Red Hat OpenShift Service on AWS 4

Logging

OpenShift Logging installation, usage, and release notes

Last Updated: 2024-04-18

Red Hat OpenShift Service on AWS 4 Logging

OpenShift Logging installation, usage, and release notes

Legal Notice

Abstract

This document provides instructions for configuring OpenShift Logging in Red Hat OpenShift Service on AWS.

Table of Contents

CHAPTER 1. RELEASE NOTES

1.1. LOGGING 5.9

NOTE

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

NOTE

The **stable** channel only provides updates to the most recent release of logging. To continue receiving updates for prior releases, you must change your subscription channel to **stable-x.y**, where **x.y** represents the major and minor version of logging you have installed. For example, **stable-5.7**.

1.1.1. Logging 5.9.0

This release includes OpenShift Logging Bug Fix Release 5.9.0

1.1.1.1. Removal notice

The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. Instances of OpenShift Elasticsearch Operator from prior logging releases, remain supported until the EOL of the logging release. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

1.1.1.2. Deprecation notice

- In Logging 5.9, Fluentd, and Kibana are deprecated and are planned to be removed in Logging 6.0, which is expected to be shipped alongside a future release of Red Hat OpenShift Service on AWS. Red Hat will provide critical and above CVE bug fixes and support for these components during the current release lifecycle, but these components will no longer receive feature enhancements. The Vector-based collector provided by the Red Hat OpenShift Logging Operator and LokiStack provided by the Loki Operator are the preferred Operators for log collection and storage. We encourage all users to adopt the Vector and Loki log stack, as this will be the stack that will be enhanced going forward.

- In Logging 5.9, the **Fields** option for the Splunk output type was never implemented and is now deprecated. It will be removed in a future release.

1.1.1.3. Enhancements

1.1.1.3.1. Log Collection

- This enhancement adds the ability to refine the process of log collection by using a workload's metadata to **drop** or **prune** logs based on their content. Additionally, it allows the collection of infrastructure logs, such as journal or container logs, and audit logs, such as **kube api** or **ovn** logs, to only collect individual sources. (LOG-2155)

- This enhancement introduces a new type of remote log receiver, the syslog receiver. You can configure it to expose a port over a network, allowing external systems to send syslog logs using compatible tools such as rsyslog. (LOG-3527)

- With this update, the **ClusterLogForwarder** API now supports log forwarding to Azure Monitor Logs, giving users better monitoring abilities. This feature helps users to maintain optimal system performance and streamline the log analysis processes in Azure Monitor, which speeds up issue resolution and improves operational efficiency. (LOG-4605)

- This enhancement improves collector resource utilization by deploying collectors as a deployment with two replicas. This occurs when the only input source defined in the **ClusterLogForwarder** custom resource (CR) is a receiver input instead of using a daemon set on all nodes. Additionally, collectors deployed in this manner do not mount the host file system. To use this enhancement, you need to annotate the **ClusterLogForwarder** CR with the **logging.openshift.io/dev-preview-enable-collector-as-deployment** annotation. (LOG-4779)

- This enhancement introduces the capability for custom tenant configuration across all supported outputs, facilitating the organization of log records in a logical manner. However, it does not permit custom tenant configuration for logging managed storage. (LOG-4843)

- With this update, the **ClusterLogForwarder** CR that specifies an application input with one or more infrastructure namespaces like **default**, **openshift***, or **kube***, now requires a service account with the **collect-infrastructure-logs** role. (LOG-4943)

- This enhancement introduces the capability for tuning some output settings, such as compression, retry duration, and maximum payloads, to match the characteristics of the receiver. Additionally, this feature includes a delivery mode to allow administrators to choose between throughput and log durability. For example, the **AtLeastOnce** option configures minimal disk buffering of collected logs so that the collector can deliver those logs after a restart. (LOG-5026)

- This enhancement adds three new Prometheus alerts, warning users about the deprecation of Elasticsearch, Fluentd, and Kibana. (LOG-5055)

1.1.1.3.2. Log Storage

- This enhancement in LokiStack improves support for OTEL by using the new V13 object storage format and enabling automatic stream sharding by default. This also prepares the collector for future enhancements and configurations. (LOG-4538)

- This enhancement introduces support for short-lived token workload identity federation with Azure and AWS log stores for STS enabled Red Hat OpenShift Service on AWS 4.14 and later clusters. Local storage requires the addition of a **CredentialMode: static** annotation under **spec.storage.secret** in the LokiStack CR. (LOG-4540)

- With this update, the validation of the Azure storage secret is now extended to give early warning for certain error conditions. (LOG-4571)

- With this update, Loki now adds upstream and downstream support for GCP workload identity federation mechanism. This allows authenticated and authorized access to the corresponding object storage services. (LOG-4754)

1.1.1.4. Bug Fixes

- Before this update, the logging must-gather could not collect any logs on a FIPS-enabled cluster. With this update, a new **oc** client is available in **cluster-logging-rhel9-operator**, and must-gather works properly on FIPS clusters. (LOG-4403)
- Before this update, the LokiStack ruler pods could not format the IPv6 pod IP in HTTP URLs used for cross-pod communication. This issue caused querying rules and alerts through the Prometheus-compatible API to fail. With this update, the LokiStack ruler pods encapsulate the IPv6 pod IP in square brackets, resolving the problem. Now, querying rules and alerts through the Prometheus-compatible API works just like in IPv4 environments. (LOG-4709)
- Before this fix, the YAML content from the logging must-gather was exported in a single line, making it unreadable. With this update, the YAML white spaces are preserved, ensuring that the file is properly formatted. (LOG-4792)
- Before this update, when the **ClusterLogForwarder** CR was enabled, the Red Hat OpenShift Logging Operator could run into a nil pointer exception when **ClusterLogging.Spec.Collection** was nil. With this update, the issue is now resolved in the Red Hat OpenShift Logging Operator. (LOG-5006)
- Before this update, in specific corner cases, replacing the **ClusterLogForwarder** CR status field caused the **resourceVersion** to constantly update due to changing timestamps in **Status** conditions. This condition led to an infinite reconciliation loop. With this update, all status conditions synchronize, so that timestamps remain unchanged if conditions stay the same. (LOG-5007)
- Before this update, there was an internal buffering behavior to **drop_newest** to address high memory consumption by the collector resulting in significant log loss. With this update, the behavior reverts to using the collector defaults. (LOG-5123)
- Before this update, the Loki Operator **ServiceMonitor** in the **openshift-operators-redhat** namespace used static token and CA files for authentication, causing errors in the Prometheus Operator in the User Workload Monitoring spec on the **ServiceMonitor** configuration. With this update, the Loki Operator **ServiceMonitor** in **openshift-operators-redhat** namespace now references a service account token secret by a **LocalReference** object. This approach allows the User Workload Monitoring spec in the Prometheus Operator to handle the Loki Operator **ServiceMonitor** successfully, enabling Prometheus to scrape the Loki Operator metrics. (LOG-5165)
- Before this update, the configuration of the Loki Operator **ServiceMonitor** could match many Kubernetes services, resulting in the Loki Operator metrics being collected multiple times. With this update, the configuration of **ServiceMonitor** now only matches the dedicated metrics service. (LOG-5212)

1.1.1.5. Known Issues

None.

1.1.1.6. CVEs

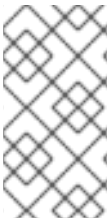
- CVE-2023-5363
- CVE-2023-5981
- CVE-2023-46218
- CVE-2024-0553

- CVE-2023-0567

1.2. LOGGING 5.8

NOTE

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



NOTE

The **stable** channel only provides updates to the most recent release of logging. To continue receiving updates for prior releases, you must change your subscription channel to **stable-x.y**, where **x.y** represents the major and minor version of logging you have installed. For example, **stable-5.7**.

1.2.1. Logging 5.8.5

This release includes OpenShift Logging Bug Fix Release 5.8.5 .

1.2.1.1. Bug fixes

- Before this update, the configuration of the Loki Operator's **ServiceMonitor** could match many Kubernetes services, resulting in the Loki Operator's metrics being collected multiple times. With this update, the configuration of **ServiceMonitor** now only matches the dedicated metrics service. (LOG-5250)

- Before this update, the Red Hat build pipeline did not use the existing build details in Loki builds and omitted information such as revision, branch, and version. With this update, the Red Hat build pipeline now adds these details to the Loki builds, fixing the issue. (LOG-5201)

- Before this update, the Loki Operator checked if the pods were running to decide if the **LokiStack** was ready. With this update, it also checks if the pods are ready, so that the readiness of the **LokiStack** reflects the state of its components. (LOG-5171)

- Before this update, running a query for log metrics caused an error in the histogram. With this update, the histogram toggle function and the chart are disabled and hidden because the histogram doesn't work with log metrics. (LOG-5044)

- Before this update, the Loki and Elasticsearch bundle had the wrong **maxOpenShiftVersion**, resulting in **IncompatibleOperatorsInstalled** alerts. With this update, including 4.16 as the **maxOpenShiftVersion** property in the bundle fixes the issue. (LOG-5272)

- Before this update, the build pipeline did not include linker flags for the build date, causing Loki builds to show empty strings for **buildDate** and **goVersion**. With this update, adding the missing linker flags in the build pipeline fixes the issue. (LOG-5274)

- Before this update, a bug in LogQL parsing left out some line filters from the query. With this update, the parsing now includes all the line filters while keeping the original query unchanged. (LOG-5270)

- Before this update, the Loki Operator **ServiceMonitor** in the **openshift-operators-redhat** namespace used static token and CA files for authentication, causing errors in the Prometheus

Operator in the User Workload Monitoring spec on the **ServiceMonitor** configuration. With this update, the Loki Operator **ServiceMonitor** in **openshift-operators-redhat** namespace now references a service account token secret by a **LocalReference** object. This approach allows the User Workload Monitoring spec in the Prometheus Operator to handle the Loki Operator **ServiceMonitor** successfully, enabling Prometheus to scrape the Loki Operator metrics. (LOG-5240)

1.2.1.2. CVEs

- CVE-2023-5363
- CVE-2023-5981
- CVE-2023-6135
- CVE-2023-46218
- CVE-2023-48795
- CVE-2023-51385
- CVE-2024-0553
- CVE-2024-0567
- CVE-2024-24786
- CVE-2024-28849

1.2.2. Logging 5.8.4

This release includes OpenShift Logging Bug Fix Release 5.8.4 .

1.2.2.1. Bug fixes

- Before this update, the developer console's logs did not account for the current namespace, resulting in query rejection for users without cluster-wide log access. With this update, all supported OCP versions ensure correct namespace inclusion. (LOG-4905)
- Before this update, the Cluster Logging Operator deployed **ClusterRoles** supporting LokiStack deployments only when the default log output was LokiStack. With this update, the roles are split into two groups: read and write. The write roles deploys based on the setting of the default log storage, just like all the roles used to do before. The read roles deploys based on whether the logging console plugin is active. (LOG-4987)
- Before this update, multiple **ClusterLogForwarders** defining the same input receiver name had their service endlessly reconciled because of changing **ownerReferences** on one service. With this update, each receiver input will have its own service named with the convention of **<CLF.Name>-<input.Name>**. (LOG-5009)
- Before this update, the **ClusterLogForwarder** did not report errors when forwarding logs to cloudwatch without a secret. With this update, the following error message appears when forwarding logs to cloudwatch without a secret: **secret must be provided for cloudwatch output**. (LOG-5021)

- Before this update, the **log_forwarder_input_info** included **application**, **infrastructure**, and **audit** input metric points. With this update, **http** is also added as a metric point. (LOG-5043)

1.2.2.2. CVEs

- CVE-2021-35937
- CVE-2021-35938
- CVE-2021-35939
- CVE-2022-3545
- CVE-2022-24963
- CVE-2022-36402
- CVE-2022-41858
- CVE-2023-2166
- CVE-2023-2176
- CVE-2023-3777
- CVE-2023-3812
- CVE-2023-4015
- CVE-2023-4622
- CVE-2023-4623
- CVE-2023-5178
- CVE-2023-5363
- CVE-2023-5388
- CVE-2023-5633
- CVE-2023-6679
- CVE-2023-7104
- CVE-2023-27043
- CVE-2023-38409
- CVE-2023-40283
- CVE-2023-42753
- CVE-2023-43804
- CVE-2023-45803
- CVE-2023-46813

- CVE-2024-20918
- CVE-2024-20919
- CVE-2024-20921
- CVE-2024-20926
- CVE-2024-20945
- CVE-2024-20952

1.2.3. Logging 5.8.3

This release includes Logging Bug Fix 5.8.3 and Logging Security Fix 5.8.3

1.2.3.1. Bug fixes

- Before this update, when configured to read a custom S3 Certificate Authority the Loki Operator would not automatically update the configuration when the name of the ConfigMap or the contents changed. With this update, the Loki Operator is watching for changes to the ConfigMap and automatically updates the generated configuration. (LOG-4969)
- Before this update, Loki outputs configured without a valid URL caused the collector pods to crash. With this update, outputs are subject to URL validation, resolving the issue. (LOG-4822)
- Before this update the Cluster Logging Operator would generate collector configuration fields for outputs that did not specify a secret to use the service account bearer token. With this update, an output does not require authentication, resolving the issue. (LOG-4962)
- Before this update, the **tls.insecureSkipVerify** field of an output was not set to a value of **true** without a secret defined. With this update, a secret is no longer required to set this value. (LOG-4963)
- Before this update, output configurations allowed the combination of an insecure (HTTP) URL with TLS authentication. With this update, outputs configured for TLS authentication require a secure (HTTPS) URL. (LOG-4893)

1.2.3.2. CVEs

- CVE-2021-35937
- CVE-2021-35938
- CVE-2021-35939
- CVE-2023-7104
- CVE-2023-27043
- CVE-2023-48795
- CVE-2023-51385
- CVE-2024-0553

1.2.4. Logging 5.8.2

This release includes OpenShift Logging Bug Fix Release 5.8.2 .

1.2.4.1. Bug fixes

- Before this update, the LokiStack ruler pods would not format the IPv6 pod IP in HTTP URLs used for cross pod communication, causing querying rules and alerts through the Prometheus-compatible API to fail. With this update, the LokiStack ruler pods encapsulate the IPv6 pod IP in square brackets, resolving the issue. (LOG-4890)
- Before this update, the developer console logs did not account for the current namespace, resulting in query rejection for users without cluster-wide log access. With this update, namespace inclusion has been corrected, resolving the issue. (LOG-4947)
- Before this update, the logging view plugin of the Red Hat OpenShift Service on AWS web console did not allow for custom node placement and tolerations. With this update, defining custom node placements and tolerations has been added to the logging view plugin of the Red Hat OpenShift Service on AWS web console. (LOG-4912)

1.2.4.2. CVEs

- CVE-2022-44638
- CVE-2023-1192
- CVE-2023-5345
- CVE-2023-20569
- CVE-2023-26159
- CVE-2023-39615
- CVE-2023-45871

1.2.5. Logging 5.8.1

This release includes OpenShift Logging Bug Fix Release 5.8.1 and OpenShift Logging Bug Fix Release 5.8.1 Kibana.

