OpenShift Container Platform 4.17

Logging

Configuring and using logging in OpenShift Container Platform

Last Updated: 2024-09-30

OpenShift Container Platform 4.17 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 6.0

1.1. LOGGING 6.0.0

This release includes Logging for Red Hat OpenShift Bug Fix Release 6.0.0

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.

Table 1.1. Upstream component versions

logging Version	Component Version					
Operator	**eventrouter**	**logfilemetricexporter**	**loki**	**lokistack-gateway**	**opa-openshift**	**vector**
6.0	0.4	1.1	3.1.0	0.1	0.1	0.37.1

1.1.1. Removal notice

- With this release, logging no longer supports the **ClusterLogging.logging.openshift.io** and **ClusterLogForwarder.logging.openshift.io** custom resources. Refer to the product documentation for details on the replacement features. (LOG-5803)
- With this release, logging no longer manages or deploys log storage (such as Elasticsearch), visualization (such as Kibana), or Fluentd-based log collectors. (LOG-5368)

NOTE

In order to continue to use Elasticsearch and Kibana managed by the elasticsearch-operator, the administrator must modify those object's ownerRefs before deleting the ClusterLogging resource.

1.1.2. New features and enhancements

- This feature introduces a new architecture for logging for Red Hat OpenShift by shifting component responsibilities to their relevant Operators, such as for storage, visualization, and collection. It introduces the **ClusterLogForwarder.observability.openshift.io** API for log collection and forwarding. Support for the **ClusterLogging.logging.openshift.io** and **ClusterLogForwarder.logging.openshift.io** APIs, along with the Red Hat managed Elastic stack (Elasticsearch and Kibana), is removed. Users are encouraged to migrate to the Red Hat **LokiStack** for log storage. Existing managed Elasticsearch deployments can be used for a limited time. Automated migration for log collection is not provided, so administrators need to create a new ClusterLogForwarder.observability.openshift.io specification to replace their previous custom resources. Refer to the official product documentation for more details. (LOG-3493)
- With this release, the responsibility for deploying the logging view plugin shifts from the Red Hat

OpenShift Logging Operator to the Cluster Observability Operator (COO). For new log storage installations that need visualization, the Cluster Observability Operator and the associated UIPlugin resource must be deployed. Refer to the Cluster Observability Operator Overview product documentation for more details. (LOG-5461)

- This enhancement sets default requests and limits for Vector collector deployments' memory and CPU usage based on Vector documentation recommendations. (LOG-4745)
- This enhancement updates Vector to align with the upstream version v0.37.1. (LOG-5296)
- This enhancement introduces an alert that triggers when log collectors buffer logs to a node's file system and use over 15% of the available space, indicating potential back pressure issues. (LOG-5381)
- This enhancement updates the selectors for all components to use common Kubernetes labels. (LOG-5906)
- This enhancement changes the collector configuration to deploy as a ConfigMap instead of a secret, allowing users to view and edit the configuration when the ClusterLogForwarder is set to Unmanaged. (LOG-5599)
- This enhancement adds the ability to configure the Vector collector log level using an annotation on the ClusterLogForwarder, with options including trace, debug, info, warn, error, or off. (LOG-5372)
- This enhancement adds validation to reject configurations where Amazon CloudWatch outputs use multiple AWS roles, preventing incorrect log routing. (LOG-5640)
- This enhancement removes the Log Bytes Collected and Log Bytes Sent graphs from the metrics dashboard. (LOG-5964)
- This enhancement updates the must-gather functionality to only capture information for inspecting Logging 6.0 components, including Vector deployments from ClusterLogForwarder.observability.openshift.io resources and the Red Hat managed LokiStack. (LOG-5949)
- This enhancement improves Azure storage secret validation by providing early warnings for specific error conditions. (LOG-4571)

1.1.3. Technology Preview features

- This release introduces a Technology Preview feature for log forwarding using OpenTelemetry. A new output type, ` OTLP `, allows sending JSON-encoded log records using the OpenTelemetry data model and resource semantic conventions. (LOG-4225)

1.1.4. Bug fixes

- Before this update, the **CollectorHighErrorRate** and **CollectorVeryHighErrorRate** alerts were still present. With this update, both alerts are removed in the logging 6.0 release but might return in a future release. (LOG-3432)

1.1.5. CVEs

- CVE-2024-34397

1.2. LOGGING 6.0

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

1.2.1. Inputs and Outputs

Inputs specify the sources of logs to be forwarded. Logging provides built-in input types: **application**, **infrastructure**, and **audit**, which select logs from different parts of your cluster. 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.

