes the existing **FlowCollector** CRD.

      ii. Install the Network Observability Operator latest version, 1.6.0.

      iii. Create the **FlowCollector** using backup that was saved in Step b.

Verification

- Run the following command:

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

  The list of results should no longer show **v1alpha1** and only show the latest version, **v1beta1**.

Additional resources

- Kubernetes Storage Version Migrator Operator

4.6. INSTALLING KAFKA (OPTIONAL)

The Kafka Operator is supported for large-scale environments. Kafka provides high-throughput and low-latency data feeds for forwarding network flow data in a more resilient, scalable way.

You can install the Kafka Operator as Red Hat AMQ Streams from the Operator Hub, just as the Loki Operator and Network Observability Operator were installed. Refer to "Configuring the FlowCollector resource with Kafka" to configure Kafka as a storage option.

> **NOTE**
>
> To uninstall Kafka, refer to the uninstallation process that corresponds with the method you used to install.

Additional resources

- Configuring the FlowCollector resource with Kafka

4.7. UNINSTALLING THE NETWORK OBSERVABILITY OPERATOR

Uninstall the Network Observability Operator using the OpenShift Container Platform web console Operator Hub, working in the **Ecosystem → Installed Operators** area.

Procedure

1. Remove the **FlowCollector** custom resource.
   a. Click **Flow Collector**, which is next to the **Network Observability Operator** in the **Provided APIs** column.
   b. Click the Options menu ⋮ for the **cluster** and select **Delete FlowCollector**.
2. Uninstall the Network Observability Operator.
   a. Navigate back to the **Operators → Installed Operators** area.
   b. Click the Options menu ⋮ next to the **Network Observability Operator** and select **Uninstall Operator**.
   c. **Home → Projects** and select **openshift-netobserv-operator**
   d. Navigate to **Actions** and select **Delete Project**
3. Remove the **FlowCollector** custom resource definition (CRD).
   a. Navigate to **Administration → CustomResourceDefinitions**.
   b. Look for **FlowCollector** and click the Options menu ⋮ .
   c. Select **Delete CustomResourceDefinition**.

IMPORTANT

The Loki Operator and Kafka remain if they were installed and must be removed separately. Additionally, you might have remaining data stored in an object store, and a persistent volume that must be removed.

CHAPTER 5. NETWORK OBSERVABILITY OPERATOR IN OPENSHIFT CONTAINER PLATFORM

The Network Observability Operator for OpenShift Container Platform deploys a monitoring pipeline. This pipeline collects and enriches network traffic flows generated by the **eBPF agent**.

5.1. VIEWING STATUSES

View the operational status of the Network Observability Operator by using the **oc get** command to check the **FlowCollector** resource status, as well as the status of the **eBPF agent**, **flowlogs-pipeline**, and console plugin Pods.

The Network Observability Operator provides the Flow Collector API. When a Flow Collector resource is created, it deploys pods and services to create and store network flows in the Loki log store, as well as to display dashboards, metrics, and flows in the OpenShift Container Platform web console.

Procedure

1. Run the following command to view the state of **FlowCollector**:

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

Example output

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

2. Check the status of pods running in the **netobserv** namespace by entering the following command:

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

Example output

```
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
```

The **flowlogs-pipeline** pods collect flows, enriches the collected flows, then send flows to the Loki storage. **netobserv-plugin** pods create a visualization plugin for the OpenShift Container Platform Console.

3. Check the status of pods running in the namespace **netobserv-privileged** by entering the following command:

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

Example output

```
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
```

The **netobserv-ebpf-agent** pods monitor network interfaces of the nodes to get flows and send them to **flowlogs-pipeline** pods.

4. If you are using the Loki Operator, check the status of the **component** pods of **LokiStack** custom resource in the **netobserv** namespace by entering the following command:

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

Example output

```
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
```

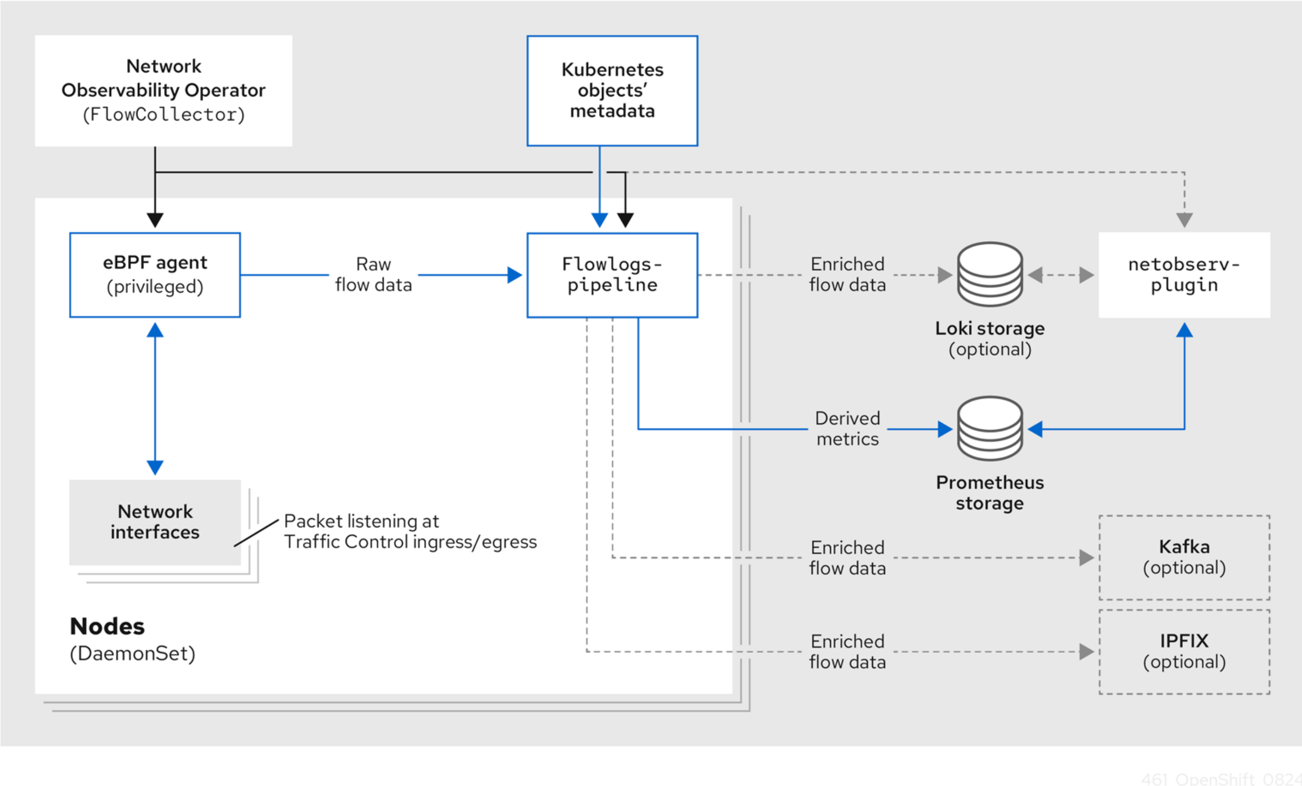
5.2. NETWORK OBSERVABLITY OPERATOR ARCHITECTURE

Review the Network Observability Operator architecture, detailing how the **FlowCollector** resource manages the **eBPF agent**, which collects and enriches flows, sending the data to Loki for storage or Prometheus for metrics.

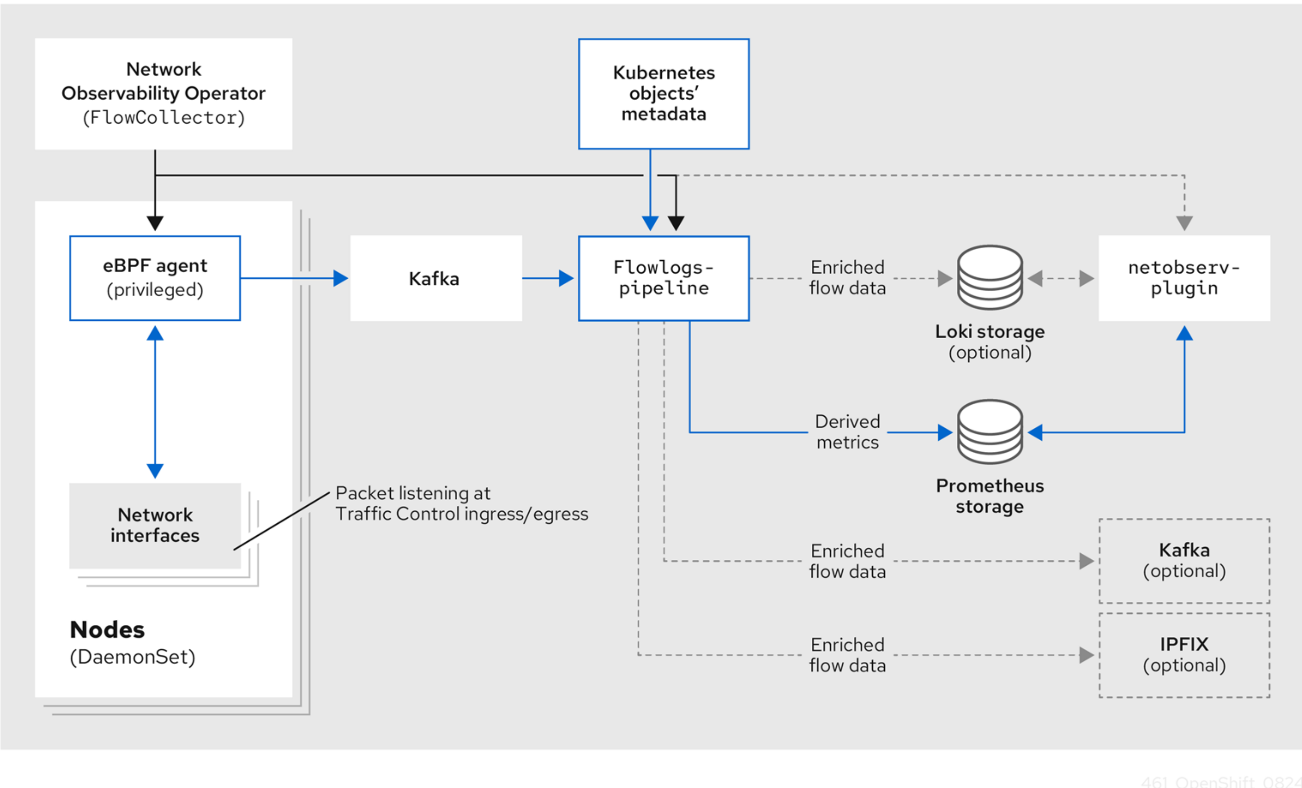
The Network Observability Operator provides the **FlowCollector** API, which is instantiated at installation and configured to reconcile the **eBPF agent**, the **flowlogs-pipeline**, and the **netobserv-plugin** components. Only a single **FlowCollector** per cluster is supported.

The **eBPF agent** runs on each cluster node with some privileges to collect network flows. The **flowlogs-pipeline** receives the network flows data and enriches the data with Kubernetes identifiers. If you choose to use Loki, the **flowlogs-pipeline** sends flow logs data to Loki for storing and indexing. The **netobserv-plugin**, which is a dynamic OpenShift Container Platform web console plugin, queries Loki to fetch network flows data. Cluster-admins can view the data in the web console.

If you do not use Loki, you can generate metrics with Prometheus. Those metrics and their related dashboards are accessible in the web console. For more information, see "Network Observability without Loki".



If you are using the Kafka option, the eBPF agent sends the network flow data to Kafka, and the **flowlogs-pipeline** reads from the Kafka topic before sending to Loki, as shown in the following diagram.



Additional resources

- Network Observability without Loki

5.3. VIEWING NETWORK OBSERVABILITY OPERATOR STATUS AND CONFIGURATION

Inspect the current status, configuration details, and generated resources of the Network Observability Operator by using the **oc describe flowcollector/cluster** command.

Procedure

1. Run the following command to view the status and configuration of the Network Observability Operator:

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

CHAPTER 6. CONFIGURING THE NETWORK OBSERVABILITY OPERATOR

Configure the Network Observability Operator by updating the cluster-wide **FlowCollector** API resource (cluster) to manage component configurations and flow collection settings.

The **FlowCollector** is explicitly created during installation. Since this resource operates cluster-wide, only a single **FlowCollector** is allowed, and it must be named **cluster**. For more information, see the FlowCollector API reference.

6.1. VIEW THE FLOWCOLLECTOR RESOURCE

View and modify the **FlowCollector** resource in the OpenShift Container Platform web console through the integrated setup, advanced form, or by editing the YAML directly to configure the Network Observability Operator.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster** then select the **YAML** tab. There, you can modify the **FlowCollector** resource to configure the Network Observability Operator.

6.1.1. Example of a FlowCollector resource

Review a comprehensive, annotated example of the **FlowCollector** custom resource that demonstrates configurations for **eBPF** sampling, conversation tracking, Loki integration, and console quick filters.

6.1.1.1. Sample FlowCollector resource

```
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: (4)
    mode: LokiStack (5)
  consolePlugin:
    register: true
    logLevel: info
    portNaming:
      enable: true
      portNames:
        "3100": loki
    quickFilters: (6)
    - 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. The Agent specification, **spec.agent.type**, must be **EBPF**. eBPF is the only OpenShift Container Platform supported option.

2. You can set the Sampling specification, **spec.agent.ebpf.sampling**, to manage resources. By default, eBPF sampling is set to **50**, so a flow has a 1 in 50 chance of being sampled. A lower sampling interval value requires more computational, memory, and storage resources. A value of **0** or **1** means all flows are sampled. It is recommended to start with the default value and refine it empirically to determine the optimal setting for your cluster.

3. The Processor specification **spec.processor.** can be set to enable conversation tracking. When enabled, conversation events are queryable in the web console. The **spec.processor.logTypes** value is **Flows**. The **spec.processor.advanced** values are **Conversations**, **EndedConversations**, or **ALL**. Storage requirements are highest for **All** and lowest for **EndedConversations**.

4. The Loki specification, **spec.loki**, specifies the Loki client. The default values match the Loki install paths mentioned in the Installing the Loki Operator section. If you used another installation method for Loki, specify the appropriate client information for your install.

5 The **LokiStack** mode automatically sets a few configurations: **querierUrl**, **ingesterUrl** and **statusUrl**, **tenantID**, and corresponding TLS configuration. Cluster roles and a cluster role binding

6 The **spec.quickFilters** specification defines filters that show up in the web console. The **Application** filter keys,**src_namespace** and **dst_namespace**, are negated (**!**), so the **Application** filter shows all traffic that *does not* originate from, or have a destination to, any **openshift-** or **netobserv** namespaces. For more information, see Configuring quick filters below.

Additional resources

- FlowCollector API reference
- Working with conversation tracking

6.2. CONFIGURING THE FLOWCOLLECTOR RESOURCE WITH KAFKA

Configure the **FlowCollector** resource to use Kafka for high-throughput and low-latency data feeds.

A Kafka instance needs to be running, and a Kafka topic dedicated to OpenShift Container Platform Network Observability must be created in that instance. For more information, see Kafka documentation with AMQ Streams.

Prerequisites

- Kafka is installed. Red Hat supports Kafka with AMQ Streams Operator.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. Under the **Provided APIs** heading for the Network Observability Operator, select **Flow Collector**.
3. Select the cluster and then click the **YAML** tab.
4. Modify the **FlowCollector** resource for OpenShift Container Platform Network Observability Operator to use Kafka, as shown in the following sample YAML:

Sample Kafka configuration in FlowCollector resource

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  deploymentModel: Kafka                  (1)
  kafka:
    address: "kafka-cluster-kafka-bootstrap.netobserv"   (2)
    topic: network-flows                  (3)
    tls:
      enable: false                       (4)
```

1. Set **spec.deploymentModel** to **Kafka** instead of **Direct** to enable the Kafka deployment model.

2. **spec.kafka.address** refers to the Kafka bootstrap server address. You can specify a port if needed, for instance **kafka-cluster-kafka-bootstrap.netobserv:9093** for using TLS on port 9093.

3. **spec.kafka.topic** should match the name of a topic created in Kafka.

4. **spec.kafka.tls** can be used to encrypt all communications to and from Kafka with TLS or mTLS. When enabled, the Kafka CA certificate must be available as a ConfigMap or a Secret, both in the namespace where the **flowlogs-pipeline** processor component is deployed (default: **netobserv**) and where the eBPF agents are deployed (default: **netobserv-privileged**). It must be referenced with **spec.kafka.tls.caCert**. When using mTLS, client secrets must be available in these namespaces as well (they can be generated for instance using the AMQ Streams User Operator) and referenced with **spec.kafka.tls.userCert**.

6.3. EXPORT ENRICHED NETWORK FLOW DATA

Configure the **FlowCollector** resource to export enriched network flow data simultaneously to Kafka, IPFIX, or an OpenTelemetry endpoint for external consumption by tools like Splunk or Prometheus.

For Kafka or IPFIX, any processor or storage that supports those inputs, such as Splunk, Elasticsearch, or Fluentd, can consume the enriched network flow data.

For OpenTelemetry, network flow data and metrics can be exported to a compatible OpenTelemetry endpoint, such as Red Hat build of OpenTelemetry or Prometheus.

Prerequisites

- Your Kafka, IPFIX, or OpenTelemetry collector endpoints are available from Network Observability **flowlogs-pipeline** pods.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster** and then select the **YAML** tab.
4. Edit the **FlowCollector** to configure **spec.exporters** as follows:

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  exporters:
  - type: Kafka          (1)
    kafka:
      address: "kafka-cluster-kafka-bootstrap.netobserv"
      topic: netobserv-flows-export   (2)
```

```
      tls:
        enable: false          (3)
  - type: IPFIX                (4)
    ipfix:
      targetHost: "ipfix-collector.ipfix.svc.cluster.local"
      targetPort: 4739
      transport: tcp or udp    (5)
  -  type: OpenTelemetry       (6)
    openTelemetry:
      targetHost: my-otelcol-collector-headless.otlp.svc
      targetPort: 4317
      type: grpc               (7)
      logs:                    (8)
        enable: true
      metrics:                 (9)
        enable: true
        prefix: netobserv
        pushTimeInterval: 20s  (10)
        expiryTime: 2m
  #    fieldsMapping:          (11)
  #      input: SrcAddr
  #      output: source.address
```

1 4 6 You can export flows to IPFIX, OpenTelemetry, and Kafka individually or concurrently.

2 The Network Observability Operator exports all flows to the configured Kafka topic.

3 You can encrypt all communications to and from Kafka with SSL/TLS or mTLS. When enabled, the Kafka CA certificate must be available as a ConfigMap or a Secret, both in the namespace where the **flowlogs-pipeline** processor component is deployed (default: netobserv). It must be referenced with **spec.exporters.tls.caCert**. When using mTLS, client secrets must be available in these namespaces as well (they can be generated for instance using the AMQ Streams User Operator) and referenced with **spec.exporters.tls.userCert**.

5 You have the option to specify transport. The default value is **tcp** but you can also specify **udp**.

7 The protocol of OpenTelemetry connection. The available options are **http** and **grpc**.

8 OpenTelemetry configuration for exporting logs, which are the same as the logs created for Loki.

9 OpenTelemetry configuration for exporting metrics, which are the same as the metrics created for Prometheus. These configurations are specified in the **spec.processor.metrics.includeList** parameter of the **FlowCollector** custom resource, along with any custom metrics you defined using the **FlowMetrics** custom resource.

10 The time interval that metrics are sent to the OpenTelemetry collector.

11 **Optional**:Network Observability network flows formats get automatically renamed to an OpenTelemetry compliant format. The **fieldsMapping** specification gives you the ability to customize the OpenTelemetry format output. For example in the YAML sample, **SrcAddr** is the Network Observability input field, and it is being renamed **source.address** in OpenTelemetry output. You can see both Network Observability and OpenTelemetry

formats in the "Network flows format reference".

After configuration, network flows data can be sent to an available output in a JSON format. For more information, see "Network flows format reference".

Additional resources

- Network flows format reference

6.4. UPDATING THE FLOWCOLLECTOR RESOURCE

As an alternative to using the web console, use the **oc patch** command with the **flowcollector** custom resource to quickly update specific specifications, such as eBPF sampling

Procedure

1. Run the following command to patch the **flowcollector** CR and update the **spec.agent.ebpf.sampling** value:

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

6.5. FILTER NETWORK FLOWS AT INGESTION

Create filters to reduce the number of generated network flows. Filtering network flows can reduce the resource usage of the network observability components.

You can configure two kinds of filters:

- eBPF agent filters
- Flowlogs-pipeline filters

6.5.1. eBPF agent filters

eBPF agent filters maximize performance because they take effect at the earliest stage of the network flows collection process.

To configure eBPF agent filters with the Network Observability Operator, see "Filtering eBPF flow data using multiple rules".

6.5.2. Flowlogs-pipeline filters

Flowlogs-pipeline filters provide greater control over traffic selection because they take effect later in the network flows collection process. They are primarily used to improve data storage.

Flowlogs-pipeline filters use a simple query language to filter network flow, as shown in the following example:

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

The query language uses the following syntax:

Table 6.1. Query language syntax

Category	Operators
Logical boolean operators (not case-sensitive)	**and**, **or**
Comparison operators	**=** (equals), **!=** (not equals), **=~** (matches regexp), **!~** (not matches regexp), **<** / **<=** (less than or equal to), **>** / **>=** (greater than or equal to)
Unary operations	**with(field)** (field is present), **without(field)** (field is absent)

You can configure flowlogs-pipeline filters in the **spec.processor.filters** section of the **FlowCollector** resource. For example:

Example YAML Flowlogs-pipeline filter

