



OpenShift Container Platform 4.16

Network Observability (网络可观察性)

在 OpenShift Container Platform 中配置和使用 Network Observability Operator

OpenShift Container Platform 4.16 Network Observability (网络可观察性)

在 OpenShift Container Platform 中配置和使用 Network Observability Operator

法律通告

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

摘要

使用 Network Observability Operator 观察和分析 OpenShift Container Platform 集群的网络流量流。

目录

第 1 章 NETWORK OBSERVABILITY OPERATOR 发行注记	4
1.1. NETWORK OBSERVABILITY OPERATOR 1.6.0	4
1.2. NETWORK OBSERVABILITY OPERATOR 1.5.0	6
1.3. NETWORK OBSERVABILITY OPERATOR 1.4.2	8
1.4. NETWORK OBSERVABILITY OPERATOR 1.4.1	8
1.5. NETWORK OBSERVABILITY OPERATOR 1.4.0	9
1.6. NETWORK OBSERVABILITY OPERATOR 1.3.0	11
1.7. NETWORK OBSERVABILITY OPERATOR 1.2.0	13
1.8. NETWORK OBSERVABILITY OPERATOR 1.1.0	14
第 2 章 关于网络可观察性	15
2.1. NETWORK OBSERVABILITY OPERATOR 的可选依赖项	15
2.2. NETWORK OBSERVABILITY OPERATOR	15
2.3. OPENSIFT CONTAINER PLATFORM 控制台集成	15
2.4. NETWORK OBSERVABILITY CLI	15
第 3 章 安装 NETWORK OBSERVABILITY OPERATOR	17
3.1. 没有 LOKI 的 NETWORK OBSERVABILITY	17
3.2. 安装 LOKI OPERATOR	18
3.3. 安装 NETWORK OBSERVABILITY OPERATOR	22
3.4. 重要的流收集器配置注意事项	24
3.5. 安装 KAFKA (可选)	24
3.6. 卸载 NETWORK OBSERVABILITY OPERATOR	25
第 4 章 OPENSIFT CONTAINER PLATFORM 中的 CLUSTER NETWORK OPERATOR	26
4.1. 查看状态	26
4.2. NETWORK OBSERVABILITY OPERATOR 架构	27
4.3. 查看 NETWORK OBSERVABILITY OPERATOR 的状态和配置	28
第 5 章 配置 NETWORK OBSERVABILITY OPERATOR	30
5.1. 查看 FLOWCOLLECTOR 资源	30
5.2. 使用 KAFKA 配置流收集器资源	32
5.3. 导出增强的网络流数据	32
5.4. 更新流收集器资源	34
5.5. 配置快速过滤器	34
5.6. 为 SR-IOV 接口流量配置监控	35
5.7. 资源管理和性能注意事项	36
第 6 章 NETWORK POLICY	38
6.1. 为 NETWORK OBSERVABILITY 创建网络策略	38
6.2. 网络策略示例	38
第 7 章 观察网络流量	40
7.1. 从 OVERVIEW 视图观察网络流量	40
7.2. 从流量流视图观察网络流量	44
7.3. 从 TOPOLOGY 视图中观察网络流量	51
7.4. 过滤网络流量	52
第 8 章 使用带有仪表板和警报的指标	54
8.1. 查看 NETWORK OBSERVABILITY 指标仪表板	54
8.2. 预定义的指标	54
8.3. 网络 OBSERVABILITY 指标	54
8.4. 创建警报	56

8.5. 自定义指标	57
8.6. 使用 FLOWMETRIC API 配置自定义指标	57
8.7. 使用 FLOWMETRIC API 配置自定义 CHART	59
第 9 章 监控 NETWORK OBSERVABILITY OPERATOR	62
9.1. 健康仪表盘	62
9.2. 健康警报	62
9.3. 查看健康信息	62
9.4. 为 NETOBSERV 仪表盘创建 LOKI 速率限制警报	63
9.5. 使用 EBPF 代理警报	64
第 10 章 调度资源	65
10.1. 特定节点中的网络 OBSERVABILITY 部署	65
第 11 章 NETWORK OBSERVABILITY CLI	67
11.1. 安装 NETWORK OBSERVABILITY CLI	67
11.2. 使用 NETWORK OBSERVABILITY CLI	68
11.3. NETWORK OBSERVABILITY CLI (OC NETOBSERV) 参考	70
第 12 章 FLOWCOLLECTOR 配置参数	74
12.1. FLOWCOLLECTOR API 规格	74
第 13 章 FLOWMETRIC 配置参数	125
13.1. FLOWMETRIC [FLOWS.NETOBSERV.IO/V1ALPHA1]	125
第 14 章 网络流格式参考	132
14.1. 网络流格式参考	132
第 15 章 NETWORK OBSERVABILITY 故障排除	137
15.1. 使用 MUST-GATHER 工具	137
15.2. 在 OPENSIFT CONTAINER PLATFORM 控制台中配置网络流量菜单条目	137
15.3. 安装 KAFKA 后 FLOWLOGS-PIPELINE 不会消耗网络流	138
15.4. 无法从 BR-INT 和 BR-EX 接口查看网络流	139
15.5. NETWORK OBSERVABILITY 控制器管理器 POD 内存不足	139
15.6. 对 LOKI 运行自定义查询	140
15.7. LOKI RESOURCEEXHAUSTED 错误故障排除	141
15.8. LOKI 空 RING 错误	141
15.9. 资源故障排除	141
15.10. LOKISTACK 速率限制错误	141

第 1 章 NETWORK OBSERVABILITY OPERATOR 发行注记

Network Observability Operator 可让管理员观察和分析 OpenShift Container Platform 集群的网络流量流。

本发行注记介绍了 OpenShift Container Platform 中 Network Observability Operator 的开发。

有关 Network Observability Operator 的概述，请参阅[关于 Network Observability Operator](#)。

1.1. NETWORK OBSERVABILITY OPERATOR 1.6.0

以下公告可用于 Network Observability Operator 1.6.0：

- [Network Observability Operator 1.6.0](#)

1.1.1. 新功能及功能增强

1.1.1.1. 增强了在没有 Loki 的情况下 Network Observability Operator 的使用

现在，在使用 Network Observability Operator 时，您可以使用 Prometheus 指标并依赖 Loki 进行存储。如需更多信息，请参阅[没有 Loki 的网络可观察性](#)。

1.1.1.2. 自定义 metrics API

您可以使用 **FlowMetrics** API 从 flowlogs 数据中创建自定义指标。flowlogs 数据可用于 Prometheus 标签，以便在仪表板上自定义集群信息。您可以为要在流和指标中识别的任何子网添加自定义标签。此功能增强还可用于使用新标签 **SrcSubnetLabel** 和 **DstSubnetLabel** 来更轻松地识别外部流量，它们同时存在于流日志和指标中。当存在外部流量时，这些字段为空，它提供了一种方法来识别它。如需更多信息，请参阅[自定义指标](#)和[FlowMetric API 参考](#)。

1.1.1.3. eBPF 性能增强

提高了 eBPF 代理的性能，在 CPU 和内存方面有以下更新：

- eBPF 代理现在使用 TCX Webhook 而不是 TC。
- **NetObserv / Health** 仪表板有一个新的部分，显示 eBPF 指标。
 - 根据新的 eBPF 指标，当 eBPF 代理丢弃流时，会向您发送警报。
- 现在，因为删除了重复的流，Loki 存储需求会显著减少。现在，使用一个带有相关网络接口列表的非重复的流，而不是使用多个流，每个网络接口都带有独立重复的流。



重要

通过对重复流机制的更新，网络流量表中的 **Interface** 和 **Interface Direction** 字段被重命名为 **Interfaces** 和 **Interface Directions**，任何使用这些字段的 **快速过滤** 查询都需要更新为使用新的 **interfaces** 和 **ifdirections**。

如需更多信息，请参阅[使用 eBPF 代理警报](#)和[快速过滤器](#)。

1.1.1.4. 基于 eBPF 集合规则的过滤

您可以使用基于规则的过滤来减少创建的流的数量。启用这个选项后，eBPF 代理统计的 **Netobserv / Health** 仪表板会提供一个过滤的**流速率**视图。如需更多信息，请参阅 [eBPF 流规则过滤器](#)。

1.1.2. 技术预览功能

这个版本中的一些功能当前还处于技术预览状态。它们并不适用于在生产环境中使用。请参阅红帽门户网站中关于对技术预览功能支持范围的信息：

技术预览功能支持范围

1.1.2.1. Network Observability CLI

您可以使用 Network Observability CLI 调试网络流量问题并进行故障排除，而无需安装 Network Observability Operator。实时捕获和可视化流和数据包数据，在捕获过程中不需要持久性存储。如需更多信息，请参阅 [Network Observability CLI](#) 和 [Network Observability CLI 1.6.0](#)

1.1.3. 程序错误修复

- 在以前的版本中，Operator Lifecycle Manager (OLM) 表单中会显示到 OpenShift 的死链接，用于创建 **FlowMetrics** API。现在，链接已被更新以指向有效的页面。([NETOBSERV-1607](#))
- 在以前的版本中，Operator Hub 中的 Network Observability Operator 描述信息中显示的一个到文档的链接有问题。在这个版本中，这个链接也被修正。([NETOBSERV-1544](#))
- 在以前的版本中，如果 Loki 被禁用，且 Loki **Mode** 被设置为 **LokiStack**，或者配置了 Loki manual TLS 配置，Network Observability Operator 仍然会尝试读取 Loki CA 证书。在这个版本中，当 Loki 被禁用时，即使在 Loki 配置中有设置，也不会读 Loki 证书。([NETOBSERV-1647](#))
- 在以前的版本中，Network Observability Operator 的 **oc must-gather** 插件只适用于 **amd64** 架构，并在所有其他架构上都会失败，这是因为对于 **oc**，插件使用了 **amd64**。现在，Network Observability Operator **oc must-gather** 插件会在任何架构平台上收集日志。
- 在以前的版本中，当使用 **不等于** 逻辑过滤 IP 地址时，Network Observability Operator 会返回请求错误。现在，对于 IP 地址和范围的 IP 过滤，可以正常使用**等于**和**不等于**逻辑。([NETOBSERV-1630](#))
- 在以前的版本中，当用户不是管理员时，错误消息与 web 控制台中的 **Network Traffic** 视图的所选标签页不一致。现在，**user not admin** 错误可以正确地任何标签页中显示。([NETOBSERV-1621](#))

1.1.4. 已知问题

- 当 eBPF 代理 **PacketDrop** 功能被启用，并抽样被配置为一个大于 **1** 的值时，报告的丢弃的字节并丢弃数据包会忽略抽样配置。虽然这样做的目的是为了不漏任何数据丢弃，但这样做的一个副作用是报告的丢弃数据与非丢弃数据的比例变得有倾向性。例如，对于一个非常高的抽样率（如 **1:1000**）中，在控制台插件中观察到的情况是，几乎所有流量都被丢弃。([NETOBSERV-1676](#))
- 在 **Overview** 选项卡中的 **Manage panels** 弹出窗口中，根据 **total, bar, donut, 或 line** 不会显示任何结果。([NETOBSERV-1540](#))
- 如果首先创建了接口，然后才部署 eBPF 代理，则不会检测到这个 SR-IOV 二级接口。只有在先部署了代理，然后再创建 SR-IOV 接口时，才会检测到它。([NETOBSERV-1697](#))
- 当禁用 Loki 时，OpenShift Web 控制台中的 **Topology** 视图始终会在网络拓扑图旁边的滑块中显示**集群**和**区域**聚合选项，即使未启用相关的功能。现在还没有可以解决这个问题的临时解决方案，只能忽略这些滑块选项。([NETOBSERV-1705](#))

- 当 Loki 被禁用时，OpenShift Web 控制台第一次加载时，可能会显示错误：**Request failed with status code 400 Loki is disabled**。作为临时解决方案，您可以继续在 **Network Traffic** 页面中切换内容，如点 **Topology** 和 **Overview** 选项卡。这个错误应该会消失。[\(NETOBSERV-1706\)](#)

1.2. NETWORK OBSERVABILITY OPERATOR 1.5.0

以下公告可用于 Network Observability Operator 1.5.0：

- [Network Observability Operator 1.5.0](#)

1.2.1. 新功能及功能增强

1.2.1.1. DNS 跟踪增强

在 1.5 中，除了 UDP 外，还支持 TCP 协议。在 **Network Traffic** 页面的 **Overview** 视图中添加了新的仪表板。如需更多信息，请参阅[配置 DNS 跟踪](#)和[使用 DNS 跟踪](#)。

1.2.1.2. 往返时间 (RTT)

您可以使用从 **fentry/tcp_rcv_established** Extended Berkeley Packet Filter (eBPF) hookpoint 中捕获的 TCP 握手 Round-Trip Time (RTT) 来读取平稳的往返时间(SRTT) 并分析网络流。在 web 控制台中的 **Overview**、**Network Traffic**、和 **Topology** 页面中，您可以监控网络流量，并使用 RTT 指标、过滤和边缘标记进行故障排除。如需更多信息，请参阅[RTT 概述](#)和[使用 RTT](#)。

1.2.1.3. 指标、仪表板和警报增强

Observe → **Dashboards** → **NetObserv** 中的 Network Observability 指标仪表板具有可用于创建 Prometheus 警报的新指标类型。现在，您可以在 **includeList** 规格中定义可用指标。在以前的版本中，这些指标在 **ignoreTags** 规格中定义。有关这些指标的完整列表，请参阅 [Network Observability Metrics](#)。

1.2.1.4. 没有 Loki 的 Network Observability 的改进

您可以使用 DNS、Packet drop 和 RTT 指标为 **Netobserv** 仪表板创建 Prometheus 警报，即使您不使用 Loki。在以前的 Network Observability 版本 1.4 中，这些指标仅适用于 **Network Traffic**、**Overview** 和 **Topology** 视图中查询和分析，这些指标在没有 Loki 的情况下不可用。如需更多信息，请参阅 [Network Observability Metrics](#)。

1.2.1.5. 可用区

您可以配置 **FlowCollector** 资源来收集有关集群可用区的信息。此配置增强了使用应用到节点的 [topology.kubernetes.io/zone](#) 标签值的网络流数据。如需更多信息，请参阅[使用可用区](#)。

1.2.1.6. 主要改进

Network Observability Operator 的 1.5 发行版本为 OpenShift Container Platform Web 控制台插件和 Operator 配置添加了改进和新功能。

性能增强

- **spec.agent.ebpf.kafkaBatchSize** 的默认设置从 **10MB** 改为 **1MB**，以便在使用 Kafka 时增强 eBPF 性能。



重要

当从现有安装升级时，不会在配置中自动设置这个新值。如果您在升级后发现 eBPF 代理内存消耗的性能出现回归的问题，您可以考虑将 **kafkaBatchSize** 减少为一个新值。

Web 控制台增强：

- 在 DNS 和 RTT 的 **Overview** 视图中添加了新的面板：Min、Max、P90、P99。
- 添加了新的面板显示选项：
 - 专注于一个面板，同时保持其他面板可查看但没有主要关注。
 - 切换图形类型。
 - 显示 **Top** 和 **Overall**。
- 自定义时间范围弹出窗口中会显示集合延迟警告。
- 增强了**管理面板**和**管理列**弹出窗口内容的可见性。
- 在 web 控制台的 **Network Traffic** 页面中，可以使用 egress QoS 的 Differentiated Services Code Point (DSCP) 字段来对 QoS DSCP 进行过滤。

配置增强：

- **spec.loki.mode** 规格中的 **LokiStack** 模式会自动设置 URL、TLS、集群角色和集群角色绑定，以及 **authToken** 值，来简化安装。**Manual** 模式允许对这些设置进行更多控制。
- API 版本从 **flows.netobserv.io/v1beta1** 更新到 **flows.netobserv.io/v1beta2**。

1.2.2. 程序错误修复

- 在以前的版本中，如果禁用了 console 插件的自动注册功能，则无法在 web 控制台界面中手动注册 console 插件。如果在 **FlowCollector** 资源中将 **spec.console.register** 值设置为 **false**，Operator 会覆盖并清除插件注册。在这个版本中，将 **spec.console.register** 值设置为 **false** 不会影响控制台插件注册或删除。因此，可以安全地手动注册插件。[\(NETOBSERV-1134\)](#)
- 在以前的版本中，使用默认指标设置，**NetObserv/Health** 仪表盘显示一个名为 **Flows Overhead** 的空图形。这个指标只能通过从 **ignoreTags** 列表中删除 "namespaces-flows" 和 "namespaces" 时才可用。在这个版本中，在使用默认指标设置时，此指标可见。[\(NETOBSERV-1351\)](#)
- 在以前的版本中，运行 eBPF 代理的节点无法使用一个特定的集群配置解析。这会导致一连串的后果，并导致无法提供一些流量指标。在这个版本中，eBPF 代理的节点 IP 由 Operator 安全提供，从 pod 状态推断出。现在，缺少的指标会被恢复。[\(NETOBSERV-1430\)](#)
- 在以前的版本中，Loki Operator 的 Loki 错误 'Input size too long' 错误中没有包括可以帮助进行故障排除的额外信息。在这个版本中，帮助信息在 web 控制台中的错误旁显示，并带有直接链接来获得更详细的信息。[\(NETOBSERV-1464\)](#)
- 在以前的版本中，控制台插件读取超时被强制为 30s。使用 **FlowCollector v1beta2** API 更新，您可以配置 **spec.loki.readTimeout** 规格来根据 Loki Operator **queryTimeout** 限制更新这个值。[\(NETOBSERV-1443\)](#)

- 在以前的版本中，Operator 捆绑包不会按预期显示 CSV 注解的一些支持功能，如 `features.operators.openshift.io/...`。在这个版本中，这些注解会在 CSV 中按预期设置。(NETOBSERV-1305)
- 在以前的版本中，在协调过程中，**FlowCollector** 状态有时会在 **DeploymentInProgress** 和 **Ready** 状态之间转换。在这个版本中，只有在所有底层组件都完全就绪时，状态才会变为 **Ready**。(NETOBSERV-1293)

1.2.3. 已知问题

- 当尝试访问 web 控制台时，OCP 4.14.10 上的缓存问题会阻止访问 **Observe** 视图。Web 控制台显示错误消息：**Failed to get a valid plugin manifest from /api/plugins/monitoring-plugin/**。推荐的解决方法是，把集群升级到最新的次版本。如果这无法解决问题，请参阅[红帽知识库文章](#)。(NETOBSERV-1493)
- 由于 Network Observability Operator 的 1.3.0 发行版本，安装 Operator 会导致出现警告内核污点。此错误的原因是，Network Observability eBPF 代理具有内存限制，可防止预分配整个 hashmap 表。Operator eBPF 代理设置 **BPF_F_NO_PREALLOC** 标志，以便在 hashmap 过内存扩展时禁用预分配。

1.3. NETWORK OBSERVABILITY OPERATOR 1.4.2

以下公告可用于 Network Observability Operator 1.4.2：

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

1.3.1. CVE

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

1.4. NETWORK OBSERVABILITY OPERATOR 1.4.1

以下公告可用于 Network Observability Operator 1.4.1：

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

1.4.1. CVE

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

1.4.2. 程序错误修复

- 在 1.4 中，向 Kafka 发送网络流数据时存在一个已知问题。Kafka 消息密钥被忽略，从而导致带有连接跟踪的错误。现在，密钥用于分区，因此来自同一连接的每个流都会发送到同一处理器。[\(NETOBSERV-926\)](#)
- 在 1.4 中，引入了 **Inner** 流方向，以考虑在同一节点上运行的 pod 间的流。在生成的 Prometheus 指标中不会考虑从流派生的 Prometheus 指标中的带有 **Inner** 方向的流，从而导致出现以下字节和数据包率。现在，派生的指标包括带有 **Inner** 方向的流，提供正确的字节和数据包率。[\(NETOBSERV-1344\)](#)

1.5. NETWORK OBSERVABILITY OPERATOR 1.4.0

以下公告可用于 Network Observability Operator 1.4.0 :

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

1.5.1. 频道删除

您必须将频道从 **v1.0.x** 切换到 **stable**，以接收最新的 Operator 更新。**v1.0.x** 频道现已被删除。

1.5.2. 新功能及功能增强

1.5.2.1. 主要改进

Network Observability Operator 的 1.4 发行版本为 OpenShift Container Platform Web 控制台插件和 Operator 配置添加了改进和新功能。

Web 控制台增强 :

- 在 Query Options 中，添加了 **Duplicate 流** 复选框，以选择要显示重复流。
- 现在您可以使用 **↑ One-way**, **↑ ↓ Back-and-forth** 和 **Swap** 过滤器过滤源和目标流。
- **Observe → Dashboards → NetObserv** 和 **NetObserv / Health** 中的 Network Observability 指标仪表盘被修改，如下所示：
 - **NetObserv** 仪表盘显示顶级字节、数据包发送、每个节点、命名空间和工作负载接收的数据包。流图从此仪表盘中删除。
 - **NetObserv / Health** 仪表盘显示流开销，以及每个节点的流率、命名空间和工作负载。
 - 基础架构和应用程序指标显示在命名空间和工作负载的 split-view 中。

如需更多信息，请参阅 [Network Observability 指标](#) 和 [快速过滤器](#)。

配置增强 :

- 现在，您可以选择为任何配置的 ConfigMap 或 Secret 引用指定不同的命名空间，如证书配置中。
- 添加 **spec.processor.clusterName** 参数，以便集群名称出现在流数据中。这在多集群上下文中很有用。使用 OpenShift Container Platform 时，留空，使其自动决定。

如需更多信息，请参阅 [流收集器示例资源](#) 和 [流收集器 API 参考](#)。

1.5.2.2. 没有 Loki 的 Network Observability

Network Observability Operator 现在可以在没有 Loki 的情况下正常工作。如果没有安装 Loki，它只能将流导出到 KAFKA 或 IPFIX 格式，并在 Network Observability 指标仪表板中提供指标。如需更多信息，请参阅[没有 Loki 的网络可观察性](#)。

1.5.2.3. DNS 跟踪

在 1.4 中，Network Observability Operator 使用 eBPF 追踪点 hook 启用 DNS 跟踪。您可以监控网络，执行安全分析，并对 web 控制台中的 **Network Traffic** 和 **Overview** 页面中的 DNS 问题进行故障排除。

如需更多信息，请参阅[配置 DNS 跟踪](#)和[使用 DNS 跟踪](#)。

1.5.2.4. SR-IOV 支持

现在，您可以使用单根 I/O 虚拟化(SR-IOV)设备从集群收集流量。如需更多信息，请参阅[配置 SR-IOV 接口流量的监控](#)。

1.5.2.5. 支持 IPFIX exporter

现在，您可以将 eBPF 丰富的网络流导出到 IPFIX 收集器。如需更多信息，请参阅[导出增强的网络流数据](#)。

1.5.2.6. 数据包丢弃

在 Network Observability Operator 的 1.4 发行版本中，eBPF 追踪点 hook 用于启用数据包丢弃跟踪。现在，您可以检测和分析数据包丢弃的原因，并做出决策来优化网络性能。在 OpenShift Container Platform 4.14 及更高版本中，会检测主机丢弃和 OVS 丢弃。在 OpenShift Container Platform 4.13 中，只检测主机丢弃。如需更多信息，请参阅[配置数据包丢弃跟踪](#)和[使用数据包丢弃](#)。

1.5.2.7. s390x 架构支持

Network Observability Operator 现在可在 **s390x** 构架中运行。以前，它在 **amd64**、**ppc64le** 或 **arm64** 上运行。

1.5.3. 程序错误修复

- 在以前的版本中，被 Network Observability 导出的 Prometheus 指标会忽略潜在的重复网络流。在相关的仪表板中，在 **Observe** → **Dashboards** 中，这可能会导致潜在的双倍率。请注意，来自 **Network Traffic** 视图中的仪表板不会受到影响。现在，会过滤网络流以消除指标计算前的重复项，仪表板中会显示正确的流量率。(NETOBSERV-1131)
- 在以前的版本中，当使用 Multus 或 SR-IOV（非默认网络命名空间）配置时，Network Observability Operator 代理无法捕获网络接口上的流量。现在，所有可用的网络命名空间都会被识别并用于捕获 SR-IOV 的流量。**FlowCollector** 和 **SRIOVnetwork** 自定义资源收集流量需要的配置。(NETOBSERV-1283)
- 在以前的版本中，在 **Operators** → **Installed Operators** 的 Network Observability Operator 详情中，**FlowCollector Status** 字段可能会报告有关部署状态的错误信息。现在，status 字段会显示正确的信息。保持的事件历史记录，按事件日期排序。(NETOBSERV-1224)
- 在以前的版本中，在网络流量负载激增时，某些 eBPF pod 被 OOM 终止，并进入 **CrashLoopBackOff** 状态。现在，eBPF 代理内存占用率有所改进，因此 pod 不会被 OOM 终止，并进入 **CrashLoopBackOff** 状态。(NETOBSERV-975)

- 在以前的版本中，当 `processor.metrics.tls` 设置为 **PROVIDED** 时，`insecureSkipVerify` 选项值被强制为 **true**。现在，您可以将 `insecureSkipVerify` 设置为 **true** 或 **false**，并在需要时提供 CA 证书。(NETOBSERV-1087)