1.2.2. 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** defines the configuration for a receiver input.

1.2.3. 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. Filters can be used 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.

1.2.4. Operator Behavior

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

- 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.

1.2.5. 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.

1.2.5.1. Quick Start

Prerequisites

- Cluster administrator permissions

Procedure

1. Install the **OpenShift Logging** and **Loki** Operators 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: '2022-06-01'
      version: v13
    secret:
      name: logging-loki-s3
      type: s3
    storageClassName: gp3-csi
  tenants:
    mode: openshift-logging
```

3. Create a service account for the collector:

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

4. Create a **ClusterRole** for the collector:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: logging-collector-logs-writer
rules:
- apiGroups:
  - loki.grafana.com
  resourceNames:
  - logs
  resources:
  - application
  - audit
  - infrastructure
  verbs:
  - create
```

5. Bind the **ClusterRole** to the service account:

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

6. Install the Cluster Observability Operator.
7. Create a **UIPlugin** 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
```

8. Add additional roles to the collector service account:

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

9. 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:
      target:
        name: logging-loki
        namespace: openshift-logging
    authentication:
      token:
        from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - name: default-logstore
    inputRefs:
    - application
    - infrastructure
    outputRefs:
    - default-lokistack
```

10. Verify that logs are visible in the Log section of the Observe tab in the OpenShift web console.

1.3. UPGRADING TO LOGGING 6.0

Logging v6.0 is a significant upgrade from previous releases, achieving several longstanding goals of Cluster Logging:

- Introduction of distinct operators to manage logging components (e.g., collectors, storage, visualization).
- Removal of support for managed log storage and visualization based on Elastic products (i.e., Elasticsearch, Kibana).
- Deprecation of the Fluentd log collector implementation.
- Removal of support for **ClusterLogging.logging.openshift.io** and **ClusterLogForwarder.logging.openshift.io** resources.

NOTE

The **cluster-logging-operator** does not provide an automated upgrade process.

Given the various configurations for log collection, forwarding, and storage, no automated upgrade is provided by the **cluster-logging-operator**. This documentation assists administrators in converting existing **ClusterLogging.logging.openshift.io** and **ClusterLogForwarder.logging.openshift.io** specifications to the new API. Examples of migrated **ClusterLogForwarder.observability.openshift.io** resources for common use cases are included.

1.3.1. Using the oc explain command

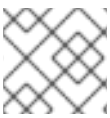
The **oc explain** command is an essential tool in the OpenShift CLI **oc** that provides detailed descriptions of the fields within Custom Resources (CRs). This command is invaluable for administrators and developers who are configuring or troubleshooting resources in an OpenShift cluster.

1.3.1.1. Resource Descriptions

oc explain offers in-depth explanations of all fields associated with a specific object. This includes standard resources like pods and services, as well as more complex entities like statefulsets and custom resources defined by Operators.

To view the documentation for the **outputs** field of the **ClusterLogForwarder** custom resource, you can use:

```
$ oc explain clusterlogforwarders.observability.openshift.io.spec.outputs
```



NOTE

In place of **clusterlogforwarder** the short form **obsclf** can be used.

This will display detailed information about these fields, including their types, default values, and any associated sub-fields.

1.3.1.2. Hierarchical Structure

The command displays the structure of resource fields in a hierarchical format, clarifying the relationships between different configuration options.

For instance, here's how you can drill down into the **storage** configuration for a **LokiStack** custom resource:

```
$ oc explain lokistacks.loki.grafana.com
$ oc explain lokistacks.loki.grafana.com.spec
$ oc explain lokistacks.loki.grafana.com.spec.storage
$ oc explain lokistacks.loki.grafana.com.spec.storage.schemas
```

Each command reveals a deeper level of the resource specification, making the structure clear.

1.3.1.3. Type Information

oc explain also indicates the type of each field (such as string, integer, or boolean), allowing you to verify that resource definitions use the correct data types.

For example:

```
$ oc explain lokistacks.loki.grafana.com.spec.size
```

This will show that **size** should be defined using an integer value.

1.3.1.4. Default Values

When applicable, the command shows the default values for fields, providing insights into what values will be used if none are explicitly specified.

Again using **lokistacks.loki.grafana.com** as an example:

```
$ oc explain lokistacks.spec.template.distributor.replicas
```

Example output

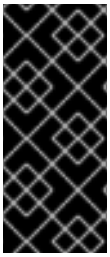
```
GROUP:      loki.grafana.com
KIND:       LokiStack
VERSION:    v1

FIELD: replicas <integer>

DESCRIPTION:
    Replicas defines the number of replica pods of the component.
```

1.3.2. Log Storage

The only managed log storage solution available in this release is a Lokistack, managed by the **loki-operator**. This solution, previously available as the preferred alternative to the managed Elasticsearch offering, remains unchanged in its deployment process.