```
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 1
        sampling: 10 2
```

1 Sends matching flows to a specific output, such as Loki, Prometheus, or an external system. When omitted, sends to all configured outputs.

2 Optional. Applies a sampling interval to limit the number of matching flows to be stored or exported. For example, **sampling: 10** means that there is a 1 in 10 chance that a flow will be kept.

Additional resources

- Filtering eBPF flow data using multiple rules

6.6. CONFIGURING QUICK FILTERS

Use the list of available source, destination, and universal filter keys to modify quick filters within the **FlowCollector** resource.

Exact matches are possible using double-quotes around values. Otherwise, partial matches are used for textual values. The bang (!) character, placed at the end of a key, means negation. See the sample **FlowCollector** resource for more context about modifying the YAML.

NOTE

The filter matching types "all of" or "any of" is a UI setting that the users can modify from the query options. It is not part of this resource configuration.

Here is a list of all available filter keys:

Table 6.2. Filter keys

Universal*	Source	Destination	Description
namespace	**src_namespace**	**dst_namespace**	Filter traffic related to a specific namespace.
name	**src_name**	**dst_name**	Filter traffic related to a given leaf resource name, such as a specific pod, service, or node (for host-network traffic).
kind	**src_kind**	**dst_kind**	Filter traffic related to a given resource kind. The resource kinds include the leaf resource (Pod, Service or Node), or the owner resource (Deployment and StatefulSet).
owner_name	**src_owner_name**	**dst_owner_name**	Filter traffic related to a given resource owner; that is, a workload or a set of pods. For example, it can be a Deployment name, a StatefulSet name, etc.
resource	**src_resource**	**dst_resource**	Filter traffic related to a specific resource that is denoted by its canonical name, that identifies it uniquely. The canonical notation is **kind.namespace.name** for namespaced kinds, or**node.name** for nodes. For example, **Deployment.my-namespace.my-web-server**.
address	**src_address**	**dst_address**	Filter traffic related to an IP address. IPv4 and IPv6 are supported. CIDR ranges are also supported.
mac	**src_mac**	**dst_mac**	Filter traffic related to a MAC address.
port	**src_port**	**dst_port**	Filter traffic related to a specific port.

Universal*	Source	Destination	Description
host_address	**src_host_address**	**dst_host_address**	Filter traffic related to the host IP address where the pods are running.
protocol	N/A	N/A	Filter traffic related to a protocol, such as TCP or UDP.

- Universal keys filter for any of source or destination. For example, filtering **name: 'my-pod'** means all traffic from **my-pod** and all traffic to **my-pod**, regardless of the matching type used, whether **Match all** or **Match any**.

6.7. RESOURCE MANAGEMENT AND PERFORMANCE CONSIDERATIONS

Review the key configuration settings, including eBPF sampling, feature enablement, and resource limits, necessary to manage performance criteria and optimize resource consumption for network observability.

The amount of resources required by network observability depends on the size of your cluster and your requirements for the cluster to ingest and store observability data. To manage resources and set performance criteria for your cluster, consider configuring the following settings. Configuring these settings might meet your optimal setup and observability needs.

The following settings can help you manage resources and performance from the outset:

eBPF Sampling

You can set the Sampling specification, **spec.agent.ebpf.sampling**, to manage resources. By default, eBPF sampling is set to **50**, so a flow has a 1 in 50 chance of being sampled. A lower sampling interval value requires more computational, memory, and storage resources. A value of **0** or **1** means all flows are sampled. It is recommended to start with the default value and refine it empirically to determine the optimal setting for your cluster.

eBPF features

The more features that are enabled, the more CPU and memory are impacted. See "Observing the network traffic" for a complete list of these features.

Without Loki

You can reduce the amount of resources that network observability requires by not using Loki and instead relying on Prometheus. For example, when network observability is configured without Loki, the total savings of memory usage are in the 20-65% range and CPU utilization is lower by 10-30%, depending upon the sampling interval value. See "Network observability without Loki" for more information.

Restricting or excluding interfaces

Reduce the overall observed traffic by setting the values for **spec.agent.ebpf.interfaces** and **spec.agent.ebpf.excludeInterfaces**. By default, the agent fetches all the interfaces in the system, except the ones listed in **excludeInterfaces** and **lo** (local interface). Note that the interface names might vary according to the Container Network Interface (CNI) used.

Performance fine-tuning

The following settings can be used to fine-tune performance after the Network Observability has been running for a while:

- **Resource requirements and limits**: Adapt the resource requirements and limits to the load and memory usage you expect on your cluster by using the **spec.agent.ebpf.resources** and **spec.processor.resources** specifications. The default limits of 800MB might be sufficient for most medium-sized clusters.
- **Cache max flows timeout**: Control how often flows are reported by the agents by using the eBPF agent's **spec.agent.ebpf.cacheMaxFlows** and **spec.agent.ebpf.cacheActiveTimeout** specifications. A larger value results in less traffic being generated by the agents, which correlates with a lower CPU load. However, a larger value leads to a slightly higher memory consumption, and might generate more latency in the flow collection.

6.7.1. Resource considerations

Review the resource considerations table, which provides baseline examples for configuration settings, such as eBPF memory limits and LokiStack size, tailored to various cluster workload sizes.

The following table outlines examples of resource considerations for clusters with certain workload sizes.

IMPORTANT

The examples outlined in the table demonstrate scenarios that are tailored to specific workloads. Consider each example only as a baseline from which adjustments can be made to accommodate your workload needs.

Table 6.3. Resource recommendations

	Extra small (10 nodes)	Small (25 nodes)	Large (250 nodes) [2]
Worker Node vCPU and memory	4 vCPUs| 16GiB mem [1]	16 vCPUs| 64GiB mem [1]	16 vCPUs| 64GiB Mem [1]
LokiStack size	**1x.extra-small**	**1x.small**	**1x.medium**
Network Observability controller memory limit	400Mi (default)	400Mi (default)	400Mi (default)
eBPF sampling interval	50 (default)	50 (default)	50 (default)
eBPF memory limit	800Mi (default)	800Mi (default)	1600Mi
cacheMaxSize	50,000	100,000 (default)	100,000 (default)
FLP memory limit	800Mi (default)	800Mi (default)	800Mi (default)

	Extra small (10 nodes)	Small (25 nodes)	Large (250 nodes)[2]
FLP Kafka partitions	–	48	48
Kafka consumer replicas	–	6	18
Kafka brokers	–	3 (default)	3 (default)

1. Tested with AWS M6i instances.

2. In addition to this worker and its controller, 3 infra nodes (size **M6i.12xlarge**) and 1 workload node (size **M6i.8xlarge**) were tested.

6.7.2. Total average memory and CPU usage

Review the table detailing the total average CPU and memory usage for network observability components under two distinct traffic scenarios (**Test 1** and **Test 2**) at different eBPF sampling values.

The following table outlines averages of total resource usage for clusters with a sampling value of **1** and **50** for two different tests: **Test 1** and **Test 2**. The tests differ in the following ways:

- **Test 1** takes into account high ingress traffic volume in addition to the total number of namespace, pods and services in an OpenShift Container Platform cluster, places load on the eBPF agent, and represents use cases with a high number of workloads for a given cluster size. For example, **Test 1** consists of 76 Namespaces, 5153 Pods, and 2305 Services with a network traffic scale of ~350 MB/s.

- **Test 2** takes into account high ingress traffic volume in addition to the total number of namespace, pods and services in an OpenShift Container Platform cluster and represents use cases with a high number of workloads for a given cluster size. For example, **Test 2** consists of 553 Namespaces, 6998 Pods, and 2508 Services with a network traffic scale of ~950 MB/s.

Since different types of cluster use cases are exemplified in the different tests, the numbers in this table do not scale linearly when compared side-by-side. Instead, they are intended to be used as a benchmark for evaluating your personal cluster usage. The examples outlined in the table demonstrate scenarios that are tailored to specific workloads. Consider each example only as a baseline from which adjustments can be made to accommodate your workload needs.

NOTE

Metrics exported to Prometheus can impact the resource usage. Cardinality values for the metrics can help determine how much resources are impacted. For more information, see "Network Flows format" in the Additional resources section.

Table 6.4. Total average resource usage

Sampling value	Resources used	Test 1 (25 nodes)	Test 2 (250 nodes)
Sampling = 50	Total NetObserv CPU Usage	1.35	5.39

Sampling value	Resources used	Test 1 (25 nodes)	Test 2 (250 nodes)
	Total NetObserv RSS (Memory) Usage	16 GB	63 GB
Sampling = 1	Total NetObserv CPU Usage	1.82	11.99
	Total NetObserv RSS (Memory) Usage	22 GB	87 GB

Summary: This table shows average total resource usage of Network Observability, which includes Agents, FLP, Kafka, and Loki with all features enabled. For details about what features are enabled, see the features covered in "Observing the network traffic", which comprises all the features that are enabled for this testing.

Additional resources

- Observing the network traffic from the traffic flows view
- Network observability without Loki
- Network Flows format reference

CHAPTER 7. NETWORK POLICY

As an administrator, you can create a network policy for the **netobserv** namespace. This policy secures inbound and outbound access to the Network Observability Operator.

7.1. CONFIGURING NETWORK POLICY BY USING THE FLOWCOLLECTOR CUSTOM RESOURCE

You can set up ingress and egress network policies to control pod traffic. This enhances security and collects only the network flow data you need. This reduces noise, supports compliance, and improves visibility into network communication.

You can configure the **FlowCollector** custom resource (CR) to deploy an egress and ingress network policy for network observability. By default, the **spec.NetworkPolicy.enable** specification is set to **true**.

If you have installed Loki, Kafka or any exporter in a different namespace that also has a network policy, you must ensure that the network observability components can communicate with them. Consider the following about your setup:

- Connection to Loki (as defined in the **FlowCollector** CR **spec.loki** parameter)
- Connection to Kafka (as defined in the **FlowCollector** CR **spec.kafka** parameter)
- Connection to any exporter (as defined in FlowCollector CR **spec.exporters** parameter)
- If you are using Loki and including it in the policy target, connection to an external object storage (as defined in your **LokiStack** related secret)

Procedure

1. In the web console, go to **Operators → Installed Operators** page.
2. Under the **Provided APIs** heading for **Network Observability**, select **Flow Collector**.
3. Select **cluster** then select the **YAML** tab.
4. Configure the **FlowCollector** CR. A sample configuration is as follows:

Example FlowCollector CR for network policy

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

(1) By default, the **enable** value is **true**.

2. Default values are **["openshift-console", "openshift-monitoring"]**.

Additional resources

- Creating a network policy using the CLI

CHAPTER 8. OBSERVING THE NETWORK TRAFFIC

As an administrator, you can observe the network traffic in the OpenShift Container Platform web console for detailed troubleshooting and analysis. This feature helps you get insights from different graphical representations of traffic flow.

8.1. OBSERVING THE NETWORK TRAFFIC FROM THE OVERVIEW VIEW

The Network Traffic **Overview** view provides aggregated flow metrics and visual insights into application communications. Administrators can use the metrics to monitor data volume, troubleshoot connectivity, and detect unusual traffic patterns across the cluster.

The **Overview** view shows aggregate network traffic in your OpenShift Container Platform cluster, allowing you to see which applications are communicating and the volume of data being transferred. It provides detailed insights by source, destination, and flow type, along with the top traffic flows and average byte rates.

As an administrator, you can troubleshoot connectivity issues, detect unusual traffic patterns, and optimize application performance. It provides a quick overview of network behavior, making it easier to prioritize actions and ensure efficient resource usage.

8.1.1. Working with the Overview view

Navigate to the network traffic **Overview** view in the OpenShift Container Platform console to see graphical representations of flow rate statistics and configure the display scope using available options.

Prerequisite

- Access to the cluster with administrator rights.

Procedure

1. Navigate to **Observe → Network Traffic**.
2. In the **Network Traffic** page, click the **Overview** tab.

You can configure the scope of each flow rate data by clicking the menu icon.

8.1.2. Configuring advanced options for the Overview view

Customize the network traffic **Overview** view by configuring advanced options, such as graph scope, label truncation, and panel management, to refine the display of flow rate statistics and traffic data.

To access the advanced options, click **Show advanced options**. You can configure the details in the graph by using the **Display options** drop-down menu. The options available are as follows:

- **Scope**: Select to view the components that network traffic flows between. You can set the scope to **Node**, **Namespace**, **Owner**, **Zones**, **Cluster** or **Resource**. **Owner** is an aggregation of resources. **Resource** can be a pod, service, node, in case of host-network traffic, or an unknown IP address. The default value is **Namespace**.
- **Truncate labels**: Select the required width of the label from the drop-down list. The default value is **M**.

8.1.2.1. Managing panels and display

You can select the required panels to be displayed, reorder them, and focus on a specific panel. To add or remove panels, click **Manage panels**.

The following panels are shown by default:

- **Top X average bytes rates**
- **Top X bytes rates stacked with total**

Other panels can be added in **Manage panels**:

- **Top X average packets rates**
- **Top X packets rates stacked with total**

Query options allows you to choose whether to show the **Top 5**, **Top 10**, or **Top 15** rates.

8.1.3. Packet drop tracking

Monitor and analyze network packet loss by using eBPF-based packet drop tracking, which identifies drop locations, detects host or OVS-specific drop reasons, and provides dedicated graphical panels in the **Overview** view.

You can configure graphical representation of network flow records with packet loss in the **Overview** view. By employing eBPF tracepoint hooks, you can gain valuable insights into packet drops for TCP, UDP, SCTP, ICMPv4, and ICMPv6 protocols, which can result in the following actions:

- Identification: Pinpoint the exact locations and network paths where packet drops are occurring. Determine whether specific devices, interfaces, or routes are more prone to drops.
- Root cause analysis: Examine the data collected by the eBPF program to understand the causes of packet drops. For example, are they a result of congestion, buffer issues, or specific network events?
- Performance optimization: With a clearer picture of packet drops, you can take steps to optimize network performance, such as adjust buffer sizes, reconfigure routing paths, or implement Quality of Service (QoS) measures.

When packet drop tracking is enabled, you can see the following panels in the **Overview** by default:

- **Top X packet dropped state stacked with total**
- **Top X packet dropped cause stacked with total**
- **Top X average dropped packets rates**
- **Top X dropped packets rates stacked with total**

Other packet drop panels are available to add in **Manage panels**:

- **Top X average dropped bytes rates**
- **Top X dropped bytes rates stacked with total**

8.1.3.1. Types of packet drops

Two kinds of packet drops are detected by network observability: host drops and OVS drops. Host drops are prefixed with **SKB_DROP** and OVS drops are prefixed with **OVS_DROP**. Dropped flows are shown in the side panel of the **Traffic flows** table along with a link to a description of each drop type. Examples of host drop reasons are as follows:

- **SKB_DROP_REASON_NO_SOCKET**: the packet dropped due to a missing socket.
- **SKB_DROP_REASON_TCP_CSUM**: the packet dropped due to a TCP checksum error.

Examples of OVS drops reasons are as follows:

- **OVS_DROP_LAST_ACTION**: OVS packets dropped due to an implicit drop action, for example due to a configured network policy.
- **OVS_DROP_IP_TTL**: OVS packets dropped due to an expired IP TTL.

See the *Additional resources* of this section for more information about enabling and working with packet drop tracking.

Additional resources

- Working with packet drops
- Network Observability metrics

8.1.4. DNS tracking

Monitor DNS activity by using eBPF-based DNS tracking to gain insights into query patterns, detect security threats, and troubleshoot latency issues through dedicated graphical panels in the **Overview** view.

You can configure graphical representation of Domain Name System (DNS) tracking of network flows in the **Overview** view. Using DNS tracking with extended Berkeley Packet Filter (eBPF) tracepoint hooks can serve various purposes:

- Network Monitoring: Gain insights into DNS queries and responses, helping network administrators identify unusual patterns, potential bottlenecks, or performance issues.
- Security Analysis: Detect suspicious DNS activities, such as domain name generation algorithms (DGA) used by malware, or identify unauthorized DNS resolutions that might indicate a security breach.
- Troubleshooting: Debug DNS-related issues by tracing DNS resolution steps, tracking latency, and identifying misconfigurations.

By default, when DNS tracking is enabled, you can see the following non-empty metrics represented in a donut or line chart in the **Overview**:

- Top X DNS Response Code
- Top X average DNS latencies with overall
- Top X 90th percentile DNS latencies

Other DNS tracking panels can be added in **Manage panels**:

- Bottom X minimum DNS latencies
- Top X maximum DNS latencies
- Top X 99th percentile DNS latencies

This feature is supported for IPv4 and IPv6 UDP and TCP protocols.

See the *Additional resources* in this section for more information about enabling and working with this view.

Additional resources

- Working with DNS tracking
- Network Observability metrics

8.1.5. Round-Trip Time

Analyze network flow latencies by using TCP Round-Trip Time (RTT) metrics, which use eBPF hookpoints to identify performance bottlenecks and troubleshoot TCP-related issues through dedicated panels in the Overview view.

You can use TCP smoothed Round-Trip Time (sRTT) to analyze network flow latencies. You can use RTT captured from the **fentry/tcp_rcv_established** eBPF hookpoint to read sRTT from the TCP socket to help with the following:

- Network Monitoring: Gain insights into TCP latencies, helping network administrators identify unusual patterns, potential bottlenecks, or performance issues.
- Troubleshooting: Debug TCP-related issues by tracking latency and identifying misconfigurations.

By default, when RTT is enabled, you can see the following TCP RTT metrics represented in the **Overview**:

- Top X 90th percentile TCP Round Trip Time with overall
- Top X average TCP Round Trip Time with overall
- Bottom X minimum TCP Round Trip Time with overall

Other RTT panels can be added in **Manage panels**:

- Top X maximum TCP Round Trip Time with overall
- Top X 99th percentile TCP Round Trip Time with overall

See the *Additional resources* in this section for more information about enabling and working with this view.

Additional resources

- Working with RTT tracing

8.1.6. eBPF flow rule filter

Control packet capture volume by using eBPF flow rule filtering to specify capture criteria based on ports and CIDR notation, while monitoring filter performance through dedicated health dashboards and Prometheus metrics.

You can use rule-based filtering to control the volume of packets cached in the eBPF flow table. For example, a filter can specify that only packets coming from port 100 should be captured. Then only the packets that match the filter are captured and the rest are dropped.

You can apply multiple filter rules.

8.1.6.1. Ingress and egress traffic filtering

Classless Inter-Domain Routing (CIDR) notation efficiently represents IP address ranges by combining the base IP address with a prefix length. For both ingress and egress traffic, the source IP address is first used to match filter rules configured with CIDR notation. If there is a match, then the filtering proceeds. If there is no match, then the destination IP is used to match filter rules configured with CIDR notation.

After matching either the source IP or the destination IP CIDR, you can pinpoint specific endpoints using the **peerIP** to differentiate the destination IP address of the packet. Based on the provisioned action, the flow data is either cached in the eBPF flow table or not cached.

8.1.6.2. Dashboard and metrics integrations

When this option is enabled, the **Netobserv/Health** dashboard for **eBPF agent statistics** now has the **Filtered flows rate** view. Additionally, in **Observe** → **Metrics** you can query **netobserv_agent_filtered_flows_total** to observe metrics with the reason in **FlowFilterAcceptCounter**, **FlowFilterNoMatchCounter** or **FlowFilterRecjectCounter**.

8.1.6.3. Flow filter configuration parameters

Reference the required and optional parameters for configuring flow filter rules in the **FlowCollector** resource, including CIDR ranges, filter actions, protocols, and specific port configurations.

Table 8.1. Required configuration parameters

Parameter	Description
enable	Set **enable** to **true** to enable the eBPF flow filtering feature.
cidr	Provides the IP address and CIDR mask for the flow filter rule. Supports both IPv4 and IPv6 address format. If you want to match against any IP, you can use **0.0.0.0/0** for IPv4 or **::/0** for IPv6.

Parameter	Description
action	Describes the action that is taken for the flow filter rule. The possible values are **Accept** or **Reject**. • For the **Accept** action matching rule, the flow data is cached in the eBPF table and updated with the global metric, **FlowFilterAcceptCounter**. • For the **Reject** action matching rule, the flow data is dropped and not cached in the eBPF table. The flow data is updated with the global metric, **FlowFilterRejectCounter**. • If the rule is not matched, the flow is cached in the eBPF table and updated with the global metric, **FlowFilterNoMatchCounter**.

Table 8.2. Optional configuration parameters

Parameter	Description
direction	Defines the direction of the flow filter rule. Possible values are **Ingress** or **Egress**.
protocol	Defines the protocol of the flow filter rule. Possible values are **TCP**, **UDP**, **SCTP**, **ICMP**, and **ICMPv6**.
tcpFlags	Defines the TCP flags to filter flows. Possible values are **SYN**, **SYN-ACK**, **ACK**, **FIN**, **RST**, **PSH**, **URG**, **ECE**, **CWR**, **FIN-ACK**, and **RST-ACK**.
ports	Defines the ports to use for filtering flows. It can be used for either source or destination ports. To filter a single port, set a single port as an integer value. For example **ports: 80**. To filter a range of ports, use a "start-end" range in string format. For example **ports: "80-100"**
sourcePorts	Defines the source port to use for filtering flows. To filter a single port, set a single port as an integer value, for example **sourcePorts: 80**. To filter a range of ports, use a "start-end" range, string format, for example **sourcePorts: "80-100"**.
destPorts	DestPorts defines the destination ports to use for filtering flows. To filter a single port, set a single port as an integer value, for example **destPorts: 80**. To filter a range of ports, use a "start-end" range in string format, for example **destPorts: "80-100"**.
icmpType	Defines the ICMP type to use for filtering flows.
icmpCode	Defines the ICMP code to use for filtering flows.
peerIP	Defines the IP address to use for filtering flows, for example: **10.10.10.10**.

Additional resources

- Filtering eBPF flow data with rules
- Network Observability metrics
- Health dashboards

8.1.7. User-defined networks

Understand how you can use user-defined networks (UDN) for flexible network segmentation and leverage the Network Observability Operator to monitor these segments through dedicated labels and name filters in the traffic flow table.

User-defined networks (UDN) improve the flexibility and segmentation capabilities of the default Layer 3 topology for a Kubernetes pod network by enabling custom Layer 2 and Layer 3 network segments, where all these segments are isolated by default. These segments act as primary or secondary networks for container pods and virtual machines that use the default OVN-Kubernetes CNI plugin.

UDNs enable a wide range of network architectures and topologies, enhancing network flexibility, security, and performance.

When the **UDNMapping** feature is enabled with Network Observability, the **Traffic** flow table has a **UDN labels** column. You can filter on **Source Network Name** and **Destination Network Name**.

Additional resources

- About user-defined networks
- Creating a UserDefinedNetwork by using the CLI
- Creating a UserDefinedNetwork by using the web console
- Working with user-defined networks

8.1.8. OVN-Kubernetes networking events

Use OVN-Kubernetes network event tracking to monitor and audit network policies, admin network policies, and egress firewall rules in your cluster.

> **IMPORTANT**
>
> OVN-Kubernetes networking events tracking is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can use the insights from tracking network events to help with the following tasks:

- Network monitoring: Monitor allowed and blocked traffic, detecting whether packets are allowed or blocked based on network policies and admin network policies.
- Network security: You can track outbound traffic and see whether it adheres to egress firewall rules. Detect unauthorized outbound connections and flag outbound traffic that violates egress rules.

See the *Additional resources* in this section for more information about enabling and working with this view.

Additional resources

- Viewing network events

8.2. OBSERVING THE NETWORK TRAFFIC FROM THE TRAFFIC FLOWS VIEW

Use the **Traffic flows** view to monitor real-time and historical network communication between cluster components. By analyzing granular flow data collected via eBPF, you can audit network traffic, validate network policies, and export data for external reporting and analysis.

The **Traffic flows** view in the Network Observability Operator provides a granular, tabular representation of network activity across a OpenShift Container Platform cluster. By leveraging eBPF technology to collect flow data, this view allows administrators to monitor real-time and historical communication between pods, services, and nodes. This visibility is essential for auditing network traffic, validating network policies, and identifying unexpected communication patterns within the cluster infrastructure.

In the **Traffic flows** interface, you can analyze specific connection details by interacting with individual rows to retrieve detailed flow information. The view supports advanced customization through the **Display options** menu, where you can adjust row density and manage columns. By selecting and reordering specific columns, you can tailor the table to highlight the most relevant data points for your environment, such as source and destination endpoints or traffic volume.

To support external analysis and reporting, the **Traffic flows** view includes data export capabilities. You can export the entire dataset or select specific fields to generate a targeted report of network activity. This functionality ensures that network flow data is accessible for long-term auditing or for use in third-party monitoring tools, providing a flexible way to document and analyze the network health of your OpenShift Container Platform environment.

8.2.1. Working with the Traffic flows view

View and analyze detailed network flow information by using the **Traffic flows** table.

As an administrator, you can navigate to **Traffic flows** table to see network flow information.

Prerequisite

- You have administrator access.

Procedure

1. Navigate to **Observe** → **Network Traffic**.
2. In the **Network Traffic** page, click the **Traffic flows** tab.

You can click on each row to get the corresponding flow information.

8.2.2. Configuring advanced options for the Traffic flows view

Customize the **Traffic flows** view by adjusting row density, selecting specific data columns, and exporting filtered flow data for external analysis.

You can customize and export the view by using **Show advanced options**. You can set the row size by using the **Display options** drop-down menu. The default value is **Normal**.

8.2.2.1. Managing columns

You can select the required columns to be displayed, and reorder them. To manage columns, click **Manage columns**.

8.2.2.2. Exporting the traffic flow data

You can export data from the **Traffic flows** view.

Procedure

1. Click **Export data**.
2. In the pop-up window, you can select the **Export all data** checkbox to export all the data, and clear the checkbox to select the required fields to be exported.
3. Click **Export**.

8.2.3. Configuring IPsec with the FlowCollector custom resource

Enable IPsec tracking in the **FlowCollector** resource to monitor encrypted traffic, adding an IPsec status column to the traffic flow view and generating a dedicated encryption dashboard.

In OpenShift Container Platform, IPsec is disabled by default. You can enable IPsec by following the instructions in "Configuring IPsec encryption".

Prerequisite

- You have enabled IPsec encryption on OpenShift Container Platform.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster** then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource for IPsec:

Example configuration of FlowCollector for IPsec

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

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

Verification

When IPsec is enabled:

- A new column named **IPsec Status** is displayed in the network observability **Traffic flows** view to show whether a flow was successfully IPsec-encrypted or if there was an error during encryption/decryption.
- A new dashboard showing the percent of encrypted traffic is generated.

Additional resources

- Configuring IPsec encryption

8.2.4. Working with conversation tracking

Configure the **FlowCollector** custom resource to enable conversation tracking for grouping and analyzing related network flows in the web console.

As an administrator, you can group network flows that are part of the same conversation. A conversation is defined as a grouping of peers that are identified by their IP addresses, ports, and protocols, resulting in an unique **Conversation Id**. You can query conversation events in the web console. These events are represented in the web console as follows:

- **Conversation start**: This event happens when a connection is starting or TCP flag intercepted
- **Conversation tick**: This event happens at each specified interval defined in the **FlowCollector** **spec.processor.conversationHeartbeatInterval** parameter while the connection is active.
- **Conversation end**: This event happens when the **FlowCollector** **spec.processor.conversationEndTimeout** parameter is reached or the TCP flag is intercepted.
- **Flow**: This is the network traffic flow that occurs within the specified interval.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster** then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource so that **spec.processor.logTypes**, **conversationEndTimeout**, and **conversationHeartbeatInterval** parameters are set according to your observation needs. A sample configuration is as follows:

Configure **FlowCollector** for conversation tracking

```yaml
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
 processor:
  logTypes: Flows  # (1)
  advanced:
   conversationEndTimeout: 10s  # (2)
   conversationHeartbeatInterval: 30s  # (3)
```

1. When **logTypes** is set to **Flows**, only the **Flow** event is exported. If you set the value to **All**, both conversation and flow events are exported and visible in the **Network Traffic** page. To focus only on conversation events, you can specify **Conversations** which exports the **Conversation start**, **Conversation tick** and **Conversation end** events; or **EndedConversations** exports only the **Conversation end** events. Storage requirements are highest for **All** and lowest for **EndedConversations**.

2. The **Conversation end** event represents the point when the **conversationEndTimeout** is reached or the TCP flag is intercepted.

3. The **Conversation tick** event represents each specified interval defined in the **FlowCollector conversationHeartbeatInterval** parameter while the network connection is active.

> **NOTE**
>
> If you update the **logType** option, the flows from the previous selection do not clear from the console plugin. For example, if you initially set **logType** to **Conversations** for a span of time until 10 AM and then move to **EndedConversations**, the console plugin shows all conversation events before 10 AM and only ended conversations after 10 AM.

5. Refresh the **Network Traffic** page on the **Traffic flows** tab. Notice there are two new columns, **Event/Type** and **Conversation Id**. All the **Event/Type** fields are **Flow** when **Flow** is the selected query option.

6. Select **Query Options** and choose the **Log Type**, **Conversation**. Now the **Event/Type** shows all of the desired conversation events.

7. Next you can filter on a specific conversation ID or switch between the **Conversation** and **Flow** log type options from the side panel.

8.2.5. Working with packet drops

Enable packet drop tracking in the Network Observability Operator by configuring the **FlowCollector** resource to monitor and visualize network data loss in the web console.

Packet loss occurs when one or more packets of network flow data fail to reach their destination. You can track these drops by editing the **FlowCollector** to the specifications in the following YAML example.

IMPORTANT

CPU and memory usage increases when this feature is enabled.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster**, and then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource for packet drops, for example:

Example FlowCollector configuration

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  agent:
    type: eBPF
    ebpf:
      features:
       - PacketDrop        (1)
      privileged: true     (2)
```

(1) You can start reporting the packet drops of each network flow by listing the **PacketDrop** parameter in the **spec.agent.ebpf.features** specification list.

