OpenShift Container Platform 4.18

Logging

Configuring and using logging in OpenShift Container Platform

Last Updated: 2025-06-10

OpenShift Container Platform 4.18 Logging

Configuring and using logging in OpenShift Container Platform

Legal Notice

Abstract

Use logging to collect, visualize, forward, and store log data to troubleshoot issues, identify performance bottlenecks, and detect security threats in OpenShift Container Platform.

Table of Contents

CHAPTER 1. LOGGING 6.2

1.1. SUPPORT

Only the configuration options described in this documentation are supported for logging.

Do not use any other configuration options, as they are unsupported. Configuration paradigms might change across OpenShift Container Platform releases, and such cases can only be handled gracefully if all configuration possibilities are controlled. If you use configurations other than those described in this documentation, your changes will be overwritten, because Operators are designed to reconcile any differences.

NOTE

If you must perform configurations not described in the OpenShift Container Platform documentation, you must set your Red Hat OpenShift Logging Operator to **Unmanaged**. An unmanaged logging instance is not supported and does not receive updates until you return its status to **Managed**.

NOTE

Logging is provided as an installable component, with a distinct release cycle from the core OpenShift Container Platform. The Red Hat OpenShift Container Platform Life Cycle Policy outlines release compatibility.

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system offered as a GA log store for logging for Red Hat OpenShift that can be visualized with the OpenShift Observability UI. The Loki configuration provided by OpenShift Logging is a short-term log store designed to enable users to perform fast troubleshooting with the collected logs. For that purpose, the logging for Red Hat OpenShift configuration of Loki has short-term storage, and is optimized for very recent queries. For long-term storage or queries over a long time period, users should look to log stores external to their cluster.

Logging for Red Hat OpenShift is an opinionated collector and normalizer of application, infrastructure, and audit logs. It is intended to be used for forwarding logs to various supported systems.

Logging is not:

- A high scale log collection system
- Security Information and Event Monitoring (SIEM) compliant
- A "bring your own" (BYO) log collector configuration
- Historical or long term log retention or storage
- A guaranteed log sink
- Secure storage - audit logs are not stored by default

1.1.1. Supported API custom resource definitions

The following table describes the supported Logging APIs.

Table 1.1. Logging API support states

CustomResourceDefinition (CRD)	ApiVersion	Support state
LokiStack	lokistack.loki.grafana.com/v1	Supported from 5.5
RulerConfig	rulerconfig.loki.grafana/v1	Supported from 5.7
AlertingRule	alertingrule.loki.grafana/v1	Supported from 5.7
RecordingRule	recordingrule.loki.grafana/v1	Supported from 5.7
LogFileMetricExporter	LogFileMetricExporter.logging.openshift.io/v1alpha1	Supported from 5.8
ClusterLogForwarder	clusterlogforwarder.observability.openshift.io/v1	Supported from 6.0

1.1.2. Unsupported configurations

You must set the Red Hat OpenShift Logging Operator to the **Unmanaged** state to modify the following components:

- The collector configuration file
- The collector daemonset

Explicitly unsupported cases include:

- **Configuring the logging collector using environment variables** You cannot use environment variables to modify the log collector.
- **Configuring how the log collector normalizes logs**You cannot modify default log normalization.

1.1.3. Support policy for unmanaged Operators

The *management state* of an Operator determines whether an Operator is actively managing the resources for its related component in the cluster as designed. If an Operator is set to an *unmanaged* state, it does not respond to changes in configuration nor does it receive updates.

While this can be helpful in non-production clusters or during debugging, Operators in an unmanaged state are unsupported and the cluster administrator assumes full control of the individual component configurations and upgrades.

An Operator can be set to an unmanaged state using the following methods:

- **Individual Operator configuration**
  Individual Operators have a **managementState** parameter in their configuration. This can be accessed in different ways, depending on the Operator. For example, the Red Hat OpenShift Logging Operator accomplishes this by modifying a custom resource (CR) that it manages, while the Cluster Samples Operator uses a cluster-wide configuration resource.

Changing the **managementState** parameter to **Unmanaged** means that the Operator is not actively managing its resources and will take no action related to the related component. Some Operators might not support this management state as it might damage the cluster and require manual recovery.

WARNING

Changing individual Operators to the **Unmanaged** state renders that particular component and functionality unsupported. Reported issues must be reproduced in **Managed** state for support to proceed.

- **Cluster Version Operator (CVO) overrides**
  The **spec.overrides** parameter can be added to the CVO's configuration to allow administrators to provide a list of overrides to the CVO's behavior for a component. Setting the **spec.overrides[].unmanaged** parameter to **true** for a component blocks cluster upgrades and alerts the administrator after a CVO override has been set:

  > Disabling ownership via cluster version overrides prevents upgrades. Please remove overrides before continuing.

WARNING

Setting a CVO override puts the entire cluster in an unsupported state. Reported issues must be reproduced after removing any overrides for support to proceed.

1.1.4. Collecting logging data for Red Hat Support

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

You can use the must-gather tool to collect diagnostic information for project-level resources, cluster-level resources, and each of the logging components. For prompt support, supply diagnostic information for both OpenShift Container Platform and logging.

1.1.4.1. About the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues.

For your logging, **must-gather** collects the following information:

- Project-level resources, including pods, configuration maps, service accounts, roles, role bindings, and events at the project level

- Cluster-level resources, including nodes, roles, and role bindings at the cluster level
- OpenShift Logging resources in the **openshift-logging** and **openshift-operators-redhat** namespaces, including health status for the log collector, the log store, and the log visualizer

When you run **oc adm must-gather**, a new pod is created on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in the current working directory.

1.1.4.2. Collecting logging data

You can use the **oc adm must-gather** CLI command to collect information about logging.

Procedure

To collect logging information with **must-gather**:

1. Navigate to the directory where you want to store the **must-gather** information.
2. Run the **oc adm must-gather** command against the logging image:

```
$ oc adm must-gather --image=$(oc -n openshift-logging get deployment.apps/cluster-logging-operator -o jsonpath='{.spec.template.spec.containers[?(@.name == "cluster-logging-operator")].image}')
```

   The **must-gather** tool creates a new directory that starts with **must-gather.local** within the current directory. For example: **must-gather.local.4157245944708210408**.

3. Create a compressed file from the **must-gather** directory that was just created. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar -cvaf must-gather.tar.gz must-gather.local.4157245944708210408
```

4. Attach the compressed file to your support case on the Red Hat Customer Portal .

1.2. LOGGING 6.2 RELEASE NOTES

1.2.1. Logging 6.2.2 Release Notes

This release includes RHBA-2025:4526.

1.2.1.1. Bug Fixes

- Before this update, logs without the **responseStatus.code** field caused parsing errors in the Loki **distributor** component. This happened when using the OpenTelemetry data model. With this update, logs without the **responseStatus.code** field are parsed correctly. (LOG-7012)
- Before this update, the Cloudwatch output supported log events up to 256 KB in size. With this update, the Cloudwatch output supports up to 1 MB in size to match the updates published by Amazon Web Services (AWS). (LOG-7013)
- Before this update, **auditd** log messages with multiple **msg** keys could cause errors in collector pods, because the standard **auditd** log format expects a single **msg** field per log entry that follows the **msg=audit(TIMESTAMP:ID)** structure. With this update, only the first **msg** value is used, which resolves the issue and ensures accurate extraction of audit metadata. (LOG-7014)

- Before this update, collector pods would enter a crash loop due to a configuration error when attempting token-based authentication with an Elasticsearch output. With this update, token authentication with an Elasticsearch output generates a valid configuration. (LOG-7017)

1.2.2. Logging 6.2.1 Release Notes

This release includes RHBA-2025:3908.

1.2.2.1. Bug Fixes

- Before this update, application programming interface (API) audit logs collected from the management cluster used the **cluster_id** value from the management cluster. With this update, API audit logs use the **cluster_id** value from the guest cluster. (LOG-4445)

- Before this update, issuing the **oc explain obsclf.spec.filters** command did not list all the supported filters in the command output. With this update, all the supported filter types are listed in the command output. (LOG-6753)

- Before this update the log collector flagged a **ClusterLogForwarder** resource with multiple inputs to a LokiStack output as invalid due to incorrect internal processing logic. This update fixes the issue. (LOG-6758)

- Before this update, issuing the **oc explain** command for the **clusterlogforwarder.spec.outputs.syslog** resource returned an incomplete result. With this update, the missing supported types for **rfc** and **enrichment** attributes are listed in the result correctly. (LOG-6869)

- Before this update, empty OpenTelemetry (OTEL) tuning configuration caused validation errors. With this update, validation rules have been updated to accept empty tuning configuration. (LOG-6878)

- Before this update the Red Hat OpenShift Logging Operator could not update the **securitycontextconstraint** resource that is required by the log collector. With this update, the required cluster role has been provided to the service account of the Red Hat OpenShift Logging Operator. As a result of which, Red Hat OpenShift Logging Operator can create or update the **securitycontextconstraint** resource. (LOG-6879)

- Before this update, the API documentation for the URL attribute of the **syslog** resource incorrectly mentioned the value **udps** as a supported value. With this update, all references to **udps** have been removed. (LOG-6896)

- Before this update, the Red Hat OpenShift Logging Operator was intermittently unable to update the object in logs due to update conflicts. This update resolves the issue and prevents conflicts during object updates by using the **Patch()** function instead of the **Update()** function. (LOG-6953)

- Before this update, Loki ingesters that got into an unhealthy state due to networking issues stayed in that state even after the network recovered. With this update, you can configure the Loki Operator to perform service discovery more often so that unhealthy ingesters can rejoin the group. (LOG-6992)

- Before this update, the Vector collector could not forward Open Virtual Network (OVN) and Auditd logs. With this update, the Vector collector can forward OVN and Auditd logs. (LOG-6997)

1.2.2.2. CVEs

- CVE-2022-49043
- CVE-2024-2236
- CVE-2024-5535
- CVE-2024-56171
- CVE-2025-24928

1.2.3. Logging 6.2.0 Release Notes

This release includes Logging for Red Hat OpenShift Bug Fix Release 6.2.0 .

1.2.3.1. New Features and Enhancements

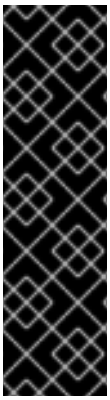
1.2.3.1.1. Log Collection

- With this update, HTTP outputs include a **proxy** field that you can use to send log data through an HTTP proxy. (LOG-6069)

1.2.3.1.2. Log Storage

- With this update, time-based stream sharding in Loki is now enabled by the Loki Operator. This solves the issue of ingesting log entries older than the sliding time-window used by Loki. (LOG-6757)
- With this update, you can configure a custom certificate authority (CA) certificate with Loki Operator when using Swift as an object store. (LOG-4818)
- With this update, you can configure workload identity federation on Google Cloud Platform (GCP) by using the Cluster Credential Operator in OpenShift 4.17 and later releases with the Loki Operator. (LOG-6158)

1.2.3.2. Technology Preview



IMPORTANT

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

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

- With this update, OpenTelemetry support offered by OpenShift Logging continues to improve, specifically in the area of enabling migrations from the ViaQ data model to **OpenTelemetry** when forwarding to **LokiStack**. (LOG-6146)
- With this update, the **structuredMetadata** field has been removed from Loki Operator in the **otlp** configuration because structured metadata is now the default type. Additionally, the

update introduces a **drop** field that administrators can use to drop **OpenTelemetry** attributes when receiving data through **OpenTelemetry** protocol (OTLP). (LOG-6507)

1.2.3.3. Bug Fixes

- Before this update, the timestamp shown in the console logs did not match the **@timestamp** field in the message. With this update the timestamp is correctly shown in the console. (LOG-6222)
- The introduction of **ClusterLogForwarder** 6.x modified the **ClusterLogForwarder** API to allow for a consistent templating mechanism. However, this was not applied to the **syslog** output spec API for the **facility** and **severity** fields. This update adds the required validation to the **ClusterLogForwarder** API for the **facility** and **severity** fields. (LOG-6661)
- Before this update, an error in the Loki Operator generating the Loki configuration caused the amount of workers to delete to be zero when **1x.pico** was set as the **LokiStack** size. With this update, the number of workers to delete is set to 10. (LOG-6781)

1.2.3.4. Known Issues

- The previous data model encoded all information in JSON. The console still uses the query of the previous data model to decode both old and new entries. The logs that are stored by using the new **OpenTelemetry** data model for the **LokiStack** output display the following error in the logging console:

  ```
  __error__ JSONParserErr
  __error_details__ Value looks like object, but can't find closing '}' symbol
  ```

  You can ignore the error as it is only a result of the query and not a data-related error. (LOG-6808)
- Currently, the API documentation incorrectly mentions **OpenTelemetry** protocol (OTLP) attributes as **included** instead of **excluded** in the descriptions of the **drop** field. (LOG-6839).

1.2.3.5. CVEs

- CVE-2020-11023
- CVE-2024-12797

1.3. LOGGING 6.2

The **ClusterLogForwarder** custom resource (CR) is the central configuration point for log collection and forwarding.

1.3.1. Inputs and outputs

Inputs specify the sources of logs to be forwarded. Logging provides the following built-in input types that select logs from different parts of your cluster:

- **application**
- **receiver**
- **infrastructure**

- **audit**

You can also define custom inputs based on namespaces or pod labels to fine-tune log selection.

Outputs define the destinations where logs are sent. Each output type has its own set of configuration options, allowing you to customize the behavior and authentication settings.

1.3.2. Receiver input type

The receiver input type enables the Logging system to accept logs from external sources. It supports two formats for receiving logs: **http** and **syslog**.

The **ReceiverSpec** field defines the configuration for a receiver input.

1.3.3. Pipelines and filters

Pipelines determine the flow of logs from inputs to outputs. A pipeline consists of one or more input refs, output refs, and optional filter refs. You can use filters to transform or drop log messages within a pipeline. The order of filters matters, as they are applied sequentially, and earlier filters can prevent log messages from reaching later stages.

1.3.4. Operator behavior

The Cluster Logging Operator manages the deployment and configuration of the collector based on the **managementState** field of the **ClusterLogForwarder** resource:

- When set to **Managed** (default), the Operator actively manages the logging resources to match the configuration defined in the spec.
- When set to **Unmanaged**, the Operator does not take any action, allowing you to manually manage the logging components.

1.3.5. Validation

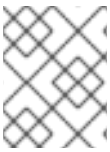
Logging includes extensive validation rules and default values to ensure a smooth and error-free configuration experience. The **ClusterLogForwarder** resource enforces validation checks on required fields, dependencies between fields, and the format of input values. Default values are provided for certain fields, reducing the need for explicit configuration in common scenarios.

1.3.6. Quick start

OpenShift Logging supports two data models:

- ViaQ (General Availability)
- OpenTelemetry (Technology Preview)

You can select either of these data models based on your requirement by configuring the **lokiStack.dataModel** field in the **ClusterLogForwarder**. ViaQ is the default data model when forwarding logs to LokiStack.



NOTE

In future releases of OpenShift Logging, the default data model will change from ViaQ to OpenTelemetry.

1.3.6.1. Quick start with ViaQ

To use the default ViaQ data model, follow these steps:

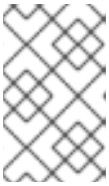
Prerequisites

- You have access to an OpenShift Container Platform cluster with **cluster-admin** permissions.
- You installed the OpenShift CLI (**oc**).
- You have access to a supported object store. For example, AWS S3, Google Cloud Storage, Azure, Swift, Minio, or OpenShift Data Foundation.

Procedure

1. Install the **Red Hat OpenShift Logging Operator**, **Loki Operator**, and **Cluster Observability Operator (COO)** from OperatorHub.
2. Create a **LokiStack** custom resource (CR) in the **openshift-logging** namespace:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  managementState: Managed
  size: 1x.extra-small
  storage:
    schemas:
    - effectiveDate: '2024-10-01'
      version: v13
    secret:
      name: logging-loki-s3
      type: s3
  storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```



NOTE

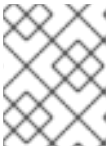
Ensure that the **logging-loki-s3** secret is created beforehand. The contents of this secret vary depending on the object storage in use. For more information, see Secrets and TLS Configuration.

3. Create a service account for the collector:

```
$ oc create sa collector -n openshift-logging
```

4. Allow the collector's service account to write data to the **LokiStack** CR:

```
$ oc adm policy add-cluster-role-to-user logging-collector-logs-writer -z collector -n openshift-logging
```



NOTE

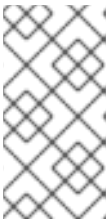
The **ClusterRole** resource is created automatically during the Cluster Logging Operator installation and does not need to be created manually.

5. To collect logs, use the service account of the collector by running the following commands:

```
$ oc adm policy add-cluster-role-to-user collect-application-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-audit-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs -z collector -n openshift-logging
```



NOTE

The example binds the collector to all three roles (application, infrastructure, and audit), but by default, only application and infrastructure logs are collected. To collect audit logs, update your **ClusterLogForwarder** configuration to include them. Assign roles based on the specific log types required for your environment.

6. Create a **UIPlugin** CR to enable the **Log** section in the **Observe** tab:

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging
spec:
  type: Logging
  logging:
    lokiStack:
      name: logging-loki
```

7. Create a **ClusterLogForwarder** CR to configure log forwarding:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: collector
  namespace: openshift-logging
spec:
  serviceAccount:
    name: collector
  outputs:
  - name: default-lokistack
    type: lokiStack
    lokiStack:
      authentication:
        token:
          from: serviceAccount
      target:
        name: logging-loki
```

```
      namespace: openshift-logging
  tls:
    ca:
      key: service-ca.crt
      configMapName: openshift-service-ca.crt
pipelines:
- name: default-logstore
  inputRefs:
  - application
  - infrastructure
  outputRefs:
  - default-lokistack
```

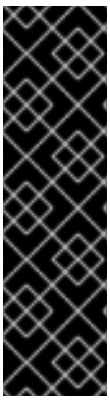
NOTE

The **dataModel** field is optional and left unset (**dataModel: ""**) by default. This allows the Cluster Logging Operator (CLO) to automatically select a data model. Currently, the CLO defaults to the ViaQ model when the field is unset, but this will change in future releases. Specifying **dataModel: ViaQ** ensures the configuration remains compatible if the default changes.

Verification

- Verify that logs are visible in the **Log** section of the **Observe** tab in the OpenShift Container Platform web console.

1.3.6.2. Quick start with OpenTelemetry



IMPORTANT

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

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

To configure OTLP ingestion and enable the OpenTelemetry data model, follow these steps:

Prerequisites

- You have access to an OpenShift Container Platform cluster with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have access to a supported object store. For example, AWS S3, Google Cloud Storage, Azure, Swift, Minio, or OpenShift Data Foundation.

Procedure

1. Install the **Red Hat OpenShift Logging Operator**, **Loki Operator**, and **Cluster Observability**

1. Install the **Red Hat OpenShift Logging Operator**, **Loki Operator**, and **Cluster Observability Operator (COO)** from OperatorHub.

2. Create a **LokiStack** custom resource (CR) in the **openshift-logging** namespace:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  managementState: Managed
  size: 1x.extra-small
  storage:
    schemas:
    - effectiveDate: '2024-10-01'
      version: v13
    secret:
      name: logging-loki-s3
      type: s3
  storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```

NOTE

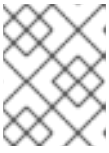
Ensure that the **logging-loki-s3** secret is created beforehand. The contents of this secret vary depending on the object storage in use. For more information, see "Secrets and TLS Configuration".

3. Create a service account for the collector:

```
$ oc create sa collector -n openshift-logging
```

4. Allow the collector's service account to write data to the **LokiStack** CR:

```
$ oc adm policy add-cluster-role-to-user logging-collector-logs-writer -z collector -n openshift-logging
```



NOTE

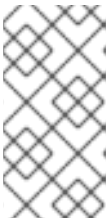
The **ClusterRole** resource is created automatically during the Cluster Logging Operator installation and does not need to be created manually.

5. To collect logs, use the service account of the collector by running the following commands:

```
$ oc adm policy add-cluster-role-to-user collect-application-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-audit-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs -z collector -n openshift-logging
```



NOTE

The example binds the collector to all three roles (application, infrastructure, and audit). By default, only application and infrastructure logs are collected. To collect audit logs, update your **ClusterLogForwarder** configuration to include them. Assign roles based on the specific log types required for your environment.

6. Create a **UIPlugin** CR to enable the **Log** section in the **Observe** tab:

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging
spec:
  type: Logging
  logging:
    lokiStack:
      name: logging-loki
```

7. Create a **ClusterLogForwarder** CR to configure log forwarding:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: collector
  namespace: openshift-logging
  annotations:
    observability.openshift.io/tech-preview-otlp-output: "enabled" 1
spec:
  serviceAccount:
    name: collector
  outputs:
  - name: loki-otlp
    type: lokiStack 2
    lokiStack:
      target:
        name: logging-loki
        namespace: openshift-logging
      dataModel: Otel 3
      authentication:
        token:
          from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - name: my-pipeline
    inputRefs:
    - application
```

```
    - infrastructure
  outputRefs:
    - loki-otlp
```

1. Use the annotation to enable the **Otel** data model, which is a Technology Preview feature.
2. Define the output type as **lokiStack**.
3. Specifies the OpenTelemetry data model.

NOTE

You cannot use **lokiStack.labelKeys** when **dataModel** is **Otel**. To achieve similar functionality when **dataModel** is **Otel**, refer to "Configuring LokiStack for OTLP data ingestion".

Verification

- To verify that OTLP is functioning correctly, complete the following steps:
  a. In the OpenShift web console, click **Observe → OpenShift Logging → LokiStack → Writes**.
  b. Check the **Distributor - Structured Metadata**section.

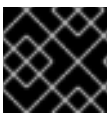
1.4. INSTALLING LOGGING

OpenShift Container Platform Operators use custom resources (CRs) to manage applications and their components. You provide high-level configuration and settings through the CR. The Operator translates high-level directives into low-level actions, based on best practices embedded within the logic of the Operator. A custom resource definition (CRD) defines a CR and lists all the configurations available to users of the Operator. Installing an Operator creates the CRDs to generate CRs.

To get started with logging, you must install the following Operators:

- Loki Operator to manage your log store.
- Red Hat OpenShift Logging Operator to manage log collection and forwarding.
- Cluster Observability Operator (COO) to manage visualization.

You can use either the OpenShift Container Platform web console or the OpenShift Container Platform CLI to install or configure logging.



IMPORTANT

You must configure the Red Hat OpenShift Logging Operator after the Loki Operator.

1.4.1. Installation by using the CLI

The following sections describe installing the Loki Operator and the Red Hat OpenShift Logging Operator by using the CLI.

1.4.1.1. Installing the Loki Operator by using the CLI

Install Loki Operator on your OpenShift Container Platform cluster to manage the log store **Loki** by using the OpenShift Container Platform command-line interface (CLI). You can deploy and configure the **Loki** log store by reconciling the resource LokiStack with the Loki Operator.

Prerequisites

- You have administrator permissions.
- You installed the OpenShift CLI (**oc**).
- You have access to a supported object store. For example: AWS S3, Google Cloud Storage, Azure, Swift, Minio, or OpenShift Data Foundation.

Procedure

1. Create a **Namespace** object for Loki Operator:

   #### Example Namespace object

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: openshift-operators-redhat (1)
     labels:
       openshift.io/cluster-monitoring: "true" (2)
   ```

   (1) You must specify **openshift-operators-redhat** as the namespace. To enable monitoring for the operator, configure Cluster Monitoring Operator to scrape metrics from the **openshift-operators-redhat** namespace and not the **openshift-operators** namespace. The **openshift-operators** namespace might contain community operators, which are untrusted and could publish a metric with the same name as an OpenShift Container Platform metric, causing conflicts.

   (2) A string value that specifies the label as shown to ensure that cluster monitoring scrapes the **openshift-operators-redhat** namespace.

2. Apply the **Namespace** object by running the following command:

   ```
   $ oc apply -f <filename>.yaml
   ```

3. Create an **OperatorGroup** object.

   #### Example OperatorGroup object

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: loki-operator
     namespace: openshift-operators-redhat (1)
   spec:
     upgradeStrategy: Default
   ```

   (1) You must specify **openshift-operators-redhat** as the namespace.

4. Apply the **OperatorGroup** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

5. Create a **Subscription** object for Loki Operator:

Example Subscription object

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat (1)
spec:
  channel: stable-6.<y> (2)
  installPlanApproval: Automatic (3)
  name: loki-operator
  source: redhat-operators (4)
  sourceNamespace: openshift-marketplace
```

(1) You must specify **openshift-operators-redhat** as the namespace.

(2) Specify **stable-6.<y>** as the channel.

(3) If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

(4) Specify **redhat-operators** as the value. If your OpenShift Container Platform cluster is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object that you created when you configured Operator Lifecycle Manager (OLM).

6. Apply the **Subscription** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

7. Create a **namespace** object for deploy the LokiStack:

Example namespace object

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging (1)
  labels:
    openshift.io/cluster-monitoring: "true" (2)
```

(1) The **openshift-logging** namespace is dedicated for all logging workloads.

(2) A string value that specifies the label, as shown, to ensure that cluster monitoring scrapes the **openshift-logging** namespace.

8. Apply the **namespace** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

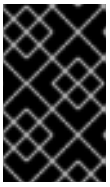
9. Create a secret with the credentials to access the object storage. For example, create a secret to access Amazon Web Services (AWS) s3.

Example Secret object

```
apiVersion: v1
kind: Secret
metadata:
  name: logging-loki-s3 (1)
  namespace: openshift-logging
stringData: (2)
  access_key_id: <access_key_id>
  access_key_secret: <access_secret>
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1
```

(1) Use the name **logging-loki-s3** to match the name used in LokiStack.

(2) For the contents of the secret see the Loki object storage section.



IMPORTANT

If there is no retention period defined on the s3 bucket or in the LokiStack custom resource (CR), then the logs are not pruned and they stay in the s3 bucket forever, which might fill up the s3 storage.

10. Apply the **Secret** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

11. Create a **LokiStack** CR:

Example LokiStack CR

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki (1)
  namespace: openshift-logging (2)
spec:
  size: 1x.small (3)
  storage:
    schemas:
    - version: v13
      effectiveDate: "<yyyy>-<mm>-<dd>" (4)
    secret:
      name: logging-loki-s3 (5)
```

```
    type: s3 6
  storageClassName: <storage_class_name> 7
  tenants:
    mode: openshift-logging 8
```

1. Use the name **logging-loki**.
2. You must specify **openshift-logging** as the namespace.
3. Specify the deployment size. Supported size options for production instances of Loki are **1x.extra-small**, **1x.small**, or **1x.medium**. Additionally, **1x.pico** is supported starting with logging 6.1.
4. For new installations this date should be set to the equivalent of "yesterday", as this will be the date from when the schema takes effect.
5. Specify the name of your log store secret.
6. Specify the corresponding storage type.
7. Specify the name of a storage class for temporary storage. For best performance, specify a storage class that allocates block storage. You can list the available storage classes for your cluster by using the **oc get storageclasses** command.
8. The **openshift-logging** mode is the default tenancy mode where a tenant is created for log types, such as audit, infrastructure, and application. This enables access control for individual users and user groups to different log streams.