1.5.4. 已知问题

- 由于 Network Observability Operator 的 1.2.0 发行版本使用 Loki Operator 5.6，Loki 证书更改会定期影响 `flowlogs-pipeline` pod，并导致丢弃流而不是写入 Loki 的流。一段时间后，问题会自行修正，但它仍然会在 Loki 证书更改过程中导致临时流数据丢失。此问题仅在在有 120 个节点或更多节点的大型环境中观察到。(NETOBSERV-980)
- 目前，当 `spec.agent.ebpf.features` 包括 `DNSTracking` 时，更大的 DNS 数据包需要 **eBPF** 代理在第一套接字缓冲区(SKB)网段外查找 DNS 标头。需要实施新的 **eBPF** 代理帮助程序功能来支持它。目前，这个问题还没有临时解决方案。(NETOBSERV-1304)
- 目前，当 `spec.agent.ebpf.features` 包括 `DNSTracking` 时，通过 TCP 数据包的 DNS 需要 **eBPF** 代理在 1st SKB 段外查找 DNS 标头。需要实施新的 **eBPF** 代理帮助程序功能来支持它。目前，这个问题还没有临时解决方案。(NETOBSERV-1245)
- 目前，在使用 **KAFKA** 部署模型时，如果配置了对话跟踪，则对话事件可能会在 Kafka 用户间重复，导致跟踪对话不一致和不正确的 volumetric 数据。因此，不建议在 `deploymentModel` 设置为 **KAFKA** 时配置对话跟踪。(NETOBSERV-926)
- 目前，当 `processor.metrics.server.tls.type` 配置为使用 **PROVIDED** 证书时，Operator 会进入一个没有就绪的状态，该状态可能会影响其性能和资源消耗。在解决此问题解决前，建议不要使用 **PROVIDED** 证书，而是使用自动生成的证书，将 `processor.metrics.server.tls.type` 设置为 **AUTO**。(NETOBSERV-1293)
- 由于 Network Observability Operator 的 1.3.0 发行版本，安装 Operator 会导致出现警告内核污点。此错误的原因是，Network Observability eBPF 代理具有内存限制，可防止预分配整个 hashmap 表。Operator eBPF 代理设置 `BPF_F_NO_PREALLOC` 标志，以便在 hashmap 过内存扩展时禁用预分配。

1.6. NETWORK OBSERVABILITY OPERATOR 1.3.0

以下公告可用于 Network Observability Operator 1.3.0：

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

1.6.1. 频道弃用

您必须将频道从 **v1.0.x** 切换到 **stable**，以接收将来的 Operator 更新。**v1.0.x** 频道已弃用，计划在以后的发行版本中删除。

1.6.2. 新功能及功能增强

1.6.2.1. Network Observability 中的多租户

- 系统管理员可以将单独的用户访问或组访问权限限制为存储在 Loki 中的流。如需更多信息，请参阅 [Network Observability 中的多租户](#)。

1.6.2.2. 基于流的指标仪表盘

- 此发行版本添加了一个新的仪表板，它概述了 OpenShift Container Platform 集群中的网络流。如需更多信息，请参阅 [Network Observability 指标](#)。

1.6.2.3. 使用 must-gather 工具进行故障排除

- 有关 Network Observability Operator 的信息现在可以包含在 must-gather 数据中以进行故障排除。如需更多信息，请参阅 [Network Observability must-gather](#)。

1.6.2.4. 现在支持多个构架

- Network Observability Operator 现在可在 **amd64**、**ppc64le** 或 **arm64** 架构上运行。在以前的版本中，它只在 **amd64** 上运行。

1.6.3. 已弃用的功能

1.6.3.1. 弃用的配置参数设置

Network Observability Operator 1.3 发行版本弃用了 **spec.Loki.authToken HOST** 设置。使用 Loki Operator 时，现在必须使用 **FORWARD** 设置。

1.6.4. 程序错误修复

- 在以前的版本中，当通过 CLI 安装 Operator 时，Cluster Monitoring Operator 所需的 **Role** 和 **RoleBinding** 不会按预期安装。从 Web 控制台安装 Operator 时，不会出现这个问题。现在，安装 Operator 的任何方法都会安装所需的 **Role** 和 **RoleBinding**。([NETOBSERV-1003](#))
- 自版本 1.2 起，Network Observability Operator 可以在流集合出现问题时引发警报。在以前的版本中，由于一个程序错误，用于禁用警报的相关配置，**spec.processor.metrics.disableAlerts** 无法正常工作，有时无效。现在，此配置已被修复，可以禁用警报。([NETOBSERV-976](#))
- 在以前的版本中，当 Network Observability 被配置为 **spec.loki.authToken** 为 **DISABLED** 时，只有 **kubeadmin** 集群管理员才能查看网络流。其他类型的集群管理员收到授权失败。现在，任何集群管理员都可以查看网络流。([NETOBSERV-972](#))
- 在以前的版本中，一个 bug 会阻止用户将 **spec.consolePlugin.portNaming.enable** 设置为 **false**。现在，此设置可以设置为 **false** 来禁用端口到服务名称转换。([NETOBSERV-971](#))
- 在以前的版本中，Cluster Monitoring Operator (Prometheus) 不会收集由 console 插件公开的指标，因为配置不正确。现在，配置已被修复，控制台插件指标可以被正确收集并从 OpenShift Container Platform Web 控制台访问。([NETOBSERV-765](#))
- 在以前的版本中，当在 **FlowCollector** 中将 **processor.metrics.tls** 设置为 **AUTO** 时，**flowlogs-pipeline servicemonitor** 不会适应适当的 TLS 方案，且指标在 web 控制台中不可见。现在，这个问题已针对 AUTO 模式解决。([NETOBSERV-1070](#))
- 在以前的版本中，证书配置（如 Kafka 和 Loki）不允许指定 namespace 字段，这意味着证书必须位于部署 Network Observability 的同一命名空间中。另外，当在 TLS/mTLS 中使用 Kafka 时，用户必须手动将证书复制到部署 **eBPF** 代理 pod 的特权命名空间，并手动管理证书更新，如证书轮转时。现在，通过在 **FlowCollector** 资源中为证书添加 namespace 字段来简化 Network Observability 设置。现在，用户可以在不同的命名空间中安装 Loki 或 Kafka，而无需在 Network Observability 命名空间中手动复制其证书。原始证书会被监视，以便在需要时自动更新副本。([NETOBSERV-773](#))
- 在以前的版本中，Network Observability 代理没有涵盖 SCTP、ICMPv4 和 ICMPv6 协议，从而导致较少的全面的网络流覆盖。现在，这些协议可以被识别以改进流覆盖。([NETOBSERV-934](#))

1.6.5. 已知问题

- 当 FlowCollector 中的 `processor.metrics.tls` 设置为 **PROVIDED** 时，`flowlogs-pipeline servicemonitor` 不会适应 TLS 方案。(NETOBSERV-1087)
- 由于 Network Observability Operator 的 1.2.0 发行版本使用 Loki Operator 5.6，Loki 证书更改会定期影响 `flowlogs-pipeline` pod，并导致丢弃流而不是写入 Loki 的流。一段时间后，问题会自行修正，但它仍然会在 Loki 证书更改过程中导致临时流数据丢失。此问题仅在在有 120 个节点或更多节点的大型环境中观察到。(NETOBSERV-980)
- 安装 Operator 时，可能会出现警告内核污点。此错误的原因是，Network Observability eBPF 代理具有内存限制，可防止预分配整个 hashmap 表。Operator eBPF 代理设置 `BPF_F_NO_PREALLOC` 标志，以便在 hashmap 过内存扩展时禁用预分配。

1.7. NETWORK OBSERVABILITY OPERATOR 1.2.0

以下公告可用于 Network Observability Operator 1.2.0：

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

1.7.1. 准备下一次更新

已安装的 Operator 的订阅指定一个更新频道，用于跟踪和接收 Operator 的更新。在 Network Observability Operator 的 1.2 发布前，唯一可用的频道为 **v1.0.x**。Network Observability Operator 的 1.2 发行版本引入了用于跟踪和接收更新的 **stable** 更新频道。您必须将频道从 **v1.0.x** 切换到 **stable**，以接收将来的 Operator 更新。**v1.0.x** 频道已弃用，计划在以后的发行版本中删除。

1.7.2. 新功能及功能增强

1.7.2.1. 流量流视图中的直方图

- 现在，您可以选择显示一段时间内流的直方图表。histogram 可让您视觉化流历史记录，而不会达到 Loki 查询的限制。如需更多信息，请参阅[使用直方图](#)。

1.7.2.2. 对话跟踪

- 现在，您可以通过 **Log Type** 查询流，它允许对同一对话一部分的网络流进行分组。如需更多信息，请参阅[使用对话](#)。

1.7.2.3. Network Observability 健康警报

- 现在，如果因为写入阶段出现错误，或者达到 Loki ingestion 速率限制，Network Observability Operator 会在 `flowlogs-pipeline` 丢弃流时自动创建自动警报。如需更多信息，请参阅[健康仪表盘](#)。

1.7.3. 程序错误修复

- 在以前的版本中，在更改 FlowCollector spec 中的 **namespace** 值后，在前一个命名空间中运行的 **eBPF** 代理 pod 没有被适当删除。现在，在上一个命名空间中运行的 pod 会被正确删除。(NETOBSERV-774)
- 在以前的版本中，在更改 FlowCollector spec（如 Loki 部分）中的 **caCert.name** 值后，FlowLogs-Pipeline pod 和 Console 插件 pod 不会重启，因此它们不知道配置更改。现在，pod 被重启，因此它们会获得配置更改。(NETOBSERV-772)

- 在以前的版本中，在不同节点上运行的 pod 间的网络流有时没有正确识别为重复，因为它们由不同的网络接口捕获。这会导致控制台插件中显示过量的指标。现在，流会被正确识别为重复，控制台插件会显示准确的指标。(NETOBSERV-755)
- 控制台插件中的 "reporter" 选项用于根据源节点或目标节点观察点过滤流。在以前的版本中，无论节点观察点是什么，这个选项都会混合流。这是因为在节点级别将网络流错误地报告为 Ingress 或 Egress。现在，网络流方向报告是正确的。源观察点的 "reporter" 选项过滤器，或目标观察点如预期。(NETOBSERV-696)
- 在以前的版本中，对于配置为直接将流作为 gRPC+protobuf 请求发送的代理，提交的有效负载可能太大，并由处理器的 GRPC 服务器拒绝。这会在有非常高负载的场景中发生，且只会发生在一些代理配置中。代理记录错误消息，例如：`grpc: received message larger than max`。因此，缺少有关这些流的信息丢失。现在，当大小超过阈值时，gRPC 有效负载被分成多个信息。因此，服务器可以维护连接状态。(NETOBSERV-617)

1.7.4. 已知问题

- 在 Network Observability Operator 的 1.2.0 发行版本中，使用 Loki Operator 5.6，Loki 证书转换会定期影响 **flowlogs-pipeline** pod，并导致丢弃流而不是写入 Loki 的流程。一段时间后，问题会自行修正，但它仍然会在 Loki 证书转换过程中导致临时流数据丢失。(NETOBSERV-980)

1.7.5. 主要的技术变化

- 在以前的版本中，您可以使用自定义命名空间安装 Network Observability Operator。此发行版本引入了 **转换 Webhook**，它更改了 **ClusterServiceVersion**。由于这个变化，所有可用的命名空间不再被列出。另外，要启用 Operator 指标集合，无法使用与其他 Operator 共享的命名空间（如 **openshift-operators** 命名空间）。现在，Operator 必须安装在 **openshift-netobserv-operator** 命名空间中。如果您之前使用自定义命名空间安装 Network Observability Operator，则无法自动升级到新的 Operator 版本。如果您之前使用自定义命名空间安装 Operator，您必须删除已安装的 Operator 实例，并在 **openshift-netobserv-operator** 命名空间中重新安装 Operator。务必要注意，对于 **FlowCollector**、Loki、Kafka 和其他插件，仍可使用自定义命名空间（如常用的 **netobserv** 命名空间）。(NETOBSERV-907)(NETOBSERV-956)

1.8. NETWORK OBSERVABILITY OPERATOR 1.1.0

以下公告可用于 Network Observability Operator 1.1.0：

- [RHSA-2023:0786 Network Observability Operator 安全公告更新](#)

Network Observability Operator 现在稳定，发行频道已升级到 **v1.1.0**。

1.8.1. 程序错误修复

- 在以前的版本中，除非 Loki **authToken** 配置被设置为 **FORWARD** 模式，否则不再强制执行身份验证，允许任何可以在 OpenShift Container Platform 集群中连接到 OpenShift Container Platform 控制台的用户，在没有身份验证的情况下检索流。现在，无论 Loki **authToken** 模式如何，只有集群管理员才能检索流。(BZ#2169468)

第 2 章 关于网络可观察性

红帽为集群管理员提供了 Network Observability Operator 来观察 OpenShift Container Platform 集群的网络流量。Network Observability Operator 使用 eBPF 技术创建网络流。然后，网络流会包括 OpenShift Container Platform 的信息。它们以 Prometheus 指标或 Loki 中的日志的形式提供。您可以在 OpenShift Container Platform 控制台中查看和分析所存储的 netflow 信息，以进一步洞察和故障排除。

2.1. NETWORK OBSERVABILITY OPERATOR 的可选依赖项

- **Loki Operator** : Loki 是后端，可用于存储所有收集的流（最大的详细级别）。您可以选择 [在没有 Loki 的情况下使用 Network Observability](#)，但有一些注意事项，如链接部分所述。如果您选择安装 Loki，建议使用红帽支持的 Loki Operator。
- **AMQ Streams Operator** : Kafka 在 OpenShift Container Platform 集群中为大规模部署提供可扩展性、弹性和高可用性。如果您选择使用 Kafka，建议使用 AMQ Streams Operator，因为红帽支持它。

2.2. NETWORK OBSERVABILITY OPERATOR

Network Observability Operator 提供 Flow Collector API 自定义资源定义。Flow Collector 实例是一个集群范围的资源，它允许配置网络流集合。Flow Collector 实例部署 pod 和服务，它组成一个监控管道，然后收集网络流并与 Kubernetes 元数据增强，然后再存储在 Loki 中或生成 Prometheus 指标。eBPF 代理作为 **daemonset** 对象部署，会创建网络流。

2.3. OPENSIFT CONTAINER PLATFORM 控制台集成

OpenShift Container Platform 控制台集成提供了概述、拓扑视图和流量流表。

2.3.1. Network Observability 指标仪表盘

在 OpenShift Container Platform 控制台中的 **Overview** 选项卡中，您可以查看集群中网络流量流的整体聚合指标。您可以选择按区、节点、命名空间、所有者、pod 和服务显示信息。过滤器和显示选项可以进一步优化指标。如需更多信息，请参阅 [从 Overview 视图中获取网络流量](#)。

在 **Observe → Dashboards** 中，**Netobserv** 仪表盘可让您快速了解 OpenShift Container Platform 集群中的网络流。**Netobserv/Health** 仪表盘提供有关 Operator 健康状况的指标。如需更多信息，请参阅 [Network Observability Metrics](#) 和 [查看健康信息](#)。

2.3.2. Network Observability 拓扑视图

OpenShift Container Platform 控制台提供 **Topology** 选项卡，显示网络流的图形表示和流量数量。拓扑视图代表 OpenShift Container Platform 组件之间的流量，作为网络图。您可以使用过滤器和显示选项重新定义图形。您可以访问区、节点、命名空间、所有者、pod 和服务的信息。

2.3.3. 流量流表

流量流表视图为原始流、非聚合过滤选项和可配置列提供视图。OpenShift Container Platform 控制台提供 **流量流** 标签页，显示网络流的数据和流量数量。

2.4. NETWORK OBSERVABILITY CLI

您可以使用 Network Observability CLI (**oc netobserv**)快速调试并排除 Network Observability 的网络问题。Network Observability CLI 是一个流和数据包视觉化工具，它依赖于 eBPF 代理将收集的数据流传输

到临时收集器 pod。在捕获过程中不需要持久性存储。运行后，输出将传输到您的本地计算机。这可以实现快速了解数据包和流数据，而无需安装 Network Observability Operator。

第 3 章 安装 NETWORK OBSERVABILITY OPERATOR

安装 Loki 是使用 Network Observability Operator 的建议前提条件。您可以选择在[没有 Loki 的情况下使用 Network Observability](#)，但有一些注意事项。

Loki Operator 集成了通过 Loki 实现多租户和身份验证的网关，以进行数据流存储。**LokiStack** 资源管理 Loki，它是一个可扩展、高度可用、多租户日志聚合系统和使用 OpenShift Container Platform 身份验证的 Web 代理。**LokiStack** 代理使用 OpenShift Container Platform 身份验证来强制实施多租户，并方便在 Loki 日志存储中保存和索引数据。



注意

Loki Operator 也可用于[配置 LokiStack 日志存储](#)。Network Observability Operator 需要一个专用的、与日志分开的 LokiStack。

3.1. 没有 LOKI 的 NETWORK OBSERVABILITY

您可以通过不执行 Loki 安装步骤，直接跳过至“安装 Network Observability Operator”，来使用没有 Loki 的 Network Observability。如果您只想将流导出到 Kafka 消费者或 IPFIX 收集器，或者您只需要仪表板指标，则不需要为 Loki 安装 Loki 或为 Loki 提供存储。下表比较了在使用和没有使用 Loki 时的功能比较。

表 3.1. 功能可用性与没有 Loki 的功能的比较

	带有 Loki	没有 Loki
Exporters	✓	✓
多租户	✓	✗
完整的过滤和聚合功能 ^[1]	✓	✗
部分过滤和聚合功能 ^[2]	✓	✓
基于流的指标和仪表板	✓	✓
流量流视图概述 ^[3]	✓	✓
流量流视图表	✓	✗
拓扑视图	✓	✓
OpenShift Container Platform 控制台 Network Traffic 标签页集成	✓	✓

1. 例如，每个 pod。
2. 例如，每个工作负载或命名空间。
3. 使用在使用 Loki 时才可以获得数据包丢弃的统计信息。

其他资源

- [导出增强的网络流数据。](#)

3.2. 安装 LOKI OPERATOR

[Loki Operator 版本 5.7+](#) 是 Network Observability 支持的 Loki Operator 版本。这些版本提供了使用 **openshift-network** 租户配置模式创建 **LokiStack** 实例的功能，并为 Network Observability 提供完全自动的、集群内身份验证和授权支持。您可以通过几种方法安装 Loki。其中一种方法是使用 OpenShift Container Platform Web 控制台 Operator Hub。

先决条件

- 支持的日志存储(AWS S3、Google Cloud Storage、Azure、Swift、Minio、OpenShift Data Foundation)
- OpenShift Container Platform 4.10+
- Linux Kernel 4.18+

流程

1. 在 OpenShift Container Platform Web 控制台中，点击 **Operators** → **OperatorHub**。
2. 从可用的 Operator 列表中选择 **Loki Operator**，然后点 **Install**。
3. 在 **Installation Mode** 下，选择 **All namespaces on the cluster**。

验证

1. 验证您安装了 Loki Operator。访问 **Operators** → **Installed Operators** 页面，并查找 **Loki Operator**。
2. 验证 **Loki Operator** 是否在所有项目中 **Status** 为 **Succeeded**。



重要

要卸载 Loki，请参考与用来安装 Loki 的方法相关的卸载过程。您可能会有剩余的 **ClusterRole** 和 **ClusterRoleBindings**、存储在对象存储中的数据，以及需要被删除的持久性卷。

3.2.1. 为 Loki 存储创建 secret

Loki Operator 支持几个日志存储选项，如 AWS S3、Google Cloud Storage、Azure、Swift、Minio、OpenShift Data Foundation。以下示例演示了如何为 AWS S3 存储创建 secret。本例中创建的 secret **loki-s3** 在 "Creating a LokiStack resource" 中引用。您可以通过 web 控制台或 CLI 中创建此 secret。

1. 使用 Web 控制台，进入到 **Project** → **All Projects** 下拉菜单，再选择 **Create Project**。将项目命名为 **netobserv**，再点 **Create**。
2. 使用右上角的 Import 图标 +。将 YAML 文件粘贴到编辑器中。下面显示了一个 S3 存储的 secret YAML 文件示例：

```
apiVersion: v1
kind: Secret
```

```

metadata:
  name: loki-s3
  namespace: netobserv 1
stringData:
  access_key_id: QUtJQUIPU0ZPRE5ON0VYQU1QTEUK
  access_key_secret:
d0phbHJYVXRuRkVNSS9LN01ERU5HL2JQeFJmaUNZRVhBTVBMRUtFWQo=
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1

```

- 1** 本文档中的安装示例在所有组件中使用相同的命名空间 **netobserv**。您可以选择将不同的命名空间用于不同的组件

验证

- 创建 secret 后，您应该会在 web 控制台的 **Workloads** → **Secrets** 下看到它。

其他资源

- [流收集器 API 参考](#)
- [流收集器示例资源](#)
- [Loki 对象存储](#)