(2) The **spec.agent.ebpf.privileged** specification value must be **true** for packet drop tracking.

Verification

- When you refresh the **Network Traffic** page, the **Overview**, **Traffic Flow**, and **Topology** views display new information about packet drops:
  a. Select new choices in **Manage panels** to choose which graphical visualizations of packet drops to display in the **Overview**.
  b. Select new choices in **Manage columns** to choose which packet drop information to display in the **Traffic flows** table.
     i. In the **Traffic Flows** view, you can also expand the side panel to view more information about packet drops. Host drops are prefixed with **SKB_DROP** and OVS drops are prefixed with **OVS_DROP**.
  c. In the **Topology** view, red lines are displayed where drops are present.

8.2.6. Working with DNS tracking

Configure the **FlowCollector** custom resource to enable DNS tracking for monitoring network performance, security analysis, and DNS troubleshooting in the web console.

You can track DNS by editing the **FlowCollector** to the specifications in the following YAML example.

IMPORTANT

CPU and memory usage increases are observed in the eBPF agent when this feature is enabled.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. Under the **Provided APIs** heading for **Network Observability**, select **Flow Collector**.
3. Select **cluster** then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource. A sample configuration is as follows:

Configure FlowCollector for DNS tracking

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  agent:
    type: eBPF
    ebpf:
      features:
       - DNSTracking        (1)
      sampling: 1         (2)
```

(1) You can set the **spec.agent.ebpf.features** parameter list to enable DNS tracking of each network flow in the web console.

(2) You can set **sampling** to a value of **1** for more accurate metrics and to capture **DNS latency**. For a **sampling** value greater than 1, you can observe flows with **DNS Response Code** and **DNS Id**, and it is unlikely that **DNS Latency** can be observed.

5. When you refresh the **Network Traffic** page, there are new DNS representations you can choose to view in the **Overview** and **Traffic Flow** views and new filters you can apply.
   a. Select new DNS choices in **Manage panels** to display graphical visualizations and DNS metrics in the **Overview**.
   b. Select new choices in **Manage columns** to add DNS columns to the **Traffic Flows** view.
   c. Filter on specific DNS metrics, such as **DNS Id**, **DNS Error** **DNS Latency** and **DNS Response Code**, and see more information from the side panel. The **DNS Latency** and **DNS Response Code** columns are shown by default.

NOTE

TCP handshake packets do not have DNS headers. TCP protocol flows without DNS headers are shown in the traffic flow data with **DNS Latency**, **ID**, and **Response code** values of "n/a". You can filter out flow data to view only flows that have DNS headers using the **Common** filter "DNSError" equal to "0".

8.2.7. Working with RTT tracing

Enable Round Trip Time (RTT) tracing by configuring the **FlowCollector** custom resource to monitor and analyze network latency across your cluster by using the web console.

You can track RTT by editing the **FlowCollector** to the specifications in the following YAML example.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. In the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster**, and then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource for RTT tracing, for example:

Example FlowCollector configuration

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

(1) You can start tracing RTT network flows by listing the **FlowRTT** parameter in the **spec.agent.ebpf.features** specification list.

Verification

When you refresh the **Network Traffic** page, the **Overview**, **Traffic Flow**, and **Topology** views display new information about RTT:

a. In the **Overview**, select new choices in **Manage panels** to choose which graphical visualizations of RTT to display.

b. In the **Traffic flows** table, the **Flow RTT** column can be seen, and you can manage display in **Manage columns**.

c. In the **Traffic Flows** view, you can also expand the side panel to view more information about RTT.

Example filtering

i. Click the **Common** filters → **Protocol**.

ii. Filter the network flow data based on **TCP**, **Ingress** direction, and look for **FlowRTT** values greater than 10,000,000 nanoseconds (10ms).

iii. Remove the **Protocol** filter.

iv. Filter for **Flow RTT** values greater than 0 in the **Common** filters.

d. In the **Topology** view, click the Display option dropdown. Then click **RTT** in the **edge labels** drop-down list.

8.2.8. Working with the eBPF Manager Operator

Integrate the eBPF Manager Operator with Network Observability to manage eBPF programs and reduce the need for privileged agent permissions.

The eBPF Manager Operator reduces the attack surface and ensures compliance, security, and conflict prevention by managing all eBPF programs. Network observability can use the eBPF Manager Operator to load hooks. As a result, you no longer need to provide the eBPF Agent with privileged mode or additional Linux capabilities such as **CAP_BPF** and **CAP_PERFMON**. The eBPF Manager Operator with network observability is only supported on 64-bit AMD architecture.

> **IMPORTANT**
>
> eBPF Manager Operator with network observability is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope.

Procedure

1. In the web console, navigate to **Operators** → **Operator Hub**.

2. Install **eBPF Manager**.

3. Check **Workloads** → **Pods** in the **bpfman** namespace to make sure they are all up and running.

4. Configure the **FlowCollector** custom resource to use the eBPF Manager Operator:

Example FlowCollector configuration

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

```
features:
  - EbpfManager
```

Verification

1. In the web console, navigate to **Operators → Installed Operators**.
2. Click **eBPF Manager Operator → All instances** tab.
   For each node, verify that a **BpfApplication** named **netobserv** and a pair of **BpfProgram** objects, one for Traffic Control (TCx) ingress and another for TCx egress, exist. If you enable other eBPF Agent features, you might have more objects.

Additional resources

- Installing the eBPF Manager Operator

8.2.9. Using the histogram

The histogram provides a visualization of network flow logs that you can use to analyze traffic volume trends and filter flow data by specific time intervals.

You can click **Show histogram** to display a toolbar view for visualizing the history of flows as a bar chart. The histogram shows the number of logs over time. You can select a part of the histogram to filter the network flow data in the table that follows the toolbar.

8.2.10. Working with availability zones

Configure the **FlowCollector** custom resource to collect availability zone data, enabling the visualization and analysis of network traffic across different cluster zones in the web console.

You can configure the **FlowCollector** to collect information about the cluster availability zones. This allows you to enrich network flow data with the **topology.kubernetes.io/zone** label value applied to the nodes.

Procedure

1. In the web console, go to **Operators → Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster** then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource so that the **spec.processor.addZone** parameter is set to **true**. A sample configuration is as follows:

   Configure FlowCollector for availability zones collection

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

```
processor:
  addZone: true
# ...
```

Verification

When you refresh the **Network Traffic** page, the **Overview**, **Traffic Flow**, and **Topology** views display new information about availability zones:

1. In the **Overview** tab, you can see **Zones** as an available **Scope**.
2. In **Network Traffic → Traffic flows**, **Zones** are viewable under the SrcK8S_Zone and DstK8S_Zone fields.
3. In the **Topology** view, you can set **Zones** as **Scope** or **Group**.

8.2.11. Filtering eBPF flow data using multiple rules

Configure multiple filtering rules in the **FlowCollector** custom resource to refine network traffic data collection by accepting or rejecting specific eBPF flows based on IP addresses and packet conditions.

IMPORTANT

- You cannot use duplicate Classless Inter-Domain Routing (CIDRs) in filter rules.
- When an IP address matches multiple filter rules, the rule with the most specific CIDR prefix (longest prefix) takes precedence.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**.
2. Under the **Provided APIs** heading for **Network Observability**, select **Flow Collector**.
3. Select **cluster**, then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource, similar to the following sample configurations:

Example YAML to sample all North-South traffic, and 1:50 East-West traffic

By default, all other flows are rejected.

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF
    ebpf:
      flowFilter:
        enable: true (1)
        rules:
         - action: Accept (2)
```

```
        cidr: 0.0.0.0/0 (3)
        sampling: 1 (4)
      - 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. To enable eBPF flow filtering, set **spec.agent.ebpf.flowFilter.enable** to **true**.
2. To define the action for the flow filter rule, set the required **action** parameter. Valid values are **Accept** or **Reject**.
3. To define the IP address and CIDR mask for the flow filter rule, set the required **cidr** parameter. This parameter supports both IPv4 and IPv6 address formats. To match any IP address, use **0.0.0.0/0** for IPv4 or **`::/0** for IPv6.
4. To define the sampling interval for matched flows and override the global sampling setting **spec.agent.ebpf.sampling**, set the **sampling** parameter.
5. To filter flows by Peer IP CIDR, set the **peerCIDR** parameter.

Example YAML to filter flows with packet drops

By default, all other flows are rejected.

```
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. To enable packet drops, set **spec.agent.ebpf.privileged** to **true**.
2. To report packet drops for each network flow, add the **PacketDrop** value to the **spec.agent.ebpf.features** list.

3 To enable eBPF flow filtering, set **spec.agent.ebpf.flowFilter.enable** to **true**.

4 To define the action for the flow filter rule, set the required **action** parameter. Valid values are **Accept** or **Reject**.

5 To filter flows containing drops, set **pktDrops** to **true**.

8.2.12. Endpoint translation (xlat)

Endpoint translation (xlat) uses eBPF to enrich network flow logs with translated pod-level metadata, providing visibility into the specific backend pods serving traffic behind services or load balancers.

You can gain visibility into the endpoints serving traffic in a consolidated view using network observability and extended Berkeley Packet Filter (eBPF). Typically, when traffic flows through a service, egressIP, or load balancer, the traffic flow information is abstracted as it is routed to one of the available pods. If you try to get information about the traffic, you can only view service related info, such as service IP and port, and not information about the specific pod that is serving the request. Often the information for both the service traffic and the virtual service endpoint is captured as two separate flows, which complicates troubleshooting.

To solve this, endpoint xlat can help in the following ways:

- Capture the network flows at the kernel level, which has a minimal impact on performance.
- Enrich the network flows with translated endpoint information, showing not only the service but also the specific backend pod, so you can see which pod served a request.

As network packets are processed, the eBPF hook enriches flow logs with metadata about the translated endpoint that includes the following pieces of information that you can view in the **Network Traffic** page in a single row:

- Source Pod IP
- Source Port
- Destination Pod IP
- Destination Port
- Conntrack Zone ID

8.2.13. Working with endpoint translation (xlat)

Enable endpoint translation (xlat) in the **FlowCollector** resource to enrich network flows with translated packet information. You can use this information to identify the specific pods and objects serving service traffic through dedicated xlat columns.

You can use network observability and eBPF to enrich network flows from a Kubernetes service with translated endpoint information, gaining insight into the endpoints serving traffic.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**.
2. In the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.

3. Select **cluster**, and then select the **YAML** tab.

4. Configure the **FlowCollector** custom resource for **PacketTranslation**, for example:

Example FlowCollector configuration

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

(1) You can start enriching network flows with translated packet information by listing the **PacketTranslation** parameter in the **spec.agent.ebpf.features** specification list.

Example filtering

When you refresh the **Network Traffic** page you can filter for information about translated packets:

1. Filter the network flow data based on **Destination kind: Service**.

2. You can see the **xlat** column, which distinguishes where translated information is displayed, and the following default columns:

   - **Xlat Zone ID**
   - **Xlat Src Kubernetes Object**
   - **Xlat Dst Kubernetes Object**

You can manage the display of additional **xlat** columns in **Manage columns**.

8.2.14. Working with user-defined networks

Configure the **FlowCollector** custom resource to enable user-defined network (UDN) mapping, providing visibility into traffic across custom network interfaces within the web console.

You can enable user-defined networks (UDN) in network observability resources. The following example shows the configuration for the **FlowCollector** resource.

Prerequisite

- You have configured UDN in Red Hat OpenShift Networking. For more information, see "Creating a UserDefinedNetwork by using the CLI" or "Creating a UserDefinedNetwork by using the web console."

Procedure

1. Edit the network observability **FlowCollector** resource by running the following command:

```
$ oc edit flowcollector
```

2. Configure the **ebpf** section of the **FlowCollector** resource:

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
    ebpf:
      sampling: 1 (1)
      privileged: true
      features:
      - UDNMapping
```

(1) Recommended so all flows are observed.

Verification

- Refresh the **Network Traffic** page to view updated UDN information in the **Traffic Flow** and **Topology** views:
  - In **Network Traffic** > **Traffic flows**, you can view UDNs under the **SrcK8S_NetworkName** and **DstK8S_NetworkName** fields.
  - In the **Topology** view, you can set **Network** as **Scope** or **Group**.

Additional resources

- Creating a UserDefinedNetwork by using the CLI
- Creating a UserDefinedNetwork by using the web console

8.2.15. Viewing network events

Configure the **FlowCollector** custom resource to enable network event tracking for auditing how security policies, firewalls, and isolation rules affect traffic flows in the web console.

> **IMPORTANT**
>
> OVN-Kubernetes networking events tracking is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.
>
> For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

You can edit the **FlowCollector** to view information about network traffic events, such as network flows that are dropped or allowed by the following resources:

- **NetworkPolicy**
- **AdminNetworkPolicy**
- **BaselineNetworkPolicy**
- **EgressFirewall**
- **UserDefinedNetwork** isolation
- Multicast ACLs

Prerequisites

- You must have **OVNObservability** enabled by setting the **TechPreviewNoUpgrade** feature set in the **FeatureGate** custom resource (CR) named **cluster**. For more information, see "Enabling feature sets using the CLI" and "Checking OVN-Kubernetes network traffic with OVS sampling using the CLI".
- You have created at least one of the following network APIs: **NetworkPolicy**, **AdminNetworkPolicy**, **BaselineNetworkPolicy**, **UserDefinedNetwork** isolation, multicast, or **EgressFirewall**.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**.
2. In the **Provided APIs** heading for the **NetworkObserv Operator**, select **Flow Collector**.
3. Select **cluster**, and then select the **YAML** tab.
4. Configure the **FlowCollector** CR to enable viewing **NetworkEvents**, for example:

Example FlowCollector configuration

```
apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
   type: eBPF
   ebpf:
  #  sampling: 1 (1)
     privileged: true (2)
     features:
      - "NetworkEvents"
```

(1) Optional: The **sampling** parameter is set to a value of 1 so that all network events are captured. If sampling **1** is too resource heavy, set sampling to something more appropriate for your needs.

2 The **privileged** parameter is set to **true** because the **OVN observability** library needs to access local Open vSwitch (OVS) socket and OpenShift Virtual Network (OVN) databases.

Verification

1. Navigate to the **Network Traffic** view and select the **Traffic flows** table.
2. You should see the new column, **Network Events**, where you can view information about impacts of one of the following network APIs you have enabled: **NetworkPolicy**, **AdminNetworkPolicy**, **BaselineNetworkPolicy**, **UserDefinedNetwork** isolation, multicast, or egress firewalls.

An example of the kind of events you could see in this column is as follows:

Example of Network Events output

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

Additional resources

- Enabling feature sets using the CLI
- Checking OVN-Kubernetes network traffic with OVS sampling using the CLI

8.3. OBSERVING THE NETWORK TRAFFIC FROM THE TOPOLOGY VIEW

The **Topology** view in the **Network Traffic** page provides a graphical representation of network flows and traffic volume across your OpenShift Container Platform cluster. As an administrator, you can use this view to monitor application traffic data and visualize the relationships between various network components.

The visualization represents network entities as nodes and traffic flows as edges. By selecting individual components within the graph, you can access a side panel containing specific metrics and health details for that resource. This interactive approach allows for rapid identification of traffic patterns and connectivity issues within the cluster.

To manage complex environments, the **Topology** view includes advanced configuration options that allow you to customize the layout and data density. You can adjust the **Scope** of the view, apply **Groups** to represent resource ownership, and choose different **Layout** algorithms to optimize the graphical display. Additionally, you can enable **Edge labels** to show real-time measurements, such as the average byte rate, directly on the flow lines.

For reporting or external analysis, the **Topology** view provides an export feature. You can download the current graphical representation as a PNG image or generate a direct link to the specific view configuration to share with other administrators. These tools ensure that network insights are both accessible and easily documented.

8.3.1. Working with the Topology view

Access the **Topology** view to visually inspect cluster network relationships and select individual components to view detailed traffic metrics and metadata.

As an administrator, you can navigate to the **Topology** view to see the details and metrics of the component.

Prerequisites

- You have administrator access.

Procedure

1. Navigate to **Observe → Network Traffic**.
2. In the **Network Traffic** page, click the **Topology** tab.

You can click each component in the **Topology** to view the details and metrics of the component.

8.3.2. Configuring the advanced options for the Topology view

Review the available advanced options in the **Topology** view to customize display settings, configure component grouping and layouts, and export the network graph as an image.

You can customize and export the view by using **Show advanced options**. The advanced options view has the following features:

- **Find in view**: To search the required components in the view.
- **Display options**: To configure the following options:
  - **Edge labels**: To show the specified measurements as edge labels. The default is to show the **Average rate** in **Bytes**.
  - **Scope**: To select the scope of components between which the network traffic flows. The default value is **Namespace**.
  - **Groups**: To enhance the understanding of ownership by grouping the components. The default value is **None**.
  - **Layout**: To select the layout of the graphical representation. The default value is **ColaNoForce**.
  - **Show**: To select the details that need to be displayed. All the options are checked by default. The options available are: **Edges**, **Edges label**, and **Badges**.
  - **Truncate labels**: To select the required width of the label from the drop-down list. The default value is **M**.
  - **Collapse groups**: To expand or collapse the groups. The groups are expanded by default. This option is disabled if **Groups** has the value of **None**.

8.3.2.1. Exporting the topology view

To export the view, click **Export topology view**. The view is downloaded in PNG format.

8.4. FILTERING THE NETWORK TRAFFIC

Review the available query options and filtering parameters in the **Network Traffic** view to optimize data searches, analyze specific log types, and manage directional traffic visibility.

By default, the **Network Traffic** page displays the traffic flow data in the cluster based on the default filters configured in the **FlowCollector** instance. You can use the filter options to observe the required data by changing the preset filter.

Alternatively, you can access the traffic flow data in the **Network Traffic** tab of the **Namespaces**, **Services**, **Routes**, **Nodes**, and **Workloads** pages which provide the filtered data of the corresponding aggregations.

Query Options

You can use **Query Options** to optimize the search results, as listed below:

- **Log Type**: The available options **Conversation** and **Flows** provide the ability to query flows by log type, such as flow log, new conversation, completed conversation, and a heartbeat, which is a periodic record with updates for long conversations. A conversation is an aggregation of flows between the same peers.
- **Match filters**: You can determine the relation between different filter parameters selected in the advanced filter. The available options are **Match all** and **Match any**. **Match all** provides results that match all the values, and **Match any** provides results that match any of the values entered. The default value is **Match all**.
- **Datasource**: You can choose the datasource to use for queries: **Loki**, **Prometheus**, or **Auto**. Notable performance improvements can be realized when using Prometheus as a datasource rather than Loki, but Prometheus supports a limited set of filters and aggregations. The default datasource is **Auto**, which uses Prometheus on supported queries or uses Loki if the query does not support Prometheus.
- **Drops filter**: You can view different levels of dropped packets with the following query options:
  - **Fully dropped** shows flow records with fully dropped packets.
  - **Containing drops** shows flow records that contain drops but can be sent.
  - **Without drops** shows records that contain sent packets.
  - **All** shows all the aforementioned records.
- **Limit**: The data limit for internal backend queries. Depending upon the matching and the filter settings, the number of traffic flow data is displayed within the specified limit.

Quick filters

The default values in **Quick filters** drop-down menu are defined in the **FlowCollector** configuration. You can modify the options from console.

Advanced filters

You can set the advanced filters, **Common**, **Source**, or **Destination**, by selecting the parameter to be filtered from the dropdown list. The flow data is filtered based on the selection. To enable or disable the applied filter, you can click on the applied filter listed below the filter options.

You can toggle between ↑ **One way** and ↑ ↓ **Back and forth** filtering. The ↑ **One way** filter shows only **Source** and **Destination** traffic according to your filter selections. You can use **Swap** to change the directional view of the **Source** and **Destination** traffic. The ↑ ↓ **Back and forth** filter includes return traffic with the **Source** and **Destination** filters. The directional flow of network traffic is shown in the **Direction** column in the Traffic flows table as **Ingress`or `Egress** for inter-node traffic and `Inner`for traffic inside a single node.

You can click **Reset defaults** to remove the existing filters, and apply the filter defined in **FlowCollector** configuration.

> **NOTE**
>
> To understand the rules of specifying the text value, click **Learn More**.

Additional resources

- Configuring Quick Filters
- Flow Collector sample resource

CHAPTER 9. NETWORK OBSERVABILITY ALERTS

The Network Observability Operator provides alerts using built-in metrics and the OpenShift Container Platform monitoring stack to quickly indicate your cluster's network health.

IMPORTANT

Network observability alerts is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

9.1. ABOUT NETWORK OBSERVABILITY ALERTS

Network observability includes predefined alerts. Use these alerts to gain insight into the health and performance of your OpenShift Container Platform applications and infrastructure.

The predefined alerts provide a quick health indication of your cluster's network in the **Network Health** dashboard. You can also customize alerts using Prometheus Query Language (PromQL) queries.

By default, network observability creates alerts that are contextual to the features you enable.

For example, packet drop-related alerts are created only if the **PacketDrop** agent feature is enabled in the **FlowCollector** custom resource (CR). Alerts are built on metrics, and you might see configuration warnings if enabled alerts are missing their required metrics.

You can configure these metrics in the **spec.processor.metrics.includeList** object of the **FlowCollector** CR.

9.1.1. List of default alert templates

These alert templates are installed by default:

PacketDropsByDevice

Triggers on high percentage of packet drops from devices (**/proc/net/dev**).

PacketDropsByKernel

Triggers on high percentage of packet drops by the kernel; it requires the **PacketDrop** agent feature.

IPsecErrors

Triggers when IPsec encryption errors are detected by network observability; it requires the **IPSec** agent feature.

NetpolDenied

Triggers when traffic denied by network policies is detected by network observability; it requires the **NetworkEvents** agent feature.

LatencyHighTrend

Triggers when an increase of TCP latency is detected by network observability; it requires the **FlowRTT** agent feature.

DNSErrors

Triggers when DNS errors are detected by network observability; it requires the **DNSTracking** agent feature.

These are operational alerts that relate to the self-health of network observability:

NetObservNoFlows

Triggers when no flows are being observed for a certain period.

NetObservLokiError

Triggers when flows are being dropped due to Loki errors.

You can configure, extend, or disable alerts for network observability. You can view the resulting **PrometheusRule** resource in the default **netobserv** namespace by running the following command:

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

9.1.2. Network Health dashboard

When alerts are enabled in the Network Observability Operator, two things happen:

- New alerts appear in **Observe → Alerting → Alerting rules** tab in the OpenShift Container Platform web console.
- A new **Network Health** dashboard appears in OpenShift Container Platform web console → **Observe**.

The **Network Health** dashboard provides a summary of triggered alerts and pending alerts, distinguishing between critical, warning, and minor issues. Alerts for rule violations are displayed in the following tabs:

- **Global**: Shows alerts that are global to the cluster.
- **Nodes**: Shows alerts for rule violations per node.
- **Namespaces**: Shows alerts for rule violations per namespace.

Click on a resource card to see more information. Next to each alert, a three dot menu appears. From this menu, you can navigate to **Network Traffic → Traffic flows** to see more detailed information for the selected resource.

9.2. ENABLING TECHNOLOGY PREVIEW ALERTS IN NETWORK OBSERVABILITY

Network Observability Operator alerts are a Technology Preview feature. To use this feature, you must enable it in the **FlowCollector** custom resource (CR), and then continue with configuring alerts to your specific needs.

Procedure

1. Edit the **FlowCollector** CR to set the experimental alerts flag to **true**:

```
apiVersion: flows.netobserv.io/v1beta1
kind: FlowCollector
```

```
metadata:
  name: flow-collector
spec:
  processor:
    advanced:
      env:
        EXPERIMENTAL_ALERTS_HEALTH: "true"
```

You can still use the existing method for creating alerts. For more information, see "Creating alerts".

9.2.1. Configuring predefined alerts

Alerts in the Network Observability Operator are defined using alert templates and variants in the **spec.processor.metrics.alerts** object of the **FlowCollector** custom resource (CR). You can customize the default templates and variants for flexible, fine-grained alerting.

After you enable alerts, the **Network Health** dashboard appears in the **Observe** section of the OpenShift Container Platform web console.

For each template, you can define a list of variants, each with their own thresholds and grouping configurations. For more information, see the "List of default alert templates".

Here is an example:

```
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
```

NOTE

Customizing an alert replaces the default configuration for that template. If you want to keep the default configurations, you must manually replicate them.

9.2.2. About the PromQL expression for alerts

Learn about the base query for Prometheus Query Language (**PromQL**), and how to customize it so you can configure network observability alerts for your specific needs.

The alerting API in the network observability **FlowCollector** custom resource (**CR**) is mapped to the Prometheus Operator API, generating a **PrometheusRule**. You can see the **PrometheusRule** in the default **netobserv** namespace by running the following command:

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

9.2.2.1. An example query for an alert in a surge of incoming traffic

This example provides the base **PromQL** query pattern for an alert about a surge in incoming traffic:

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

This query calculates the byte rate coming from the **openshift-ingress** namespace to any of your workloads' namespaces over the past 30 minutes.

You can customize the query, including retaining only some rates, running the query for specific time periods, and setting a final threshold.

Filtering noise

Appending **> 1000** to this query retains only the rates observed that are greater than **1 KB/s**, which eliminates noise from low-bandwidth consumers.
(sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}[30m])) by (DstK8S_Namespace) > 1000)

The byte rate is relative to the sampling interval defined in the **FlowCollector** custom resource (**CR**) configuration. If the sampling interval is **1:100**, the actual traffic might be approximately 100 times higher than the reported metrics.

Time comparison

You can run the same query for a particular period of time using the **offset** modifier. For example, a query for one day earlier can be run using **offset 1d**, and a query for five hours ago can be run using **offset 5h**.
sum(rate(netobserv_workload_ingress_bytes_total{SrcK8S_Namespace="openshift-ingress"}[30m] offset 1d)) by (DstK8S_Namespace))

You can use the formula **100 * (<query now> - <query from the previous day>) / <query from the previous day>** to calculate the percentage of increase compared to the previous day. This value can be negative if the byte rate today is lower than the previous day.

Final threshold

You can apply a final threshold to filter increases that are lower than the desired percentage. For example, **> 100** eliminates increases that are lower than 100%.

Together, the complete expression for the **PrometheusRule** looks like the following:

```
...
      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. Alert metadata fields

The Network Observability Operator uses components from other OpenShift Container Platform features, such as the monitoring stack, to enhance visibility into network traffic. For more information, see: "Monitoring stack architecture".