1.2.5.1. Enhancements

1.2.5.1.1. Log Collection

- With this update, while configuring Vector as a collector, you can add logic to the Red Hat OpenShift Logging Operator to use a token specified in the secret in place of the token associated with the service account. (LOG-4780)
- With this update, the **BoltDB Shipper** Loki dashboards are now renamed to **Index** dashboards. (LOG-4828)

1.2.5.2. Bug fixes

- Before this update, the **ClusterLogForwarder** created empty indices after enabling the parsing of JSON logs, even when the rollover conditions were not met. With this update, the **ClusterLogForwarder** skips the rollover when the **write-index** is empty. (LOG-4452)
- Before this update, the Vector set the **default** log level incorrectly. With this update, the correct log level is set by improving the enhancement of regular expression, or **regexp**, for log level detection. (LOG-4480)
- Before this update, during the process of creating index patterns, the default alias was missing from the initial index in each log output. As a result, Kibana users were unable to create index patterns by using OpenShift Elasticsearch Operator. This update adds the missing aliases to OpenShift Elasticsearch Operator, resolving the issue. Kibana users can now create index patterns that include the **{app,infra,audit}-000001** indexes. (LOG-4683)
- Before this update, Fluentd collector pods were in a **CrashLoopBackOff** state due to binding of the Prometheus server on IPv6 clusters. With this update, the collectors work properly on IPv6 clusters. (LOG-4706)
- Before this update, the Red Hat OpenShift Logging Operator would undergo numerous reconciliations whenever there was a change in the **ClusterLogForwarder**. With this update, the Red Hat OpenShift Logging Operator disregards the status changes in the collector daemonsets that triggered the reconciliations. (LOG-4741)
- Before this update, the Vector log collector pods were stuck in the **CrashLoopBackOff** state on IBM Power machines. With this update, the Vector log collector pods start successfully on IBM Power architecture machines. (LOG-4768)
- Before this update, forwarding with a legacy forwarder to an internal LokiStack would produce SSL certificate errors using Fluentd collector pods. With this update, the log collector service account is used by default for authentication, using the associated token and **ca.crt**. (LOG-4791)
- Before this update, forwarding with a legacy forwarder to an internal LokiStack would produce SSL certificate errors using Vector collector pods. With this update, the log collector service account is used by default for authentication and also using the associated token and **ca.crt**. (LOG-4852)
- Before this fix, IPv6 addresses would not be parsed correctly after evaluating a host or multiple hosts for placeholders. With this update, IPv6 addresses are correctly parsed. (LOG-4811)
- Before this update, it was necessary to create a **ClusterRoleBinding** to collect audit permissions for HTTP receiver inputs. With this update, it is not necessary to create the **ClusterRoleBinding** because the endpoint already depends upon the cluster certificate authority. (LOG-4815)
- Before this update, the Loki Operator did not mount a custom CA bundle to the ruler pods. As a result, during the process to evaluate alerting or recording rules, object storage access failed. With this update, the Loki Operator mounts the custom CA bundle to all ruler pods. The ruler pods can download logs from object storage to evaluate alerting or recording rules. (LOG-4836)
- Before this update, while removing the **inputs.receiver** section in the **ClusterLogForwarder**, the HTTP input services and its associated secrets were not deleted. With this update, the HTTP input resources are deleted when not needed. (LOG-4612)
- Before this update, the **ClusterLogForwarder** indicated validation errors in the status, but the outputs and the pipeline status did not accurately reflect the specific issues. With this update,

the pipeline status displays the validation failure reasons correctly in case of misconfigured outputs, inputs, or filters. (LOG-4821)

- Before this update, changing a **LogQL** query that used controls such as time range or severity changed the label matcher operator defining it like a regular expression. With this update, regular expression operators remain unchanged when updating the query. (LOG-4841)

1.2.5.3. CVEs

- CVE-2007-4559
- CVE-2021-3468
- CVE-2021-3502
- CVE-2021-3826
- CVE-2021-43618
- CVE-2022-3523
- CVE-2022-3565
- CVE-2022-3594
- CVE-2022-4285
- CVE-2022-38457
- CVE-2022-40133
- CVE-2022-40982
- CVE-2022-41862
- CVE-2022-42895
- CVE-2023-0597
- CVE-2023-1073
- CVE-2023-1074
- CVE-2023-1075
- CVE-2023-1076
- CVE-2023-1079
- CVE-2023-1206
- CVE-2023-1249
- CVE-2023-1252
- CVE-2023-1652
- CVE-2023-1855

- CVE-2023-1981
- CVE-2023-1989
- CVE-2023-2731
- CVE-2023-3138
- CVE-2023-3141
- CVE-2023-3161
- CVE-2023-3212
- CVE-2023-3268
- CVE-2023-3316
- CVE-2023-3358
- CVE-2023-3576
- CVE-2023-3609
- CVE-2023-3772
- CVE-2023-3773
- CVE-2023-4016
- CVE-2023-4128
- CVE-2023-4155
- CVE-2023-4194
- CVE-2023-4206
- CVE-2023-4207
- CVE-2023-4208
- CVE-2023-4273
- CVE-2023-4641
- CVE-2023-22745
- CVE-2023-26545
- CVE-2023-26965
- CVE-2023-26966
- CVE-2023-27522
- CVE-2023-29491

- CVE-2023-29499
- CVE-2023-30456
- CVE-2023-31486
- CVE-2023-32324
- CVE-2023-32573
- CVE-2023-32611
- CVE-2023-32665
- CVE-2023-33203
- CVE-2023-33285
- CVE-2023-33951
- CVE-2023-33952
- CVE-2023-34241
- CVE-2023-34410
- CVE-2023-35825
- CVE-2023-36054
- CVE-2023-37369
- CVE-2023-38197
- CVE-2023-38545
- CVE-2023-38546
- CVE-2023-39191
- CVE-2023-39975
- CVE-2023-44487

1.2.6. Logging 5.8.0

This release includes OpenShift Logging Bug Fix Release 5.8.0 and OpenShift Logging Bug Fix Release 5.8.0 Kibana.

1.2.6.1. Deprecation notice

In Logging 5.8, Elasticsearch, Fluentd, and Kibana are deprecated and are planned to be removed in Logging 6.0, which is expected to be shipped alongside a future release of Red Hat OpenShift Service on AWS. Red Hat will provide critical and above CVE bug fixes and support for these components during the current release lifecycle, but these components will no longer receive feature enhancements. The Vector-based collector provided by the Red Hat OpenShift Logging Operator and LokiStack

provided by the Loki Operator are the preferred Operators for log collection and storage. We encourage all users to adopt the Vector and Loki log stack, as this will be the stack that will be enhanced going forward.

1.2.6.2. Enhancements

1.2.6.2.1. Log Collection

- With this update, the LogFileMetricExporter is no longer deployed with the collector by default. You must manually create a **LogFileMetricExporter** custom resource (CR) to generate metrics from the logs produced by running containers. If you do not create the **LogFileMetricExporter** CR, you may see a **No datapoints found** message in the Red Hat OpenShift Service on AWS web console dashboard for **Produced Logs**. (LOG-3819)

- With this update, you can deploy multiple, isolated, and RBAC-protected **ClusterLogForwarder** custom resource (CR) instances in any namespace. This allows independent groups to forward desired logs to any destination while isolating their configuration from other collector deployments. (LOG-1343)

IMPORTANT

In order to support multi-cluster log forwarding in additional namespaces other than the **openshift-logging** namespace, you must update the Red Hat OpenShift Logging Operator to watch all namespaces. This functionality is supported by default in new Red Hat OpenShift Logging Operator version 5.8 installations.

- With this update, you can use the flow control or rate limiting mechanism to limit the volume of log data that can be collected or forwarded by dropping excess log records. The input limits prevent poorly-performing containers from overloading the Logging and the output limits put a ceiling on the rate of logs shipped to a given data store. (LOG-884)

- With this update, you can configure the log collector to look for HTTP connections and receive logs as an HTTP server, also known as a webhook. (LOG-4562)

- With this update, you can configure audit polices to control which Kubernetes and OpenShift API server events are forwarded by the log collector. (LOG-3982)

1.2.6.2.2. Log Storage

- With this update, LokiStack administrators can have more fine-grained control over who can access which logs by granting access to logs on a namespace basis. (LOG-3841)

- With this update, the Loki Operator introduces **PodDisruptionBudget** configuration on LokiStack deployments to ensure normal operations during Red Hat OpenShift Service on AWS cluster restarts by keeping ingestion and the query path available. (LOG-3839)

- With this update, the reliability of existing LokiStack installations are seamlessly improved by applying a set of default Affinity and Anti-Affinity policies. (LOG-3840)

- With this update, you can manage zone-aware data replication as an administrator in LokiStack, in order to enhance reliability in the event of a zone failure. (LOG-3266)

- With this update, a new supported small-scale LokiStack size of 1x.extra-small is introduced for Red Hat OpenShift Service on AWS clusters hosting a few workloads and smaller ingestion volumes (up to 100GB/day). (LOG-4329)

- With this update, the LokiStack administrator has access to an official Loki dashboard to inspect the storage performance and the health of each component. (LOG-4327)

1.2.6.2.3. Log Console

- With this update, you can enable the Logging Console Plugin when Elasticsearch is the default Log Store. (LOG-3856)

- With this update, Red Hat OpenShift Service on AWS application owners can receive notifications for application log-based alerts on the Red Hat OpenShift Service on AWS web console **Developer** perspective for Red Hat OpenShift Service on AWS version 4.14 and later. (LOG-3548)

1.2.6.3. Known Issues

- Currently, Splunk log forwarding might not work after upgrading to version 5.8 of the Red Hat OpenShift Logging Operator. This issue is caused by transitioning from OpenSSL version 1.1.1 to version 3.0.7. In the newer OpenSSL version, there is a default behavior change, where connections to TLS 1.2 endpoints are rejected if they do not expose the RFC 5746 extension. As a workaround, enable TLS 1.3 support on the TLS terminating load balancer in front of the Splunk HEC (HTTP Event Collector) endpoint. Splunk is a third-party system and this should be configured from the Splunk end.

- Currently, there is a flaw in handling multiplexed streams in the HTTP/2 protocol, where you can repeatedly make a request for a new multiplex stream and immediately send an **RST_STREAM** frame to cancel it. This created extra work for the server set up and tore down the streams, resulting in a denial of service due to server resource consumption. There is currently no workaround for this issue. (LOG-4609)

- Currently, when using FluentD as the collector, the collector pod cannot start on the Red Hat OpenShift Service on AWS IPv6-enabled cluster. The pod logs produce the **fluentd pod [error]: unexpected error error_class=SocketError error="getaddrinfo: Name or service not known** error. There is currently no workaround for this issue. (LOG-4706)

- Currently, the log alert is not available on an IPv6-enabled cluster. There is currently no workaround for this issue. (LOG-4709)

- Currently, **must-gather** cannot gather any logs on a FIPS-enabled cluster, because the required OpenSSL library is not available in the **cluster-logging-rhel9-operator**. There is currently no workaround for this issue. (LOG-4403)

- Currently, when deploying the logging version 5.8 on a FIPS-enabled cluster, the collector pods cannot start and are stuck in **CrashLoopBackOff** status, while using FluentD as a collector. There is currently no workaround for this issue. (LOG-3933)

1.2.6.4. CVEs

- CVE-2023-40217

1.3. LOGGING 5.7

NOTE

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

NOTE

The **stable** channel only provides updates to the most recent release of logging. To continue receiving updates for prior releases, you must change your subscription channel to **stable-x.y**, where **x.y** represents the major and minor version of logging you have installed. For example, **stable-5.7**.

1.3.1. Logging 5.7.8

This release includes OpenShift Logging Bug Fix Release 5.7.8 .

1.3.1.1. Bug fixes

- Before this update, there was a potential conflict when the same name was used for the **outputRefs** and **inputRefs** parameters in the **ClusterLogForwarder** custom resource (CR). As a result, the collector pods entered in a **CrashLoopBackOff** status. With this update, the output labels contain the **OUTPUT_** prefix to ensure a distinction between output labels and pipeline names. (LOG-4383)
- Before this update, while configuring the JSON log parser, if you did not set the **structuredTypeKey** or **structuredTypeName** parameters for the Cluster Logging Operator, no alert would display about an invalid configuration. With this update, the Cluster Logging Operator informs you about the configuration issue. (LOG-4441)
- Before this update, if the **hecToken** key was missing or incorrect in the secret specified for a Splunk output, the validation failed because the Vector forwarded logs to Splunk without a token. With this update, if the **hecToken** key is missing or incorrect, the validation fails with the **A non-empty hecToken entry is required** error message. (LOG-4580)
- Before this update, selecting a date from the **Custom time range** for logs caused an error in the web console. With this update, you can select a date from the time range model in the web console successfully. (LOG-4684)

1.3.1.2. CVEs

- CVE-2023-40217
- CVE-2023-44487

1.3.2. Logging 5.7.7

This release includes OpenShift Logging Bug Fix Release 5.7.7 .

1.3.2.1. Bug fixes

- Before this update, FluentD normalized the logs emitted by the EventRouter differently from Vector. With this update, the Vector produces log records in a consistent format. (LOG-4178)
- Before this update, there was an error in the query used for the **FluentD Buffer Availability**

graph in the metrics dashboard created by the Cluster Logging Operator as it showed the minimum buffer usage. With this update, the graph shows the maximum buffer usage and is now renamed to **FluentD Buffer Usage**. (LOG-4555)

- Before this update, deploying a LokiStack on IPv6-only or dual-stack Red Hat OpenShift Service on AWS clusters caused the LokiStack memberlist registration to fail. As a result, the distributor pods went into a crash loop. With this update, an administrator can enable IPv6 by setting the **lokistack.spec.hashRing.memberlist.enableIPv6:** value to **true**, which resolves the issue. (LOG-4569)
- Before this update, the log collector relied on the default configuration settings for reading the container log lines. As a result, the log collector did not read the rotated files efficiently. With this update, there is an increase in the number of bytes read, which allows the log collector to efficiently process rotated files. (LOG-4575)
- Before this update, the unused metrics in the Event Router caused the container to fail due to excessive memory usage. With this update, there is reduction in the memory usage of the Event Router by removing the unused metrics. (LOG-4686)

1.3.2.2. CVEs

- CVE-2023-0800
- CVE-2023-0801
- CVE-2023-0802
- CVE-2023-0803
- CVE-2023-0804
- CVE-2023-2002
- CVE-2023-3090
- CVE-2023-3390
- CVE-2023-3776
- CVE-2023-4004
- CVE-2023-4527
- CVE-2023-4806
- CVE-2023-4813
- CVE-2023-4863
- CVE-2023-4911
- CVE-2023-5129
- CVE-2023-20593
- CVE-2023-29491

- CVE-2023-30630
- CVE-2023-35001
- CVE-2023-35788

1.3.3. Logging 5.7.6

This release includes OpenShift Logging Bug Fix Release 5.7.6 .

1.3.3.1. Bug fixes

- Before this update, the collector relied on the default configuration settings for reading the container log lines. As a result, the collector did not read the rotated files efficiently. With this update, there is an increase in the number of bytes read, which allows the collector to efficiently process rotated files. (LOG-4501)
- Before this update, when users pasted a URL with predefined filters, some filters did not reflect. With this update, the UI reflects all the filters in the URL. (LOG-4459)
- Before this update, forwarding to Loki using custom labels generated an error when switching from Fluentd to Vector. With this update, the Vector configuration sanitizes labels in the same way as Fluentd to ensure the collector starts and correctly processes labels. (LOG-4460)
- Before this update, the Observability Logs console search field did not accept special characters that it should escape. With this update, it is escaping special characters properly in the query. (LOG-4456)
- Before this update, the following warning message appeared while sending logs to Splunk: **Timestamp was not found.** With this update, the change overrides the name of the log field used to retrieve the Timestamp and sends it to Splunk without warning. (LOG-4413)
- Before this update, the CPU and memory usage of Vector was increasing over time. With this update, the Vector configuration now contains the **expire_metrics_secs=60** setting to limit the lifetime of the metrics and cap the associated CPU usage and memory footprint. (LOG-4171)
- Before this update, the LokiStack gateway cached authorized requests very broadly. As a result, this caused wrong authorization results. With this update, LokiStack gateway caches on a more fine-grained basis which resolves this issue. (LOG-4393)
- Before this update, the Fluentd runtime image included builder tools which were unnecessary at runtime. With this update, the builder tools are removed, resolving the issue. (LOG-4467)

1.3.3.2. CVEs

- CVE-2023-3899
- CVE-2023-4456
- CVE-2023-32360
- CVE-2023-34969

1.3.4. Logging 5.7.4

This release includes OpenShift Logging Bug Fix Release 5.7.4 .