12. Apply the **LokiStack** CR object by running the following command:

```
$ oc apply -f <filename>.yaml
```

Verification

- Verify the installation by running the following command:

```
$ oc get pods -n openshift-logging
```

Example output

```
$ oc get pods -n openshift-logging
NAME                                READY   STATUS    RESTARTS   AGE
logging-loki-compactor-0            1/1     Running   0          42m
logging-loki-distributor-7d7688bcb9-dvcj8   1/1     Running   0          42m
logging-loki-gateway-5f6c75f879-bl7k9       2/2     Running   0          42m
logging-loki-gateway-5f6c75f879-xhq98       2/2     Running   0          42m
logging-loki-index-gateway-0        1/1     Running   0          42m
logging-loki-ingester-0             1/1     Running   0          42m
logging-loki-querier-6b7b56bccc-2v9q4       1/1     Running   0          42m
logging-loki-query-frontend-84fb57c578-gq2f7        1/1     Running   0          42m
```

1.4.1.2. Installing Red Hat OpenShift Logging Operator by using the CLI

Install Red Hat OpenShift Logging Operator on your OpenShift Container Platform cluster to collect and forward logs to a log store by using the OpenShift CLI (**oc**).

Prerequisites

- You have administrator permissions.
- You installed the OpenShift CLI (**oc**).
- You installed and configured Loki Operator.
- You have created the **openshift-logging** namespace.

Procedure

1. Create an **OperatorGroup** object:

   Example OperatorGroup object

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: cluster-logging
     namespace: openshift-logging 1
   spec:
     upgradeStrategy: Default
   ```

   1 You must specify **openshift-logging** as the namespace.

2. Apply the **OperatorGroup** object by running the following command:

   ```
   $ oc apply -f <filename>.yaml
   ```

3. Create a **Subscription** object for Red Hat OpenShift Logging Operator:

   Example Subscription object

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: cluster-logging
     namespace: openshift-logging 1
   spec:
     channel: stable-6.<y> 2
     installPlanApproval: Automatic 3
     name: cluster-logging
     source: redhat-operators 4
     sourceNamespace: openshift-marketplace
   ```

   1 You must specify **openshift-logging** as the namespace.

   2 Specify **stable-6.<y>** as the channel.

3 If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new operator version is available in the selected channel. If the approval

4 Specify **redhat-operators** as the value. If your OpenShift Container Platform cluster is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object that you created when you configured Operator Lifecycle Manager (OLM).

4. Apply the **Subscription** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

5. Create a service account to be used by the log collector:

```
$ oc create sa logging-collector -n openshift-logging
```

6. Assign the necessary permissions to the service account for the collector to be able to collect and forward logs. In this example, the collector is provided permissions to collect logs from both infrastructure and application logs.

```
$ oc adm policy add-cluster-role-to-user logging-collector-logs-writer -z logging-collector -n openshift-logging
$ oc adm policy add-cluster-role-to-user collect-application-logs -z logging-collector -n openshift-logging
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs -z logging-collector -n openshift-logging
```

7. Create a **ClusterLogForwarder** CR:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging 1
spec:
  serviceAccount:
    name: logging-collector 2
  outputs:
  - name: lokistack-out
    type: lokiStack 3
    lokiStack:
      target: 4
        name: logging-loki
        namespace: openshift-logging
      authentication:
        token:
          from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
```

```
pipelines:
- name: infra-app-logs
  inputRefs: 5
  - application
  - infrastructure
  outputRefs:
  - lokistack-out
```

1. You must specify the **openshift-logging** namespace.
2. Specify the name of the service account created before.
3. Select the **lokiStack** output type to send logs to the **LokiStack** instance.
4. Point the **ClusterLogForwarder** to the **LokiStack** instance created earlier.
5. Select the log output types you want to send to the **LokiStack** instance.

8. Apply the **ClusterLogForwarder CR** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

Verification

1. Verify the installation by running the following command:

```
$ oc get pods -n openshift-logging
```

Example output

```
$ oc get pods -n openshift-logging
NAME                                READY   STATUS    RESTARTS   AGE
cluster-logging-operator-fb7f7cf69-8jsbq   1/1     Running   0          98m
instance-222js                      2/2     Running   0          18m
instance-g9ddv                      2/2     Running   0          18m
instance-hfqq8                      2/2     Running   0          18m
instance-sphwg                      2/2     Running   0          18m
instance-vv7zn                      2/2     Running   0          18m
instance-wk5zz                      2/2     Running   0          18m
logging-loki-compactor-0            1/1     Running   0          42m
logging-loki-distributor-7d7688bcb9-dvcj8   1/1     Running   0          42m
logging-loki-gateway-5f6c75f879-bl7k9       2/2     Running   0          42m
logging-loki-gateway-5f6c75f879-xhq98       2/2     Running   0          42m
logging-loki-index-gateway-0        1/1     Running   0          42m
logging-loki-ingester-0             1/1     Running   0          42m
logging-loki-querier-6b7b56bccc-2v9q4       1/1     Running   0          42m
logging-loki-query-frontend-84fb57c578-gq2f7   1/1     Running   0          42m
```

1.4.1.3. Installing the Logging UI plugin by using the CLI

Install the Logging UI plugin by using the command-line interface (CLI) so that you can visualize logs.

Prerequisites

- You have administrator permissions.
- You installed the OpenShift CLI (**oc**).
- You installed and configured Loki Operator.

Procedure

1. Install the Cluster Observability Operator. For more information, see Installing the Cluster Observability Operator.
2. Create a **UIPlugin** custom resource (CR):

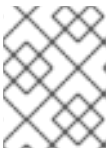
   Example UIPlugin CR

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging (1)
spec:
  type: Logging (2)
  logging:
    lokiStack:
      name: logging-loki (3)
```

   (1) Set **name** to **logging**.

   (2) Set **type** to **Logging**.

   (3) The **name** value must match the name of your LokiStack instance.



   NOTE

   If you did not install LokiStack in the **openshift-logging** namespace, set the LokiStack namespace under the **lokiStack** configuration.

3. Apply the **UIPlugin** CR object by running the following command:

```
$ oc apply -f <filename>.yaml
```

Verification

1. Access the OpenShift Container Platform web console, and refresh the page if a pop-up message instructs you to do so.
2. Navigate to the **Observe → Logs** panel, where you can run LogQL queries. You can also query logs for individual pods from the **Aggregated Logs** tab of a specific pod.

1.4.2. Installation by using the web console

The following sections describe installing the Loki Operator and the Red Hat OpenShift Logging Operator by using the web console.

1.4.2.1. Installing Logging by using the web console

Install Loki Operator on your OpenShift Container Platform cluster to manage the log store **Loki** from the OperatorHub by using the OpenShift Container Platform web console. You can deploy and configure the **Loki** log store by reconciling the resource LokiStack with the Loki Operator.

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console **Administrator** perspective, go to **Operators → OperatorHub**.
2. Type Loki Operator in the **Filter by keyword** field. Click **Loki Operator** in the list of available Operators, and then click **Install**.

IMPORTANT

The Community Loki Operator is not supported by Red Hat.

3. Select **stable-x.y** as the **Update channel**.
The Loki Operator must be deployed to the global Operator group namespace **openshift-operators-redhat**, so the **Installation mode** and **Installed Namespace** are already selected. If this namespace does not already exist, it will be created for you.
4. Select **Enable Operator-recommended cluster monitoring on this namespace.**
This option sets the **openshift.io/cluster-monitoring: "true"** label in the **Namespace** object. You must select this option to ensure that cluster monitoring scrapes the **openshift-operators-redhat** namespace.
5. For **Update approval** select **Automatic**, then click **Install**.
If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new Operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

NOTE

An Operator might display a **Failed** status before the installation completes. If the Operator install completes with an **InstallSucceeded** message, refresh the page.

6. While the Operator installs, create the namespace to which the log store will be deployed.
   a. Click **+** in the top right of the screen to access the **Import YAML** page.
   b. Add the YAML definition for the **openshift-logging** namespace:

Example namespace object

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging (1)
  labels:
    openshift.io/cluster-monitoring: "true" (2)
```

(1) The **openshift-logging** namespace is dedicated for all logging workloads.

(2) A string value that specifies the label, as shown, to ensure that cluster monitoring scrapes the **openshift-logging** namespace.

c. Click **Create**.

7. Create a secret with the credentials to access the object storage.

a. Click **+** in the top right of the screen to access the **Import YAML** page.

b. Add the YAML definition for the secret. For example, create a secret to access Amazon Web Services (AWS) s3:

Example Secret object

```
apiVersion: v1
kind: Secret
metadata:
  name: logging-loki-s3 (1)
  namespace: openshift-logging (2)
stringData: (3)
  access_key_id: <access_key_id>
  access_key_secret: <access_key>
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1
```

(1) Note down the name used for the secret **logging-loki-s3** to use it later when creating the **LokiStack** resource.

(2) Set the namespace to **openshift-logging** as that will be the namespace used to deploy **LokiStack**.

(3) For the contents of the secret see the Loki object storage section.

IMPORTANT

If there is no retention period defined on the s3 bucket or in the LokiStack custom resource (CR), then the logs are not pruned and they stay in the s3 bucket forever, which might fill up the s3 storage.

c. Click **Create**.

8. Navigate to the **Installed Operators** page. Select the Loki Operator under the **Provided APIs**

8. Navigate to the **Installed Operators** page. Select the Loki Operator under the **Provided APIs** find the **LokiStack** resource and click **Create Instance**.

9. Select **YAML view**, and then use the following template to create a **LokiStack** CR:

Example LokiStack CR

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki (1)
  namespace: openshift-logging (2)
spec:
  size: 1x.small (3)
  storage:
    schemas:
    - version: v13
      effectiveDate: "<yyyy>-<mm>-<dd>"
    secret:
      name: logging-loki-s3 (4)
      type: s3 (5)
  storageClassName: <storage_class_name> (6)
  tenants:
    mode: openshift-logging (7)
```

(1) Use the name **logging-loki**.

(2) You must specify **openshift-logging** as the namespace.

(3) Specify the deployment size. Supported size options for production instances of Loki are **1x.extra-small**, **1x.small**, or **1x.medium**. Additionally, 1x.pico is supported starting with logging 6.1.

(4) Specify the name of your log store secret.

(5) Specify the corresponding storage type.

(6) Specify the name of a storage class for temporary storage. For best performance, specify a storage class that allocates block storage. You can list the available storage classes for your cluster by using the **oc get storageclasses** command.

(7) The **openshift-logging** mode is the default tenancy mode where a tenant is created for log types, such as audit, infrastructure, and application. This enables access control for individual users and user groups to different log streams.

10. Click **Create**.

Verification

1. In the **LokiStack** tab veriy that you see your **LokiStack** instance.

2. In the **Status** column, verify that you see the message **Condition: Ready** with a green checkmark.

1.4.2.2. Installing Red Hat OpenShift Logging Operator by using the web console

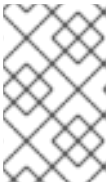
Install Red Hat OpenShift Logging Operator on your OpenShift Container Platform cluster to collect and forward logs to a log store from the OperatorHub by using the OpenShift Container Platform web console.

Prerequisites

- You have administrator permissions.
- You have access to the OpenShift Container Platform web console.
- You installed and configured Loki Operator.

Procedure

1. In the OpenShift Container Platform web console **Administrator** perspective, go to **Operators → OperatorHub**.
2. Type Red Hat OpenShift Logging Operator in the **Filter by keyword** field. Click **Red Hat OpenShift Logging Operator** in the list of available Operators, and then click **Install**.
3. Select **stable-x.y** as the **Update channel**. The latest version is already selected in the **Version** field.
   The Red Hat OpenShift Logging Operator must be deployed to the logging namespace **openshift-logging**, so the **Installation mode** and **Installed Namespace** are already selected. If this namespace does not already exist, it will be created for you.
4. Select **Enable Operator-recommended cluster monitoring on this namespace.**
   This option sets the **openshift.io/cluster-monitoring: "true"** label in the **Namespace** object. You must select this option to ensure that cluster monitoring scrapes the **openshift-logging** namespace.
5. For **Update approval** select **Automatic**, then click **Install**.
   If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.



NOTE

An Operator might display a **Failed** status before the installation completes. If the operator installation completes with an **InstallSucceeded** message, refresh the page.

6. While the operator installs, create the service account that will be used by the log collector to collect the logs.
   a. Click the **+** in the top right of the screen to access the **Import YAML** page.
   b. Enter the YAML definition for the service account.

Example ServiceAccount object

```
apiVersion: v1
kind: ServiceAccount
```

```
metadata:
  name: logging-collector 1
  namespace: openshift-logging 2
```

1 Note down the name used for the service account **logging-collector** to use it later when creating the **ClusterLogForwarder** resource.

2 Set the namespace to **openshift-logging** because that is the namespace for deploying the **ClusterLogForwarder** resource.

c. Click the **Create** button.

7. Create the **ClusterRoleBinding** objects to grant the necessary permissions to the log collector for accessing the logs that you want to collect and to write the log store, for example infrastructure and application logs.

a. Click the **+** in the top right of the screen to access the **Import YAML** page.

b. Enter the YAML definition for the **ClusterRoleBinding** resources.

Example ClusterRoleBinding resources

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: logging-collector:write-logs
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: logging-collector-logs-writer 1
subjects:
- kind: ServiceAccount
  name: logging-collector
  namespace: openshift-logging
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: logging-collector:collect-application
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: collect-application-logs 2
subjects:
- kind: ServiceAccount
  name: logging-collector
  namespace: openshift-logging
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: logging-collector:collect-infrastructure
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
```

```
    name: collect-infrastructure-logs 3
  subjects:
  - kind: ServiceAccount
    name: logging-collector
    namespace: openshift-logging
```

1. The cluster role to allow the log collector to write logs to LokiStack.
2. The cluster role to allow the log collector to collect logs from applications.
3. The cluster role to allow the log collector to collect logs from infrastructure.

c. Click the **Create** button.

8. Go to the **Operators → Installed Operators** page. Select the operator and click the **All instances** tab.
9. After granting the necessary permissions to the service account, navigate to the **Installed Operators** page. Select the Red Hat OpenShift Logging Operator under the **Provided APIs**, find the **ClusterLogForwarder** resource and click **Create Instance**.
10. Select **YAML view**, and then use the following template to create a **ClusterLogForwarder** CR:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging 1
spec:
  serviceAccount:
    name: logging-collector 2
  outputs:
  - name: lokistack-out
    type: lokiStack 3
    lokiStack:
      target: 4
        name: logging-loki
        namespace: openshift-logging
      authentication:
        token:
          from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - name: infra-app-logs
    inputRefs: 5
    - application
    - infrastructure
    outputRefs:
    - lokistack-out
```

1. You must specify **openshift-logging** as the namespace.
2. Specify the name of the service account created earlier.
3. Select the **lokiStack** output type to send logs to the **LokiStack** instance.
4. Point the **ClusterLogForwarder** to the **LokiStack** instance created earlier.
5. Select the log output types you want to send to the **LokiStack** instance.

11. Click **Create**.

Verification

1. In the **ClusterLogForwarder** tab verify that you see your **ClusterLogForwarder** instance.
2. In the **Status** column, verify that you see the messages:
   - **Condition: observability.openshift.io/Authorized**
   - **observability.openshift.io/Valid, Ready**

1.4.2.3. Installing the Logging UI plugin by using the web console

Install the Logging UI plugin by using the web console so that you can visualize logs.

Prerequisites

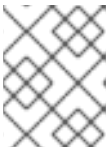
- You have administrator permissions.
- You have access to the OpenShift Container Platform web console.
- You installed and configured Loki Operator.

Procedure

1. Install the Cluster Observability Operator. For more information, see Installing the Cluster Observability Operator.
2. Navigate to the **Installed Operators** page. Under **Provided APIs**, select **ClusterObservabilityOperator**. Find the **UIPlugin** resource and click **Create Instance**.
3. Select the YAML view, and then use the following template to create a **UIPlugin** custom resource (CR):

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging 1
spec:
  type: Logging 2
  logging:
    lokiStack:
      name: logging-loki 3
```

1. Set **name** to **logging**.
2. Set **type** to **Logging**.
3. The **name** value must match the name of your LokiStack instance.



NOTE

If you did not install LokiStack in the **openshift-logging** namespace, set the LokiStack namespace under the **lokiStack** configuration.

4. Click **Create**.

Verification

1. Refresh the page when a pop-up message instructs you to do so.
2. Navigate to the **Observe → Logs** panel, where you can run LogQL queries. You can also query logs for individual pods from the **Aggregated Logs** tab of a specific pod.

Additional resources

- About OVN-Kubernetes network policy

1.5. CONFIGURING LOG FORWARDING

The **ClusterLogForwarder** (CLF) allows users to configure forwarding of logs to various destinations. It provides a flexible way to select log messages from different sources, send them through a pipeline that can transform or filter them, and forward them to one or more outputs.

Key Functions of the ClusterLogForwarder

- Selects log messages using inputs
- Forwards logs to external destinations using outputs
- Filters, transforms, and drops log messages using filters
- Defines log forwarding pipelines connecting inputs, filters and outputs

1.5.1. Setting up log collection

This release of Cluster Logging requires administrators to explicitly grant log collection permissions to the service account associated with **ClusterLogForwarder**. This was not required in previous releases for the legacy logging scenario consisting of a **ClusterLogging** and, optionally, a **ClusterLogForwarder.logging.openshift.io** resource.

The Red Hat OpenShift Logging Operator provides **collect-audit-logs**, **collect-application-logs**, and **collect-infrastructure-logs** cluster roles, which enable the collector to collect audit logs, application logs, and infrastructure logs respectively.

Setup log collection by binding the required cluster roles to your service account.

1.5.1.1. Legacy service accounts

To use the existing legacy service account **logcollector**, create the following **ClusterRoleBinding**:

```
$ oc adm policy add-cluster-role-to-user collect-application-logs system:serviceaccount:openshift-logging:logcollector
```

```
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs system:serviceaccount:openshift-logging:logcollector
```

Additionally, create the following **ClusterRoleBinding** if collecting audit logs:

```
$ oc adm policy add-cluster-role-to-user collect-audit-logs system:serviceaccount:openshift-logging:logcollector
```

1.5.1.2. Creating service accounts

Prerequisites

- The Red Hat OpenShift Logging Operator is installed in the **openshift-logging** namespace.
- You have administrator permissions.

Procedure

1. Create a service account for the collector. If you want to write logs to storage that requires a token for authentication, you must include a token in the service account.
2. Bind the appropriate cluster roles to the service account:

Example binding command

```
$ oc adm policy add-cluster-role-to-user <cluster_role_name> system:serviceaccount:<namespace_name>:<service_account_name>
```

1.5.1.2.1. Cluster Role Binding for your Service Account

The role_binding.yaml file binds the ClusterLogging operator's ClusterRole to a specific ServiceAccount, allowing it to manage Kubernetes resources cluster-wide.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: manager-rolebinding
roleRef:                                  1
  apiGroup: rbac.authorization.k8s.io     2
  kind: ClusterRole                       3
  name: cluster-logging-operator          4
subjects:                                 5
  - kind: ServiceAccount                  6
    name: cluster-logging-operator        7
    namespace: openshift-logging          8
```

1. roleRef: References the ClusterRole to which the binding applies.
2. apiGroup: Indicates the RBAC API group, specifying that the ClusterRole is part of Kubernetes' RBAC system.
3. kind: Specifies that the referenced role is a ClusterRole, which applies cluster-wide.
4. name: The name of the ClusterRole being bound to the ServiceAccount, here cluster-logging-operator.
5. subjects: Defines the entities (users or service accounts) that are being granted the permissions from the ClusterRole.
6. kind: Specifies that the subject is a ServiceAccount.
7. Name: The name of the ServiceAccount being granted the permissions.
8. namespace: Indicates the namespace where the ServiceAccount is located.

1.5.1.2.2. Writing application logs

The write-application-logs-clusterrole.yaml file defines a ClusterRole that grants permissions to write application logs to the Loki logging application.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-logging-write-application-logs
rules:                                          (1)
  - apiGroups:                                  (2)
      - loki.grafana.com                        (3)
    resources:                                  (4)
      - application                             (5)
    resourceNames:                              (6)
      - logs                                    (7)
    verbs:                                      (8)
      - create                                  (9)
```

1. rules: Specifies the permissions granted by this ClusterRole.
2. apiGroups: Refers to the API group loki.grafana.com, which relates to the Loki logging system.
3. loki.grafana.com: The API group for managing Loki-related resources.
4. resources: The resource type that the ClusterRole grants permission to interact with.
5. application: Refers to the application resources within the Loki logging system.
6. resourceNames: Specifies the names of resources that this role can manage.
7. logs: Refers to the log resources that can be created.
8. verbs: The actions allowed on the resources.

9 create: Grants permission to create new logs in the Loki system.

1.5.1.2.3. Writing audit logs

The write-audit-logs-clusterrole.yaml file defines a ClusterRole that grants permissions to create audit logs in the Loki logging system.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-logging-write-audit-logs
rules:                          1
  - apiGroups:                  2
      - loki.grafana.com        3
    resources:                  4
      - audit                   5
    resourceNames:              6
      - logs                    7
    verbs:                      8
      - create                  9
```

1 rules: Defines the permissions granted by this ClusterRole.

2 apiGroups: Specifies the API group loki.grafana.com.

3 loki.grafana.com: The API group responsible for Loki logging resources.

4 resources: Refers to the resource type this role manages, in this case, audit.

5 audit: Specifies that the role manages audit logs within Loki.

6 resourceNames: Defines the specific resources that the role can access.

7 logs: Refers to the logs that can be managed under this role.

8 verbs: The actions allowed on the resources.

9 create: Grants permission to create new audit logs.

1.5.1.2.4. Writing infrastructure logs

The write-infrastructure-logs-clusterrole.yaml file defines a ClusterRole that grants permission to create infrastructure logs in the Loki logging system.

Sample YAML

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-logging-write-infrastructure-logs
rules:                          1
  - apiGroups:                  2
```

```
  - loki.grafana.com                    3
 resources:                             4
  - infrastructure                      5
 resourceNames:                         6
  - logs                                7
 verbs:                                 8
  - create                              9
```

1. rules: Specifies the permissions this ClusterRole grants.
2. apiGroups: Specifies the API group for Loki-related resources.
3. loki.grafana.com: The API group managing the Loki logging system.
4. resources: Defines the resource type that this role can interact with.
5. infrastructure: Refers to infrastructure-related resources that this role manages.
6. resourceNames: Specifies the names of resources this role can manage.
7. logs: Refers to the log resources related to infrastructure.
8. verbs: The actions permitted by this role.
9. create: Grants permission to create infrastructure logs in the Loki system.

1.5.1.2.5. ClusterLogForwarder editor role

The clusterlogforwarder-editor-role.yaml file defines a ClusterRole that allows users to manage ClusterLogForwarders in OpenShift.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: clusterlogforwarder-editor-role
rules:                                  1
  - apiGroups:                          2
      - observability.openshift.io      3
    resources:                          4
      - clusterlogforwarders            5
    verbs:                              6
      - create                          7
      - delete                          8
      - get                             9
      - list                            10
      - patch                           11
      - update                          12
      - watch                           13
```

1. rules: Specifies the permissions this ClusterRole grants.
2. apiGroups: Refers to the OpenShift-specific API group

3. obervability.openshift.io: The API group for managing observability resources, like logging.
4. resources: Specifies the resources this role can manage.
5. clusterlogforwarders: Refers to the log forwarding resources in OpenShift.
6. verbs: Specifies the actions allowed on the ClusterLogForwarders.
7. create: Grants permission to create new ClusterLogForwarders.
8. delete: Grants permission to delete existing ClusterLogForwarders.
9. get: Grants permission to retrieve information about specific ClusterLogForwarders.
10. list: Allows listing all ClusterLogForwarders.
11. patch: Grants permission to partially modify ClusterLogForwarders.
12. update: Grants permission to update existing ClusterLogForwarders.
13. watch: Grants permission to monitor changes to ClusterLogForwarders.

1.5.2. Modifying log level in collector

To modify the log level in the collector, you can set the **observability.openshift.io/log-level** annotation to **trace**, **debug**, **info**, **warn**, **error**, and **off**.

Example log level annotation

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: collector
  annotations:
    observability.openshift.io/log-level: debug
# ...
```

1.5.3. Managing the Operator

The **ClusterLogForwarder** resource has a **managementState** field that controls whether the operator actively manages its resources or leaves them Unmanaged:

Managed

(default) The operator will drive the logging resources to match the desired state in the CLF spec.

Unmanaged

The operator will not take any action related to the logging components.

This allows administrators to temporarily pause log forwarding by setting **managementState** to **Unmanaged**.

1.5.4. Structure of the ClusterLogForwarder

The CLF has a **spec** section that contains the following key components:

Inputs

Select log messages to be forwarded. Built-in input types **application**, **infrastructure** and **audit** forward logs from different parts of the cluster. You can also define custom inputs.

Outputs

Define destinations to forward logs to. Each output has a unique name and type-specific configuration.

Pipelines

Define the path logs take from inputs, through filters, to outputs. Pipelines have a unique name and consist of a list of input, output and filter names.

Filters

Transform or drop log messages in the pipeline. Users can define filters that match certain log fields and drop or modify the messages. Filters are applied in the order specified in the pipeline.

1.5.4.1. Inputs

Inputs are configured in an array under **spec.inputs**. There are three built-in input types:

application

Selects logs from all application containers, excluding those in infrastructure namespaces.

infrastructure

Selects logs from nodes and from infrastructure components running in the following namespaces:

- **default**
- **kube**
- **openshift**
- Containing the **kube-** or **openshift-** prefix

audit

Selects logs from the OpenShift API server audit logs, Kubernetes API server audit logs, ovn audit logs, and node audit logs from auditd.

Users can define custom inputs of type **application** that select logs from specific namespaces or using pod labels.

1.5.4.2. Outputs

Outputs are configured in an array under **spec.outputs**. Each output must have a unique name and a type. Supported types are:

azureMonitor

Forwards logs to Azure Monitor.

cloudwatch

Forwards logs to AWS CloudWatch.

elasticsearch

Forwards logs to an external Elasticsearch instance.

googleCloudLogging

Forwards logs to Google Cloud Logging.

http

Forwards logs to a generic HTTP endpoint.

kafka

Forwards logs to a Kafka broker.

loki

Forwards logs to a Loki logging backend.

lokistack

Forwards logs to the logging supported combination of Loki and web proxy with OpenShift Container Platform authentication integration. LokiStack's proxy uses OpenShift Container Platform authentication to enforce multi-tenancy

otlp

Forwards logs using the OpenTelemetry Protocol.

splunk

Forwards logs to Splunk.

syslog

Forwards logs to an external syslog server.

Each output type has its own configuration fields.

1.5.5. Configuring OTLP output

Cluster administrators can use the OpenTelemetry Protocol (OTLP) output to collect and forward logs to OTLP receivers. The OTLP output uses the specification defined by the OpenTelemetry Observability framework to send data over HTTP with JSON encoding.

IMPORTANT

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

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

Procedure

- Create or edit a **ClusterLogForwarder** custom resource (CR) to enable forwarding using OTLP by adding the following annotation:

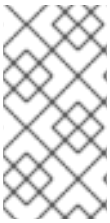
  Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  annotations:
    observability.openshift.io/tech-preview-otlp-output: "enabled" 1
```

```
  name: clf-otlp
spec:
  serviceAccount:
    name: <service_account_name>
  outputs:
  - name: otlp
    type: otlp
    otlp:
      tuning:
        compression: gzip
        deliveryMode: AtLeastOnce
        maxRetryDuration: 20
        maxWrite: 10M
        minRetryDuration: 5
      url: <otlp_url> 2
  pipelines:
  - inputRefs:
    - application
    - infrastructure
    - audit
    name: otlp-logs
    outputRefs:
    - otlp
```

1 Use this annotation to enable the OpenTelemetry Protocol (OTLP) output, which is a Technology Preview feature.

2 This URL must be absolute and is a placeholder for the OTLP endpoint where logs are sent.



NOTE

The OTLP output uses the OpenTelemetry data model, which is different from the ViaQ data model that is used by other output types. It adheres to the OTLP using OpenTelemetry Semantic Conventions defined by the OpenTelemetry Observability framework.