Some metadata must be configured for the alert definitions. This metadata is used by Prometheus and the **Alertmanager** service from the monitoring stack, or by the **Network Health** dashboard.

The following example shows an **AlertingRule** resource with the configured metadata:

```
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
```

where:

spec.groups.rules.alert.labels.netobserv

Specifies the alert for the **Network Health** dashboard to detect when set to **true**.

spec.groups.rules.alert.labels.severity

Specifies the severity of the alert. The following values are valid: **critical**, **warning**, or **info**.

You can leverage the output labels from the defined **PromQL** expression in the **message** annotation. In the example, since results are grouped per **DstK8S_Namespace**, the expression **{{ $labels.DstK8S_Namespace }}** is used in the message text.

The **netobserv_io_network_health** annotation is optional, and controls how the alert is rendered on the **Network Health** page.

The **netobserv_io_network_health** annotation is a JSON string consisting of the following fields:

Table 9.1. Fields for the netobserv_io_network_health annotation

Field	Type	Description
namespaceLabels	List of strings	One or more labels that hold namespaces. When provided, the alert appears under the **Namespaces** tab.
nodeLabels	List of strings	One or more labels that hold node names. When provided, the alert appears under the **Nodes** tab.
threshold	String	The alert threshold, expected to match the threshold defined in the **PromQL** expression.
unit	String	The data unit, used only for display purposes.
upperBound	String	An upper bound value used to compute the score on a closed scale. Metric values exceeding this bound are clamped.
links	List of objects	A list of links to display contextually with the alert. Each link requires a **name** (display name) and **url**.
trafficLinkFilter	String	An additional filter to inject into the URL for the **Network Traffic** page.

The **namespaceLabels** and **nodeLabels** are mutually exclusive. If neither is provided, the alert appears under the **Global** tab.

9.2.3. Creating custom alert rules

Use the Prometheus Query Language (**PromQL**) to define a custom **AlertingRule** resource to trigger alerts based on specific network metrics (e.g., traffic surges).

Prerequisites

- Familiarity with **PromQL**.
- You have installed OpenShift Container Platform 4.14 or later.

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the Network Observability Operator.

Procedure

1. Create a YAML file named **custom-alert.yaml** that contains your **AlertingRule** resource.
2. Apply the custom alert rule by running the following command:

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

Verification

1. Verify that the **PrometheusRule** resource was created in the **netobserv** namespace by running the following command:

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

   The output should include the **netobserv-alerts** rule you just created, confirming that the resource was generated correctly.

2. Confirm the rule is active by checking the **Network Health** dashboard in the OpenShift Container Platform web console → **Observe**.

9.2.4. Disabling predefined alerts

Alert templates can be disabled in the **spec.processor.metrics.disableAlerts** field of the **FlowCollector** custom resource (CR). This setting accepts a list of alert template names. For a list of alert template names, see: "List of default alerts".

If a template is disabled and overridden in the **spec.processor.metrics.alerts** field, the disable setting takes precedence and the alert rule is not created.

Additional resources

- List of default alerts
- Viewing network observability metrics dashboards
- Creating alerts
- Monitoring stack architecture

CHAPTER 10. USING METRICS WITH DASHBOARDS AND ALERTS

The Network Observability Operator uses the **flowlogs-pipeline** component to generate metrics from flow logs. Use these metrics to set custom alerts and view dashboards for network activity analysis.

10.1. VIEWING NETWORK OBSERVABILITY METRICS DASHBOARDS

View network observability metrics dashboards using the **Overview** tab in the OpenShift Container Platform console to monitor overall traffic flow and system health, with options to filter metrics by node, namespace, owner, pod, and service.

Procedure

1. In the web console **Observe → Dashboards**, select the **Netobserv** dashboard.
2. View network traffic metrics in the following categories, with each having the subset per node, namespace, source, and destination:
   - **Byte rates**
   - **Packet drops**
   - **DNS**
   - **RTT**
3. Select the **Netobserv/Health** dashboard.
4. View metrics about the health of the Operator in the following categories, with each having the subset per node, namespace, source, and destination.
   - **Flows**
   - **Flows Overhead**
   - **Flow rates**
   - **Agents**
   - **Processor**
   - **Operator**

Infrastructure and **Application** metrics are shown in a split-view for namespace and workloads.

10.2. NETWORK OBSERVABILITY METRICS

Review the comprehensive list of network observability metrics, prefixed by **netobserv_**, which you can configure in the **FlowCollector** resource and use to monitor traffic and create Prometheus alerts.

Metrics generated by the **flowlogs-pipeline** are configurable in the **spec.processor.metrics.includeList** of the **FlowCollector** custom resource to add or remove metrics.

You can also create alerts by using the **includeList** metrics in Prometheus rules, as shown in the example "Creating alerts".

When looking for these metrics in Prometheus, such as in the Console through **Observe → Metrics**, or when defining alerts, all the metrics names are prefixed with **netobserv_**. For example, **netobserv_namespace_flows_total**. Available metrics names are as follows:

includeList metrics names

Names followed by an asterisk * are enabled by default.

- **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 metrics names

When the **PacketDrop** feature is enabled in **spec.agent.ebpf.features** (with **privileged** mode), the following additional metrics are available:

- **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 metrics names

When the **DNSTracking** feature is enabled in **spec.agent.ebpf.features**, the following additional metrics are available:

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

FlowRTT metrics names

When the **FlowRTT** feature is enabled in **spec.agent.ebpf.features**, the following additional metrics are available:

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

Network events metrics names

When **NetworkEvents** feature is enabled, this metric is available by default:

- **namespace_network_policy_events_total**

10.3. CREATING ALERTS

Create custom **AlertingRule** resources based on **Netobserv** dashboard metrics to define conditions that trigger alerts in the OpenShift Container Platform console.

Prerequisites

- You have access to the cluster as a user with the cluster-admin role or with view permissions for all projects.
- You have the Network Observability Operator installed.

Procedure

1. Create a YAML file by clicking the import icon, **+**.
2. Add an alerting rule configuration to the YAML file. In the YAML sample that follows, an alert is created for when the cluster ingress traffic reaches a given threshold of 10 MBps per destination workload.

```
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      1
      for: 30s
      labels:
        severity: warning
```

1. The **netobserv_workload_ingress_bytes_total** metric is enabled by default in **spec.processor.metrics.includeList**.

3. Click **Create** to apply the configuration file to the cluster.

10.4. CUSTOM METRICS

Define custom metrics from flowlog data using the **FlowMetric** API, leveraging log fields as Prometheus labels to customize dashboard information and monitor specific cluster data.

In every flowlogs data that is collected, there are several fields labeled per log, such as source name and destination name. These fields can be leveraged as Prometheus labels to enable the customization of cluster information on your dashboard.

10.5. CONFIGURING CUSTOM METRICS BY USING FLOWMETRIC API

Configure the **FlowMetric** API to create custom Prometheus metrics by mapping flow log fields as labels to meet specific monitoring needs.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. In the **Provided APIs** heading for the **NetObserv Operator**, select **FlowMetric**.
3. In the **Project:** dropdown list, select the project of the Network Observability Operator instance.
4. Click **Create FlowMetric**.
5. Configure the **FlowMetric** resource, similar to the following sample configurations:

Generate a metric that tracks ingress bytes received from cluster external sources

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-traffic
  namespace: netobserv      1
```

```
spec:
  metricName: cluster_external_ingress_bytes_total (2)
  type: Counter (3)
  valueField: Bytes
  direction: Ingress (4)
  labels:
[DstK8S_HostName,DstK8S_Namespace,DstK8S_OwnerName,DstK8S_OwnerType] (5)
  filters: (6)
  - field: SrcSubnetLabel
    matchType: Absence
```

1. The **FlowMetric** resources need to be created in the namespace defined in the **FlowCollector spec.namespace**, which is **netobserv** by default.
2. The name of the Prometheus metric, which in the web console appears with the prefix **netobserv-<metricName>**.
3. The **type** specifies the type of metric. The **Counter type** is useful for counting bytes or packets.
4. The direction of traffic to capture. If not specified, both ingress and egress are captured, which can lead to duplicated counts.
5. Labels define what the metrics look like and the relationship between the different entities and also define the metrics cardinality. For example, **SrcK8S_Name** is a high cardinality metric.
6. Refines results based on the listed criteria. In this example, selecting only the cluster external traffic is done by matching only flows where **SrcSubnetLabel** is absent. This assumes the subnet labels feature is enabled (via **spec.processor.subnetLabels**), which is done by default.

Verification

1. Once the pods refresh, navigate to **Observe** → **Metrics**.
2. In the **Expression** field, type the metric name to view the corresponding result. You can also enter an expression, such as **topk(5, sum(rate(netobserv_cluster_external_ingress_bytes_total{DstK8S_Namespace="my-namespace"}[2m])) by (DstK8S_HostName, DstK8S_OwnerName, DstK8S_OwnerType))**

Show RTT latency for cluster external ingress traffic

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-rtt
  namespace: netobserv (1)
spec:
  metricName: cluster_external_ingress_rtt_seconds
  type: Histogram (2)
  valueField: TimeFlowRttNs
  direction: Ingress
```

```
  labels:
[DstK8S_HostName,DstK8S_Namespace,DstK8S_OwnerName,DstK8S_OwnerType]
  filters:
  - field: SrcSubnetLabel
    matchType: Absence
  - field: TimeFlowRttNs
    matchType: Presence
  divider: "1000000000" (3)
  buckets: [".001", ".005", ".01", ".02", ".03", ".04", ".05", ".075", ".1", ".25", "1"] (4)
```

1. The **FlowMetric** resources need to be created in the namespace defined in the **FlowCollector spec.namespace**, which is **netobserv** by default.
2. The **type** specifies the type of metric. The **Histogram type** is useful for a latency value (**TimeFlowRttNs**).
3. Since the Round-trip time (RTT) is provided as nanos in flows, use a divider of 1 billion to convert into seconds, which is standard in Prometheus guidelines.
4. The custom buckets specify precision on RTT, with optimal precision ranging between 5ms and 250ms.

Verification

1. Once the pods refresh, navigate to **Observe** → **Metrics**.
2. In the **Expression** field, you can type the metric name to view the corresponding result.

10.6. CREATING METRICS FROM NESTED OR ARRAY FIELDS IN THE TRAFFIC FLOWS TABLE

Create a **FlowMetric** custom resource to generate metrics for nested or array fields in the **Traffic flows** table, such as **Network events** or **Interfaces**.

IMPORTANT

OVN Observability / Viewing **NetworkEvents** is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

IMPORTANT

OVN Observability and the ability to view and track network events is available only in OpenShift Container Platform 4.17 and 4.18.

The following example shows how to generate metrics from the **Network events** field for network policy events.

Prerequisites

- Enable **NetworkEvents feature**. See the Additional resources for how to do this.
- A network policy specified.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. In the **Provided APIs** heading for the **NetObserv Operator**, select **FlowMetric**.
3. In the **Project** dropdown list, select the project of the Network Observability Operator instance.
4. Click **Create FlowMetric**.
5. Create **FlowMetric** resources to add the following configurations:

Configuration counting network policy events per policy name and namespace

```
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] (1)
  filters:
  - field: NetworkEvents>Feature
    value: acl
  flatten: [NetworkEvents] (2)
  remap: (3)
    "NetworkEvents>Type": type
    "NetworkEvents>Namespace": namespace
    "NetworkEvents>Name": name
    "NetworkEvents>Direction": direction
```

1. These labels represent the nested fields for **Network Events** from the **Traffic flows** table. Each network event has a specific type, namespace, name, action, and direction. You can alternatively specify the **Interfaces** if **NetworkEvents** is unavailable in your OpenShift Container Platform version.
2. Optional: You can choose to represent a field that contains a list of items as distinct items.
3. Optional: You can rename the fields in Prometheus.

Verification

1. In the web console, navigate to **Observe** → **Dashboards** and scroll down to see the **Network Policy** tab.

2. You should begin seeing metrics filter in based on the metric you created along with the network policy specifications.

IMPORTANT

High cardinality can affect the memory usage of Prometheus. You can check whether specific labels have high cardinality in the Network Flows format reference .

10.7. CONFIGURING CUSTOM CHARTS USING FLOWMETRIC API

Generate custom charts for OpenShift Container Platform web console dashboards by defining the charts section of the **FlowMetric** custom resource.

You can view custom charts as an administrator in the **Dashboard** menu.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.

2. In the **Provided APIs** heading for the **NetObserv Operator**, select **FlowMetric**.

3. In the **Project:** dropdown list, select the project of the Network Observability Operator instance.

4. Click **Create FlowMetric**.

5. Configure the **FlowMetric** resource, similar to the following sample configurations:

Chart for tracking ingress bytes received from cluster external sources

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-traffic
  namespace: netobserv  1
# ...
  charts:
  - dashboardName: Main  2
    title: External ingress traffic
    unit: Bps
    type: SingleStat
    queries:
    - promQL: "sum(rate($METRIC[2m]))"
      legend: ""
  - dashboardName: Main  3
    sectionName: External
    title: Top external ingress traffic per workload
    unit: Bps
    type: StackArea
    queries:
    - promQL: "sum(rate($METRIC{DstK8S_Namespace!=\"\"}[2m])) by (DstK8S_Namespace,
```

```
DstK8S_OwnerName)"
      legend: "{{DstK8S_Namespace}} / {{DstK8S_OwnerName}}"
# ...
```

1. The **FlowMetric** resources need to be created in the namespace defined in the **FlowCollector spec.namespace**, which is **netobserv** by default.

Verification

1. Once the pods refresh, navigate to **Observe** → **Dashboards**.
2. Search for the **NetObserv / Main** dashboard. View two panels under the **NetObserv / Main** dashboard, or optionally a dashboard name that you create:
   - A textual single statistic showing the global external ingress rate summed across all dimensions
   - A timeseries graph showing the same metric per destination workload

For more information about the query language, refer to the Prometheus documentation.

Chart for RTT latency for cluster external ingress traffic

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-traffic
  namespace: netobserv (1)
# ...
  charts:
  - dashboardName: Main (2)
    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. The **FlowMetric** resources need to be created in the namespace defined in the **FlowCollector** **spec.namespace**, which is **netobserv** by default.

2 3 4 Using a different **dashboardName** creates a new dashboard that is prefixed with **Netobserv**. For example, **Netobserv / <dashboard_name>**.

This example uses the **histogram_quantile** function to show **p50** and **p99**.

You can show averages of histograms by dividing the metric, **$METRIC_sum**, by the metric, **$METRIC_count**, which are automatically generated when you create a histogram. With the preceding example, the Prometheus query to do this is as follows:

```
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"
```

Verification

1. Once the pods refresh, navigate to **Observe** → **Dashboards**.
2. Search for the **NetObserv / Main** dashboard. View the new panel under the **NetObserv / Main** dashboard, or optionally a dashboard name that you create.

For more information about the query language, refer to the Prometheus documentation.

10.8. DETECTING SYN FLOODING USING THE FLOWMETRIC API AND TCP FLAGS

Deploy a custom **AlertingRule** and **FlowMetric** configuration to monitor TCP flags, enabling real-time detection and alerting for SYN flooding attacks on the cluster.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. In the **Provided APIs** heading for the **NetObserv Operator**, select **FlowMetric**.
3. In the **Project** dropdown list, select the project of the Network Observability Operator instance.
4. Click **Create FlowMetric**.
5. Create **FlowMetric** resources to add the following configurations:

   Configuration counting flows per destination host and resource, with TCP flags

```
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_N
amespace,Flags]
```

Configuration counting flows per source host and resource, with TCP flags

```
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_N
amespace,Flags]
```

6. Deploy the following **AlertingRule** resource to alert for SYN flooding:

AlertingRule for SYN flooding

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: netobserv-syn-alerts
  namespace: openshift-monitoring
# ...
  spec:
  groups:
  - name: NetObservSYNAlerts
    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) In this example, the threshold for the alert is **300**; however, you can adapt this value empirically. A threshold that is too low might produce false-positives, and if it's too high it might miss actual attacks.

Verification

1. In the web console, click **Manage Columns** in the **Network Traffic** table view and click **TCP flags**.
2. In the **Network Traffic** table view, filter on **TCP protocol SYN TCPFlag**. A large number of flows with the same **byteSize** indicates a SYN flood.
3. Go to **Observe** → **Alerting** and select the **Alerting Rules** tab.
4. Filter on **netobserv-synflood-in alert**. The alert should fire when SYN flooding occurs.

Additional resources

- Filtering eBPF flow data using a global rule
- Creating alerting rules for user-defined projects
- Troubleshooting high cardinality metrics- Determining why Prometheus is consuming a lot of disk space

CHAPTER 11. MONITORING THE NETWORK OBSERVABILITY OPERATOR

Use the OpenShift Container Platform web console to monitor alerts related to the Network Observability Operator's health. This helps you maintain system stability and quickly detect operational issues.

11.1. HEALTH DASHBOARDS

View the Network Observability Operator health dashboards in the OpenShift Container Platform web console to monitor the health status, resource usage, and internal statistics of the operator and its components.

Metrics are located in the **Observe → Dashboards** page in the OpenShift Container Platform web console. You can view metrics about the health of the Network Observability Operator in the following categories:

- **Flows per second**
- **Sampling**
- **Errors last minute**
- **Dropped flows per second**
- **Flowlogs-pipeline statistics**
- **Flowlogs-pipleine statistics views**
- **eBPF agent statistics views**
- **Operator statistics**
- **Resource usage**

11.2. HEALTH ALERTS

Understand the health alerts generated by the Network Observability Operator, which trigger banners when conditions like Loki ingestion errors, zero flow ingestion, or dropped eBPF flows occur.

A health alert banner that directs you to the dashboard can appear on the **Network Traffic** and **Home** pages if an alert is triggered. Alerts are generated in the following cases:

- The **NetObservLokiError** alert occurs if the **flowlogs-pipeline** workload is dropping flows because of Loki errors, such as if the Loki ingestion rate limit has been reached.
- The **NetObservNoFlows** alert occurs if no flows are ingested for a certain amount of time.
- The **NetObservFlowsDropped** alert occurs if the Network Observability eBPF agent hashmap table is full, and the eBPF agent processes flows with degraded performance, or when the capacity limiter is triggered.

11.3. VIEWING HEALTH INFORMATION

View the **Netobserv/Health** dashboard within the OpenShift Container Platform web console to monitor the health status and resource usage of the Network Observability Operator and its components.

Prerequisites

- You have the Network Observability Operator installed.
- You have access to the cluster as a user with the **cluster-admin** role or with view permissions for all projects.

Procedure

1. From the **Administrator** perspective in the web console, navigate to **Observe → Dashboards**.
2. From the **Dashboards** dropdown, select **Netobserv/Health**.
3. View the metrics about the health of the Operator that are displayed on the page.

11.3.1. Disabling health alerts

Disable specific health alerts, such as **NetObservLokiError** or **NetObservNoFlows**, by editing the **FlowCollector** resource and using the **spec.processor.metrics.disableAlerts** specification.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster** then select the **YAML** tab.
4. Add **spec.processor.metrics.disableAlerts** to disable health alerts, as in the following YAML sample:

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

(1) You can specify one or a list with both types of alerts to disable.

11.4. CREATING LOKI RATE LIMIT ALERTS FOR THE NETOBSERV DASHBOARD

Create a custom **AlertingRule** resource based on Loki metrics to monitor for and trigger alerts when the Loki ingestion rate limits are reached, indicated by HTTP 429 errors.

You can create custom alerting rules for the **Netobserv** dashboard metrics to trigger alerts when Loki rate limits have been reached.

Prerequisites

- You have access to the cluster as a user with the cluster-admin role or with view permissions for all projects.
- You have the Network Observability Operator installed.

Procedure

1. Create a YAML file by clicking the import icon, **+**.
2. Add an alerting rule configuration to the YAML file. In the YAML sample that follows, an alert is created for when Loki rate limits have been reached:

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: loki-alerts
  namespace: openshift-monitoring
spec:
  groups:
  - name: LokiRateLimitAlerts
    rules:
    - alert: LokiTenantRateLimit
      annotations:
        message: |-
          {{ $labels.job }} {{ $labels.route }} is experiencing 429 errors.
        summary: "At any number of requests are responded with the rate limit error code."
      expr: sum(irate(loki_request_duration_seconds_count{status_code="429"}[1m])) by (job, namespace, route) / sum(irate(loki_request_duration_seconds_count[1m])) by (job, namespace, route) * 100 > 0
      for: 10s
      labels:
        severity: warning
```

3. Click **Create** to apply the configuration file to the cluster.

11.5. USING THE EBPF AGENT ALERT

Resolve the **NetObservAgentFlowsDropped** alert, which occurs when the eBPF agent hashmap is full, by increasing the **spec.agent.ebpf.cacheMaxFlows** value in the **FlowCollector** custom resource.

An alert, **NetObservAgentFlowsDropped**, is also triggered when the capacity limiter is triggered. If you see this alert, consider increasing the **cacheMaxFlows** in the **FlowCollector**, as shown in the following example.

> **NOTE**
>
> Increasing the **cacheMaxFlows** might increase the memory usage of the eBPF agent.

Procedure

1. In the web console, navigate to **Operators** → **Installed Operators**.
2. Under the **Provided APIs** heading for the **Network Observability Operator**, select **Flow Collector**.
3. Select **cluster**, and then select the **YAML** tab.
4. Increase the **spec.agent.ebpf.cacheMaxFlows** value, as shown in the following YAML sample:

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

(1) Increase the **cacheMaxFlows** value from its value at the time of the **NetObservAgentFlowsDropped** alert.

Additional resources

- Creating alerting rules for user-defined projects

CHAPTER 12. SCHEDULING RESOURCES

Taints and tolerations help you control which nodes host certain pods. Use these tools, along with node selectors, to guide the placement of network observability components.

A node selector specifies a map of key/value pairs that are defined using custom labels on nodes and selectors specified in pods.

For the pod to be eligible to run on a node, the pod must have the same key/value node selector as the label on the node.

12.1. NETWORK OBSERVABILITY DEPLOYMENT IN SPECIFIC NODES

Configure the **FlowCollector** resource using scheduling specifications, including **NodeSelector**, **Tolerations**, and **Affinity**, to control the deployment of network observability components on specific nodes.