3.2.2. 创建 LokiStack 自定义资源

您可以使用 Web 控制台或 OpenShift CLI (**oc**) 部署 **LokiStack** 自定义资源(CR)来创建命名空间或新项目。

流程

1. 进入到 **Operators** → **Installed Operators**, 从 **Project** 下拉菜单查看 **All projects**。
2. 查找 **Loki Operator**。在详情的 **Provided APIs** 下, 选择 **LokiStack**。
3. 点 **Create LokiStack**。
4. 确保在 **Form View** 或 **YAML** 视图中指定以下字段：

```

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 本文档中的安装示例在所有组件中使用相同的命名空间 **netobserv**。您可以选择使用不同的命名空间。
- 2 指定部署大小。在 Loki Operator 5.8 及更新的版本中，Loki 实例支持的大小选项为 **1x.extra-small**、**1x.small** 或 **1x.medium**。



重要

对于部署大小，无法更改 **1x** 值。

- 3 使用集群中可用于 **ReadWriteOnce** 访问模式的存储类名称。您可以使用 **oc get storageclasses** 查看集群中的可用内容。



重要

您不能重复使用用于日志的相同 **LokiStack** CR。

5. 点 **Create**。

3.2.3. 为 **cluster-admin** 用户角色创建新组



重要

以 **cluster-admin** 用户身份查询多个命名空间的应用程序日志，其中集群中所有命名空间的字符总和大于 5120，会导致错误 **Parse error: input size too long (XXXX > 5120)**。为了更好地控制 LokiStack 中日志的访问，请使 **cluster-admin** 用户成为 **cluster-admin** 组的成员。如果 **cluster-admin** 组不存在，请创建它并将所需的用户添加到其中。

使用以下步骤为具有 **cluster-admin** 权限的用户创建新组。

流程

1. 输入以下命令创建新组：

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

2. 输入以下命令将所需的用户添加到 **cluster-admin** 组中：

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

3. 输入以下命令在组中添加 **cluster-admin** 用户角色：

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

3.2.4. 自定义 **admin** 组访问

如果您的部署有很多需要更广泛的用户，您可以使用 `adminGroup` 字段创建自定义组。属于 `LokiStack CR` 的 `adminGroups` 字段中指定的任何组的成员的用户被视为 `admin`。如果还被分配了 `cluster-logging-application-view` 角色，则管理员用户有权访问所有命名空间中的所有应用程序日志。

LokiStack CR 示例

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

- ❶ 自定义管理组仅在此模式中可用。
- ❷ 为此字段输入空 list `[]` 值会禁用 `admin` 组。
- ❸ 覆盖默认组(`system:cluster-admins,cluster-admin,dedicated-admin`)

3.2.5. Loki 部署大小

Loki 的大小使用 `1x.<size>` 格式，其中值 `1x` 是实例数量，`<size>` 指定性能功能。



重要

对于部署大小，无法更改 `1x` 值。

表 3.2. Loki 大小

	1x.demo	1x.extra-small	1x.small	1x.medium
数据传输	仅用于演示	100GB/day	500GB/day	2TB/day
每秒查询数 (QPS)	仅用于演示	1-25 QPS at 200ms	25-50 QPS at 200ms	25-75 QPS at 200ms
复制因子	None	2	2	2
总 CPU 请求	None	14 个 vCPU	34 个 vCPU	54 个 vCPU
内存请求总数	None	31Gi	67Gi	139Gi
磁盘请求总数	40Gi	430Gi	430Gi	590Gi

3.2.6. LokiStack ingestion 限制和健康警报

LokiStack 实例会根据配置的大小带有默认设置。您可以覆盖其中的一些设置，如 ingestion 和查询限制。如果您在 Console 插件中或 **flowlogs-pipeline** 日志中发现 Loki 错误，则可能需要更新它们。Web 控制台中的自动警报会在达到这些限制时通知您。

以下是配置的限制示例：

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

有关这些设置的更多信息，请参阅 [LokiStack API 参考](#)。

3.2.7. 在网络可观察性中启用多租户

Network Observability Operator 中的多租户允许并将单独的用户访问或组访问限制为存储在 Loki 中的流。项目 admins 启用了访问权限。对某些命名空间具有有限访问权限的项目管理员只能访问这些命名空间的流。

前提条件

- 您至少安装了 [Loki Operator 版本 5.7](#)
- 您必须以项目管理员身份登录

流程

1. 运行以下命令，授权 **user1** 的读权限：

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

现在，数据仅限于允许的用户命名空间。例如，可以访问单个命名空间的用户可以查看此命名空间内部的所有流，以及从到这个命名空间流进入这个命名空间。项目管理员有权访问 OpenShift Container Platform 控制台中的 Administrator 视角，以访问 [Network Flows Traffic 页面](#)。

3.3. 安装 NETWORK OBSERVABILITY OPERATOR

您可以使用 OpenShift Container Platform Web 控制台 Operator Hub 安装 Network Observability Operator。安装 Operator 时，它提供 **FlowCollector** 自定义资源定义 (CRD)。在创建 **FlowCollector** 时，您可以在 web 控制台中设置规格。



重要

Operator 的实际内存消耗取决于集群大小和部署的资源数量。可能需要调整内存消耗。如需更多信息，请参阅“重要流收集器配置注意事项”部分中的“网络 Observability 控制器管理器 pod 内存不足”。

先决条件

- 如果您选择使用 Loki，请安装 [Loki Operator 版本 5.7+](#)。
- 您必须具有 **cluster-admin** 权限。
- 需要以下支持的架构之一：**amd64, ppc64le, arm64, 或 s390x**。
- Red Hat Enterprise Linux (RHEL) 9 支持的任何 CPU。
- 必须使用 OVN-Kubernetes 或 OpenShift SDN 配置为主网络插件，并可以选择使用二级接口，如 Multus 和 SR-IOV。



注意

另外，这个安装示例使用 **netobserv** 命名空间，该命名空间在所有组件中使用。您可以选择使用不同的命名空间。

流程

1. 在 OpenShift Container Platform Web 控制台中，点击 **Operators → OperatorHub**。
2. 从 **OperatorHub** 中的可用 Operator 列表中选择 **Network Observability Operator**，然后点 **Install**。
3. 选中 **Enable Operator recommended cluster monitoring on this Namespace** 的复选框。
4. 导航到 **Operators → Installed Operators**。在 **Provided APIs for Network Observability** 下，选择 **Flow Collector** 链接。
5. 进入 **Flow Collector** 选项卡，然后点 **Create FlowCollector**。在表单视图中进行以下选择：
 - a. **spec.agent.ebpf.Sampling**：指定流的抽样大小。较低的抽样大小会对资源利用率造成负面影响。如需更多信息，请参阅“FlowCollector API 参考”，**spec.agent.ebpf**。
 - b. 如果使用 Loki，请设置以下规格：
 - i. **spec.loki.mode**：将其设置为 **LokiStack** 模式，它会自动设置 URL、TLS、集群角色和集群角色绑定，以及 **authToken** 值。或者，使用 **Manual** 模式对这些设置的配置进行更多控制。
 - ii. **spec.loki.loki.name**：将其设置为您的 **LokiStack** 资源的名称。在本文档中，我们使用 **loki**。
 - c. 可选：如果您在大型环境中，请考虑使用 Kafka 配置 **FlowCollector** 以更具弹性且可扩展的方式转发数据。请参阅“Important Flow Collector 配置注意事项”部分的“使用 Kafka 存储配置流收集器资源”。
 - d. 可选：在创建 **FlowCollector** 前配置其他可选设置。例如，如果您选择不使用 Loki，您可以将导出流配置为 Kafka 或 IPFIX。请参阅“重要流收集器配置注意事项”一节中的“导出增强的网络流数据到 Kafka 和 IPFIX”。

6. 点 Create。

验证

要确认这一点，当您进入到 **Observe** 时，您应该看到选项中列出的 **Network Traffic**。

如果 OpenShift Container Platform 集群中没有应用程序流量，默认过滤器可能会显示 "No results"，这会导致没有视觉流。在过滤器选择旁边，选择 **Clear all filters** 来查看流。

3.4. 重要的流收集器配置注意事项

创建 **FlowCollector** 实例后，您可以重新配置它，但 pod 被终止并重新创建，这可能会具有破坏性。因此，您可以考虑在第一次创建 **FlowCollector** 时配置以下选项：

- [使用 Kafka 配置流收集器资源](#)
- [将增强的网络流数据导出到 Kafka 或 IPFIX](#)
- [为 SR-IOV 接口流量配置监控](#)
- [使用对话跟踪](#)
- [使用 DNS 跟踪](#)
- [使用数据包丢弃](#)

其他资源

有关流收集器规格和 Network Observability Operator 架构和资源使用的常规信息，请参阅以下资源：

- [流收集器 API 参考](#)
- [流收集器示例资源](#)
- [资源注意事项](#)
- [Network Observability 控制器管理器 pod 内存不足](#)
- [Network Observability 架构](#)

3.5. 安装 KAFKA（可选）

Kafka Operator 支持大规模环境。Kafka 提供高吞吐量和低延迟数据源，以便以更具弹性且可扩展的方式转发网络流数据。您可以从 Operator Hub 将 Kafka Operator 作为 [Red Hat AMQ Streams](#) 安装，就像 Loki Operator 和 Network Observability Operator 安装一样。请参阅“[使用 Kafka 配置 FlowCollector 资源](#)”以将 Kafka 配置为存储选项。



注意

要卸载 Kafka，请参考与用来安装的方法对应的卸载过程。

其他资源

[使用 Kafka 配置 FlowCollector 资源。](#)

3.6. 卸载 NETWORK OBSERVABILITY OPERATOR

您可以使用 OpenShift Container Platform Web 控制台 Operator Hub 来卸载 Network Observability Operator，在 **Operators → Installed Operators** 区域中工作。

流程

1. 删除 **FlowCollector** 自定义资源。
 - a. 点 **Provided APIs** 列中的 **Network Observability Operator** 旁边的 **Flow Collector**。
 - b. 为**集群**点选项菜单 ，然后选择 **Delete FlowCollector**。
2. 卸载 Network Observability Operator。
 - a. 返回到 **Operators → Installed Operators** 区。
 - b. 点 **Network Observability Operator** 旁边的选项菜单  并选择 **Uninstall Operator**。
 - c. **Home → Projects** 并选择 **openshift-netobserv-operator**
 - d. 进入到 **Actions**，再选择 **Delete Project**
3. 删除 **FlowCollector** 自定义资源定义 (CRD)。
 - a. 进入到 **Administration → CustomResourceDefinitions**。
 - b. 查找 **FlowCollector** 并点选项菜单 。
 - c. 选择 **Delete CustomResourceDefinition**。



重要

如果已安装，Loki Operator 和 Kafka 会保留，需要被独立删除。另外，在一个对象存储中可能会有剩余的数据，以及持久性卷，这需要被删除。

第 4 章 OPENSIFT CONTAINER PLATFORM 中的 CLUSTER NETWORK OPERATOR

Network Observability 是一个 OpenShift 操作器，它部署一个监控管道，以收集并增强网络流量流，由 Network Observability eBPF 代理生成。

4.1. 查看状态

Network Observability Operator 提供 Flow Collector API。创建 Flow Collector 资源时，它会部署 pod 和服务，以在 Loki 日志存储中创建和存储网络流，并在 OpenShift Container Platform Web 控制台中显示仪表板、指标和流。

流程

1. 运行以下命令来查看 **Flowcollector** 的状态：

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

输出示例

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

2. 输入以下命令检查在 **netobserv** 命名空间中运行的 pod 状态：

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

输出示例

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

flowlogs-pipeline pod 收集流，增强收集的流，然后将流发送到 Loki 存储。**netobserv-plugin** pod 为 OpenShift Container Platform 控制台创建一个视觉化插件。

1. 输入以下命令检查在 **netobserv-privileged** 命名空间中运行的 pod 状态：

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

输出示例

NAME	READY	STATUS	RESTARTS	AGE
netobserv-ebpf-agent-4lpp6	1/1	Running	0	151m
netobserv-ebpf-agent-6gbrk	1/1	Running	0	151m
netobserv-ebpf-agent-klpl9	1/1	Running	0	151m
netobserv-ebpf-agent-vrcnf	1/1	Running	0	151m
netobserv-ebpf-agent-xf5jh	1/1	Running	0	151m

netobserv-ebpf-agent pod 监控节点的网络接口以获取流并将其发送到 flowlogs-pipeline pod。

1. 如果使用 Loki Operator，请输入以下命令检查在 **openshift-operators-redhat** 命名空间中运行的 pod 状态：

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

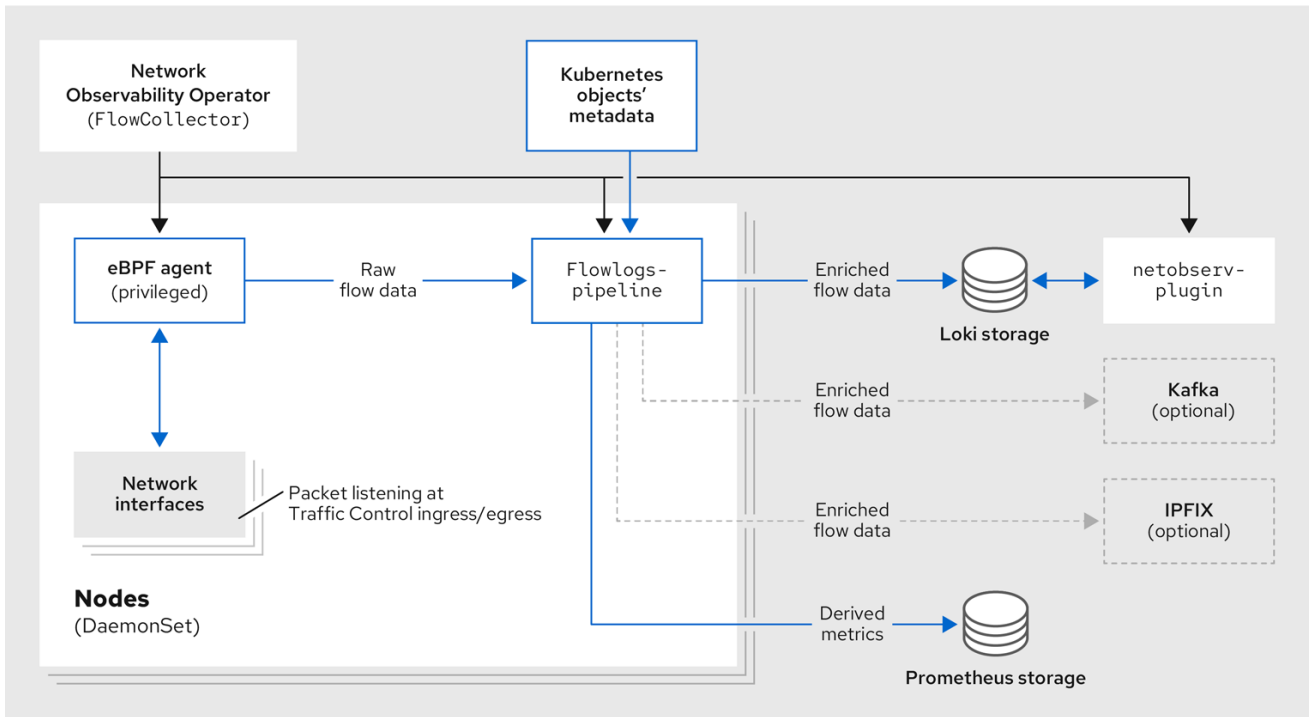
输出示例

NAME	READY	STATUS	RESTARTS	AGE
loki-operator-controller-manager-5f6cff4f9d-jq25h	2/2	Running	0	18h
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

4.2. NETWORK OBSERVABILITY OPERATOR 架构

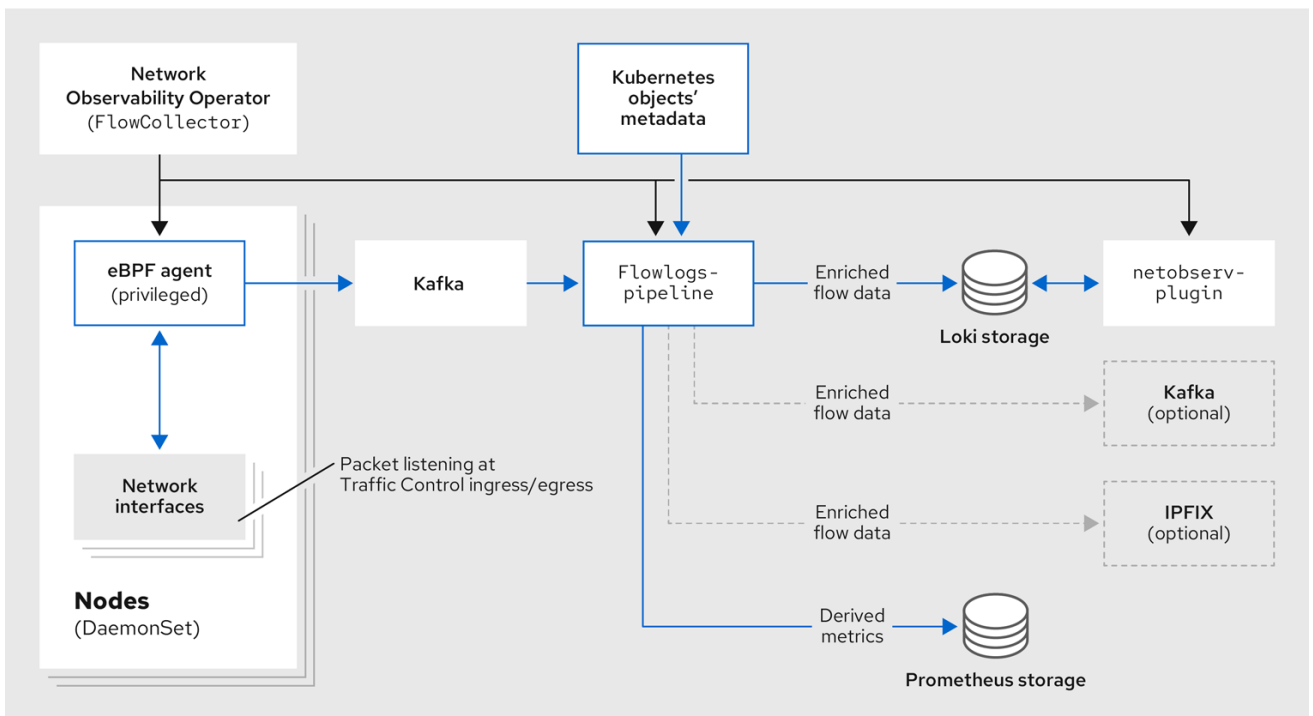
Network Observability Operator 提供 **FlowCollector** API，它在安装时实例化，并配置为协调 **eBPF 代理**、**flowlogs-pipeline** 和 **netobserv-plugin** 组件。只支持每个集群有一个 **FlowCollector**。

eBPF 代理 在每个集群节点上运行，具有一些特权来收集网络流。**flowlogs-pipeline** 接收网络流数据，并使用 Kubernetes 标识符增强数据。如果使用 Loki，**flowlogs-pipeline** 会将流日志数据发送到 Loki 以存储和索引。**netobserv-plugin**，它是一个动态 OpenShift Container Platform Web 控制台插件，查询 Loki 来获取网络流数据。Cluster-admins 可以在 web 控制台中查看数据。



351_OpenShift_0823

如果您使用 Kafka 选项，eBPF 代理将网络流数据发送到 Kafka，并且 **flowlogs-pipeline** 在发送到 Loki 前从 Kafka 主题读取，如下图所示。



351_OpenShift_0823

4.3. 查看 NETWORK OBSERVABILITY OPERATOR 的状态和配置

您可以使用 `oc describe` 命令来检查 **FlowCollector** 的状态并查看其详情。

流程

1. 运行以下命令，以查看 Network Observability Operator 的状态和配置：

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

第 5 章 配置 NETWORK OBSERVABILITY OPERATOR

您可以更新 Flow Collector API 资源，以配置 Network Observability Operator 及其受管组件。流收集器在安装过程中显式创建。由于此资源在集群范围内运行，因此只允许一个 **FlowCollector**，它必须被命名为 **cluster**。

5.1. 查看 FLOWCOLLECTOR 资源

您可以在 OpenShift Container Platform Web 控制台中直接查看和编辑 YAML。

流程

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。在这里，您可以修改 **FlowCollector** 资源来配置 Network Observability operator。

以下示例显示了 OpenShift Container Platform Network Observability operator 的 **FlowCollector** 资源示例：

抽样 FlowCollector 资源

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF 1
    ebpf:
      sampling: 50 2
      logLevel: info
      privileged: false
      resources:
        requests:
          memory: 50Mi
          cpu: 100m
        limits:
          memory: 800Mi
  processor: 3
    logLevel: info
    resources:
      requests:
        memory: 100Mi
        cpu: 100m
      limits:
        memory: 800Mi
    logTypes: Flows
    advanced:
      conversationEndTimeout: 10s

```

```

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

```

- 1 Agent 规格 **spec.agent.type** 必须是 **EBPF**。eBPF 是唯一的 OpenShift Container Platform 支持的选项。
- 2 您可以设置 Sampling 规格 **spec.agent.ebpf.sampling**，以管理资源。低抽样值可能会消耗大量计算、内存和存储资源。您可以通过指定一个抽样比率值来缓解这个问题。100 表示每 100 个流进行 1 个抽样。值 0 或 1 表示捕获所有流。数值越低，返回的流和派生指标的准确性会增加。默认情况下，eBPF 抽样设置为 50，因此每 50 个流抽样 1 个。请注意，更多抽样流也意味着需要更多存储。建议以默认值开始，并逐渐进行调整，以决定您的集群可以管理哪些设置。
- 3 Processor 规格 **spec.processor** 可以设置为启用对话跟踪。启用后，可在 web 控制台中查询对话事件。**spec.processor.logTypes** 的值为 **Flows**。**spec.processor.advanced** 的值为 **Conversations, EndedConversations**, 或 **ALL**。**All** 对于存储的请求最高，**EndedConversations** 对于存储的要求最低。
- 4 Loki 规格 **spec.loki** 指定 Loki 客户端。默认值与安装 Loki Operator 部分中提到的 Loki 安装路径匹配。如果您为 Loki 使用另一个安装方法，请为安装指定适当的客户端信息。
- 5 **LokiStack** 模式会自动设置几个配置：**querierUrl**、**ingesterUrl** 和 **statusUrl**、**tenantID**，以及对应的 TLS 配置。创建集群角色和集群角色绑定用于读取和写入日志到 Loki。将 **authToken** 设置为 **Forward**。您可以使用 **Manual** 模式手动设置它们。
- 6 **spec.quickFilters** 规范定义了要在 web 控制台中显示的过滤器。**Application** 过滤器键 **src_namespace** 和 **dst_namespace** 是负的 (!)，因此 **Application** 过滤器显示不是来自、或目的地是 **openshift-** 或 **netobserv** 命名空间的所有流量。如需更多信息，请参阅配置快速过滤器。

其他资源

有关对话跟踪的更多信息，请参阅[使用对话](#)。

5.2. 使用 KAFKA 配置流收集器资源

您可以将 **FlowCollector** 资源配置为使用 Kafka 进行高吞吐量和低延迟数据源。需要运行一个 Kafka 实例，并在该实例中创建专用于 OpenShift Container Platform Network Observability 的 Kafka 主题。如需更多信息，请参阅 [AMQ Streams 的 Kafka 文档](#)。

先决条件

- Kafka 已安装。红帽支持使用 AMQ Streams Operator 的 Kafka。

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 Network Observability Operator 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择集群，然后点 **YAML** 选项卡。
4. 修改 OpenShift Container Platform Network Observability Operator 的 **FlowCollector** 资源以使用 Kafka，如下例所示：

FlowCollector 资源中的 Kafka 配置示例

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

- 1 将 **spec.deploymentModel** 设置为 **Kafka**，而不是 **Direct** 来启用 Kafka 部署模型。
- 2 **spec.kafka.address** 是指 Kafka bootstrap 服务器地址。如果需要，您可以指定一个端口，如 **kafka-cluster-kafka-bootstrap.netobserv:9093**，以便在端口 9093 上使用 TLS。
- 3 **spec.kafka.topic** 应与 Kafka 中创建的主题名称匹配。
- 4 **spec.kafka.tls** 可用于加密所有与带有 TLS 或 mTLS 的 Kafka 的通信。启用后，Kafka CA 证书必须作为 ConfigMap 或 Secret 提供，两者都位于部署了 **flowlogs-pipeline** 处理器组件的命名空间中（默认为 **netobserv**）以及 eBPF 代理被部署的位置（默认为 **netobserv-privileged**）。它必须通过 **spec.kafka.tls.caCert** 引用。使用 mTLS 时，客户端 secret 还必须在这些命名空间中可用（它们也可以使用 AMQ Streams User Operator 生成并使用 **spec.kafka.tls.userCert**）。

5.3. 导出增强的网络流数据

您可以同时将网络流发送到 Kafka、IPFIX 或这两者。任何支持 Kafka 或 IPFIX 输入的处理程序或存储（如 Splunk、Elasticsearch 或 Fluentd）都可以使用增强的网络流数据。

先决条件

- 您的 Kafka 或 IPFIX 收集器端点可从 Network Observability **flowlogs-pipeline** pod 中提供。

流程

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 编辑 **FlowCollector** 以配置 **spec.exporters**，如下所示：

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

- 2 Network Observability Operator 将所有流导出到配置的 Kafka 主题。
- 3 您可以加密所有使用 SSL/TLS 或 mTLS 的 Kafka 的通信。启用后，Kafka CA 证书必须作为 ConfigMap 或 Secret 提供，两者都位于部署了 **flowlogs-pipeline** 处理程序组件的命名空间中（默认为 netobserv）。它必须使用 **spec.exporters.tls.caCert** 引用。使用 mTLS 时，客户端 secret 还必须在这些命名空间中可用（它们也可以使用 AMQ Streams User Operator 生成并使用 **spec.exporters.tls.userCert** 引用）。

1 4 您可以将流导出到 IPFIX，而不是或与将流导出到 Kafka。

5 您可以选择指定传输。默认值为 **tcp**，但您也可以指定 **udp**。

5. 配置后，网络流数据可以以 JSON 格式发送到可用输出。如需更多信息，请参阅 [网络流格式参考](#)。

其他资源

有关指定流格式的更多信息，请参阅[网络流格式参考](#)。

5.4. 更新流收集器资源

作为在 OpenShift Container Platform Web 控制台中编辑 YAML 的替代方法，您可以通过修补 **flowcollector** 自定义资源 (CR) 来配置规格，如 eBPF 抽样：

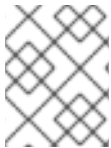
流程

1. 运行以下命令来修补 **flowcollector** CR 并更新 **spec.agent.ebpf.sampling** 值：

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

5.5. 配置快速过滤器

您可以修改 **FlowCollector** 资源中的过滤器。可以使用双引号对值进行完全匹配。否则，会对文本值进行部分匹配。感叹号(!)字符放置在键的末尾，表示负效果。有关修改 YAML 的更多信息，请参阅示例 **FlowCollector** 资源。



注意

匹配类型 "all" 或 "any of" 的过滤器是用户可从查询选项进行修改的 UI 设置。它不是此资源配置的一部分。

以下是所有可用过滤器键的列表：

表 5.1. 过滤器键

Universe*	Source	目的地	描述
namespace	src_namespace	dst_namespace	过滤与特定命名空间相关的流量。
name	src_name	dst_name	过滤与给定叶资源名称相关的流量，如特定 pod、服务或节点（用于 host-network 流量）。
kind	src_kind	dst_kind	过滤与给定资源类型相关的流量。资源类型包括叶资源 (Pod、Service 或 Node) 或所有者资源 (Deployment 和 StatefulSet)。
owner_name	src_owner_name	dst_owner_name	过滤与给定资源所有者相关的流量；即工作负载或一组 pod。例如，它可以是 Deployment 名称、StatefulSet 名称等。
resource	src_resource	dst_resource	过滤与特定资源相关的流量，它们通过其规范名称表示，以唯一标识它。规范表示法是 kind.namespace.name 用于命名空间类型，或 node.name 用于节点。例如， Deployment.my-namespace.my-web-server 。

Universal*	Source	目的地	描述
address	src_address	dst_address	过滤与 IP 地址相关的流量。支持 IPv4 和 IPv6。还支持 CIDR 范围。
mac	src_mac	dst_mac	过滤与 MAC 地址相关的流量。
port	src_port	dst_port	过滤与特定端口相关的流量。
host_address	src_host_address	dst_host_address	过滤与运行 pod 的主机 IP 地址相关的流量。
protocol	N/A	N/A	过滤与协议相关的流量，如 TCP 或 UDP。

- 任何源或目的地的通用密钥过滤器。例如，过滤 **name: 'my-pod'** 表示从 **my-pod** 和所有流量到 **my-pod** 的所有流量，无论使用的匹配类型是什么，无论是 **匹配所有** 还是 **匹配任何**。