1.5.5.1. Pipelines

Pipelines are configured in an array under **spec.pipelines**. Each pipeline must have a unique name and consists of:

inputRefs

Names of inputs whose logs should be forwarded to this pipeline.

outputRefs

Names of outputs to send logs to.

filterRefs

(optional) Names of filters to apply.

The order of filterRefs matters, as they are applied sequentially. Earlier filters can drop messages that will not be processed by later filters.

1.5.5.2. Filters

Filters are configured in an array under **spec.filters**. They can match incoming log messages based on the value of structured fields and modify or drop them.

Administrators can configure the following types of filters:

1.5.6. Enabling multi-line exception detection

Enables multi-line error detection of container logs.

WARNING

Enabling this feature could have performance implications and may require additional computing resources or alternate logging solutions.

Log parsers often incorrectly identify separate lines of the same exception as separate exceptions. This leads to extra log entries and an incomplete or inaccurate view of the traced information.

Example java exception

```
java.lang.NullPointerException: Cannot invoke "String.toString()" because "<param1>" is null
    at testjava.Main.handle(Main.java:47)
    at testjava.Main.printMe(Main.java:19)
    at testjava.Main.main(Main.java:10)
```

- To enable logging to detect multi-line exceptions and reassemble them into a single log entry, ensure that the **ClusterLogForwarder** Custom Resource (CR) contains a **detectMultilineErrors** field under the **.spec.filters**.

Example ClusterLogForwarder CR

```
apiVersion: "observability.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name>
  namespace: <log_forwarder_namespace>
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: <name>
    type: detectMultilineException
  pipelines:
    - inputRefs:
        - <input-name>
      name: <pipeline-name>
      filterRefs:
        - <filter-name>
      outputRefs:
        - <output-name>
```

1.5.6.1. Details

When log messages appear as a consecutive sequence forming an exception stack trace, they are combined into a single, unified log record. The first log message's content is replaced with the concatenated content of all the message fields in the sequence.

The collector supports the following languages:

- Java
- JS
- Ruby
- Python
- Golang
- PHP
- Dart

1.5.7. Forwarding logs over HTTP

To enable forwarding logs over HTTP, specify **http** as the output type in the **ClusterLogForwarder** custom resource (CR).

Procedure

- Create or edit the **ClusterLogForwarder** CR using the template below:

  Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name>
  namespace: <log_forwarder_namespace>
spec:
  managementState: Managed
  outputs:
  - name: <output_name>
    type: http
    http:
      headers: 1
        h1: v1
        h2: v2
      authentication:
        username:
          key: username
          secretName: <http_auth_secret>
        password:
          key: password
          secretName: <http_auth_secret>
      timeout: 300
      proxyURL: <proxy_url> 2
```

```
    url: <url> 3
  tls:
    insecureSkipVerify: 4
    ca:
      key: <ca_certificate>
      secretName: <secret_name> 5
pipelines:
  - inputRefs:
      - application
    name: pipe1
    outputRefs:
      - <output_name> 6
serviceAccount:
  name: <service_account_name> 7
```

1. Additional headers to send with the log record.
2. Optional: URL of the HTTP/HTTPS proxy that should be used to forward logs over http or https from this output. This setting overrides any default proxy settings for the cluster or the node.
3. Destination address for logs.
4. Values are either **true** or **false**.
5. Secret name for destination credentials.
6. This value should be the same as the output name.
7. The name of your service account.

1.5.8. Forwarding logs using the syslog protocol

You can use the syslog RFC3164 or RFC5424 protocol to send a copy of your logs to an external log aggregator that is configured to accept the protocol instead of, or in addition to, the default Elasticsearch log store. You are responsible for configuring the external log aggregator, such as a syslog server, to receive the logs from OpenShift Container Platform.

To configure log forwarding using the syslog protocol, you must create a **ClusterLogForwarder** custom resource (CR) with one or more outputs to the syslog servers, and pipelines that use those outputs. The syslog output can use a UDP, TCP, or TLS connection.

Prerequisites

- You must have a logging server that is configured to receive the logging data using the specified protocol or format.

Procedure

1. Create or edit a YAML file that defines the **ClusterLogForwarder** CR object:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
```

```
  name: collector
spec:
  managementState: Managed
  outputs:
  - name: rsyslog-east (1)
    syslog:
      appName: <app_name> (2)
      enrichment: KubernetesMinimal
      facility: <facility_value> (3)
      msgId: <message_ID> (4)
      payloadKey: <record_field> (5)
      procId: <process_ID> (6)
      rfc: <RFC3164_or_RFC5424> (7)
      severity: informational (8)
      tuning:
        deliveryMode: <AtLeastOnce_or_AtMostOnce> (9)
      url: <url> (10)
    tls: (11)
      ca:
        key: ca-bundle.crt
        secretName: syslog-secret
    type: syslog
  pipelines:
  - inputRefs: (12)
    - application
    name: syslog-east (13)
    outputRefs:
    - rsyslog-east
  serviceAccount: (14)
    name: logcollector
```

(1) Specify a name for the output.

(2) Optional: Specify the value for the **APP-NAME** part of the syslog message header. The value must conform with The Syslog Protocol. The value can be a combination of static and dynamic values consisting of field paths followed by ||, and then followed by another field path or a static value. The maximum length of the final values is truncated to 48 characters. You must encase a dynamic value curly brackets and the value must be followed with a static fallback value separated with ||. Static values can only contain alphanumeric characters along with dashes, underscores, dots and forward slashes. Example value: <value1>-{.<value2>||"none"}.

(3) Optional: Specify the value for **Facility** part of the syslog-msg header.

(4) Optional: Specify the value for **MSGID** part of the syslog-msg header. The value can be a combination of static and dynamic values consisting of field paths followed by ||, and then followed by another field path or a static value. The maximum length of the final values is truncated to 32 characters. You must encase a dynamic value curly brackets and the value must be followed with a static fallback value separated with ||. Static values can only contain alphanumeric characters along with dashes, underscores, dots and forward slashes. Example value: <value1>-{.<value2>||"none"}.

(5) Optional: Specify the record field to use as the payload. The **payloadKey** value must be a single field path encased in single curly brackets **{}**. Example: {.<value>}.

6 Optional: Specify the value for the **PROCID** part of the syslog message header. The value must conform with The Syslog Protocol. The value can be a combination of static and

7 Optional: Set the RFC that the generated messages conform to. The value can be **RFC3164** or **RFC5424**.

8 Optional: Set the severity level for the message. For more information, see The Syslog Protocol.

9 Optional: Set the delivery mode for log forwarding. The value can be either **AtLeastOnce**, or **AtMostOnce**.

10 Specify the absolute URL with a scheme. Valid schemes are: **tcp**, **tls**, and **udp**. For example: **tls://syslog-receiver.example.com:6514**.

11 Specify the settings for controlling options of the transport layer security (TLS) client connections.

12 Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.

13 Specify a name for the pipeline.

14 The name of your service account.

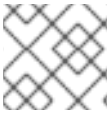
2. Create the CR object:

```
$ oc create -f <filename>.yaml
```

1.5.8.1. Adding log source information to the message output

You can add **namespace_name**, **pod_name**, and **container_name** elements to the **message** field of the record by adding the **enrichment** field to your **ClusterLogForwarder** custom resource (CR).

```
# ...
  spec:
    outputs:
    - name: syslogout
      syslog:
        enrichment: KubernetesMinimal
        facility: user
        payloadKey: message
        rfc: RFC3164
        severity: debug
      type: syslog
      url: tls://syslog-receiver.example.com:6514
    pipelines:
    - inputRefs:
      - application
      name: test-app
      outputRefs:
      - syslogout
# ...
```



NOTE

This configuration is compatible with both RFC3164 and RFC5424.

Example syslog message output with enrichment: None

```
2025-03-03T11:48:01+00:00  example-worker-x  syslogsyslogserverd846bb9b: {...}
```

Example syslog message output with enrichment: KubernetesMinimal

```
2025-03-03T11:48:01+00:00  example-worker-x  syslogsyslogserverd846bb9b:
namespace_name=cakephp-project container_name=mysql pod_name=mysql-1-wr96h,message:
{...}
```

1.5.9. Configuring content filters to drop unwanted log records

When the **drop** filter is configured, the log collector evaluates log streams according to the filters before forwarding. The collector drops unwanted log records that match the specified configuration.

Procedure

1. Add a configuration for a filter to the **filters** spec in the **ClusterLogForwarder** CR. The following example shows how to configure the **ClusterLogForwarder** CR to drop log records based on regular expressions:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: <filter_name>
    type: drop (1)
    drop: (2)
    - test: (3)
      - field: .kubernetes.labels."foo-bar/baz" (4)
        matches: .+ (5)
      - field: .kubernetes.pod_name
        notMatches: "my-pod" (6)
  pipelines:
  - name: <pipeline_name> (7)
    filterRefs: ["<filter_name>"]
# ...
```

(1) Specifies the type of filter. The **drop** filter drops log records that match the filter configuration.

(2) Specifies configuration options for applying the **drop** filter.

3 Specifies the configuration for tests that are used to evaluate whether a log record is dropped.

- If all the conditions specified for a test are true, the test passes and the log record is dropped.
- When multiple tests are specified for the **drop** filter configuration, if any of the tests pass, the record is dropped.
- If there is an error evaluating a condition, for example, the field is missing from the log record being evaluated, that condition evaluates to false.

4 Specifies a dot-delimited field path, which is a path to a field in the log record. The path can contain alpha-numeric characters and underscores (**a-zA-Z0-9_**), for example, **.kubernetes.namespace_name**. If segments contain characters outside of this range, the segment must be in quotes, for example, **.kubernetes.labels."foo.bar-bar/baz"**. You can include multiple field paths in a single **test** configuration, but they must all evaluate to true for the test to pass and the **drop** filter to be applied.

5 Specifies a regular expression. If log records match this regular expression, they are dropped. You can set either the **matches** or **notMatches** condition for a single **field** path, but not both.

6 Specifies a regular expression. If log records do not match this regular expression, they are dropped. You can set either the **matches** or **notMatches** condition for a single **field** path, but not both.

7 Specifies the pipeline that the **drop** filter is applied to.

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

Additional examples

The following additional example shows how you can configure the **drop** filter to only keep higher priority log records:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: important
    type: drop
    drop:
    - test:
      - field: .message
        notMatches: "(?i)critical|error"
      - field: .level
        matches: "info|warning"
# ...
```

In addition to including multiple field paths in a single **test** configuration, you can also include additional tests that are treated as *OR* checks. In the following example, records are dropped if either **test** configuration evaluates to true. However, for the second **test** configuration, both field specs must be true for it to be evaluated to true:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: important
    type: drop
    drop:
    - test:
      - field: .kubernetes.namespace_name
        matches: "^open"
    - test:
      - field: .log_type
        matches: "application"
      - field: .kubernetes.pod_name
        notMatches: "my-pod"
# ...
```

1.5.10. Overview of API audit filter

OpenShift API servers generate audit events for each API call, detailing the request, response, and the identity of the requester, leading to large volumes of data. The API Audit filter uses rules to enable the exclusion of non-essential events and the reduction of event size, facilitating a more manageable audit trail. Rules are checked in order, and checking stops at the first match. The amount of data that is included in an event is determined by the value of the **level** field:

- **None**: The event is dropped.
- **Metadata**: Audit metadata is included, request and response bodies are removed.
- **Request**: Audit metadata and the request body are included, the response body is removed.
- **RequestResponse**: All data is included: metadata, request body and response body. The response body can be very large. For example, **oc get pods -A** generates a response body containing the YAML description of every pod in the cluster.

The **ClusterLogForwarder** custom resource (CR) uses the same format as the standard Kubernetes audit policy, while providing the following additional functions:

Wildcards

Names of users, groups, namespaces, and resources can have a leading or trailing * asterisk character. For example, the namespace **openshift-*** matches **openshift-apiserver** or **openshift-authentication**. Resource */**status** matches **Pod**/**status** or **Deployment**/**status**.

Default Rules

Events that do not match any rule in the policy are filtered as follows:

- Read-only system events such as **get**, **list**, and **watch** are dropped.

- Service account write events that occur within the same namespace as the service account are dropped.
- All other events are forwarded, subject to any configured rate limits.

To disable these defaults, either end your rules list with a rule that has only a **level** field or add an empty rule.

Omit Response Codes

A list of integer status codes to omit. You can drop events based on the HTTP status code in the response by using the **OmitResponseCodes** field, which lists HTTP status codes for which no events are created. The default value is **[404, 409, 422, 429]**. If the value is an empty list, **[]**, then no status codes are omitted.

The **ClusterLogForwarder** CR audit policy acts in addition to the OpenShift Container Platform audit policy. The **ClusterLogForwarder** CR audit filter changes what the log collector forwards and provides the ability to filter by verb, user, group, namespace, or resource. You can create multiple filters to send different summaries of the same audit stream to different places. For example, you can send a detailed stream to the local cluster log store and a less detailed stream to a remote site.

NOTE

You must have a cluster role **collect-audit-logs** to collect the audit logs. The following example provided is intended to illustrate the range of rules possible in an audit policy and is not a recommended configuration.

Example audit policy

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name>
  namespace: <log_forwarder_namespace>
spec:
  serviceAccount:
    name: <service_account_name>
  pipelines:
    - name: my-pipeline
      inputRefs: audit (1)
      filterRefs: my-policy (2)
  filters:
    - name: my-policy
      type: kubeAPIAudit
      kubeAPIAudit:
        # Don't generate audit events for all requests in RequestReceived stage.
        omitStages:
          - "RequestReceived"

        rules:
          # Log pod changes at RequestResponse level
          - level: RequestResponse
            resources:
            - group: ""
              resources: ["pods"]
```

```
    # Log "pods/log", "pods/status" at Metadata level
    - level: Metadata
      resources:
      - group: ""
        resources: ["pods/log", "pods/status"]

    # Don't log requests to a configmap called "controller-leader"
    - level: None
      resources:
      - group: ""
        resources: ["configmaps"]
        resourceNames: ["controller-leader"]

    # Don't log watch requests by the "system:kube-proxy" on endpoints or services
    - level: None
      users: ["system:kube-proxy"]
      verbs: ["watch"]
      resources:
      - group: "" # core API group
        resources: ["endpoints", "services"]

    # Don't log authenticated requests to certain non-resource URL paths.
    - level: None
      userGroups: ["system:authenticated"]
      nonResourceURLs:
      - "/api*" # Wildcard matching.
      - "/version"

    # Log the request body of configmap changes in kube-system.
    - level: Request
      resources:
      - group: "" # core API group
        resources: ["configmaps"]
      # This rule only applies to resources in the "kube-system" namespace.
      # The empty string "" can be used to select non-namespaced resources.
      namespaces: ["kube-system"]

    # Log configmap and secret changes in all other namespaces at the Metadata level.
    - level: Metadata
      resources:
      - group: "" # core API group
        resources: ["secrets", "configmaps"]

    # Log all other resources in core and extensions at the Request level.
    - level: Request
      resources:
      - group: "" # core API group
      - group: "extensions" # Version of group should NOT be included.

    # A catch-all rule to log all other requests at the Metadata level.
    - level: Metadata
```

1 The log types that are collected. The value for this field can be **audit** for audit logs, **application** for application logs, **infrastructure** for infrastructure logs, or a named input that has been defined for your application.

2 The name of your audit policy.

1.5.11. Filtering application logs at input by including the label expressions or a matching label key and values

You can include the application logs based on the label expressions or a matching label key and its values by using the **input** selector.

Procedure

1. Add a configuration for a filter to the **input** spec in the **ClusterLogForwarder** CR. The following example shows how to configure the **ClusterLogForwarder** CR to include logs based on label expressions or matched label key/values:

   Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  inputs:
    - name: mylogs
      application:
        selector:
          matchExpressions:
          - key: env 1
            operator: In 2
            values: ["prod", "qa"] 3
          - key: zone
            operator: NotIn
            values: ["east", "west"]
          matchLabels: 4
            app: one
            name: app1
      type: application
# ...
```

   1 Specifies the label key to match.

   2 Specifies the operator. Valid values include: **In**, **NotIn**, **Exists**, and **DoesNotExist**.

   3 Specifies an array of string values. If the **operator** value is either **Exists** or **DoesNotExist**, the value array must be empty.

   4 Specifies an exact key or value mapping.

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

1.5.12. Configuring content filters to prune log records

When the **prune** filter is configured, the log collector evaluates log streams according to the filters before forwarding. The collector prunes log records by removing low value fields such as pod annotations.

Procedure

1. Add a configuration for a filter to the **prune** spec in the **ClusterLogForwarder** CR.
   The following example shows how to configure the **ClusterLogForwarder** CR to prune log records based on field paths:

IMPORTANT

If both are specified, records are pruned based on the **notIn** array first, which takes precedence over the **in** array. After records have been pruned by using the **notIn** array, they are then pruned by using the **in** array.

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: <filter_name>
    type: prune (1)
    prune: (2)
      in: [.kubernetes.annotations, .kubernetes.namespace_id] (3)
      notIn: [.kubernetes,.log_type,.message,."@timestamp"] (4)
  pipelines:
  - name: <pipeline_name> (5)
    filterRefs: ["<filter_name>"]
# ...
```

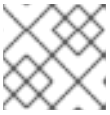
(1) Specify the type of filter. The **prune** filter prunes log records by configured fields.

(2) Specify configuration options for applying the **prune** filter. The **in** and **notIn** fields are specified as arrays of dot-delimited field paths, which are paths to fields in log records. These paths can contain alpha-numeric characters and underscores (**a-zA-Z0-9_**), for example, **.kubernetes.namespace_name**. If segments contain characters outside of this range, the segment must be in quotes, for example, **.kubernetes.labels."foo.bar-bar/baz"**.

(3) Optional: Any fields that are specified in this array are removed from the log record.

(4) Optional: Any fields that are not specified in this array are removed from the log record.

(5) Specify the pipeline that the **prune** filter is applied to.



NOTE

The filters exempts the **log_type**, **.log_source**, and **.message** fields.

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

1.5.13. Filtering the audit and infrastructure log inputs by source

You can define the list of **audit** and **infrastructure** sources to collect the logs by using the **input** selector.

Procedure

1. Add a configuration to define the **audit** and **infrastructure** sources in the **ClusterLogForwarder** CR.
   The following example shows how to configure the **ClusterLogForwarder** CR to define **audit** and **infrastructure** sources:

   Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  inputs:
    - name: mylogs1
      type: infrastructure
      infrastructure:
        sources: 1
          - node
    - name: mylogs2
      type: audit
      audit:
        sources: 2
          - kubeAPI
          - openshiftAPI
          - ovn
# ...
```

1. Specifies the list of infrastructure sources to collect. The valid sources include:
   - **node**: Journal log from the node
   - **container**: Logs from the workloads deployed in the namespaces
2. Specifies the list of audit sources to collect. The valid sources include:
   - **kubeAPI**: Logs from the Kubernetes API servers
   - **openshiftAPI**: Logs from the OpenShift API servers

- **auditd**: Logs from a node auditd service
- **ovn**: Logs from an open virtual network service

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

1.5.14. Filtering application logs at input by including or excluding the namespace or container name

You can include or exclude the application logs based on the namespace and container name by using the **input** selector.

Procedure

1. Add a configuration to include or exclude the namespace and container names in the **ClusterLogForwarder** CR.
   The following example shows how to configure the **ClusterLogForwarder** CR to include or exclude namespaces and container names:

   Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  inputs:
    - name: mylogs
      application:
        includes:
          - namespace: "my-project" 1
            container: "my-container" 2
        excludes:
          - container: "other-container*" 3
            namespace: "other-namespace" 4
      type: application
# ...
```

1. Specifies that the logs are only collected from these namespaces.
2. Specifies that the logs are only collected from these containers.
3. Specifies the pattern of namespaces to ignore when collecting the logs.
4. Specifies the set of containers to ignore when collecting the logs.

NOTE

The **excludes** field takes precedence over the **includes** field.

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

1.6. STORING LOGS WITH LOKISTACK

You can configure a **LokiStack** custom resource (CR) to store application, audit, and infrastructure-related logs.

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system offered as a GA log store for logging for Red Hat OpenShift that can be visualized with the OpenShift Observability UI. The Loki configuration provided by OpenShift Logging is a short-term log store designed to enable users to perform fast troubleshooting with the collected logs. For that purpose, the logging for Red Hat OpenShift configuration of Loki has short-term storage, and is optimized for very recent queries. For long-term storage or queries over a long time period, users should look to log stores external to their cluster.

1.6.1. Loki deployment sizing

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

The **1x.pico** configuration defines a single Loki deployment with minimal resource and limit requirements, offering high availability (HA) support for all Loki components. This configuration is suited for deployments that do not require a single replication factor or auto-compaction.

Disk requests are similar across size configurations, allowing customers to test different sizes to determine the best fit for their deployment needs.

IMPORTANT

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

Table 1.2. Loki sizing

	1x.demo	1x.pico [6.1+ only]	1x.extra-small	1x.small	1x.medium
Data transfer	Demo use only	50GB/day	100GB/day	500GB/day	2TB/day
Queries per second (QPS)	Demo use only	1-25 QPS at 200ms	1-25 QPS at 200ms	25-50 QPS at 200ms	25-75 QPS at 200ms
Replication factor	None	2	2	2	2
Total CPU requests	None	7 vCPUs	14 vCPUs	34 vCPUs	54 vCPUs
Total CPU requests if using the ruler	None	8 vCPUs	16 vCPUs	42 vCPUs	70 vCPUs

	1x.demo	1x.pico [6.1+ only]	1x.extra-small	1x.small	1x.medium
Total memory requests	None	17Gi	31Gi	67Gi	139Gi
Total memory requests if using the ruler	None	18Gi	35Gi	83Gi	171Gi
Total disk requests	40Gi	590Gi	430Gi	430Gi	590Gi
Total disk requests if using the ruler	60Gi	910Gi	750Gi	750Gi	910Gi

1.6.2. Prerequisites

- You have installed the Loki Operator by using the command-line interface (CLI) or web console.
- You have created a **serviceAccount** CR in the same namespace as the **ClusterLogForwarder** CR.
- You have assigned the **collect-audit-logs**, **collect-application-logs**, and **collect-infrastructure-logs** cluster roles to the **serviceAccount** CR.

1.6.3. Core set up and configuration

Use role-based access controls, basic monitoring, and pod placement to deploy Loki.

1.6.4. Authorizing LokiStack rules RBAC permissions

Administrators can allow users to create and manage their own alerting and recording rules by binding cluster roles to usernames. Cluster roles are defined as **ClusterRole** objects that contain necessary role-based access control (RBAC) permissions for users.

The following cluster roles for alerting and recording rules are available for LokiStack:

Rule name	Description
alertingrules.loki.grafana.com-v1-admin	Users with this role have administrative-level access to manage alerting rules. This cluster role grants permissions to create, read, update, delete, list, and watch **AlertingRule** resources within the **loki.grafana.com/v1** API group.

Rule name	Description
alertingrules.loki.grafana.com-v1-crdview	Users with this role can view the definitions of Custom Resource Definitions (CRDs) related to **AlertingRule** resources within the **loki.grafana.com/v1** API group, but do not have permissions for modifying or managing these resources.
alertingrules.loki.grafana.com-v1-edit	Users with this role have permission to create, update, and delete **AlertingRule** resources.
alertingrules.loki.grafana.com-v1-view	Users with this role can read **AlertingRule** resources within the **loki.grafana.com/v1** API group. They can inspect configurations, labels, and annotations for existing alerting rules but cannot make any modifications to them.
recordingrules.loki.grafana.com-v1-admin	Users with this role have administrative-level access to manage recording rules. This cluster role grants permissions to create, read, update, delete, list, and watch **RecordingRule** resources within the **loki.grafana.com/v1** API group.
recordingrules.loki.grafana.com-v1-crdview	Users with this role can view the definitions of Custom Resource Definitions (CRDs) related to **RecordingRule** resources within the **loki.grafana.com/v1** API group, but do not have permissions for modifying or managing these resources.
recordingrules.loki.grafana.com-v1-edit	Users with this role have permission to create, update, and delete **RecordingRule** resources.
recordingrules.loki.grafana.com-v1-view	Users with this role can read **RecordingRule** resources within the **loki.grafana.com/v1** API group. They can inspect configurations, labels, and annotations for existing alerting rules but cannot make any modifications to them.

1.6.4.1. Examples

To apply cluster roles for a user, you must bind an existing cluster role to a specific username.

Cluster roles can be cluster or namespace scoped, depending on which type of role binding you use. When a **RoleBinding** object is used, as when using the **oc adm policy add-role-to-user** command, the cluster role only applies to the specified namespace. When a **ClusterRoleBinding** object is used, as when using the **oc adm policy add-cluster-role-to-user** command, the cluster role applies to all namespaces in the cluster.

The following example command gives the specified user create, read, update and delete (CRUD) permissions for alerting rules in a specific namespace in the cluster:

Example cluster role binding command for alerting rule CRUD permissions in a specific namespace

```
$ oc adm policy add-role-to-user alertingrules.loki.grafana.com-v1-admin -n <namespace>
<username>
```

The following command gives the specified user administrator permissions for alerting rules in all namespaces:

Example cluster role binding command for administrator permissions

```
$ oc adm policy add-cluster-role-to-user alertingrules.loki.grafana.com-v1-admin <username>
```

1.6.5. Creating a log-based alerting rule with Loki

The **AlertingRule** CR contains a set of specifications and webhook validation definitions to declare groups of alerting rules for a single **LokiStack** instance. In addition, the webhook validation definition provides support for rule validation conditions:

- If an **AlertingRule** CR includes an invalid **interval** period, it is an invalid alerting rule
- If an **AlertingRule** CR includes an invalid **for** period, it is an invalid alerting rule.
- If an **AlertingRule** CR includes an invalid LogQL **expr**, it is an invalid alerting rule.
- If an **AlertingRule** CR includes two groups with the same name, it is an invalid alerting rule.
- If none of the above applies, an alerting rule is considered valid.