The **spec.agent.ebpf.advanced.scheduling**, **spec.processor.advanced.scheduling**, and **spec.consolePlugin.advanced.scheduling** specifications have the following configurable settings:

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

Sample **FlowCollector** resource for **spec.<component>.advanced.scheduling**

```
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: """
# ...
```

Additional resources

- Understanding taints and tolerations
- Assign Pods to Nodes (Kubernetes documentation)
- Pod Priority and Preemption (Kubernetes documentation)

CHAPTER 13. SECONDARY NETWORKS

You can configure the Network Observability Operator to collect and enrich network flow data from secondary networks, such as **SR-IOV** and **OVN-Kubernetes**.

13.1. PREREQUISITES

- Access to an OpenShift Container Platform cluster with an additional network interface, such as a secondary interface or an L2 network.

13.2. CONFIGURING MONITORING FOR SR-IOV INTERFACE TRAFFIC

Configure the **FlowCollector** resource to monitor traffic on Single Root I/O Virtualization (SR-IOV) device by setting the **spec.agent.ebpf.privileged** field to **true**, which enables the eBPF agent to monitor other network namespaces.

The eBPF agent monitors other network namespaces in addition to the host network namespaces, which are monitored by default. When a pod with a virtual functions (VF) interface is created, a new network namespace is created. With **SRIOVNetwork** policy **IPAM** configurations specified, the VF interface is migrated from the host network namespace to the pod network namespace.

Prerequisites

- Access to an OpenShift Container Platform cluster with a SR-IOV device.
- The **SRIOVNetwork** custom resource (CR) **spec.ipam** configuration must be set with an IP address from the range that the interface lists or from other plugins.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**.
2. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
3. Select **cluster** and then select the **YAML** tab.
4. Configure the **FlowCollector** custom resource. A sample configuration is as follows:

Configure FlowCollector for SR-IOV monitoring

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

1 The **spec.agent.ebpf.privileged** field value must be set to **true** to enable SR-IOV monitoring.

Additional resources

- Configuring an SR-IOV network device

13.3. CONFIGURING VIRTUAL MACHINE (VM) SECONDARY NETWORK INTERFACES FOR NETWORK OBSERVABILITY

Configure the **FlowCollector** to monitor VM secondary network traffic by setting the eBPF agent to **privileged** mode and defining the indexing for secondary networks, enabling the capture and enrichment of flows from OpenShift Virtualization.

Network flows coming from VMs that are connected to the default internal pod network are automatically captured by network observability.

Procedure

1. Get information about the virtual machine launcher pod by running the following command. This information is used in Step 5:

```
$ 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", (1)
        "interface": "podc0f69e19ba2", (2)
        "ips": [ (3)
          "10.10.10.15"
        ],
        "mac": "02:fb:f8:00:00:12", (4)
        "dns": {}
      }]
  name: virt-launcher-fedora-aqua-fowl-13-zr2x9
  namespace: my-vms
spec:
```

```
# ...
status:
# ...
```

1. The name of the secondary network.
2. The network interface name of the secondary network.
3. The list of IPs used by the secondary network.
4. The MAC address used for secondary network.

2. In the web console, navigate to **Operators** → **Installed Operators**.
3. Under the **Provided APIs** heading for the **NetObserv Operator**, select **Flow Collector**.
4. Select **cluster** and then select the **YAML** tab.
5. Configure **FlowCollector** based on the information you found from the additional network investigation:

```
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. Ensure that the eBPF agent is in **privileged** mode so that flows are collected for secondary interfaces.
2. Define the fields to use for indexing the virtual machine launcher pods. It is recommended to use the **MAC** address as the indexing field to get network flows enrichment for secondary interfaces. If you have overlapping MAC address between pods, then additional indexing fields, such as **IP** and **Interface**, could be added to have accurate enrichment.
3. If your additional network information has a MAC address, add **MAC** to the field list.
4. Specify the name of the network found in the **k8s.v1.cni.cncf.io/network-status** annotation. Usually <namespace>/<network_attachement_definition_name>.

6. Observe VM traffic:
   a. Navigate to the **Network Traffic** page.

b. Filter by **Source** IP using your virtual machine IP found in **k8s.v1.cni.cncf.io/network-status** annotation.

c. View both **Source** and **Destination** fields, which should be enriched, and identify the VM launcher pods and the VM instance as owners.

CHAPTER 14. NETWORK OBSERVABILITY CLI

14.1. INSTALLING THE NETWORK OBSERVABILITY CLI

The Network Observability CLI (**oc netobserv**) is deployed separately from the Network Observability Operator. The CLI is available as an OpenShift CLI (**oc**) plugin. It provides a lightweight way to quickly debug and troubleshoot with network observability.

14.1.1. About the Network Observability CLI

Use the Network Observability CLI (**oc netobserv**) to quickly debug and troubleshoot networking issues. This tool provides instant, live insight into flows and packets without installing the Network Observability Operator.

The Network Observability CLI is a flow and packet visualization tool that relies on eBPF agents to stream collected data to an ephemeral collector pod. It requires no persistent storage during the capture. After the run, the output is transferred to your local machine.

> **IMPORTANT**
>
> CLI capture is meant to run only for short durations, such as 8-10 minutes. If it runs for too long, it can be difficult to delete the running process.

14.1.2. Installing the Network Observability CLI

The Network Observability CLI gives you a lightweight way to quickly debug and troubleshoot network observability. It must be installed separately.

Installing the Network Observability CLI (**oc netobserv**) is a separate procedure from the Network Observability Operator installation. This means that, even if the Operator is installed from the software catalog, the **CLI** must be installed separately.

> **NOTE**
>
> Users can optionally use Krew to install the **netobserv** CLI plugin. For more information, see "Installing a CLI plugin with Krew".

Prerequisites

- You must install the OpenShift CLI (**oc**).
- You must have a macOS or Linux operating system.
- You must install either **docker** or **podman**.

> **NOTE**
>
> You can use **podman** or **docker** to run the installation commands. This procedure uses **podman**.

Procedure

1. Log in to the **Red Hat registry** by running the following command:

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

2. Extract the **oc-netobserv** file from the image by running the following commands:

```
$ 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. Move the extracted file to a directory that is on the system's **PATH**, such as **/usr/local/bin/**, by running the following command:

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

Verification

1. Verify that **oc netobserv** is available:

```
$ oc netobserv version
```

This command should produce an outcome similar to the following example:

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

Additional resources

- Installing and using CLI plugins
- Installing the CLI Manager Operator

14.2. USING THE NETWORK OBSERVABILITY CLI

You can visualize and filter the flows and packets data directly in the terminal to see specific usage, such as identifying who is using a specific port. The Network Observability CLI collects flows as JSON and database files or packets as a PCAP file, which you can use with third-party tools.

14.2.1. Capturing flows

Capture network flows and apply filters based on resources or zones directly in the CLI. This helps you solve complex use cases, such as visualizing the Round-Trip Time (RTT) between two different zones.

Table visualization in the CLI provides viewing and flow search capabilities.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the Network Observability CLI (**oc netobserv**) plugin.

Procedure

1. Capture flows with filters enabled by running the following command:

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

2. Add filters to the **live table filter** prompt in the terminal to further refine the incoming flows. For example:

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

3. Use the **PageUp** and **PageDown** keys to toggle between **None**, **Resource**, **Zone**, **Host**, **Owner** and **all of the above**.

4. To stop capturing, press **Ctrl**+**C**. The data that was captured is written to two separate files in an **./output** directory located in the same path used to install the CLI.

5. View the captured data in the **./output/flow/<capture_date_time>.json** JSON file, which contains JSON arrays of the captured data.

Example JSON file


```
{
  "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. You can use SQLite to inspect the **./output/flow/<capture_date_time>.db** database file. For example:

   a. Open the file by running the following command:

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

b. Query the data by running a SQLite **SELECT** statement, for example:

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

Example output

```
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. Capturing packets

Use the Network Observability CLI to capture network packets. You can apply filters and refine them live in the terminal for accurate, real-time debugging.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the Network Observability CLI (**oc netobserv**) plugin.

Procedure

1. Run the packet capture with filters enabled:

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

2. Add filters to the **live table filter** prompt in the terminal to refine the incoming packets. An example filter is as follows:

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

3. Use the **PageUp** and **PageDown** keys to toggle between **None**, **Resource**, **Zone**, **Host**, **Owner** and **all of the above**.
4. To stop capturing, press **Ctrl**+**C**.
5. View the captured data, which is written to a single file in an **./output/pcap** directory located in the same path that was used to install the CLI:
   a. The **./output/pcap/<capture_date_time>.pcap** file can be opened with Wireshark.

14.2.3. Capturing metrics

Generate on-demand network observability dashboards in Prometheus using a service monitor. This allows you to quickly view and analyze network metrics.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Install the Network Observability CLI (**oc netobserv**) plugin.

Procedure

1. Capture metrics with filters enabled by running the following command:

   Example output

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

2. Open the link provided in the terminal to view the **NetObserv / On-Demand** dashboard:

   Example URL

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

   > **NOTE**
   >
   > Features that are not enabled present as empty graphs.

14.2.4. Cleaning the Network Observability CLI

Use **oc netobserv cleanup** to manually remove all components installed by the Network Observability CLI from your cluster. While the client runs this command automatically after a capture, you may need to run it manually if you face connectivity issues.

Procedure

- Run the following command:

  ```
  $ oc netobserv cleanup
  ```

Additional resources

- Network Observability CLI reference

14.3. NETWORK OBSERVABILITY CLI (OC NETOBSERV) REFERENCE

The Network Observability CLI (**oc netobserv**) has most features and filtering options that are available for the Network Observability Operator. You can pass command-line arguments to enable features or filtering options.

14.3.1. Network Observability CLI usage

You can use the Network Observability CLI (**oc netobserv**) to pass command line arguments to capture flows data, packets data, and metrics for further analysis and enable features supported by the Network Observability Operator.

14.3.1.1. Syntax

The basic syntax for **oc netobserv** commands:

oc netobserv syntax

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

(1) Feature options can only be used with the **oc netobserv flows** command. They cannot be used with the **oc netobserv packets** command.

14.3.1.2. Basic commands

Table 14.1. Basic commands

Command	Description
flows	Capture flows information. For subcommands, see the "Flows capture options" table.
packets	Capture packets data. For subcommands, see the "Packets capture options" table.
metrics	Capture metrics data. For subcommands, see the "Metrics capture options" table.
follow	Follow collector logs when running in background.
stop	Stop collection by removing agent daemonset.
copy	Copy collector generated files locally.
cleanup	Remove the Network Observability CLI components.
version	Print the software version.
help	Show help.

14.3.1.3. Flows capture options

Flows capture has mandatory commands as well as additional options, such as enabling extra features about packet drops, DNS latencies, Round-trip time, and filtering.

oc netobserv flows syntax

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

Option	Description	Default
--enable_all	enable all eBPF features	false
--enable_dns	enable DNS tracking	false
--enable_ipsec	enable IPsec tracking	false
--enable_network_events	enable network events monitoring	false
--enable_pkt_translation	enable packet translation	false
--enable_pkt_drop	enable packet drop	false
--enable_rtt	enable RTT tracking	false
--enable_udn_mapping	enable User Defined Network mapping	false
--get-subnets	get subnets information	false
--privileged	force eBPF agent privileged mode	auto
--sampling	packets sampling interval	1
--background	run in background	false
--copy	copy the output files locally	prompt
--log-level	components logs	info
--max-time	maximum capture time	5m
--max-bytes	maximum capture bytes	50000000 = 50MB
--action	filter action	Accept
--cidr	filter CIDR	0.0.0.0/0
--direction	filter direction	-
--dport	filter destination port	-
--dport_range	filter destination port range	-

Option	Description	Default
--dports	filter on either of two destination ports	-
--drops	filter flows with only dropped packets	false
--icmp_code	filter ICMP code	-
--icmp_type	filter ICMP type	-
--node-selector	capture on specific nodes	-
--peer_ip	filter peer IP	-
--peer_cidr	filter peer CIDR	-
--port_range	filter port range	-
--port	filter port	-
--ports	filter on either of two ports	-
--protocol	filter protocol	-
--query	filter flows using a custom query	-
--sport_range	filter source port range	-
--sport	filter source port	-
--sports	filter on either of two source ports	-
--tcp_flags	filter TCP flags	-
--interfaces	list of interfaces to monitor, comma separated	-
--exclude_interfaces	list of interfaces to exclude, comma separated	lo

Example running flows capture on TCP protocol and port 49051 with PacketDrop and RTT features enabled:

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

14.3.1.4. Packets capture options

You can filter packets capture data the as same as flows capture by using the filters. Certain features, such as packets drop, DNS, RTT, and network events, are only available for flows and metrics capture.

oc netobserv packets syntax

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

Option	Description	Default
--background	run in background	false
--copy	copy the output files locally	prompt
--log-level	components logs	info
--max-time	maximum capture time	5m
--max-bytes	maximum capture bytes	50000000 = 50MB
--action	filter action	Accept
--cidr	filter CIDR	0.0.0.0/0
--direction	filter direction	-
--dport	filter destination port	-
--dport_range	filter destination port range	-
--dports	filter on either of two destination ports	-
--drops	filter flows with only dropped packets	false
--icmp_code	filter ICMP code	-
--icmp_type	filter ICMP type	-
--node-selector	capture on specific nodes	-
--peer_ip	filter peer IP	-
--peer_cidr	filter peer CIDR	-
--port_range	filter port range	-

Option	Description	Default
--port	filter port	-
--ports	filter on either of two ports	-
--protocol	filter protocol	-
--query	filter flows using a custom query	-
--sport_range	filter source port range	-
--sport	filter source port	-
--sports	filter on either of two source ports	-
--tcp_flags	filter TCP flags	-

Example running packets capture on TCP protocol and port 49051:

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

14.3.1.5. Metrics capture options

You can enable features and use filters on metrics capture, the same as flows capture. The generated graphs fill accordingly in the dashboard.

oc netobserv metrics syntax

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

Option	Description	Default
--enable_all	enable all eBPF features	false
--enable_dns	enable DNS tracking	false
--enable_ipsec	enable IPsec tracking	false
--enable_network_events	enable network events monitoring	false
--enable_pkt_translation	enable packet translation	false
--enable_pkt_drop	enable packet drop	false
--enable_rtt	enable RTT tracking	false

Option	Description	Default
--enable_udn_mapping	enable User Defined Network mapping	false
--get-subnets	get subnets information	false
--privileged	force eBPF agent privileged mode	auto
--sampling	packets sampling interval	1
--background	run in background	false
--log-level	components logs	info
--max-time	maximum capture time	1h
--action	filter action	Accept
--cidr	filter CIDR	0.0.0.0/0
--direction	filter direction	-
--dport	filter destination port	-
--dport_range	filter destination port range	-
--dports	filter on either of two destination ports	-
--drops	filter flows with only dropped packets	false
--icmp_code	filter ICMP code	-
--icmp_type	filter ICMP type	-
--node-selector	capture on specific nodes	-
--peer_ip	filter peer IP	-
--peer_cidr	filter peer CIDR	-
--port_range	filter port range	-
--port	filter port	-

Option	Description	Default
--ports	filter on either of two ports	-
--protocol	filter protocol	-
--query	filter flows using a custom query	-
--sport_range	filter source port range	-
--sport	filter source port	-
--sports	filter on either of two source ports	-
--tcp_flags	filter TCP flags	-
--include_list	list of metric names to generate, comma separated	namespace_flows_total,node_ingress_bytes_total,node_egress_bytes_total,workload_ingress_bytes_total
--interfaces	list of interfaces to monitor, comma separated	-
--exclude_interfaces	list of interfaces to exclude, comma separated	lo

Example running metrics capture for TCP drops

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

CHAPTER 15. FLOWCOLLECTOR API REFERENCE

The **FlowCollector** API is the underlying schema used to pilot and configure the deployments for collecting network flows. This reference guide helps you manage those critical settings.

15.1. FLOWCOLLECTOR API SPECIFICATIONS

Description

FlowCollector is the schema for the network flows collection API, which pilots and configures the underlying deployments.

Type

object

Property	Type	Description
apiVersion	**string**	APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and might reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
kind	**string**	Kind is a string value representing the REST resource this object represents. Servers might infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds
metadata	**object**	Standard object’s metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

Property	Type	Description
spec	**object**	Defines the desired state of the FlowCollector resource. *: the mention of "unsupported" or "deprecated" for a feature throughout this document means that this feature is not officially supported by Red Hat. It might have been, for example, contributed by the community and accepted without a formal agreement for maintenance. The product maintainers might provide some support for these features as a best effort only.

15.1.1. .metadata

Description

Standard object's metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

Type

object

15.1.2. .spec

Description

Defines the desired state of the FlowCollector resource.

*: the mention of "unsupported" or "deprecated" for a feature throughout this document means that this feature is not officially supported by Red Hat. It might have been, for example, contributed by the community and accepted without a formal agreement for maintenance. The product maintainers might provide some support for these features as a best effort only.

Type

object

Property	Type	Description
agent	**object**	Agent configuration for flows extraction.
consolePlugin	**object**	**consolePlugin** defines the settings related to the OpenShift Container Platform Console plugin, when available.

Property	Type	Description
deploymentModel	**string**	**deploymentModel** defines the desired type of deployment for flow processing. Possible values are: - **Direct** (default) to make the flow processor listen directly from the agents. Only recommended on small clusters, below 15 nodes. - **Kafka** to make flows sent to a Kafka pipeline before consumption by the processor. Kafka can provide better scalability, resiliency, and high availability (for more details, see https://www.redhat.com/en/topics/integration/what-is-apache-kafka).
exporters	**array**	**exporters** defines additional optional exporters for custom consumption or storage.
kafka	**object**	Kafka configuration, allowing to use Kafka as a broker as part of the flow collection pipeline. Available when the **spec.deploymentModel** is **Kafka**.
loki	**object**	**loki**, the flow store, client settings.
namespace	**string**	Namespace where Network Observability pods are deployed.
networkPolicy	**object**	**networkPolicy** defines network policy settings for Network Observability components isolation.

Property	Type	Description
processor	**object**	**processor** defines the settings of the component that receives the flows from the agent, enriches them, generates metrics, and forwards them to the Loki persistence layer and/or any available exporter.
prometheus	**object**	**prometheus** defines Prometheus settings, such as querier configuration used to fetch metrics from the Console plugin.

15.1.3. .spec.agent

Description

Agent configuration for flows extraction.

Type

object

Property	Type	Description
ebpf	**object**	**ebpf** describes the settings related to the eBPF-based flow reporter when **spec.agent.type** is set to **eBPF**.
type	**string**	**type** [deprecated (*)] selects the flows tracing agent. Previously, this field allowed to select between **eBPF** or **IPFIX**. Only **eBPF** is allowed now, so this field is deprecated and is planned for removal in a future version of the API.

15.1.4. .spec.agent.ebpf

Description

ebpf describes the settings related to the eBPF-based flow reporter when **spec.agent.type** is set to **eBPF**.

Type

object

Property	Type	Description
advanced	**object**	**advanced** allows setting some aspects of the internal configuration of the eBPF agent. This section is aimed mostly for debugging and fine-grained performance optimizations, such as **GOGC** and **GOMAXPROCS** environment variables. Set these values at your own risk. You can also override the default Linux capabilities from there.
cacheActiveTimeout	**string**	**cacheActiveTimeout** is the max period during which the reporter aggregates flows before sending. Increasing **cacheMaxFlows** and **cacheActiveTimeout** can decrease the network traffic overhead and the CPU load, however you can expect higher memory consumption and an increased latency in the flow collection.
cacheMaxFlows	**integer**	**cacheMaxFlows** is the max number of flows in an aggregate; when reached, the reporter sends the flows. Increasing **cacheMaxFlows** and **cacheActiveTimeout** can decrease the network traffic overhead and the CPU load, however you can expect higher memory consumption and an increased latency in the flow collection.
excludeInterfaces	**array (string)**	**excludeInterfaces** contains the interface names that are excluded from flow tracing. An entry enclosed by slashes, such as **/br-/**, is matched as a regular expression. Otherwise it is matched as a case-sensitive string.
features	**array (string)**	List of additional features to enable. They are all disabled by default. Enabling additional

Property	Type	Description
		features might have performance impacts. Possible values are: - **PacketDrop**: Enable the packets drop flows logging feature. This feature requires mounting the kernel debug filesystem, so the eBPF agent pods must run as privileged via **spec.agent.ebpf.privileged**. - **DNSTracking**: Enable the DNS tracking feature. - **FlowRTT**: Enable flow latency (sRTT) extraction in the eBPF agent from TCP traffic. - **NetworkEvents**: Enable the network events monitoring feature, such as correlating flows and network policies. This feature requires mounting the kernel debug filesystem, so the eBPF agent pods must run as privileged via **spec.agent.ebpf.privileged**. It requires using the OVN-Kubernetes network plugin with the Observability feature. IMPORTANT: This feature is available as a Technology Preview. - **PacketTranslation**: Enable enriching flows with packet translation information, such as Service NAT. - **EbpfManager**: [Unsupported (*)]. Use eBPF Manager to manage Network Observability eBPF programs. Pre-requisite: the eBPF Manager operator (or upstream bpfman operator) must be installed. - **UDNMapping**: Enable interfaces mapping to User Defined Networks (UDN). This feature requires mounting the kernel debug filesystem, so the eBPF agent pods must run as privileged via **spec.agent.ebpf.privileged**. It

Property	Type	Description
		requires using the OVN-Kubernetes network plugin with the Observability feature. - **IPSec**, to track flows between nodes with IPsec encryption.
flowFilter	**object**	**flowFilter** defines the eBPF agent configuration regarding flow filtering.
imagePullPolicy	**string**	**imagePullPolicy** is the Kubernetes pull policy for the image defined above
interfaces	**array (string)**	**interfaces** contains the interface names from where flows are collected. If empty, the agent fetches all the interfaces in the system, excepting the ones listed in **excludeInterfaces**. An entry enclosed by slashes, such as **/br-/**, is matched as a regular expression. Otherwise it is matched as a case-sensitive string.
kafkaBatchSize	**integer**	**kafkaBatchSize** limits the maximum size of a request in bytes before being sent to a partition. Ignored when not using Kafka. Default: 1MB.
logLevel	**string**	**logLevel** defines the log level for the Network Observability eBPF Agent
metrics	**object**	**metrics** defines the eBPF agent configuration regarding metrics.
privileged	**boolean**	Privileged mode for the eBPF Agent container. When set to **true**, the agent is able to capture more traffic, including from secondary interfaces. When ignored or set to **false**, the operator sets granular capabilities (BPF, PERFMON, NET_ADMIN) to the container. Some agent features require the privileged mode, such as packet drops tracking (see **features**) and SR-IOV support.

Property	Type	Description
resources	**object**	**resources** are the compute resources required by this container. For more information, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
sampling	**integer**	Sampling interval of the eBPF probe. 100 means one packet on 100 is sent. 0 or 1 means all packets are sampled.

15.1.5. .spec.agent.ebpf.advanced

Description

advanced allows setting some aspects of the internal configuration of the eBPF agent. This section is aimed mostly for debugging and fine-grained performance optimizations, such as **GOGC** and **GOMAXPROCS** environment variables. Set these values at your own risk. You can also override the default Linux capabilities from there.

Type

object

Property	Type	Description
capOverride	**array (string)**	Linux capabilities override, when not running as privileged. Default capabilities are BPF, PERFMON and NET_ADMIN.
env	**object (string)**	**env** allows passing custom environment variables to underlying components. Useful for passing some very concrete performance-tuning options, such as **GOGC** and **GOMAXPROCS**, that should not be publicly exposed as part of the FlowCollector descriptor, as they are only useful in edge debug or support scenarios.
scheduling	**object**	scheduling controls how the pods are scheduled on nodes.

15.1.6. .spec.agent.ebpf.advanced.scheduling

Description

scheduling controls how the pods are scheduled on nodes.

Type

object

Property	Type	Description
affinity	**object**	If specified, the pod's scheduling constraints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.
nodeSelector	**object (string)**	**nodeSelector** allows scheduling of pods only onto nodes that have each of the specified labels. For documentation, refer to https://kubernetes.io/docs/concepts/configuration/assign-pod-node/.
priorityClassName	**string**	If specified, indicates the pod's priority. For documentation, refer to https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption. If not specified, default priority is used, or zero if there is no default.
tolerations	**array**	**tolerations** is a list of tolerations that allow the pod to schedule onto nodes with matching taints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

15.1.7. .spec.agent.ebpf.advanced.scheduling.affinity

Description

If specified, the pod's scheduling constraints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

Type

object

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

Description

tolerations is a list of tolerations that allow the pod to schedule onto nodes with matching taints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

Type

array

15.1.9. .spec.agent.ebpf.flowFilter

Description

flowFilter defines the eBPF agent configuration regarding flow filtering.