5.6. 为 SR-IOV 接口流量配置监控

要使用单根 I/O 虚拟化(SR-IOV)设备从集群收集流量，您必须将 **FlowCollector spec.agent.ebpf.privileged** 字段设置为 **true**。然后，eBPF 代理除主机网络命名空间外监控其他网络命名空间，这些命名空间会被默认监控。当创建具有虚拟功能(VF)接口的 pod 时，会创建一个新的网络命名空间。指定 **SRIOVNetwork** 策略 **IPAM** 配置后，VF 接口从主机网络命名空间迁移到 pod 网络命名空间。

先决条件

- 使用 SR-IOV 设备访问 OpenShift Container Platform 集群。
- **SRIOVNetwork** 自定义资源(CR) **spec.ipam** 配置必须使用接口列表或其他插件范围内的 IP 地址设置。

流程

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 配置 **FlowCollector** 自定义资源。示例配置示例如下：

为 SR-IOV 监控配置 FlowCollector

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

```

kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
agent:
  type: eBPF
  ebpf:
    privileged: true ①

```

① **spec.agent.ebpf.privileged** 字段值必须设置为 **true** 以启用 SR-IOV 监控。

其他资源

有关创建 **SriovNetwork** 自定义资源的更多信息，请参阅使用 [CNI VRF 插件创建额外的 SR-IOV 网络附加](#)。

5.7. 资源管理和性能注意事项

Network Observability 所需的资源量取决于集群的大小以及集群要存储可观察数据的要求。要管理集群的资源并设置性能标准，请考虑配置以下设置。配置这些设置可能会满足您的最佳设置和可观察性需求。

以下设置可帮助您管理 `outset` 中的资源和性能：

eBPF Sampling

您可以设置 Sampling 规格 **spec.agent.ebpf.sampling**，以管理资源。较小的抽样值可能会消耗大量计算、内存和存储资源。您可以通过指定一个抽样比率值来缓解这个问题。**100** 表示每 100 个流进行 1 个抽样。值 **0** 或 **1** 表示捕获所有流。较小的值会导致返回的流和派生指标的准确性增加。默认情况下，eBPF 抽样设置为 50，因此每 50 个流抽样 1 个。请注意，更多抽样流也意味着需要更多存储。考虑以默认值开始，并逐步优化，以确定您的集群可以管理哪些设置。

限制或排除接口

通过为 **spec.agent.ebpf.interfaces** 和 **spec.agent.ebpf.excludeInterfaces** 设置值来减少观察到的流量。默认情况下，代理获取系统中的所有接口，但 **excludeInterfaces** 和 **lo**（本地接口）中列出的接口除外。请注意，接口名称可能会因使用的 Container Network Interface (CNI) 而异。

以下设置可用于在 Network Observability 运行后对性能进行微调：

资源要求和限制

使用 **spec.agent.ebpf.resources** 和 **spec.processor.resources** 规格，将资源要求和限制调整为集群中预期的负载和内存用量。800MB 的默认限制可能足以满足大多数中型集群。

缓存最大流超时

使用 eBPF 代理的 **spec.agent.ebpf.cacheMaxFlows** 和 **spec.agent.ebpf.cacheActiveTimeout** 规格来控制代理报告的频率。较大的值会导致代理生成较少的流量，与较低 CPU 负载相关联。但是，较大的值会导致内存消耗稍有更高的，并可能会在流集合中生成更多延迟。

5.7.1. 资源注意事项

下表概述了具有特定工作负载的集群的资源注意事项示例。



重要

表中概述的示例演示了为特定工作负载量身定制的场景。每个示例仅作为基准，可以进行调整以适应您的工作负载需求。

表 5.2. 资源建议

	Extra small (10 个节点)	Small (25 个节点)	Medium (65 个节点) [2]	Large (120 个节点) [2]
Worker 节点 vCPU 和内存	4 个 vCPU 16GiB 内存 [1]	16 个 vCPU 64GiB 内存 [1]	16 个 vCPU 64GiB 内存 [1]	16 个 vCPU 64GiB Mem [1]
LokiStack 大小	1x.extra-small	1x.small	1x.small	1x.medium
Network Observability 控制器内存限值	400Mi (默认)	400Mi (默认)	400Mi (默认)	400Mi (默认)
eBPF 抽样率	50 (默认)	50 (默认)	50 (默认)	50 (默认)
eBPF 内存限值	800Mi (默认)	800Mi (默认)	800Mi (默认)	1600Mi
FLP 内存限制	800Mi (默认)	800Mi (默认)	800Mi (默认)	800Mi (默认)
FLP Kafka 分区	N/A	48	48	48
Kafka 消费者副本	N/A	24	24	24
Kafka 代理	N/A	3 (默认)	3 (默认)	3 (默认)

1. 使用 AWS M6i 实例测试。
2. 除了此 worker 及其控制器外，还会测试 3 infra 节点 (size **M6i.12xlarge**) 和 1 个工作负载节点 (size **M6i.8xlarge**)。

第 6 章 NETWORK POLICY

作为具有 **admin** 角色的用户，您可以为 **netobserv** 命名空间创建一个网络策略，以保护对 Network Observability Operator 的入站访问。

6.1. 为 NETWORK OBSERVABILITY 创建网络策略

您可能需要创建一个网络策略来保护到 **netobserv** 命名空间的入口流量。在 Web 控制台中，您可以使用表单视图创建网络策略。

流程

1. 进入到 **Networking** → **NetworkPolicies**。
2. 从 **Project** 下拉菜单中选择 **netobserv** 项目。
3. 为策略命名。在本例中，策略名为 **allow-ingress**。
4. 点 **Add ingress rule** 三次来创建三个入站规则。
5. 指定以下内容：
 - a. 对第一个 **Ingress** 规则设置以下规格：
 - i. 从 **Add allowed source** 下拉菜单中选择 **Allow pods**。
 - b. 为第二个 **Ingress** 规则设置以下规格：
 - i. 从 **Add allowed source** 下拉菜单中选择 **Allow pods**。
 - ii. 点 + **Add namespace selector**。
 - iii. 添加标签 **kubernetes.io/metadata.name**，以及选择器 **openshift-console**。
 - c. 为第三个 **Ingress** 规则设置以下规格：
 - i. 从 **Add allowed source** 下拉菜单中选择 **Allow pods**。
 - ii. 点 + **Add namespace selector**。
 - iii. 添加标签 **kubernetes.io/metadata.name**，和选择器 **openshift-monitoring**。

验证

1. 进入到 **Observe** → **Network Traffic**。
2. 查看 **Traffic Flows** 选项卡，或任何标签页，以验证是否显示数据。
3. 进入到 **Observe** → **Dashboards**。在 **NetObserv/Health** 选择中，验证流是否嵌套并发送到 **Loki**，这在第一个图形中表示。

6.2. 网络策略示例

以下注解了 **netobserv** 命名空间的 **NetworkPolicy** 对象示例：

网络策略示例

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

- ❶ 描述策略应用到的 pod 的选择器。策略对象只能选择定义 **NetworkPolicy** 对象的项目中的 pod。在本文档中，它是安装 Network Observability Operator 的项目，即 **netobserv** 项目。
- ❷ 与策略对象允许从中入口流量的 pod 匹配的选择器。默认为选择器与 **NetworkPolicy** 相同的命名空间中的 pod 匹配。
- ❸ 当指定 **namespaceSelector** 时，选择器与指定命名空间中的 pod 匹配。

其他资源

[使用 CLI 创建网络策略](#)

第 7 章 观察网络流量

作为管理员，您可以观察 OpenShift Container Platform 控制台中的网络流量，以了解故障排除和分析的详细故障排除和分析。此功能帮助您从不同的流量流的图形表示获得见解。观察网络流量有几种可用的视图。

7.1. 从 OVERVIEW 视图观察网络流量

Overview 视图显示集群中网络流量流的整体聚合指标。作为管理员，您可以使用可用的显示选项监控统计信息。

7.1.1. 使用 Overview 视图

作为管理员，您可以进入到 **Overview** 视图来查看流速率统计的图形表示。

流程

1. 进入到 **Observe** → **Network Traffic**。
2. 在 **Network Traffic** 页面中，点 **Overview** 选项卡。

您可以通过点菜单图标来配置每个流速率数据的范围。

7.1.2. 为 Overview 视图配置高级选项

您可以使用高级选项自定义图形视图。要访问高级选项，点 **Show advanced options**。您可以使用 **Display options** 下拉菜单在图形中配置详情。可用的选项如下：

- **Scope**：选择查看网络流量流在其中进行的组件。您可以将范围设置为 **Node, Namespace, Owner, Zones, Cluster** 或 **Resource**。**Owner** 是一个资源聚合。**Resource** 可以是一个 pod、服务、节点（主机网络流量），或未知 IP 地址。默认值为 **Namespace**。
- **Truncate labels**：从下拉列表中选择标签所需的宽度。默认值为 **M**。

7.1.2.1. 管理面板和显示

您可以选择显示所需的面板，对它们进行重新排序，并专注于特定的面板。要添加或删除面板，请点 **Manage panels**。

默认情况下会显示以下面板：

- 顶级 X 平均字节率
- 顶级 X 字节率堆栈总计

在 **Manage panels** 中可以添加其他面板：

- 顶级 X 平均数据包率
- 顶级 X 数据包速率堆栈总计

通过 **查询选项**，您可以选择是否显示 **Top 5**、**Top 10** 或 **Top 15** 率。

7.1.3. 数据包丢弃跟踪

您可以在 **Overview** 视图中使用数据包丢失来配置网络流记录的图形表示。通过使用 eBPF 追踪点 hook，您可以获得对 TCP、UDP、SCTP、ICMPv4 和 ICMPv6 协议的数据包丢失的宝贵见解，这可能会导致以下操作：

- **身份识别**：指定发生数据包丢失的确切位置和网络路径。确定特定设备、接口或路由是否更易于丢失。
- **根本原因分析**：检查 eBPF 程序收集的数据，以了解数据包丢失的原因。例如，它们是拥塞、缓冲区问题还是特定的网络事件的结果？
- **性能优化**：通过数据包丢失的清晰了解，您可以采取步骤来优化网络性能，如调整缓冲区大小、重新配置路由路径或实施服务质量(QoS)测量。

启用数据包丢失跟踪后，您可以在 **Overview** 中默认看到以下面板：

- 顶级 X 数据包丢失状态的堆栈为总计
- 顶级 X 数据包丢失会导致堆栈总计
- 顶级 X 平均丢失的数据包率
- 顶级 X 丢失的数据包速率堆栈为总计

可以在**管理面板**中添加其他数据包丢失面板：

- 顶级 X 平均丢失的字节率
- 顶级 X 丢失的字节速率堆栈为总计

7.1.3.1. 数据包丢失的类型

Network Observability 检测到两种类型的数据包丢失：host drops 和 OVS drops。主机丢失的带有 **SKB_DROP** 前缀，OVS drops 带有 **OVS_DROP** 前缀。丢失流在 **流量流** 表的侧面面板中显示，以及指向每个丢失类型的描述链接。主机丢失原因示例如下：

- **SKB_DROP_REASON_NO_SOCKET**：由于缺少套接字而丢失数据包。
- **SKB_DROP_REASON_TCP_CSUM**：由于 TCP checksum 错误而丢失的数据包。

OVS 丢失原因示例如下：

- **OVS_DROP_LAST_ACTION**：OVS 数据包因为隐式丢失操作而丢失，例如因为配置了网络策略。
- **OVS_DROP_IP_TTL**：由于 IP TTL 已过期的 OVS 数据包丢失。

有关启用和使用数据包丢失跟踪的更多信息，请参阅本节的**附加资源**。

其他资源

- [使用数据包丢失](#)
- [网络 Observability 指标](#)

7.1.4. DNS 跟踪

您可以在 **Overview** 视图中配置对网络流的域名系统(DNS)跟踪的图形表示。使用带有扩展 Berkeley Packet Filter (eBPF)追踪点 hook 的 DNS 跟踪可以满足各种目的：

- **网络监控**：深入了解 DNS 查询和响应，帮助网络管理员识别异常模式、潜在瓶颈或性能问题。
- **安全分析**：拒绝 DNS 活动，如恶意软件使用的域名生成算法(DGA)，或者识别可能指示安全漏洞的未授权 DNS 解析。
- **故障排除**：通过追踪 DNS 解析步骤、跟踪延迟和识别错误配置来调试与 DNS 相关的问题。

默认情况下，当启用 DNS 跟踪时，您可以在 **Overview** 中的 donut 或 line chart 中看到以下非空指标：

- 顶级 X DNS 响应代码
- 顶级 X 平均 DNS 延迟数
- 顶部 X 90th percentile DNS 延迟

可以在**管理面板**中添加其他 DNS 跟踪面板：

- 底部 X 最小 DNS 延迟
- 顶级 X 最大 DNS 延迟
- 顶级 X 99th percentile DNS latencies

IPv4 和 IPv6 UDP 和 TCP 协议支持此功能。

有关启用和使用此视图的更多信息，请参阅本节中的*附加资源*。

其他资源

- [使用 DNS 跟踪](#)
- [网络 Observability 指标](#)

7.1.5. 往返时间 (RTT)

您可以使用 TCP smoothed Round-Trip Time (sRTT) 来分析网络流延迟。您可以使用在 **fentry/tcp_rcv_established** eBPF hookpoint 捕获的 RTT 来读取来自 TCP 套接字的 sRTT 来帮助实现：

- **网络监控**：深入了解 TCP 延迟，帮助网络管理员识别异常模式、潜在瓶颈或性能问题。
- **故障排除**：通过跟踪延迟和识别错误配置来调试与 TCP 相关的问题。

默认情况下，当启用 RTT 时，您可以看到 **Overview** 中代表的以下 TCP RTT 指标：

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

可以在**管理面板**中添加其他 RTT 面板：

- 总的 Top X maximum TCP Round Trip Time

- 总的 Top X 99th percentile TCP Round Trip Time

有关启用和使用此视图的更多信息，请参阅本节中的[附加资源](#)。

其他资源

- [使用 RTT 追踪](#)

7.1.6. eBPF 流规则过滤

您可以使用基于规则的过滤来控制缓存在 eBPF 流表中的数据包的数量。例如，可以指定一个过滤，只记录来自端口 100 的数据包。然后，只有与过滤匹配的数据包才会被缓存，不会缓存其他数据包。

7.1.6.1. 入口和出口流量过滤

CIDR 表示法通过将基本 IP 地址与前缀长度相结合来有效地表示 IP 地址范围。对于入口和出口流量，首先使用源 IP 地址来匹配使用 CIDR 表示法配置的过滤规则。如果存在匹配项，则过滤将继续。如果没有匹配项，则使用目标 IP 匹配使用 CIDR 表示法配置的过滤规则。

在匹配源 IP 或目标 IP CIDR 后，您可以使用 **peerIP** 找出特定的端点以区分数据包的目标 IP 地址。根据置备的操作，流数据会在 eBPF 流表中缓存，或者不缓存。

7.1.6.2. 仪表板和指标集成

启用这个选项后，eBPF 代理统计的 Netobserv/Health 仪表板会提供一个过滤的**流速率**视图。另外，在 **Observe → Metrics** 中，您可以查询 **netobserv_agent_filtered_flows_total** 来观察 **FlowFilterAcceptCounter**、**FlowFilterNoMatchCounter** 或 **FlowFilterRejectCounter** 的原因。

7.1.6.3. 流过滤配置参数

流过滤规则包括了必须的和可选的参数。

表 7.1. 所需的配置参数

参数	描述
enable	将 enable 设置为 true 以启用 eBPF 流过滤功能。
cidr	为流过滤规则提供 IP 地址和 CIDR 掩码。支持 IPv4 和 IPv6 地址格式。如果要与任何 IP 匹配，可以使用 0.0.0.0/0 （用于 IPv4），或 ::/0 （用于 IPv6）。
action	<p>描述为流过滤规则执行的操作。可能的值为 Accept 或 Reject。</p> <ul style="list-style-type: none"> • 对于 Accept 操作匹配规则，流数据在 eBPF 表中缓存，并使用全局指标 FlowFilterAcceptCounter 更新。 • 如果匹配规则的操作是 Reject，流数据将被丢弃，且不会在 eBPF 表中缓存。流数据使用全局指标 FlowFilterRejectCounter 更新。 • 如果规则不匹配，流会在 eBPF 表中缓存，并使用全局指标 FlowFilterNoMatchCounter 更新。

表 7.2. 可选的配置参数

参数	描述
direction	定义流过滤规则的方向。可能的值为 Ingress 或 Egress 。
protocol	定义流过滤规则的协议。可能的值有 TCP 、 UDP 、 SCTP 、 ICMP 和 ICMPv6 。
ports	定义用于过滤流的端口。它可用于源或目标端口。要过滤一个单一端口，使用一个整数值。例如 ports: 80 。要过滤一系列端口，使用"开始-结束"范围形式的字符串。例如 ports: "80-100"
sourcePorts	定义用于过滤流的源端口。要过滤一个单一端口，使用一个整数值。例如 sourcePorts: 80 。要过滤一系列端口，使用"开始-结束"范围形式的字符串。例如 sourcePorts: "80-100" 。
destPorts	DestPorts 定义用于过滤流的目标端口。要过滤一个单一端口，使用一个整数值。例如 destPorts: 80 。要过滤一系列端口，使用"开始-结束"范围形式的字符串。例如 destPorts: "80-100" 。
icmpType	定义用于过滤流的 ICMP 类型。
icmpCode	定义用于过滤流的 ICMP 代码。
peerIP	定义用于过滤流的 IP 地址，例如： 10.10.10.10 。

其他资源

- [使用规则过滤 eBPF 流数据](#)
- [网络 Observability 指标](#)
- [健康仪表盘](#)

7.2. 从流量流视图观察网络流量

流量流 视图显示网络流的数据以及表中的流量数量。作为管理员，您可以使用流量流表监控应用程序间的流量数量。

7.2.1. 使用流量流视图

作为管理员，您可以进入 **流量流** 表来查看网络流信息。

流程

1. 进入到 **Observe** → **Network Traffic**。
2. 在 **Network Traffic** 页面中，点 **流量流** 选项卡。

您可以点击每行来获取对应的流信息。

7.2.2. 为流量流视图配置高级选项

您可以使用 **Show advanced options** 自定义和导出视图。您可以使用 **Display options** 下拉菜单设置行大小。默认值为 **Normal**。

7.2.2.1. 管理列

您可以选择显示所需的列，并对它们进行重新排序。若要管理列，可点 **Manage 列**。

7.2.2.2. 导出流量流数据

您可以从**流量流**视图导出数据。

流程

1. 点 **Export data**。
2. 在弹出窗口中，您可以选择 **Export all data** 复选框，以导出所有数据，然后清除复选框以选择要导出的必填字段。
3. 单击 **Export**。

7.2.3. 使用对话跟踪

作为管理员，您可以对属于同一对话的网络流进行分组。对话被定义为一组由 IP 地址、端口和协议标识的对等点，从而产生唯一的 **Conversation Id**。您可以在 web 控制台中查询对话事件。这些事件在 web 控制台中表示，如下所示：

- **Conversation start**：连接启动或 TCP 标记被截获时发生此事件
- **Conversation tick**：此事件在连接处于活跃状态时根据 **FlowCollector spec.processor.conversationHeartbeatInterval** 参数中定义每个指定间隔发生。
- **Conversation end**：当达到 **FlowCollector spec.processor.conversationEndTimeout** 参数或 TCP 标志被截获时，会发生此事件。
- **Flow**：这是在指定间隔内的网络流量流。

流程

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 配置 **FlowCollector** 自定义资源，以便根据您的观察需求设置 **spec.processor.logTypes**，**conversationEndTimeout**，和 **conversationHeartbeatInterval** 参数。示例配置示例如下：

配置 FlowCollector 以对话跟踪

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

```
spec:
  processor:
    logTypes: Flows 1
  advanced:
    conversationEndTimeout: 10s 2
    conversationHeartbeatInterval: 30s 3
```

- 1 当 **logTypes** 设置为 **Flows** 时，只会导出 **Flow** 事件。如果将值设为 **All**，则会在 **Network Traffic** 页面中导出并看到对话和流事件。要只专注于对话事件，您可以指定 **Conversations**，它会导出 **Conversation start**, **Conversation tick** and **Conversation end** 事件；或指定 **EndedConversations**，它只导出 **Conversation end** 事件。**All** 对于存储的请求最高，**EndedConversations** 对于存储的要求最低。
- 2 **Conversation end** 事件表示达到了 **conversationEndTimeout**，或 TCP 标志被截获。
- 3 **Conversation tick** 事件表示当网络连接活跃时，在 **FlowCollector** **conversationHeartbeatInterval** 参数中定义的每个指定间隔。



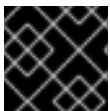
注意

如果您更新了 **logType** 选项，则之前选择中的流不会从控制台插件中清除。例如，如果您最初将 **logType** 设置为 **Conversations**，持续到 10 AM，然后移到 **EndedConversations**，控制台插件会显示 10 AM 之前的所有对话事件，且仅在 10 AM 后终止对话。

5. 刷新 **Traffic flows** 标签页中的 **Network Traffic**。通知请注意，有两个新列：**Event/Type** 和 **Conversation Id**。当 **Flow** 是所选查询选项时，所有 **Event/Type** 字段都是 **Flow**。
6. 选择 **Query Options** 并选择 **Log Type, Conversation**。现在，**Event/Type** 会显示所有所需的对话事件。
7. 接下来，您可以过滤侧面板中的 **Conversation** 和 **Flow** 日志类型选项的特定对话 ID 或切换。

7.2.4. 使用数据包丢弃

当网络流数据的一个或多个数据包无法访问其目的地时，会发生数据包丢失。您可以通过将 **FlowCollector** 编辑到以下 YAML 示例中的规格来跟踪这些丢弃。



重要

启用此功能时，CPU 和内存用量会增加。

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **集群**，然后选择 **YAML** 选项卡。
4. 为数据包丢弃配置 **FlowCollector** 自定义资源，例如：

FlowCollector 配置示例

```

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

```

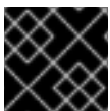
- 1 您可以通过列出 **spec.agent.ebpf.features** 规格列表中的 **PacketDrop** 参数开始报告每个网络流的数据包丢弃。
- 2 对于数据包丢弃跟踪，**spec.agent.ebpf.privileged** 规格值必须是 **true**。

验证

- 刷新 **Network Traffic** 页面时，**Overview**、**Traffic Flow** 和 **Topology** 视图会显示有关数据包丢弃的新信息：
 - a. 在 **Manage** 面板中选择新选择，以选择要在 **Overview** 中显示的数据包丢弃的图形视觉化。
 - b. 在 **Manage** 列中选择新选择，以选择要在**流量流表**中显示哪些数据包丢弃信息。
 - i. 在 **流量流视图** 中，您还可以展开侧面板来查看有关数据包丢弃的更多信息。主机丢弃的带有 **SKB_DROP** 前缀，OVS drops 带有 **OVS_DROP** 前缀。
 - c. 在 **Topology** 视图中，会显示红色的行，其中出现 drops。

7.2.5. 使用 DNS 跟踪

使用 DNS 跟踪，您可以监控网络、进行安全分析并对 DNS 问题进行故障排除。您可以通过将 **FlowCollector** 编辑到以下 YAML 示例中的规格来跟踪 DNS。



重要

启用这个功能时，在 eBPF 代理中观察到 CPU 和内存用量。

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 **Network Observability** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 配置 **FlowCollector** 自定义资源。示例配置示例如下：

为 DNS 跟踪配置 FlowCollector

```

apiVersion: flows.netobserv.io/v1beta2

```

```

kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  agent:
    type: eBPF
    ebpf:
      features:
        - DNSTracking
      sampling: 1

```

- 1 您可以设置 **spec.agent.ebpf.features** 参数列表，以便在 web 控制台中启用每个网络流的 DNS 跟踪。
- 2 您可以将 **sampling** 设置为 1，以获得更准确的指标并捕获 DNS 延迟。如果 **sampling** 的值大于 1，您可以使用 **DNS Response Code** 和 **DNS Id** 观察流，且不太可能观察 DNS 延迟。

5. 刷新 **Network Traffic** 页面时，您可以选择在 **Overview** 和 **Traffic Flow** 视图和可以应用的新过滤器中查看新的 DNS 表示。
 - a. 在 **Manage 面板** 中选择新的 DNS 选项，在 **Overview** 中显示图形视觉化和 DNS 指标。
 - b. 在 **Manage 列** 中选择新选择，将 DNS 列添加到 **流量流视图** 中。
 - c. 过滤特定 DNS 指标，如 **DNS Id**、**DNS Error** **DNS Latency** 和 **DNS Response Code**，并在侧面面板中查看更多信息。默认情况下会显示 **DNS Latency** 和 **DNS Response Code** 列。



注意

TCP 握手数据包没有 DNS 标头。在 **DNS Latency**、**ID** 和 **响应代码** 值 "n/a" 的流量流中会显示没有 DNS 标头的 TCP 协议流。您可以过滤掉流数据，以只查看使用**通用过滤器** "DNSError" 等于 "0" 的 DNS 标头的流。