Table 1.3. AlertingRule definitions

Tenant type	Valid namespaces for **AlertingRule** CRs
application	**<your_application_namespace>**
audit	**openshift-logging**
infrastructure	**openshift-**/*, **kube-**/*, **default**

Procedure

1. Create an **AlertingRule** custom resource (CR):

   Example infrastructure AlertingRule CR

   ```
   apiVersion: loki.grafana.com/v1
   kind: AlertingRule
   metadata:
     name: loki-operator-alerts
   ```

```
    namespace: openshift-operators-redhat (1)
    labels: (2)
      openshift.io/<label_name>: "true"
  spec:
    tenantID: "infrastructure" (3)
    groups:
      - name: LokiOperatorHighReconciliationError
        rules:
          - alert: HighPercentageError
            expr: | (4)
              sum(rate({kubernetes_namespace_name="openshift-operators-redhat",
kubernetes_pod_name=~"loki-operator-controller-manager.*"} |= "error" [1m])) by (job)
                /
              sum(rate({kubernetes_namespace_name="openshift-operators-redhat",
kubernetes_pod_name=~"loki-operator-controller-manager.*"}[1m])) by (job)
                > 0.01
            for: 10s
            labels:
              severity: critical (5)
            annotations:
              summary: High Loki Operator Reconciliation Errors (6)
              description: High Loki Operator Reconciliation Errors (7)
```

1. The namespace where this **AlertingRule** CR is created must have a label matching the LokiStack **spec.rules.namespaceSelector** definition.
2. The **labels** block must match the LokiStack **spec.rules.selector** definition.
3. **AlertingRule** CRs for **infrastructure** tenants are only supported in the **openshift-***, **kube-***, or **default** namespaces.
4. The value for **kubernetes_namespace_name:** must match the value for **metadata.namespace**.
5. The value of this mandatory field must be **critical**, **warning**, or **info**.
6. This field is mandatory.
7. This field is mandatory.

Example application AlertingRule CR

```
  apiVersion: loki.grafana.com/v1
  kind: AlertingRule
  metadata:
    name: app-user-workload
    namespace: app-ns (1)
    labels: (2)
      openshift.io/<label_name>: "true"
  spec:
    tenantID: "application"
    groups:
      - name: AppUserWorkloadHighError
        rules:
```

```
        - alert:
          expr: | 3
            sum(rate({kubernetes_namespace_name="app-ns",
kubernetes_pod_name=~"podName.*"} |= "error" [1m])) by (job)
          for: 10s
          labels:
            severity: critical 4
          annotations:
            summary: 5
            description: 6
```

1. The namespace where this **AlertingRule** CR is created must have a label matching the LokiStack **spec.rules.namespaceSelector** definition.
2. The **labels** block must match the LokiStack **spec.rules.selector** definition.
3. Value for **kubernetes_namespace_name:** must match the value for **metadata.namespace**.
4. The value of this mandatory field must be **critical**, **warning**, or **info**.
5. The value of this mandatory field is a summary of the rule.
6. The value of this mandatory field is a detailed description of the rule.

2. Apply the **AlertingRule** CR:

```
$ oc apply -f <filename>.yaml
```

1.6.6. Configuring Loki to tolerate memberlist creation failure

In an OpenShift Container Platform cluster, administrators generally use a non-private IP network range. As a result, the LokiStack memberlist configuration fails because, by default, it only uses private IP networks.

As an administrator, you can select the pod network for the memberlist configuration. You can modify the **LokiStack** custom resource (CR) to use the **podIP** address in the **hashRing** spec. To configure the **LokiStack** CR, use the following command:

```
$ oc patch LokiStack logging-loki -n openshift-logging  --type=merge -p '{"spec": {"hashRing":
{"memberlist":{"instanceAddrType":"podIP"},"type":"memberlist"}}}'
```

Example LokiStack to include podIP

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  hashRing:
    type: memberlist
```

```
    memberlist:
      instanceAddrType: podIP
# ...
```

1.6.7. Enabling stream-based retention with Loki

You can configure retention policies based on log streams. Rules for these may be set globally, per-tenant, or both. If you configure both, tenant rules apply before global rules.

IMPORTANT

If there is no retention period defined on the s3 bucket or in the LokiStack custom resource (CR), then the logs are not pruned and they stay in the s3 bucket forever, which might fill up the s3 storage.

NOTE

Schema v13 is recommended.

Procedure

1. Create a **LokiStack** CR:
   - Enable stream-based retention globally as shown in the following example:

     Example global stream-based retention for AWS

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
   global: (1)
     retention: (2)
       days: 20
       streams:
       - days: 4
         priority: 1
         selector: '{kubernetes_namespace_name=~"test.+"}' (3)
       - days: 1
         priority: 1
         selector: '{log_type="infrastructure"}'
  managementState: Managed
  replicationFactor: 1
  size: 1x.small
  storage:
    schemas:
    - effectiveDate: "2020-10-11"
      version: v13
    secret:
      name: logging-loki-s3
      type: aws
```

```
  storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```

1. Sets retention policy for all log streams. **Note: This field does not impact the retention period for stored logs in object storage.**
2. Retention is enabled in the cluster when this block is added to the CR.
3. Contains the LogQL query used to define the log stream.spec: limits:

- Enable stream-based retention per-tenant basis as shown in the following example:

Example per-tenant stream-based retention for AWS

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
    global:
      retention:
        days: 20
    tenants: 1
      application:
        retention:
          days: 1
          streams:
            - days: 4
              selector: '{kubernetes_namespace_name=~"test.+"}' 2
      infrastructure:
        retention:
          days: 5
          streams:
            - days: 1
              selector: '{kubernetes_namespace_name=~"openshift-cluster.+"}'
  managementState: Managed
  replicationFactor: 1
  size: 1x.small
  storage:
    schemas:
    - effectiveDate: "2020-10-11"
      version: v13
    secret:
      name: logging-loki-s3
      type: aws
  storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```

1. Sets retention policy by tenant. Valid tenant types are **application**, **audit**, and **infrastructure**.

2 Contains the LogQL query used to define the log stream.

2. Apply the **LokiStack** CR:

```
$ oc apply -f <filename>.yaml
```

NOTE

This is not for managing the retention for stored logs. Global retention periods for stored logs to a supported maximum of 30 days is configured with your object storage.

1.6.8. Loki pod placement

You can control which nodes the Loki pods run on, and prevent other workloads from using those nodes, by using tolerations or node selectors on the pods.

You can apply tolerations to the log store pods with the LokiStack custom resource (CR) and apply taints to a node with the node specification. A taint on a node is a **key:value** pair that instructs the node to repel all pods that do not allow the taint. Using a specific **key:value** pair that is not on other pods ensures that only the log store pods can run on that node.

Example LokiStack with node selectors

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  template:
    compactor: 1
      nodeSelector:
        node-role.kubernetes.io/infra: "" 2
    distributor:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    gateway:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    indexGateway:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    ingester:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    querier:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    queryFrontend:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    ruler:
```

```
      nodeSelector:
        node-role.kubernetes.io/infra: ""
# ...
```

1. Specifies the component pod type that applies to the node selector.
2. Specifies the pods that are moved to nodes containing the defined label.

Example LokiStack CR with node selectors and tolerations

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  template:
    compactor:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    distributor:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    indexGateway:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    ingester:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
```

```
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    querier:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    queryFrontend:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    ruler:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    gateway:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
  # ...
```

To configure the **nodeSelector** and **tolerations** fields of the LokiStack (CR), you can use the **oc explain** command to view the description and fields for a particular resource:

```
$ oc explain lokistack.spec.template
```

Example output

```
KIND:     LokiStack
VERSION:  loki.grafana.com/v1
```

```
RESOURCE: template <Object>

DESCRIPTION:
    Template defines the resource/limits/tolerations/nodeselectors per
    component

FIELDS:
  compactor <Object>
    Compactor defines the compaction component spec.

  distributor <Object>
    Distributor defines the distributor component spec.
...
```

For more detailed information, you can add a specific field:

```
$ oc explain lokistack.spec.template.compactor
```

Example output

```
KIND:     LokiStack
VERSION:  loki.grafana.com/v1

RESOURCE: compactor <Object>

DESCRIPTION:
    Compactor defines the compaction component spec.

FIELDS:
  nodeSelector <map[string]string>
    NodeSelector defines the labels required by a node to schedule the
    component onto it.
...
```

1.6.9. Enhanced reliability and performance

Use the following configurations to ensure reliability and efficiency of Loki in production.

1.6.10. Enabling authentication to cloud-based log stores using short-lived tokens

Workload identity federation enables authentication to cloud-based log stores using short-lived tokens.

Procedure

- Use one of the following options to enable authentication:
    - If you use the OpenShift Container Platform web console to install the Loki Operator, clusters that use short-lived tokens are automatically detected. You are prompted to create roles and supply the data required for the Loki Operator to create a **CredentialsRequest** object, which populates a secret.
    - If you use the OpenShift CLI (**oc**) to install the Loki Operator, you must manually create a **Subscription** object using the appropriate template for your storage provider, as shown in the following examples. This authentication strategy is only supported for the storage

providers indicated.

Example Azure sample subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat
spec:
  channel: "stable-6.0"
  installPlanApproval: Manual
  name: loki-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
      - name: CLIENTID
        value: <your_client_id>
      - name: TENANTID
        value: <your_tenant_id>
      - name: SUBSCRIPTIONID
        value: <your_subscription_id>
      - name: REGION
        value: <your_region>
```

Example AWS sample subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat
spec:
  channel: "stable-6.0"
  installPlanApproval: Manual
  name: loki-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: ROLEARN
      value: <role_ARN>
```

1.6.11. Configuring Loki to tolerate node failure

The Loki Operator supports setting pod anti-affinity rules to request that pods of the same component are scheduled on different available nodes in the cluster.

Affinity is a property of pods that controls the nodes on which they prefer to be scheduled. Anti-affinity is a property of pods that prevents a pod from being scheduled on a node.

In OpenShift Container Platform, *pod affinity* and *pod anti-affinity* allow you to constrain which nodes your pod is eligible to be scheduled on based on the key-value labels on other pods.

The Operator sets default, preferred **podAntiAffinity** rules for all Loki components, which includes the **compactor**, **distributor**, **gateway**, **indexGateway**, **ingester**, **querier**, **queryFrontend**, and **ruler** components.

You can override the preferred **podAntiAffinity** settings for Loki components by configuring required settings in the **requiredDuringSchedulingIgnoredDuringExecution** field:

Example user settings for the ingester component

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  template:
    ingester:
      podAntiAffinity:
      # ...
        requiredDuringSchedulingIgnoredDuringExecution: 1
        - labelSelector:
            matchLabels: 2
              app.kubernetes.io/component: ingester
          topologyKey: kubernetes.io/hostname
# ...
```

1. The stanza to define a required rule.
2. The key-value pair (label) that must be matched to apply the rule.

1.6.12. LokiStack behavior during cluster restarts

When an OpenShift Container Platform cluster is restarted, LokiStack ingestion and the query path continue to operate within the available CPU and memory resources available for the node. This means that there is no downtime for the LokiStack during OpenShift Container Platform cluster updates. This behavior is achieved by using **PodDisruptionBudget** resources. The Loki Operator provisions **PodDisruptionBudget** resources for Loki, which determine the minimum number of pods that must be available per component to ensure normal operations under certain conditions.

1.6.13. Advanced deployment and scalability

To configure high availability, scalability, and error handling, use the following information.

1.6.14. Zone aware data replication

The Loki Operator offers support for zone-aware data replication through pod topology spread constraints. Enabling this feature enhances reliability and safeguards against log loss in the event of a single zone failure. When configuring the deployment size as **1x.extra-small**, **1x.small**, or **1x.medium**, the **replication.factor** field is automatically set to 2.

To ensure proper replication, you need to have at least as many availability zones as the replication factor specifies. While it is possible to have more availability zones than the replication factor, having

fewer zones can lead to write failures. Each zone should host an equal number of instances for optimal operation.

Example LokiStack CR with zone replication enabled

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
 name: logging-loki
 namespace: openshift-logging
spec:
 replicationFactor: 2 (1)
 replication:
  factor: 2 (2)
  zones:
  - maxSkew: 1 (3)
    topologyKey: topology.kubernetes.io/zone (4)
```

1. Deprecated field, values entered are overwritten by **replication.factor**.
2. This value is automatically set when deployment size is selected at setup.
3. The maximum difference in number of pods between any two topology domains. The default is 1, and you cannot specify a value of 0.
4. Defines zones in the form of a topology key that corresponds to a node label.

1.6.15. Recovering Loki pods from failed zones

In OpenShift Container Platform a zone failure happens when specific availability zone resources become inaccessible. Availability zones are isolated areas within a cloud provider's data center, aimed at enhancing redundancy and fault tolerance. If your OpenShift Container Platform cluster is not configured to handle this, a zone failure can lead to service or data loss.

Loki pods are part of a StatefulSet, and they come with Persistent Volume Claims (PVCs) provisioned by a **StorageClass** object. Each Loki pod and its PVCs reside in the same zone. When a zone failure occurs in a cluster, the StatefulSet controller automatically attempts to recover the affected pods in the failed zone.

WARNING

The following procedure will delete the PVCs in the failed zone, and all data contained therein. To avoid complete data loss the replication factor field of the **LokiStack** CR should always be set to a value greater than 1 to ensure that Loki is replicating.

Prerequisites

- Verify your **LokiStack** CR has a replication factor greater than 1.

- Zone failure detected by the control plane, and nodes in the failed zone are marked by cloud provider integration.

The StatefulSet controller automatically attempts to reschedule pods in a failed zone. Because the associated PVCs are also in the failed zone, automatic rescheduling to a different zone does not work. You must manually delete the PVCs in the failed zone to allow successful re-creation of the stateful Loki Pod and its provisioned PVC in the new zone.

Procedure

1. List the pods in **Pending** status by running the following command:

```
$ oc get pods --field-selector status.phase==Pending -n openshift-logging
```

Example oc get pods output

```
NAME                          READY   STATUS    RESTARTS   AGE (1)
logging-loki-index-gateway-1  0/1     Pending   0          17m
logging-loki-ingester-1       0/1     Pending   0          16m
logging-loki-ruler-1          0/1     Pending   0          16m
```

(1) These pods are in **Pending** status because their corresponding PVCs are in the failed zone.

2. List the PVCs in **Pending** status by running the following command:

```
$ oc get pvc -o=json -n openshift-logging | jq '.items[] | select(.status.phase == "Pending") | .metadata.name' -r
```

Example oc get pvc output

```
storage-logging-loki-index-gateway-1
storage-logging-loki-ingester-1
wal-logging-loki-ingester-1
storage-logging-loki-ruler-1
wal-logging-loki-ruler-1
```

3. Delete the PVC(s) for a pod by running the following command:

```
$ oc delete pvc <pvc_name> -n openshift-logging
```

4. Delete the pod(s) by running the following command:

```
$ oc delete pod <pod_name> -n openshift-logging
```

Once these objects have been successfully deleted, they should automatically be rescheduled in an available zone.

1.6.15.1. Troubleshooting PVC in a terminating state

The PVCs might hang in the terminating state without being deleted, if PVC metadata finalizers are set to **kubernetes.io/pv-protection**. Removing the finalizers should allow the PVCs to delete successfully.

- Remove the finalizer for each PVC by running the command below, then retry deletion.

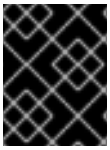
```
$ oc patch pvc <pvc_name> -p '{"metadata":{"finalizers":null}}' -n openshift-logging
```

1.6.16. Troubleshooting Loki rate limit errors

If the Log Forwarder API forwards a large block of messages that exceeds the rate limit to Loki, Loki generates rate limit (**429**) errors.

These errors can occur during normal operation. For example, when adding the logging to a cluster that already has some logs, rate limit errors might occur while the logging tries to ingest all of the existing log entries. In this case, if the rate of addition of new logs is less than the total rate limit, the historical data is eventually ingested, and the rate limit errors are resolved without requiring user intervention.

In cases where the rate limit errors continue to occur, you can fix the issue by modifying the **LokiStack** custom resource (CR).



IMPORTANT

The **LokiStack** CR is not available on Grafana-hosted Loki. This topic does not apply to Grafana-hosted Loki servers.

Conditions

- The Log Forwarder API is configured to forward logs to Loki.
- Your system sends a block of messages that is larger than 2 MB to Loki. For example:

```
"values":[["1630410392689800468","{\"kind\":\"Event\",\"apiVersion\":\
.......
......
......
......
\"received_at\":\"2021-08-31T11:46:32.800278+00:00\",\"version\":\"1.7.4
1.6.0\"}},\"@timestamp\":\"2021-08-
31T11:46:32.799692+00:00\",\"viaq_index_name\":\"audit-
write\",\"viaq_msg_id\":\"MzFjYjJkZjItNjY0MC00YWU4LWIwMTEtNGNmM2E5ZmViMGU4\",\"lo
g_type\":\"audit\"}"]]}]}
```

- After you enter **oc logs -n openshift-logging -l component=collector**, the collector logs in your cluster show a line containing one of the following error messages:

```
429 Too Many Requests Ingestion rate limit exceeded
```

Example Vector error message

```
2023-08-25T16:08:49.301780Z  WARN sink{component_kind="sink"
component_id=default_loki_infra component_type=loki component_name=default_loki_infra}:
vector::sinks::util::retries: Retrying after error. error=Server responded with an error: 429 Too
Many Requests internal_log_rate_limit=true
```

The error is also visible on the receiving end. For example, in the LokiStack ingester pod:

Example Loki ingester error message

```
level=warn ts=2023-08-30T14:57:34.155592243Z caller=grpc_logging.go:43
duration=1.434942ms method=/logproto.Pusher/Push err="rpc error: code = Code(429) desc
= entry with timestamp 2023-08-30 14:57:32.012778399 +0000 UTC ignored, reason: 'Per
stream rate limit exceeded (limit: 3MB/sec) while attempting to ingest for stream
```

Procedure

- Update the **ingestionBurstSize** and **ingestionRate** fields in the **LokiStack** CR:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
    global:
      ingestion:
        ingestionBurstSize: 16 (1)
        ingestionRate: 8 (2)
# ...
```

(1) The **ingestionBurstSize** field defines the maximum local rate-limited sample size per distributor replica in MB. This value is a hard limit. Set this value to at least the maximum logs size expected in a single push request. Single requests that are larger than the **ingestionBurstSize** value are not permitted.

(2) The **ingestionRate** field is a soft limit on the maximum amount of ingested samples per second in MB. Rate limit errors occur if the rate of logs exceeds the limit, but the collector retries sending the logs. As long as the total average is lower than the limit, the system recovers and errors are resolved without user intervention.

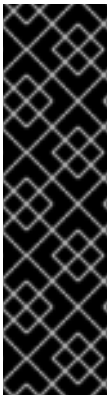
1.7. OTLP DATA INGESTION IN LOKI

You can use an API endpoint by using the OpenTelemetry Protocol (OTLP) with Logging. As OTLP is a standardized format not specifically designed for Loki, OTLP requires an additional Loki configuration to map data format of OpenTelemetry to data model of Loki. OTLP lacks concepts such as *stream labels* or *structured metadata*. Instead, OTLP provides metadata about log entries as **attributes**, grouped into the following three categories:

- Resource
- Scope
- Log

You can set metadata for multiple entries simultaneously or individually as needed.

1.7.1. Configuring LokiStack for OTLP data ingestion



IMPORTANT

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

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

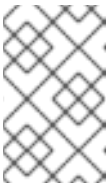
To configure a **LokiStack** custom resource (CR) for OTLP ingestion, follow these steps:

Prerequisites

- Ensure that your Loki setup supports structured metadata, introduced in schema version 13 to enable OTLP log ingestion.

Procedure

1. Set the schema version:
   - When creating a new **LokiStack** CR, set **version: v13** in the storage schema configuration.



NOTE

For existing configurations, add a new schema entry with **version: v13** and an **effectiveDate** in the future. For more information on updating schema versions, see Upgrading Schemas (Grafana documentation).

2. Configure the storage schema as follows:

Example configure storage schema

```
# ...
spec:
  storage:
    schemas:
    - version: v13
      effectiveDate: 2024-10-25
```

Once the **effectiveDate** has passed, the v13 schema takes effect, enabling your **LokiStack** to store structured metadata.

1.7.2. Attribute mapping

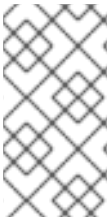
When you set the Loki Operator to the **openshift-logging** mode, Loki Operator automatically applies a default set of attribute mappings. These mappings align specific OTLP attributes with stream labels and structured metadata of Loki.

For typical setups, these default mappings are sufficient. However, you might need to customize attribute mapping in the following cases:

- Using a custom collector: If your setup includes a custom collector that generates additional attributes that you do not want to store, consider customizing the mapping to ensure these attributes are dropped by Loki.
- Adjusting attribute detail levels: If the default attribute set is more detailed than necessary, you can reduce it to essential attributes only. This can avoid excessive data storage and streamline the logging process.

1.7.2.1. Custom attribute mapping for OpenShift

When using the Loki Operator in **openshift-logging** mode, attribute mapping follow OpenShift default values, but you can configure custom mappings to adjust default values. In the **openshift-logging** mode, you can configure custom attribute mappings globally for all tenants or for individual tenants as needed. When you define custom mappings, they are appended to the OpenShift default values. If you do not need default labels, you can disable them in the tenant configuration.



NOTE

A major difference between the Loki Operator and Loki lies in inheritance handling. Loki copies only **default_resource_attributes_as_index_labels** to tenants by default, while the Loki Operator applies the entire global configuration to each tenant in the **openshift-logging** mode.

Within **LokiStack**, attribute mapping configuration is managed through the **limits** setting. See the following example **LokiStack** configuration:

```
# ...
spec:
  limits:
    global:
      otlp: {} (1)
    tenants:
      application: (2)
        otlp: {}
```

1. Defines global OTLP attribute configuration.
2. Defines the OTLP attribute configuration for the **application** tenant within the **openshift-logging** mode. You can also configure **infrastructure** and **audit** tenants in addition to **application** tenants.

NOTE

You can use both global and per-tenant OTLP configurations for mapping attributes to stream labels.

Stream labels derive only from resource-level attributes, which the **LokiStack** resource structure reflects. See the following **LokiStack** example configuration:

```
spec:
  limits:
    global:
      otlp:
        streamLabels:
```

```
        resourceAttributes:
        - name: "k8s.namespace.name"
        - name: "k8s.pod.name"
        - name: "k8s.container.name"
```

You can drop attributes of type resource, scope, or log from the log entry.

```
# ...
spec:
  limits:
    global:
      otlp:
        streamLabels:
# ...
        drop:
          resourceAttributes:
          - name: "process.command_line"
          - name: "k8s\\.pod\\.labels\\..+"
            regex: true
          scopeAttributes:
          - name: "service.name"
          logAttributes:
          - name: "http.route"
```

You can use regular expressions by setting **regex: true** to apply a configuration for attributes with similar names.

IMPORTANT

Avoid using regular expressions for stream labels, as this can increase data volume.

Attributes that are not explicitly set as stream labels or dropped from the entry are saved as structured metadata by default.

1.7.2.2. Customizing OpenShift defaults

In the **openshift-logging** mode, certain attributes are required and cannot be removed from the configuration due to their role in OpenShift functions. Other attributes, labeled **recommended**, might be dropped if performance is impacted. For information about the attributes, see OpenTelemetry data model attributes.

When using the **openshift-logging** mode without custom attributes, you can achieve immediate compatibility with OpenShift tools. If additional attributes are needed as stream labels or some attributes need to be droped, use custom configuration. Custom configurations can merge with default configurations.

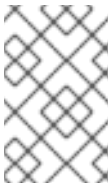
1.7.2.3. Removing recommended attributes

To reduce default attributes in the **openshift-logging** mode, disable recommended attributes:

```
# ...
spec:
  tenants:
    mode: openshift-logging
```

```
openshift:
  otlp:
    disableRecommendedAttributes: true (1)
```

(1) Set **disableRecommendedAttributes: true** to remove recommended attributes, which limits default attributes to the required attributes or stream labels.



NOTE

This setting might negatively impact query performance, as it removes default stream labels. You must pair this option with a custom attribute configuration to retain attributes essential for queries.

1.7.3. Additional resources

- Loki labels (Grafana documentation)
- Structured metadata (Grafana documentation)
- OpenTelemetry data model
- OpenTelemetry attribute (OpenTelemetry documentation)

1.8. VISUALIZATION FOR LOGGING

Visualization for logging is provided by deploying the Logging UI Plugin of the Cluster Observability Operator, which requires Operator installation.

CHAPTER 2. LOGGING 6.1

2.1. SUPPORT

Only the configuration options described in this documentation are supported for logging.

Do not use any other configuration options, as they are unsupported. Configuration paradigms might change across OpenShift Container Platform releases, and such cases can only be handled gracefully if all configuration possibilities are controlled. If you use configurations other than those described in this documentation, your changes will be overwritten, because Operators are designed to reconcile any differences.

NOTE

If you must perform configurations not described in the OpenShift Container Platform documentation, you must set your Red Hat OpenShift Logging Operator to **Unmanaged**. An unmanaged logging instance is not supported and does not receive updates until you return its status to **Managed**.

NOTE

Logging is provided as an installable component, with a distinct release cycle from the core OpenShift Container Platform. The Red Hat OpenShift Container Platform Life Cycle Policy outlines release compatibility.

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system offered as a GA log store for logging for Red Hat OpenShift that can be visualized with the OpenShift Observability UI. The Loki configuration provided by OpenShift Logging is a short-term log store designed to enable users to perform fast troubleshooting with the collected logs. For that purpose, the logging for Red Hat OpenShift configuration of Loki has short-term storage, and is optimized for very recent queries. For long-term storage or queries over a long time period, users should look to log stores external to their cluster.

Logging for Red Hat OpenShift is an opinionated collector and normalizer of application, infrastructure, and audit logs. It is intended to be used for forwarding logs to various supported systems.

Logging is not:

- A high scale log collection system
- Security Information and Event Monitoring (SIEM) compliant
- A "bring your own" (BYO) log collector configuration
- Historical or long term log retention or storage
- A guaranteed log sink
- Secure storage - audit logs are not stored by default

2.1.1. Supported API custom resource definitions

The following table describes the supported Logging APIs.