Type

object

Property	Type	Description
action	**string**	**action** defines the action to perform on the flows that match the filter. The available options are **Accept**, which is the default, and **Reject**.
cidr	**string**	**cidr** defines the IP CIDR to filter flows by. Examples: **10.10.10.0/24** or **100:100:100:100::/64**
destPorts	**integer-or-string**	**destPorts** optionally defines the destination ports to filter flows by. To filter a single port, set a single port as an integer value. For example, **destPorts: 80**. To filter a range of ports, use a "start-end" range in string format. For example, **destPorts: "80-100"**. To filter two ports, use a "port1,port2" in string format. For example, **ports: "80,100"**.
direction	**string**	**direction** optionally defines a direction to filter flows by. The available options are **Ingress** and **Egress**.
enable	**boolean**	Set **enable** to **true** to enable the eBPF flow filtering feature.

Property	Type	Description
icmpCode	**integer**	**icmpCode**, for Internet Control Message Protocol (ICMP) traffic, optionally defines the ICMP code to filter flows by.
icmpType	**integer**	**icmpType**, for ICMP traffic, optionally defines the ICMP type to filter flows by.
peerCIDR	**string**	**peerCIDR** defines the Peer IP CIDR to filter flows by. Examples: **10.10.10.0/24** or **100:100:100:100::/64**
peerIP	**string**	**peerIP** optionally defines the remote IP address to filter flows by. Example: **10.10.10.10**.
pktDrops	**boolean**	**pktDrops** optionally filters only flows containing packet drops.
ports	**integer-or-string**	**ports** optionally defines the ports to filter flows by. It is used both for source and destination ports. To filter a single port, set a single port as an integer value. For example, **ports: 80**. To filter a range of ports, use a "start-end" range in string format. For example, **ports: "80-100"**. To filter two ports, use a "port1,port2" in string format. For example, **ports: "80,100"**.
protocol	**string**	**protocol** optionally defines a protocol to filter flows by. The available options are **TCP**, **UDP**, **ICMP**, **ICMPv6**, and **SCTP**.

Property	Type	Description
rules	**array**	**rules** defines a list of filtering rules on the eBPF Agents. When filtering is enabled, by default, flows that don't match any rule are rejected. To change the default, you can define a rule that accepts everything: **{ action: "Accept", cidr: "0.0.0.0/0" }**, and then refine with rejecting rules.
sampling	**integer**	**sampling** is the sampling interval for the matched packets, overriding the global sampling defined at **spec.agent.ebpf.sampling**.
sourcePorts	**integer-or-string**	**sourcePorts** optionally defines the source ports to filter flows by. To filter a single port, set a single port as an integer value. For example, **sourcePorts: 80**. To filter a range of ports, use a "start-end" range in string format. For example, **sourcePorts: "80-100"**. To filter two ports, use a "port1,port2" in string format. For example, **ports: "80,100"**.
tcpFlags	**string**	**tcpFlags** optionally defines TCP flags to filter flows by. In addition to the standard flags (RFC-9293), you can also filter by one of the three following combinations: **SYN-ACK**, **FIN-ACK**, and **RST-ACK**.

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

Description

rules defines a list of filtering rules on the eBPF Agents. When filtering is enabled, by default, flows that don't match any rule are rejected. To change the default, you can define a rule that accepts everything: **{ action: "Accept", cidr: "0.0.0.0/0" }**, and then refine with rejecting rules.

Type

array

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

Description

EBPFFlowFilterRule defines the desired eBPF agent configuration regarding flow filtering rule.

Type

object

Property	Type	Description
action	**string**	**action** defines the action to perform on the flows that match the filter. The available options are **Accept**, which is the default, and **Reject**.
cidr	**string**	**cidr** defines the IP CIDR to filter flows by. Examples: **10.10.10.0/24** or **100:100:100:100::/64**
destPorts	**integer-or-string**	**destPorts** optionally defines the destination ports to filter flows by. To filter a single port, set a single port as an integer value. For example, **destPorts: 80**. To filter a range of ports, use a "start-end" range in string format. For example, **destPorts: "80-100"**. To filter two ports, use a "port1,port2" in string format. For example, **ports: "80,100"**.
direction	**string**	**direction** optionally defines a direction to filter flows by. The available options are **Ingress** and **Egress**.
icmpCode	**integer**	**icmpCode**, for Internet Control Message Protocol (ICMP) traffic, optionally defines the ICMP code to filter flows by.
icmpType	**integer**	**icmpType**, for ICMP traffic, optionally defines the ICMP type to filter flows by.
peerCIDR	**string**	**peerCIDR** defines the Peer IP CIDR to filter flows by. Examples: **10.10.10.0/24** or **100:100:100:100::/64**

Property	Type	Description
peerIP	**string**	**peerIP** optionally defines the remote IP address to filter flows by. Example: **10.10.10.10**.
pktDrops	**boolean**	**pktDrops** optionally filters only flows containing packet drops.
ports	**integer-or-string**	**ports** optionally defines the ports to filter flows by. It is used both for source and destination ports. To filter a single port, set a single port as an integer value. For example, **ports: 80**. To filter a range of ports, use a "start-end" range in string format. For example, **ports: "80-100"**. To filter two ports, use a "port1,port2" in string format. For example, **ports: "80,100"**.
protocol	**string**	**protocol** optionally defines a protocol to filter flows by. The available options are **TCP**, **UDP**, **ICMP**, **ICMPv6**, and **SCTP**.
sampling	**integer**	**sampling** is the sampling interval for the matched packets, overriding the global sampling defined at **spec.agent.ebpf.sampling**.
sourcePorts	**integer-or-string**	**sourcePorts** optionally defines the source ports to filter flows by. To filter a single port, set a single port as an integer value. For example, **sourcePorts: 80**. To filter a range of ports, use a "start-end" range in string format. For example, **sourcePorts: "80-100"**. To filter two ports, use a "port1,port2" in string format. For example, **ports: "80,100"**.

Property	Type	Description
tcpFlags	**string**	**tcpFlags** optionally defines TCP flags to filter flows by. In addition to the standard flags (RFC-9293), you can also filter by one of the three following combinations: **SYN-ACK**, **FIN-ACK**, and **RST-ACK**.

15.1.12. .spec.agent.ebpf.metrics

Description

metrics defines the eBPF agent configuration regarding metrics.

Type

object

Property	Type	Description
disableAlerts	**array (string)**	**disableAlerts** is a list of alerts that should be disabled. Possible values are: **NetObservDroppedFlows**, which is triggered when the eBPF agent is missing packets or flows, such as when the BPF hashmap is busy or full, or the capacity limiter is being triggered.
enable	**boolean**	Set **enable** to **false** to disable eBPF agent metrics collection. It is enabled by default.
server	**object**	Metrics server endpoint configuration for the Prometheus scraper.

15.1.13. .spec.agent.ebpf.metrics.server

Description

Metrics server endpoint configuration for the Prometheus scraper.

Type

object

Property	Type	Description
port	**integer**	The metrics server HTTP port.
tls	**object**	TLS configuration.

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

Description

TLS configuration.

Type

object

Required

- **type**

Property	Type	Description
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the provided certificate. If set to **true**, the **providedCaFile** field is ignored.
provided	**object**	TLS configuration when **type** is set to **Provided**.
providedCaFile	**object**	Reference to the CA file when **type** is set to **Provided**.
type	**string**	Select the type of TLS configuration: - **Disabled** (default) to not configure TLS for the endpoint. - **Provided** to manually provide cert file and a key file. [Unsupported (*)]. - **Auto** to use OpenShift Container Platform auto generated certificate using annotations.

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

Description

TLS configuration when **type** is set to **Provided**.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

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

Description

Reference to the CA file when **type** is set to **Provided**.

Type

object

Property	Type	Description
file	**string**	File name within the config map or secret.
name	**string**	Name of the config map or secret containing the file.

Property	Type	Description
namespace	**string**	Namespace of the config map or secret containing the file. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the file reference: **configmap** or **secret**.

15.1.17. .spec.agent.ebpf.resources

Description

resources are the compute resources required by this container. For more information, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

Type

object

Property	Type	Description
limits	**integer-or-string**	Limits describes the maximum amount of compute resources allowed. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	**integer-or-string**	Requests describes the minimum amount of compute resources required. If Requests is omitted for a container, it defaults to Limits if that is explicitly specified, otherwise to an implementation-defined value. Requests cannot exceed Limits. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

15.1.18. .spec.consolePlugin

Description

consolePlugin defines the settings related to the OpenShift Container Platform Console plugin, when available.

Type
: **object**

Property	Type	Description
advanced	**object**	**advanced** allows setting some aspects of the internal configuration of the console plugin. This section is aimed mostly for debugging and fine-grained performance optimizations, such as **GOGC** and **GOMAXPROCS** environment variables. Set these values at your own risk.
autoscaler	**object**	**autoscaler** spec of a horizontal pod autoscaler to set up for the plugin Deployment. Refer to HorizontalPodAutoscaler documentation (autoscaling/v2).
enable	**boolean**	Enables the console plugin deployment.
imagePullPolicy	**string**	**imagePullPolicy** is the Kubernetes pull policy for the image defined above
logLevel	**string**	**logLevel** for the console plugin backend
portNaming	**object**	**portNaming** defines the configuration of the port-to-service name translation
quickFilters	**array**	**quickFilters** configures quick filter presets for the Console plugin
replicas	**integer**	**replicas** defines the number of replicas (pods) to start.
resources	**object**	**resources**, in terms of compute resources, required by this container. For more information, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

15.1.19. .spec.consolePlugin.advanced

Description

advanced allows setting some aspects of the internal configuration of the console plugin. This section is aimed mostly for debugging and fine-grained performance optimizations, such as **GOGC** and **GOMAXPROCS** environment variables. Set these values at your own risk.

Type

object

Property	Type	Description
args	**array (string)**	**args** allows passing custom arguments to underlying components. Useful for overriding some parameters, such as a URL or a configuration path, that should not be publicly exposed as part of the FlowCollector descriptor, as they are only useful in edge debug or support scenarios.
env	**object (string)**	**env** allows passing custom environment variables to underlying components. Useful for passing some very concrete performance-tuning options, such as **GOGC** and **GOMAXPROCS**, that should not be publicly exposed as part of the FlowCollector descriptor, as they are only useful in edge debug or support scenarios.
port	**integer**	**port** is the plugin service port. Do not use 9002, which is reserved for metrics.

Property	Type	Description
register	**boolean**	**register** allows, when set to **true**, to automatically register the provided console plugin with the OpenShift Container Platform Console operator. When set to **false**, you can still register it manually by editing console.operator.openshift.io/cluster with the following command: **oc patch console.operator.openshift.io cluster --type='json' -p '[{"op": "add", "path": "/spec/plugins/-", "value": "netobserv-plugin"}]'**
scheduling	**object**	**scheduling** controls how the pods are scheduled on nodes.

15.1.20. .spec.consolePlugin.advanced.scheduling

Description

scheduling controls how the pods are scheduled on nodes.

Type

object

Property	Type	Description
affinity	**object**	If specified, the pod's scheduling constraints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.
nodeSelector	**object (string)**	**nodeSelector** allows scheduling of pods only onto nodes that have each of the specified labels. For documentation, refer to https://kubernetes.io/docs/concepts/configuration/assign-pod-node/.

Property	Type	Description
priorityClassName	**string**	If specified, indicates the pod's priority. For documentation, refer to https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption. If not specified, default priority is used, or zero if there is no default.
tolerations	**array**	**tolerations** is a list of tolerations that allow the pod to schedule onto nodes with matching taints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

15.1.21. .spec.consolePlugin.advanced.scheduling.affinity

Description

If specified, the pod's scheduling constraints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

Type

object

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

Description

tolerations is a list of tolerations that allow the pod to schedule onto nodes with matching taints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

Type

array

15.1.23. .spec.consolePlugin.autoscaler

Description

autoscaler spec of a horizontal pod autoscaler to set up for the plugin Deployment. Refer to HorizontalPodAutoscaler documentation (autoscaling/v2).

Type

object

15.1.24. .spec.consolePlugin.portNaming

Description

portNaming defines the configuration of the port-to-service name translation

Type

object

Property	Type	Description
enable	**boolean**	Enable the console plugin port-to-service name translation
portNames	**object (string)**	**portNames** defines additional port names to use in the console, for example, **portNames: {"3100": "loki"}**.

15.1.25. .spec.consolePlugin.quickFilters

Description

quickFilters configures quick filter presets for the Console plugin

Type

array

15.1.26. .spec.consolePlugin.quickFilters[]

Description

QuickFilter defines preset configuration for Console's quick filters

Type

object

Required

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

Property	Type	Description
default	**boolean**	**default** defines whether this filter should be active by default or not
filter	**object (string)**	**filter** is a set of keys and values to be set when this filter is selected. Each key can relate to a list of values using a coma-separated string, for example, **filter: {"src_namespace": "namespace1,namespace2"}**.

Property	Type	Description
name	**string**	Name of the filter, that is displayed in the Console

15.1.27. .spec.consolePlugin.resources

Description

resources, in terms of compute resources, required by this container. For more information, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

Type

object

Property	Type	Description
limits	**integer-or-string**	Limits describes the maximum amount of compute resources allowed. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	**integer-or-string**	Requests describes the minimum amount of compute resources required. If Requests is omitted for a container, it defaults to Limits if that is explicitly specified, otherwise to an implementation-defined value. Requests cannot exceed Limits. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

15.1.28. .spec.exporters

Description

exporters defines additional optional exporters for custom consumption or storage.

Type

array

15.1.29. .spec.exporters[]

Description

FlowCollectorExporter defines an additional exporter to send enriched flows to.

Type

object

Required

- **type**

Property	Type	Description
ipfix	**object**	IPFIX configuration, such as the IP address and port to send enriched IPFIX flows to.
kafka	**object**	Kafka configuration, such as the address and topic, to send enriched flows to.
openTelemetry	**object**	OpenTelemetry configuration, such as the IP address and port to send enriched logs or metrics to.
type	**string**	**type** selects the type of exporters. The available options are **Kafka**, **IPFIX**, and **OpenTelemetry**.

15.1.30. .spec.exporters[].ipfix

Description

IPFIX configuration, such as the IP address and port to send enriched IPFIX flows to.

Type

object

Required

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

Property	Type	Description
targetHost	**string**	Address of the IPFIX external receiver.
targetPort	**integer**	Port for the IPFIX external receiver.
transport	**string**	Transport protocol (**TCP** or **UDP**) to be used for the IPFIX connection, defaults to **TCP**.

15.1.31. .spec.exporters[].kafka

Description

Kafka configuration, such as the address and topic, to send enriched flows to.

Type

object

Required

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

Property	Type	Description
address	**string**	Address of the Kafka server
sasl	**object**	SASL authentication configuration. [Unsupported (*)].
tls	**object**	TLS client configuration. When using TLS, verify that the address matches the Kafka port used for TLS, generally 9093.
topic	**string**	Kafka topic to use. It must exist. Network Observability does not create it.

15.1.32. .spec.exporters[].kafka.sasl

Description

SASL authentication configuration. [Unsupported (*)].

Type

object

Property	Type	Description
clientIDReference	**object**	Reference to the secret or config map containing the client ID
clientSecretReference	**object**	Reference to the secret or config map containing the client secret
type	**string**	Type of SASL authentication to use, or **Disabled** if SASL is not used

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

Description

Reference to the secret or config map containing the client ID

Type

object

Property	Type	Description
file	**string**	File name within the config map or secret.
name	**string**	Name of the config map or secret containing the file.
namespace	**string**	Namespace of the config map or secret containing the file. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the file reference: **configmap** or **secret**.

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

Description

Reference to the secret or config map containing the client secret

Type

object

Property	Type	Description
file	**string**	File name within the config map or secret.
name	**string**	Name of the config map or secret containing the file.

Property	Type	Description
namespace	**string**	Namespace of the config map or secret containing the file. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the file reference: **configmap** or **secret**.

15.1.35. .spec.exporters[].kafka.tls

Description

TLS client configuration. When using TLS, verify that the address matches the Kafka port used for TLS, generally 9093.

Type

object

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

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

Description

caCert defines the reference of the certificate for the Certificate Authority.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

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

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.

Property	Type	Description
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.38. .spec.exporters[].openTelemetry

Description

OpenTelemetry configuration, such as the IP address and port to send enriched logs or metrics to.

Type

object

Required

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

Property	Type	Description
fieldsMapping	**array**	Custom fields mapping to an OpenTelemetry conformant format. By default, Network Observability format proposal is used: https://github.com/rhobs/observability-data-model/blob/main/network-observability.md#format-proposal . As there is currently no accepted standard for L3 or L4 enriched network logs, you can freely override it with your own.
headers	**object (string)**	Headers to add to messages (optional)

Property	Type	Description
logs	**object**	OpenTelemetry configuration for logs.
metrics	**object**	OpenTelemetry configuration for metrics.
protocol	**string**	Protocol of the OpenTelemetry connection. The available options are **http** and **grpc**.
targetHost	**string**	Address of the OpenTelemetry receiver.
targetPort	**integer**	Port for the OpenTelemetry receiver.
tls	**object**	TLS client configuration.

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

Description

Custom fields mapping to an OpenTelemetry conformant format. By default, Network Observability format proposal is used: https://github.com/rhobs/observability-data-model/blob/main/network-observability.md#format-proposal . As there is currently no accepted standard for L3 or L4 enriched network logs, you can freely override it with your own.

Type

array

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

Description

Type

object

Property	Type	Description
input	**string**	
multiplier	**integer**	
output	**string**	

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

Description

OpenTelemetry configuration for logs.

Type

object

Property	Type	Description
enable	**boolean**	Set **enable** to **true** to send logs to an OpenTelemetry receiver.

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

Description

OpenTelemetry configuration for metrics.

Type

object

Property	Type	Description
enable	**boolean**	Set **enable** to **true** to send metrics to an OpenTelemetry receiver.
pushTimeInterval	**string**	Specify how often metrics are sent to a collector.

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

Description

TLS client configuration.

Type

object

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.

Property	Type	Description
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

15.1.44. .spec.exporters[].openTelemetry.tls.caCert

Description

caCert defines the reference of the certificate for the Certificate Authority.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

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

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.46. .spec.kafka

Description

Kafka configuration, allowing to use Kafka as a broker as part of the flow collection pipeline. Available when the **spec.deploymentModel** is **Kafka**.

Type

object

Required

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

Property	Type	Description
address	**string**	Address of the Kafka server

Property	Type	Description
sasl	**object**	SASL authentication configuration. [Unsupported (*)].
tls	**object**	TLS client configuration. When using TLS, verify that the address matches the Kafka port used for TLS, generally 9093.
topic	**string**	Kafka topic to use. It must exist. Network Observability does not create it.

15.1.47. .spec.kafka.sasl

Description

SASL authentication configuration. [Unsupported (*)].

Type

object

Property	Type	Description
clientIDReference	**object**	Reference to the secret or config map containing the client ID
clientSecretReference	**object**	Reference to the secret or config map containing the client secret
type	**string**	Type of SASL authentication to use, or **Disabled** if SASL is not used

15.1.48. .spec.kafka.sasl.clientIDReference

Description

Reference to the secret or config map containing the client ID

Type

object

Property	Type	Description
file	**string**	File name within the config map or secret.

Property	Type	Description
name	**string**	Name of the config map or secret containing the file.
namespace	**string**	Namespace of the config map or secret containing the file. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the file reference: **configmap** or **secret**.

15.1.49. .spec.kafka.sasl.clientSecretReference

Description

Reference to the secret or config map containing the client secret

Type

object

Property	Type	Description
file	**string**	File name within the config map or secret.
name	**string**	Name of the config map or secret containing the file.
namespace	**string**	Namespace of the config map or secret containing the file. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the file reference: **configmap** or **secret**.

15.1.50. .spec.kafka.tls

Description

TLS client configuration. When using TLS, verify that the address matches the Kafka port used for TLS, generally 9093.

Type

object

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

15.1.51. .spec.kafka.tls.caCert

Description

caCert defines the reference of the certificate for the Certificate Authority.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.

Property	Type	Description
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.52. .spec.kafka.tls.userCert

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.

Property	Type	Description
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.53. .spec.loki

Description

loki, the flow store, client settings.

Type

object

Required

- **mode**

Property	Type	Description
advanced	**object**	**advanced** allows setting some aspects of the internal configuration of the Loki clients. This section is aimed mostly for debugging and fine-grained performance optimizations.
enable	**boolean**	Set **enable** to **true** to store flows in Loki. The Console plugin can use either Loki or Prometheus as a data source for metrics (see also **spec.prometheus.querier**), or both. Not all queries are transposable from Loki to Prometheus. Hence, if Loki is disabled, some features of the plugin are disabled as well, such as getting per-pod information or viewing raw flows. If both Prometheus and Loki are enabled, Prometheus takes precedence and Loki is used as a fallback for queries that Prometheus cannot handle. If they are both disabled, the Console plugin is not deployed.
lokiStack	**object**	Loki configuration for **LokiStack** mode. This is useful for an easy Loki Operator configuration. It is ignored for other modes.

Property	Type	Description
manual	**object**	Loki configuration for **Manual** mode. This is the most flexible configuration. It is ignored for other modes.
microservices	**object**	Loki configuration for **Microservices** mode. Use this option when Loki is installed using the microservices deployment mode (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode). It is ignored for other modes.
mode	**string**	**mode** must be set according to the installation mode of Loki: - Use **LokiStack** when Loki is managed using the Loki Operator - Use **Monolithic** when Loki is installed as a monolithic workload - Use **Microservices** when Loki is installed as microservices, but without Loki Operator - Use **Manual** if none of the options above match your setup
monolithic	**object**	Loki configuration for **Monolithic** mode. Use this option when Loki is installed using the monolithic deployment mode (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode). It is ignored for other modes.
readTimeout	**string**	**readTimeout** is the maximum console plugin loki query total time limit. A timeout of zero means no timeout.
writeBatchSize	**integer**	**writeBatchSize** is the maximum batch size (in bytes) of Loki logs to accumulate before sending.

Property	Type	Description
writeBatchWait	**string**	**writeBatchWait** is the maximum time to wait before sending a Loki batch.
writeTimeout	**string**	**writeTimeout** is the maximum Loki time connection / request limit. A timeout of zero means no timeout.

15.1.54. .spec.loki.advanced

Description

advanced allows setting some aspects of the internal configuration of the Loki clients. This section is aimed mostly for debugging and fine-grained performance optimizations.

Type

object

Property	Type	Description
excludeLabels	**array (string)**	**excludeLabels** is a list of fields to be excluded from the list of Loki labels. [Unsupported (*)].
staticLabels	**object (string)**	**staticLabels** is a map of common labels to set on each flow in Loki storage.
writeMaxBackoff	**string**	**writeMaxBackoff** is the maximum backoff time for Loki client connection between retries.
writeMaxRetries	**integer**	**writeMaxRetries** is the maximum number of retries for Loki client connections.
writeMinBackoff	**string**	**writeMinBackoff** is the initial backoff time for Loki client connection between retries.

15.1.55. .spec.loki.lokiStack

Description

Loki configuration for **LokiStack** mode. This is useful for an easy Loki Operator configuration. It is ignored for other modes.

Type

object

Required

- **name**

Property	Type	Description
name	**string**	Name of an existing LokiStack resource to use.
namespace	**string**	Namespace where this **LokiStack** resource is located. If omitted, it is assumed to be the same as **spec.namespace**.

15.1.56. .spec.loki.manual

Description

Loki configuration for **Manual** mode. This is the most flexible configuration. It is ignored for other modes.

Type

object

Property	Type	Description
authToken	**string**	**authToken** describes the way to get a token to authenticate to Loki. - **Disabled** does not send any token with the request. - **Forward** forwards the user token for authorization. - **Host** [deprecated (*)] - uses the local pod service account to authenticate to Loki. When using the Loki Operator, this must be set to **Forward**.

Property	Type	Description
ingesterUrl	**string**	**ingesterUrl** is the address of an existing Loki ingester service to push the flows to. When using the Loki Operator, set it to the Loki gateway service with the **network** tenant set in path, for example https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network.
querierUrl	**string**	**querierUrl** specifies the address of the Loki querier service. When using the Loki Operator, set it to the Loki gateway service with the **network** tenant set in path, for example https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network.
statusTls	**object**	TLS client configuration for Loki status URL.
statusUrl	**string**	**statusUrl** specifies the address of the Loki **/ready**, **/metrics** and **/config** endpoints, in case it is different from the Loki querier URL. If empty, the **querierUrl** value is used. This is useful to show error messages and some context in the frontend. When using the Loki Operator, set it to the Loki HTTP query frontend service, for example https://loki-query-frontend-http.netobserv.svc:3100/. **statusTLS** configuration is used when **statusUrl** is set.
tenantID	**string**	**tenantID** is the Loki **X-Scope-OrgID** that identifies the tenant for each request. When using the Loki Operator, set it to **network**, which corresponds to a special tenant mode.
tls	**object**	TLS client configuration for Loki URL.

15.1.57. .spec.loki.manual.statusTls

Description

TLS client configuration for Loki status URL.

Type

object

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

15.1.58. .spec.loki.manual.statusTls.caCert

Description

caCert defines the reference of the certificate for the Certificate Authority.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.