1.3.4.1. Bug fixes

- Before this update, when forwarding logs to CloudWatch, a **namespaceUUID** value was not appended to the **logGroupName** field. With this update, the **namespaceUUID** value is included, so a **logGroupName** in CloudWatch appears as **logGroupName: vectorcw.b443fb9e-bd4c-4b6a-b9d3-c0097f9ed286**. (LOG-2701)
- Before this update, when forwarding logs over HTTP to an off-cluster destination, the Vector collector was unable to authenticate to the cluster-wide HTTP proxy even though correct credentials were provided in the proxy URL. With this update, the Vector log collector can now authenticate to the cluster-wide HTTP proxy. (LOG-3381)
- Before this update, the Operator would fail if the Fluentd collector was configured with Splunk as an output, due to this configuration being unsupported. With this update, configuration validation rejects unsupported outputs, resolving the issue. (LOG-4237)
- Before this update, when the Vector collector was updated an **enabled = true** value in the TLS configuration for AWS Cloudwatch logs and the GCP Stackdriver caused a configuration error. With this update, **enabled = true** value will be removed for these outputs, resolving the issue. (LOG-4242)
- Before this update, the Vector collector occasionally panicked with the following error message in its log: **thread 'vector-worker' panicked at 'all branches are disabled and there is no else branch', src/kubernetes/reflector.rs:26:9**. With this update, the error has been resolved. (LOG-4275)
- Before this update, an issue in the Loki Operator caused the **alert-manager** configuration for the application tenant to disappear if the Operator was configured with additional options for that tenant. With this update, the generated Loki configuration now contains both the custom and the auto-generated configuration. (LOG-4361)
- Before this update, when multiple roles were used to authenticate using STS with AWS Cloudwatch forwarding, a recent update caused the credentials to be non-unique. With this update, multiple combinations of STS roles and static credentials can once again be used to authenticate with AWS Cloudwatch. (LOG-4368)
- Before this update, Loki filtered label values for active streams but did not remove duplicates, making Grafana's Label Browser unusable. With this update, Loki filters out duplicate label values for active streams, resolving the issue. (LOG-4389)
- Pipelines with no **name** field specified in the **ClusterLogForwarder** custom resource (CR) stopped working after upgrading to OpenShift Logging 5.7. With this update, the error has been resolved. (LOG-4120)

1.3.4.2. CVEs

- CVE-2022-25883
- CVE-2023-22796

1.3.5. Logging 5.7.3

This release includes OpenShift Logging Bug Fix Release 5.7.3 .

1.3.5.1. Bug fixes

- Before this update, when viewing logs within the Red Hat OpenShift Service on AWS web console, cached files caused the data to not refresh. With this update the bootstrap files are not cached, resolving the issue. (LOG-4100)
- Before this update, the Loki Operator reset errors in a way that made identifying configuration problems difficult to troubleshoot. With this update, errors persist until the configuration error is resolved. (LOG-4156)
- Before this update, the LokiStack ruler did not restart after changes were made to the **RulerConfig** custom resource (CR). With this update, the Loki Operator restarts the ruler pods after the **RulerConfig** CR is updated. (LOG-4161)
- Before this update, the vector collector terminated unexpectedly when input match label values contained a / character within the **ClusterLogForwarder**. This update resolves the issue by quoting the match label, enabling the collector to start and collect logs. (LOG-4176)
- Before this update, the Loki Operator terminated unexpectedly when a **LokiStack** CR defined tenant limits, but not global limits. With this update, the Loki Operator can process **LokiStack** CRs without global limits, resolving the issue. (LOG-4198)
- Before this update, Fluentd did not send logs to an Elasticsearch cluster when the private key provided was passphrase-protected. With this update, Fluentd properly handles passphrase-protected private keys when establishing a connection with Elasticsearch. (LOG-4258)
- Before this update, clusters with more than 8,000 namespaces caused Elasticsearch to reject queries because the list of namespaces was larger than the **http.max_header_size** setting. With this update, the default value for header size has been increased, resolving the issue. (LOG-4277)
- Before this update, label values containing a / character within the **ClusterLogForwarder** CR would cause the collector to terminate unexpectedly. With this update, slashes are replaced with underscores, resolving the issue. (LOG-4095)
- Before this update, the Cluster Logging Operator terminated unexpectedly when set to an unmanaged state. With this update, a check to ensure that the **ClusterLogging** resource is in the correct Management state before initiating the reconciliation of the **ClusterLogForwarder** CR, resolving the issue. (LOG-4177)
- Before this update, when viewing logs within the Red Hat OpenShift Service on AWS web console, selecting a time range by dragging over the histogram didn't work on the aggregated logs view inside the pod detail. With this update, the time range can be selected by dragging on the histogram in this view. (LOG-4108)
- Before this update, when viewing logs within the Red Hat OpenShift Service on AWS web console, queries longer than 30 seconds timed out. With this update, the timeout value can be configured in the configmap/logging-view-plugin. (LOG-3498)
- Before this update, when viewing logs within the Red Hat OpenShift Service on AWS web console, clicking the **more data available** option loaded more log entries only the first time it was clicked. With this update, more entries are loaded with each click. (OU-188)
- Before this update, when viewing logs within the Red Hat OpenShift Service on AWS web console, clicking the **streaming** option would only display the **streaming logs** message without showing the actual logs. With this update, both the message and the log stream are displayed correctly. (OU-166)

1.3.5.2. CVEs

- CVE-2020-24736
- CVE-2022-48281
- CVE-2023-1667
- CVE-2023-2283
- CVE-2023-24329
- CVE-2023-26115
- CVE-2023-26136
- CVE-2023-26604
- CVE-2023-28466

1.3.6. Logging 5.7.2

This release includes OpenShift Logging Bug Fix Release 5.7.2 .

1.3.6.1. Bug fixes

- Before this update, it was not possible to delete the **openshift-logging** namespace directly due to the presence of a pending finalizer. With this update, the finalizer is no longer utilized, enabling direct deletion of the namespace. (LOG-3316)
- Before this update, the **run.sh** script would display an incorrect **chunk_limit_size** value if it was changed according to the Red Hat OpenShift Service on AWS documentation. However, when setting the **chunk_limit_size** via the environment variable **$BUFFER_SIZE_LIMIT**, the script would show the correct value. With this update, the **run.sh** script now consistently displays the correct **chunk_limit_size** value in both scenarios. (LOG-3330)
- Before this update, the Red Hat OpenShift Service on AWS web console's logging view plugin did not allow for custom node placement or tolerations. This update adds the ability to define node placement and tolerations for the logging view plugin. (LOG-3749)
- Before this update, the Cluster Logging Operator encountered an Unsupported Media Type exception when trying to send logs to DataDog via the Fluentd HTTP Plugin. With this update, users can seamlessly assign the content type for log forwarding by configuring the HTTP header Content-Type. The value provided is automatically assigned to the **content_type** parameter within the plugin, ensuring successful log transmission. (LOG-3784)
- Before this update, when the **detectMultilineErrors** field was set to **true** in the **ClusterLogForwarder** custom resource (CR), PHP multi-line errors were recorded as separate log entries, causing the stack trace to be split across multiple messages. With this update, multi-line error detection for PHP is enabled, ensuring that the entire stack trace is included in a single log message. (LOG-3878)
- Before this update, **ClusterLogForwarder** pipelines containing a space in their name caused the Vector collector pods to continuously crash. With this update, all spaces, dashes (-), and dots (.) in pipeline names are replaced with underscores (_). (LOG-3945)
- Before this update, the **log_forwarder_output** metric did not include the **http** parameter. This update adds the missing parameter to the metric. (LOG-3997)

- Before this update, Fluentd did not identify some multi-line JavaScript client exceptions when they ended with a colon. With this update, the Fluentd buffer name is prefixed with an underscore, resolving the issue. (LOG-4019)

- Before this update, when configuring log forwarding to write to a Kafka output topic which matched a key in the payload, logs dropped due to an error. With this update, Fluentd's buffer name has been prefixed with an underscore, resolving the issue.(LOG-4027)

- Before this update, the LokiStack gateway returned label values for namespaces without applying the access rights of a user. With this update, the LokiStack gateway applies permissions to label value requests, resolving the issue. (LOG-4049)

- Before this update, the Cluster Logging Operator API required a certificate to be provided by a secret when the **tls.insecureSkipVerify** option was set to **true**. With this update, the Cluster Logging Operator API no longer requires a certificate to be provided by a secret in such cases. The following configuration has been added to the Operator's CR:

  ```
  tls.verify_certificate = false
  tls.verify_hostname = false
  ```

  (LOG-3445)

- Before this update, the LokiStack route configuration caused queries running longer than 30 seconds to timeout. With this update, the LokiStack global and per-tenant **queryTimeout** settings affect the route timeout settings, resolving the issue. (LOG-4052)

- Before this update, a prior fix to remove defaulting of the **collection.type** resulted in the Operator no longer honoring the deprecated specs for resource, node selections, and tolerations. This update modifies the Operator behavior to always prefer the **collection.logs** spec over those of **collection**. This varies from previous behavior that allowed using both the preferred fields and deprecated fields but would ignore the deprecated fields when **collection.type** was populated. (LOG-4185)

- Before this update, the Vector log collector did not generate TLS configuration for forwarding logs to multiple Kafka brokers if the broker URLs were not specified in the output. With this update, TLS configuration is generated appropriately for multiple brokers. (LOG-4163)

- Before this update, the option to enable passphrase for log forwarding to Kafka was unavailable. This limitation presented a security risk as it could potentially expose sensitive information. With this update, users now have a seamless option to enable passphrase for log forwarding to Kafka. (LOG-3314)

- Before this update, Vector log collector did not honor the **tlsSecurityProfile** settings for outgoing TLS connections. After this update, Vector handles TLS connection settings appropriately. (LOG-4011)

- Before this update, not all available output types were included in the **log_forwarder_output_info** metrics. With this update, metrics contain Splunk and Google Cloud Logging data which was missing previously. (LOG-4098)

- Before this update, when **follow_inodes** was set to **true**, the Fluentd collector could crash on file rotation. With this update, the **follow_inodes** setting does not crash the collector. (LOG-4151)

- Before this update, the Fluentd collector could incorrectly close files that should be watched because of how those files were tracked. With this update, the tracking parameters have been corrected. (LOG-4149)

- Before this update, forwarding logs with the Vector collector and naming a pipeline in the **ClusterLogForwarder** instance **audit**, **application** or **infrastructure** resulted in collector pods staying in the **CrashLoopBackOff** state with the following error in the collector log:

  ```
  ERROR vector::cli: Configuration error. error=redefinition of table transforms.audit for key transforms.audit
  ```

  After this update, pipeline names no longer clash with reserved input names, and pipelines can be named **audit**, **application** or **infrastructure**. (LOG-4218)

- Before this update, when forwarding logs to a syslog destination with the Vector collector and setting the **addLogSource** flag to **true**, the following extra empty fields were added to the forwarded messages: **namespace_name=**, **container_name=**, and **pod_name=**. With this update, these fields are no longer added to journal logs. (LOG-4219)

- Before this update, when a **structuredTypeKey** was not found, and a **structuredTypeName** was not specified, log messages were still parsed into structured object. With this update, parsing of logs is as expected. (LOG-4220)

1.3.6.2. CVEs

- CVE-2021-26341
- CVE-2021-33655
- CVE-2021-33656
- CVE-2022-1462
- CVE-2022-1679
- CVE-2022-1789
- CVE-2022-2196
- CVE-2022-2663
- CVE-2022-3028
- CVE-2022-3239
- CVE-2022-3522
- CVE-2022-3524
- CVE-2022-3564
- CVE-2022-3566
- CVE-2022-3567
- CVE-2022-3619
- CVE-2022-3623
- CVE-2022-3625

- CVE-2022-3627
- CVE-2022-3628
- CVE-2022-3707
- CVE-2022-3970
- CVE-2022-4129
- CVE-2022-20141
- CVE-2022-25147
- CVE-2022-25265
- CVE-2022-30594
- CVE-2022-36227
- CVE-2022-39188
- CVE-2022-39189
- CVE-2022-41218
- CVE-2022-41674
- CVE-2022-42703
- CVE-2022-42720
- CVE-2022-42721
- CVE-2022-42722
- CVE-2022-43750
- CVE-2022-47929
- CVE-2023-0394
- CVE-2023-0461
- CVE-2023-1195
- CVE-2023-1582
- CVE-2023-2491
- CVE-2023-22490
- CVE-2023-23454
- CVE-2023-23946
- CVE-2023-25652

- CVE-2023-25815
- CVE-2023-27535
- CVE-2023-29007

1.3.7. Logging 5.7.1

This release includes: OpenShift Logging Bug Fix Release 5.7.1 .

1.3.7.1. Bug fixes

- Before this update, the presence of numerous noisy messages within the Cluster Logging Operator pod logs caused reduced log readability, and increased difficulty in identifying important system events. With this update, the issue is resolved by significantly reducing the noisy messages within Cluster Logging Operator pod logs. (LOG-3482)
- Before this update, the API server would reset the value for the **CollectorSpec.Type** field to **vector**, even when the custom resource used a different value. This update removes the default for the **CollectorSpec.Type** field to restore the previous behavior. (LOG-4086)
- Before this update, a time range could not be selected in the Red Hat OpenShift Service on AWS web console by clicking and dragging over the logs histogram. With this update, clicking and dragging can be used to successfully select a time range. (LOG-4501)
- Before this update, clicking on the **Show Resources** link in the Red Hat OpenShift Service on AWS web console did not produce any effect. With this update, the issue is resolved by fixing the functionality of the "Show Resources" link to toggle the display of resources for each log entry. (LOG-3218)

1.3.7.2. CVEs

- CVE-2023-21930
- CVE-2023-21937
- CVE-2023-21938
- CVE-2023-21939
- CVE-2023-21954
- CVE-2023-21967
- CVE-2023-21968
- CVE-2023-28617

1.3.8. Logging 5.7.0

This release includes OpenShift Logging Bug Fix Release 5.7.0 .

1.3.8.1. Enhancements

With this update, you can enable logging to detect multi-line exceptions and reassemble them into a single log entry.

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, with a value of **true**.

1.3.8.2. Known Issues

None.

1.3.8.3. Bug fixes

- Before this update, the **nodeSelector** attribute for the Gateway component of the LokiStack did not impact node scheduling. With this update, the **nodeSelector** attribute works as expected. (LOG-3713)

1.3.8.4. CVEs

- CVE-2023-1999
- CVE-2023-28617

CHAPTER 2. 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 Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS. The Red Hat OpenShift Container Platform Life Cycle Policy outlines release compatibility.

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
- Historical or long term log retention or storage
- A guaranteed log sink
- Secure storage - audit logs are not stored by default

2.1. SUPPORTED API CUSTOM RESOURCE DEFINITIONS

LokiStack development is ongoing. Not all APIs are currently supported.

Table 2.1. Loki API support states

CustomResourceDefinition (CRD)	ApiVersion	Support state
LokiStack	lokistack.loki.grafana.com/v1	Supported in 5.5
RulerConfig	rulerconfig.loki.grafana/v1	Supported in 5.7
AlertingRule	alertingrule.loki.grafana/v1	Supported in 5.7

CustomResourceDefinition (CRD)	ApiVersion	Support state
RecordingRule	recordingrule.loki.grafana/v1	Supported in 5.7

2.2. UNSUPPORTED CONFIGURATIONS

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

- The **Elasticsearch** custom resource (CR)
- The Kibana deployment
- The **fluent.conf** file
- The Fluentd daemon set

You must set the OpenShift Elasticsearch Operator to the **Unmanaged** state to modify the Elasticsearch deployment files.

Explicitly unsupported cases include:

- **Configuring default log rotation**. You cannot modify the default log rotation configuration.
- **Configuring the collected log location** You cannot change the location of the log collector output file, which by default is **/var/log/fluentd/fluentd.log**.
- **Throttling log collection**. You cannot throttle down the rate at which the logs are read in by the log collector.
- **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.

2.3. 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.

2.4. 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 Red Hat OpenShift Service on AWS and logging.

> **NOTE**
>
> Do not use the **hack/logging-dump.sh** script. The script is no longer supported and does not collect data.

2.4.1. 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.

2.4.2. 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 .

CHAPTER 3. TROUBLESHOOTING LOGGING

3.1. VIEWING LOGGING STATUS

You can view the status of the Red Hat OpenShift Logging Operator and other logging components.

3.1.1. Viewing the status of the Red Hat OpenShift Logging Operator

You can view the status of the Red Hat OpenShift Logging Operator.

Prerequisites

- The Red Hat OpenShift Logging Operator and OpenShift Elasticsearch Operator are installed.