Table 2.1. Logging API support states

CustomResourceDefinition (CRD)	ApiVersion	Support state
LokiStack	lokistack.loki.grafana.com/v1	Supported from 5.5
RulerConfig	rulerconfig.loki.grafana/v1	Supported from 5.7
AlertingRule	alertingrule.loki.grafana/v1	Supported from 5.7
RecordingRule	recordingrule.loki.grafana/v1	Supported from 5.7
LogFileMetricExporter	LogFileMetricExporter.logging.openshift.io/v1alpha1	Supported from 5.8
ClusterLogForwarder	clusterlogforwarder.observability.openshift.io/v1	Supported from 6.0

2.1.2. Unsupported configurations

You must set the Red Hat OpenShift Logging Operator to the **Unmanaged** state to modify the following components:

- The collector configuration file
- The collector daemonset

Explicitly unsupported cases include:

- **Configuring the logging collector using environment variables** You cannot use environment variables to modify the log collector.
- **Configuring how the log collector normalizes logs**You cannot modify default log normalization.

2.1.3. Support policy for unmanaged Operators

The *management state* of an Operator determines whether an Operator is actively managing the resources for its related component in the cluster as designed. If an Operator is set to an *unmanaged* state, it does not respond to changes in configuration nor does it receive updates.

While this can be helpful in non-production clusters or during debugging, Operators in an unmanaged state are unsupported and the cluster administrator assumes full control of the individual component configurations and upgrades.

An Operator can be set to an unmanaged state using the following methods:

- **Individual Operator configuration**
  Individual Operators have a **managementState** parameter in their configuration. This can be accessed in different ways, depending on the Operator. For example, the Red Hat OpenShift Logging Operator accomplishes this by modifying a custom resource (CR) that it manages, while the Cluster Samples Operator uses a cluster-wide configuration resource.

Changing the **managementState** parameter to **Unmanaged** means that the Operator is not actively managing its resources and will take no action related to the related component. Some Operators might not support this management state as it might damage the cluster and require manual recovery.

WARNING

Changing individual Operators to the **Unmanaged** state renders that particular component and functionality unsupported. Reported issues must be reproduced in **Managed** state for support to proceed.

- **Cluster Version Operator (CVO) overrides**
  The **spec.overrides** parameter can be added to the CVO's configuration to allow administrators to provide a list of overrides to the CVO's behavior for a component. Setting the **spec.overrides[].unmanaged** parameter to **true** for a component blocks cluster upgrades and alerts the administrator after a CVO override has been set:

  > Disabling ownership via cluster version overrides prevents upgrades. Please remove overrides before continuing.

WARNING

Setting a CVO override puts the entire cluster in an unsupported state. Reported issues must be reproduced after removing any overrides for support to proceed.

2.1.4. Collecting logging data for Red Hat Support

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

You can use the must-gather tool to collect diagnostic information for project-level resources, cluster-level resources, and each of the logging components. For prompt support, supply diagnostic information for both OpenShift Container Platform and logging.

2.1.4.1. About the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues.

For your logging, **must-gather** collects the following information:

- Project-level resources, including pods, configuration maps, service accounts, roles, role bindings, and events at the project level

- Cluster-level resources, including nodes, roles, and role bindings at the cluster level
- OpenShift Logging resources in the **openshift-logging** and **openshift-operators-redhat** namespaces, including health status for the log collector, the log store, and the log visualizer

When you run **oc adm must-gather**, a new pod is created on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in the current working directory.

2.1.4.2. Collecting logging data

You can use the **oc adm must-gather** CLI command to collect information about logging.

Procedure

To collect logging information with **must-gather**:

1. Navigate to the directory where you want to store the **must-gather** information.
2. Run the **oc adm must-gather** command against the logging image:

```
$ oc adm must-gather --image=$(oc -n openshift-logging get deployment.apps/cluster-logging-operator -o jsonpath='{.spec.template.spec.containers[?(@.name == "cluster-logging-operator")].image}')
```

   The **must-gather** tool creates a new directory that starts with **must-gather.local** within the current directory. For example: **must-gather.local.4157245944708210408**.

3. Create a compressed file from the **must-gather** directory that was just created. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar -cvaf must-gather.tar.gz must-gather.local.4157245944708210408
```

4. Attach the compressed file to your support case on the Red Hat Customer Portal .

2.2. LOGGING 6.1 RELEASE NOTES

2.2.1. Logging 6.1.6 Release Notes

This release includes RHBA-2025:4529.

2.2.1.1. Bug fixes

- Before this update, collector pods would enter a crash loop due to a configuration error when attempting token-based authentication with an Elasticsearch output. With this update, token authentication with an Elasticsearch output generates a valid configuration. (LOG-7018)
- Before this update, **auditd** log messages with multiple **msg** keys could cause errors in collector pods, because the standard **auditd** log format expects a single **msg** field per log entry that follows the **msg=audit(TIMESTAMP:ID)** structure. With this update, only the first **msg** value is used, which resolves the issue and ensures accurate extraction of audit metadata. (LOG-7029)

2.2.1.2. CVEs

- CVE-2022-49043
- CVE-2024-2236
- CVE-2024-5535
- CVE-2024-56171
- CVE-2025-24928

NOTE

For detailed information on Red Hat security ratings, review Severity ratings.

2.2.2. Logging 6.1.5 Release Notes

This release includes RHSA-2025:3907.

2.2.2.1. New features and enhancements

- Before this update, time-based stream sharding was not enabled in Loki, which resulted in Loki being unable to save historical data. With this update, Loki Operator enables time-based stream sharding in Loki, which helps Loki save historical data. (LOG-6991)

2.2.2.2. Bug fixes

- Before this update, the Vector collector could not forward Open Virtual Network (OVN) and Auditd logs. With this update, the Vector collector can forward OVN and Auditd logs. (LOG-6996)

2.2.2.3. CVEs

- CVE-2025-30204

NOTE

For detailed information on Red Hat security ratings, review Severity ratings.

2.2.3. Logging 6.1.4 Release Notes

This release includes Logging for Red Hat OpenShift Bug Fix Release 6.1.4 .

2.2.3.1. Bug fixes

- Before this update, Red Hat Managed Elasticsearch failed to receive logs if the index name did not follow the required patterns (**app-**, **infra-**, **audit-**), resulting in an **index_not_found_exception** error due to a restricted automatic index creation. With this update, improved documentation and explanations in the **oc explain obsclf.spec.outputs.elasticsearch.index** command clarify the index naming limitations, helping users configure log forwarding correctly. (LOG-6623)

- Before this update, when you used **1x.pico** as the LokiStack size, the number of delete workers was set to zero. This issue occurred because of an error in the Operator that generates the Loki configuration. With this update, the number of delete workers is set to ten. (LOG-6797)

- Before this update, the Operator failed to update the **securitycontextconstraint** object required by the log collector, which was a regression from previous releases. With this update, the Operator restores the cluster role to the service account and updates the resource. (LOG-6816)

2.2.3.2. CVEs

- CVE-2022-49043
- CVE-2024-45336
- CVE-2024-45338
- CVE-2024-56171
- CVE-2025-24928
- CVE-2025-27144

NOTE

For detailed information on Red Hat security ratings, review Severity ratings.

2.2.4. Logging 6.1.3 Release Notes

This release includes Logging for Red Hat OpenShift Bug Fix Release 6.1.3 .

2.2.4.1. Bug Fixes

- Before this update, when using the new **1x.pico** size with the Loki Operator, the **PodDisruptionBudget** created for the Ingester pod allowed Kubernetes to evict two of the three Ingester pods. With this update, the Operator now creates a **PodDisruptionBudget** that allows eviction of only a single Ingester pod. (LOG-6693)
- Before this update, the Operator did not support templating of **syslog facility** and **severity level**, which was consistent with the rest of the API. Instead, the Operator relied upon the **5.x** API, which is no longer supported. With this update, the Operator supports templating by adding the required validation to the API and rejecting resources that do not match the required format. (LOG-6788)
- Before this update, empty **OTEL** tuning configuration caused a validation error. With this update, the validation rules allow empty **OTEL** tuning configurations. (LOG-6532)

2.2.4.2. CVEs

- CVE-2020-11023
- CVE-2024-9287
- CVE-2024-12797

2.2.5. Logging 6.1.2 Release Notes

This release includes Logging for Red Hat OpenShift Bug Fix Release 6.1.2 .

2.2.5.1. New Features and Enhancements

- This enhancement adds **OTel** semantic stream labels to the **lokiStack** output so that you can query logs by using both **ViaQ** and **OTel** stream labels. (LOG-6579)

2.2.5.2. Bug Fixes

- Before this update, the collector alerting rules contained summary and message fields. With this update, the collector alerting rules contain summary and description fields. (LOG-6126)
- Before this update, the collector metrics dashboard could get removed after an Operator upgrade due to a race condition during the transition from the old to the new pod deployment. With this update, labels are added to the dashboard **ConfigMap** to identify the upgraded deployment as the current owner so that it will not be removed. (LOG-6280)
- Before this update, when you included infrastructure namespaces in application inputs, their **log_type** would be set to **application**. With this update, the **log_type** of infrastructure namespaces included in application inputs is set to **infrastructure**. (LOG-6373)
- Before this update, the Cluster Logging Operator used a cached client to fetch the **SecurityContextConstraint** cluster resource, which could result in an error when the cache is invalid. With this update, the Operator now always retrieves data from the API server instead of using a cache. (LOG-6418)
- Before this update, the logging **must-gather** did not collect resources such as **UIPlugin**, **ClusterLogForwarder**, **LogFileMetricExporter**, and **LokiStack**. With this update, the **must-gather** now collects all of these resources and places them in their respective namespace directory instead of the **cluster-logging** directory. (LOG-6422)
- Before this update, the Vector startup script attempted to delete buffer lock files during startup. With this update, the Vector startup script no longer attempts to delete buffer lock files during startup. (LOG-6506)
- Before this update, the API documentation incorrectly claimed that **lokiStack** outputs would default the target namespace, which could prevent the collector from writing to that output. With this update, this claim has been removed from the API documentation and the Cluster Logging Operator now validates that a target namespace is present. (LOG-6573)
- Before this update, the Cluster Logging Operator could deploy the collector with output configurations that were not referenced by any inputs. With this update, a validation check for the **ClusterLogForwarder** resource prevents the Operator from deploying the collector. (LOG-6585)

2.2.5.3. CVEs

- CVE-2019-12900

2.2.6. Logging 6.1.1 Release Notes

This release includes Logging for Red Hat OpenShift Bug Fix Release 6.1.1 .

2.2.6.1. New Features and Enhancements

- With this update, the Loki Operator supports configuring the workload identity federation on the Google Cloud Platform (GCP) by using the Cluster Credential Operator (CCO) in OpenShift Container Platform 4.17 or later. (LOG-6420)

2.2.6.2. Bug Fixes

- Before this update, the collector was discarding longer audit log messages with the following error message: **Internal log [Found line that exceeds max_line_bytes; discarding.]**With this update, the discarding of longer audit messages is avoided by increasing the audit configuration thresholds: The maximum line size, **max_line_bytes**, is **3145728** bytes. The maximum number of bytes read during a read cycle, **max_read_bytes**, is **262144** bytes. (LOG-6379)

- Before this update, an input receiver service was repeatedly created and deleted, causing issues with mounting the TLS secrets. With this update, the service is created once and only deleted if it is not defined in the **ClusterLogForwarder** custom resource. (LOG-6383)

- Before this update, pipeline validation might have entered an infinite loop if a name was a substring of another name. With this update, stricter name equality checks prevent the infinite loop. (LOG-6405)

- Before this update, the collector alerting rules included the summary and message fields. With this update, the collector alerting rules include the summary and description fields. (LOG-6407)

- Before this update, setting up the custom audit inputs in the **ClusterLogForwarder** custom resource with configured **LokiStack** output caused errors due to the nil pointer dereference. With this update, the Operator performs the nil checks, preventing such errors. (LOG-6449)

- Before this update, the **ValidLokistackOTLPOutputs** condition appeared in the status of the **ClusterLogForwarder** custom resource even when the output type is not **LokiStack**. With this update, the **ValidLokistackOTLPOutputs** condition is removed, and the validation messages for the existing output conditions are corrected. (LOG-6469)

- Before this update, the collector did not correctly mount the **/var/log/oauth-server/** path, which prevented the collection of the audit logs. With this update, the volume mount is added, and the audit logs are collected as expected. (LOG-6484)

- Before this update, the **must-gather** script of the Red Hat OpenShift Logging Operator might have failed to gather the LokiStack data. With this update, the **must-gather** script is fixed, and the LokiStack data is gathered reliably. (LOG-6498)

- Before this update, the collector did not correctly mount the **oauth-apiserver** audit log file. As a result, such audit logs were not collected. With this update, the volume mount is correctly mounted, and the logs are collected as expected. (LOG-6533)

2.2.6.3. CVEs

- CVE-2019-12900
- CVE-2024-2511
- CVE-2024-3596
- CVE-2024-4603
- CVE-2024-4741
- CVE-2024-5535
- CVE-2024-10963
- CVE-2024-50602

2.2.7. Logging 6.1.0 Release Notes

This release includes Logging for Red Hat OpenShift Bug Fix Release 6.1.0 .

2.2.7.1. New Features and Enhancements

2.2.7.1.1. Log Collection

- This enhancement adds the source **iostream** to the attributes sent from collected container logs. The value is set to either **stdout** or **stderr** based on how the collector received it. (LOG-5292)
- With this update, the default memory limit for the collector increases from 1024 Mi to 2048 Mi. Users should adjust resource limits based on their cluster's specific needs and specifications. (LOG-6072)
- With this update, users can now set the syslog output delivery mode of the **ClusterLogForwarder** CR to either **AtLeastOnce** or **AtMostOnce.** (LOG-6355)

2.2.7.1.2. Log Storage

- With this update, the new **1x.pico** LokiStack size supports clusters with fewer workloads and lower log volumes (up to 50GB/day). (LOG-5939)

2.2.7.2. Technology Preview

IMPORTANT

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

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

- With this update, OpenTelemetry logs can now be forwarded using the **OTel** (OpenTelemetry) data model to a Red Hat Managed LokiStack instance. To enable this feature, add the **observability.openshift.io/tech-preview-otlp-output: "enabled"** annotation to your **ClusterLogForwarder** configuration. For additional configuration information, see OTLP Forwarding.
- With this update, a **dataModel** field has been added to the **lokiStack** output specification. Set the **dataModel** to **Otel** to configure log forwarding using the OpenTelemetry data format. The default is set to **Viaq**. For information about data mapping see OTLP Specification.

2.2.7.3. Bug Fixes

None.

2.2.7.4. CVEs

- CVE-2024-6119
- CVE-2024-6232

2.3. LOGGING 6.1

The **ClusterLogForwarder** custom resource (CR) is the central configuration point for log collection and forwarding.

2.3.1. Inputs and outputs

Inputs specify the sources of logs to be forwarded. Logging provides the following built-in input types that select logs from different parts of your cluster:

- **application**
- **receiver**
- **infrastructure**
- **audit**

You can also define custom inputs based on namespaces or pod labels to fine-tune log selection.

Outputs define the destinations where logs are sent. Each output type has its own set of configuration options, allowing you to customize the behavior and authentication settings.

2.3.2. Receiver input type

The receiver input type enables the Logging system to accept logs from external sources. It supports two formats for receiving logs: **http** and **syslog**.

The **ReceiverSpec** field defines the configuration for a receiver input.

2.3.3. Pipelines and filters

Pipelines determine the flow of logs from inputs to outputs. A pipeline consists of one or more input refs, output refs, and optional filter refs. You can use filters to transform or drop log messages within a pipeline. The order of filters matters, as they are applied sequentially, and earlier filters can prevent log messages from reaching later stages.

2.3.4. Operator behavior

The Cluster Logging Operator manages the deployment and configuration of the collector based on the **managementState** field of the **ClusterLogForwarder** resource:

- When set to **Managed** (default), the Operator actively manages the logging resources to match the configuration defined in the spec.
- When set to **Unmanaged**, the Operator does not take any action, allowing you to manually manage the logging components.

2.3.5. Validation

Logging includes extensive validation rules and default values to ensure a smooth and error-free configuration experience. The **ClusterLogForwarder** resource enforces validation checks on required fields, dependencies between fields, and the format of input values. Default values are provided for certain fields, reducing the need for explicit configuration in common scenarios.

2.3.6. Quick start

OpenShift Logging supports two data models:

- ViaQ (General Availability)
- OpenTelemetry (Technology Preview)

You can select either of these data models based on your requirement by configuring the **lokiStack.dataModel** field in the **ClusterLogForwarder**. ViaQ is the default data model when forwarding logs to LokiStack.

NOTE

In future releases of OpenShift Logging, the default data model will change from ViaQ to OpenTelemetry.

2.3.6.1. Quick start with ViaQ

To use the default ViaQ data model, follow these steps:

Prerequisites

- You have access to an OpenShift Container Platform cluster with **cluster-admin** permissions.
- You installed the OpenShift CLI (**oc**).
- You have access to a supported object store. For example, AWS S3, Google Cloud Storage, Azure, Swift, Minio, or OpenShift Data Foundation.

Procedure

1. Install the **Red Hat OpenShift Logging Operator**, **Loki Operator**, and **Cluster Observability Operator (COO)** from OperatorHub.
2. Create a **LokiStack** custom resource (CR) in the **openshift-logging** namespace:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  managementState: Managed
  size: 1x.extra-small
  storage:
    schemas:
    - effectiveDate: '2024-10-01'
      version: v13
    secret:
```

```
    name: logging-loki-s3
    type: s3
storageClassName: gp3-csi
tenants:
  mode: openshift-logging
```

NOTE

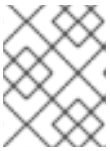
Ensure that the **logging-loki-s3** secret is created beforehand. The contents of this secret vary depending on the object storage in use. For more information, see Secrets and TLS Configuration.

3. Create a service account for the collector:

```
$ oc create sa collector -n openshift-logging
```

4. Allow the collector's service account to write data to the **LokiStack** CR:

```
$ oc adm policy add-cluster-role-to-user logging-collector-logs-writer -z collector -n openshift-logging
```



NOTE

The **ClusterRole** resource is created automatically during the Cluster Logging Operator installation and does not need to be created manually.

5. To collect logs, use the service account of the collector by running the following commands:

```
$ oc adm policy add-cluster-role-to-user collect-application-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-audit-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs -z collector -n openshift-logging
```

NOTE

The example binds the collector to all three roles (application, infrastructure, and audit), but by default, only application and infrastructure logs are collected. To collect audit logs, update your **ClusterLogForwarder** configuration to include them. Assign roles based on the specific log types required for your environment.

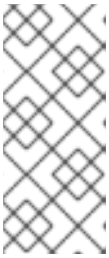
6. Create a **UIPlugin** CR to enable the **Log** section in the **Observe** tab:

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging
spec:
  type: Logging
```

```
  logging:
    lokiStack:
      name: logging-loki
```

7. Create a **ClusterLogForwarder** CR to configure log forwarding:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: collector
  namespace: openshift-logging
spec:
  serviceAccount:
    name: collector
  outputs:
  - name: default-lokistack
    type: lokiStack
    lokiStack:
      authentication:
        token:
          from: serviceAccount
      target:
        name: logging-loki
        namespace: openshift-logging
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - name: default-logstore
    inputRefs:
    - application
    - infrastructure
    outputRefs:
    - default-lokistack
```



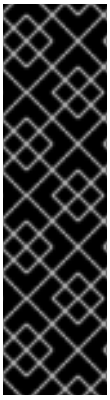
NOTE

The **dataModel** field is optional and left unset (**dataModel: ""**) by default. This allows the Cluster Logging Operator (CLO) to automatically select a data model. Currently, the CLO defaults to the ViaQ model when the field is unset, but this will change in future releases. Specifying **dataModel: ViaQ** ensures the configuration remains compatible if the default changes.

Verification

- Verify that logs are visible in the **Log** section of the **Observe** tab in the OpenShift Container Platform web console.

2.3.6.2. Quick start with OpenTelemetry



IMPORTANT

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

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

To configure OTLP ingestion and enable the OpenTelemetry data model, follow these steps:

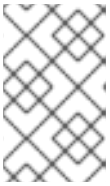
Prerequisites

- You have access to an OpenShift Container Platform cluster with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have access to a supported object store. For example, AWS S3, Google Cloud Storage, Azure, Swift, Minio, or OpenShift Data Foundation.

Procedure

1. Install the **Red Hat OpenShift Logging Operator**, **Loki Operator**, and **Cluster Observability Operator (COO)** from OperatorHub.
2. Create a **LokiStack** custom resource (CR) in the **openshift-logging** namespace:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  managementState: Managed
  size: 1x.extra-small
  storage:
    schemas:
    - effectiveDate: '2024-10-01'
      version: v13
    secret:
      name: logging-loki-s3
      type: s3
  storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```



NOTE

Ensure that the **logging-loki-s3** secret is created beforehand. The contents of this secret vary depending on the object storage in use. For more information, see "Secrets and TLS Configuration".

3. Create a service account for the collector:

```
$ oc create sa collector -n openshift-logging
```

4. Allow the collector's service account to write data to the **LokiStack** CR:

```
$ oc adm policy add-cluster-role-to-user logging-collector-logs-writer -z collector -n openshift-logging
```

NOTE

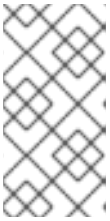
The **ClusterRole** resource is created automatically during the Cluster Logging Operator installation and does not need to be created manually.

5. To collect logs, use the service account of the collector by running the following commands:

```
$ oc adm policy add-cluster-role-to-user collect-application-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-audit-logs -z collector -n openshift-logging
```

```
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs -z collector -n openshift-logging
```



NOTE

The example binds the collector to all three roles (application, infrastructure, and audit). By default, only application and infrastructure logs are collected. To collect audit logs, update your **ClusterLogForwarder** configuration to include them. Assign roles based on the specific log types required for your environment.

6. Create a **UIPlugin** CR to enable the **Log** section in the **Observe** tab:

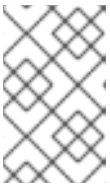
```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging
spec:
  type: Logging
  logging:
    lokiStack:
      name: logging-loki
```

7. Create a **ClusterLogForwarder** CR to configure log forwarding:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: collector
  namespace: openshift-logging
  annotations:
```

```
    observability.openshift.io/tech-preview-otlp-output: "enabled" 1
spec:
  serviceAccount:
    name: collector
  outputs:
  - name: loki-otlp
    type: lokiStack 2
    lokiStack:
      target:
        name: logging-loki
        namespace: openshift-logging
      dataModel: Otel 3
      authentication:
        token:
          from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - name: my-pipeline
    inputRefs:
    - application
    - infrastructure
    outputRefs:
    - loki-otlp
```

1. Use the annotation to enable the **Otel** data model, which is a Technology Preview feature.
2. Define the output type as **lokiStack**.
3. Specifies the OpenTelemetry data model.



NOTE

You cannot use **lokiStack.labelKeys** when **dataModel** is **Otel**. To achieve similar functionality when **dataModel** is **Otel**, refer to "Configuring LokiStack for OTLP data ingestion".

Verification

- To verify that OTLP is functioning correctly, complete the following steps:
  a. In the OpenShift web console, click **Observe → OpenShift Logging → LokiStack → Writes**.
  b. Check the **Distributor - Structured Metadata**section.

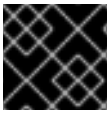
2.4. INSTALLING LOGGING

OpenShift Container Platform Operators use custom resources (CRs) to manage applications and their components. You provide high-level configuration and settings through the CR. The Operator translates high-level directives into low-level actions, based on best practices embedded within the logic of the Operator. A custom resource definition (CRD) defines a CR and lists all the configurations available to users of the Operator. Installing an Operator creates the CRDs to generate CRs.

To get started with logging, you must install the following Operators:

- Loki Operator to manage your log store.
- Red Hat OpenShift Logging Operator to manage log collection and forwarding.
- Cluster Observability Operator (COO) to manage visualization.

You can use either the OpenShift Container Platform web console or the OpenShift Container Platform CLI to install or configure logging.



IMPORTANT

You must configure the Red Hat OpenShift Logging Operator after the Loki Operator.

2.4.1. Installation by using the CLI

The following sections describe installing the Loki Operator and the Red Hat OpenShift Logging Operator by using the CLI.

2.4.1.1. Installing the Loki Operator by using the CLI

Install Loki Operator on your OpenShift Container Platform cluster to manage the log store **Loki** by using the OpenShift Container Platform command-line interface (CLI). You can deploy and configure the **Loki** log store by reconciling the resource LokiStack with the Loki Operator.

Prerequisites

- You have administrator permissions.
- You installed the OpenShift CLI (**oc**).
- You have access to a supported object store. For example: AWS S3, Google Cloud Storage, Azure, Swift, Minio, or OpenShift Data Foundation.

Procedure

1. Create a **Namespace** object for Loki Operator:

   Example Namespace object

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: openshift-operators-redhat 1
     labels:
       openshift.io/cluster-monitoring: "true" 2
   ```

   1 You must specify **openshift-operators-redhat** as the namespace. To enable monitoring for the operator, configure Cluster Monitoring Operator to scrape metrics from the **openshift-operators-redhat** namespace and not the **openshift-operators** namespace. The **openshift-operators** namespace might contain community operators, which are untrusted and could publish a metric with the same name as an OpenShift Container Platform metric, causing conflicts.

2 A string value that specifies the label as shown to ensure that cluster monitoring scrapes the **openshift-operators-redhat** namespace.

2. Apply the **Namespace** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

3. Create an **OperatorGroup** object.

Example OperatorGroup object

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat 1
spec:
  upgradeStrategy: Default
```

1 You must specify **openshift-operators-redhat** as the namespace.

4. Apply the **OperatorGroup** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

5. Create a **Subscription** object for Loki Operator:

Example Subscription object

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat 1
spec:
  channel: stable-6.<y> 2
  installPlanApproval: Automatic 3
  name: loki-operator
  source: redhat-operators 4
  sourceNamespace: openshift-marketplace
```

1 You must specify **openshift-operators-redhat** as the namespace.

2 Specify **stable-6.<y>** as the channel.

3 If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

4 Specify **redhat-operators** as the value. If your OpenShift Container Platform cluster is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object that you created when you configured Operator Lifecycle

Manager (OLM).

6. Apply the **Subscription** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

7. Create a **namespace** object for deploy the LokiStack:

Example namespace object

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging (1)
  labels:
    openshift.io/cluster-monitoring: "true" (2)
```

(1) The **openshift-logging** namespace is dedicated for all logging workloads.

(2) A string value that specifies the label, as shown, to ensure that cluster monitoring scrapes the **openshift-logging** namespace.

8. Apply the **namespace** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

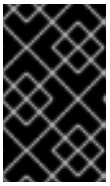
9. Create a secret with the credentials to access the object storage. For example, create a secret to access Amazon Web Services (AWS) s3.

Example Secret object

```
apiVersion: v1
kind: Secret
metadata:
  name: logging-loki-s3 (1)
  namespace: openshift-logging
stringData: (2)
  access_key_id: <access_key_id>
  access_key_secret: <access_secret>
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1
```

(1) Use the name **logging-loki-s3** to match the name used in LokiStack.

(2) For the contents of the secret see the Loki object storage section.



IMPORTANT

If there is no retention period defined on the s3 bucket or in the LokiStack custom resource (CR), then the logs are not pruned and they stay in the s3 bucket forever, which might fill up the s3 storage.

10. Apply the **Secret** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

11. Create a **LokiStack** CR:

Example LokiStack CR

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki (1)
  namespace: openshift-logging (2)
spec:
  size: 1x.small (3)
  storage:
    schemas:
    - version: v13
      effectiveDate: "<yyyy>-<mm>-<dd>" (4)
    secret:
      name: logging-loki-s3 (5)
      type: s3 (6)
  storageClassName: <storage_class_name> (7)
  tenants:
    mode: openshift-logging (8)
```

(1) Use the name **logging-loki**.