Property	Type	Description
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.59. .spec.loki.manual.statusTls.userCert

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.

Property	Type	Description
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.60. .spec.loki.manual.tls

Description

TLS client configuration for Loki URL.

Type

object

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

15.1.61. .spec.loki.manual.tls.caCert

Description

caCert defines the reference of the certificate for the Certificate Authority.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.

Property	Type	Description
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.62. .spec.loki.manual.tls.userCert

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.

Property	Type	Description
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.63. .spec.loki.microservices

Description

Loki configuration for **Microservices** mode. Use this option when Loki is installed using the microservices deployment mode (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode). It is ignored for other modes.

Type

object

Property	Type	Description
ingesterUrl	**string**	**ingesterUrl** is the address of an existing Loki ingester service to push the flows to.
querierUrl	**string**	**querierURL** specifies the address of the Loki querier service.
tenantID	**string**	**tenantID** is the Loki **X-Scope-OrgID** header that identifies the tenant for each request.
tls	**object**	TLS client configuration for Loki URL.

15.1.64. .spec.loki.microservices.tls

Description

TLS client configuration for Loki URL.

Type
: **object**

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

15.1.65. .spec.loki.microservices.tls.caCert

Description
: **caCert** defines the reference of the certificate for the Certificate Authority.

Type
: **object**

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.

Property	Type	Description
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.66. .spec.loki.microservices.tls.userCert

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.

Property	Type	Description
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.67. .spec.loki.monolithic

Description

Loki configuration for **Monolithic** mode. Use this option when Loki is installed using the monolithic deployment mode (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode). It is ignored for other modes.

Type

object

Property	Type	Description
tenantID	**string**	**tenantID** is the Loki **X-Scope-OrgID** header that identifies the tenant for each request.
tls	**object**	TLS client configuration for Loki URL.
url	**string**	**url** is the unique address of an existing Loki service that points to both the ingester and the querier.

15.1.68. .spec.loki.monolithic.tls

Description

TLS client configuration for Loki URL.

Type

object

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.

Property	Type	Description
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

15.1.69. .spec.loki.monolithic.tls.caCert

Description

caCert defines the reference of the certificate for the Certificate Authority.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.70. .spec.loki.monolithic.tls.userCert

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.71. .spec.networkPolicy

Description

networkPolicy defines network policy settings for Network Observability components isolation.

Type

object

Property	Type	Description
additionalNamespaces	**array (string)**	**additionalNamespaces** contains additional namespaces allowed to connect to the Network Observability namespace. It provides flexibility in the network policy configuration, but if you need a more specific configuration, you can disable it and install your own instead.

Property	Type	Description
enable	**boolean**	Deploys network policies on the namespaces used by Network Observability (main and privileged). These network policies better isolate the Network Observability components to prevent undesired connections from and to them. This option is enabled by default when using with OVNKubernetes, and disabled otherwise (it has not been tested with other CNIs). When disabled, you can manually create the network policies for the Network Observability components.

15.1.72. .spec.processor

Description

processor defines the settings of the component that receives the flows from the agent, enriches them, generates metrics, and forwards them to the Loki persistence layer and/or any available exporter.

Type

object

Property	Type	Description
addZone	**boolean**	**addZone** allows availability zone awareness by labelling flows with their source and destination zones. This feature requires the "topology.kubernetes.io/zone" label to be set on nodes.
advanced	**object**	**advanced** allows setting some aspects of the internal configuration of the flow processor. This section is aimed mostly for debugging and fine-grained performance optimizations, such as **GOGC** and **GOMAXPROCS** environment variables. Set these values at your own risk.

Property	Type	Description
clusterName	**string**	**clusterName** is the name of the cluster to appear in the flows data. This is useful in a multi-cluster context. When using OpenShift Container Platform, leave empty to make it automatically determined.
deduper	**object**	**deduper** allows you to sample or drop flows identified as duplicates, in order to save on resource usage.
filters	**array**	**filters** lets you define custom filters to limit the amount of generated flows. These filters provide more flexibility than the eBPF Agent filters (in **spec.agent.ebpf.flowFilter**), such as allowing to filter by Kubernetes namespace, but with a lesser improvement in performance.
imagePullPolicy	**string**	**imagePullPolicy** is the Kubernetes pull policy for the image defined above
kafkaConsumerAutoscaler	**object**	**kafkaConsumerAutoscaler** is the spec of a horizontal pod autoscaler to set up for **flowlogs-pipeline-transformer**, which consumes Kafka messages. This setting is ignored when Kafka is disabled. Refer to HorizontalPodAutoscaler documentation (autoscaling/v2).
kafkaConsumerBatchSize	**integer**	**kafkaConsumerBatchSize** indicates to the broker the maximum batch size, in bytes, that the consumer accepts. Ignored when not using Kafka. Default: 10MB.
kafkaConsumerQueueCapacity	**integer**	**kafkaConsumerQueueCapacity** defines the capacity of the internal message queue used in the Kafka consumer client. Ignored when not using Kafka.

Property	Type	Description
kafkaConsumerReplicas	**integer**	**kafkaConsumerReplicas** defines the number of replicas (pods) to start for **flowlogs-pipeline-transformer**, which consumes Kafka messages. This setting is ignored when Kafka is disabled.
logLevel	**string**	**logLevel** of the processor runtime
logTypes	**string**	**logTypes** defines the desired record types to generate. Possible values are: - **Flows** to export regular network flows. This is the default. - **Conversations** to generate events for started conversations, ended conversations as well as periodic "tick" updates. Note that in this mode, Prometheus metrics are not accurate on long-standing conversations. - **EndedConversations** to generate only ended conversations events. Note that in this mode, Prometheus metrics are not accurate on long-standing conversations. - **All** to generate both network flows and all conversations events. It is not recommended due to the impact on resources footprint.
metrics	**object**	**Metrics** define the processor configuration regarding metrics

Property	Type	Description
multiClusterDeployment	**boolean**	Set **multiClusterDeployment** to **true** to enable multi clusters feature. This adds **clusterName** label to flows data
resources	**object**	**resources** are the compute resources required by this container. For more information, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
subnetLabels	**object**	**subnetLabels** allows to define custom labels on subnets and IPs or to enable automatic labelling of recognized subnets in OpenShift Container Platform, which is used to identify cluster external traffic. When a subnet matches the source or destination IP of a flow, a corresponding field is added: **SrcSubnetLabel** or **DstSubnetLabel**.

15.1.73. .spec.processor.advanced

Description

advanced allows setting some aspects of the internal configuration of the flow processor. This section is aimed mostly for debugging and fine-grained performance optimizations, such as **GOGC** and **GOMAXPROCS** environment variables. Set these values at your own risk.

Type

object

Property	Type	Description
conversationEndTimeout	**string**	**conversationEndTimeout** is the time to wait after a network flow is received, to consider the conversation ended. This delay is ignored when a FIN packet is collected for TCP flows (see **conversationTerminatingTimeout** instead).

Property	Type	Description
conversationHeartbeatInterval	**string**	**conversationHeartbeatInterval** is the time to wait between "tick" events of a conversation
conversationTerminatingTimeout	**string**	**conversationTerminatingTimeout** is the time to wait from detected FIN flag to end a conversation. Only relevant for TCP flows.
dropUnusedFields	**boolean**	**dropUnusedFields** [deprecated (*)] this setting is not used anymore.
enableKubeProbes	**boolean**	**enableKubeProbes** is a flag to enable or disable Kubernetes liveness and readiness probes
env	**object (string)**	**env** allows passing custom environment variables to underlying components. Useful for passing some very concrete performance-tuning options, such as **GOGC** and **GOMAXPROCS**, that should not be publicly exposed as part of the FlowCollector descriptor, as they are only useful in edge debug or support scenarios.
healthPort	**integer**	**healthPort** is a collector HTTP port in the Pod that exposes the health check API
port	**integer**	Port of the flow collector (host port). By convention, some values are forbidden. It must be greater than 1024 and different from 4500, 4789 and 6081.
profilePort	**integer**	**profilePort** allows setting up a Go pprof profiler listening to this port
scheduling	**object**	scheduling controls how the pods are scheduled on nodes.

Property	Type	Description
secondaryNetworks	**array**	Defines secondary networks to be checked for resources identification. To guarantee a correct identification, indexed values must form an unique identifier across the cluster. If the same index is used by several resources, those resources might be incorrectly labeled.

15.1.74. .spec.processor.advanced.scheduling

Description

scheduling controls how the pods are scheduled on nodes.

Type

object

Property	Type	Description
affinity	**object**	If specified, the pod's scheduling constraints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.
nodeSelector	**object (string)**	**nodeSelector** allows scheduling of pods only onto nodes that have each of the specified labels. For documentation, refer to https://kubernetes.io/docs/concepts/configuration/assign-pod-node/.
priorityClassName	**string**	If specified, indicates the pod's priority. For documentation, refer to https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption. If not specified, default priority is used, or zero if there is no default.

Property	Type	Description
tolerations	**array**	**tolerations** is a list of tolerations that allow the pod to schedule onto nodes with matching taints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

15.1.75. .spec.processor.advanced.scheduling.affinity

Description

If specified, the pod's scheduling constraints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

Type

object

15.1.76. .spec.processor.advanced.scheduling.tolerations

Description

tolerations is a list of tolerations that allow the pod to schedule onto nodes with matching taints. For documentation, refer to https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling.

Type

array

15.1.77. .spec.processor.advanced.secondaryNetworks

Description

Defines secondary networks to be checked for resources identification. To guarantee a correct identification, indexed values must form an unique identifier across the cluster. If the same index is used by several resources, those resources might be incorrectly labeled.

Type

array

15.1.78. .spec.processor.advanced.secondaryNetworks[]

Description

Type

object

Required

- **index**

- **name**

Property	Type	Description
index	**array (string)**	**index** is a list of fields to use for indexing the pods. They should form a unique Pod identifier across the cluster. Can be any of: **MAC**, **IP**, **Interface**. Fields absent from the 'k8s.v1.cni.cncf.io/network-status' annotation must not be added to the index.
name	**string**	**name** should match the network name as visible in the pods annotation 'k8s.v1.cni.cncf.io/network-status'.

15.1.79. .spec.processor.deduper

Description

deduper allows you to sample or drop flows identified as duplicates, in order to save on resource usage.

Type

object

Property	Type	Description

Property	Type	Description
mode	**string**	Set the Processor de-duplication mode. It comes in addition to the Agent-based deduplication, since the Agent cannot de-duplicate same flows reported from different nodes. - Use **Drop** to drop every flow considered as duplicates, allowing saving more on resource usage but potentially losing some information such as the network interfaces used from peer, or network events. - Use **Sample** to randomly keep only one flow on 50, which is the default, among the ones considered as duplicates. This is a compromise between dropping every duplicate or keeping every duplicate. This sampling action comes in addition to the Agent-based sampling. If both Agent and Processor sampling values are **50**, the combined sampling is 1:2500. - Use **Disabled** to turn off Processor-based de-duplication.
sampling	**integer**	**sampling** is the sampling interval when deduper **mode** is **Sample**. For example, a value of **50** means that 1 flow in 50 is sampled.

15.1.80. .spec.processor.filters

Description

filters lets you define custom filters to limit the amount of generated flows. These filters provide more flexibility than the eBPF Agent filters (in **spec.agent.ebpf.flowFilter**), such as allowing to filter by Kubernetes namespace, but with a lesser improvement in performance.

Type

array

15.1.81. .spec.processor.filters[]

Description

FLPFilterSet defines the desired configuration for FLP-based filtering satisfying all conditions.

Type

object

Property	Type	Description
outputTarget	**string**	If specified, these filters target a single output: **Loki**, **Metrics** or **Exporters**. By default, all outputs are targeted.
query	**string**	A query that selects the network flows to keep. More information about this query language in https://github.com/netobserv/flowlogs-pipeline/blob/main/docs/filtering.md.
sampling	**integer**	**sampling** is an optional sampling interval to apply to this filter. For example, a value of **50** means that 1 matching flow in 50 is sampled.

15.1.82. .spec.processor.kafkaConsumerAutoscaler

Description

kafkaConsumerAutoscaler is the spec of a horizontal pod autoscaler to set up for **flowlogs-pipeline-transformer**, which consumes Kafka messages. This setting is ignored when Kafka is disabled. Refer to HorizontalPodAutoscaler documentation (autoscaling/v2).

Type

object

15.1.83. .spec.processor.metrics

Description

Metrics define the processor configuration regarding metrics

Type

object

Property	Type	Description

Property	Type	Description
alerts	**array**	**alerts** is a list of alerts to be created for Prometheus AlertManager, organized by templates and variants [Unsupported (*)]. This is currently an experimental feature behind a feature gate. To enable, edit **spec.processor.advanced.env** by adding **EXPERIMENTAL_ALERTS_HEALTH** set to **true**. More information on alerts: https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md
disableAlerts	**array (string)**	**disableAlerts** is a list of alert groups that should be disabled from the default set of alerts. Possible values are: **NetObservNoFlows**, **NetObservLokiError**, **PacketDropsByKernel**, **PacketDropsByDevice**, **IPsecErrors**, **NetpolDenied**, **LatencyHighTrend**, **DNSErrors**, **ExternalEgressHighTrend**, **ExternalIngressHighTrend**, **CrossAZ**. More information on alerts: https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md

Property	Type	Description
includeList	**array (string)**	**includeList** is a list of metric names to specify which ones to generate. The names correspond to the names in Prometheus without the prefix. For example, **namespace_egress_packets_total** shows up as **netobserv_namespace_egress_packets_total** in Prometheus. Note that the more metrics you add, the bigger is the impact on Prometheus workload resources. Metrics enabled by default are: **namespace_flows_total**, **node_ingress_bytes_total**, **node_egress_bytes_total**, **workload_ingress_bytes_total**, **workload_egress_bytes_total**, **namespace_drop_packets_total** (when **PacketDrop** feature is enabled), **namespace_rtt_seconds** (when **FlowRTT** feature is enabled), **namespace_dns_latency_seconds** (when **DNSTracking** feature is enabled), **namespace_network_policy_events_total** (when **NetworkEvents** feature is enabled). More information, with full list of available metrics: https://github.com/netobserv/network-observability-operator/blob/main/docs/Metrics.md
server	**object**	Metrics server endpoint configuration for Prometheus scraper

15.1.84. .spec.processor.metrics.alerts

Description

alerts is a list of alerts to be created for Prometheus AlertManager, organized by templates and variants [Unsupported (*)]. This is currently an experimental feature behind a feature gate. To enable, edit **spec.processor.advanced.env** by adding **EXPERIMENTAL_ALERTS_HEALTH** set to

true. More information on alerts: https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md

Type

array

15.1.85. .spec.processor.metrics.alerts[]

Description

Type

object

Required

- **template**
- **variants**

Property	Type	Description
template	**string**	Alert template name. Possible values are: **PacketDropsByKernel**, **PacketDropsByDevice**, **IPsecErrors**, **NetpolDenied**, **LatencyHighTrend**, **DNSErrors**, **ExternalEgressHighTrend**, **ExternalIngressHighTrend**, **CrossAZ**. More information on alerts: https://github.com/netobserv/network-observability-operator/blob/main/docs/Alerts.md
variants	**array**	A list of variants for this template

15.1.86. .spec.processor.metrics.alerts[].variants

Description

A list of variants for this template

Type

array

15.1.87. .spec.processor.metrics.alerts[].variants[]

Description

Type

object

Required

- **thresholds**

Property	Type	Description
groupBy	**string**	Optional grouping criteria, possible values are: **Node**, **Namespace**, **Workload**.
lowVolumeThreshold	**string**	The low volume threshold allows to ignore metrics with a too low volume of traffic, in order to improve signal-to-noise. It is provided as an absolute rate (bytes per second or packets per second, depending on the context). When provided, it must be parsable as a float.
thresholds	**object**	Thresholds of the alert per severity. They are expressed as a percentage of errors above which the alert is triggered. They must be parsable as floats.
trendDuration	**string**	For trending alerts, the duration interval for baseline comparison. For example, "2h" means comparing against a 2-hours average. Defaults to 2h.
trendOffset	**string**	For trending alerts, the time offset for baseline comparison. For example, "1d" means comparing against yesterday. Defaults to 1d.

15.1.88. .spec.processor.metrics.alerts[].variants[].thresholds

Description

Thresholds of the alert per severity. They are expressed as a percentage of errors above which the alert is triggered. They must be parsable as floats.

Type

object

Property	Type	Description

Property	Type	Description
critical	**string**	Threshold for severity **critical**. Leave empty to not generate a Critical alert.
info	**string**	Threshold for severity **info**. Leave empty to not generate an Info alert.
warning	**string**	Threshold for severity **warning**. Leave empty to not generate a Warning alert.

15.1.89. .spec.processor.metrics.server

Description

Metrics server endpoint configuration for Prometheus scraper

Type

object

Property	Type	Description
port	**integer**	The metrics server HTTP port.
tls	**object**	TLS configuration.

15.1.90. .spec.processor.metrics.server.tls

Description

TLS configuration.

Type

object

Required

- **type**

Property	Type	Description
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the provided certificate. If set to **true**, the **providedCaFile** field is ignored.

Property	Type	Description
provided	**object**	TLS configuration when **type** is set to **Provided**.
providedCaFile	**object**	Reference to the CA file when **type** is set to **Provided**.
type	**string**	Select the type of TLS configuration: - **Disabled** (default) to not configure TLS for the endpoint. - **Provided** to manually provide cert file and a key file. [Unsupported (*)]. - **Auto** to use OpenShift Container Platform auto generated certificate using annotations.

15.1.91. .spec.processor.metrics.server.tls.provided

Description

TLS configuration when **type** is set to **Provided**.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.

Property	Type	Description
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.92. .spec.processor.metrics.server.tls.providedCaFile

Description

Reference to the CA file when **type** is set to **Provided**.

Type

object

Property	Type	Description
file	**string**	File name within the config map or secret.
name	**string**	Name of the config map or secret containing the file.
namespace	**string**	Namespace of the config map or secret containing the file. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the file reference: **configmap** or **secret**.

15.1.93. .spec.processor.resources

Description

resources are the compute resources required by this container. For more information, see https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

Type
object

Property	Type	Description
limits	**integer-or-string**	Limits describes the maximum amount of compute resources allowed. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	**integer-or-string**	Requests describes the minimum amount of compute resources required. If Requests is omitted for a container, it defaults to Limits if that is explicitly specified, otherwise to an implementation-defined value. Requests cannot exceed Limits. More info: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

15.1.94. .spec.processor.subnetLabels

Description
subnetLabels allows to define custom labels on subnets and IPs or to enable automatic labelling of recognized subnets in OpenShift Container Platform, which is used to identify cluster external traffic. When a subnet matches the source or destination IP of a flow, a corresponding field is added: **SrcSubnetLabel** or **DstSubnetLabel**.

Type
object

Property	Type	Description
customLabels	**array**	**customLabels** allows to customize subnets and IPs labelling, such as to identify cluster-external workloads or web services. If you enable **openShiftAutoDetect**, **customLabels** can override the detected subnets in case they overlap.

Property	Type	Description
openShiftAutoDetect	**boolean**	**openShiftAutoDetect** allows, when set to **true**, to detect automatically the machines, pods and services subnets based on the OpenShift Container Platform install configuration and the Cluster Network Operator configuration. Indirectly, this is a way to accurately detect external traffic: flows that are not labeled for those subnets are external to the cluster. Enabled by default on OpenShift Container Platform.

15.1.95. .spec.processor.subnetLabels.customLabels

Description

customLabels allows to customize subnets and IPs labelling, such as to identify cluster-external workloads or web services. If you enable **openShiftAutoDetect**, **customLabels** can override the detected subnets in case they overlap.

Type

array

15.1.96. .spec.processor.subnetLabels.customLabels[]

Description

SubnetLabel allows to label subnets and IPs, such as to identify cluster-external workloads or web services.

Type

object

Required

- **cidrs**
- **name**

Property	Type	Description
cidrs	**array (string)**	List of CIDRs, such as **["1.2.3.4/32"]**.
name	**string**	Label name, used to flag matching flows.

15.1.97. .spec.prometheus

Description

prometheus defines Prometheus settings, such as querier configuration used to fetch metrics from the Console plugin.

Type

object

Property	Type	Description
querier	**object**	Prometheus querying configuration, such as client settings, used in the Console plugin.

15.1.98. .spec.prometheus.querier

Description

Prometheus querying configuration, such as client settings, used in the Console plugin.

Type

object

Required

- **mode**

Property	Type	Description

Property	Type	Description
enable	**boolean**	When **enable** is **true**, the Console plugin queries flow metrics from Prometheus instead of Loki whenever possible. It is enbaled by default: set it to **false** to disable this feature. The Console plugin can use either Loki or Prometheus as a data source for metrics (see also **spec.loki**), or both. Not all queries are transposable from Loki to Prometheus. Hence, if Loki is disabled, some features of the plugin are disabled as well, such as getting per-pod information or viewing raw flows. If both Prometheus and Loki are enabled, Prometheus takes precedence and Loki is used as a fallback for queries that Prometheus cannot handle. If they are both disabled, the Console plugin is not deployed.
manual	**object**	Prometheus configuration for **Manual** mode.
mode	**string**	**mode** must be set according to the type of Prometheus installation that stores Network Observability metrics: - Use **Auto** to try configuring automatically. In OpenShift Container Platform, it uses the Thanos querier from OpenShift Container Platform Cluster Monitoring - Use **Manual** for a manual setup
timeout	**string**	**timeout** is the read timeout for console plugin queries to Prometheus. A timeout of zero means no timeout.

15.1.99. .spec.prometheus.querier.manual

Description

Prometheus configuration for **Manual** mode.

Type

object

Property	Type	Description
forwardUserToken	**boolean**	Set **true** to forward logged in user token in queries to Prometheus
tls	**object**	TLS client configuration for Prometheus URL.
url	**string**	**url** is the address of an existing Prometheus service to use for querying metrics.

15.1.100. .spec.prometheus.querier.manual.tls

Description

TLS client configuration for Prometheus URL.

Type

object

Property	Type	Description
caCert	**object**	**caCert** defines the reference of the certificate for the Certificate Authority.
enable	**boolean**	Enable TLS
insecureSkipVerify	**boolean**	**insecureSkipVerify** allows skipping client-side verification of the server certificate. If set to **true**, the **caCert** field is ignored.
userCert	**object**	**userCert** defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

15.1.101. .spec.prometheus.querier.manual.tls.caCert

Description

caCert defines the reference of the certificate for the Certificate Authority.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

15.1.102. .spec.prometheus.querier.manual.tls.userCert

Description

userCert defines the user certificate reference and is used for mTLS. When you use one-way TLS, you can ignore this property.

Type

object

Property	Type	Description
certFile	**string**	**certFile** defines the path to the certificate file name within the config map or secret.
certKey	**string**	**certKey** defines the path to the certificate private key file name within the config map or secret. Omit when the key is not necessary.

Property	Type	Description
name	**string**	Name of the config map or secret containing certificates.
namespace	**string**	Namespace of the config map or secret containing certificates. If omitted, the default is to use the same namespace as where Network Observability is deployed. If the namespace is different, the config map or the secret is copied so that it can be mounted as required.
type	**string**	Type for the certificate reference: **configmap** or **secret**.

CHAPTER 16. FLOWMETRIC CONFIGURATION PARAMETERS

The **FlowMetric** API is used to generate custom observability metrics from collected network flow logs.

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

Description

FlowMetric is the API allowing to create custom metrics from the collected flow logs.

Type

object

Property	Type	Description
apiVersion	**string**	APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and might reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
kind	**string**	Kind is a string value representing the REST resource this object represents. Servers might infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds
metadata	**object**	Standard object’s metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

Property	Type	Description
spec	**object**	FlowMetricSpec defines the desired state of FlowMetric The provided API allows you to customize these metrics according to your needs. When adding new metrics or modifying existing labels, you must carefully monitor the memory usage of Prometheus workloads as this could potentially have a high impact. Cf https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric To check the cardinality of all Network Observability metrics, run as **promql**: **count({*name*=~"netobserv.*"}) by (*name*)**.

16.1.1. .metadata

Description

Standard object's metadata. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

Type

object

16.1.2. .spec

Description

FlowMetricSpec defines the desired state of FlowMetric The provided API allows you to customize these metrics according to your needs.
When adding new metrics or modifying existing labels, you must carefully monitor the memory usage of Prometheus workloads as this could potentially have a high impact. Cf https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric

To check the cardinality of all Network Observability metrics, run as **promql**: **count({*name*=~"netobserv.*"}) by (*name*)**.