7.2.6. 使用 RTT 追踪

您可以通过将 **FlowCollector** 编辑到以下 YAML 示例中的规格来跟踪 RTT。

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题中，选择 **Flow Collector**。
3. 选择 **集群**，然后选择 **YAML** 选项卡。
4. 为 RTT 追踪配置 **FlowCollector** 自定义资源，例如：

FlowCollector 配置示例

```

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

```

```
spec:
  namespace: netobserv
  agent:
    type: eBPF
    eBPF:
      features:
        - FlowRTT ❶
```

- ❶ 您可以通过列出 **spec.agent.eBPF.features** 规格列表中的 **FlowRTT** 参数来启动追踪 RTT 网络流。

验证

刷新 **Network Traffic** 页面时，**Overview**、**Traffic Flow** 和 **Topology** 视图会显示有关 RTT 的新信息：

- 在 **Overview** 中，在 **Manage** 面板中选择新选择，以选择要显示的 RTT 的图形可视化。
- 在 **Traffic flows** 表中，可以看到 **Flow RTT** 列，您可以在 **Manage** 列中管理显示。
- 在 **流量流视图** 中，您还可以展开侧面板来查看有关 RTT 的更多信息。

过滤示例

- 点 **Common** 过滤 → **Protocol**。
 - 根据 **TCP**、**Ingress** 方向过滤网络流数据，并查找大于 10,000,000 纳秒(10ms)的 **FlowRTT** 值。
 - 删除 **Protocol** 过滤。
 - 在 **Common** 过滤器中过滤大于 0 的 **Flow RTT** 值。
- d. 在 **Topology** 视图中，点 **Display** 选项下拉菜单。然后点 **edge labels** 下拉列表中的 **RTT**。

7.2.6.1. 使用直方图

您可以点 **Show histogram** 来显示工具栏视图，以使用栏图的形式可视化流历史记录。histogram 显示一段时间内的日志数量。您可以选择直方图的一部分在下面的工具栏中过滤网络流数据。

7.2.7. 使用可用区

您可以配置 **FlowCollector** 以收集有关集群可用区的信息。这可让您使用应用到节点的 topology.kubernetes.io/zone 标签值增强网络流数据。

流程

- 在 Web 控制台中，进入 **Operators** → **Installed Operators**。
- 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
- 选择 **cluster**，然后选择 **YAML** 选项卡。
- 配置 **FlowCollector** 自定义资源，使 **spec.processor.addZone** 参数设置为 **true**。示例配置示例如下：

为可用区集合配置 FlowCollector

```

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

```

验证

刷新 **Network Traffic** 页面时，**Overview**、**Traffic Flow** 和 **Topology** 视图会显示有关可用区的新信息：

1. 在 **Overview** 选项卡中，您可以将 **Zones** 视为可用 **Scope**。
2. 在 **Network Traffic** → **Traffic flows** 中，**Zone** 可以在 **SrcK8S_Zone** 和 **DstK8S_Zone** 字段下查看。
3. 在 **Topology** 视图中，您可以将 **Zones** 设置为 **Scope** 或 **Group**。

7.2.8. 使用全局规则过滤 eBPF 流数据

您可以使用全局规则配置 **FlowCollector** 来过滤 eBPF 流，以控制在 eBPF 流表中缓存的数据包流。

流程

1. 在 Web 控制台中，进入到 **Operators** → **Installed Operators**。
2. 在 **Network Observability** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **集群**，然后选择 **YAML** 选项卡。
4. 配置 **FlowCollector** 自定义资源，类似以下示例配置：

例 7.1. 过滤到特定 Pod IP 端点的 Kubernetes 服务流量

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF
    ebpf:
      flowFilter:
        action: Accept 1
        cidr: 172.210.150.1/24 2
        protocol: SCTP
        direction: Ingress

```

```

destPortRange: 80-100
peerIP: 10.10.10.10
enable: true ③

```

- ① 必需的 **action** 参数描述了为流过滤规则执行的操作。可能的值有 **Accept** 或 **Reject**。
- ② 必需的 **cidr** 参数为流过滤规则提供 IP 地址和子网掩码，并支持 IPv4 和 IPv6 地址格式。如果要匹配任何 IP 地址，您可以使用 **0.0.0.0/0**（用于 IPv4），**::/0**（用于 IPv6）。
- ③ 您必须将 **spec.agent.ebpf.flowFilter.enable** 设置为 **true** 才能启用此功能。

例 7.2. 请参阅流到集群外的任何地址

```

apiVersion: flows.netobserv.io/v1beta2
kind: FlowCollector
metadata:
  name: cluster
spec:
  namespace: netobserv
  deploymentModel: Direct
  agent:
    type: eBPF
    ebpf:
      flowFilter:
        action: Accept ①
        cidr: 0.0.0.0/0 ②
        protocol: TCP
        direction: Egress
        sourcePort: 100
        peerIP: 192.168.127.12 ③
        enable: true ④

```

- ① 您可以根据 **flowFilter** 规格中的条件 **Accept** 流。
- ② **cidr** 值 **0.0.0.0/0** 匹配任何 IP 地址。
- ③ 在 **peerIP** 被配置为 **192.168.127.12** 后的流。
- ④ 您必须将 **spec.agent.ebpf.flowFilter.enable** 设置为 **true** 才能启用此功能。

7.3. 从 TOPOLOGY 视图中观察网络流量

Topology 视图提供了网络流和流量数量的图形表示。作为管理员，您可以使用 **Topology** 视图监控应用程序间的流量数据。

7.3.1. 使用 Topology 视图

作为管理员，您可以进入到 **Topology** 视图来查看组件的详情和指标。

流程

1. 进入到 **Observe → Network Traffic**。
2. 在 **Network Traffic** 页面中，点 **Topology** 选项卡。

您可以点 **Topology** 中的每个组件来查看组件的详情和指标。

7.3.2. 为 **Topology** 视图配置高级选项

您可以使用 **Show advanced options** 自定义和导出视图。高级选项视图具有以下功能：

- **Find in view**: 要在视图中搜索所需组件。
- **Display options** : 要配置以下选项：
 - **Edge labels** : 将指定的测量显示为边缘标签。默认值为显示 **Average rate** (以 **Bytes** 为单位)。
 - **Scope** : 选择网络流量流之间的组件范围。默认值为 **Namespace**。
 - **组** : 通过对组件进行分组来充分了解所有权。默认值为 **None**。
 - **Layout** : 要选择图形表示的布局。默认值为 **ColaNoForce**。
 - **显示** : 要选择需要显示的详细信息。默认检查所有选项。可以选项为：**Edges, Edges label, 和 Badges**。
 - **Truncate labels** : 从下拉列表中选择标签所需的宽度。默认值为 **M**。
 - **折叠组** : 要展开或折叠组。默认会扩展组。如果 **Groups** 的值为 **None**, 这个选项会被禁用。

7.3.2.1. 导出拓扑视图

要导出视图，点 **Export topology view**。该视图以 **PNG** 格式下载。

7.4. 过滤网络流量

默认情况下，**Network Traffic** 页面根据 **FlowCollector** 实例中配置的默认过滤器显示集群中的流量流数据。您可以通过更改 **preset** 过滤器，使用过滤器选项观察所需的数据。

查询选项

您可以使用 **Query Options** 来优化搜索结果，如下所示：

- **日志类型** : 可用选项 **Conversation** 和 **Flows** 提供了按日志类型查询流的能力，如流日志、新对话、完成对话和心跳，这是长期对话的定期记录。对话是同一对等点之间的流聚合。
- **Match filters** : 您可以确定高级过滤器中选择的过滤器参数之间的关系。可用的选项包括 **Match all** 和 **Match any**。 **Match all** 提供与所有值都匹配的结果，而 **Match any** 则提供与输入的任何值匹配的结果。默认值为 **Match all**。
- **DataSource** : 您可以选择用于查询的数据源：**Loki**、**Prometheus** 或 **Auto**。当使用 **Prometheus** 而不是 **Loki** 用作数据源时，性能会有显著的提高，但 **Prometheus** 支持的过滤和聚合的功能有限。默认数据源是 **Auto**。如果查询支持，使用 **Prometheus**，如果查询不支持 **Prometheus**，则使用 **Loki**。

- **丢弃过滤器**：您可以使用以下查询选项查看不同的丢弃数据包级别：
 - **完全丢弃** 显示带有完全丢弃的数据包的流记录。
 - **包含丢弃** 显示包含丢弃但可以发送的流记录。
 - **没有丢弃** 显示包含已发送数据包的记录。
 - **All** 显示上述所有记录。
- **Limit**：内部后端查询的数据限制。根据匹配和过滤器设置，流量流数据的数量显示在指定的限制中。

快速过滤器

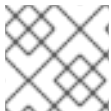
Quick 过滤器 下拉菜单中的默认值在 **FlowCollector** 配置中定义。您可从控制台修改选项。

高级过滤器

您可以通过从下拉列表中选择要过滤的参数来设置高级过滤器、**Common**、**Source** 或 **Destination**。流数据根据选择进行过滤。要启用或禁用应用的过滤器，您可以点过滤器选项下面列出的应用过滤器。

您可以切换 **↑ One way ↑** 和 **↓ Back and forth** 过滤。**↑ 单向**过滤只根据过滤器选择显示 **Source** 和 **Destination** 流量。您可以使用 **Swap** 来更改 **Source** 和 **Destination** 流量的方向视图。**↑ ↓ Back and forth** 过滤器包括带有 **Source** 和 **Destination** 过滤器的返回流量。网络流量的方向流在流量流表中的 **Direction** 列中显示为 **Ingress** or **Egress**（对于不同节点间的流量）和 'Inner'（对于单一节点内的流量）。

您可以点 **Reset default** 删除现有过滤器，并应用 **FlowCollector** 配置中定义的过滤器。



注意

要了解指定文本值的规则，请点[了解更多](#)。

另外，您也可以访问 **Namespaces, Services, Routes, Nodes, and Workloads** 页中的 **Network Traffic** 标签页，它们提供了相关部分的聚合过滤数据。

其他资源

有关在 **FlowCollector** 中配置快速过滤器的更多信息，请参阅[配置快速过滤器](#) 和 [流收集器示例资源](#)。

第 8 章 使用带有仪表板和警报的指标

Network Observability Operator 使用 **flowlogs-pipeline** 从流日志生成指标。您可以通过设置自定义警报和查看仪表板来使用这些指标。

8.1. 查看 NETWORK OBSERVABILITY 指标仪表板

在 OpenShift Container Platform 控制台中的 **Overview** 选项卡中，您可以查看集群中网络流量流的整体聚合指标。您可以选择按节点、命名空间、所有者、pod 和服务显示信息。您还可以使用过滤器和显示选项来进一步优化指标。

流程

1. 在 Web 控制台 **Observe** → **Dashboards** 中，选择 **Netobserv** 仪表板。
2. 查看以下类别中的网络流量指标，每个指标都有每个节点、命名空间、源和目标的子集：
 - 字节率
 - 数据包丢弃
 - DNS
 - RTT
3. 选择 **Netobserv/Health** 仪表板。
4. 在以下类别中查看有关 Operator 健康的指标，每个类别都有每个节点的子集、命名空间、源和目的地。
 - 流
 - 流开销
 - 流率
 - 代理
 - 处理器
 - Operator

基础架构和应用程序指标显示在命名空间和工作负载的 split-view 中。

8.2. 预定义的指标

flowlogs-pipeline 生成的指标可在 **FlowCollector** 自定义资源的 **spec.processor.metrics.includeList** 中进行配置，以添加或删除指标。

8.3. 网络 OBSERVABILITY 指标

您还可以使用 Prometheus 规则中的 **includeList** 指标创建警报，如"创建警报"所示。

在 Prometheus 中查找这些指标时，如通过 **Observe** → **Metrics** 或定义警报时，所有指标名称都有 **netobserv_** 前缀。例如 **netobserv_namespace_flows_total**。可用的指标名称如下：

includeList 指标名称

默认情况下启用带有星号 * 的名称。

- 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 指标名称

当在 `spec.agent.ebpf.features`（具有 `privileged` 模式）中启用 `PacketDrop` 功能时，可以使用以下额外的指标：

- 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 指标名称

当在 `spec.agent.ebpf.features` 中启用了 `DNSTracking` 功能时，可以使用以下额外的指标：

- namespace_dns_latency_seconds *
- node_dns_latency_seconds
- workload_dns_latency_seconds

FlowRTT 指标名称

当在 `spec.agent.ebpf.features` 中启用 **FlowRTT** 功能时，可以使用以下额外指标：

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

8.4. 创建警报

您可以为 Netobserv 仪表盘指标创建自定义警报规则，以便在满足某些定义条件时触发警报。

先决条件

- 您可以使用具有 `cluster-admin` 角色的用户访问集群，或者具有所有项目的查看权限。
- 已安装 Network Observability Operator。

流程

1. 点导入图标 + 创建 YAML 文件。
2. 向 YAML 文件添加警报规则配置。在以下 YAML 示例中，当集群入口流量达到每个目标工作负载的指定阈值 10 MBps 时，会为警报创建一个警报。

```

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 在 `spec.processor.metrics.includeList` 中默认启用 `netobserv_workload_ingress_bytes_total` 指标。

3. 点 **Create** 将配置文件应用到集群。

8.5. 自定义指标

您可以使用 **FlowMetric** API 从 flowlogs 数据中创建自定义指标。在收集的每个流日志数据中，每个日志都标记了多个字段，如源名称和目标名称。这些字段可以用作 Prometheus 标签，以便在仪表板上自定义集群信息。

8.6. 使用 FLOWMETRIC API 配置自定义指标

您可以使用 flowlogs data 字段作为 Prometheus 标签，将 **FlowMetric** API 配置为创建自定义指标。您可以在项目中添加多个 **FlowMetric** 资源，以查看多个仪表板视图。

流程

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题中，选择 **FlowMetric**。
3. 在 **Project:** 下拉列表中，选择 **Network Observability Operator** 实例的项目。
4. 点 **Create FlowMetric**。
5. 配置 **FlowMetric** 资源，类似于以下示例配置：

例 8.1. 生成一个指标，用于跟踪从集群外部源接收的 ingress 字节

```

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** **FlowMetric** 资源需要在 **FlowCollector spec.namespace** 中定义的命名空间中创建，默认为 **netobserv**。
- 2** Prometheus 指标的名称，它们在 web 控制台中的显示带有 **netobserv-<metricName>** 前缀。
- 3** **type** 指定指标类型。**Counter type** 有助于计算字节和数据包。
- 4** 要捕获的流方向。如果没有指定，入口和出口数据都会捕获，这可能会导致重复计数。
- 5** 标签定义了指标外，以及不同实体之间的关系，同时还定义了指标基数（cardinality）。例如，**SrcK8S_Name** 是一个高基数指标。
- 6** 根据列出的标准进一步细化结果。在本例中，通过仅匹配没有 **SrcSubnetLabel** 的流来实现只选择集群外部流量的目的。这假设启用了子网标签功能（通过

个例 8.1 中，我们指定了 `spec.processor.subnetLabels`，这是默认行为。

验证

1. pod 刷新后，进入 **Observe** → **Metrics**。
2. 在 **Expression** 字段中，键入指标名称来查看对应的结果。您也可以输入一个表达式，如 `topk(5, sum(rate(netobserv_cluster_external_ingress_bytes_total{DstK8S_Namespace="my-namespace"}[2m])) by (DstK8S_HostName, DstK8S_OwnerName, DstK8S_OwnerType))`

例 8.2. 显示集群外部入口流量的 RTT 延迟

```

apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-rtt
  namespace: netobserv ①
spec:
  metricName: cluster_external_ingress_rtt_seconds
  type: Histogram ②
  valueField: TimeFlowRttNs
  direction: Ingress
  labels:
  [DstK8S_HostName,DstK8S_Namespace,DstK8S_OwnerName,DstK8S_OwnerType]
  filters:
  - field: SrcSubnetLabel
    matchType: Absence
  - field: TimeFlowRttNs
    matchType: Presence
  divider: "1000000000" ③
  buckets: [".001", ".005", ".01", ".02", ".03", ".04", ".05", ".075", ".1", ".25", "1"] ④

```

- ① **FlowMetric** 资源需要在 **FlowCollector spec.namespace** 中定义的命名空间中创建，默认为 **netobserv**。
- ② **type** 指定指标类型。**Histogram type** 可用于延迟值 (**TimeFlowRttNs**)。
- ③ 由于在流中 Round-trip time (RTT) 的单位为纳秒，需要将这个值除以十亿 (1 billion) 转换为秒，这是 Prometheus 指南中的标准。
- ④ 自定义存储桶指定 RTT 的精度，其最佳精度范围介于 5ms 和 250ms 之间。

验证

1. pod 刷新后，进入 **Observe** → **Metrics**。
2. 在 **Expression** 字段中，键入指标名称来查看对应的结果。



重要

高基数可能会影响 Prometheus 的内存用量。您可以检查特定标签是否以有高基数 ([Network Flows 格式参考](#))。

8.7. 使用 FLOWMETRIC API 配置自定义 CHART

您可以在 OpenShift Container Platform web 控制台中为仪表板生成图表，如果您是管理员，可以通过定义 **FlowMetric** 资源的 **charts** 部分在 **Dashboard** 菜单中查看。

流程

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题中，选择 **FlowMetric**。
3. 在 **Project:** 下拉列表中，选择 **Network Observability Operator** 实例的项目。
4. 点 **Create FlowMetric**。
5. 配置 **FlowMetric** 资源，类似于以下示例配置：

例 8.3. 用于跟踪从集群外部源接收的 ingress 字节的图表

```

apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-traffic
  namespace: netobserv ❶
# ...
charts:
- dashboardName: Main ❷
  title: External ingress traffic
  unit: Bps
  type: SingleStat
  queries:
  - promQL: "sum(rate($METRIC[2m]))"
    legend: ""
- dashboardName: Main ❸
  sectionName: External
  title: Top external ingress traffic per workload
  unit: Bps
  type: StackArea
  queries:
  - promQL: "sum(rate($METRIC{DstK8S_Namespace!=\"\"}[2m])) by (DstK8S_Namespace, DstK8S_OwnerName)"
    legend: "{{DstK8S_Namespace}} / {{DstK8S_OwnerName}}"
# ...

```

- ❶ **FlowMetric** 资源需要在 **FlowCollector spec.namespace** 中定义的命名空间中创建，默认为 **netobserv**。

验证

1. pod 刷新后, 进入到 **Observe** → **Dashboards**。
2. 搜索 **NetObserv / Main** 仪表板。查看 **NetObserv / Main** 仪表板下的两个面板, 或您创建的仪表板名称 (可选) :
 - 一个静态的文本形式的统计数据, 显示所有维度中的全局外部入口率总和
 - 一个时间序列图, 为每个目标工作负载显示相同指标

有关查询语言的更多信息, 请参阅 [Prometheus 文档](#)。

例 8.4. 集群外部入口流量的 RTT 延迟的图表

```

apiVersion: flows.netobserv.io/v1alpha1
kind: FlowMetric
metadata:
  name: flowmetric-cluster-external-ingress-traffic
  namespace: netobserv ❶
# ...
charts:
- dashboardName: Main ❷
  title: External ingress TCP latency
  unit: seconds
  type: SingleStat
  queries:
  - promQL: "histogram_quantile(0.99, sum(rate($METRIC_bucket[2m])) by (le)) > 0"
    legend: "p99"
- dashboardName: Main ❸
  sectionName: External
  title: "Top external ingress sRTT per workload, p50 (ms)"
  unit: seconds
  type: Line
  queries:
  - promQL: "histogram_quantile(0.5, sum(rate($METRIC_bucket{DstK8S_Namespace!=\"\"}
[2m])) by (le,DstK8S_Namespace,DstK8S_OwnerName))*1000 > 0"
    legend: "{{DstK8S_Namespace}} / {{DstK8S_OwnerName}}"
- dashboardName: Main ❹
  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}}"
# ...

```

❶ **FlowMetric** 资源需要在 **FlowCollector spec.namespace** 中定义的命名空间中创建, 默认为 **netobserv**。

❷❸❹ 使用不同的 **dashboardName** 创建一个前缀为 **Netobserv** 的新仪表板。例如, **Netobserv / <dashboard_name>**。

这个示例使用 `histogram_quantile` 函数来显示 **p50** 和 **p99**。

您可以通过将指标 `$METRIC_sum` 除以 `$METRIC_count` 来显示平均直方图。它们在创建直方图时自动生成。在上例中，要执行此操作的 Prometheus 查询如下：

```
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"
```

验证

1. pod 刷新后，进入到 **Observe → Dashboards**。
2. 搜索 **NetObserv / Main** 仪表板。查看 **NetObserv / Main** 仪表板下的新面板，或您创建的仪表板名称（可选）。

有关查询语言的更多信息，请参阅 [Prometheus 文档](#)。

其他资源

- [为用户定义的项目创建警报规则](#)。
- [对高基数指标进行故障排除 - 确定为什么 Prometheus 消耗大量磁盘空间](#)

第 9 章 监控 NETWORK OBSERVABILITY OPERATOR

您可以使用 Web 控制台监控与 Network Observability Operator 健康相关的警报。

9.1. 健康仪表盘

Network Observability Operator 健康和资源使用情况的指标位于 web 控制台的 **Observe → Dashboards** 页面中。您可以按以下类别查看 Operator 健康状况的指标：

- Flows per second
- Sampling
- Errors last minute
- Dropped flows per second
- Flowlogs-pipeline statistics
- Flowlogs-pipeline statistics views
- eBPF agent statistics views
- Operator statistics
- 资源使用量

9.2. 健康警报

当触发警报时，您定向到仪表盘的健康警报横幅可能会出现在 **Network Traffic** 和 **Home** 页面中。在以下情况下生成警报：

- 如果 **flowlogs-pipeline** 工作负载因为 Loki 错误而丢弃流，如已经达到 Loki ingestion 速率限制，则 **NetObservLokiError** 警报发生。
- 如果在一个时间段内没有流，则会发出 **NetObservNoFlows** 警报。
- 如果 Network Observability eBPF 代理的 hashmap 表已满，并且 eBPF 代理处理性能下降或触发了容量限制器时，则会发出 **NetObservFlowsDropped** 警报。

9.3. 查看健康信息

您可从 web 控制台的 **Dashboards** 页面中访问 Network Observability Operator 健康和资源使用的指标。

先决条件

- 已安装 Network Observability Operator。
- 您可以使用具有 **cluster-admin** 角色或所有项目的查看权限的用户访问集群。

流程

1. 从 web 控制台中的 **Administrator** 视角，进入到 **Observe → Dashboards**。
2. 从 **Dashboards** 下拉菜单中选择 **Netobserv/Health**。

3. 查看页面中显示的 Operator 健康状况的指标。

9.3.1. 禁用健康警报

您可以通过编辑 **FlowCollector** 资源来选择不使用健康警报：

1. 在 Web 控制台中，进入到 **Operators → Installed Operators**。
2. 在 **NetObserv Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
3. 选择 **cluster**，然后选择 **YAML** 选项卡。
4. 添加 **spec.processor.metrics.disableAlerts** 来禁用健康警报，如下例所示：

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

- ❶ 您可以指定一个或多个包含要禁用的警报类型的列表。

9.4. 为 NETOBSERV 仪表盘创建 LOKI 速率限制警报

您可以为 **Netobserv** 仪表盘指标创建自定义警报规则，以便在达到 **Loki** 速率限制时触发警报。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群，或者具有所有项目的查看权限。
- 已安装 **Network Observability Operator**。

流程

1. 点导入图标 + 创建 **YAML** 文件。
2. 向 **YAML** 文件添加警报规则配置。在以下 **YAML** 示例中，当达到 **Loki** 速率限制时，会创建一个警报：

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: loki-alerts
  namespace: openshift-monitoring
spec:
  groups:
    - name: LokiRateLimitAlerts
      rules:
        - alert: LokiTenantRateLimit
          annotations:
            message: |-
```

```

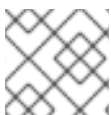
    {{ $labels.job }} {{ $labels.route }} is experiencing 429 errors.
    summary: "At any number of requests are responded with the rate limit error code."
    expr: sum(irate(loki_request_duration_seconds_count{status_code="429"}[1m])) by (job,
namespace, route) / sum(irate(loki_request_duration_seconds_count[1m])) by (job,
namespace, route) * 100 > 0
    for: 10s
    labels:
    severity: warning

```

- 点 **Create** 将配置文件应用到集群。

9.5. 使用 EBPF 代理警报

当 Network Observability eBPF 代理 hashmap 表已满或触发了容量限制器，则会发出警报 **NetObservAgentFlowsDropped**。如果您看到此警报，请考虑增加 **FlowCollector** 中的 **cacheMaxFlows**，如下例所示。



注意

增加 **cacheMaxFlows** 可能会增加 eBPF 代理的内存用量。

流程

- 在 Web 控制台中，进入到 **Operators → Installed Operators**。
- 在 **Network Observability Operator** 的 **Provided APIs** 标题下，选择 **Flow Collector**。
- 选择 **集群**，然后选择 **YAML** 选项卡。
- 增加 **spec.agent.ebpf.cacheMaxFlows** 值，如以下 YAML 示例所示：

```

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