(2) You must specify **openshift-logging** as the namespace.

(3) Specify the deployment size. Supported size options for production instances of Loki are **1x.extra-small**, **1x.small**, or **1x.medium**. Additionally, **1x.pico** is supported starting with logging 6.1.

(4) For new installations this date should be set to the equivalent of "yesterday", as this will be the date from when the schema takes effect.

(5) Specify the name of your log store secret.

(6) Specify the corresponding storage type.

(7) Specify the name of a storage class for temporary storage. For best performance, specify a storage class that allocates block storage. You can list the available storage classes for your cluster by using the **oc get storageclasses** command.

(8) The **openshift-logging** mode is the default tenancy mode where a tenant is created for log types, such as audit, infrastructure, and application. This enables access control for individual users and user groups to different log streams.

12. Apply the **LokiStack** CR object by running the following command:

```
$ oc apply -f <filename>.yaml
```

Verification

- Verify the installation by running the following command:

```
$ oc get pods -n openshift-logging
```

Example output

```
$ oc get pods -n openshift-logging
NAME                                READY   STATUS    RESTARTS   AGE
logging-loki-compactor-0            1/1     Running   0          42m
logging-loki-distributor-7d7688bcb9-dvcj8   1/1     Running   0          42m
logging-loki-gateway-5f6c75f879-bl7k9       2/2     Running   0          42m
logging-loki-gateway-5f6c75f879-xhq98       2/2     Running   0          42m
logging-loki-index-gateway-0        1/1     Running   0          42m
logging-loki-ingester-0             1/1     Running   0          42m
logging-loki-querier-6b7b56bccc-2v9q4       1/1     Running   0          42m
logging-loki-query-frontend-84fb57c578-gq2f7   1/1     Running   0          42m
```

2.4.1.2. Installing Red Hat OpenShift Logging Operator by using the CLI

Install Red Hat OpenShift Logging Operator on your OpenShift Container Platform cluster to collect and forward logs to a log store by using the OpenShift CLI (**oc**).

Prerequisites

- You have administrator permissions.
- You installed the OpenShift CLI (**oc**).
- You installed and configured Loki Operator.
- You have created the **openshift-logging** namespace.

Procedure

1. Create an **OperatorGroup** object:

Example OperatorGroup object

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging 1
spec:
  upgradeStrategy: Default
```

1 You must specify **openshift-logging** as the namespace.

2. Apply the **OperatorGroup** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

3. Create a **Subscription** object for Red Hat OpenShift Logging Operator:

Example Subscription object

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: cluster-logging
  namespace: openshift-logging (1)
spec:
  channel: stable-6.<y> (2)
  installPlanApproval: Automatic (3)
  name: cluster-logging
  source: redhat-operators (4)
  sourceNamespace: openshift-marketplace
```

(1) You must specify **openshift-logging** as the namespace.

(2) Specify **stable-6.<y>** as the channel.

(3) If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

(4) Specify **redhat-operators** as the value. If your OpenShift Container Platform cluster is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object that you created when you configured Operator Lifecycle Manager (OLM).

4. Apply the **Subscription** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

5. Create a service account to be used by the log collector:

```
$ oc create sa logging-collector -n openshift-logging
```

6. Assign the necessary permissions to the service account for the collector to be able to collect and forward logs. In this example, the collector is provided permissions to collect logs from both infrastructure and application logs.

```
$ oc adm policy add-cluster-role-to-user logging-collector-logs-writer -z logging-collector -n openshift-logging
$ oc adm policy add-cluster-role-to-user collect-application-logs -z logging-collector -n openshift-logging
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs -z logging-collector -n openshift-logging
```

7. Create a **ClusterLogForwarder** CR:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging (1)
spec:
  serviceAccount:
    name: logging-collector (2)
  outputs:
  - name: lokistack-out
    type: lokiStack (3)
    lokiStack:
      target: (4)
        name: logging-loki
        namespace: openshift-logging
      authentication:
        token:
          from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - name: infra-app-logs
    inputRefs: (5)
    - application
    - infrastructure
    outputRefs:
    - lokistack-out
```

(1) You must specify the **openshift-logging** namespace.

(2) Specify the name of the service account created before.

(3) Select the **lokiStack** output type to send logs to the **LokiStack** instance.

(4) Point the **ClusterLogForwarder** to the **LokiStack** instance created earlier.

(5) Select the log output types you want to send to the **LokiStack** instance.

8. Apply the **ClusterLogForwarder CR** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

Verification

1. Verify the installation by running the following command:

```
$ oc get pods -n openshift-logging
```

Example output

```
$ oc get pods -n openshift-logging
NAME                                READY   STATUS    RESTARTS   AGE
cluster-logging-operator-fb7f7cf69-8jsbq   1/1     Running   0          98m
instance-222js                      2/2     Running   0          18m
instance-g9ddv                      2/2     Running   0          18m
instance-hfqq8                      2/2     Running   0          18m
instance-sphwg                      2/2     Running   0          18m
instance-vv7zn                      2/2     Running   0          18m
instance-wk5zz                      2/2     Running   0          18m
logging-loki-compactor-0            1/1     Running   0          42m
logging-loki-distributor-7d7688bcb9-dvcj8   1/1     Running   0          42m
logging-loki-gateway-5f6c75f879-bl7k9       2/2     Running   0          42m
logging-loki-gateway-5f6c75f879-xhq98       2/2     Running   0          42m
logging-loki-index-gateway-0        1/1     Running   0          42m
logging-loki-ingester-0             1/1     Running   0          42m
logging-loki-querier-6b7b56bccc-2v9q4       1/1     Running   0          42m
logging-loki-query-frontend-84fb57c578-gq2f7   1/1     Running   0          42m
```

nclude::modules/log6x-installing-the-logging-ui-plug-in-cli.adoc[leveloffset=+2]

2.4.2. Installation by using the web console

The following sections describe installing the Loki Operator and the Red Hat OpenShift Logging Operator by using the web console.

2.4.2.1. Installing Logging by using the web console

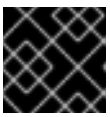
Install Loki Operator on your OpenShift Container Platform cluster to manage the log store **Loki** from the OperatorHub by using the OpenShift Container Platform web console. You can deploy and configure the **Loki** log store by reconciling the resource LokiStack with the Loki Operator.

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console **Administrator** perspective, go to **Operators → OperatorHub**.
2. Type Loki Operator in the **Filter by keyword** field. Click **Loki Operator** in the list of available Operators, and then click **Install**.



IMPORTANT

The Community Loki Operator is not supported by Red Hat.

3. Select **stable-x.y** as the **Update channel**.

The Loki Operator must be deployed to the global Operator group namespace **openshift-operators-redhat**, so the **Installation mode** and **Installed Namespace** are already selected. If this namespace does not already exist, it will be created for you.

4. Select **Enable Operator-recommended cluster monitoring on this namespace.**
This option sets the **openshift.io/cluster-monitoring: "true"** label in the **Namespace** object. You must select this option to ensure that cluster monitoring scrapes the **openshift-operators-redhat** namespace.

5. For **Update approval** select **Automatic**, then click **Install**.
If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new Operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

NOTE

An Operator might display a **Failed** status before the installation completes. If the Operator install completes with an **InstallSucceeded** message, refresh the page.

6. While the Operator installs, create the namespace to which the log store will be deployed.
   a. Click **+** in the top right of the screen to access the **Import YAML** page.
   b. Add the YAML definition for the **openshift-logging** namespace:

Example namespace object

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-logging (1)
  labels:
    openshift.io/cluster-monitoring: "true" (2)
```

(1) The **openshift-logging** namespace is dedicated for all logging workloads.

(2) A string value that specifies the label, as shown, to ensure that cluster monitoring scrapes the **openshift-logging** namespace.

   c. Click **Create**.

7. Create a secret with the credentials to access the object storage.
   a. Click **+** in the top right of the screen to access the **Import YAML** page.
   b. Add the YAML definition for the secret. For example, create a secret to access Amazon Web Services (AWS) s3:

Example Secret object

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

```
    name: logging-loki-s3 (1)
    namespace: openshift-logging (2)
  stringData: (3)
    access_key_id: <access_key_id>
    access_key_secret: <access_key>
    bucketnames: s3-bucket-name
    endpoint: https://s3.eu-central-1.amazonaws.com
    region: eu-central-1
```

(1) Note down the name used for the secret **logging-loki-s3** to use it later when creating the **LokiStack** resource.

(2) Set the namespace to **openshift-logging** as that will be the namespace used to deploy **LokiStack**.

(3) For the contents of the secret see the Loki object storage section.

IMPORTANT

If there is no retention period defined on the s3 bucket or in the LokiStack custom resource (CR), then the logs are not pruned and they stay in the s3 bucket forever, which might fill up the s3 storage.

c. Click **Create**.

8. Navigate to the **Installed Operators** page. Select the Loki Operator under the **Provided APIs** find the **LokiStack** resource and click **Create Instance**.

9. Select **YAML view**, and then use the following template to create a **LokiStack** CR:

Example LokiStack CR

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki (1)
  namespace: openshift-logging (2)
spec:
  size: 1x.small (3)
  storage:
    schemas:
    - version: v13
      effectiveDate: "<yyyy>-<mm>-<dd>"
    secret:
      name: logging-loki-s3 (4)
      type: s3 (5)
  storageClassName: <storage_class_name> (6)
  tenants:
    mode: openshift-logging (7)
```

(1) Use the name **logging-loki**.

2 You must specify **openshift-logging** as the namespace.

3 Specify the deployment size. Supported size options for production instances of Loki are **1x.extra-small**, **1x.small**, or **1x.medium**. Additionally, 1x.pico is supported starting with logging 6.1.

4 Specify the name of your log store secret.

5 Specify the corresponding storage type.

6 Specify the name of a storage class for temporary storage. For best performance, specify a storage class that allocates block storage. You can list the available storage classes for your cluster by using the **oc get storageclasses** command.

7 The **openshift-logging** mode is the default tenancy mode where a tenant is created for log types, such as audit, infrastructure, and application. This enables access control for individual users and user groups to different log streams.

10. Click **Create**.

Verification

1. In the **LokiStack** tab veriy that you see your **LokiStack** instance.
2. In the **Status** column, verify that you see the message **Condition: Ready** with a green checkmark.

2.4.2.2. Installing Red Hat OpenShift Logging Operator by using the web console

Install Red Hat OpenShift Logging Operator on your OpenShift Container Platform cluster to collect and forward logs to a log store from the OperatorHub by using the OpenShift Container Platform web console.

Prerequisites

- You have administrator permissions.
- You have access to the OpenShift Container Platform web console.
- You installed and configured Loki Operator.

Procedure

1. In the OpenShift Container Platform web console **Administrator** perspective, go to **Operators → OperatorHub**.
2. Type Red Hat OpenShift Logging Operator in the **Filter by keyword** field. Click **Red Hat OpenShift Logging Operator** in the list of available Operators, and then click **Install**.
3. Select **stable-x.y** as the **Update channel**. The latest version is already selected in the **Version** field.
   The Red Hat OpenShift Logging Operator must be deployed to the logging namespace **openshift-logging**, so the **Installation mode** and **Installed Namespace** are already selected. If this namespace does not already exist, it will be created for you.

4. Select **Enable Operator-recommended cluster monitoring on this namespace.**
   This option sets the **openshift.io/cluster-monitoring: "true"** label in the **Namespace** object. You must select this option to ensure that cluster monitoring scrapes the **openshift-logging** namespace.

5. For **Update approval** select **Automatic**, then click **Install**.
   If the approval strategy in the subscription is set to **Automatic**, the update process initiates as soon as a new operator version is available in the selected channel. If the approval strategy is set to **Manual**, you must manually approve pending updates.

NOTE

An Operator might display a **Failed** status before the installation completes. If the operator installation completes with an **InstallSucceeded** message, refresh the page.

6. While the operator installs, create the service account that will be used by the log collector to collect the logs.

   a. Click the **+** in the top right of the screen to access the **Import YAML** page.

   b. Enter the YAML definition for the service account.

Example ServiceAccount object

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: logging-collector 1
  namespace: openshift-logging 2
```

1. Note down the name used for the service account **logging-collector** to use it later when creating the **ClusterLogForwarder** resource.

2. Set the namespace to **openshift-logging** because that is the namespace for deploying the **ClusterLogForwarder** resource.

   c. Click the **Create** button.

7. Create the **ClusterRoleBinding** objects to grant the necessary permissions to the log collector for accessing the logs that you want to collect and to write the log store, for example infrastructure and application logs.

   a. Click the **+** in the top right of the screen to access the **Import YAML** page.

   b. Enter the YAML definition for the **ClusterRoleBinding** resources.

Example ClusterRoleBinding resources

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: logging-collector:write-logs
roleRef:
```

```
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: logging-collector-logs-writer (1)
subjects:
- kind: ServiceAccount
  name: logging-collector
  namespace: openshift-logging
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: logging-collector:collect-application
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: collect-application-logs (2)
subjects:
- kind: ServiceAccount
  name: logging-collector
  namespace: openshift-logging
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: logging-collector:collect-infrastructure
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: collect-infrastructure-logs (3)
subjects:
- kind: ServiceAccount
  name: logging-collector
  namespace: openshift-logging
```

(1) The cluster role to allow the log collector to write logs to LokiStack.

(2) The cluster role to allow the log collector to collect logs from applications.

(3) The cluster role to allow the log collector to collect logs from infrastructure.

c. Click the **Create** button.

8. Go to the **Operators → Installed Operators** page. Select the operator and click the **All instances** tab.

9. After granting the necessary permissions to the service account, navigate to the **Installed Operators** page. Select the Red Hat OpenShift Logging Operator under the **Provided APIs**, find the **ClusterLogForwarder** resource and click **Create Instance**.

10. Select **YAML view**, and then use the following template to create a **ClusterLogForwarder** CR:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
```

```
metadata:
  name: instance
  namespace: openshift-logging 1
spec:
  serviceAccount:
    name: logging-collector 2
  outputs:
  - name: lokistack-out
    type: lokiStack 3
    lokiStack:
      target: 4
        name: logging-loki
        namespace: openshift-logging
      authentication:
        token:
          from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - name: infra-app-logs
    inputRefs: 5
    - application
    - infrastructure
    outputRefs:
    - lokistack-out
```

1. You must specify **openshift-logging** as the namespace.
2. Specify the name of the service account created earlier.
3. Select the **lokiStack** output type to send logs to the **LokiStack** instance.
4. Point the **ClusterLogForwarder** to the **LokiStack** instance created earlier.
5. Select the log output types you want to send to the **LokiStack** instance.

11. Click **Create**.

Verification

1. In the **ClusterLogForwarder** tab verify that you see your **ClusterLogForwarder** instance.
2. In the **Status** column, verify that you see the messages:
   - **Condition: observability.openshift.io/Authorized**
   - **observability.openshift.io/Valid, Ready**

2.4.2.3. Installing the Logging UI plugin by using the web console

Install the Logging UI plugin by using the web console so that you can visualize logs.

Prerequisites

- You have administrator permissions.
- You have access to the OpenShift Container Platform web console.
- You installed and configured Loki Operator.

Procedure

1. Install the Cluster Observability Operator. For more information, see Installing the Cluster Observability Operator.
2. Navigate to the **Installed Operators** page. Under **Provided APIs**, select **ClusterObservabilityOperator**. Find the **UIPlugin** resource and click **Create Instance**.
3. Select the YAML view, and then use the following template to create a **UIPlugin** custom resource (CR):

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging 1
spec:
  type: Logging 2
  logging:
    lokiStack:
      name: logging-loki 3
```

1 Set **name** to **logging**.

2 Set **type** to **Logging**.

3 The **name** value must match the name of your LokiStack instance.

NOTE

If you did not install LokiStack in the **openshift-logging** namespace, set the LokiStack namespace under the **lokiStack** configuration.

4. Click **Create**.

Verification

1. Refresh the page when a pop-up message instructs you to do so.
2. Navigate to the **Observe → Logs** panel, where you can run LogQL queries. You can also query logs for individual pods from the **Aggregated Logs** tab of a specific pod.

Additional resources

- About OVN-Kubernetes network policy

2.5. CONFIGURING LOG FORWARDING

The **ClusterLogForwarder** (CLF) allows users to configure forwarding of logs to various destinations. It provides a flexible way to select log messages from different sources, send them through a pipeline that can transform or filter them, and forward them to one or more outputs.

Key Functions of the ClusterLogForwarder

- Selects log messages using inputs
- Forwards logs to external destinations using outputs
- Filters, transforms, and drops log messages using filters
- Defines log forwarding pipelines connecting inputs, filters and outputs

2.5.1. Setting up log collection

This release of Cluster Logging requires administrators to explicitly grant log collection permissions to the service account associated with **ClusterLogForwarder**. This was not required in previous releases for the legacy logging scenario consisting of a **ClusterLogging** and, optionally, a **ClusterLogForwarder.logging.openshift.io** resource.

The Red Hat OpenShift Logging Operator provides **collect-audit-logs**, **collect-application-logs**, and **collect-infrastructure-logs** cluster roles, which enable the collector to collect audit logs, application logs, and infrastructure logs respectively.

Setup log collection by binding the required cluster roles to your service account.

2.5.1.1. Legacy service accounts

To use the existing legacy service account **logcollector**, create the following **ClusterRoleBinding**:

```
$ oc adm policy add-cluster-role-to-user collect-application-logs system:serviceaccount:openshift-logging:logcollector
```

```
$ oc adm policy add-cluster-role-to-user collect-infrastructure-logs system:serviceaccount:openshift-logging:logcollector
```

Additionally, create the following **ClusterRoleBinding** if collecting audit logs:

```
$ oc adm policy add-cluster-role-to-user collect-audit-logs system:serviceaccount:openshift-logging:logcollector
```

2.5.1.2. Creating service accounts

Prerequisites

- The Red Hat OpenShift Logging Operator is installed in the **openshift-logging** namespace.
- You have administrator permissions.

Procedure

1. Create a service account for the collector. If you want to write logs to storage that requires a token for authentication, you must include a token in the service account.

2. Bind the appropriate cluster roles to the service account:

Example binding command

```
$ oc adm policy add-cluster-role-to-user <cluster_role_name> system:serviceaccount:
<namespace_name>:<service_account_name>
```

2.5.1.2.1. Cluster Role Binding for your Service Account

The role_binding.yaml file binds the ClusterLogging operator's ClusterRole to a specific ServiceAccount, allowing it to manage Kubernetes resources cluster-wide.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: manager-rolebinding
roleRef:                                  1
  apiGroup: rbac.authorization.k8s.io     2
  kind: ClusterRole                       3
  name: cluster-logging-operator          4
subjects:                                 5
  - kind: ServiceAccount                  6
    name: cluster-logging-operator        7
    namespace: openshift-logging          8
```

1. roleRef: References the ClusterRole to which the binding applies.
2. apiGroup: Indicates the RBAC API group, specifying that the ClusterRole is part of Kubernetes' RBAC system.
3. kind: Specifies that the referenced role is a ClusterRole, which applies cluster-wide.
4. name: The name of the ClusterRole being bound to the ServiceAccount, here cluster-logging-operator.
5. subjects: Defines the entities (users or service accounts) that are being granted the permissions from the ClusterRole.
6. kind: Specifies that the subject is a ServiceAccount.
7. Name: The name of the ServiceAccount being granted the permissions.
8. namespace: Indicates the namespace where the ServiceAccount is located.

2.5.1.2.2. Writing application logs

The write-application-logs-clusterrole.yaml file defines a ClusterRole that grants permissions to write application logs to the Loki logging application.

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRole
metadata:
  name: cluster-logging-write-application-logs
rules:                                          1
  - apiGroups:                                  2
      - loki.grafana.com                        3
    resources:                                  4
      - application                             5
    resourceNames:                              6
      - logs                                    7
    verbs:                                      8
      - create                                  9
```

1. rules: Specifies the permissions granted by this ClusterRole.
2. apiGroups: Refers to the API group loki.grafana.com, which relates to the Loki logging system.
3. loki.grafana.com: The API group for managing Loki-related resources.
4. resources: The resource type that the ClusterRole grants permission to interact with.
5. application: Refers to the application resources within the Loki logging system.
6. resourceNames: Specifies the names of resources that this role can manage.
7. logs: Refers to the log resources that can be created.
8. verbs: The actions allowed on the resources.
9. create: Grants permission to create new logs in the Loki system.

2.5.1.2.3. Writing audit logs

The write-audit-logs-clusterrole.yaml file defines a ClusterRole that grants permissions to create audit logs in the Loki logging system.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-logging-write-audit-logs
rules:                                          1
  - apiGroups:                                  2
      - loki.grafana.com                        3
    resources:                                  4
      - audit                                   5
    resourceNames:                              6
      - logs                                    7
    verbs:                                      8
      - create                                  9
```

1. rules: Defines the permissions granted by this ClusterRole.

2 apiGroups: Specifies the API group loki.grafana.com.

3 loki.grafana.com: The API group responsible for Loki logging resources.

4 resources: Refers to the resource type this role manages, in this case, audit.

5 audit: Specifies that the role manages audit logs within Loki.

6 resourceNames: Defines the specific resources that the role can access.

7 logs: Refers to the logs that can be managed under this role.

8 verbs: The actions allowed on the resources.

9 create: Grants permission to create new audit logs.

2.5.1.2.4. Writing infrastructure logs

The write-infrastructure-logs-clusterrole.yaml file defines a ClusterRole that grants permission to create infrastructure logs in the Loki logging system.

Sample YAML

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-logging-write-infrastructure-logs
rules:                                  1
  - apiGroups:                          2
      - loki.grafana.com                3
    resources:                          4
      - infrastructure                  5
    resourceNames:                      6
      - logs                            7
    verbs:                              8
      - create                          9
```

1 rules: Specifies the permissions this ClusterRole grants.

2 apiGroups: Specifies the API group for Loki-related resources.

3 loki.grafana.com: The API group managing the Loki logging system.

4 resources: Defines the resource type that this role can interact with.

5 infrastructure: Refers to infrastructure-related resources that this role manages.

6 resourceNames: Specifies the names of resources this role can manage.

7 logs: Refers to the log resources related to infrastructure.

8 verbs: The actions permitted by this role.

9 create: Grants permission to create infrastructure logs in the Loki system.

2.5.1.2.5. ClusterLogForwarder editor role

The clusterlogforwarder-editor-role.yaml file defines a ClusterRole that allows users to manage ClusterLogForwarders in OpenShift.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: clusterlogforwarder-editor-role
rules:                                    (1)
  - apiGroups:                            (2)
      - observability.openshift.io        (3)
    resources:                            (4)
      - clusterlogforwarders              (5)
    verbs:                                (6)
      - create                            (7)
      - delete                            (8)
      - get                               (9)
      - list                              (10)
      - patch                             (11)
      - update                            (12)
      - watch                             (13)
```

1. rules: Specifies the permissions this ClusterRole grants.
2. apiGroups: Refers to the OpenShift-specific API group
3. obervability.openshift.io: The API group for managing observability resources, like logging.
4. resources: Specifies the resources this role can manage.
5. clusterlogforwarders: Refers to the log forwarding resources in OpenShift.
6. verbs: Specifies the actions allowed on the ClusterLogForwarders.
7. create: Grants permission to create new ClusterLogForwarders.
8. delete: Grants permission to delete existing ClusterLogForwarders.
9. get: Grants permission to retrieve information about specific ClusterLogForwarders.
10. list: Allows listing all ClusterLogForwarders.
11. patch: Grants permission to partially modify ClusterLogForwarders.
12. update: Grants permission to update existing ClusterLogForwarders.
13. watch: Grants permission to monitor changes to ClusterLogForwarders.

2.5.2. Modifying log level in collector

To modify the log level in the collector, you can set the **observability.openshift.io/log-level** annotation to **trace**, **debug**, **info**, **warn**, **error**, and **off**.

Example log level annotation

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: collector
  annotations:
    observability.openshift.io/log-level: debug
# ...
```

2.5.3. Managing the Operator

The **ClusterLogForwarder** resource has a **managementState** field that controls whether the operator actively manages its resources or leaves them Unmanaged:

Managed

(default) The operator will drive the logging resources to match the desired state in the CLF spec.

Unmanaged

The operator will not take any action related to the logging components.

This allows administrators to temporarily pause log forwarding by setting **managementState** to **Unmanaged**.

2.5.4. Structure of the ClusterLogForwarder

The CLF has a **spec** section that contains the following key components:

Inputs

Select log messages to be forwarded. Built-in input types **application**, **infrastructure** and **audit** forward logs from different parts of the cluster. You can also define custom inputs.

Outputs

Define destinations to forward logs to. Each output has a unique name and type-specific configuration.

Pipelines

Define the path logs take from inputs, through filters, to outputs. Pipelines have a unique name and consist of a list of input, output and filter names.

Filters

Transform or drop log messages in the pipeline. Users can define filters that match certain log fields and drop or modify the messages. Filters are applied in the order specified in the pipeline.

2.5.4.1. Inputs

Inputs are configured in an array under **spec.inputs**. There are three built-in input types:

application

Selects logs from all application containers, excluding those in infrastructure namespaces.

infrastructure

Selects logs from nodes and from infrastructure components running in the following namespaces:

- **default**

- **kube**
- **openshift**
- Containing the **kube-** or **openshift-** prefix

audit

Selects logs from the OpenShift API server audit logs, Kubernetes API server audit logs, ovn audit logs, and node audit logs from auditd.

Users can define custom inputs of type **application** that select logs from specific namespaces or using pod labels.

2.5.4.2. Outputs

Outputs are configured in an array under **spec.outputs**. Each output must have a unique name and a type. Supported types are:

azureMonitor

Forwards logs to Azure Monitor.

cloudwatch

Forwards logs to AWS CloudWatch.

googleCloudLogging

Forwards logs to Google Cloud Logging.

http

Forwards logs to a generic HTTP endpoint.

kafka

Forwards logs to a Kafka broker.

loki

Forwards logs to a Loki logging backend.

lokistack

Forwards logs to the logging supported combination of Loki and web proxy with OpenShift Container Platform authentication integration. LokiStack's proxy uses OpenShift Container Platform authentication to enforce multi-tenancy

otlp

Forwards logs using the OpenTelemetry Protocol.

splunk

Forwards logs to Splunk.

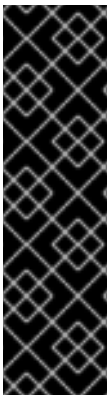
syslog

Forwards logs to an external syslog server.

Each output type has its own configuration fields.

2.5.5. Configuring OTLP output

Cluster administrators can use the OpenTelemetry Protocol (OTLP) output to collect and forward logs to OTLP receivers. The OTLP output uses the specification defined by the OpenTelemetry Observability framework to send data over HTTP with JSON encoding.



IMPORTANT

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

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

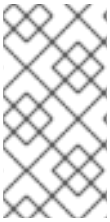
Procedure

- Create or edit a **ClusterLogForwarder** custom resource (CR) to enable forwarding using OTLP by adding the following annotation:

  Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  annotations:
    observability.openshift.io/tech-preview-otlp-output: "enabled" 1
  name: clf-otlp
spec:
  serviceAccount:
    name: <service_account_name>
  outputs:
  - name: otlp
    type: otlp
    otlp:
      tuning:
        compression: gzip
        deliveryMode: AtLeastOnce
        maxRetryDuration: 20
        maxWrite: 10M
        minRetryDuration: 5
      url: <otlp_url> 2
  pipelines:
  - inputRefs:
    - application
    - infrastructure
    - audit
    name: otlp-logs
    outputRefs:
    - otlp
```

1. Use this annotation to enable the OpenTelemetry Protocol (OTLP) output, which is a Technology Preview feature.
2. This URL must be absolute and is a placeholder for the OTLP endpoint where logs are sent.