IMPORTANT

To continue using an existing Red Hat managed Elasticsearch or Kibana deployment provided by the **elasticsearch-operator**, remove the owner references from the **Elasticsearch** resource named **elasticsearch**, and the **Kibana** resource named **kibana** in the **openshift-logging** namespace before removing the **ClusterLogging** resource named **instance** in the same namespace.

1. Temporarily set **ClusterLogging** to state **Unmanaged**

```
$ oc -n openshift-logging patch clusterlogging/instance -p '{"spec":{"managementState":
"Unmanaged"}}' --type=merge
```

2. Remove **ClusterLogging ownerReferences** from the **Elasticsearch** resource
   The following command ensures that **ClusterLogging** no longer owns the **Elasticsearch** resource. Updates to the **ClusterLogging** resource's **logStore** field will no longer affect the **Elasticsearch** resource.

```
$ oc -n openshift-logging patch elasticsearch/elasticsearch -p '{"metadata":
{"ownerReferences": []}}' --type=merge
```

3. Remove **ClusterLogging ownerReferences** from the **Kibana** resource
   The following command ensures that **ClusterLogging** no longer owns the **Kibana** resource. Updates to the **ClusterLogging** resource's **visualization** field will no longer affect the **Kibana** resource.

```
$ oc -n openshift-logging patch kibana/kibana -p '{"metadata":{"ownerReferences": []}}' --
type=merge
```

4. Set **ClusterLogging** to state **Managed**

```
$ oc -n openshift-logging patch clusterlogging/instance -p '{"spec":{"managementState": "Managed"}}'
--type=merge
```

1.3.3. Log Visualization

The OpenShift console UI plugin for log visualization has been moved to the **cluster-observability-operator** from the **cluster-logging-operator**.

1.3.4. Log Collection and Forwarding

Log collection and forwarding configurations are now specified under the new API, part of the **observability.openshift.io** API group. The following sections highlight the differences from the old API resources.

NOTE

Vector is the only supported collector implementation.

1.3.5. Management, Resource Allocation, and Workload Scheduling

Configuration for management state (e.g., Managed, Unmanaged), resource requests and limits, tolerations, and node selection is now part of the new **ClusterLogForwarder** API.

Previous Configuration

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
spec:
  managementState: "Managed"
```

```
  collection:
    resources:
      limits: {}
      requests: {}
    nodeSelector: {}
    tolerations: {}
```

Current Configuration

```
apiVersion: "observability.openshift.io/v1"
kind: ClusterLogForwarder
spec:
  managementState: Managed
  collector:
    resources:
      limits: {}
      requests: {}
    nodeSelector: {}
    tolerations: {}
```

1.3.6. Input Specifications

The input specification is an optional part of the **ClusterLogForwarder** specification. Administrators can continue to use the predefined values of **application**, **infrastructure**, and **audit** to collect these sources.

1.3.6.1. Application Inputs

Namespace and container inclusions and exclusions have been consolidated into a single field.

5.9 Application Input with Namespace and Container Includes and Excludes

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
spec:
  inputs:
   - name: application-logs
     type: application
     application:
       namespaces:
       - foo
       - bar
       includes:
       - namespace: my-important
         container: main
       excludes:
       - container: too-verbose
```

6.0 Application Input with Namespace and Container Includes and Excludes

```
apiVersion: "observability.openshift.io/v1"
kind: ClusterLogForwarder
spec:
  inputs:
```

```
 - name: application-logs
   type: application
   application:
     includes:
     - namespace: foo
     - namespace: bar
     - namespace: my-important
       container: main
     excludes:
     - container: too-verbose
```

NOTE

application, **infrastructure**, and **audit** are reserved words and cannot be used as names when defining an input.

1.3.6.2. Input Receivers

Changes to input receivers include:

- Explicit configuration of the type at the receiver level.
- Port settings moved to the receiver level.