```

- 将 **cacheMaxFlows** 从发出 **NetObservAgentFlowsDropped** 警报时的值增加到一个更高的值。

其他资源

- 有关在仪表板上创建警报的更多信息，请参阅 [为用户定义的项目创建警报规则](#)。

第 10 章 调度资源

通过污点和容限，节点可以控制哪些 pod 应该（或不应该）调度到节点上。

节点选择器指定一个键/值对映射，该映射使用 pod 中指定的自定义标签和选择器定义。

要使 pod 有资格在节点上运行，pod 必须具有与节点上标签相同的键值节点选择器。

10.1. 特定节点中的网络 OBSERVABILITY 部署

您可以配置 **FlowCollector** 来控制特定节点中的 Network Observability 组件的部署。spec.agent.ebpf.advanced.scheduling,spec.processor.advanced.scheduling, 和 spec.consolePlugin.advanced.scheduling 规格有以下可进行配置的设置：

- **NodeSelector**
- **容限 (Tolerations)**
- **关联性**
- **PriorityClassName**

spec.<component>.advanced.scheduling的 FlowCollector 资源示例

```
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: ""
  # ...
```

其他资源

- [了解污点和容限](#)

- [将 Pod 分配给节点](#) (Kubernetes 文档)
- [Pod 优先级和抢占](#) (Kubernetes 文档)

第 11 章 NETWORK OBSERVABILITY CLI

11.1. 安装 NETWORK OBSERVABILITY CLI

Network Observability CLI (**oc netobserv**) 与 Network Observability Operator 分开部署。CLI 可作为 OpenShift CLI (**oc**) 插件提供。它提供快速调试和对网络可观察性进行故障排除的轻量级方法。



重要

Network Observability CLI (**oc netobserv**) 只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议 (SLA) 支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅[技术预览功能支持范围](#)。

11.1.1. 关于 Network Observability CLI

您可以使用 Network Observability CLI (**oc netobserv**) 快速调试并排除网络问题。Network Observability CLI 是一个流和数据包视觉化工具，它依赖于 eBPF 代理将收集的数据流传输到临时收集器 pod。在捕获过程中不需要持久性存储。运行后，输出将传输到您的本地计算机。这可以实现快速了解数据包和流数据，而无需安装 Network Observability Operator。



重要

CLI 捕获只适用于在一个简短的持续时间段，如 8-10 分钟。如果运行时间过长，可能很难删除正在运行的进程。

11.1.2. 安装 Network Observability CLI

安装 Network Observability CLI (**oc netobserv**) 是与 Network Observability Operator 安装分开的步骤。这意味着，即使您有从 OperatorHub 安装 Operator，也需要单独安装 CLI。



注意

您可以选择使用 Krew 来安装 **netobserv** CLI 插件。如需更多信息，请参阅“使用 Krew 安装 CLI 插件”。

先决条件

- 您必须安装 OpenShift CLI (**oc**)。
- 您必须有一个 macOS 或 Linux 操作系统。

流程

1. 下载 [oc netobserv CLI tar 文件](#)。
2. 解包存档：

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

3. 使文件可执行：

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

4. 将提取的 **netobserv-cli** 二进制文件移到 **PATH** 上的目录中，如 **/usr/local/bin/**：

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

验证

- 验证 **oc netobserv** 是否可用：

```
$ oc netobserv version
```

输出示例

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

其他资源

- [安装和使用 CLI 插件](#)
- [使用 Krew 安装 CLI 插件](#)

11.2. 使用 NETWORK OBSERVABILITY CLI

您可以在终端中直接视觉化和过滤流和数据包数据，以查看特定使用情况，例如识别谁正在使用特定端口。Network Observability CLI 将流作为 JSON 和数据库文件或数据包作为 PCAP 文件来收集，该文件可用于第三方工具。

11.2.1. 捕获流

您可以捕获数据中的任何资源或区域的流程和过滤，以解决两个区域之间的 Round-Trip Time (RTT)。CLI 中的表视觉化提供了查看和流搜索功能。

先决条件

- 安装 OpenShift CLI (**oc**)。
- 安装 Network Observability CLI (**oc netobserv**) 插件。

流程

1. 运行以下命令，通过过滤捕获流：

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

2. 在终端的 **live table filter** 提示下添加过滤以进一步优化传入的流。例如：

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


3. 要停止捕获，按 **Ctrl+C**。捕获的数据被写入两个单独的文件，位于用于安装 CLI 的同一路径中的 **./output** 目录中。
4. 在 **./output/flow/<capture_date_time>.json** JSON 文件中查看捕获的数据，其中包含了捕获数据的 JSON 阵列。

JSON 文件示例

```
{
  "AgentIP": "10.0.1.76",
  "Bytes": 561,
  "DnsErrno": 0,
  "Dscp": 20,
  "DstAddr": "f904:ece9:ba63:6ac7:8018:1e5:7130:0",
  "DstMac": "0A:58:0A:80:00:37",
  "DstPort": 9999,
  "Duplicate": false,
  "Etype": 2048,
  "Flags": 16,
  "FlowDirection": 0,
  "IfDirection": 0,
  "Interface": "ens5",
  "K8S_FlowLayer": "infra",
  "Packets": 1,
  "Proto": 6,
  "SrcAddr": "3e06:6c10:6440:2:a80:37:b756:270f",
  "SrcMac": "0A:58:0A:80:00:01",
  "SrcPort": 46934,
  "TimeFlowEndMs": 1709741962111,
  "TimeFlowRttNs": 121000,
  "TimeFlowStartMs": 1709741962111,
  "TimeReceived": 1709741964
}
```

5. 您可以使用 SQLite 检查 **./output/flow/<capture_date_time>.db** 数据库文件。例如：
 - a. 运行以下命令打开该文件：

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

- b. 运行 SQLite **SELECT** 语句来查询数据，例如：

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

输出示例

```
12|NoError|58747|10.128.0.63|57856||17|172.30.0.10|53|284|1
11|NoError|20486|10.128.0.52|56575||17|169.254.169.254|53|225|1
11|NoError|59544|10.128.0.103|51089||17|172.30.0.10|53|307|1
13|NoError|32519|10.128.0.52|55241||17|169.254.169.254|53|254|1
12|NoError|32519|10.0.0.3|55241||17|169.254.169.254|53|254|1
15|NoError|57673|10.128.0.19|59051||17|172.30.0.10|53|313|1
13|NoError|35652|10.0.0.3|46532||17|169.254.169.254|53|183|1
```

```
32|NoError|37326|10.0.0.3|52718||17|169.254.169.254|53|169|1
14|NoError|14530|10.0.0.3|58203||17|169.254.169.254|53|246|1
15|NoError|40548|10.0.0.3|45933||17|169.254.169.254|53|174|1
```

11.2.2. 捕获数据包

您可以使用 Network Observability CLI 捕获数据包。

先决条件

- 安装 OpenShift CLI (**oc**)。
- 安装 Network Observability CLI (**oc netobserv**) 插件。

流程

1. 在启用了过滤的情况下运行数据包捕获：

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

2. 在终端的 **live table filter** 提示下添加过滤以进一步优化传入的数据包。过滤示例如下：

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

3. 要停止捕获，按 **Ctrl+C**。
4. 查看捕获的数据，这些数据被写入一个文件中，该文件位于用于安装 CLI 的同一路径中的 **./output/pcap** 目录中：
 - a. **./output/pcap/<capture_date_time>.pcap** 文件可以使用 wireshark 打开。

11.2.3. 清理 Network Observability CLI

您可以通过运行 **oc netobserv cleanup** 来手动清理 CLI 工作负载。此命令从集群中删除所有 CLI 组件。

当您结束捕获时，此命令由客户端自动运行。如果您遇到连接问题，可能需要手动运行它。

流程

- 运行以下命令：

```
$ oc netobserv cleanup
```

其他资源

- [Network Observability CLI 参考](#)

11.3. NETWORK OBSERVABILITY CLI (OC NETOBSERV) 参考

Network Observability CLI (**oc netobserv**) 具有 Network Observability Operator 所具有的大多数功能和过滤选项。您可以传递命令行参数来启用功能或过滤选项。

11.3.1. oc netobserv CLI 参考

Network Observability CLI (**oc netobserv**) 是一个用于捕获流数据和数据包数据的 CLI 工具，用于进一步分析。

oc netobserv 语法

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

- 1 功能选项只能与 **oc netobserv flow** 命令一起使用。它们不能与 **oc netobserv packets** 命令一起使用。

表 11.1. 基本命令

命令	描述
flows	捕获流信息。有关子命令，请参阅 "Flow 捕获子命令"表。
packets	从一个特定协议或端口对来捕获数据包，如 netobserv packets -- filter=tcp,80 。有关数据包捕获的更多信息，请参阅 "Packet 捕获子命令"表。
cleanup	删除 Network Observability CLI 组件。
version	打印软件版本。
帮助	显示帮助。

11.3.1.1. Network Observability 功能

Network Observability 增强了显示区域、节点、所有者和资源名称，包括有关数据包丢弃的可选功能，DNS 延迟和往返时间只能在捕获流时启用。这些不会在数据包捕获 pcap 输出文件中出现。

Network Observability 增强语法

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

表 11.2. Network Observability 增强选项

选项	描述	可能的值	default
--enable_pktdrop	启用数据包丢弃。	true, false	false
--enable_rtt	启用往返时间 (RTT) 。	true, false	false
--enable_dns	启用 DNS 跟踪。	true, false	false
--help	显示帮助。	-	-

选项	描述	可能的值	default
--interfaces	在流上匹配的接口。例如, "eth0,eth1"。	"<interface>"	-

11.3.1.2. 流捕获选项

流捕获有强制命令以及附加选项, 如启用有关数据包丢弃、DNS 延迟、往返时间和过滤的额外功能。

oc netobserv flows 语法

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

表 11.3. 流捕获过滤器选项

选项	描述	可能的值	必需	default
--enable_filter	启用流过滤。	true, false	是	false
--action	应用流的操作。	Accept, Reject	是	Accept
--cidr	在流上匹配的 CIDR。	1.1.1.0/24, 1::100/64, 或 0.0.0.0/0	是	0.0.0.0/0
--protocol	在流上匹配的协议	TCP, UDP, SCTP, ICMP, or ICMPv6	否	-
--direction	在流上匹配的方向	Ingress, Egress	否	-
--dport	在流上匹配的目的地端口。	80,443, 或 49051	否	-
--sport	在流上匹配的源端口。	80,443, 或 49051	否	-
--port	在流上匹配的端口。	80,443, 或 49051	否	-
--sport_range	在流上匹配的源端口范围。	80-100 或 443-445	否	-
--dport_range	要在流上匹配的目标端口范围。	80-100	否	-
--port_range	在流上匹配的端口范围。	80-100 或 443-445	否	-

选项	描述	可能的值	必需	default
--icmp_type	在流上匹配的 ICMP 类型。	8 或 13	否	-
--icmp_code	在流上匹配的 ICMP 代码。	0 或 1	否	-
--peer_ip	在流上匹配的对等 IP。	1.1.1.1 或 1::1	否	-

11.3.1.3. 数据包捕获选项

您可以在端口和协议中过滤数据包捕获数据。

oc netobserv packets 语法

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

表 11.4. 数据包捕获过滤选项

选项	描述	必需	default
<protocol>,<port>	从一个特定协议和端口对捕获数据包。使用逗号作为分隔符。例如： tcp,80 代表 tcp 协议和端口 80 。	是	-

第 12 章 FLOWCOLLECTOR 配置参数

FlowCollector 是网络流集合 API 的 Schema，它试用并配置底层部署。

12.1. FLOWCOLLECTOR API 规格

描述

FlowCollector 是网络流集合 API 的 schema，它试用并配置底层部署。

类型

object

属性	类型	描述
apiVersion	字符串	APIVersion 定义对象的这个表示法的版本化的 schema。服务器应该将识别的模式转换为最新的内部值，并可拒绝未识别的值。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
kind	字符串	kind 是一个字符串值，代表此对象所代表的 REST 资源。服务器可以从客户端向其提交请求的端点推断。无法更新。采用驼峰拼写法 (CamelCase)。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds
metadata	object	标准对象元数据。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata
spec	object	定义 FlowCollector 资源所需状态。 *：在本文中声明的"不支持"或"弃用"的功能代表红帽不正式支持这些功能。例如，这些功能可能由社区提供，并在没有正式维护协议的情况下被接受。产品维护人员可能只为这些功能提供一些支持。

12.1.1. .metadata

描述

标准对象元数据。更多信息：<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

类型

object

12.1.2. .spec

描述

定义 FlowCollector 资源所需状态。

*：在本文档中声明的"不支持"或"弃用"的功能代表红帽不正式支持这些功能。例如，这些功能可能由社区提供，并在没有正式维护协议的情况下被接受。产品维护人员可能只为这些功能提供一些支持。

类型

object

属性	类型	描述
agent	object	流提取的代理配置。
consolePlugin	object	ConsolePlugin 定义与 OpenShift Container Platform 控制台插件相关的设置（如果可用）。
deploymentModel	string	<p>deploymentModel 定义所需的用于流处理的部署类型。可能的值有：</p> <ul style="list-style-type: none"> - Direct（默认）直接从代理监听流处理器。 - Kafka 在处理器使用前，将流发送到 Kafka 管道。 <p>Kafka 可以提供更好的可扩展性、弹性和高可用性（更多详情，请参阅 https://www.redhat.com/en/topics/integration/what-is-apache-kafka）。</p>
exporters	数组	exporters 为自定义消耗或存储定义了额外的可选导出器。

属性	类型	描述
kafka	object	Kafka 配置，允许使用 Kafka 作为流集合管道的一部分。当 spec.deploymentModel 是 Kafka 时可用。
loki	object	loki ，流存储、客户端设置。
namespace	string	部署 Network Observability pod 的命名空间。
processor	object	processor 定义从代理接收流的组件设置，增强它们，生成指标，并将它们转发到 Loki 持久层和/或任何可用的导出器。
prometheus	object	prometheus 定义 Prometheus 设置，如用于从控制台插件获取指标的 querier 配置。

12.1.3. .spec.agent

描述

流提取的代理配置。

类型

object

属性	类型	描述
ebpf	object	ebpf 描述了当 spec.agent.type 设置为 eBPF 时，与基于 eBPF 的流报告程序相关的设置。
type	string	type [弃用 (*)] 选择流追踪代理。在以前的版本中，此字段可以用于选择使用 eBPF 或 IPFIX 。现在，因为只允许使用 eBPF ，此字段已弃用，计划在以后的 API 版本中删除。

12.1.4. .spec.agent.ebpf

描述

ebpf 描述了当 **spec.agent.type** 设置为 **eBPF** 时，与基于 eBPF 的流报告程序相关的设置。

类型

object

属性	类型	描述
advanced	object	advanced 允许设置 eBPF 代理的内部配置的一些方面。本节主要用于调试和精细的性能优化，如 GOGC 和 GOMAXPROCS 环境变量。在设置这些值时，您需要自己考虑相关的风险。
cacheActiveTimeout	string	cacheActiveTimeout 是报告者在发送前聚合流的最大周期。增加 cacheMaxFlows 和 cacheActiveTimeout 可能会降低网络流量开销和 CPU 负载，但您可以预期更高的内存消耗和流集中的延迟增加。
cacheMaxFlows	整数	cacheMaxFlows 是聚合中的最大流数；达到时，报告者会发送流。增加 cacheMaxFlows 和 cacheActiveTimeout 可能会降低网络流量开销和 CPU 负载，但您可以预期更高的内存消耗和流集中的延迟增加。
excludeInterfaces	数组（字符串）	excludeInterfaces 包含从流追踪中排除的接口名称。条目用斜杠括起，如 /br- ，与正则表达式匹配。否则，它将匹配为区分大小写的字符串。
功能	数组（字符串）	要启用的额外功能列表。它们都默认禁用。启用其他功能可能会对性能有影响。可能的值有： PacketDrop : 启用数据包丢弃日志功能。此功能需要挂载内核调试文件系统，因此 eBPF pod 必须以特权方式运行。如果没有设置 spec.agent.ebpf.privileged 参数，会报告一个错误。 - DNSTracking : 启用 DNS 跟踪功能。 - FlowRTT : 从 TCP 流量在 eBPF 代理中启用流延迟 (sRTT)。
flowFilter	object	flowFilter 定义有关流过滤的 eBPF 代理配置。
imagePullPolicy	string	imagePullPolicy 是上面定义的镜像的 Kubernetes pull 策略

属性	类型	描述
interfaces	数组（字符串）	接口 包含从中收集流的接口名称。如果为空，代理会获取系统中的所有接口，但 excludeInterfaces 中列出的接口除外。条目用斜杠括起，如 /br- ，与正则表达式匹配。否则，它将匹配为区分大小写的字符串。
kafkaBatchSize	整数	kafkaBatchSize 在发送到分区前限制请求的最大大小（以字节为单位）。如果不使用 Kafka，则忽略。默认：1MB。
logLevel	string	LogLevel 定义 Network Observability eBPF 代理的日志级别
metrics	object	metrics 定义了有关指标的 eBPF 代理配置。
privileged	布尔值	eBPF Agent 容器的特权模式。当忽略或设置为 false 时，Operator 会将粒度功能(BPF、PERFMON、NET_ADMIN、SYS_RESOURCE) 设置为容器。如果出于某种原因而无法设置这些功能，例如，如果旧的内核版本不知道 CAP_BPF，那么您可以打开此模式以获取更多全局权限。有些代理功能需要特权模式，如数据包丢弃跟踪（请参阅 功能 ）和 SR-IOV 支持。
resources	object	resources 是此容器所需的计算资源。如需更多信息，请参阅 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
sampling	整数	流报告器的抽样率。100 表示发送 100 个流中的一个。0 或 1 表示所有流都是抽样的。

12.1.5. .spec.agent.ebpf.advanced

描述

advanced 允许设置 eBPF 代理的内部配置的一些方面。本节主要用于调试和精细的性能优化，如 **GOGC** 和 **GOMAXPROCS** 环境变量。在设置这些值时，您需要自己考虑相关的风险。

类型

object

属性	类型	描述
env	对象（字符串）	env 允许将自定义环境变量传递给底层组件。对于传递一些非常严格的性能调整选项（如 GOGC 、 GOMAXPROCS ）非常有用，它们不应作为 FlowCollector 描述符的一部分公开，因为它们仅在边缘调试或支持场景中有用。
scheduling	object	调度控制如何在节点上调度 pod。

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

描述

调度控制如何在节点上调度 pod。

类型

object

属性	类型	描述
关联性	object	如果指定，pod 的调度限制。有关文档，请参阅 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling 。
nodeSelector	对象（字符串）	nodeSelector 只允许将 pod 调度到具有每个指定标签的节点上。有关文档，请参阅 https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ 。
priorityClassName	string	如果指定，指示 pod 的优先级。有关文档，请参阅 https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption 。如果没有指定，则使用默认优先级，如果没有默认值，则使用零。

属性	类型	描述
容限 (tolerations)	数组	tolerations 是一个容限 (toleration) 列表, 允许 pod 调度到具有匹配污点的节点。有关文档, 请参阅 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling 。

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

描述

如果指定, pod 的调度限制。有关文档, 请参阅 <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling>。

类型

object

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

描述

tolerations 是一个容限 (toleration) 列表, 允许 pod 调度到具有匹配污点的节点。有关文档, 请参阅 <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling>。

类型

数组

12.1.9. .spec.agent.ebpf.flowFilter

描述

flowFilter 定义有关流过滤的 eBPF 代理配置。

类型

object

属性	类型	描述
action	string	action 定义对与过滤匹配的流执行的操作。
cidr	string	cidr 定义用于过滤流的 IP CIDR。示例： 10.10.10.0/24 或 100:100:100:100::/64

属性	类型	描述
destPorts	integer-or-string	destPorts 定义要过滤流的目标端口。要过滤一个单一端口，使用一个整数值。例如： destPorts: 80 。要过滤一系列端口，使用"开始-结束"范围形式的字符串。例如： destPorts: "80-100" 。
direction	string	direction 定义要过滤流的方向。
enable	布尔值	将 enable 设置为 true 以启用 eBPF 流过滤功能。
icmpCode	整数	icmpCode ，用于互联网控制消息协议(ICMP)流量，定义要过滤流的 ICMP 代码。
icmpType	整数	icmptype ，ICMP 流量定义要过滤流的 ICMP 类型。
peerIP	string	peerIP 定义要过滤流的 IP 地址。示例： 10.10.10.10 。
ports	integer-or-string	ports 定义要过滤流的端口。它用于源和目标端口。要过滤一个单一端口，使用一个整数值。例如： ports: 80 。要过滤一系列端口，使用"开始-结束"范围形式的字符串。例如： ports: "80-100" 。
protocol	string	protocol 定义要过滤流的协议。
sourcePorts	integer-or-string	sourcePorts 定义要过滤流的源端口。要过滤一个单一端口，使用一个整数值。例如： sourcePorts: 80 。要过滤一系列端口，使用"开始-结束"范围形式的字符串。例如： sourcePorts: "80-100" 。

12.1.10. .spec.agent.ebpf.metrics

描述

metrics 定义了有关指标的 eBPF 代理配置。

类型

object

属性	类型	描述
disableAlerts	数组（字符串）	disableAlerts 是应禁用的警报列表。可能的值有： NetObservDroppedFlows ，它会在 eBPF 代理丢弃流时触发，例如当 BPF hashmap 已满或触发容量限制器时。
enable	布尔值	将 enable 设置为 false 以禁用 eBPF 代理指标集合。它会被默认启用。
server	object	Prometheus scraper 的指标服务器端点配置。

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

描述

Prometheus scraper 的指标服务器端点配置。

类型

object

属性	类型	描述
port	整数	指标服务器 HTTP 端口。
tls	object	TLS 配置。

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

描述

TLS 配置。

类型

object

属性	类型	描述
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过提供的证书的客户端侧验证。如果设置为 true ，则忽略 providedCaFile 字段。
provided	object	当 type 设置为 Provided 时的 TLS 配置。

属性	类型	描述
providedCaFile	object	当将 type 设置为 Provided 时，对 CA 文件的引用。
type	string	选择 TLS 配置类型： - Disabled （默认）没有为端点配置 TLS。- Provided 来手动提供证书文件和密钥文件。[不支持 (*)]。- Auto 使用 OpenShift Container Platform 字段生成的证书，用于注解。

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

描述

当 **type** 设置为 **Provided** 时的 TLS 配置。

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

当将 **type** 设置为 **Provided** 时，对 CA 文件的引用。

类型

object

属性	类型	描述
file	string	配置映射或 secret 中的文件名。
name	string	包含该文件的配置映射或 secret 的名称。
namespace	string	包含该文件的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	文件引用的 type: "configmap" 或 "secret"。

12.1.15. .spec.agent.ebpf.resources**描述**

resources 是此容器所需的计算资源。如需更多信息，请参阅

<https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

类型**object**

属性	类型	描述
limits	integer-or-string	限制描述了允许的最大计算资源量。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	integer-or-string	Requests 描述了所需的最少计算资源。如果容器省略了 Requests，则默认为 Limits（如果明确指定），否则默认为实现定义的值。请求不能超过限值。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

12.1.16. .spec.consolePlugin**描述**

ConsolePlugin 定义与 OpenShift Container Platform 控制台插件相关的设置（如果可用）。

类型

object

属性	类型	描述
advanced	object	advanced 允许设置控制台插件的内部配置的一些方面。本节主要用于调试和精细的性能优化，如 GOGC 和 GOMAXPROCS 环境变量。在设置这些值时，您需要自己考虑相关的风险。
autoscaler	object	Pod 横向自动扩展的 autoscaler 规格来为插件部署设置。请参阅 HorizontalPodAutoscaler 文档 (autoscaling/v2)。
enable	布尔值	启用 console 插件部署。
imagePullPolicy	string	imagePullPolicy 是上面定义的镜像的 Kubernetes pull 策略
logLevel	string	控制台插件后端的 logLevel
portNaming	object	portNaming 定义端口到服务名称转换的配置
quickFilters	数组	quickFilters 为 Console 插件配置快速过滤器预设置
replicas	整数	replicas 定义要启动的副本 (pod) 数。
resources	object	resources ，根据此容器所需的计算资源。如需更多信息，请参阅 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

12.1.17. .spec.consolePlugin.advanced

描述

advanced 允许设置控制台插件的内部配置的一些方面。本节主要用于调试和精细的性能优化，如 **GOGC** 和 **GOMAXPROCS** 环境变量。在设置这些值时，您需要自己考虑相关的风险。