NOTE

The OTLP output uses the OpenTelemetry data model, which is different from the ViaQ data model that is used by other output types. It adheres to the OTLP using OpenTelemetry Semantic Conventions defined by the OpenTelemetry Observability framework.

2.5.5.1. Pipelines

Pipelines are configured in an array under **spec.pipelines**. Each pipeline must have a unique name and consists of:

inputRefs

Names of inputs whose logs should be forwarded to this pipeline.

outputRefs

Names of outputs to send logs to.

filterRefs

(optional) Names of filters to apply.

The order of filterRefs matters, as they are applied sequentially. Earlier filters can drop messages that will not be processed by later filters.

2.5.5.2. Filters

Filters are configured in an array under **spec.filters**. They can match incoming log messages based on the value of structured fields and modify or drop them.

Administrators can configure the following types of filters:

2.5.5.3. Enabling multi-line exception detection

Enables multi-line error detection of container logs.

WARNING

Enabling this feature could have performance implications and may require additional computing resources or alternate logging solutions.

Log parsers often incorrectly identify separate lines of the same exception as separate exceptions. This leads to extra log entries and an incomplete or inaccurate view of the traced information.

Example java exception

```
java.lang.NullPointerException: Cannot invoke "String.toString()" because "<param1>" is null
    at testjava.Main.handle(Main.java:47)
    at testjava.Main.printMe(Main.java:19)
    at testjava.Main.main(Main.java:10)
```

- To enable logging to detect multi-line exceptions and reassemble them into a single log entry, ensure that the **ClusterLogForwarder** Custom Resource (CR) contains a **detectMultilineErrors** field under the **.spec.filters**.

Example ClusterLogForwarder CR

```
apiVersion: "observability.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name>
  namespace: <log_forwarder_namespace>
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: <name>
    type: detectMultilineException
  pipelines:
    - inputRefs:
        - <input-name>
      name: <pipeline-name>
      filterRefs:
        - <filter-name>
      outputRefs:
        - <output-name>
```

2.5.5.3.1. Details

When log messages appear as a consecutive sequence forming an exception stack trace, they are combined into a single, unified log record. The first log message's content is replaced with the concatenated content of all the message fields in the sequence.

The collector supports the following languages:

- Java
- JS
- Ruby
- Python
- Golang
- PHP
- Dart

2.5.5.4. Configuring content filters to drop unwanted log records

When the **drop** filter is configured, the log collector evaluates log streams according to the filters before forwarding. The collector drops unwanted log records that match the specified configuration.

Procedure

1. Add a configuration for a filter to the **filters** spec in the **ClusterLogForwarder** CR. The following example shows how to configure the **ClusterLogForwarder** CR to drop log records based on regular expressions:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: <filter_name>
    type: drop (1)
    drop: (2)
    - test: (3)
      - field: .kubernetes.labels."foo-bar/baz" (4)
        matches: .+ (5)
      - field: .kubernetes.pod_name
        notMatches: "my-pod" (6)
  pipelines:
  - name: <pipeline_name> (7)
    filterRefs: ["<filter_name>"]
# ...
```

(1) Specifies the type of filter. The **drop** filter drops log records that match the filter configuration.

(2) Specifies configuration options for applying the **drop** filter.

(3) Specifies the configuration for tests that are used to evaluate whether a log record is dropped.

- If all the conditions specified for a test are true, the test passes and the log record is dropped.
- When multiple tests are specified for the **drop** filter configuration, if any of the tests pass, the record is dropped.
- If there is an error evaluating a condition, for example, the field is missing from the log record being evaluated, that condition evaluates to false.

(4) Specifies a dot-delimited field path, which is a path to a field in the log record. The path can contain alpha-numeric characters and underscores (**a-zA-Z0-9_**), for example, **.kubernetes.namespace_name**. If segments contain characters outside of this range, the segment must be in quotes, for example, **.kubernetes.labels."foo.bar-bar/baz"**. You can include multiple field paths in a single **test** configuration, but they must all evaluate to true for the test to pass and the **drop** filter to be applied.

(5) Specifies a regular expression. If log records match this regular expression, they are dropped. You can set either the **matches** or **notMatches** condition for a single **field** path, but not both.

6 Specifies a regular expression. If log records do not match this regular expression, they are dropped. You can set either the **matches** or **notMatches** condition for a single **field** path, but not both.

7 Specifies the pipeline that the **drop** filter is applied to.

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

Additional examples

The following additional example shows how you can configure the **drop** filter to only keep higher priority log records:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: important
    type: drop
    drop:
    - test:
      - field: .message
        notMatches: "(?i)critical|error"
      - field: .level
        matches: "info|warning"
# ...
```

In addition to including multiple field paths in a single **test** configuration, you can also include additional tests that are treated as *OR* checks. In the following example, records are dropped if either **test** configuration evaluates to true. However, for the second **test** configuration, both field specs must be true for it to be evaluated to true:

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: important
    type: drop
    drop:
    - test:
      - field: .kubernetes.namespace_name
        matches: "^open"
    - test:
      - field: .log_type
        matches: "application"
```

```
      - field: .kubernetes.pod_name
        notMatches: "my-pod"
# ...
```

2.5.5.5. Overview of API audit filter

OpenShift API servers generate audit events for each API call, detailing the request, response, and the identity of the requester, leading to large volumes of data. The API Audit filter uses rules to enable the exclusion of non-essential events and the reduction of event size, facilitating a more manageable audit trail. Rules are checked in order, and checking stops at the first match. The amount of data that is included in an event is determined by the value of the **level** field:

- **None**: The event is dropped.
- **Metadata**: Audit metadata is included, request and response bodies are removed.
- **Request**: Audit metadata and the request body are included, the response body is removed.
- **RequestResponse**: All data is included: metadata, request body and response body. The response body can be very large. For example, **oc get pods -A** generates a response body containing the YAML description of every pod in the cluster.

The **ClusterLogForwarder** custom resource (CR) uses the same format as the standard Kubernetes audit policy, while providing the following additional functions:

Wildcards

Names of users, groups, namespaces, and resources can have a leading or trailing * asterisk character. For example, the namespace **openshift-*** matches **openshift-apiserver** or **openshift-authentication**. Resource */**status** matches **Pod**/**status** or **Deployment**/**status**.

Default Rules

Events that do not match any rule in the policy are filtered as follows:

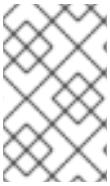
- Read-only system events such as **get**, **list**, and **watch** are dropped.
- Service account write events that occur within the same namespace as the service account are dropped.
- All other events are forwarded, subject to any configured rate limits.

To disable these defaults, either end your rules list with a rule that has only a **level** field or add an empty rule.

Omit Response Codes

A list of integer status codes to omit. You can drop events based on the HTTP status code in the response by using the **OmitResponseCodes** field, which lists HTTP status codes for which no events are created. The default value is **[404, 409, 422, 429]**. If the value is an empty list, **[]**, then no status codes are omitted.

The **ClusterLogForwarder** CR audit policy acts in addition to the OpenShift Container Platform audit policy. The **ClusterLogForwarder** CR audit filter changes what the log collector forwards and provides the ability to filter by verb, user, group, namespace, or resource. You can create multiple filters to send different summaries of the same audit stream to different places. For example, you can send a detailed stream to the local cluster log store and a less detailed stream to a remote site.



NOTE

You must have a cluster role **collect-audit-logs** to collect the audit logs. The following example provided is intended to illustrate the range of rules possible in an audit policy and is not a recommended configuration.

Example audit policy

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name>
  namespace: <log_forwarder_namespace>
spec:
  serviceAccount:
    name: <service_account_name>
  pipelines:
    - name: my-pipeline
      inputRefs: audit (1)
      filterRefs: my-policy (2)
  filters:
    - name: my-policy
      type: kubeAPIAudit
      kubeAPIAudit:
        # Don't generate audit events for all requests in RequestReceived stage.
        omitStages:
          - "RequestReceived"

        rules:
          # Log pod changes at RequestResponse level
          - level: RequestResponse
            resources:
            - group: ""
              resources: ["pods"]

          # Log "pods/log", "pods/status" at Metadata level
          - level: Metadata
            resources:
            - group: ""
              resources: ["pods/log", "pods/status"]

          # Don't log requests to a configmap called "controller-leader"
          - level: None
            resources:
            - group: ""
              resources: ["configmaps"]
              resourceNames: ["controller-leader"]

          # Don't log watch requests by the "system:kube-proxy" on endpoints or services
          - level: None
            users: ["system:kube-proxy"]
            verbs: ["watch"]
            resources:
            - group: "" # core API group
              resources: ["endpoints", "services"]
```

```
    # Don't log authenticated requests to certain non-resource URL paths.
    - level: None
      userGroups: ["system:authenticated"]
      nonResourceURLs:
      - "/api*" # Wildcard matching.
      - "/version"

    # Log the request body of configmap changes in kube-system.
    - level: Request
      resources:
      - group: "" # core API group
        resources: ["configmaps"]
      # This rule only applies to resources in the "kube-system" namespace.
      # The empty string "" can be used to select non-namespaced resources.
      namespaces: ["kube-system"]

    # Log configmap and secret changes in all other namespaces at the Metadata level.
    - level: Metadata
      resources:
      - group: "" # core API group
        resources: ["secrets", "configmaps"]

    # Log all other resources in core and extensions at the Request level.
    - level: Request
      resources:
      - group: "" # core API group
      - group: "extensions" # Version of group should NOT be included.

    # A catch-all rule to log all other requests at the Metadata level.
    - level: Metadata
```

1. The log types that are collected. The value for this field can be **audit** for audit logs, **application** for application logs, **infrastructure** for infrastructure logs, or a named input that has been defined for your application.

2. The name of your audit policy.

2.5.5.6. Filtering application logs at input by including the label expressions or a matching label key and values

You can include the application logs based on the label expressions or a matching label key and its values by using the **input** selector.

Procedure

1. Add a configuration for a filter to the **input** spec in the **ClusterLogForwarder** CR. The following example shows how to configure the **ClusterLogForwarder** CR to include logs based on label expressions or matched label key/values:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
```

```
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  inputs:
    - name: mylogs
      application:
        selector:
          matchExpressions:
          - key: env 1
            operator: In 2
            values: ["prod", "qa"] 3
          - key: zone
            operator: NotIn
            values: ["east", "west"]
          matchLabels: 4
            app: one
            name: app1
      type: application
# ...
```

1. Specifies the label key to match.
2. Specifies the operator. Valid values include: **In**, **NotIn**, **Exists**, and **DoesNotExist**.
3. Specifies an array of string values. If the **operator** value is either **Exists** or **DoesNotExist**, the value array must be empty.
4. Specifies an exact key or value mapping.

2. Apply the **ClusterLogForwarder** CR by running the following command:

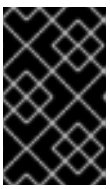
```
$ oc apply -f <filename>.yaml
```

2.5.5.7. Configuring content filters to prune log records

When the **prune** filter is configured, the log collector evaluates log streams according to the filters before forwarding. The collector prunes log records by removing low value fields such as pod annotations.

Procedure

1. Add a configuration for a filter to the **prune** spec in the **ClusterLogForwarder** CR.
   The following example shows how to configure the **ClusterLogForwarder** CR to prune log records based on field paths:



IMPORTANT

If both are specified, records are pruned based on the **notIn** array first, which takes precedence over the **in** array. After records have been pruned by using the **notIn** array, they are then pruned by using the **in** array.

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  filters:
  - name: <filter_name>
    type: prune (1)
    prune: (2)
      in: [.kubernetes.annotations, .kubernetes.namespace_id] (3)
      notIn: [.kubernetes,.log_type,.message,."@timestamp"] (4)
  pipelines:
  - name: <pipeline_name> (5)
    filterRefs: ["<filter_name>"]
# ...
```

1. Specify the type of filter. The **prune** filter prunes log records by configured fields.
2. Specify configuration options for applying the **prune** filter. The **in** and **notIn** fields are specified as arrays of dot-delimited field paths, which are paths to fields in log records. These paths can contain alpha-numeric characters and underscores (**a-zA-Z0-9_**), for example, **.kubernetes.namespace_name**. If segments contain characters outside of this range, the segment must be in quotes, for example, **.kubernetes.labels."foo.bar-bar/baz"**.
3. Optional: Any fields that are specified in this array are removed from the log record.
4. Optional: Any fields that are not specified in this array are removed from the log record.
5. Specify the pipeline that the **prune** filter is applied to.

NOTE

The filters exempts the **log_type**, **.log_source**, and **.message** fields.

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

2.5.6. Filtering the audit and infrastructure log inputs by source

You can define the list of **audit** and **infrastructure** sources to collect the logs by using the **input** selector.

Procedure

1. Add a configuration to define the **audit** and **infrastructure** sources in the **ClusterLogForwarder** CR.
   The following example shows how to configure the **ClusterLogForwarder** CR to define **audit** and **infrastructure** sources:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  inputs:
    - name: mylogs1
      type: infrastructure
      infrastructure:
        sources: 1
          - node
    - name: mylogs2
      type: audit
      audit:
        sources: 2
          - kubeAPI
          - openshiftAPI
          - ovn
# ...
```

1. Specifies the list of infrastructure sources to collect. The valid sources include:
   - **node**: Journal log from the node
   - **container**: Logs from the workloads deployed in the namespaces
2. Specifies the list of audit sources to collect. The valid sources include:
   - **kubeAPI**: Logs from the Kubernetes API servers
   - **openshiftAPI**: Logs from the OpenShift API servers
   - **auditd**: Logs from a node auditd service
   - **ovn**: Logs from an open virtual network service

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

2.5.7. Filtering application logs at input by including or excluding the namespace or container name

You can include or exclude the application logs based on the namespace and container name by using the **input** selector.

Procedure

1. Add a configuration to include or exclude the namespace and container names in the **ClusterLogForwarder** CR.

The following example shows how to configure the **ClusterLogForwarder** CR to include or exclude namespaces and container names:

Example ClusterLogForwarder CR

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
# ...
spec:
  serviceAccount:
    name: <service_account_name>
  inputs:
    - name: mylogs
      application:
        includes:
          - namespace: "my-project" 1
            container: "my-container" 2
        excludes:
          - container: "other-container*" 3
            namespace: "other-namespace" 4
      type: application
# ...
```

1. Specifies that the logs are only collected from these namespaces.
2. Specifies that the logs are only collected from these containers.
3. Specifies the pattern of namespaces to ignore when collecting the logs.
4. Specifies the set of containers to ignore when collecting the logs.

NOTE

The **excludes** field takes precedence over the **includes** field.

2. Apply the **ClusterLogForwarder** CR by running the following command:

```
$ oc apply -f <filename>.yaml
```

2.6. STORING LOGS WITH LOKISTACK

You can configure a **LokiStack** CR to store application, audit, and infrastructure-related logs.

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system offered as a GA log store for logging for Red Hat OpenShift that can be visualized with the OpenShift Observability UI. The Loki configuration provided by OpenShift Logging is a short-term log store designed to enable users to perform fast troubleshooting with the collected logs. For that purpose, the logging for Red Hat OpenShift configuration of Loki has short-term storage, and is optimized for very recent queries. For long-term storage or queries over a long time period, users should look to log stores external to their cluster.

2.6.1. Loki deployment sizing

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

The **1x.pico** configuration defines a single Loki deployment with minimal resource and limit requirements, offering high availability (HA) support for all Loki components. This configuration is suited for deployments that do not require a single replication factor or auto-compaction.

Disk requests are similar across size configurations, allowing customers to test different sizes to determine the best fit for their deployment needs.

IMPORTANT

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

Table 2.2. Loki sizing

	1x.demo	1x.pico [6.1+ only]	1x.extra-small	1x.small	1x.medium
Data transfer	Demo use only	50GB/day	100GB/day	500GB/day	2TB/day
Queries per second (QPS)	Demo use only	1-25 QPS at 200ms	1-25 QPS at 200ms	25-50 QPS at 200ms	25-75 QPS at 200ms
Replication factor	None	2	2	2	2
Total CPU requests	None	7 vCPUs	14 vCPUs	34 vCPUs	54 vCPUs
Total CPU requests if using the ruler	None	8 vCPUs	16 vCPUs	42 vCPUs	70 vCPUs
Total memory requests	None	17Gi	31Gi	67Gi	139Gi
Total memory requests if using the ruler	None	18Gi	35Gi	83Gi	171Gi
Total disk requests	40Gi	590Gi	430Gi	430Gi	590Gi
Total disk requests if using the ruler	60Gi	910Gi	750Gi	750Gi	910Gi

2.6.2. Prerequisites

- You have installed the Loki Operator by using the CLI or web console.

- You have a **serviceAccount** in the same namespace in which you create the **ClusterLogForwarder**.
- The **serviceAccount** is assigned **collect-audit-logs**, **collect-application-logs**, and **collect-infrastructure-logs** cluster roles.

2.6.3. Core Setup and Configuration

Role-based access controls, basic monitoring, and pod placement to deploy Loki.

2.6.4. Authorizing LokiStack rules RBAC permissions

Administrators can allow users to create and manage their own alerting and recording rules by binding cluster roles to usernames. Cluster roles are defined as **ClusterRole** objects that contain necessary role-based access control (RBAC) permissions for users.

The following cluster roles for alerting and recording rules are available for LokiStack:

Rule name	Description
alertingrules.loki.grafana.com-v1-admin	Users with this role have administrative-level access to manage alerting rules. This cluster role grants permissions to create, read, update, delete, list, and watch **AlertingRule** resources within the **loki.grafana.com/v1** API group.
alertingrules.loki.grafana.com-v1-crdview	Users with this role can view the definitions of Custom Resource Definitions (CRDs) related to **AlertingRule** resources within the **loki.grafana.com/v1** API group, but do not have permissions for modifying or managing these resources.
alertingrules.loki.grafana.com-v1-edit	Users with this role have permission to create, update, and delete **AlertingRule** resources.
alertingrules.loki.grafana.com-v1-view	Users with this role can read **AlertingRule** resources within the **loki.grafana.com/v1** API group. They can inspect configurations, labels, and annotations for existing alerting rules but cannot make any modifications to them.
recordingrules.loki.grafana.com-v1-admin	Users with this role have administrative-level access to manage recording rules. This cluster role grants permissions to create, read, update, delete, list, and watch **RecordingRule** resources within the **loki.grafana.com/v1** API group.

Rule name	Description
recordingrules.loki.grafana.com-v1-crdview	Users with this role can view the definitions of Custom Resource Definitions (CRDs) related to **RecordingRule** resources within the **loki.grafana.com/v1** API group, but do not have permissions for modifying or managing these resources.
recordingrules.loki.grafana.com-v1-edit	Users with this role have permission to create, update, and delete **RecordingRule** resources.
recordingrules.loki.grafana.com-v1-view	Users with this role can read **RecordingRule** resources within the **loki.grafana.com/v1** API group. They can inspect configurations, labels, and annotations for existing alerting rules but cannot make any modifications to them.

2.6.4.1. Examples

To apply cluster roles for a user, you must bind an existing cluster role to a specific username.

Cluster roles can be cluster or namespace scoped, depending on which type of role binding you use. When a **RoleBinding** object is used, as when using the **oc adm policy add-role-to-user** command, the cluster role only applies to the specified namespace. When a **ClusterRoleBinding** object is used, as when using the **oc adm policy add-cluster-role-to-user** command, the cluster role applies to all namespaces in the cluster.

The following example command gives the specified user create, read, update and delete (CRUD) permissions for alerting rules in a specific namespace in the cluster:

Example cluster role binding command for alerting rule CRUD permissions in a specific namespace

```
$ oc adm policy add-role-to-user alertingrules.loki.grafana.com-v1-admin -n <namespace>
<username>
```

The following command gives the specified user administrator permissions for alerting rules in all namespaces:

Example cluster role binding command for administrator permissions

```
$ oc adm policy add-cluster-role-to-user alertingrules.loki.grafana.com-v1-admin <username>
```

2.6.5. Creating a log-based alerting rule with Loki

The **AlertingRule** CR contains a set of specifications and webhook validation definitions to declare groups of alerting rules for a single **LokiStack** instance. In addition, the webhook validation definition provides support for rule validation conditions:

- If an **AlertingRule** CR includes an invalid **interval** period, it is an invalid alerting rule
- If an **AlertingRule** CR includes an invalid **for** period, it is an invalid alerting rule.
- If an **AlertingRule** CR includes an invalid LogQL **expr**, it is an invalid alerting rule.
- If an **AlertingRule** CR includes two groups with the same name, it is an invalid alerting rule.
- If none of the above applies, an alerting rule is considered valid.

Table 2.3. AlertingRule definitions

Tenant type	Valid namespaces for **AlertingRule** CRs
application	**<your_application_namespace>**
audit	**openshift-logging**
infrastructure	**openshift-/***, **kube-/***, **default**

Procedure

1. Create an **AlertingRule** custom resource (CR):

Example infrastructure AlertingRule CR

```
apiVersion: loki.grafana.com/v1
kind: AlertingRule
metadata:
  name: loki-operator-alerts
  namespace: openshift-operators-redhat (1)
  labels: (2)
    openshift.io/<label_name>: "true"
spec:
  tenantID: "infrastructure" (3)
  groups:
    - name: LokiOperatorHighReconciliationError
      rules:
        - alert: HighPercentageError
          expr: | (4)
            sum(rate({kubernetes_namespace_name="openshift-operators-redhat",
kubernetes_pod_name=~"loki-operator-controller-manager.*"} |= "error" [1m])) by (job)
              /
            sum(rate({kubernetes_namespace_name="openshift-operators-redhat",
kubernetes_pod_name=~"loki-operator-controller-manager.*"}[1m])) by (job)
              > 0.01
          for: 10s
          labels:
            severity: critical (5)
          annotations:
            summary: High Loki Operator Reconciliation Errors (6)
            description: High Loki Operator Reconciliation Errors (7)
```

1. The namespace where this **AlertingRule** CR is created must have a label matching the LokiStack **spec.rules.namespaceSelector** definition.
2. The **labels** block must match the LokiStack **spec.rules.selector** definition.
3. **AlertingRule** CRs for **infrastructure** tenants are only supported in the **openshift-***, **kube-***, or **default** namespaces.
4. The value for **kubernetes_namespace_name:** must match the value for **metadata.namespace**.
5. The value of this mandatory field must be **critical**, **warning**, or **info**.
6. This field is mandatory.
7. This field is mandatory.

Example application AlertingRule CR

```
apiVersion: loki.grafana.com/v1
kind: AlertingRule
metadata:
  name: app-user-workload
  namespace: app-ns 1
  labels: 2
    openshift.io/<label_name>: "true"
spec:
  tenantID: "application"
  groups:
    - name: AppUserWorkloadHighError
      rules:
        - alert:
          expr: | 3
            sum(rate({kubernetes_namespace_name="app-ns",
kubernetes_pod_name=~"podName.*"} |= "error" [1m])) by (job)
          for: 10s
          labels:
            severity: critical 4
          annotations:
            summary: 5
            description: 6
```

1. The namespace where this **AlertingRule** CR is created must have a label matching the LokiStack **spec.rules.namespaceSelector** definition.
2. The **labels** block must match the LokiStack **spec.rules.selector** definition.
3. Value for **kubernetes_namespace_name:** must match the value for **metadata.namespace**.
4. The value of this mandatory field must be **critical**, **warning**, or **info**.
5. The value of this mandatory field is a summary of the rule.

6 The value of this mandatory field is a detailed description of the rule.

2. Apply the **AlertingRule** CR:

```
$ oc apply -f <filename>.yaml
```

2.6.6. Configuring Loki to tolerate memberlist creation failure

In an OpenShift Container Platform cluster, administrators generally use a non-private IP network range. As a result, the LokiStack memberlist configuration fails because, by default, it only uses private IP networks.