5.9 Input Receivers

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
spec:
  inputs:
  - name: an-http
    receiver:
      http:
        port: 8443
        format: kubeAPIAudit
  - name: a-syslog
    receiver:
      type: syslog
      syslog:
        port: 9442
```

6.0 Input Receivers

```
apiVersion: "observability.openshift.io/v1"
kind: ClusterLogForwarder
spec:
  inputs:
  - name: an-http
    type: receiver
    receiver:
      type: http
      port: 8443
      http:
        format: kubeAPIAudit
```

```
  - name: a-syslog
    type: receiver
    receiver:
      type: syslog
      port: 9442
```

1.3.7. Output Specifications

High-level changes to output specifications include:

- URL settings moved to each output type specification.
- Tuning parameters moved to each output type specification.
- Separation of TLS configuration from authentication.
- Explicit configuration of keys and secret/configmap for TLS and authentication.

1.3.8. Secrets and TLS Configuration

Secrets and TLS configurations are now separated into authentication and TLS configuration for each output. They must be explicitly defined in the specification rather than relying on administrators to define secrets with recognized keys. Upgrading TLS and authorization configurations requires administrators to understand previously recognized keys to continue using existing secrets. Examples in the following sections provide details on how to configure **ClusterLogForwarder** secrets to forward to existing Red Hat managed log storage solutions.

1.3.9. Red Hat Managed Elasticsearch

v5.9 Forwarding to Red Hat Managed Elasticsearch

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
spec:
  logStore:
    type: elasticsearch
```

v6.0 Forwarding to Red Hat Managed Elasticsearch

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
  - name: default-elasticsearch
    type: elasticsearch
    elasticsearch:
      url: https://elasticsearch:9200
      version: 6
```

```
    index: <log_type>-write-{+yyyy.MM.dd}
  tls:
    ca:
      key: ca-bundle.crt
      secretName: collector
    certificate:
      key: tls.crt
      secretName: collector
    key:
      key: tls.key
      secretName: collector
pipelines:
- outputRefs:
  - default-elasticsearch
- inputRefs:
  - application
  - infrastructure
```

NOTE

In this example, application logs are written to the **application-write** alias/index instead of **app-write**.

1.3.10. Red Hat Managed LokiStack

v5.9 Forwarding to Red Hat Managed LokiStack

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
spec:
  logStore:
    type: lokistack
    lokistack:
      name: lokistack-dev
```

v6.0 Forwarding to Red Hat Managed LokiStack

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
  - name: default-lokistack
    type: lokiStack
    lokiStack:
      target:
        name: lokistack-dev
        namespace: openshift-logging
      authentication:
```

```
      token:
        from: serviceAccount
    tls:
      ca:
        key: service-ca.crt
        configMapName: openshift-service-ca.crt
  pipelines:
  - outputRefs:
    - default-lokistack
  - inputRefs:
    - application
    - infrastructure
```

1.3.11. Filters and Pipeline Configuration

Pipeline configurations now define only the routing of input sources to their output destinations, with any required transformations configured separately as filters. All attributes of pipelines from previous releases have been converted to filters in this release. Individual filters are defined in the **filters** specification and referenced by a pipeline.

5.9 Filters

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
spec:
  pipelines:
   - name: application-logs
     parse: json
     labels:
       foo: bar
     detectMultilineErrors: true
```

6.0 Filter Configuration

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
spec:
  filters:
  - name: detectexception
    type: detectMultilineException
  - name: parse-json
    type: parse
  - name: labels
    type: openShiftLabels
    openShiftLabels:
      foo: bar
  pipelines:
  - name: application-logs
    filterRefs:
    - detectexception
    - labels
    - parse-json
```

1.3.12. Validation and Status

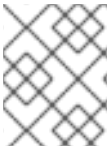
Most validations are enforced when a resource is created or updated, providing immediate feedback. This is a departure from previous releases, where validation occurred post-creation and required inspecting the resource status. Some validation still occurs post-creation for cases where it is not possible to validate at creation or update time.

Instances of the **ClusterLogForwarder.observability.openshift.io** must satisfy the following conditions before the operator will deploy the log collector: Authorized, Valid, Ready. An example of these conditions is:

6.0 Status Conditions