Type

object

Required

- **type**

Property	Type	Description
buckets	**array (string)**	A list of buckets to use when **type** is "Histogram". The list must be parsable as floats. When not set, Prometheus default buckets are used.
charts	**array**	Charts configuration, for the OpenShift Container Platform Console in the administrator view, Dashboards menu.
direction	**string**	Filter for ingress, egress or any direction flows. When set to **Ingress**, it is equivalent to adding the regular expression filter on **FlowDirection**: **0|2**. When set to **Egress**, it is equivalent to adding the regular expression filter on **FlowDirection**: **1|2**.
divider	**string**	When nonzero, scale factor (divider) of the value. Metric value = Flow value / Divider.
filters	**array**	**filters** is a list of fields and values used to restrict which flows are taken into account. Refer to the documentation for the list of available fields: https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference.
flatten	**array (string)**	**flatten** is a list of array-type fields that must be flattened, such as Interfaces or NetworkEvents. Flattened fields generate one metric per item in that field. For instance, when flattening **Interfaces** on a bytes counter, a flow having Interfaces [br-ex, ens5] increases one counter for **br-ex** and another for **ens5**.

Property	Type	Description
labels	**array (string)**	**labels** is a list of fields that should be used as Prometheus labels, also known as dimensions (for example: **SrcK8S_Namespace**). From choosing labels results the level of granularity of this metric, and the available aggregations at query time. It must be done carefully as it impacts the metric cardinality (cf https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric). In general, avoid setting very high cardinality labels such as IP or MAC addresses. "SrcK8S_OwnerName" or "DstK8S_OwnerName" should be preferred over "SrcK8S_Name" or "DstK8S_Name" as much as possible. Refer to the documentation for the list of available fields: https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference.
metricName	**string**	Name of the metric. In Prometheus, it is automatically prefixed with "netobserv_". Leave empty to generate the name based on the **FlowMetric** resource name.
remap	**object (string)**	Set the **remap** property to use different names for the generated metric labels than the flow fields. Use the origin flow fields as keys, and the desired label names as values.

Property	Type	Description
type	**string**	Metric type: "Counter", "Histogram" or "Gauge". Use "Counter" for any value that increases over time and on which you can compute a rate, such as Bytes or Packets. Use "Histogram" for any value that must be sampled independently, such as latencies. Use "Gauge" for other values that don't necessitate accuracy over time (gauges are sampled only every N seconds when Prometheus fetches the metric).
valueField	**string**	**valueField** is the flow field that must be used as a value for this metric (for example: **Bytes**). This field must hold numeric values. Leave empty to count flows rather than a specific value per flow. Refer to the documentation for the list of available fields: https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_observability/json-flows-format-reference.

16.1.3. .spec.charts

Description

Charts configuration, for the OpenShift Container Platform Console in the administrator view, Dashboards menu.

Type

array

16.1.4. .spec.charts[]

Description

Configures charts / dashboard generation associated to a metric

Type

object

Required

- **dashboardName**
- **queries**

- **title**
- **type**

Property	Type	Description
dashboardName	**string**	Name of the containing dashboard. If this name does not refer to an existing dashboard, a new dashboard is created.
queries	**array**	List of queries to be displayed on this chart. If **type** is **SingleStat** and multiple queries are provided, this chart is automatically expanded in several panels (one per query).
sectionName	**string**	Name of the containing dashboard section. If this name does not refer to an existing section, a new section is created. If **sectionName** is omitted or empty, the chart is placed in the global top section.
title	**string**	Title of the chart.
type	**string**	Type of the chart.
unit	**string**	Unit of this chart. Only a few units are currently supported. Leave empty to use generic number.

16.1.5. .spec.charts[].queries

Description

List of queries to be displayed on this chart. If **type** is **SingleStat** and multiple queries are provided, this chart is automatically expanded in several panels (one per query).

Type

array

16.1.6. .spec.charts[].queries[]

Description

Configures PromQL queries

Type

object

Required

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

Property	Type	Description
legend	**string**	The query legend that applies to each timeseries represented in this chart. When multiple timeseries are displayed, you should set a legend that distinguishes each of them. It can be done with the following format: **{{ Label }}**. For example, if the **promQL** groups timeseries per label such as: **sum(rate($METRIC[2m])) by (Label1, Label2)**, you might write as the legend: **Label1={{ Label1 }}, Label2={{ Label2 }}**.
promQL	**string**	The **promQL** query to be run against Prometheus. If the chart **type** is **SingleStat**, this query should only return a single timeseries. For other types, a top 7 is displayed. You can use **$METRIC** to refer to the metric defined in this resource. For example: **sum(rate($METRIC[2m]))**. To learn more about **promQL**, refer to the Prometheus documentation: https://prometheus.io/docs/prometheus/latest/querying/basics/
top	**integer**	Top N series to display per timestamp. Does not apply to **SingleStat** chart type.

16.1.7. .spec.filters

Description

filters is a list of fields and values used to restrict which flows are taken into account. Refer to the documentation for the list of available fields:

https://docs.redhat.com/en/documentation/openshift_container_platform/latest/html/network_obser flows-format-reference.

Type

array

16.1.8. .spec.filters[]

Description

Type

object

Required

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

Property	Type	Description
field	**string**	Name of the field to filter on (for example: **SrcK8S_Namespace**).
matchType	**string**	Type of matching to apply
value	**string**	Value to filter on. When **matchType** is **Equal** or **NotEqual**, you can use field injection with **$(SomeField)** to refer to any other field of the flow.

CHAPTER 17. NETWORK FLOWS FORMAT REFERENCE

Review the specifications for the network flow format, which is used internally and for exporting flow data to Kafka.

17.1. NETWORK FLOWS FORMAT REFERENCE

This is the specification of the network flows format. That format is used when a Kafka exporter is configured, for Prometheus metrics labels as well as internally for the Loki store.

The "Filter ID" column shows which related name to use when defining Quick Filters (see **spec.consolePlugin.quickFilters** in the **FlowCollector** specification).

The "Loki label" column is useful when querying Loki directly: label fields need to be selected using stream selectors.

The "Cardinality" column gives information about the implied metric cardinality if this field was to be used as a Prometheus label with the **FlowMetrics** API. Refer to the **FlowMetrics** documentation for more information on using this API.

Name	Type	Description	Filter ID	Loki label	Cardinality	OpenTelemetry
Bytes	number	Number of bytes	n/a	no	avoid	bytes
DnsErrno	number	Error number returned from DNS tracker ebpf hook function	**dns_errno**	no	fine	dns.errno
DnsFlags	number	DNS flags for DNS record	n/a	no	fine	dns.flags
DnsFlagsResponseCode	string	Parsed DNS header RCODEs name	**dns_flag_response_code**	no	fine	dns.responsecode
DnsId	number	DNS record id	**dns_id**	no	avoid	dns.id
DnsLatencyMs	number	Time between a DNS request and response, in milliseconds	**dns_latency**	no	avoid	dns.latency
Dscp	number	Differentiated Services Code Point (DSCP) value	**dscp**	no	fine	dscp
DstAddr	string	Destination IP address (ipv4 or ipv6)	**dst_address**	no	avoid	destination.address

Name	Type	Description	Filter ID	Loki label	Cardinality	OpenTelemetry
DstK8S_HostIP	string	Destination node IP	**dst_host_address**	no	fine	destination.k8s.host.address
DstK8S_HostName	string	Destination node name	**dst_host_name**	no	fine	destination.k8s.host.name
DstK8S_Name	string	Name of the destination Kubernetes object, such as Pod name, Service name or Node name.	**dst_name**	no	careful	destination.k8s.name
DstK8S_Namespace	string	Destination namespace	**dst_namespace**	yes	fine	destination.k8s.namespace.name
DstK8S_NetworkName	string	Destination network name	**dst_network**	no	fine	n/a
DstK8S_OwnerName	string	Name of the destination owner, such as Deployment name, StatefulSet name, etc.	**dst_owner_name**	yes	fine	destination.k8s.owner.name
DstK8S_OwnerType	string	Kind of the destination owner, such as Deployment, StatefulSet, etc.	**dst_kind**	no	fine	destination.k8s.owner.kind
DstK8S_Type	string	Kind of the destination Kubernetes object, such as Pod, Service or Node.	**dst_kind**	yes	fine	destination.k8s.kind
DstK8S_Zone	string	Destination availability zone	**dst_zone**	yes	fine	destination.zone
DstMac	string	Destination MAC address	**dst_mac**	no	avoid	destination.mac
DstPort	number	Destination port	**dst_port**	no	careful	destination.port

Name	Type	Description	Filter ID	Loki label	Cardinality	OpenTelemetry
DstSubnetLabel	string	Destination subnet label	**dst_subnet_label**	no	fine	destination.subnet.label
Flags	string[]	List of TCP flags comprised in the flow, according to RFC-9293, with additional custom flags to represent the following per-packet combinations: - SYN_ACK - FIN_ACK - RST_ACK	**tcp_flags**	no	careful	tcp.flags
FlowDirection	number	Flow interpreted direction from the node observation point. Can be one of: - 0: Ingress (incoming traffic, from the node observation point) - 1: Egress (outgoing traffic, from the node observation point) - 2: Inner (with the same source and destination node)	**node_direction**	yes	fine	host.direction
IPSecStatus	string	Status of the IPsec encryption (on egress, given by the kernel xfrm_output function) or decryption (on ingress, via xfrm_input)	**ipsec_status**	no	fine	n/a
IcmpCode	number	ICMP code	**icmp_code**	no	fine	icmp.code
IcmpType	number	ICMP type	**icmp_type**	no	fine	icmp.type
IfDirections	number[]	Flow directions from the network interface observation point. Can be one of: - 0: Ingress (interface incoming traffic) - 1: Egress (interface outgoing traffic)	**ifdirections**	no	fine	interface.directions
Interfaces	string[]	Network interfaces	**interfaces**	no	careful	interface.names

Name	Type	Description	Filter ID	Loki label	Cardinality	OpenTelemetry
K8S_ClusterName	string	Cluster name or identifier	**cluster_name**	yes	fine	k8s.cluster.name
K8S_FlowLayer	string	Flow layer: 'app' or 'infra'	**flow_layer**	yes	fine	k8s.layer
NetworkEvents	object[]	Network events, such as network policy actions, composed of nested fields: - Feature (such as "acl" for network policies) - Type (such as an "AdminNetworkPolicy") - Namespace (namespace where the event applies, if any) - Name (name of the resource that triggered the event) - Action (such as "allow" or "drop") - Direction (Ingress or Egress)	**network_events**	no	avoid	n/a
Packets	number	Number of packets	n/a	no	avoid	packets
PktDropBytes	number	Number of bytes dropped by the kernel	n/a	no	avoid	drops.bytes
PktDropLatestDropCause	string	Latest drop cause	**pkt_drop_cause**	no	fine	drops.latestcause
PktDropLatestFlags	number	TCP flags on last dropped packet	n/a	no	fine	drops.latestflags
PktDropLatestState	string	TCP state on last dropped packet	**pkt_drop_state**	no	fine	drops.lateststate
PktDropPackets	number	Number of packets dropped by the kernel	n/a	no	avoid	drops.packets

Name	Type	Description	Filter ID	Loki label	Cardinality	OpenTelemetry
Proto	number	L4 protocol	**protocol**	no	fine	protocol
Sampling	number	Sampling interval used for this flow	n/a	no	fine	n/a
SrcAddr	string	Source IP address (ipv4 or ipv6)	**src_address**	no	avoid	source.address
SrcK8S_HostIP	string	Source node IP	**src_host_address**	no	fine	source.k8s.host.address
SrcK8S_HostName	string	Source node name	**src_host_name**	no	fine	source.k8s.host.name
SrcK8S_Name	string	Name of the source Kubernetes object, such as Pod name, Service name or Node name.	**src_name**	no	careful	source.k8s.name
SrcK8S_Namespace	string	Source namespace	**src_namespace**	yes	fine	source.k8s.namespace.name
SrcK8S_NetworkName	string	Source network name	**src_network**	no	fine	n/a
SrcK8S_OwnerName	string	Name of the source owner, such as Deployment name, StatefulSet name, etc.	**src_owner_name**	yes	fine	source.k8s.owner.name
SrcK8S_OwnerType	string	Kind of the source owner, such as Deployment, StatefulSet, etc.	**src_kind**	no	fine	source.k8s.owner.kind
SrcK8S_Type	string	Kind of the source Kubernetes object, such as Pod, Service or Node.	**src_kind**	yes	fine	source.k8s.kind
SrcK8S_Zone	string	Source availability zone	**src_zone**	yes	fine	source.zone

Name	Type	Description	Filter ID	Loki label	Cardinality	OpenTelemetry
SrcMac	string	Source MAC address	**src_mac**	no	avoid	source.mac
SrcPort	number	Source port	**src_port**	no	careful	source.port
SrcSubnetLabel	string	Source subnet label	**src_subnet_label**	no	fine	source.subnet.label
TimeFlowEndMs	number	End timestamp of this flow, in milliseconds	n/a	no	avoid	timeflowend
TimeFlowRttNs	number	TCP Smoothed Round Trip Time (SRTT), in nanoseconds	**time_flow_rtt**	no	avoid	tcp.rtt
TimeFlowStartMs	number	Start timestamp of this flow, in milliseconds	n/a	no	avoid	timeflowstart
TimeReceived	number	Timestamp when this flow was received and processed by the flow collector, in seconds	n/a	no	avoid	timereceived
Udns	string[]	List of User Defined Networks	**udns**	no	careful	n/a
XlatDstAddr	string	packet translation destination address	**xlat_dst_address**	no	avoid	n/a
XlatDstPort	number	packet translation destination port	**xlat_dst_port**	no	careful	n/a
XlatSrcAddr	string	packet translation source address	**xlat_src_address**	no	avoid	n/a
XlatSrcPort	number	packet translation source port	**xlat_src_port**	no	careful	n/a
ZoneId	number	packet translation zone id	**xlat_zone_id**	no	avoid	n/a

Name	Type	Description	Filter ID	Loki label	Cardinality	OpenTelemetry
_HashId	string	In conversation tracking, the conversation identifier	**id**	no	avoid	n/a
_RecordType	string	Type of record: **flowLog** for regular flow logs, or **newConnection**, **heartbeat**, **endConnection** for conversation tracking	**type**	yes	fine	n/a

CHAPTER 18. TROUBLESHOOTING NETWORK OBSERVABILITY

Perform diagnostic actions to troubleshoot common issues related to the Network Observability Operator and its components.

18.1. USING THE MUST-GATHER TOOL

Use the must-gather tool to collect diagnostic information about Network Observability Operator resources, including pod logs and configuration details, to assist in troubleshooting cluster issues.

Procedure

1. Navigate to the directory where you want to store the must-gather data.
2. Run the following command to collect cluster-wide must-gather resources:

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

18.2. CONFIGURING NETWORK TRAFFIC MENU ENTRY IN THE OPENSHIFT CONTAINER PLATFORM CONSOLE

Restore a missing network traffic menu entry in the **Observe** menu of the OpenShift Container Platform console by manually registering the console plugin in the **FlowCollector** resource and the console operator configuration.

Prerequisites

- You have installed OpenShift Container Platform version 4.10 or newer.

Procedure

1. Check if the **spec.consolePlugin.register** field is set to **true** by running the following command:

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

Example output

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

2. Optional: Add the **netobserv-plugin** plugin by manually editing the Console Operator config:

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

Example output

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

3. Optional: Set the **spec.consolePlugin.register** field to **true** by running the following command:

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

Example output

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

4. Ensure the status of console pods is **running** by running the following command:

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

5. Restart the console pods by running the following command:

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

6. Clear your browser cache and history.

7. Check the status of network observability plugin pods by running the following command:

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

Example output

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

8. Check the logs of the network observability plugin pods by running the following command:

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

Example output

```
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. FLOWLOGS-PIPELINE DOES NOT CONSUME NETWORK FLOWS AFTER INSTALLING KAFKA

Resolve issues where the flow-pipeline fails to consume network flows from Kafka by manually restarting the flow-pipeline pods to restore the connection between the flow collector and your Kafka deployment.

If you deployed the flow collector first with **deploymentModel: KAFKA** and then deployed Kafka, the flow collector might not connect correctly to Kafka. Manually restart the flow-pipeline pods where Flowlogs-pipeline does not consume network flows from Kafka.

Procedure

1. Delete the flow-pipeline pods to restart them by running the following command:

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

18.4. FAILING TO SEE NETWORK FLOWS FROM BOTH BR-INT AND BR-EX INTERFACES

Resolve issues with missing network flows by removing interface restrictions on virtual bridge devices like **br-int** and **br-ex**, ensuring the eBPF agent can attach to the appropriate Layer 3 interfaces.

br-ex and **br-int** are virtual bridge devices operated at OSI layer 2. The eBPF agent works at the IP and TCP levels, layers 3 and 4 respectively. You can expect that the eBPF agent captures the network traffic passing through **br-ex** and **br-int**, when the network traffic is processed by other interfaces such as physical host or virtual pod interfaces. If you restrict the eBPF agent network interfaces to attach only to **br-ex** and **br-int**, you do not see any network flow.

Manually remove the part in the **interfaces** or **excludeInterfaces** that restricts the network interfaces to **br-int** and **br-ex**.

Procedure

1. Remove the **interfaces: ['br-int', 'br-ex']** field. This allows the agent to fetch information from all the interfaces. Alternatively, you can specify the Layer-3 interface for example, **eth0**. Run the following command:

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

Example output

```
apiVersion: flows.netobserv.io/v1alpha1
kind: FlowCollector
metadata:
  name: cluster
spec:
  agent:
    type: EBPF
```

```
ebpf:
  interfaces: [ 'br-int', 'br-ex' ] (1)
```

(1) Specifies the network interfaces.

18.5. NETWORK OBSERVABILITY CONTROLLER MANAGER POD RUNS OUT OF MEMORY

Resolve memory issues with the Network Observability Operator by increasing the memory limits in the **Subscription** object to prevent the controller manager pod from running out of memory.

You can increase memory limits for the Network Observability Operator by editing the **spec.config.resources.limits.memory** specification in the **Subscription** object.

Procedure

1. In the web console, navigate to **Operators → Installed Operators**
2. Click **Network Observability** and then select **Subscription**.
3. From the **Actions** menu, click **Edit Subscription**.
   a. Alternatively, you can use the CLI to open the YAML configuration for the **Subscription** object by running the following command:

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

4. Edit the **Subscription** object to add the **config.resources.limits.memory** specification and set the value to account for your memory requirements. See the Additional resources for more information about resource considerations:

```
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) For example, you can increase the memory limit to **800Mi**.

2. This value should not be edited, but note that it changes depending on the most current release of the Operator.

18.6. RUNNING CUSTOM QUERIES TO LOKI

Troubleshoot network flow data by running custom Loki queries to retrieve available labels or filter logs by specific criteria, such as source namespaces, using the command-line interface.

There are two examples of ways to do this, which you can adapt according to your needs by replacing the <api_token> with your own.

> **NOTE**
>
> These examples use the **netobserv** namespace for the Network Observability Operator and Loki deployments. Additionally, the examples assume that the LokiStack is named **loki**. You can optionally use a different namespace and naming by adapting the examples, specifically the **-n netobserv** or the **loki-gateway** URL.

Prerequisites

- Installed Loki Operator for use with Network Observability Operator.

Procedure

1. To get all available labels, run the following command:

```
$ 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
```

2. To get all flows from the source namespace, **my-namespace**, run the following command:

```
$ 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
```

Additional resources

- Resource considerations

18.7. TROUBLESHOOTING LOKI RESOURCEEXHAUSTED ERROR

Resolve Loki **ResourceExhausted** errors by adjusting the **batchSize** in the **FlowCollector** resource or the maximum message size settings in your Loki configuration to ensure flow data stays within memory limits.

Loki may return a **ResourceExhausted** error when network flow data sent by network observability exceeds the configured maximum message size. If you are using the Red Hat Loki Operator, this maximum message size is configured to 100 MiB.

Procedure

1. Navigate to **Operators → Installed Operators**, viewing **All projects** from the **Project** drop-down menu.
2. In the **Provided APIs** list, select the Network Observability Operator.
3. Click the **Flow Collector** then the **YAML view** tab.
   a. If you are using the Loki Operator, check that the **spec.loki.batchSize** value does not exceed 98 MiB.
   b. If you are using a Loki installation method that is different from the Red Hat Loki Operator, such as Grafana Loki, verify that the **grpc_server_max_recv_msg_size** Grafana Loki server setting is higher than the **FlowCollector** resource **spec.loki.batchSize** value. If it is not, you must either increase the **grpc_server_max_recv_msg_size** value, or decrease the **spec.loki.batchSize** value so that it is lower than the limit.
4. Click **Save** if you edited the **FlowCollector**.

18.8. LOKI EMPTY RING ERROR

Investigate and resolve Loki "empty ring" errors by checking pod health, clearing old persistent volume claims, or restarting pods to restore connectivity and ensure network flows are properly stored and displayed.

The Loki "empty ring" error results in flows not being stored in Loki and not showing up in the web console. This error might happen in various situations. A single workaround to address them all does not exist. There are some actions you can take to investigate the logs in your Loki pods, and verify that the **LokiStack** is healthy and ready.

Some of the situations where this error is observed are as follows:

- After a **LokiStack** is uninstalled and reinstalled in the same namespace, old PVCs are not removed, which can cause this error.
  - **Action**: You can try removing the **LokiStack** again, removing the PVC, then reinstalling the **LokiStack**.
- After a certificate rotation, this error can prevent communication with the **flowlogs-pipeline** and **console-plugin** pods.
  - **Action**: You can restart the pods to restore the connectivity.

18.9. RESOURCE TROUBLESHOOTING

18.10. LOKISTACK RATE LIMIT ERRORS

Resolve Loki rate limit errors and prevent data loss by updating the **LokiStack** resource to increase the ingestion rate and burst limits for your network observability data streams.

A rate-limit placed on the Loki tenant can result in potential temporary loss of data and a 429 error: **Per stream rate limit exceeded (limit:xMB/sec) while attempting to ingest for stream**. You might consider having an alert set to notify you of this error. For more information, see "Creating Loki rate limit alerts for the NetObserv dashboard" in the Additional resources of this section.

You can update the LokiStack CRD with the **perStreamRateLimit** and **perStreamRateLimitBurst** specifications, as shown in the following procedure.

Procedure

1. Navigate to **Operators → Installed Operators**, viewing **All projects** from the **Project** dropdown.
2. Look for **Loki Operator**, and select the **LokiStack** tab.
3. Create or edit an existing **LokiStack** instance using the **YAML view** to add the **perStreamRateLimit** and **perStreamRateLimitBurst** specifications:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: loki
  namespace: netobserv
spec:
  limits:
    global:
      ingestion:
        perStreamRateLimit: 6        (1)
        perStreamRateLimitBurst: 30  (2)
  tenants:
    mode: openshift-network
  managementState: Managed
```

(1) The default value for **perStreamRateLimit** is **3**.

(2) The default value for **perStreamRateLimitBurst** is **15**.

4. Click **Save**.

Verification

Once you update the **perStreamRateLimit** and **perStreamRateLimitBurst** specifications, the pods in your cluster restart and the 429 rate-limit error no longer occurs.

18.11. RUNNING A LARGE QUERY RESULTS IN LOKI ERRORS

Understand how you can mitigate Loki timeout and request errors when running large queries by using indexed filters, leveraging Prometheus for long time ranges, creating custom metrics, or adjusting Loki and FlowCollector performance settings.

When running large queries for a long time, Loki errors can occur, such as a **timeout** or **too many outstanding requests**. There is no complete corrective for this issue, but there are several ways to mitigate it:

Adapt your query to add an indexed filter

With Loki queries, you can query on both indexed and non-indexed fields or labels. Queries that contain filters on labels perform better. For example, if you query for a particular Pod, which is not an indexed field, you can add its Namespace to the query. The list of indexed fields can be found in the "Network flows format reference", in the **Loki label** column.

Consider querying Prometheus rather than Loki

Prometheus is a better fit than Loki to query on large time ranges. However, whether or not you can use Prometheus instead of Loki depends on the use case. For example, queries on Prometheus are much faster than on Loki, and large time ranges do not impact performance. But Prometheus metrics do not contain as much information as flow logs in Loki. The Network Observability OpenShift web console automatically favors Prometheus over Loki if the query is compatible; otherwise, it defaults to Loki. If your query does not run against Prometheus, you can change some filters or aggregations to make the switch. In the OpenShift web console, you can force the use of Prometheus. An error message is displayed when incompatible queries fail, which can help you figure out which labels to change to make the query compatible. For example, changing a filter or an aggregation from **Resource** or **Pods** to **Owner**.

Consider using the FlowMetrics API to create your own metric

If the data that you need isn't available as a Prometheus metric, you can use the FlowMetrics API to create your own metric. For more information, see "FlowMetrics API Reference" and "Configuring custom metrics by using FlowMetric API".

Configure Loki to improve the query performance

If the problem persists, you can consider configuring Loki to improve the query performance. Some options depend on the installation mode you used for Loki, such as using the Operator and **LokiStack**, or **Monolithic** mode, or **Microservices** mode.

- In **LokiStack** or **Microservices** modes, try increasing the number of querier replicas .
- Increase the query timeout. You must also increase the Network Observability read timeout to Loki in the **FlowCollector spec.loki.readTimeout**.

Additional resources

- Network flows format reference
- FlowMetric API reference
- Configuring custom metrics by using FlowMetric API