As an administrator, you can select the pod network for the memberlist configuration. You can modify the **LokiStack** custom resource (CR) to use the **podIP** address in the **hashRing** spec. To configure the **LokiStack** CR, use the following command:

```
$ oc patch LokiStack logging-loki -n openshift-logging  --type=merge -p '{"spec": {"hashRing":
{"memberlist":{"instanceAddrType":"podIP"},"type":"memberlist"}}}'
```

Example LokiStack to include podIP

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  hashRing:
    type: memberlist
    memberlist:
      instanceAddrType: podIP
# ...
```

2.6.7. Enabling stream-based retention with Loki

You can configure retention policies based on log streams. Rules for these may be set globally, per-tenant, or both. If you configure both, tenant rules apply before global rules.

IMPORTANT

If there is no retention period defined on the s3 bucket or in the LokiStack custom resource (CR), then the logs are not pruned and they stay in the s3 bucket forever, which might fill up the s3 storage.

NOTE

Schema v13 is recommended.

Procedure

1. Create a **LokiStack** CR:

- Enable stream-based retention globally as shown in the following example:

  Example global stream-based retention for AWS

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
   global: (1)
     retention: (2)
       days: 20
       streams:
       - days: 4
         priority: 1
         selector: '{kubernetes_namespace_name=~"test.+"}' (3)
       - days: 1
         priority: 1
         selector: '{log_type="infrastructure"}'
  managementState: Managed
  replicationFactor: 1
  size: 1x.small
  storage:
    schemas:
    - effectiveDate: "2020-10-11"
      version: v13
    secret:
      name: logging-loki-s3
      type: aws
  storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```

(1) Sets retention policy for all log streams. **Note: This field does not impact the retention period for stored logs in object storage.**

(2) Retention is enabled in the cluster when this block is added to the CR.

(3) Contains the LogQL query used to define the log stream.spec: limits:

- Enable stream-based retention per-tenant basis as shown in the following example:

  Example per-tenant stream-based retention for AWS

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
    global:
```

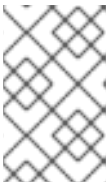
```
    retention:
      days: 20
    tenants: 1
      application:
        retention:
          days: 1
          streams:
            - days: 4
              selector: '{kubernetes_namespace_name=~"test.+"}' 2
      infrastructure:
        retention:
          days: 5
          streams:
            - days: 1
              selector: '{kubernetes_namespace_name=~"openshift-cluster.+"}'
  managementState: Managed
  replicationFactor: 1
  size: 1x.small
  storage:
    schemas:
    - effectiveDate: "2020-10-11"
      version: v13
    secret:
      name: logging-loki-s3
      type: aws
  storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```

1 Sets retention policy by tenant. Valid tenant types are **application**, **audit**, and **infrastructure**.

2 Contains the LogQL query used to define the log stream.

2. Apply the **LokiStack** CR:

```
$ oc apply -f <filename>.yaml
```



NOTE

This is not for managing the retention for stored logs. Global retention periods for stored logs to a supported maximum of 30 days is configured with your object storage.

2.6.8. Loki pod placement

You can control which nodes the Loki pods run on, and prevent other workloads from using those nodes, by using tolerations or node selectors on the pods.

You can apply tolerations to the log store pods with the LokiStack custom resource (CR) and apply taints to a node with the node specification. A taint on a node is a **key:value** pair that instructs the node to repel all pods that do not allow the taint. Using a specific **key:value** pair that is not on other pods ensures that only the log store pods can run on that node.

Example LokiStack with node selectors

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  template:
    compactor: 1
      nodeSelector:
        node-role.kubernetes.io/infra: "" 2
    distributor:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    gateway:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    indexGateway:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    ingester:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    querier:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    queryFrontend:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
    ruler:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
# ...
```

1 Specifies the component pod type that applies to the node selector.

2 Specifies the pods that are moved to nodes containing the defined label.

Example LokiStack CR with node selectors and tolerations

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  template:
    compactor:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
```

```
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
  distributor:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
  indexGateway:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
  ingester:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
  querier:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
  queryFrontend:
    nodeSelector:
      node-role.kubernetes.io/infra: ""
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/infra
      value: reserved
    - effect: NoExecute
      key: node-role.kubernetes.io/infra
      value: reserved
```

```
    ruler:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
    gateway:
      nodeSelector:
        node-role.kubernetes.io/infra: ""
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/infra
        value: reserved
      - effect: NoExecute
        key: node-role.kubernetes.io/infra
        value: reserved
# ...
```

To configure the **nodeSelector** and **tolerations** fields of the LokiStack (CR), you can use the **oc explain** command to view the description and fields for a particular resource:

```
$ oc explain lokistack.spec.template
```

Example output

```
KIND:     LokiStack
VERSION:  loki.grafana.com/v1

RESOURCE: template <Object>

DESCRIPTION:
     Template defines the resource/limits/tolerations/nodeselectors per
     component

FIELDS:
   compactor <Object>
     Compactor defines the compaction component spec.

   distributor <Object>
     Distributor defines the distributor component spec.
...
```

For more detailed information, you can add a specific field:

```
$ oc explain lokistack.spec.template.compactor
```

Example output

```
KIND:     LokiStack
VERSION:  loki.grafana.com/v1
```

```
RESOURCE: compactor <Object>

DESCRIPTION:
    Compactor defines the compaction component spec.

FIELDS:
  nodeSelector <map[string]string>
    NodeSelector defines the labels required by a node to schedule the
    component onto it.
...
```

2.6.9. Enhanced Reliability and Performance

Configurations to ensure Loki's reliability and efficiency in production.

2.6.10. Enabling authentication to cloud-based log stores using short-lived tokens

Workload identity federation enables authentication to cloud-based log stores using short-lived tokens.

Procedure

- Use one of the following options to enable authentication:
  - If you use the OpenShift Container Platform web console to install the Loki Operator, clusters that use short-lived tokens are automatically detected. You are prompted to create roles and supply the data required for the Loki Operator to create a **CredentialsRequest** object, which populates a secret.
  - If you use the OpenShift CLI (**oc**) to install the Loki Operator, you must manually create a **Subscription** object using the appropriate template for your storage provider, as shown in the following examples. This authentication strategy is only supported for the storage providers indicated.

    Example Azure sample subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat
spec:
  channel: "stable-6.0"
  installPlanApproval: Manual
  name: loki-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
      - name: CLIENTID
        value: <your_client_id>
      - name: TENANTID
        value: <your_tenant_id>
      - name: SUBSCRIPTIONID
```

```
      value: <your_subscription_id>
    - name: REGION
      value: <your_region>
```

Example AWS sample subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat
spec:
  channel: "stable-6.0"
  installPlanApproval: Manual
  name: loki-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: ROLEARN
      value: <role_ARN>
```

2.6.11. Configuring Loki to tolerate node failure

The Loki Operator supports setting pod anti-affinity rules to request that pods of the same component are scheduled on different available nodes in the cluster.

Affinity is a property of pods that controls the nodes on which they prefer to be scheduled. Anti-affinity is a property of pods that prevents a pod from being scheduled on a node.

In OpenShift Container Platform, *pod affinity* and *pod anti-affinity* allow you to constrain which nodes your pod is eligible to be scheduled on based on the key-value labels on other pods.

The Operator sets default, preferred **podAntiAffinity** rules for all Loki components, which includes the **compactor**, **distributor**, **gateway**, **indexGateway**, **ingester**, **querier**, **queryFrontend**, and **ruler** components.

You can override the preferred **podAntiAffinity** settings for Loki components by configuring required settings in the **requiredDuringSchedulingIgnoredDuringExecution** field:

Example user settings for the ingester component

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
# ...
  template:
    ingester:
      podAntiAffinity:
      # ...
        requiredDuringSchedulingIgnoredDuringExecution: 1
        - labelSelector:
```

```
          matchLabels: 2
            app.kubernetes.io/component: ingester
          topologyKey: kubernetes.io/hostname
# ...
```

1. The stanza to define a required rule.
2. The key-value pair (label) that must be matched to apply the rule.

2.6.12. LokiStack behavior during cluster restarts

When an OpenShift Container Platform cluster is restarted, LokiStack ingestion and the query path continue to operate within the available CPU and memory resources available for the node. This means that there is no downtime for the LokiStack during OpenShift Container Platform cluster updates. This behavior is achieved by using **PodDisruptionBudget** resources. The Loki Operator provisions **PodDisruptionBudget** resources for Loki, which determine the minimum number of pods that must be available per component to ensure normal operations under certain conditions.

2.6.13. Advanced Deployment and Scalability

Specialized configurations for high availability, scalability, and error handling.

2.6.14. Zone aware data replication

The Loki Operator offers support for zone-aware data replication through pod topology spread constraints. Enabling this feature enhances reliability and safeguards against log loss in the event of a single zone failure. When configuring the deployment size as **1x.extra-small**, **1x.small**, or **1x.medium**, the **replication.factor** field is automatically set to 2.

To ensure proper replication, you need to have at least as many availability zones as the replication factor specifies. While it is possible to have more availability zones than the replication factor, having fewer zones can lead to write failures. Each zone should host an equal number of instances for optimal operation.

Example LokiStack CR with zone replication enabled

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
 name: logging-loki
 namespace: openshift-logging
spec:
 replicationFactor: 2 1
 replication:
  factor: 2 2
  zones:
  - maxSkew: 1 3
    topologyKey: topology.kubernetes.io/zone 4
```

1. Deprecated field, values entered are overwritten by **replication.factor**.
2. This value is automatically set when deployment size is selected at setup.

3 The maximum difference in number of pods between any two topology domains. The default is 1, and you cannot specify a value of 0.

4 Defines zones in the form of a topology key that corresponds to a node label.

2.6.15. Recovering Loki pods from failed zones

In OpenShift Container Platform a zone failure happens when specific availability zone resources become inaccessible. Availability zones are isolated areas within a cloud provider's data center, aimed at enhancing redundancy and fault tolerance. If your OpenShift Container Platform cluster is not configured to handle this, a zone failure can lead to service or data loss.

Loki pods are part of a StatefulSet, and they come with Persistent Volume Claims (PVCs) provisioned by a **StorageClass** object. Each Loki pod and its PVCs reside in the same zone. When a zone failure occurs in a cluster, the StatefulSet controller automatically attempts to recover the affected pods in the failed zone.

WARNING

The following procedure will delete the PVCs in the failed zone, and all data contained therein. To avoid complete data loss the replication factor field of the **LokiStack** CR should always be set to a value greater than 1 to ensure that Loki is replicating.

Prerequisites

- Verify your **LokiStack** CR has a replication factor greater than 1.
- Zone failure detected by the control plane, and nodes in the failed zone are marked by cloud provider integration.

The StatefulSet controller automatically attempts to reschedule pods in a failed zone. Because the associated PVCs are also in the failed zone, automatic rescheduling to a different zone does not work. You must manually delete the PVCs in the failed zone to allow successful re-creation of the stateful Loki Pod and its provisioned PVC in the new zone.

Procedure

1. List the pods in **Pending** status by running the following command:

```
$ oc get pods --field-selector status.phase==Pending -n openshift-logging
```

Example oc get pods output

```
NAME                           READY   STATUS    RESTARTS   AGE 1
logging-loki-index-gateway-1   0/1     Pending   0          17m
logging-loki-ingester-1        0/1     Pending   0          16m
logging-loki-ruler-1           0/1     Pending   0          16m
```

1 These pods are in **Pending** status because their corresponding PVCs are in the failed zone.

2. List the PVCs in **Pending** status by running the following command:

```
$ oc get pvc -o=json -n openshift-logging | jq '.items[] | select(.status.phase == "Pending") | .metadata.name' -r
```

Example oc get pvc output

```
storage-logging-loki-index-gateway-1
storage-logging-loki-ingester-1
wal-logging-loki-ingester-1
storage-logging-loki-ruler-1
wal-logging-loki-ruler-1
```

3. Delete the PVC(s) for a pod by running the following command:

```
$ oc delete pvc <pvc_name> -n openshift-logging
```

4. Delete the pod(s) by running the following command:

```
$ oc delete pod <pod_name> -n openshift-logging
```

Once these objects have been successfully deleted, they should automatically be rescheduled in an available zone.

2.6.15.1. Troubleshooting PVC in a terminating state

The PVCs might hang in the terminating state without being deleted, if PVC metadata finalizers are set to **kubernetes.io/pv-protection**. Removing the finalizers should allow the PVCs to delete successfully.

- Remove the finalizer for each PVC by running the command below, then retry deletion.

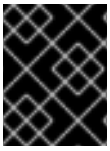
```
$ oc patch pvc <pvc_name> -p '{"metadata":{"finalizers":null}}' -n openshift-logging
```

2.6.16. Troubleshooting Loki rate limit errors

If the Log Forwarder API forwards a large block of messages that exceeds the rate limit to Loki, Loki generates rate limit (**429**) errors.

These errors can occur during normal operation. For example, when adding the logging to a cluster that already has some logs, rate limit errors might occur while the logging tries to ingest all of the existing log entries. In this case, if the rate of addition of new logs is less than the total rate limit, the historical data is eventually ingested, and the rate limit errors are resolved without requiring user intervention.

In cases where the rate limit errors continue to occur, you can fix the issue by modifying the **LokiStack** custom resource (CR).



IMPORTANT

The **LokiStack** CR is not available on Grafana-hosted Loki. This topic does not apply to Grafana-hosted Loki servers.

Conditions

- The Log Forwarder API is configured to forward logs to Loki.
- Your system sends a block of messages that is larger than 2 MB to Loki. For example:

```
"values":[["1630410392689800468","{\"kind\":\"Event\",\"apiVersion\":\
.......
......
......
......
\"received_at\":\"2021-08-31T11:46:32.800278+00:00\",\"version\":\"1.7.4
1.6.0\"}},\"@timestamp\":\"2021-08-
31T11:46:32.799692+00:00\",\"viaq_index_name\":\"audit-
write\",\"viaq_msg_id\":\"MzFjYjJkZjItNjY0MC00YWU4LWIwMTEtNGNmM2E5ZmViMGU4\",\"lo
g_type\":\"audit\"}"]]}]}
```

- After you enter **oc logs -n openshift-logging -l component=collector**, the collector logs in your cluster show a line containing one of the following error messages:

```
429 Too Many Requests Ingestion rate limit exceeded
```

Example Vector error message

```
2023-08-25T16:08:49.301780Z  WARN sink{component_kind="sink"
component_id=default_loki_infra component_type=loki component_name=default_loki_infra}:
vector::sinks::util::retries: Retrying after error. error=Server responded with an error: 429 Too
Many Requests internal_log_rate_limit=true
```

The error is also visible on the receiving end. For example, in the LokiStack ingester pod:

Example Loki ingester error message

```
level=warn ts=2023-08-30T14:57:34.155592243Z caller=grpc_logging.go:43
duration=1.434942ms method=/logproto.Pusher/Push err="rpc error: code = Code(429) desc
= entry with timestamp 2023-08-30 14:57:32.012778399 +0000 UTC ignored, reason: 'Per
stream rate limit exceeded (limit: 3MB/sec) while attempting to ingest for stream
```

Procedure

- Update the **ingestionBurstSize** and **ingestionRate** fields in the **LokiStack** CR:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
```

```
    global:
      ingestion:
        ingestionBurstSize: 16 (1)
        ingestionRate: 8 (2)
# ...
```

(1) The **ingestionBurstSize** field defines the maximum local rate-limited sample size per distributor replica in MB. This value is a hard limit. Set this value to at least the maximum logs size expected in a single push request. Single requests that are larger than the **ingestionBurstSize** value are not permitted.

(2) The **ingestionRate** field is a soft limit on the maximum amount of ingested samples per second in MB. Rate limit errors occur if the rate of logs exceeds the limit, but the collector retries sending the logs. As long as the total average is lower than the limit, the system recovers and errors are resolved without user intervention.

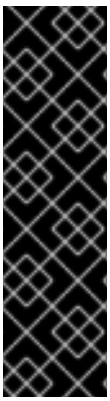
2.7. OTLP DATA INGESTION IN LOKI

You can use an API endpoint by using the OpenTelemetry Protocol (OTLP) with Logging 6.1. As OTLP is a standardized format not specifically designed for Loki, OTLP requires an additional Loki configuration to map data format of OpenTelemetry to data model of Loki. OTLP lacks concepts such as *stream labels* or *structured metadata*. Instead, OTLP provides metadata about log entries as **attributes**, grouped into the following three categories:

- Resource
- Scope
- Log

You can set metadata for multiple entries simultaneously or individually as needed.

2.7.1. Configuring LokiStack for OTLP data ingestion



IMPORTANT

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

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

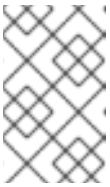
To configure a **LokiStack** custom resource (CR) for OTLP ingestion, follow these steps:

Prerequisites

- Ensure that your Loki setup supports structured metadata, introduced in schema version 13 to enable OTLP log ingestion.

Procedure

1. Set the schema version:
   - When creating a new **LokiStack** CR, set **version: v13** in the storage schema configuration.



NOTE

For existing configurations, add a new schema entry with **version: v13** and an **effectiveDate** in the future. For more information on updating schema versions, see Upgrading Schemas (Grafana documentation).

2. Configure the storage schema as follows:

Example configure storage schema

```
# ...
spec:
  storage:
    schemas:
    - version: v13
      effectiveDate: 2024-10-25
```

Once the **effectiveDate** has passed, the v13 schema takes effect, enabling your **LokiStack** to store structured metadata.

2.7.2. Attribute mapping

When you set the Loki Operator to the **openshift-logging** mode, Loki Operator automatically applies a default set of attribute mappings. These mappings align specific OTLP attributes with stream labels and structured metadata of Loki.

For typical setups, these default mappings are sufficient. However, you might need to customize attribute mapping in the following cases:

- Using a custom collector: If your setup includes a custom collector that generates additional attributes, consider customizing the mapping to ensure these attributes are retained in Loki.
- Adjusting attribute detail levels: If the default attribute set is more detailed than necessary, you can reduce it to essential attributes only. This can avoid excessive data storage and streamline the logging process.

IMPORTANT

Attributes that are not mapped to either stream labels or structured metadata are not stored in Loki.

2.7.2.1. Custom attribute mapping for OpenShift

When using the Loki Operator in **openshift-logging** mode, attribute mapping follow OpenShift default values, but you can configure custom mappings to adjust default values. In the **openshift-logging** mode, you can configure custom attribute mappings globally for all tenants or for individual tenants as needed. When you define custom mappings, they are appended to the OpenShift default values. If you do not need default labels, you can disable them in the tenant configuration.

NOTE

A major difference between the Loki Operator and Loki lies in inheritance handling. Loki copies only **default_resource_attributes_as_index_labels** to tenants by default, while the Loki Operator applies the entire global configuration to each tenant in the **openshift-logging** mode.

Within **LokiStack**, attribute mapping configuration is managed through the **limits** setting. See the following example **LokiStack** configuration:

```
# ...
spec:
  limits:
    global:
      otlp: {} 1
    tenants:
      application:
        otlp: {} 2
```

1 Defines global OTLP attribute configuration.

2 OTLP attribute configuration for the **application** tenant within **openshift-logging** mode.

NOTE

Both global and per-tenant OTLP configurations can map attributes to stream labels or structured metadata. At least one stream label is required to save a log entry to Loki storage, so ensure this configuration meets that requirement.

Stream labels derive only from resource-level attributes, which the **LokiStack** resource structure reflects:

```
spec:
  limits:
    global:
      otlp:
        streamLabels:
          resourceAttributes:
          - name: "k8s.namespace.name"
          - name: "k8s.pod.name"
          - name: "k8s.container.name"
```

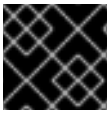
Structured metadata, in contrast, can be generated from resource, scope or log-level attributes:

```
# ...
spec:
  limits:
    global:
      otlp:
        streamLabels:
# ...
        structuredMetadata:
          resourceAttributes:
```

```
      - name: "process.command_line"
      - name: "k8s\\.pod\\.labels\\..+"
        regex: true
    scopeAttributes:
    - name: "service.name"
    logAttributes:
    - name: "http.route"
```

TIP

Use regular expressions by setting **regex: true** for attributes names when mapping similar attributes in Loki.



IMPORTANT

Avoid using regular expressions for stream labels, as this can increase data volume.

2.7.2.2. Customizing OpenShift defaults

In **openshift-logging** mode, certain attributes are required and cannot be removed from the configuration due to their role in OpenShift functions. Other attributes, labeled **recommended**, might be disabled if performance is impacted.

When using the **openshift-logging** mode without custom attributes, you can achieve immediate compatibility with OpenShift tools. If additional attributes are needed as stream labels or structured metadata, use custom configuration. Custom configurations can merge with default configurations.

2.7.2.3. Removing recommended attributes

To reduce default attributes in **openshift-logging** mode, disable recommended attributes:

```
# ...
spec:
  tenants:
    mode: openshift-logging
    openshift:
      otlp:
        disableRecommendedAttributes: true (1)
```

(1) Set **disableRecommendedAttributes: true** to remove recommended attributes, which limits default attributes to the **required attributes**.

NOTE

This option is beneficial if the default attributes causes performance or storage issues. This setting might negatively impact query performance, as it removes default stream labels. You should pair this option with a custom attribute configuration to retain attributes essential for queries.

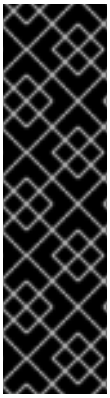
2.7.3. Additional resources

- Loki labels

- Structured metadata
- OpenTelemetry attribute

2.8. OPENTELEMETRY DATA MODEL

This document outlines the protocol and semantic conventions for Red Hat OpenShift Logging's OpenTelemetry support with Logging 6.1.



IMPORTANT

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

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

2.8.1. Forwarding and ingestion protocol

Red Hat OpenShift Logging collects and forwards logs to OpenTelemetry endpoints using OTLP Specification. OTLP encodes, transports, and delivers telemetry data. You can also deploy Loki storage, which provides an OTLP endpont to ingest log streams. This document defines the semantic conventions for the logs collected from various OpenShift cluster sources.

2.8.2. Semantic conventions

The log collector in this solution gathers the following log streams:

- Container logs
- Cluster node journal logs
- Cluster node auditd logs
- Kubernetes and OpenShift API server logs
- OpenShift Virtual Network (OVN) logs

You can forward these streams according to the semantic conventions defined by OpenTelemetry semantic attributes. The semantic conventions in OpenTelemetry define a resource as an immutable representation of the entity producing telemetry, identified by attributes. For example, a process running in a container includes attributes such as **container_name**, **cluster_id**, **pod_name**, **namespace**, and possibly **deployment** or **app_name**. These attributes are grouped under the resource object, which helps reduce repetition and optimizes log transmission as telemetry data.

In addition to resource attributes, logs might also contain scope attributes specific to instrumentation libraries and log attributes specific to each log entry. These attributes provide greater detail about each log entry and enhance filtering capabilities when querying logs in storage.

The following sections define the attributes that are generally forwarded.

2.8.2.1. Log entry structure

All log streams include the following log data fields:

The **Applicable Sources** column indicates which log sources each field applies to:

- **all**: This field is present in all logs.
- **container**: This field is present in Kubernetes container logs, both application and infrastructure.
- **audit**: This field is present in Kubernetes, OpenShift API, and OVN logs.
- **auditd**: This field is present in node auditd logs.
- **journal**: This field is present in node journal logs.

Name	Applicable Sources	Comment
body	all	
observedTimeUnixNano	all	
timeUnixNano	all	
severityText	container, journal	
attributes	all	(Optional) Present when forwarding stream specific attributes

2.8.2.2. Attributes

Log entries include a set of resource, scope, and log attributes based on their source, as described in the following table.

The **Location** column specifies the type of attribute:

- **resource**: Indicates a resource attribute
- **scope**: Indicates a scope attribute
- **log**: Indicates a log attribute

The **Storage** column indicates whether the attribute is stored in a LokiStack using the default **openshift-logging** mode and specifies where the attribute is stored:

- **stream label**:
  - Enables efficient filtering and querying based on specific labels.
  - Can be labeled as **required** if the Loki Operator enforces this attribute in the configuration.
- **structured metadata**:

- Allows for detailed filtering and storage of key-value pairs.
- Enables users to use direct labels for streamlined queries without requiring JSON parsing.

With OTLP, users can filter queries directly by labels rather than using JSON parsing, improving the speed and efficiency of queries.

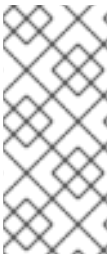
Name	Location	Applicable Sources	Storage (LokiStack)	Comment
log_source	resource	all	required stream label	(DEPRECATED) Compatibility attribute, contains same information as **openshift.log.source**
log_type	resource	all	required stream label	(DEPRECATED) Compatibility attribute, contains same information as **openshift.log.type**
kubernetes.container_name	resource	container	stream label	(DEPRECATED) Compatibility attribute, contains same information as **k8s.container.name**
kubernetes.host	resource	all	stream label	(DEPRECATED) Compatibility attribute, contains same information as **k8s.node.name**
kubernetes.namespace_name	resource	container	required stream label	(DEPRECATED) Compatibility attribute, contains same information as **k8s.namespace.name**

Name	Location	Applicable Sources	Storage (LokiStack)	Comment
kubernetes.pod_name	resource	container	stream label	(DEPRECATED) Compatibility attribute, contains same information as **k8s.pod.name**
openshift.cluster_id	resource	all		(DEPRECATED) Compatibility attribute, contains same information as **openshift.cluster.uid**
level	log	container, journal		(DEPRECATED) Compatibility attribute, contains same information as **severityText**
openshift.cluster.uid	resource	all	required stream label	
openshift.log.source	resource	all	required stream label	
openshift.log.type	resource	all	required stream label	
openshift.labels.*	resource	all	structured metadata	
k8s.node.name	resource	all	stream label	
k8s.namespace.name	resource	container	required stream label	
k8s.container.name	resource	container	stream label	
k8s.pod.labels.*	resource	container	structured metadata	
k8s.pod.name	resource	container	stream label	

Name	Location	Applicable Sources	Storage (LokiStack)	Comment
k8s.pod.uid	resource	container	structured metadata	
k8s.cronjob.name	resource	container	stream label	Conditionally forwarded based on creator of pod
k8s.daemonset.name	resource	container	stream label	Conditionally forwarded based on creator of pod
k8s.deployment.name	resource	container	stream label	Conditionally forwarded based on creator of pod
k8s.job.name	resource	container	stream label	Conditionally forwarded based on creator of pod
k8s.replicaset.name	resource	container	structured metadata	Conditionally forwarded based on creator of pod
k8s.statefulset.name	resource	container	stream label	Conditionally forwarded based on creator of pod
log.iostream	log	container	structured metadata	
k8s.audit.event.level	log	audit	structured metadata	
k8s.audit.event.stage	log	audit	structured metadata	
k8s.audit.event.user_agent	log	audit	structured metadata	
k8s.audit.event.request.uri	log	audit	structured metadata	
k8s.audit.event.response.code	log	audit	structured metadata	

Name	Location	Applicable Sources	Storage (LokiStack)	Comment
k8s.audit.event.annotation.*	log	audit	structured metadata	
k8s.audit.event.object_ref.resource	log	audit	structured metadata	
k8s.audit.event.object_ref.name	log	audit	structured metadata	
k8s.audit.event.object_ref.namespace	log	audit	structured metadata	
k8s.audit.event.object_ref.api_group	log	audit	structured metadata	
k8s.audit.event.object_ref.api_version	log	audit	structured metadata	
k8s.user.username	log	audit	structured metadata	
k8s.user.groups	log	audit	structured metadata	
process.executable.name	resource	journal	structured metadata	
process.executable.path	resource	journal	structured metadata	
process.command_line	resource	journal	structured metadata	
process.pid	resource	journal	structured metadata	
service.name	resource	journal	stream label	
systemd.t.*	log	journal	structured metadata	

Name	Location	Applicable Sources	Storage (LokiStack)	Comment
systemd.u.*	log	journal	structured metadata	



NOTE

Attributes marked as **Compatibility attribute** support minimal backward compatibility with the ViaQ data model. These attributes are deprecated and function as a compatibility layer to ensure continued UI functionality. These attributes will remain supported until the Logging UI fully supports the OpenTelemetry counterparts in future releases.

Loki changes the attribute names when persisting them to storage. The names will be lowercased, and all characters in the set: (**.**,/,**-**) will be replaced by underscores (_). For example, **k8s.namespace.name** will become **k8s_namespace_name**.

2.8.3. Additional resources

- Semantic Conventions
- Logs Data Model
- General Logs Attributes

2.9. VISUALIZATION FOR LOGGING

Visualization for logging is provided by deploying the Logging UI Plugin of the Cluster Observability Operator, which requires Operator installation.