```
apiVersion: observability.openshift.io/v1
kind: ClusterLogForwarder
status:
  conditions:
  - lastTransitionTime: "2024-09-13T03:28:44Z"
    message: 'permitted to collect log types: [application]'
    reason: ClusterRolesExist
    status: "True"
    type: observability.openshift.io/Authorized
  - lastTransitionTime: "2024-09-13T12:16:45Z"
    message: ""
    reason: ValidationSuccess
    status: "True"
    type: observability.openshift.io/Valid
  - lastTransitionTime: "2024-09-13T12:16:45Z"
    message: ""
    reason: ReconciliationComplete
    status: "True"
    type: Ready
  filterConditions:
  - lastTransitionTime: "2024-09-13T13:02:59Z"
    message: filter "detectexception" is valid
    reason: ValidationSuccess
    status: "True"
    type: observability.openshift.io/ValidFilter-detectexception
  - lastTransitionTime: "2024-09-13T13:02:59Z"
    message: filter "parse-json" is valid
    reason: ValidationSuccess
    status: "True"
    type: observability.openshift.io/ValidFilter-parse-json
  inputConditions:
  - lastTransitionTime: "2024-09-13T12:23:03Z"
    message: input "application1" is valid
    reason: ValidationSuccess
    status: "True"
    type: observability.openshift.io/ValidInput-application1
  outputConditions:
  - lastTransitionTime: "2024-09-13T13:02:59Z"
    message: output "default-lokistack-application1" is valid
    reason: ValidationSuccess
    status: "True"
    type: observability.openshift.io/ValidOutput-default-lokistack-application1
  pipelineConditions:
  - lastTransitionTime: "2024-09-13T03:28:44Z"
    message: pipeline "default-before" is valid
```

```
reason: ValidationSuccess
status: "True"
type: observability.openshift.io/ValidPipeline-default-before
```



NOTE

Conditions that are satisfied and applicable have a "status" value of "True". Conditions with a status other than "True" provide a reason and a message explaining the issue.

1.4. 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

1.4.1. 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.

1.4.1.1. 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
```

1.4.1.2. 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>
```

1.4.1.2.1. 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.

1.4.1.2.2. 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
```

Annotations

<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.

1.4.1.2.3. 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 1 rules: Defines the permissions granted by this ClusterRole.

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

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

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

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

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

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

8 8 verbs: The actions allowed on the resources.

9 9 create: Grants permission to create new audit logs.

1.4.1.2.4. 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.

1.4.1.2.5. 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.

1.4.2. 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
# ...
```

1.4.3. 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**.

1.4.4. 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.

1.4.4.1. 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 such as **default**, **openshift**, or any namespace with the **kube-** or **openshift-** prefix.

infrastructure

Selects logs from infrastructure components running in **default** and **openshift** namespaces and node logs.

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.

1.4.4.2. 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.

1.4.4.3. 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.

1.4.4.4. 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:

1.4.4.5. 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>
```

1.4.4.5.1. 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

1.4.4.6. 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"
# ...
```

1.4.4.7. 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.

1.4.4.8. 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:

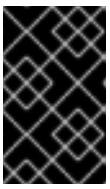
```
$ oc apply -f <filename>.yaml
```

1.4.4.9. 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
```

1.4.5. 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
```

1.4.6. 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
# ...
```

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
```

1.5. STORING LOGS WITH LOKISTACK

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

1.5.1. 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.

1.5.1.1. Core Setup and Configuration

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

1.5.2. 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.
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.

1.5.2.1. 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>
```

1.5.3. 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 1.2. 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
```

1.5.4. 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"}}}'
```

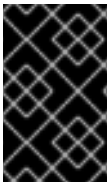
Example LokiStack to include podIP

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

```
  namespace: openshift-logging
spec:
# ...
  hashRing:
    type: memberlist
    memberlist:
      instanceAddrType: podIP
# ...
```

1.5.5. 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

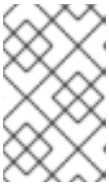
```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki
  namespace: openshift-logging
spec:
  limits:
    global:
      retention:
        days: 20
    tenants: 1
      application:
        retention:
          days: 1
          streams:
            - days: 4
              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.

1.5.6. 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
      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.
...
```

1.5.6.1. Enhanced Reliability and Performance

Configurations to ensure Loki's reliability and efficiency in production.

1.5.7. 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>
```

1.5.8. 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.

1.5.9. 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.

1.5.9.1. Advanced Deployment and Scalability

Specialized configurations for high availability, scalability, and error handling.

1.5.10. 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.

1.5.11. 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.

1.5.11.1. 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
```

1.5.12. 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
```

Example Fluentd error message

```
2023-08-30 14:52:15 +0000 [warn]: [default_loki_infra] failed to flush the buffer. retry_times=2
next_retry_time=2023-08-30 14:52:19 +0000
chunk="604251225bf5378ed1567231a1c03b8b"
error_class=Fluent::Plugin::LokiOutput::LogPostError error="429 Too Many Requests
Ingestion rate limit exceeded for user infrastructure (limit: 4194304 bytes/sec) while
attempting to ingest '4082' lines totaling '7820025' bytes, reduce log volume or contact your
Loki administrator to see if the limit can be increased\n"
```

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.

1.6. VISUALIZATION FOR LOGGING

Visualization for logging is provided by installing the Cluster Observability Operator.