Procedure

1. Change to the **openshift-logging** project by running the following command:

```
$ oc project openshift-logging
```

2. Get the **ClusterLogging** instance status by running the following command:

```
$ oc get clusterlogging instance -o yaml
```

Example output

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
# ...
status: (1)
  collection:
    logs:
      fluentdStatus:
        daemonSet: fluentd (2)
        nodes:
          collector-2rhqp: ip-10-0-169-13.ec2.internal
          collector-6fgjh: ip-10-0-165-244.ec2.internal
          collector-6l2ff: ip-10-0-128-218.ec2.internal
          collector-54nx5: ip-10-0-139-30.ec2.internal
          collector-flpnn: ip-10-0-147-228.ec2.internal
          collector-n2frh: ip-10-0-157-45.ec2.internal
        pods:
          failed: []
          notReady: []
          ready:
          - collector-2rhqp
          - collector-54nx5
          - collector-6fgjh
          - collector-6l2ff
          - collector-flpnn
          - collector-n2frh
  logstore: (3)
    elasticsearchStatus:
```

```
    - ShardAllocationEnabled:  all
      cluster:
        activePrimaryShards:    5
        activeShards:           5
        initializingShards:     0
        numDataNodes:           1
        numNodes:               1
        pendingTasks:           0
        relocatingShards:       0
        status:                 green
        unassignedShards:       0
      clusterName:              elasticsearch
      nodeConditions:
        elasticsearch-cdm-mkkdys93-1:
      nodeCount:  1
      pods:
        client:
          failed:
          notReady:
          ready:
          - elasticsearch-cdm-mkkdys93-1-7f7c6-mjm7c
        data:
          failed:
          notReady:
          ready:
          - elasticsearch-cdm-mkkdys93-1-7f7c6-mjm7c
        master:
          failed:
          notReady:
          ready:
          - elasticsearch-cdm-mkkdys93-1-7f7c6-mjm7c
visualization:  4
    kibanaStatus:
    - deployment: kibana
      pods:
        failed: []
        notReady: []
        ready:
        - kibana-7fb4fd4cc9-f2nls
      replicaSets:
      - kibana-7fb4fd4cc9
      replicas: 1
```

1. In the output, the cluster status fields appear in the **status** stanza.
2. Information on the Fluentd pods.
3. Information on the Elasticsearch pods, including Elasticsearch cluster health, **green**, **yellow**, or **red**.
4. Information on the Kibana pods.

3.1.1.1. Example condition messages

The following are examples of some condition messages from the **Status.Nodes** section of the **ClusterLogging** instance.

A status message similar to the following indicates a node has exceeded the configured low watermark and no shard will be allocated to this node:

Example output

```
nodes:
- conditions:
  - lastTransitionTime: 2019-03-15T15:57:22Z
    message: Disk storage usage for node is 27.5gb (36.74%). Shards will be not
      be allocated on this node.
    reason: Disk Watermark Low
    status: "True"
    type: NodeStorage
  deploymentName: example-elasticsearch-clientdatamaster-0-1
  upgradeStatus: {}
```

A status message similar to the following indicates a node has exceeded the configured high watermark and shards will be relocated to other nodes:

Example output

```
nodes:
- conditions:
  - lastTransitionTime: 2019-03-15T16:04:45Z
    message: Disk storage usage for node is 27.5gb (36.74%). Shards will be relocated
      from this node.
    reason: Disk Watermark High
    status: "True"
    type: NodeStorage
  deploymentName: cluster-logging-operator
  upgradeStatus: {}
```

A status message similar to the following indicates the Elasticsearch node selector in the CR does not match any nodes in the cluster:

Example output

```
  Elasticsearch Status:
    Shard Allocation Enabled:  shard allocation unknown
    Cluster:
      Active Primary Shards:  0
      Active Shards:          0
      Initializing Shards:    0
      Num Data Nodes:         0
      Num Nodes:              0
      Pending Tasks:          0
      Relocating Shards:      0
      Status:                 cluster health unknown
      Unassigned Shards:      0
    Cluster Name:             elasticsearch
    Node Conditions:
      elasticsearch-cdm-mkkdys93-1:
```

```
        Last Transition Time:  2019-06-26T03:37:32Z
        Message:               0/5 nodes are available: 5 node(s) didn't match node selector.
        Reason:                Unschedulable
        Status:                True
        Type:                  Unschedulable
      elasticsearch-cdm-mkkdys93-2:
    Node Count:  2
    Pods:
      Client:
        Failed:
        Not Ready:
          elasticsearch-cdm-mkkdys93-1-75dd69dccd-f7f49
          elasticsearch-cdm-mkkdys93-2-67c64f5f4c-n58vl
        Ready:
      Data:
        Failed:
        Not Ready:
          elasticsearch-cdm-mkkdys93-1-75dd69dccd-f7f49
          elasticsearch-cdm-mkkdys93-2-67c64f5f4c-n58vl
        Ready:
      Master:
        Failed:
        Not Ready:
          elasticsearch-cdm-mkkdys93-1-75dd69dccd-f7f49
          elasticsearch-cdm-mkkdys93-2-67c64f5f4c-n58vl
        Ready:
```

A status message similar to the following indicates that the requested PVC could not bind to PV:

Example output

```
    Node Conditions:
      elasticsearch-cdm-mkkdys93-1:
        Last Transition Time:  2019-06-26T03:37:32Z
        Message:               pod has unbound immediate PersistentVolumeClaims (repeated 5 times)
        Reason:                Unschedulable
        Status:                True
        Type:                  Unschedulable
```

A status message similar to the following indicates that the Fluentd pods cannot be scheduled because the node selector did not match any nodes:

Example output

```
  Status:
    Collection:
      Logs:
        Fluentd Status:
          Daemon Set:  fluentd
          Nodes:
          Pods:
            Failed:
            Not Ready:
            Ready:
```

3.1.2. Viewing the status of logging components

You can view the status for a number of logging components.

Prerequisites

- The Red Hat OpenShift Logging Operator and OpenShift Elasticsearch Operator are installed.

Procedure

1. Change to the **openshift-logging** project.

```
$ oc project openshift-logging
```

2. View the status of logging environment:

```
$ oc describe deployment cluster-logging-operator
```

Example output

```
Name:                   cluster-logging-operator

....

Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable

....

Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  62m   deployment-controller  Scaled up replica set cluster-
logging-operator-574b8987df to 1----
```

3. View the status of the logging replica set:

   a. Get the name of a replica set:

Example output

```
$ oc get replicaset
```

Example output

```
NAME                                      DESIRED   CURRENT   READY   AGE
cluster-logging-operator-574b8987df       1         1         1       159m
elasticsearch-cdm-uhr537yu-1-6869694fb    1         1         1       157m
elasticsearch-cdm-uhr537yu-2-857b6d676f   1         1         1       156m
elasticsearch-cdm-uhr537yu-3-5b6fdd8cfd   1         1         1       155m
kibana-5bd5544f87                         1         1         1       157m
```

b. Get the status of the replica set:

```
$ oc describe replicaset cluster-logging-operator-574b8987df
```

Example output

```
Name:          cluster-logging-operator-574b8987df

....

Replicas:      1 current / 1 desired
Pods Status:    1 Running / 0 Waiting / 0 Succeeded / 0 Failed

....

Events:
  Type    Reason            Age   From                   Message
  ----    ------            ----  ----                   -------
  Normal  SuccessfulCreate  66m   replicaset-controller  Created pod: cluster-logging-
operator-574b8987df-qjhqv----
```

3.2. TROUBLESHOOTING LOG FORWARDING

3.2.1. Redeploying Fluentd pods

When you create a **ClusterLogForwarder** custom resource (CR), if the Red Hat OpenShift Logging Operator does not redeploy the Fluentd pods automatically, you can delete the Fluentd pods to force them to redeploy.

Prerequisites

- You have created a **ClusterLogForwarder** custom resource (CR) object.

Procedure

- Delete the Fluentd pods to force them to redeploy by running the following command:

```
$ oc delete pod --selector logging-infra=collector
```

3.2.2. 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.

3.3. TROUBLESHOOTING LOGGING ALERTS

You can use the following procedures to troubleshoot logging alerts on your cluster.

3.3.1. Elasticsearch cluster health status is red

At least one primary shard and its replicas are not allocated to a node. Use the following procedure to troubleshoot this alert.

TIP

Some commands in this documentation reference an Elasticsearch pod by using a **$ES_POD_NAME** shell variable. If you want to copy and paste the commands directly from this documentation, you must set this variable to a value that is valid for your Elasticsearch cluster.

You can list the available Elasticsearch pods by running the following command:

```
$ oc -n openshift-logging get pods -l component=elasticsearch
```

Choose one of the pods listed and set the **$ES_POD_NAME** variable, by running the following command:

```
$ export ES_POD_NAME=<elasticsearch_pod_name>
```

You can now use the **$ES_POD_NAME** variable in commands.

Procedure

1. Check the Elasticsearch cluster health and verify that the cluster **status** is red by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME -- health
```

2. List the nodes that have joined the cluster by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cat/nodes?v
```

3. List the Elasticsearch pods and compare them with the nodes in the command output from the previous step, by running the following command:

```
$ oc -n openshift-logging get pods -l component=elasticsearch
```

4. If some of the Elasticsearch nodes have not joined the cluster, perform the following steps.

   a. Confirm that Elasticsearch has an elected master node by running the following command and observing the output:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cat/master?v
```

   b. Review the pod logs of the elected master node for issues by running the following command and observing the output:

```
$ oc logs <elasticsearch_master_pod_name> -c elasticsearch -n openshift-logging
```

   c. Review the logs of nodes that have not joined the cluster for issues by running the following command and observing the output:

```
$ oc logs <elasticsearch_node_name> -c elasticsearch -n openshift-logging
```

5. If all the nodes have joined the cluster, check if the cluster is in the process of recovering by running the following command and observing the output:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cat/recovery?active_only=true
```

   If there is no command output, the recovery process might be delayed or stalled by pending tasks.

6. Check if there are pending tasks by running the following command and observing the output:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- health | grep number_of_pending_tasks
```

7. If there are pending tasks, monitor their status. If their status changes and indicates that the cluster is recovering, continue waiting. The recovery time varies according to the size of the cluster and other factors. Otherwise, if the status of the pending tasks does not change, this indicates that the recovery has stalled.

8. If it seems like the recovery has stalled, check if the **cluster.routing.allocation.enable** value is set to **none**, by running the following command and observing the output:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cluster/settings?pretty
```

9. If the **cluster.routing.allocation.enable** value is set to **none**, set it to **all**, by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cluster/settings?pretty \
  -X PUT -d '{"persistent": {"cluster.routing.allocation.enable":"all"}}'
```

10. Check if any indices are still red by running the following command and observing the output:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cat/indices?v
```

11. If any indices are still red, try to clear them by performing the following steps.

    a. Clear the cache by running the following command:

    ```
    $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
      -- es_util --query=<elasticsearch_index_name>/_cache/clear?pretty
    ```

    b. Increase the max allocation retries by running the following command:

    ```
    $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
      -- es_util --query=<elasticsearch_index_name>/_settings?pretty \
      -X PUT -d '{"index.allocation.max_retries":10}'
    ```

    c. Delete all the scroll items by running the following command:

    ```
    $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
      -- es_util --query=_search/scroll/_all -X DELETE
    ```

    d. Increase the timeout by running the following command:

    ```
    $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
      -- es_util --query=<elasticsearch_index_name>/_settings?pretty \
      -X PUT -d '{"index.unassigned.node_left.delayed_timeout":"10m"}'
    ```

12. If the preceding steps do not clear the red indices, delete the indices individually.

    a. Identify the red index name by running the following command:

    ```
    $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
      -- es_util --query=_cat/indices?v
    ```

    b. Delete the red index by running the following command:

    ```
    $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
      -- es_util --query=<elasticsearch_red_index_name> -X DELETE
    ```

13. If there are no red indices and the cluster status is red, check for a continuous heavy processing load on a data node.

    a. Check if the Elasticsearch JVM Heap usage is high by running the following command:

    ```
    $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
      -- es_util --query=_nodes/stats?pretty
    ```

    In the command output, review the **node_name.jvm.mem.heap_used_percent** field to determine the JVM Heap usage.

    b. Check for high CPU utilization. For more information about CPU utilitzation, see the Red Hat OpenShift Service on AWS "Reviewing monitoring dashboards" documentation.

Additional resources

- Reviewing monitoring dashboards
- Fix a red or yellow cluster status

3.3.2. Elasticsearch cluster health status is yellow

Replica shards for at least one primary shard are not allocated to nodes. Increase the node count by adjusting the **nodeCount** value in the **ClusterLogging** custom resource (CR).

Additional resources

- Fix a red or yellow cluster status

3.3.3. Elasticsearch node disk low watermark reached

Elasticsearch does not allocate shards to nodes that reach the low watermark.

TIP

Some commands in this documentation reference an Elasticsearch pod by using a **$ES_POD_NAME** shell variable. If you want to copy and paste the commands directly from this documentation, you must set this variable to a value that is valid for your Elasticsearch cluster.

You can list the available Elasticsearch pods by running the following command:

```
$ oc -n openshift-logging get pods -l component=elasticsearch
```

Choose one of the pods listed and set the **$ES_POD_NAME** variable, by running the following command:

```
$ export ES_POD_NAME=<elasticsearch_pod_name>
```

You can now use the **$ES_POD_NAME** variable in commands.

Procedure

1. Identify the node on which Elasticsearch is deployed by running the following command:

```
$ oc -n openshift-logging get po -o wide
```

2. Check if there are unassigned shards by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cluster/health?pretty | grep unassigned_shards
```

3. If there are unassigned shards, check the disk space on each node, by running the following command:

```
$ for pod in `oc -n openshift-logging get po -l component=elasticsearch -o
jsonpath='{.items[*].metadata.name}'`; \
  do echo $pod; oc -n openshift-logging exec -c elasticsearch $pod \
  -- df -h /elasticsearch/persistent; done
```

4. In the command output, check the **Use** column to determine the used disk percentage on that node.

Example output

```
elasticsearch-cdm-kcrsda6l-1-586cc95d4f-h8zq8
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme1n1     19G  522M   19G   3% /elasticsearch/persistent
elasticsearch-cdm-kcrsda6l-2-5b548fc7b-cwwk7
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme2n1     19G  522M   19G   3% /elasticsearch/persistent
elasticsearch-cdm-kcrsda6l-3-5dfc884d99-59tjw
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme3n1     19G  528M   19G   3% /elasticsearch/persistent
```

If the used disk percentage is above 85%, the node has exceeded the low watermark, and shards can no longer be allocated to this node.

5. To check the current **redundancyPolicy**, run the following command:

```
$ oc -n openshift-logging get es elasticsearch \
  -o jsonpath='{.spec.redundancyPolicy}'
```

If you are using a **ClusterLogging** resource on your cluster, run the following command:

```
$ oc -n openshift-logging get cl \
  -o jsonpath='{.items[*].spec.logStore.elasticsearch.redundancyPolicy}'
```

If the cluster **redundancyPolicy** value is higher than the **SingleRedundancy** value, set it to the **SingleRedundancy** value and save this change.

6. If the preceding steps do not fix the issue, delete the old indices.

   a. Check the status of all indices on Elasticsearch by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME -- indices
```

   b. Identify an old index that can be deleted.

c. Delete the index by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=<elasticsearch_index_name> -X DELETE
```

3.3.4. Elasticsearch node disk high watermark reached

Elasticsearch attempts to relocate shards away from a node that has reached the high watermark to a node with low disk usage that has not crossed any watermark threshold limits.

To allocate shards to a particular node, you must free up some space on that node. If increasing the disk space is not possible, try adding a new data node to the cluster, or decrease the total cluster redundancy policy.

TIP

Some commands in this documentation reference an Elasticsearch pod by using a **$ES_POD_NAME** shell variable. If you want to copy and paste the commands directly from this documentation, you must set this variable to a value that is valid for your Elasticsearch cluster.

You can list the available Elasticsearch pods by running the following command:

```
$ oc -n openshift-logging get pods -l component=elasticsearch
```

Choose one of the pods listed and set the **$ES_POD_NAME** variable, by running the following command:

```
$ export ES_POD_NAME=<elasticsearch_pod_name>
```

You can now use the **$ES_POD_NAME** variable in commands.