类型

object

属性	类型	描述
args	数组 (字符串)	args 允许将自定义参数传递给底层组件。对于覆盖某些参数 (如 <code>url</code> 或配置路径) 非常有用, 它们不应作为 <code>FlowCollector</code> 描述符的一部分公开, 因为它们仅在边缘调试或支持场景中有用。
env	对象 (字符串)	env 允许将自定义环境变量传递给底层组件。对于传递一些非常严格的性能调整选项 (如 GOGC 、 GOMAXPROCS) 非常有用, 它们不应作为 <code>FlowCollector</code> 描述符的一部分公开, 因为它们仅在边缘调试或支持场景中有用。
port	整数	port 是插件服务端口。不要使用 9002, 它为指标保留。
register	布尔值	register 允许 (设置为 true 时), 在 OpenShift Container Platform Console Operator 中自动注册提供的控制台插件。当设置为 false 时, 您仍然可以使用以下命令编辑 <code>console.operator.openshift.io/cluster</code> 来手动注册它: oc patch console.operator.openshift.io cluster --type='json' -p '[{"op": "add", "path": "/spec/plugins/-", "value": "netobserv-plugin"}]'
scheduling	object	scheduling 控制如何在节点上调度 pod。

12.1.18. .spec.consolePlugin.advanced.scheduling

描述

scheduling 控制如何在节点上调度 pod。

类型

object

属性	类型	描述
----	----	----

属性	类型	描述
关联性	object	如果指定，pod 的调度限制。有关文档，请参阅 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling 。
nodeSelector	对象（字符串）	nodeSelector 只允许将 pod 调度到具有每个指定标签的节点上。有关文档，请参阅 https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ 。
priorityClassName	string	如果指定，指示 pod 的优先级。有关文档，请参阅 https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption 。如果没有指定，则使用默认优先级，如果没有默认值，则使用零。
容限（tolerations）	数组	tolerations 是一个容限 (toleration) 列表，允许 pod 调度到具有匹配污点的节点。有关文档，请参阅 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling 。

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

描述

如果指定，pod 的调度限制。有关文档，请参阅 <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling>。

类型

object

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

描述

tolerations 是一个容限 (toleration) 列表，允许 pod 调度到具有匹配污点的节点。有关文档，请参阅 <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling>。

类型

数组

12.1.21. .spec.consolePlugin.autoscaler

描述

Pod 横向自动扩展的 **autoscaler** 规格来为插件部署设置。请参阅 `HorizontalPodAutoscaler` 文档 (`autoscaling/v2`)。

类型

object

12.1.22. .spec.consolePlugin.portNaming**描述**

portNaming 定义端口到服务名称转换的配置

类型

object

属性	类型	描述
enable	布尔值	启用 console 插件端口到服务名称转换
portNames	对象 (字符串)	portNames 定义了控制台使用的额外端口名称, 例如 portNames: {"3100": "loki"} 。

12.1.23. .spec.consolePlugin.quickFilters**描述**

quickFilters 为 Console 插件配置快速过滤器预设置

类型

数组

12.1.24. .spec.consolePlugin.quickFilters[]**描述**

QuickFilter 为控制台的快速过滤器定义预设置配置

类型

object

必填

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

属性	类型	描述
default	布尔值	default 定义默认是否应激活此过滤器

属性	类型	描述
filter	对象 (字符串)	filter 是一组在选择此过滤器时要设置的键和值。每个键都可以与使用组合字符串的值列表相关，例如 filter: {"src_namespace": "namespace1,namespace2"} 。
name	string	过滤器的名称，在控制台中显示

12.1.25. .spec.consolePlugin.resources

描述

resources，根据此容器所需的计算资源。如需更多信息，请参阅 <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

类型

object

属性	类型	描述
limits	integer-or-string	限制描述了允许的最大计算资源量。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	integer-or-string	Requests 描述了所需的最少计算资源。如果容器省略了 Requests，则默认为 Limits（如果明确指定），否则默认为实现定义的值。请求不能超过限值。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

12.1.26. .spec.exporters

描述

exporters 为自定义消耗或存储定义了额外的可选导出器。

类型

数组

12.1.27. .spec.exporters[]

描述

FlowCollectorExporter 定义了一个额外的导出器来发送增强的流

类型

object

必填

- **type**

属性	类型	描述
ipfix	object	IPFIX 配置，如 IP 地址和端口，以将增强的 IPFIX 流发送到。
kafka	object	Kafka 配置（如地址和主题）将增强的流发送到。
type	string	type 选择导出器类型。可用的选项有 Kafka 和 IPFIX 。

12.1.28. .spec.exporters[].ipfix

描述

IPFIX 配置，如 IP 地址和端口，以将增强的 IPFIX 流发送到。

类型

object

必填

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

属性	类型	描述
targetHost	string	IPFIX 外部接收器的地址
targetPort	整数	IPFIX 外部接收器的端口
传输	string	用于 IPFIX 连接的传输协议(TCP 或 UDP)，默认为 TCP 。

12.1.29. .spec.exporters[].kafka

描述

Kafka 配置（如地址和主题）将增强的流发送到。

类型

object

必填

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

属性	类型	描述
address	string	Kafka 服务器的地址
sasl	object	SASL 身份验证配置。 [Unsupported packagemanifests]。
tls	object	TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。
topic	string	要使用的 Kafka 主题。它必须存在。Network Observability 不会创建它。

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

描述

SASL 身份验证配置。[Unsupported packagemanifests]。

类型

object

属性	类型	描述
clientIDReference	object	对包含客户端 ID 的 secret 或配置映射的引用
clientSecretReference	object	对包含客户端 secret 的 secret 或配置映射的引用
type	string	使用的 SASL 身份验证类型，如果没有使用 SASL，则为 Disabled

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

描述

对包含客户端 ID 的 secret 或配置映射的引用

类型

object

属性	类型	描述
file	string	配置映射或 secret 中的文件名。
name	string	包含该文件的配置映射或 secret 的名称。
namespace	string	包含该文件的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	文件引用的 type: "configmap" 或 "secret"。

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

描述

对包含客户端 secret 的 secret 或配置映射的引用

类型

object

属性	类型	描述
file	string	配置映射或 secret 中的文件名。
name	string	包含该文件的配置映射或 secret 的名称。
namespace	string	包含该文件的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	文件引用的 type: "configmap" 或 "secret"。

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

描述

TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。

类型

object

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true ，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

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

描述

caCert 定义证书颁发机构的证书引用

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

12.1.36. .spec.kafka

描述

Kafka 配置，允许使用 Kafka 作为流集合管道的一部分。当 **spec.deploymentModel** 是 **Kafka** 时可用。

类型

object

必填

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

属性	类型	描述
address	string	Kafka 服务器的地址
sasl	object	SASL 身份验证配置。 [Unsupported packagemanifests]。

属性	类型	描述
tls	object	TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。
topic	string	要使用的 Kafka 主题。它必须存在。Network Observability 不会创建它。

12.1.37. .spec.kafka.sasl

描述

SASL 身份验证配置。[Unsupported packagemanifests]。

类型

object

属性	类型	描述
clientIDReference	object	对包含客户端 ID 的 secret 或配置映射的引用
clientSecretReference	object	对包含客户端 secret 的 secret 或配置映射的引用
type	string	使用的 SASL 身份验证类型，如果没有使用 SASL，则为 Disabled

12.1.38. .spec.kafka.sasl.clientIDReference

描述

对包含客户端 ID 的 secret 或配置映射的引用

类型

object

属性	类型	描述
file	string	配置映射或 secret 中的文件名。
name	string	包含该文件的配置映射或 secret 的名称。

属性	类型	描述
namespace	string	包含该文件的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	文件引用的 type: "configmap" 或 "secret"。

12.1.39. .spec.kafka.sasl.clientSecretReference

描述

对包含客户端 secret 的 secret 或配置映射的引用

类型

object

属性	类型	描述
file	string	配置映射或 secret 中的文件名。
name	string	包含该文件的配置映射或 secret 的名称。
namespace	string	包含该文件的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	文件引用的 type: "configmap" 或 "secret"。

12.1.40. .spec.kafka.tls

描述

TLS 客户端配置。在使用 TLS 时，验证地址是否与用于 TLS 的 Kafka 端口匹配，通常为 9093。

类型

object

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true ，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

12.1.41. .spec.kafka.tls.caCert

描述

caCert 定义证书颁发机构的证书引用

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

12.1.42. .spec.kafka.tls.userCert

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

12.1.43. .spec.loki

描述

loki，流存储、客户端设置。

类型

object

属性	类型	描述
advanced	object	advanced 允许设置 Loki 客户端的内部配置的一些方面。本节主要用于调试和精细的性能优化。

属性	类型	描述
enable	布尔值	将 enable 设置为 true 以在 Loki 中存储流。Console 插件可以使用 Loki 或 Prometheus（或两者）作为指标的数据源（请参阅 spec.prometheus.querier ）。并非所有查询都可从 Loki 转换为 Prometheus。因此，如果 Loki 被禁用，插件的一些功能也会被禁用，如获取每个 pod 的信息或查看原始流。如果启用了 Prometheus 和 Loki，Prometheus 会优先使用，Loki 作为 Prometheus 无法处理的查询的回退系统。如果两者都被禁用，则不会部署 Console 插件。
lokiStack	object	LokiStack 模式的 Loki 配置。这对简单的 Loki Operator 配置很有用。对于其他模式，它会被忽略。
manual	object	Manual 模式的 Loki 配置。这是最灵活的配置。对于其他模式，它会被忽略。
微服务	object	对于 Microservices 模式的 Loki 配置。当使用微服务部署模式 (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode) 安装 Loki 时使用这个选项。对于其他模式，它会被忽略。
模式	string	mode 必须根据 Loki 的安装模式设置： <ul style="list-style-type: none"> - 当 Loki 通过 Loki Operator 管理时使用 LokiStack - 当 Loki 作为单体工作负载安装时，使用 Monolithic - 当 Loki 作为微服务安装，但没有 Loki Operator 时，使用 Microservices - 如果以上选项与您的设置都不匹配，则使用 Manual

属性	类型	描述
monolithic	object	Monolithic 模式的 Loki 配置。当使用单体部署模式 (https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode) 安装 Loki 时使用这个选项。对于其他模式，它会被忽略。
readTimeout	string	readTimeout 是最大控制台插件 loki 查询总时间的限制。超时值为零表示没有超时。
writeBatchSize	整数	writeBatchSize 是在发送前累积的 Loki 日志的最大批处理大小 (以字节为单位)。
writeBatchWait	string	writeBatchWait 是发送 Loki 批处理前等待的最长时间。
writeTimeout	string	writeTimeout 是最大 Loki 时间连接 / 请求限制。超时值为零表示没有超时。

12.1.44. .spec.loki.advanced

描述

advanced 允许设置 Loki 客户端的内部配置的一些方面。本节主要用于调试和精细的性能优化。

类型

object

属性	类型	描述
staticLabels	对象 (字符串)	staticLabels 是 Loki 存储中的每个流上设置的通用标签映射。
writeMaxBackoff	string	writeMaxBackoff 是 Loki 客户端连接在重试之间的最大 backoff 时间。
writeMaxRetries	整数	writeMaxRetries 是 Loki 客户端连接的最大重试次数。
writeMinBackoff	string	writeMinBackoff 是 Loki 客户端连接在重试之间的初始 backoff 时间。

12.1.45. .spec.loki.lokiStack

描述

LokiStack 模式的 Loki 配置。这对简单的 Loki Operator 配置很有用。对于其他模式，它会被忽略。

类型

object

属性	类型	描述
name	string	要使用的现有 LokiStack 资源的名称。
namespace	string	此 LokiStack 资源所在的命名空间。如果省略，则假定它与 spec.namespace 相同。

12.1.46. .spec.loki.manual

描述

Manual 模式的 Loki 配置。这是最灵活的配置。对于其他模式，它会被忽略。

类型

object

属性	类型	描述
authToken	string	<p>authToken 描述了获取与 Loki 进行身份验证的令牌的方法。</p> <ul style="list-style-type: none"> - Disabled 请求不发送任何令牌。 - Forward 转发用户令牌用于验证。 - Host [已启用 (*)] - 使用本地 pod 服务帐户用于 Loki 进行身份验证。 <p>使用 Loki Operator 时，必须设置为 Forward。</p>

属性	类型	描述
ingesterUrl	string	ingesterUrl 是现有 Loki ingester 服务的地址，用于将流推送到。使用 Loki Operator 时，使用路径中设置的 network 租户将其设置为 Loki 网关服务，例如 https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network 。
querierUrl	string	querierUrl 指定 Loki querier 服务的地址。使用 Loki Operator 时，使用路径中设置的 network 租户将其设置为 Loki 网关服务，例如 https://loki-gateway-http.netobserv.svc:8080/api/logs/v1/network 。
statusTls	object	Loki 状态 URL 的 TLS 客户端配置。
statusUrl	string	statusUrl 指定 Loki /ready 、 /metrics 和 /config 端点的地址，如果它与 Loki querier URL 不同。如果为空，则使用 querierUrl 值。这可用于在前端中显示错误消息和一些上下文。使用 Loki Operator 时，将其设置为 Loki HTTP 查询前端服务，例如 https://loki-query-frontend-http.netobserv.svc:3100/ 。当设置 statusUrl 时，使用 statusTLS 配置。
tenantID	string	tenantId 是 Loki X-Scope-OrgID ，用于标识每个请求的租户。使用 Loki Operator 时，将其设置为 network ，这对应于一个特殊的租户模式。
tls	object	Loki URL 的 TLS 客户端配置。

12.1.47. .spec.loki.manual.statusTls

描述

Loki 状态 URL 的 TLS 客户端配置。

类型

object

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true ，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

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

描述

caCert 定义证书颁发机构的证书引用

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

12.1.50. .spec.loki.manual.tls

描述

Loki URL 的 TLS 客户端配置。

类型

object

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true ，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

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

描述

caCert 定义证书颁发机构的证书引用

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。

属性	类型	描述
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

12.1.53. .spec.loki.microservices

描述

对于 **Microservices** 模式的 Loki 配置。当使用微服务部署模式 (<https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#microservices-mode>) 安装 Loki 时使用这个选项。对于其他模式，它会被忽略。

类型

object

属性	类型	描述
ingesterUrl	string	ingesterUrl 是现有 Loki ingester 服务的地址，用于将流推送到。
querierUrl	string	querierUrl 指定 Loki querier 服务的地址。
tenantID	string	tenantID 是 Loki X-Scope-OrgID 标头，用于标识每个请求的租户。
tls	object	Loki URL 的 TLS 客户端配置。

12.1.54. .spec.loki.microservices.tls

描述

Loki URL 的 TLS 客户端配置。

类型

object

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用

属性	类型	描述
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true ，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

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

描述

caCert 定义证书颁发机构的证书引用

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

12.1.57. .spec.loki.monolithic

描述

Monolithic 模式的 Loki 配置。当使用单体部署模式 (<https://grafana.com/docs/loki/latest/fundamentals/architecture/deployment-modes/#monolithic-mode>) 安装 Loki 时使用这个选项。对于其他模式，它会被忽略。

类型

object

属性	类型	描述
tenantID	string	tenantID 是 Loki X-Scope-OrgID 标头，用于标识每个请求的租户。
tls	object	Loki URL 的 TLS 客户端配置。
url	string	url 是现有 Loki 服务的唯一地址，它同时指向 ingester 和 querier。

12.1.58. .spec.loki.monolithic.tls

描述

Loki URL 的 TLS 客户端配置。

类型

object

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true ，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

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

描述

caCert 定义证书颁发机构的证书引用

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

12.1.61. .spec.processor

描述

processor 定义从代理接收流的组件设置，增强它们，生成指标，并将它们转发到 Loki 持久层和/或任何可用的导出器。

类型

object

属性	类型	描述
addZone	布尔值	addZone 通过使用其源和目标区标记流来允许可用区感知。此功能要求在节点上设置 "topology.kubernetes.io/zone" 标签。
advanced	object	advanced 允许设置流处理器的内部配置。本节主要用于调试和精细的性能优化，如 GOGC 和 GOMAXPROCS 环境变量。在设置这些值时，您需要自己考虑相关的风险。

属性	类型	描述
clusterName	string	clusterName 是要出现在流数据中的集群名称。这在多集群上下文中很有用。使用 OpenShift Container Platform 时，留空，使其自动决定。
imagePullPolicy	string	imagePullPolicy 是上面定义的镜像的 Kubernetes pull 策略
kafkaConsumerAutoscaler	object	kafkaConsumerAutoscaler 是 Pod 横向自动扩展的 spec，用于 flowlogs-pipeline-transformer ，它使用 Kafka 信息。当 Kafka 被禁用时，会忽略此设置。请参阅 HorizontalPodAutoscaler 文档 (autoscaling/v2) 。
kafkaConsumerBatchSize	整数	kafkaConsumerBatchSize 表示代理消费者接受的最大批处理大小（以字节为单位）。如果不使用 Kafka，则忽略。默认：10MB。
kafkaConsumerQueueCapacity	整数	kafkaConsumerQueueCapacity 定义 Kafka 消费者客户端中使用的内部消息队列的容量。如果不使用 Kafka，则忽略。
kafkaConsumerReplicas	整数	kafkaConsumerReplicas 定义了为 flowlogs-pipeline-transformer 启动的副本数，它使用 Kafka 信息。当 Kafka 被禁用时，会忽略此设置。
logLevel	string	处理器运行时的 logLevel

属性	类型	描述
logTypes	string	logTypes 定义要生成的记录类型。可能的值有： <ul style="list-style-type: none"> - Flows (默认) 导出常规网络流 - Conversations 为开始的对话、结束的对话以及定期的"tick"更新生成事件 - EndedConversations 只生成结束的对话事件 - All 生成网络流和所有对话事件
metrics	object	Metrics 定义有关指标的处理器配置
multiClusterDeployment	布尔值	将 multiClusterDeployment 设置为 true 以启用多集群功能。这会为流数据添加 clusterName 标签
resources	object	resources 是此容器所需的计算资源。如需更多信息，请参阅 https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
subnetLabels	object	subnetLabels 允许在子网和 IP 上定义自定义标签，或者在 OpenShift Container Platform 中启用自动标记可识别的子网，用于标识集群外部流量。当子网与流的源或目标 IP 匹配时，会添加一个对应的字段： SrcSubnetLabel 或 DstSubnetLabel 。

12.1.62. .spec.processor.advanced

描述

advanced 允许设置流处理器的内部配置。本节主要用于调试和精细的性能优化，如 **GOGC** 和 **GOMAXPROCS** 环境变量。在设置这些值时，您需要自己考虑相关的风险。

类型

object

属性	类型	描述
----	----	----

属性	类型	描述
conversationEndTimeout	string	conversationEndTimeout 是收到网络流后等待的时间，请考虑对话结束的时间。为 TCP 流收集 FIN 数据包时会忽略此延迟（请参阅 conversationTerminatingTimeout ）。
conversationHeartbeatInterval	string	conversationHeartbeatInterval 是对话的 "tick" 事件间的等待时间
conversationTerminatingTimeout	string	conversationTerminatingTimeout 是从检测到的 FIN 标志到对话结束间的等待时间。只适用于 TCP 流。
dropUnusedFields	布尔值	dropUnusedFields [已弃用 (*)] 此设置不再使用。
enableKubeProbes	布尔值	enableKubeProbes 是一个用于启用或禁用 Kubernetes 存活度和就绪度探测的标志
env	对象（字符串）	env 允许将自定义环境变量传递给底层组件。对于传递一些非常严格的性能调整选项（如 GOGC 、 GOMAXPROCS ）非常有用，它们不应作为 FlowCollector 描述符的一部分公开，因为它们仅在边缘调试或支持场景中有用。
healthPort	整数	healthPort 是 Pod 中的收集器 HTTP 端口，用于公开健康检查 API
port	整数	流收集器（主机端口）的端口。按照惯例，一些值会被禁止。它必须大于 1024，且不能是 4500、4789 和 6081。
profilePort	整数	profilePort 允许设置侦听此端口的 Go pprof profiler
scheduling	object	调度控制如何在节点上调度 pod。

12.1.63. .spec.processor.advanced.scheduling

描述

调度控制如何在节点上调度 pod。

类型

object

属性	类型	描述
关联性	object	如果指定，pod 的调度限制。有关文档，请参阅 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling 。
nodeSelector	对象（字符串）	nodeSelector 只允许将 pod 调度到具有每个指定标签的节点上。有关文档，请参阅 https://kubernetes.io/docs/concepts/configuration/assign-pod-node/ 。
priorityClassName	string	如果指定，指示 pod 的优先级。有关文档，请参阅 https://kubernetes.io/docs/concepts/scheduling-eviction/pod-priority-preemption/#how-to-use-priority-and-preemption 。如果没有指定，则使用默认优先级，如果没有默认值，则使用零。
容限（tolerations）	数组	tolerations 是一个容限 (toleration) 列表，允许 pod 调度到具有匹配污点的节点。有关文档，请参阅 https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling 。

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

描述

如果指定，pod 的调度限制。有关文档，请参阅 <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling>。

类型

object

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

描述

tolerations 是一个容限 (toleration) 列表, 允许 pod 调度到具有匹配污点的节点。有关文档, 请参阅 <https://kubernetes.io/docs/reference/kubernetes-api/workload-resources/pod-v1/#scheduling>。

类型

数组

12.1.66. .spec.processor.kafkaConsumerAutoscaler

描述

kafkaConsumerAutoscaler 是 Pod 横向自动扩展的 spec, 用于 **flowlogs-pipeline-transformer**, 它使用 Kafka 信息。当 Kafka 被禁用时, 会忽略此设置。请参阅 HorizontalPodAutoscaler 文档 (autoscaling/v2)。

类型

object

12.1.67. .spec.processor.metrics

描述

Metrics 定义有关指标的处理配置

类型

object

属性	类型	描述
disableAlerts	数组 (字符串)	<p>disableAlerts 是应禁用的警报列表。可能的值有：</p> <p>NetObservNoFlows, 它会在特定时间段内没有观察到流时触发。</p> <p>NetObservLokiError, 它会在因为 Loki 错误而丢弃流时触发。</p>

属性	类型	描述
includeList	数组 (字符串)	<p>includeList 是一个指标名称列表，用于指定要生成的名称。名称与 Prometheus 中没有前缀的名称对应。例</p> <p>如，namespace_egress_packets_total 在 Prometheus 中显示为 netobserv_namespace_egress_packets_total。请注意，您添加的指标越大，对 Prometheus 工作负载资源的影响更大。默认启用的指标包括：namespace_flows_total, node_ingress_bytes_total, workload_ingress_bytes_total, namespace_drop_packets_total (当启用了 PacketDrop 功能), namespace_rtt_seconds (当启用了 FlowRTT 功能), namespace_dns_latency_seconds (当启用了 DNSTracking 功能)。如需更多信息，包含可用指标的完整列表： https://github.com/netobserv/network-observability-operator/blob/main/docs/Metrics.md</p>
server	object	Prometheus scraper 的指标服务器端点配置

12.1.68. .spec.processor.metrics.server

描述

Prometheus scraper 的指标服务器端点配置

类型

object

属性	类型	描述
port	整数	指标服务器 HTTP 端口。
tls	object	TLS 配置。

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

描述

TLS 配置。

类型

object

属性	类型	描述
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过提供的证书的客户端侧验证。如果设置为 true ，则忽略 providedCaFile 字段。
provided	object	当 type 设置为 Provided 时的 TLS 配置。
providedCaFile	object	当将 type 设置为 Provided 时，对 CA 文件的引用。
type	string	选择 TLS 配置类型： <ul style="list-style-type: none"> - Disabled（默认）没有为端点配置 TLS。 - Provided 来手动提供证书文件和密钥文件。[不支持 (*)]。 - Auto 使用 OpenShift Container Platform 字段生成的证书，用于注解。

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

描述

当 **type** 设置为 **Provided** 时的 TLS 配置。

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。

属性	类型	描述
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

当将 **type** 设置为 **Provided** 时，对 CA 文件的引用。

类型

object

属性	类型	描述
file	string	配置映射或 secret 中的文件名。
name	string	包含该文件的配置映射或 secret 的名称。
namespace	string	包含该文件的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	文件引用的 type: "configmap" 或 "secret"。

12.1.72. .spec.processor.resources

描述

resources 是此容器所需的计算资源。如需更多信息，请参阅 <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>

类型

object

属性	类型	描述
limits	integer-or-string	限制描述了允许的最大计算资源量。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/
requests	integer-or-string	Requests 描述了所需的最少计算资源。如果容器省略了 Requests，则默认为 Limits（如果明确指定），否则默认为实现定义的值。请求不能超过限值。更多信息： https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/

12.1.73. .spec.processor.subnetLabels

描述

subnetLabels 允许在子网和 IP 上定义自定义标签，或者在 OpenShift Container Platform 中启用自动标记可识别的子网，用于标识集群外部流量。当子网与流的源或目标 IP 匹配时，会添加一个对应的字段：**SrcSubnetLabel** 或 **DstSubnetLabel**。

类型

object

属性	类型	描述
customLabels	数组	customLabels 允许自定义子网和 IP 标签，如标识集群外部工作负载或 Web 服务。如果启用 openShiftAutoDetect ，则 customLabels 可以在重叠时覆盖检测到的子网。
openShiftAutoDetect	布尔值	openShiftAutoDetect 允许当设置为 true 时，根据 OpenShift Container Platform 安装配置和 Cluster Network Operator 配置来自动检测机器、Pod 和服务子网。间接来说，这是准确检测外部流量的方法：针对这些子网没有标记的流是集群外部的。在 OpenShift Container Platform 中默认启用。