Procedure

1. Identify the node on which Elasticsearch is deployed by running the following command:

```
$ oc -n openshift-logging get po -o wide
```

2. Check the disk space on each node:

```
$ for pod in `oc -n openshift-logging get po -l component=elasticsearch -o
jsonpath='{.items[*].metadata.name}'`; \
  do echo $pod; oc -n openshift-logging exec -c elasticsearch $pod \
  -- df -h /elasticsearch/persistent; done
```

3. Check if the cluster is rebalancing:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_cluster/health?pretty | grep relocating_shards
```

If the command output shows relocating shards, the high watermark has been exceeded. The default value of the high watermark is 90%.

4. Increase the disk space on all nodes. If increasing the disk space is not possible, try adding a new data node to the cluster, or decrease the total cluster redundancy policy.

5. To check the current **redundancyPolicy**, run the following command:

```
$ oc -n openshift-logging get es elasticsearch \
  -o jsonpath='{.spec.redundancyPolicy}'
```

   If you are using a **ClusterLogging** resource on your cluster, run the following command:

```
$ oc -n openshift-logging get cl \
  -o jsonpath='{.items[*].spec.logStore.elasticsearch.redundancyPolicy}'
```

   If the cluster **redundancyPolicy** value is higher than the **SingleRedundancy** value, set it to the **SingleRedundancy** value and save this change.

6. If the preceding steps do not fix the issue, delete the old indices.

   a. Check the status of all indices on Elasticsearch by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME -- indices
```

   b. Identify an old index that can be deleted.

   c. Delete the index by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=<elasticsearch_index_name> -X DELETE
```

3.3.5. Elasticsearch node disk flood watermark reached

Elasticsearch enforces a read-only index block on every index that has both of these conditions:

- One or more shards are allocated to the node.
- One or more disks exceed the flood stage.

Use the following procedure to troubleshoot this alert.

TIP

Some commands in this documentation reference an Elasticsearch pod by using a **$ES_POD_NAME** shell variable. If you want to copy and paste the commands directly from this documentation, you must set this variable to a value that is valid for your Elasticsearch cluster.

You can list the available Elasticsearch pods by running the following command:

```
$ oc -n openshift-logging get pods -l component=elasticsearch
```

Choose one of the pods listed and set the **$ES_POD_NAME** variable, by running the following command:

```
$ export ES_POD_NAME=<elasticsearch_pod_name>
```

You can now use the **$ES_POD_NAME** variable in commands.

Procedure

1. Get the disk space of the Elasticsearch node:

```
$ for pod in `oc -n openshift-logging get po -l component=elasticsearch -o
jsonpath='{.items[*].metadata.name}'`; \
  do echo $pod; oc -n openshift-logging exec -c elasticsearch $pod \
  -- df -h /elasticsearch/persistent; done
```

2. In the command output, check the **Avail** column to determine the free disk space on that node.

   #### Example output

```
elasticsearch-cdm-kcrsda6l-1-586cc95d4f-h8zq8
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme1n1     19G  522M   19G   3% /elasticsearch/persistent
elasticsearch-cdm-kcrsda6l-2-5b548fc7b-cwwk7
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme2n1     19G  522M   19G   3% /elasticsearch/persistent
elasticsearch-cdm-kcrsda6l-3-5dfc884d99-59tjw
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme3n1     19G  528M   19G   3% /elasticsearch/persistent
```

3. Increase the disk space on all nodes. If increasing the disk space is not possible, try adding a new data node to the cluster, or decrease the total cluster redundancy policy.

4. To check the current **redundancyPolicy**, run the following command:

```
$ oc -n openshift-logging get es elasticsearch \
  -o jsonpath='{.spec.redundancyPolicy}'
```

   If you are using a **ClusterLogging** resource on your cluster, run the following command:

```
$ oc -n openshift-logging get cl \
  -o jsonpath='{.items[*].spec.logStore.elasticsearch.redundancyPolicy}'
```

If the cluster **redundancyPolicy** value is higher than the **SingleRedundancy** value, set it to the **SingleRedundancy** value and save this change.

5. If the preceding steps do not fix the issue, delete the old indices.

   a. Check the status of all indices on Elasticsearch by running the following command:

   ```
   $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME -- indices
   ```

   b. Identify an old index that can be deleted.

   c. Delete the index by running the following command:

   ```
   $ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
     -- es_util --query=<elasticsearch_index_name> -X DELETE
   ```

6. Continue freeing up and monitoring the disk space. After the used disk space drops below 90%, unblock writing to this node by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=_all/_settings?pretty \
  -X PUT -d '{"index.blocks.read_only_allow_delete": null}'
```

3.3.6. Elasticsearch JVM heap usage is high

The Elasticsearch node Java virtual machine (JVM) heap memory used is above 75%. Consider increasing the heap size.

3.3.7. Aggregated logging system CPU is high

System CPU usage on the node is high. Check the CPU of the cluster node. Consider allocating more CPU resources to the node.

3.3.8. Elasticsearch process CPU is high

Elasticsearch process CPU usage on the node is high. Check the CPU of the cluster node. Consider allocating more CPU resources to the node.

3.3.9. Elasticsearch disk space is running low

Elasticsearch is predicted to run out of disk space within the next 6 hours based on current disk usage. Use the following procedure to troubleshoot this alert.

Procedure

1. Get the disk space of the Elasticsearch node:

```
$ for pod in `oc -n openshift-logging get po -l component=elasticsearch -o
jsonpath='{.items[*].metadata.name}'`; \
  do echo $pod; oc -n openshift-logging exec -c elasticsearch $pod \
  -- df -h /elasticsearch/persistent; done
```

2. In the command output, check the **Avail** column to determine the free disk space on that node.

Example output

```
elasticsearch-cdm-kcrsda6l-1-586cc95d4f-h8zq8
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme1n1     19G  522M   19G   3% /elasticsearch/persistent
elasticsearch-cdm-kcrsda6l-2-5b548fc7b-cwwk7
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme2n1     19G  522M   19G   3% /elasticsearch/persistent
elasticsearch-cdm-kcrsda6l-3-5dfc884d99-59tjw
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme3n1     19G  528M   19G   3% /elasticsearch/persistent
```

3. Increase the disk space on all nodes. If increasing the disk space is not possible, try adding a new data node to the cluster, or decrease the total cluster redundancy policy.

4. To check the current **redundancyPolicy**, run the following command:

```
$ oc -n openshift-logging get es elasticsearch -o jsonpath='{.spec.redundancyPolicy}'
```

If you are using a **ClusterLogging** resource on your cluster, run the following command:

```
$ oc -n openshift-logging get cl \
  -o jsonpath='{.items[*].spec.logStore.elasticsearch.redundancyPolicy}'
```

If the cluster **redundancyPolicy** value is higher than the **SingleRedundancy** value, set it to the **SingleRedundancy** value and save this change.

5. If the preceding steps do not fix the issue, delete the old indices.

   a. Check the status of all indices on Elasticsearch by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME -- indices
```

   b. Identify an old index that can be deleted.

   c. Delete the index by running the following command:

```
$ oc exec -n openshift-logging -c elasticsearch $ES_POD_NAME \
  -- es_util --query=<elasticsearch_index_name> -X DELETE
```

Additional resources

- Fix a red or yellow cluster status

3.3.10. Elasticsearch FileDescriptor usage is high

Based on current usage trends, the predicted number of file descriptors on the node is insufficient. Check the value of **max_file_descriptors** for each node as described in the Elasticsearch File Descriptors documentation.

3.4. VIEWING THE STATUS OF THE ELASTICSEARCH LOG STORE

You can view the status of the OpenShift Elasticsearch Operator and for a number of Elasticsearch components.

3.4.1. Viewing the status of the Elasticsearch log store

You can view the status of the Elasticsearch log store.

Prerequisites

- The Red Hat OpenShift Logging Operator and OpenShift Elasticsearch Operator are installed.

Procedure

1. Change to the **openshift-logging** project by running the following command:

```
$ oc project openshift-logging
```

2. To view the status:

   a. Get the name of the Elasticsearch log store instance by running the following command:

```
$ oc get Elasticsearch
```

Example output

```
NAME            AGE
elasticsearch   5h9m
```

   b. Get the Elasticsearch log store status by running the following command:

```
$ oc get Elasticsearch <Elasticsearch-instance> -o yaml
```

For example:

```
$ oc get Elasticsearch elasticsearch -n openshift-logging -o yaml
```

The output includes information similar to the following:

Example output

```
status: (1)
  cluster: (2)
    activePrimaryShards: 30
    activeShards: 60
    initializingShards: 0
    numDataNodes: 3
    numNodes: 3
    pendingTasks: 0
    relocatingShards: 0
    status: green
    unassignedShards: 0
  clusterHealth: ""
  conditions: [] (3)
```

```
nodes: (4)
- deploymentName: elasticsearch-cdm-zjf34ved-1
  upgradeStatus: {}
- deploymentName: elasticsearch-cdm-zjf34ved-2
  upgradeStatus: {}
- deploymentName: elasticsearch-cdm-zjf34ved-3
  upgradeStatus: {}
pods: (5)
  client:
    failed: []
    notReady: []
    ready:
    - elasticsearch-cdm-zjf34ved-1-6d7fbf844f-sn422
    - elasticsearch-cdm-zjf34ved-2-dfbd988bc-qkzjz
    - elasticsearch-cdm-zjf34ved-3-c8f566f7c-t7zkt
  data:
    failed: []
    notReady: []
    ready:
    - elasticsearch-cdm-zjf34ved-1-6d7fbf844f-sn422
    - elasticsearch-cdm-zjf34ved-2-dfbd988bc-qkzjz
    - elasticsearch-cdm-zjf34ved-3-c8f566f7c-t7zkt
  master:
    failed: []
    notReady: []
    ready:
    - elasticsearch-cdm-zjf34ved-1-6d7fbf844f-sn422
    - elasticsearch-cdm-zjf34ved-2-dfbd988bc-qkzjz
    - elasticsearch-cdm-zjf34ved-3-c8f566f7c-t7zkt
shardAllocationEnabled: all
```

1. In the output, the cluster status fields appear in the **status** stanza.

2. The status of the Elasticsearch log store:

   - The number of active primary shards.
   - The number of active shards.
   - The number of shards that are initializing.
   - The number of Elasticsearch log store data nodes.
   - The total number of Elasticsearch log store nodes.
   - The number of pending tasks.
   - The Elasticsearch log store status: **green**, **red**, **yellow**.
   - The number of unassigned shards.

3. Any status conditions, if present. The Elasticsearch log store status indicates the reasons from the scheduler if a pod could not be placed. Any events related to the following conditions are shown:

   - Container Waiting for both the Elasticsearch log store and proxy containers.

- Container Terminated for both the Elasticsearch log store and proxy containers.
- Pod unschedulable. Also, a condition is shown for a number of issues; see **Example condition messages**.

4. The Elasticsearch log store nodes in the cluster, with **upgradeStatus**.
5. The Elasticsearch log store client, data, and master pods in the cluster, listed under **failed**, **notReady**, or **ready** state.

3.4.1.1. Example condition messages

The following are examples of some condition messages from the **Status** section of the Elasticsearch instance.

The following status message indicates that a node has exceeded the configured low watermark, and no shard will be allocated to this node.

```
status:
  nodes:
  - conditions:
    - lastTransitionTime: 2019-03-15T15:57:22Z
      message: Disk storage usage for node is 27.5gb (36.74%). Shards will be not
        be allocated on this node.
      reason: Disk Watermark Low
      status: "True"
      type: NodeStorage
    deploymentName: example-elasticsearch-cdm-0-1
    upgradeStatus: {}
```

The following status message indicates that a node has exceeded the configured high watermark, and shards will be relocated to other nodes.

```
status:
  nodes:
  - conditions:
    - lastTransitionTime: 2019-03-15T16:04:45Z
      message: Disk storage usage for node is 27.5gb (36.74%). Shards will be relocated
        from this node.
      reason: Disk Watermark High
      status: "True"
      type: NodeStorage
    deploymentName: example-elasticsearch-cdm-0-1
    upgradeStatus: {}
```

The following status message indicates that the Elasticsearch log store node selector in the custom resource (CR) does not match any nodes in the cluster:

```
status:
    nodes:
    - conditions:
      - lastTransitionTime: 2019-04-10T02:26:24Z
        message: '0/8 nodes are available: 8 node(s) didn''t match node selector.'
```

```
        reason: Unschedulable
        status: "True"
        type: Unschedulable
```

The following status message indicates that the Elasticsearch log store CR uses a non-existent persistent volume claim (PVC).

```
status:
  nodes:
  - conditions:
    - last Transition Time:  2019-04-10T05:55:51Z
      message:              pod has unbound immediate PersistentVolumeClaims (repeated 5 times)
      reason:               Unschedulable
      status:               True
      type:                 Unschedulable
```

The following status message indicates that your Elasticsearch log store cluster does not have enough nodes to support the redundancy policy.

```
status:
  clusterHealth: ""
  conditions:
  - lastTransitionTime: 2019-04-17T20:01:31Z
    message: Wrong RedundancyPolicy selected. Choose different RedundancyPolicy or
      add more nodes with data roles
    reason: Invalid Settings
    status: "True"
    type: InvalidRedundancy
```

This status message indicates your cluster has too many control plane nodes:

```
status:
  clusterHealth: green
  conditions:
    - lastTransitionTime: '2019-04-17T20:12:34Z'
      message: >-
        Invalid master nodes count. Please ensure there are no more than 3 total
        nodes with master roles
      reason: Invalid Settings
      status: 'True'
      type: InvalidMasters
```

The following status message indicates that Elasticsearch storage does not support the change you tried to make.

For example:

```
status:
  clusterHealth: green
  conditions:
    - lastTransitionTime: "2021-05-07T01:05:13Z"
      message: Changing the storage structure for a custom resource is not supported
      reason: StorageStructureChangeIgnored
      status: 'True'
      type: StorageStructureChangeIgnored
```

The **reason** and **type** fields specify the type of unsupported change:

StorageClassNameChangeIgnored

Unsupported change to the storage class name.

StorageSizeChangeIgnored

Unsupported change the storage size.

StorageStructureChangeIgnored

Unsupported change between ephemeral and persistent storage structures.

> **IMPORTANT**
>
> If you try to configure the **ClusterLogging** CR to switch from ephemeral to persistent storage, the OpenShift Elasticsearch Operator creates a persistent volume claim (PVC) but does not create a persistent volume (PV). To clear the **StorageStructureChangeIgnored** status, you must revert the change to the **ClusterLogging** CR and delete the PVC.

3.4.2. Viewing the status of the log store components

You can view the status for a number of the log store components.

Elasticsearch indices

You can view the status of the Elasticsearch indices.

1. Get the name of an Elasticsearch pod:

```
$ oc get pods --selector component=elasticsearch -o name
```

Example output

```
pod/elasticsearch-cdm-1godmszn-1-6f8495-vp4lw
pod/elasticsearch-cdm-1godmszn-2-5769cf-9ms2n
pod/elasticsearch-cdm-1godmszn-3-f66f7d-zqkz7
```

2. Get the status of the indices:

```
$ oc exec elasticsearch-cdm-4vjor49p-2-6d4d7db474-q2w7z -- indices
```

Example output

```
Defaulting container name to elasticsearch.
Use 'oc describe pod/elasticsearch-cdm-4vjor49p-2-6d4d7db474-q2w7z -n openshift-
logging' to see all of the containers in this pod.

green  open   infra-000002                                      S4QANnf1QP6NgCegfnrnbQ
3   1     119926           0        157            78
green  open   audit-000001                                       8_EQx77iQCSTzFOXtxRqFw
3   1          0           0          0             0
green  open   .security                                        iDjscH7aSUGhIdq0LheLBQ   1
```

```
1          5           0          0            0
green  open   .kibana_-377444158_kubeadmin
yBywZ9GfSrKebz5gWBZbjw   3   1          1           0          0            0
green  open   infra-000001                                    z6Dpe__ORgiopEpW6Yl44A
3   1     871000           0        874           436
green  open   app-000001                                      hIrazQCeSISewG3c2VIvsQ
3   1       2453           0          3            1
green  open   .kibana_1                                     JCitcBMSQxKOvIq6iQW6wg
1   1          0           0          0            0
green  open   .kibana_-1595131456_user1                              gIYFIEGRRe-
ka0W3okS-mQ   3   1          1           0          0            0
```

Log store pods

You can view the status of the pods that host the log store.

1. Get the name of a pod:

```
$ oc get pods --selector component=elasticsearch -o name
```

Example output

```
pod/elasticsearch-cdm-1godmszn-1-6f8495-vp4lw
pod/elasticsearch-cdm-1godmszn-2-5769cf-9ms2n
pod/elasticsearch-cdm-1godmszn-3-f66f7d-zqkz7
```

2. Get the status of a pod:

```
$ oc describe pod elasticsearch-cdm-1godmszn-1-6f8495-vp4lw
```

The output includes the following status information:

Example output

```
....
Status:             Running

....

Containers:
  elasticsearch:
    Container ID:   cri-o://b7d44e0a9ea486e27f47763f5bb4c39dfd2
    State:          Running
      Started:      Mon, 08 Jun 2020 10:17:56 -0400
    Ready:          True
    Restart Count:  0
    Readiness:  exec [/usr/share/elasticsearch/probe/readiness.sh] delay=10s timeout=30s
period=5s #success=1 #failure=3

....

  proxy:
    Container ID:  cri-
o://3f77032abaddbb1652c116278652908dc01860320b8a4e741d06894b2f8f9aa1
    State:          Running
```

```
    Started:      Mon, 08 Jun 2020 10:18:38 -0400
    Ready:          True
    Restart Count:  0

....

Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True

....

Events:          <none>
```

Log storage pod deployment configuration

You can view the status of the log store deployment configuration.

1. Get the name of a deployment configuration:

```
$ oc get deployment --selector component=elasticsearch -o name
```

Example output

```
deployment.extensions/elasticsearch-cdm-1gon-1
deployment.extensions/elasticsearch-cdm-1gon-2
deployment.extensions/elasticsearch-cdm-1gon-3
```

2. Get the deployment configuration status:

```
$ oc describe deployment elasticsearch-cdm-1gon-1
```

The output includes the following status information:

Example output

```
....
  Containers:
   elasticsearch:
    Image:      registry.redhat.io/openshift-logging/elasticsearch6-rhel8
    Readiness:  exec [/usr/share/elasticsearch/probe/readiness.sh] delay=10s timeout=30s
period=5s #success=1 #failure=3

....

Conditions:
  Type           Status   Reason
  ----           ------   ------
  Progressing    Unknown  DeploymentPaused
  Available      True     MinimumReplicasAvailable
```

```
....

Events:         <none>
```

Log store replica set

You can view the status of the log store replica set.

1. Get the name of a replica set:

```
$ oc get replicaSet --selector component=elasticsearch -o name

replicaset.extensions/elasticsearch-cdm-1gon-1-6f8495
replicaset.extensions/elasticsearch-cdm-1gon-2-5769cf
replicaset.extensions/elasticsearch-cdm-1gon-3-f66f7d
```

2. Get the status of the replica set:

```
$ oc describe replicaSet elasticsearch-cdm-1gon-1-6f8495
```

The output includes the following status information:

Example output

```
....
  Containers:
   elasticsearch:
    Image:      registry.redhat.io/openshift-logging/elasticsearch6-
rhel8@sha256:4265742c7cdd85359140e2d7d703e4311b6497eec7676957f455d6908e7b1
c25
    Readiness:  exec [/usr/share/elasticsearch/probe/readiness.sh] delay=10s timeout=30s
period=5s #success=1 #failure=3

....

Events:         <none>
```

3.4.3. Elasticsearch cluster status

A dashboard in the **Observe** section of the OpenShift Cluster Manager displays the status of the Elasticsearch cluster.

To get the status of the OpenShift Elasticsearch cluster, visit the dashboard in the **Observe** section of the OpenShift Cluster Manager at **<cluster_url>/monitoring/dashboards/grafana-dashboard-cluster-logging**.

Elasticsearch status fields

eo_elasticsearch_cr_cluster_management_state

Shows whether the Elasticsearch cluster is in a managed or unmanaged state. For example:

```
eo_elasticsearch_cr_cluster_management_state{state="managed"} 1
eo_elasticsearch_cr_cluster_management_state{state="unmanaged"} 0
```

eo_elasticsearch_cr_restart_total

Shows the number of times the Elasticsearch nodes have restarted for certificate restarts, rolling restarts, or scheduled restarts. For example:

```
eo_elasticsearch_cr_restart_total{reason="cert_restart"} 1
eo_elasticsearch_cr_restart_total{reason="rolling_restart"} 1
eo_elasticsearch_cr_restart_total{reason="scheduled_restart"} 3
```

es_index_namespaces_total

Shows the total number of Elasticsearch index namespaces. For example:

```
Total number of Namespaces.
es_index_namespaces_total 5
```

es_index_document_count

Shows the number of records for each namespace. For example:

```
es_index_document_count{namespace="namespace_1"} 25
es_index_document_count{namespace="namespace_2"} 10
es_index_document_count{namespace="namespace_3"} 5
```

The "Secret Elasticsearch fields are either missing or empty" message

If Elasticsearch is missing the **admin-cert**, **admin-key**, **logging-es.crt**, or **logging-es.key** files, the dashboard shows a status message similar to the following example:

```
message": "Secret \"elasticsearch\" fields are either missing or empty: [admin-cert, admin-key,
logging-es.crt, logging-es.key]",
"reason": "Missing Required Secrets",
```

CHAPTER 4. ABOUT LOGGING

As a cluster administrator, you can deploy logging on an Red Hat OpenShift Service on AWS cluster, and use it to collect and aggregate node system audit logs, application container logs, and infrastructure logs. You can forward logs to your chosen log outputs, including on-cluster, Red Hat managed log storage. You can also visualize your log data in the Red Hat OpenShift Service on AWS web console, or the Kibana web console, depending on your deployed log storage solution.

NOTE

The Kibana web console is now deprecated is planned to be removed in a future logging release.

Red Hat OpenShift Service on AWS cluster administrators can deploy logging by using Operators. For information, see Installing logging.

The Operators are responsible for deploying, upgrading, and maintaining logging. After the Operators are installed, you can create a **ClusterLogging** custom resource (CR) to schedule logging pods and other resources necessary to support logging. You can also create a **ClusterLogForwarder** CR to specify which logs are collected, how they are transformed, and where they are forwarded to.

NOTE

Because the internal Red Hat OpenShift Service on AWS Elasticsearch log store does not provide secure storage for audit logs, audit logs are not stored in the internal Elasticsearch instance by default. If you want to send the audit logs to the default internal Elasticsearch log store, for example to view the audit logs in Kibana, you must use the Log Forwarding API as described in Forward audit logs to the log store .

4.1. LOGGING ARCHITECTURE

The major components of the logging are:

Collector

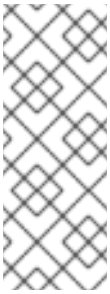
The collector is a daemonset that deploys pods to each Red Hat OpenShift Service on AWS node. It collects log data from each node, transforms the data, and forwards it to configured outputs. You can use the Vector collector or the legacy Fluentd collector.

NOTE

Fluentd is deprecated and is planned to be removed in a future release. Red Hat provides bug fixes and support for this feature during the current release lifecycle, but this feature no longer receives enhancements. As an alternative to Fluentd, you can use Vector instead.

Log store

The log store stores log data for analysis and is the default output for the log forwarder. You can use the default LokiStack log store, the legacy Elasticsearch log store, or forward logs to additional external log stores.

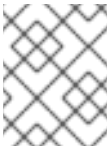


NOTE

The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

Visualization

You can use a UI component to view a visual representation of your log data. The UI provides a graphical interface to search, query, and view stored logs. The Red Hat OpenShift Service on AWS web console UI is provided by enabling the Red Hat OpenShift Service on AWS console plugin.



NOTE

The Kibana web console is now deprecated is planned to be removed in a future logging release.

Logging collects container logs and node logs. These are categorized into types:

Application logs

Container logs generated by user applications running in the cluster, except infrastructure container applications.

Infrastructure logs

Container logs generated by infrastructure namespaces: **openshift***, **kube***, or **default**, as well as journald messages from nodes.

Audit logs

Logs generated by auditd, the node audit system, which are stored in the **/var/log/audit/audit.log** file, and logs from the **auditd**, **kube-apiserver**, **openshift-apiserver** services, as well as the **ovn** project if enabled.

Additional resources

- Log visualization with the web console

4.2. ABOUT DEPLOYING LOGGING

Administrators can deploy the logging by using the Red Hat OpenShift Service on AWS web console or the OpenShift CLI (**oc**) to install the logging Operators. The Operators are responsible for deploying, upgrading, and maintaining the logging.

Administrators and application developers can view the logs of the projects for which they have view access.

4.2.1. Logging custom resources

You can configure your logging deployment with custom resource (CR) YAML files implemented by each Operator.

Red Hat OpenShift Logging Operator:

- **ClusterLogging** (CL) - After the Operators are installed, you create a **ClusterLogging** custom resource (CR) to schedule logging pods and other resources necessary to support the logging. The **ClusterLogging** CR deploys the collector and forwarder, which currently are both implemented by a daemonset running on each node. The Red Hat OpenShift Logging Operator watches the **ClusterLogging** CR and adjusts the logging deployment accordingly.

- **ClusterLogForwarder** (CLF) - Generates collector configuration to forward logs per user configuration.

Loki Operator:

- **LokiStack** - Controls the Loki cluster as log store and the web proxy with Red Hat OpenShift Service on AWS authentication integration to enforce multi-tenancy.

OpenShift Elasticsearch Operator:

> **NOTE**
>
> These CRs are generated and managed by the OpenShift Elasticsearch Operator. Manual changes cannot be made without being overwritten by the Operator.

- **ElasticSearch** - Configure and deploy an Elasticsearch instance as the default log store.

- **Kibana** - Configure and deploy Kibana instance to search, query and view logs.

4.3. CLOUDWATCH RECOMMENDATION FOR RED HAT OPENSHIFT SERVICE ON AWS

Red Hat recommends that you use the AWS CloudWatch solution for your logging needs.

4.3.1. Logging requirements

Hosting your own logging stack requires a large amount of compute resources and storage, which might be dependent on your cloud service quota. The compute resource requirements can start at 48 GB or more, while the storage requirement can be as large as 1600 GB or more. The logging stack runs on your worker nodes, which reduces your available workload resource. With these considerations, hosting your own logging stack increases your cluster operating costs.

Next steps

- See Forwarding logs to Amazon CloudWatch for instructions.

4.3.2. About JSON Red Hat OpenShift Service on AWS Logging

You can use JSON logging to configure the Log Forwarding API to parse JSON strings into a structured object. You can perform the following tasks:

- Parse JSON logs

- Configure JSON log data for Elasticsearch

- Forward JSON logs to the Elasticsearch log store

4.3.3. About collecting and storing Kubernetes events

The Red Hat OpenShift Service on AWS Event Router is a pod that watches Kubernetes events and logs them for collection by Red Hat OpenShift Service on AWS Logging. You must manually deploy the Event Router.

For information, see About collecting and storing Kubernetes events .

4.3.4. About troubleshooting Red Hat OpenShift Service on AWS Logging

You can troubleshoot the logging issues by performing the following tasks:

- Viewing logging status
- Viewing the status of the log store
- Understanding logging alerts
- Collecting logging data for Red Hat Support
- Troubleshooting for critical alerts

4.3.5. About exporting fields

The logging system exports fields. Exported fields are present in the log records and are available for searching from Elasticsearch and Kibana.

For information, see About exporting fields.

4.3.6. About event routing

The Event Router is a pod that watches Red Hat OpenShift Service on AWS events so they can be collected by logging. The Event Router collects events from all projects and writes them to **STDOUT**. Fluentd collects those events and forwards them into the Red Hat OpenShift Service on AWS Elasticsearch instance. Elasticsearch indexes the events to the **infra** index.

You must manually deploy the Event Router.

For information, see Collecting and storing Kubernetes events .

CHAPTER 5. INSTALLING LOGGING

You can deploy logging by installing the Red Hat OpenShift Logging Operator. The Red Hat OpenShift Logging Operator creates and manages the components of the logging stack.

NOTE

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

IMPORTANT

For new installations, use Vector and LokiStack. Elasticsearch and Fluentd are deprecated and are planned to be removed in a future release.

5.1. INSTALLING THE RED HAT OPENSHIFT LOGGING OPERATOR BY USING THE WEB CONSOLE

You can install the Red Hat OpenShift Logging Operator by using the Red Hat OpenShift Service on AWS web console.

Prerequisites

- You have administrator permissions.
- You have access to the Red Hat OpenShift Service on AWS web console.

Procedure

1. In the Red Hat OpenShift Service on AWS web console, click **Operators** → **OperatorHub**.
2. Type **OpenShift Logging** in the **Filter by keyword** box.
3. Choose **Red Hat OpenShift Logging** from the list of available Operators, and click **Install**.
4. In the **role ARN** text field, type "no_role_arn". Although the specific ARN for the cluster is not required, the **role ARN** text field must be populated to continue the installation process.

   NOTE

   The **role ARN** text field is only applicable for ROSA clusters with STS.

5. Select **stable-5.y** as the **Update channel**.

   NOTE

   The **stable** channel only provides updates to the most recent release of logging. To continue receiving updates for prior releases, you must change your subscription channel to **stable-x.y**, where **x.y** represents the major and minor version of logging you have installed. For example, **stable-5.7**.

6. Select a **Version**.
7. Ensure that **A specific namespace on the cluster** is selected under **Installation mode**.
8. Ensure that **Operator recommended namespace** is **openshift-logging** under **Installed Namespace**.
9. Select an **Update approval**.
   - The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.
   - The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.
10. Select **Enable** or **Disable** for the **Console plugin**.
11. Click **Install**.

Verification

1. Verify that the **Red Hat OpenShift Logging Operator** is installed by switching to the **Operators → Installed Operators** page.
2. In the **Status** column, verify that you see green checkmarks with **InstallSucceeded** and the text **Up to date**.

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

If the Operator does not show as installed, choose one of the following troubleshooting options:

- Go to the **Operators → Installed Operators** page, and inspect the **Status** column for any errors or failures.
- Go to the **Workloads → Pods** page, and check the logs in any pods in the **openshift-logging** project that are reporting issues.

5.2. CREATING A CLUSTERLOGGING OBJECT BY USING THE WEB CONSOLE

After you have installed the logging Operators, you must create a **ClusterLogging** custom resource to configure log storage, visualization, and the log collector for your cluster.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have access to the Red Hat OpenShift Service on AWS web console **Administrator** perspective.

Procedure

1. Navigate to the **Custom Resource Definitions** page.
2. On the **Custom Resource Definitions** page, click **ClusterLogging**.
3. On the **Custom Resource Definition details** page, select **View Instances** from the **Actions** menu.
4. On the **ClusterLoggings** page, click **Create ClusterLogging**.
5. In the **collection** section, select a **Collector Implementation**.

> **NOTE**
>
> Fluentd is deprecated and is planned to be removed in a future release. Red Hat provides bug fixes and support for this feature during the current release lifecycle, but this feature no longer receives enhancements. As an alternative to Fluentd, you can use Vector instead.

6. In the **logStore** section, select a type.

> **NOTE**
>
> The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

7. Click **Create**.

5.3. INSTALLING THE RED HAT OPENSHIFT LOGGING OPERATOR BY USING THE CLI

You can use the OpenShift CLI (**oc**) to install the Red Hat OpenShift Logging Operator.

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **Namespace** object as a YAML file:

Example Namespace object

```
apiVersion: v1
kind: Namespace
metadata:
  name: <name> (1)
```

```
  annotations:
    openshift.io/node-selector: ""
  labels:
    openshift.io/cluster-monitoring: "true"
```

1. You must specify **openshift-logging** as the name of the namespace for logging versions 5.7 and earlier versions. For logging 5.8 and later versions, you can use any name.

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

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

3. Create an **OperatorGroup** object as a YAML file:

Example OperatorGroup object

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: cluster-logging
  namespace: openshift-logging (1)
spec:
  targetNamespaces:
  - openshift-logging (2)
```

1, 2. You must specify the **openshift-logging** namespace for logging versions 5.7 and older. For logging 5.8 and later versions, you can use any namespace.

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

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

5. Create a **Subscription** object to subscribe the namespace to the 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 (2)
  name: cluster-logging
  source: redhat-operators (3)
  sourceNamespace: openshift-marketplace
```

1. You must specify the **openshift-logging** namespace for logging versions 5.7 and older. For logging 5.8 and later versions, you can use any namespace.

2. Specify **stable** or **stable-x.y** as the channel.
3. Specify **redhat-operators**. If your Red Hat OpenShift Service on AWS cluster is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object you created when you configured the Operator Lifecycle Manager (OLM).

6. Apply the subscription by running the following command:

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

The Red Hat OpenShift Logging Operator is installed to the **openshift-logging** namespace.

Verification

1. Run the following command:

```
$ oc get csv -n <namespace>
```

2. Observe the output and confirm that the Red Hat OpenShift Logging Operator exists in the namespace:

Example output

```
NAMESPACE                                       NAME                                    DISPLAY
VERSION            REPLACES   PHASE
...
openshift-logging                               clusterlogging.5.8.0-202007012112.p0
OpenShift Logging        5.8.0-202007012112.p0            Succeeded
...
```

5.4. CREATING A CLUSTERLOGGING OBJECT BY USING THE CLI

This default logging configuration supports a wide array of environments. Review the topics on tuning and configuring components for information about modifications you can make.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have installed the OpenShift Elasticsearch Operator for your log store.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **ClusterLogging** object as a YAML file:

Example ClusterLogging object

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
```

```
  name: instance (1)
  namespace: openshift-logging
spec:
  managementState: Managed (2)
  logStore:
    type: elasticsearch (3)
    retentionPolicy: (4)
      application:
        maxAge: 1d
      infra:
        maxAge: 7d
      audit:
        maxAge: 7d
    elasticsearch:
      nodeCount: 3 (5)
      storage:
        storageClassName: <storage_class_name> (6)
        size: 200G
      resources: (7)
          limits:
            memory: 16Gi
          requests:
            memory: 16Gi
      proxy: (8)
        resources:
          limits:
            memory: 256Mi
          requests:
            memory: 256Mi
      redundancyPolicy: SingleRedundancy
  visualization:
    type: kibana (9)
    kibana:
      replicas: 1
  collection:
    type: fluentd (10)
    fluentd: {}
```

1. The name must be **instance**.
2. The OpenShift Logging management state. In some cases, if you change the OpenShift Logging defaults, you must set this to **Unmanaged**. However, an unmanaged deployment does not receive updates until OpenShift Logging is placed back into a managed state.
3. Settings for configuring Elasticsearch. Using the CR, you can configure shard replication policy and persistent storage.
4. Specify the length of time that Elasticsearch should retain each log source. Enter an integer and a time designation: weeks(w), hours(h/H), minutes(m) and seconds(s). For example, **7d** for seven days. Logs older than the **maxAge** are deleted. You must specify a retention policy for each log source or the Elasticsearch indices will not be created for that source.
5. Specify the number of Elasticsearch nodes. See the note that follows this list.