12.1.74. .spec.processor.subnetLabels.customLabels

描述

customLabels 允许自定义子网和 IP 标签，如标识集群外部工作负载或 Web 服务。如果启用 **openShiftAutoDetect**，则 **customLabels** 可以在重叠时覆盖检测到的子网。

类型

数组

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

描述

SubnetLabel 允许标记子网和 IP，如标识集群外部工作负载或 Web 服务。

类型

object

属性	类型	描述
cidrs	数组（字符串）	CIDR 列表，如 ["1.2.3.4/32"]。
name	string	标签名称，用于标记匹配流。

12.1.76. .spec.prometheus

描述

prometheus 定义 Prometheus 设置，如用于从控制台插件获取指标的 querier 配置。

类型

object

属性	类型	描述
querier	object	Prometheus 查询配置，如控制台插件中使用的客户端设置。

12.1.77. .spec.prometheus.querier

描述

Prometheus 查询配置，如控制台插件中使用的客户端设置。

类型

object

属性	类型	描述
----	----	----

属性	类型	描述
enable	布尔值	当 enable 为 true 时，Console 插件会查询 Prometheus 中的流指标，而不是 Loki。它默认是 enable ：将其设置为 false 来禁用此功能。Console 插件可以使用 Loki 或 Prometheus（或两者）作为指标的数据源（请参阅 spec.loki ）。并非所有查询都可从 Loki 转换为 Prometheus。因此，如果 Loki 被禁用，插件的一些功能也会被禁用，如获取每个 pod 的信息或查看原始流。如果启用了 Prometheus 和 Loki，Prometheus 会优先使用，Loki 作为 Prometheus 无法处理的查询的回退系统。如果两者都被禁用，则不会部署 Console 插件。
manual	object	用于 Manual 模式的 Prometheus 配置。
模式	string	模式 必须根据存储 Network Observability 指标的 Prometheus 安装类型来设置： - 使用 Auto 尝试自动配置。在 OpenShift Container Platform 中，它使用 OpenShift Container Platform Cluster Monitoring 中的 Thanos querier - 使用 Manual 用于手工设置
timeout	string	timeout 是控制台插件查询 Prometheus 的读取超时。超时值为零表示没有超时。

12.1.78. .spec.prometheus.querier.manual

描述

用于 **Manual** 模式的 Prometheus 配置。

类型

object

属性	类型	描述
----	----	----

属性	类型	描述
forwardUserToken	布尔值	设置 true 以将查询中的登录用户令牌转发到 Prometheus
tls	object	Prometheus URL 的 TLS 客户端配置。
url	string	url 是现有 Prometheus 服务的地址，用于查询指标。

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

描述

Prometheus URL 的 TLS 客户端配置。

类型

object

属性	类型	描述
caCert	object	caCert 定义证书颁发机构的证书引用
enable	布尔值	启用 TLS
insecureSkipVerify	布尔值	insecureSkipVerify 允许跳过服务器证书的客户端验证。如果设置为 true ，则忽略 caCert 字段。
userCert	object	userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

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

描述

caCert 定义证书颁发机构的证书引用

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。

属性	类型	描述
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。
type	string	证书引用的类型： configmap 或 secret 。

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

描述

userCert 定义用户证书引用，用于 mTLS（您可以在使用单向 TLS 时忽略它）

类型

object

属性	类型	描述
certFile	string	certFile 定义配置映射或 secret 中证书文件名的路径。
certKey	string	certKey 定义配置映射 / Secret 中证书私钥文件名的路径。当不需要键时会省略。
name	string	包含证书的配置映射或 Secret 的名称。
namespace	string	包含证书的配置映射或 secret 的命名空间。如果省略，则默认为使用与部署 Network Observability 相同的命名空间。如果命名空间不同，则复制配置映射或 secret，以便可以根据需要挂载它。

属性	类型	描述
type	string	证书引用的类型： configmap 或 secret 。

第 13 章 FLOWMETRIC 配置参数

FlowMetric 是允许从收集的流日志创建自定义指标的 API。

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

描述

FlowMetric 是允许从收集的流日志创建自定义指标的 API。

类型

object

属性	类型	描述
apiVersion	string	APIVersion 定义对象的这个表示法的版本化的 schema。服务器应该将识别的模式转换为最新的内部值，并可拒绝未识别的值。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
kind	string	kind 是一个字符串值，代表此对象所代表的 REST 资源。服务器可以从客户端向其提交请求的端点推断。无法更新。采用驼峰拼写法 (CamelCase)。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds
metadata	object	标准对象元数据。更多信息： https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata

属性	类型	描述
spec	object	<p>FlowMetricSpec 定义所需的 FlowMetric。提供的 API 允许您根据您的需要自定义这些指标。</p> <p>在添加新指标或修改现有标签时，您必须仔细监控 Prometheus 工作负载的内存用量，因为这可能会产生重大影响。Cf https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric</p> <p>要检查所有 Network Observability 指标的基数 (cardinality)，由 <code>(name)promql: count({name=~"netobserv.*"})</code> 运行。</p>

13.1.1. .metadata

描述

标准对象元数据。更多信息：<https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#metadata>

类型

object

13.1.2. .spec

描述

FlowMetricSpec 定义所需的 FlowMetric。提供的 API 允许您根据您的需要自定义这些指标。在添加新指标或修改现有标签时，您必须仔细监控 Prometheus 工作负载的内存用量，因为这可能会产生重大影响。Cf <https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric>

要检查所有 Network Observability 指标的基数 (cardinality)，由 `(name)promql: count({name=~"netobserv.*"})` 运行。

类型

object

必填

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

属性	类型	描述
bucket	数组 (字符串)	当 type 为 "Histogram" 时使用的存储桶列表列表必须显示为浮点数。如果没有设置, 使用 Prometheus 默认存储桶。
charts	数组	管理员视图中的 OpenShift Container Platform 控制台图表配置 Dashboards 菜单。
direction	string	过滤入口、出口或任何方向流。当设置为 Ingress 时, 它等同于在 FlowDirection 中添加正则表达式过滤: 0 2 。当设置为 Egress 时, 它等同于在 FlowDirection 中添加正则表达式过滤: 1 2 。
divider	string	当非零时, 缩放因素 (divider) 的值。指标值 = Flow value / Divider。
过滤器	数组	filters 是用于限制考虑哪些流的字段和值列表。通常, 需要使用这些过滤消除重复项: Duplicate != "true" 和 FlowDirection = "0" 有关可用字段列表, 请参阅相关文档: https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html 。

属性	类型	描述
labels	数组 (字符串)	labels 是一个应该用作 Prometheus 标签的字段列表，也称为维度。从选择标签时，此指标的粒度级别以及查询时可用的聚合结果。它需要小心执行，因为它会影响指标基数 (cf https://rhobs-handbook.netlify.app/products/openshiftmonitoring/telemetry.md/#what-is-the-cardinality-of-a-metric)。通常，避免设置非常高的基数标签，如 IP 或 MAC 地址。"SrcK8S_OwnerName" 或 "DstK8S_OwnerName" 应优先于 "SrcK8S_Name" 或 "DstK8S_Name"。有关可用字段列表，请参阅相关文档： https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html 。
metricName	string	指标的名称。在 Prometheus 中，它会自动带有 "netobserv_" 前缀。
type	string	指标类型："Counter" 或 "Histogram"。使用 "Counter"，对随时间增加的任何值以及您可以计算速率（如 Bytes 或 Packets）的任何值。对于需要独立抽样的任何值（如延迟），使用 "Histogram"。
valueField	string	valueField 是流字段，必须用作此指标的值。此字段必须是数字值。如果保留为空，计算流数，而不是每个流的特定值。有关可用字段列表，请参阅相关文档： https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html 。

13.1.3. .spec.charts

描述

管理员视图中的 OpenShift Container Platform 控制台图表配置 Dashboards 菜单。

类型

数组

13.1.4. .spec.charts[]

描述

配置与一个指标关联的图表/仪表板的生成

类型

object

必填

- dashboardName
- queries
- title
- type

属性	类型	描述
dashboardName	string	包含仪表板的名称。如果此名称没有引用现有仪表板，则会创建一个新仪表板。
queries	数组	要在此图表中显示的查询列表。如果 type 是 SingleStat ，并提供了多个查询，则在多个面板中会自动扩展此图表（每个查询一个）。
sectionName	string	包含仪表板部分的名称。如果此名称没有引用现有部分，则会创建一个新部分。如果省略 sectionName 或为空，则图表被放置在全局 top 部分中。
title	string	图表的标题。
type	string	图表的类型。
unit	string	此图表的单元。目前只支持几个单元。如果保留为空，则使用通用数字。

13.1.5. .spec.charts[].queries

描述

要在此图表中显示的查询列表。如果 **type** 是 **SingleStat**，并提供了多个查询，则在多个面板中会自动扩展此图表（每个查询一个）。

类型

数组

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

描述

配置 PromQL 查询

类型

object

必填

- 图例
- promQL
- top

属性	类型	描述
图例	string	适用于此图表中每个时间序列的查询图例。显示多个时间序列时，您应该设置一个可区分每个时间序列的图例。它可以使用以下格式： {{ Label }} 。例如，如果根据每个标签对时间序列进行分组，如 sum(rate(\$METRIC[2m])) by (Label1, Label2) ，您可以写为图例: Label1={{ Label1 }} ， Label2={{ Label2 }} 。
promQL	string	要针对 Prometheus 运行 promQL 查询。如果图表 type 是 SingleStat ，则此查询应只返回单个时间序列。对于其他类型，会显示前 7 个。您可以使用 \$METRIC 来引用此资源中定义的指标。例如： sum(rate(\$METRIC[2m])) 。如需了解更多有关 promQL 的信息，请参阅 Prometheus 文档： https://prometheus.io/docs/prometheus/latest/querying/basics/
top	整数	每个时间戳显示的最大 N 系列。不适用于 SingleStat 图表类型。

13.1.7. .spec.filters

描述

filters 是用于限制考虑哪些流的字段和值列表。通常，需要使用这些过滤消除重复项：**Duplicate != "true"** 和 **FlowDirection = "0"**有关可用字段列表，请参阅相关文档：

https://docs.openshift.com/container-platform/latest/observability/network_observability/json-flows-format-reference.html。

类型

数组

13.1.8. .spec.filters[]

描述

类型

object

必填

- field
- matchType

属性	类型	描述
field	string	要过滤的字段名称
matchType	string	要应用的匹配类型
value	string	要过滤的值。当 matchType 为 Equal 或 NotEqual 时，您可以使用带有 \$(SomeField) 的字段注入来引用流的任何其他字段。

第 14 章 网络流格式参考

这些是网络流格式的规格，在内部和将流导出到 Kafka 时使用。

14.1. 网络流格式参考

这是网络流格式的规格。在配置了 Kafka 导出器时，使用这种格式，用于 Prometheus 指标标签，以及 Loki 存储的内部。

"Filter ID"列显示定义 Quick Filters 时使用的相关名称（请参阅 **FlowCollector** 规格中的 **spec.consolePlugin.quickFilters**）。

当直接查询 Loki 时，"Loki label"列很有用：需要使用 [流选择器](#) 来选择标签字段。

"Cardinality" 列提供有关在此字段用作带有 **FlowMetric** API 的 Prometheus 标签时代表的指标卡inality 的信息。如需更多信息，请参阅"FlowMetric API 参考"。

Name	类型	描述	过滤 ID	Loki 标签	基准
Bytes	number	字节数	不适用	否	avoid
DnsErrno	number	从 DNS tracker ebpf hook 功能返回的错误数	dns_errno	否	fine
DnsFlags	number	DNS 记录的 DNS 标记	不适用	否	fine
DnsFlagsResponseCode	string	解析的 DNS 标头 RCODE 名称	dns_flag_response_code	否	fine
DnsId	number	DNS 记录 ID	dns_id	否	avoid
DnsLatencyMs	number	DNS 请求和响应之间的时间（以毫秒为单位）	dns_latency	否	avoid
Dscp	number	差异化服务代码点 (DSCP) 值	dscp	否	fine
DstAddr	string	目标 IP 地址 (ipv4 或 ipv6)	dst_address	否	avoid
DstK8S_HostIP	string	目的地节点 IP	dst_host_address	否	fine
DstK8S_HostName	string	目标节点名称	dst_host_name	否	fine
DstK8S_Name	string	目标 Kubernetes 对象的名称，如 Pod 名称、服务名称或节点名称。	dst_name	否	careful

Name	类型	描述	过滤 ID	Loki 标签	基准
DstK8S_ Namespace	string	目标命名空间	dst_namespace	是	fine
DstK8S_ OwnerName	string	目标所有者的名称，如 Deployment 名称、StatefulSet 名称等。	dst_owner_name	是	fine
DstK8S_ OwnerType	string	目标所有者的类型，如 Deployment、StatefulSet 等。	dst_kind	否	fine
DstK8S_ Type	string	目标 Kubernetes 对象的类型，如 Pod、Service 或 Node。	dst_kind	是	fine
DstK8S_ Zone	string	目标可用区	dst_zone	是	fine
DstMac	string	目标 MAC 地址	dst_mac	否	avoid
DstPort	number	目的地端口	dst_port	否	careful
DstSubnetLabel	string	目的地子网标签	dst_subnet_label	否	fine
Duplicate	布尔值	指明此流是否也从同一主机上的另一个接口捕获	不适用	是	fine
标记	number	在流中包括唯一 TCP 标志的逻辑 OR 组合，根据 RFC-9293，带有额外的自定义标志来代表每个数据包的组合： - SYN+ACK (0x100) - FIN+ACK (0x200) - RST+ACK (0x400)	不适用	否	fine
FlowDirection	number	来自节点观察点的流解释方向。可以是： - 0: Ingress（来自节点观察点的流量） - 1: Egress（从节点观察到流量，来自节点观察点） - 2: Inner（具有相同源和目标节点）	node_direction	是	fine
IcmpCode	number	ICMP 代码	icmp_code	否	fine

Name	类型	描述	过滤 ID	Loki 标签	基准
IcmpType	number	ICMP 类型	icmp_type	否	fine
IfDirections	number	来自网络接口观察点的流方向。可以是： - 0: Ingress (接口传入的流量) - 1: Egress (接口传出流量)	ifdirections	否	fine
接口	string	网络接口	interfaces	否	careful
K8S_ClusterName	string	集群名称或标识符	cluster_name	是	fine
K8S_FlowLayer	string	流层：'app' 或 'infra'	flow_layer	否	fine
Packets	number	数据包数	不适用	否	avoid
PktDropBytes	number	内核丢弃的字节数	不适用	否	avoid
PktDropLatestDropCause	string	最新丢弃原因	pkt_drop_cause	否	fine
PktDropLatestFlags	number	最后丢弃的数据包上的 TCP 标志	不适用	否	fine
PktDropLatestState	string	最后丢弃的数据包上的 TCP 状态	pkt_drop_state	否	fine
PktDropPackets	number	内核丢弃的数据包数	不适用	否	avoid
Proto	number	L4 协议	protocol	否	fine
SrcAddr	string	源 IP 地址 (ipv4 或 ipv6)	src_address	否	avoid
SrcK8S_HostIP	string	源节点 IP	src_host_address	否	fine

Name	类型	描述	过滤 ID	Loki 标签	基准
SrcK8S_HostName	string	源节点名称	src_host_name	否	fine
SrcK8S_Name	string	源 Kubernetes 对象的名称，如 Pod 名称、服务名称或节点名称。	src_name	否	careful
SrcK8S_Namespace	string	源命名空间	src_namespace	是	fine
SrcK8S_OwnerName	string	源所有者的名称，如 Deployment 名称、StatefulSet 名称等。	src_owner_name	是	fine
SrcK8S_OwnerType	string	源所有者的类型，如 Deployment、StatefulSet 等。	src_kind	否	fine
SrcK8S_Type	string	源 Kubernetes 对象的类型，如 Pod、Service 或 Node。	src_kind	是	fine
SrcK8S_Zone	string	源可用区	src_zone	是	fine
SrcMac	string	源 MAC 地址	src_mac	否	avoid
SrcPort	number	源端口	src_port	否	careful
SrcSubnetLabel	string	源子网标签	src_subnet_label	否	fine
TimeFlowEndMs	number	此流的结束时间戳，以毫秒为单位	不适用	否	avoid
TimeFlowRttNs	number	TCP Smoothed Round Trip Time (SRTT)，单位为纳秒	time_flow_rtt	否	avoid
TimeFlowStartMs	number	开始此流的时间戳，以毫秒为单位	不适用	否	avoid
TimeReceived	number	由流被流收集器接收并处理时的时间戳，以秒为单位	不适用	否	avoid
_HashId	string	在对话跟踪中，对话标识符	id	否	avoid

Name	类型	描述	过滤 ID	Loki 标签	基准
_Record Type	string	记录类型：'flowLog' 用于常规流日志，或 'newConnection', 'heartbeat', 'endConnection' 用于对话跟踪	type	是	fine

第 15 章 NETWORK OBSERVABILITY 故障排除

为了协助对 Network Observability 问题进行故障排除，可以执行一些故障排除操作。

15.1. 使用 MUST-GATHER 工具

您可以使用 `must-gather` 工具来收集有关 Network Observability Operator 资源和集群范围资源的信息，如 pod 日志、**FlowCollector** 和 **Webhook** 配置。

流程

1. 进入到要存储 `must-gather` 数据的目录。
2. 运行以下命令来收集集群范围的 `must-gather` 资源：

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

15.2. 在 OPENSIFT CONTAINER PLATFORM 控制台中配置网络流量菜单条目

当网络流量菜单条目没有在 OpenShift Container Platform 控制台的 **Observe** 菜单中列出时，在 OpenShift Container Platform 控制台中手动配置网络流量菜单条目。

先决条件

- 已安装 OpenShift Container Platform 版本 4.10 或更高版本。

流程

1. 运行以下命令，检查 `spec.consolePlugin.register` 字段是否已设置为 `true`：

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

输出示例

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

2. 可选：通过手动编辑 Console Operator 配置来添加 `netobserv-plugin` 插件：

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

输出示例

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

3. 可选：运行以下命令，将 `spec.consolePlugin.register` 字段设置为 `true`：

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

输出示例

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

4. 运行以下命令，确保控制台 pod 的状态为 `running`：

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

5. 运行以下命令重启控制台 pod：

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

6. 清除浏览器缓存和历史记录。

7. 运行以下命令，检查 Network Observability 插件 pod 的状态：

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

输出示例

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

8. 运行以下命令，检查 Network Observability 插件 pod 的日志：

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

输出示例

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

15.3. 安装 KAFKA 后 FLOWLOGS-PIPELINE 不会消耗网络流

如果您首先使用 **deploymentModel: KAFKA** 部署了流收集器，然后部署 Kafka，则流收集器可能无法正确连接到 Kafka。手动重启 flow-pipeline pod，其中 Flowlogs-pipeline 不使用 Kafka 中的网络流。

流程

1. 运行以下命令，删除 flow-pipeline pod 来重启它们：

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

15.4. 无法从 BR-INT 和 BR-EX 接口查看网络流

br-ex' 和 br-int 是 OSI 第 2 层操作的虚拟网桥设备。eBPF 代理分别在 IP 和 TCP 级别（第 3 和 4 层）中工作。当网络流量由物理主机或虚拟 pod 接口处理时，您可以预期 eBPF 代理捕获通过 br-ex 和 br-int 的网络流量。如果您限制 eBPF 代理网络接口只附加到 br-ex 和 br-int，则不会看到任何网络流。

手动删除限制网络接口到 br-int 和 br-ex 的 interfaces 或 excludeInterfaces 中的部分。

流程

1. 删除 **interfaces: ['br-int', 'br-ex']** 字段。这允许代理从所有接口获取信息。或者，您可以指定 Layer-3 接口，例如 **eth0**。运行以下命令：

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

输出示例

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

- ❶ 指定网络接口。

15.5. NETWORK OBSERVABILITY 控制器管理器 POD 内存不足

您可以通过编辑 **Subscription** 对象中的 **spec.config.resources.limits.memory** 规格来为 Network Observability Operator 增加内存限值。

流程

1. 在 Web 控制台中，进入到 Operators → Installed Operators
2. 点 Network Observability，然后选择 Subscription。
3. 在 Actions 菜单中，点 Edit Subscription。
 - a. 另外，您可以运行以下命令来使用 CLI 为 **Subscription** 对象打开 YAML 配置：

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

4. 编辑 **Subscription** 对象以添加 **config.resources.limits.memory** 规格，并将值设置为考虑您的内存要求。有关资源注意事项的更多信息，请参阅附加资源：

```
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 例如，您可以将内存限值增加到 **800Mi**。
- ② 这个值不应该被编辑，但请注意，它会根据 Operator 的最当前的版本进行更改。

15.6. 对 LOKI 运行自定义查询

若要进行故障排除，可以对 Loki 运行自定义查询。有两种方法示例，您可以通过将 `<api_token>` 替换为您自己的 `<api_token>` 来根据您的需要进行调整。



注意

这些示例为 Network Observability Operator 和 Loki 部署使用 **netobserv** 命名空间。另外，示例假定 LokiStack 名为 **loki**。您可以通过调整示例（特别是 **-n netobserv** 或 **loki-gateway** URL）来使用不同的命名空间和命名。

先决条件

- 安装 Loki Operator 以用于 Network Observability Operator

流程

- 要获取所有可用标签，请运行以下命令：

```
$ 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
```


- 要从源命名空间 **my-namespace** 获取所有流，请运行以下命令：

```
$ 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
```

其他资源

- [资源注意事项](#)

15.7. LOKI RESOURCEEXHAUSTED 错误故障排除

当 Network Observability 发送的网络流数据时，Loki 可能会返回一个 **ResourceExhausted** 错误，超过配置的最大消息大小。如果使用 Red Hat Loki Operator，则这个最大消息大小被配置为 100 MiB。

流程

1. 进入到 **Operators** → **Installed Operators**，从 **Project** 下拉菜单中选择 **All projects**。
2. 在 **Provided APIs** 列表中，选择 **Network Observability Operator**。
3. 点 **Flow Collector**，然后点 **YAML view** 选项卡。
 - a. 如果使用 Loki Operator，请检查 **spec.loki.batchSize** 值没有超过 98 MiB。
 - b. 如果您使用与 Red Hat Loki Operator 不同的 Loki 安装方法，如 Grafana Loki，请验证 **grpc_server_max_recv_msg_size** [Grafana Loki 服务器设置](#) 高于 **FlowCollector** 资源 **spec.loki.batchSize** 值。如果没有，您必须增加 **grpc_server_max_recv_msg_size** 值，或者减少 **spec.loki.batchSize** 值，使其小于限制。
4. 如果您编辑 **FlowCollector**，点 **Save**。

15.8. LOKI 空 RING 错误

Loki "empty ring" 错误会导致流没有存储在 Loki 中，且不显示在 web 控制台中。这个错误可能会在各种情况下发生。还没有解决的一个临时解决方案。您可以采取一些操作来调查 Loki pod 中的日志，并验证 **LokiStack** 是否健康并就绪。

观察到此错误的一些情况如下：

- 在卸载并在同一个命名空间中重新安装 **LokiStack** 后，旧的 PVC 不会被删除，这会导致这个错误。
 - **Action**：您可以尝试再次删除 **LokiStack**，删除 PVC，然后重新安装 **LokiStack**。
- 证书轮转后，这个错误可能会阻止与 **flowlogs-pipeline** 和 **console-plugin** pod 的通信。
 - **Action**：您可以重启 pod 来恢复连接。

15.9. 资源故障排除

15.10. LOKISTACK 速率限制错误

放置在 Loki 租户上的速率限制可能会导致数据潜在的临时丢失，而 429 错误：**Per stream rate limit exceeded (limit:xMB/sec) while attempting to ingest for stream**。您可能会考虑设置了警报来通知您这个错误。如需更多信息，请参阅本节的附加资源中的“为 NetObserv 仪表盘创建 Loki 速率限制警报”。

您可以使用 **perStreamRateLimit** 和 **perStreamRateLimitBurst** 规格更新 LokiStack CRD，如以下步骤所示。

流程

1. 进入到 Operators → Installed Operators，从 Project 下拉菜单查看 All projects。
2. 查找 Loki Operator，然后选择 LokiStack 选项卡。
3. 使用 YAML 视图 创建或编辑现有 LokiStack 实例，以添加 **perStreamRateLimit** 和 **perStreamRateLimitBurst** 规格：

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: loki
  namespace: netobserv
spec:
  limits:
    global:
      ingestion:
        perStreamRateLimit: 6 1
        perStreamRateLimitBurst: 30 2
  tenants:
    mode: openshift-network
    managementState: Managed
```

- 1 **perStreamRateLimit** 的默认值为 3。
- 2 **perStreamRateLimitBurst** 的默认值为 15。

4. 点击 Save。

验证

更新 **perStreamRateLimit** 和 **perStreamRateLimitBurst** 规格后，集群重启中的 pod，429 rate-limit 错误不再发生。