6. Enter the name of an existing storage class for Elasticsearch storage. For best performance, specify a storage class that allocates block storage. If you do not specify a

7. Specify the CPU and memory requests for Elasticsearch as needed. If you leave these values blank, the OpenShift Elasticsearch Operator sets default values that should be sufficient for most deployments. The default values are **16Gi** for the memory request and **1** for the CPU request.

8. Specify the CPU and memory requests for the Elasticsearch proxy as needed. If you leave these values blank, the OpenShift Elasticsearch Operator sets default values that should be sufficient for most deployments. The default values are **256Mi** for the memory request and **100m** for the CPU request.

9. Settings for configuring Kibana. Using the CR, you can scale Kibana for redundancy and configure the CPU and memory for your Kibana nodes. For more information, see **Configuring the log visualizer**.

10. Settings for configuring Fluentd. Using the CR, you can configure Fluentd CPU and memory limits. For more information, see "Configuring Fluentd".

NOTE

The maximum number of Elasticsearch control plane nodes is three. If you specify a **nodeCount** greater than **3**, Red Hat OpenShift Service on AWS creates three Elasticsearch nodes that are Master-eligible nodes, with the master, client, and data roles. The additional Elasticsearch nodes are created as data-only nodes, using client and data roles. Control plane nodes perform cluster-wide actions such as creating or deleting an index, shard allocation, and tracking nodes. Data nodes hold the shards and perform data-related operations such as CRUD, search, and aggregations. Data-related operations are I/O-, memory-, and CPU-intensive. It is important to monitor these resources and to add more Data nodes if the current nodes are overloaded.

For example, if **nodeCount=4**, the following nodes are created:

```
$ oc get deployment
```

Example output

```
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
cluster-logging-operator              1/1     1            1           18h
elasticsearch-cd-x6kdekli-1           1/1     1            1           6m54s
elasticsearch-cdm-x6kdekli-1          1/1     1            1           18h
elasticsearch-cdm-x6kdekli-2          1/1     1            1           6m49s
elasticsearch-cdm-x6kdekli-3          1/1     1            1           6m44s
```

The number of primary shards for the index templates is equal to the number of Elasticsearch data nodes.

Verification

You can verify the installation by listing the pods in the **openshift-logging** project.

- List the pods by running the following command:

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

Observe the pods for components of the logging, similar to the following list:

Example output

```
NAME                                          READY   STATUS    RESTARTS   AGE
cluster-logging-operator-66f77ffccb-ppzbg     1/1     Running   0          7m
elasticsearch-cdm-ftuhduuw-1-ffc4b9566-q6bhp  2/2     Running   0          2m40s
elasticsearch-cdm-ftuhduuw-2-7b4994dbfc-rd2gc 2/2     Running   0          2m36s
elasticsearch-cdm-ftuhduuw-3-84b5ff7ff8-gqnm2 2/2     Running   0          2m4s
collector-587vb                               1/1     Running   0          2m26s
collector-7mpb9                               1/1     Running   0          2m30s
collector-flm6j                               1/1     Running   0          2m33s
collector-gn4rn                               1/1     Running   0          2m26s
collector-nlgb6                               1/1     Running   0          2m30s
collector-snpkt                               1/1     Running   0          2m28s
kibana-d6d5668c5-rppqm                        2/2     Running   0          2m39s
```

5.5. POSTINSTALLATION TASKS

After you have installed the Red Hat OpenShift Logging Operator, you can configure your deployment by creating and modifying a **ClusterLogging** custom resource (CR).

TIP

If you are not using the Elasticsearch log store, you can remove the internal Elasticsearch **logStore** and Kibana **visualization** components from the **ClusterLogging** custom resource (CR). Removing these components is optional but saves resources. See Removing unused components if you do not use the Elasticsearch log store.

5.5.1. About the ClusterLogging custom resource

To make changes to your logging environment, create and modify the **ClusterLogging** custom resource (CR).

Sample ClusterLogging custom resource (CR)

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance (1)
  namespace: openshift-logging (2)
spec:
  managementState: Managed (3)
# ...
```

1. The CR name must be **instance**.
2. The CR must be installed to the **openshift-logging** namespace.
3. The Red Hat OpenShift Logging Operator management state. When the state is set to **unmanaged**, the Operator is in an unsupported state and does not receive updates.

5.5.2. Configuring log storage

You can configure which log storage type your logging uses by modifying the **ClusterLogging** custom resource (CR).

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator and an internal log store that is either the LokiStack or Elasticsearch.
- You have created a **ClusterLogging** CR.

> **NOTE**
>
> The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

Procedure

1. Modify the **ClusterLogging** CR **logStore** spec:

ClusterLogging CR example

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
# ...
spec:
# ...
  logStore:
    type: <log_store_type> (1)
    elasticsearch: (2)
      nodeCount: <integer>
      resources: {}
      storage: {}
      redundancyPolicy: <redundancy_type> (3)
    lokistack: (4)
      name: {}
# ...
```

1. Specify the log store type. This can be either **lokistack** or **elasticsearch**.
2. Optional configuration options for the Elasticsearch log store.
3. Specify the redundancy type. This value can be **ZeroRedundancy**, **SingleRedundancy**, **MultipleRedundancy**, or **FullRedundancy**.

4 Optional configuration options for LokiStack.

Example ClusterLogging CR to specify LokiStack as the log store

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

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

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

5.5.3. Configuring the log collector

You can configure which log collector type your logging uses by modifying the **ClusterLogging** custom resource (CR).

NOTE

Fluentd is deprecated and is planned to be removed in a future release. Red Hat provides bug fixes and support for this feature during the current release lifecycle, but this feature no longer receives enhancements. As an alternative to Fluentd, you can use Vector instead.

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator.
- You have created a **ClusterLogging** CR.

Procedure

1. Modify the **ClusterLogging** CR **collection** spec:

ClusterLogging CR example

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
# ...
```

```
spec:
# ...
  collection:
    type: <log_collector_type> (1)
    resources: {}
    tolerations: {}
# ...
```

(1) The log collector type you want to use for the logging. This can be **vector** or **fluentd**.

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

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

5.5.4. Configuring the log visualizer

You can configure which log visualizer type your logging uses by modifying the **ClusterLogging** custom resource (CR).

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator.
- You have created a **ClusterLogging** CR.

IMPORTANT

If you want to use the Red Hat OpenShift Service on AWS web console for visualization, you must enable the logging Console Plugin. See the documentation about "Log visualization with the web console".

Procedure

1. Modify the **ClusterLogging** CR **visualization** spec:

ClusterLogging CR example

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
# ...
spec:
# ...
  visualization:
    type: <visualizer_type> (1)
    kibana: (2)
      resources: {}
      nodeSelector: {}
      proxy: {}
```

```
      replicas: {}
      tolerations: {}
    ocpConsole: 3
      logsLimit: {}
      timeout: {}
# ...
```

1. The type of visualizer you want to use for your logging. This can be either **kibana** or **ocp-console**. The Kibana console is only compatible with deployments that use Elasticsearch log storage, while the Red Hat OpenShift Service on AWS console is only compatible with LokiStack deployments.
2. Optional configurations for the Kibana console.
3. Optional configurations for the Red Hat OpenShift Service on AWS web console.

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

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

5.5.5. Allowing traffic between projects when network isolation is enabled

Your cluster network plugin might enforce network isolation. If so, you must allow network traffic between the projects that contain the operators deployed by OpenShift Logging.

Network isolation blocks network traffic between pods or services that are in different projects. The logging installs the *OpenShift Elasticsearch Operator* in the **openshift-operators-redhat** project and the *Red Hat OpenShift Logging Operator* in the **openshift-logging** project. Therefore, you must allow traffic between these two projects.

Red Hat OpenShift Service on AWS offers two supported choices for the network plugin, OpenShift SDN and OVN-Kubernetes. These two providers implement various network isolation policies.

OpenShift SDN has three modes:

network policy

This is the default mode. If no policy is defined, it allows all traffic. However, if a user defines a policy, they typically start by denying all traffic and then adding exceptions. This process might break applications that are running in different projects. Therefore, explicitly configure the policy to allow traffic to egress from one logging-related project to the other.

subnet

This mode allows all traffic. It does not enforce network isolation. No action is needed.

OVN-Kubernetes always uses a **network policy**. Therefore, as with OpenShift SDN, you must configure the policy to allow traffic to egress from one logging-related project to the other.

Procedure

- If you are using OpenShift SDN in **multitenant** mode, join the two projects. For example:

```
$ oc adm pod-network join-projects --to=openshift-operators-redhat openshift-logging
```

- Otherwise, for OpenShift SDN in **network policy** mode and OVN-Kubernetes, perform the following actions:

  a. Set a label on the **openshift-operators-redhat** namespace. For example:

  ```
  $ oc label namespace openshift-operators-redhat project=openshift-operators-redhat
  ```

  b. Create a network policy object in the **openshift-logging** namespace that allows ingress from the **openshift-operators-redhat**, **openshift-monitoring** and **openshift-ingress** projects to the openshift-logging project. For example:

  ```
  apiVersion: networking.k8s.io/v1
  kind: NetworkPolicy
  metadata:
    name: allow-from-openshift-monitoring-ingress-operators-redhat
  spec:
    ingress:
    - from:
      - podSelector: {}
    - from:
      - namespaceSelector:
          matchLabels:
            project: "openshift-operators-redhat"
    - from:
      - namespaceSelector:
          matchLabels:
            name: "openshift-monitoring"
    - from:
      - namespaceSelector:
          matchLabels:
            network.openshift.io/policy-group: ingress
    podSelector: {}
    policyTypes:
    - Ingress
  ```

Additional resources

- About network policy
- About the OpenShift SDN default CNI network provider
- About the OVN-Kubernetes default Container Network Interface (CNI) network provider

CHAPTER 6. UPDATING LOGGING

There are two types of logging updates: minor release updates (5.y.z) and major release updates (5.y).

6.1. MINOR RELEASE UPDATES

If you installed the logging Operators using the **Automatic** update approval option, your Operators receive minor version updates automatically. You do not need to complete any manual update steps.

If you installed the logging Operators using the **Manual** update approval option, you must manually approve minor version updates. For more information, see Manually approving a pending Operator update.

6.2. MAJOR RELEASE UPDATES

For major version updates you must complete some manual steps.

For major release version compatibility and support information, see OpenShift Operator Life Cycles.

6.3. UPGRADING THE RED HAT OPENSHIFT LOGGING OPERATOR TO WATCH ALL NAMESPACES

In logging 5.7 and older versions, the Red Hat OpenShift Logging Operator only watches the **openshift-logging** namespace. If you want the Red Hat OpenShift Logging Operator to watch all namespaces on your cluster, you must redeploy the Operator. You can complete the following procedure to redeploy the Operator without deleting your logging components.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have administrator permissions.

Procedure

1. Delete the subscription by running the following command:

```
$ oc -n openshift-logging delete subscription <subscription>
```

2. Delete the Operator group by running the following command:

```
$ oc -n openshift-logging delete operatorgroup <operator_group_name>
```

3. Delete the cluster service version (CSV) by running the following command:

```
$ oc delete clusterserviceversion cluster-logging.<version>
```

4. Redeploy the Red Hat OpenShift Logging Operator by following the "Installing Logging" documentation.

Verification

- Check that the **targetNamespaces** field in the **OperatorGroup** resource is not present or is set to an empty string.
  To do this, run the following command and inspect the output:

```
$ oc get operatorgroup <operator_group_name> -o yaml
```

Example output

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-logging-f52cn
  namespace: openshift-logging
spec:
  upgradeStrategy: Default
status:
  namespaces:
  - ""
# ...
```

6.4. UPDATING THE RED HAT OPENSHIFT LOGGING OPERATOR

To update the Red Hat OpenShift Logging Operator to a new major release version, you must modify the update channel for the Operator subscription.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.
- You have access to the Red Hat OpenShift Service on AWS web console and are viewing the **Administrator** perspective.

Procedure

1. Navigate to **Operators → Installed Operators**.
2. Select the **openshift-logging** project.
3. Click the **Red Hat OpenShift Logging** Operator.
4. Click **Subscription**. In the **Subscription details** section, click the **Update channel** link. This link text might be **stable** or **stable-5.y**, depending on your current update channel.
5. In the **Change Subscription Update Channel** window, select the latest major version update channel, **stable-5.y**, and click **Save**. Note the **cluster-logging.v5.y.z** version.

Verification

1. Wait for a few seconds, then click **Operators → Installed Operators**. Verify that the Red Hat OpenShift Logging Operator version matches the latest **cluster-logging.v5.y.z** version.
2. On the **Operators → Installed Operators** page, wait for the **Status** field to report **Succeeded**.

6.5. UPDATING THE LOKI OPERATOR

To update the Loki Operator to a new major release version, you must modify the update channel for the Operator subscription.

Prerequisites

- You have installed the Loki Operator.
- You have administrator permissions.
- You have access to the Red Hat OpenShift Service on AWS web console and are viewing the **Administrator** perspective.

Procedure

1. Navigate to **Operators → Installed Operators**.
2. Select the **openshift-operators-redhat** project.
3. Click the **Loki Operator**.
4. Click **Subscription**. In the **Subscription details** section, click the **Update channel** link. This link text might be **stable** or **stable-5.y**, depending on your current update channel.
5. In the **Change Subscription Update Channel** window, select the latest major version update channel, **stable-5.y**, and click **Save**. Note the **loki-operator.v5.y.z** version.

Verification

1. Wait for a few seconds, then click **Operators → Installed Operators**. Verify that the Loki Operator version matches the latest **loki-operator.v5.y.z** version.
2. On the **Operators → Installed Operators** page, wait for the **Status** field to report **Succeeded**.

6.6. UPDATING THE OPENSHIFT ELASTICSEARCH OPERATOR

To update the OpenShift Elasticsearch Operator to the current version, you must modify the subscription.

NOTE

The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

Prerequisites

- If you are using Elasticsearch as the default log store, and Kibana as the UI, update the OpenShift Elasticsearch Operator before you update the Red Hat OpenShift Logging Operator.

IMPORTANT

If you update the Operators in the wrong order, Kibana does not update and the Kibana custom resource (CR) is not created. To fix this issue, delete the Red Hat OpenShift Logging Operator pod. When the Red Hat OpenShift Logging Operator pod redeploys, it creates the Kibana CR and Kibana becomes available again.

- The Logging status is healthy:
  - All pods have a **ready** status.
  - The Elasticsearch cluster is healthy.
- Your Elasticsearch and Kibana data is backed up .
- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**) for the verification steps.

Procedure

1. In the Red Hat Hybrid Cloud Console, click **Operators → Installed Operators**.
2. Select the **openshift-operators-redhat** project.
3. Click **OpenShift Elasticsearch Operator**.
4. Click **Subscription → Channel**.
5. In the **Change Subscription Update Channel** window, select **stable-5.y** and click **Save**. Note the **elasticsearch-operator.v5.y.z** version.
6. Wait for a few seconds, then click **Operators → Installed Operators**. Verify that the OpenShift Elasticsearch Operator version matches the latest **elasticsearch-operator.v5.y.z** version.
7. On the **Operators → Installed Operators** page, wait for the **Status** field to report **Succeeded**.

Verification

1. Verify that all Elasticsearch pods have a **Ready** status by entering the following command and observing the output:

```
$ oc get pod -n openshift-logging --selector component=elasticsearch
```

Example output

```
NAME                                            READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-1pbrl44l-1-55b7546f4c-mshhk   2/2     Running   0          31m
elasticsearch-cdm-1pbrl44l-2-5c6d87589f-gx5hk   2/2     Running   0          30m
elasticsearch-cdm-1pbrl44l-3-88df5d47-m45jc     2/2     Running   0          29m
```

2. Verify that the Elasticsearch cluster status is **green** by entering the following command and observing the output:

```
$ oc exec -n openshift-logging -c elasticsearch elasticsearch-cdm-1pbrl44l-1-55b7546f4c-mshhk -- health
```

Example output

```
{
  "cluster_name" : "elasticsearch",
  "status" : "green",
}
```

3. Verify that the Elasticsearch cron jobs are created by entering the following commands and observing the output:

```
$ oc project openshift-logging
```

```
$ oc get cronjob
```

Example output

```
NAME                    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
elasticsearch-im-app    */15 * * * *   False     0        <none>          56s
elasticsearch-im-audit  */15 * * * *   False     0        <none>          56s
elasticsearch-im-infra  */15 * * * *   False     0        <none>          56s
```

4. Verify that the log store is updated to the correct version and the indices are **green** by entering the following command and observing the output:

```
$ oc exec -c elasticsearch <any_es_pod_in_the_cluster> -- indices
```

Verify that the output includes the **app-00000x**, **infra-00000x**, **audit-00000x**, **.security** indices:

Example 6.1. Sample output with indices in a green status

```
Tue Jun 30 14:30:54 UTC 2020
health status index                                                    uuid                   pri rep
docs.count docs.deleted store.size pri.store.size
green  open   infra-000008
bnBvUFEXTWi92z3zWAzieQ   3 1       222195            0       289           144
green  open   infra-000004                                                rtDSzoqsSl6saisSK7Au1Q
3 1       226717            0       297           148
green  open   infra-000012
RSf_kUwDSR2xEuKRZMPqZQ   3 1       227623            0       295           147
green  open   .kibana_7                                                   1SJdCqlZTPWIIAaOUd78yg
1 1           4            0         0             0
green  open   infra-000010
iXwL3bnqTuGEABbUDa6OVw   3 1       248368            0       317           158
green  open   infra-000009
YN9EsULWSNaxWeeNvOs0RA   3 1       258799            0       337           168
green  open   infra-000014
YP0U6R7FQ_GVQVQZ6Yh9Ig   3 1       223788            0       292           146
green  open   infra-000015
JRBbAbEmSMqK5X40df9HbQ   3 1       224371            0       291           145
green  open   .orphaned.2020.06.30
```

```
n_xQC2dWQzConkvQqei3YA   3 1          9          0        0            0
green  open   infra-000007
llkkAVSzSOmosWTSAJM_hg   3 1      228584          0      296          148
green  open   infra-000005
d9BoGQdiQASsS3BBFm2iRA   3 1      227987          0      297          148
green  open   infra-000003                                          1-
goREK1QUKlQPAIVkWVaQ   3 1      226719          0      295          147
green  open   .security                                          zeT65uOuRTKZMjg_bbUc1g
1 1          5          0        0            0
green  open   .kibana-377444158_kubeadmin                                wvMhDwJkR-
mRZQO84K0gUQ   3 1          1          0        0            0
green  open   infra-000006                                          5H-
KBSXGQKiO7hdapDE23g   3 1      226676          0      295          147
green  open   infra-000001                                          eH53BQ-
bSxSWR5xYZB6lVg   3 1      341800          0      443          220
green  open   .kibana-6
RVp7TemSSemGJcsSUmuf3A   1 1          4          0        0            0
green  open   infra-000011
J7XWBauWSTe0jnzX02fU6A   3 1      226100          0      293          146
green  open   app-000001
axSAFfONQDmKwatkjPXdtw   3 1      103186          0      126           57
green  open   infra-000016
m9c1iRLtStWSF1GopaRyCg   3 1       13685          0       19            9
green  open   infra-000002                                          Hz6WvINtTvKcQzw-
ewmbYg   3 1      228994          0      296          148
green  open   infra-000013                                          KR9mMFUpQl-
jraYtanyIGw   3 1      228166          0      298          148
green  open   audit-000001
eERqLdLmQOiQDFES1LBATQ   3 1          0          0        0            0
```

5. Verify that the log visualizer is updated to the correct version by entering the following command and observing the output:

```
$ oc get kibana kibana -o json
```

Verify that the output includes a Kibana pod with the **ready** status:

Example 6.2. Sample output with a ready Kibana pod

```
[
{
"clusterCondition": {
"kibana-5fdd766ffd-nb2jj": [
{
"lastTransitionTime": "2020-06-30T14:11:07Z",
"reason": "ContainerCreating",
"status": "True",
"type": ""
},
{
"lastTransitionTime": "2020-06-30T14:11:07Z",
"reason": "ContainerCreating",
"status": "True",
"type": ""
```

```
}
]
},
"deployment": "kibana",
"pods": {
"failed": [],
"notReady": []
"ready": []
},
"replicaSets": [
"kibana-5fdd766ffd"
],
"replicas": 1
}
]
```

CHAPTER 7. VISUALIZING LOGS

7.1. ABOUT LOG VISUALIZATION

You can visualize your log data in the Red Hat OpenShift Service on AWS web console, or the Kibana web console, depending on your deployed log storage solution. The Kibana console can be used with ElasticSearch log stores, and the Red Hat OpenShift Service on AWS web console can be used with the ElasticSearch log store or the LokiStack.

NOTE

The Kibana web console is now deprecated is planned to be removed in a future logging release.

7.1.1. Configuring the log visualizer

You can configure which log visualizer type your logging uses by modifying the **ClusterLogging** custom resource (CR).

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator.
- You have created a **ClusterLogging** CR.

IMPORTANT

If you want to use the Red Hat OpenShift Service on AWS web console for visualization, you must enable the logging Console Plugin. See the documentation about "Log visualization with the web console".

Procedure

1. Modify the **ClusterLogging** CR **visualization** spec:

ClusterLogging CR example

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
# ...
spec:
# ...
  visualization:
    type: <visualizer_type> (1)
    kibana: (2)
      resources: {}
      nodeSelector: {}
      proxy: {}
```

```
      replicas: {}
      tolerations: {}
    ocpConsole: (3)
      logsLimit: {}
      timeout: {}
# ...
```

1. The type of visualizer you want to use for your logging. This can be either **kibana** or **ocp-console**. The Kibana console is only compatible with deployments that use Elasticsearch log storage, while the Red Hat OpenShift Service on AWS console is only compatible with LokiStack deployments.
2. Optional configurations for the Kibana console.
3. Optional configurations for the Red Hat OpenShift Service on AWS web console.

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

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

7.1.2. Viewing logs for a resource

Resource logs are a default feature that provides limited log viewing capability. You can view the logs for various resources, such as builds, deployments, and pods by using the OpenShift CLI (**oc**) and the web console.

TIP

To enhance your log retrieving and viewing experience, install the logging. The logging aggregates all the logs from your Red Hat OpenShift Service on AWS cluster, such as node system audit logs, application container logs, and infrastructure logs, into a dedicated log store. You can then query, discover, and visualize your log data through the Kibana console or the Red Hat OpenShift Service on AWS web console. Resource logs do not access the logging log store.

7.1.2.1. Viewing resource logs

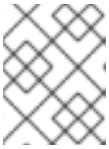
You can view the log for various resources in the OpenShift CLI (**oc**) and web console. Logs read from the tail, or end, of the log.

Prerequisites

- Access to the OpenShift CLI (**oc**).

Procedure (UI)

1. In the Red Hat OpenShift Service on AWS console, navigate to **Workloads** → **Pods** or navigate to the pod through the resource you want to investigate.



NOTE

Some resources, such as builds, do not have pods to query directly. In such instances, you can locate the **Logs** link on the **Details** page for the resource.

2. Select a project from the drop-down menu.
3. Click the name of the pod you want to investigate.
4. Click **Logs**.

Procedure (CLI)

- View the log for a specific pod:

```
$ oc logs -f <pod_name> -c <container_name>
```

where:

-f
Optional: Specifies that the output follows what is being written into the logs.

<pod_name>
Specifies the name of the pod.

<container_name>
Optional: Specifies the name of a container. When a pod has more than one container, you must specify the container name.

For example:

```
$ oc logs ruby-58cd97df55-mww7r
```

```
$ oc logs -f ruby-57f7f4855b-znl92 -c ruby
```

The contents of log files are printed out.

- View the log for a specific resource:

```
$ oc logs <object_type>/<resource_name> (1)
```

(1) Specifies the resource type and name.

For example:

```
$ oc logs deployment/ruby
```

The contents of log files are printed out.

7.2. LOG VISUALIZATION WITH THE WEB CONSOLE

You can use the Red Hat OpenShift Service on AWS web console to visualize log data by configuring the logging Console Plugin.

For information about configuring the plugin during the logging installation, see Installing the logging using the web console.

If you have already installed the logging and want to configure the plugin, use one of the following procedures.

7.2.1. Enabling the logging Console Plugin after you have installed the Red Hat OpenShift Logging Operator

You can enable the logging Console Plugin as part of the Red Hat OpenShift Logging Operator installation, but you can also enable the plugin if you have already installed the Red Hat OpenShift Logging Operator with the plugin disabled.

Prerequisites

- You have administrator permissions.
- You have installed the Red Hat OpenShift Logging Operator and selected **Disabled** for the **Console plugin**.
- You have access to the Red Hat OpenShift Service on AWS web console.

Procedure

1. In the Red Hat OpenShift Service on AWS web console **Administrator** perspective, navigate to **Operators → Installed Operators**.
2. Click **Red Hat OpenShift Logging**. This takes you to the Operator **Details** page.
3. In the **Details** page, click **Disabled** for the **Console plugin** option.
4. In the **Console plugin enablement** dialog, select **Enable**.
5. Click **Save**.
6. Verify that the **Console plugin** option now shows **Enabled**.
7. The web console displays a pop-up window when changes have been applied. The window prompts you to reload the web console. Refresh the browser when you see the pop-up window to apply the changes.

7.2.2. Configuring the logging Console Plugin when you have the Elasticsearch log store and LokiStack installed

In logging version 5.8 and later, if the Elasticsearch log store is your default log store but you have also installed the LokiStack, you can enable the logging Console Plugin by using the following procedure.

Prerequisites

- You have administrator permissions.
- You have installed the Red Hat OpenShift Logging Operator, the OpenShift Elasticsearch Operator, and the Loki Operator.
- You have installed the OpenShift CLI (**oc**).
- You have created a **ClusterLogging** custom resource (CR).

Procedure

1. Ensure that the logging Console Plugin is enabled by running the following command:

```
$ oc get consoles.operator.openshift.io cluster -o yaml |grep logging-view-plugin  \
|| oc patch consoles.operator.openshift.io cluster  --type=merge \
--patch '{ "spec": { "plugins": ["logging-view-plugin"]}}'
```

2. Add the **.metadata.annotations.logging.openshift.io/ocp-console-migration-target: lokistack-dev** annotation to the **ClusterLogging** CR, by running the following command:

```
$ oc patch clusterlogging instance --type=merge --patch \
'{ "metadata": { "annotations": { "logging.openshift.io/ocp-console-migration-target":
"lokistack-dev" }}}' \
-n openshift-logging
```

Example output

```
clusterlogging.logging.openshift.io/instance patched
```

Verification

- Verify that the annotation was added successfully, by running the following command and observing the output:

```
$ oc get clusterlogging instance \
-o=jsonpath='{.metadata.annotations.logging\.openshift\.io/ocp-console-migration-target}' \
-n openshift-logging
```

Example output

```
"lokistack-dev"
```

The logging Console Plugin pod is now deployed. You can view logging data by navigating to the Red Hat OpenShift Service on AWS web console and viewing the **Observe → Logs** page.

7.3. VIEWING CLUSTER DASHBOARDS

The **Logging/Elasticsearch Nodes** and **Openshift Logging** dashboards in the OpenShift Cluster Manager contain in-depth details about your Elasticsearch instance and the individual Elasticsearch nodes that you can use to prevent and diagnose problems.

The **OpenShift Logging** dashboard contains charts that show details about your Elasticsearch instance at a cluster level, including cluster resources, garbage collection, shards in the cluster, and Fluentd statistics.

The **Logging/Elasticsearch Nodes** dashboard contains charts that show details about your Elasticsearch instance, many at node level, including details on indexing, shards, resources, and so forth.

7.3.1. Accessing the Elasticsearch and OpenShift Logging dashboards

You can view the **Logging/Elasticsearch Nodes** and **OpenShift Logging** dashboards in the OpenShift Cluster Manager.

Procedure

To launch the dashboards:

1. In the Red Hat OpenShift Service on AWS Red Hat Hybrid Cloud Console, click **Observe → Dashboards**.

2. On the **Dashboards** page, select **Logging/Elasticsearch Nodes** or **OpenShift Logging** from the **Dashboard** menu.
   For the **Logging/Elasticsearch Nodes** dashboard, you can select the Elasticsearch node you want to view and set the data resolution.

   The appropriate dashboard is displayed, showing multiple charts of data.

3. Optional: Select a different time range to display or refresh rate for the data from the **Time Range** and **Refresh Interval** menus.

For information on the dashboard charts, see About the OpenShift Logging dashboard and About the Logging/Elastisearch Nodes dashboard.

7.3.2. About the OpenShift Logging dashboard

The **OpenShift Logging** dashboard contains charts that show details about your Elasticsearch instance at a cluster-level that you can use to diagnose and anticipate problems.

Table 7.1. OpenShift Logging charts

Metric	Description
Elastic Cluster Status	The current Elasticsearch status: • ONLINE - Indicates that the Elasticsearch instance is online. • OFFLINE - Indicates that the Elasticsearch instance is offline.
Elastic Nodes	The total number of Elasticsearch nodes in the Elasticsearch instance.
Elastic Shards	The total number of Elasticsearch shards in the Elasticsearch instance.
Elastic Documents	The total number of Elasticsearch documents in the Elasticsearch instance.
Total Index Size on Disk	The total disk space that is being used for the Elasticsearch indices.
Elastic Pending Tasks	The total number of Elasticsearch changes that have not been completed, such as index creation, index mapping, shard allocation, or shard failure.

Metric	Description
Elastic JVM GC time	The amount of time that the JVM spent executing Elasticsearch garbage collection operations in the cluster.
Elastic JVM GC Rate	The total number of times that JVM executed garbage activities per second.
Elastic Query/Fetch Latency Sum	• Query latency: The average time each Elasticsearch search query takes to execute. • Fetch latency: The average time each Elasticsearch search query spends fetching data. Fetch latency typically takes less time than query latency. If fetch latency is consistently increasing, it might indicate slow disks, data enrichment, or large requests with too many results.
Elastic Query Rate	The total queries executed against the Elasticsearch instance per second for each Elasticsearch node.
CPU	The amount of CPU used by Elasticsearch, Fluentd, and Kibana, shown for each component.
Elastic JVM Heap Used	The amount of JVM memory used. In a healthy cluster, the graph shows regular drops as memory is freed by JVM garbage collection.
Elasticsearch Disk Usage	The total disk space used by the Elasticsearch instance for each Elasticsearch node.
File Descriptors In Use	The total number of file descriptors used by Elasticsearch, Fluentd, and Kibana.
FluentD emit count	The total number of Fluentd messages per second for the Fluentd default output, and the retry count for the default output.
FluentD Buffer Usage	The percent of the Fluentd buffer that is being used for chunks. A full buffer might indicate that Fluentd is not able to process the number of logs received.
Elastic rx bytes	The total number of bytes that Elasticsearch has received from FluentD, the Elasticsearch nodes, and other sources.

Metric	Description
Elastic Index Failure Rate	The total number of times per second that an Elasticsearch index fails. A high rate might indicate an issue with indexing.
FluentD Output Error Rate	The total number of times per second that FluentD is not able to output logs.

7.3.3. Charts on the Logging/Elasticsearch nodes dashboard

The **Logging/Elasticsearch Nodes** dashboard contains charts that show details about your Elasticsearch instance, many at node-level, for further diagnostics.

Elasticsearch status

The **Logging/Elasticsearch Nodes** dashboard contains the following charts about the status of your Elasticsearch instance.

Table 7.2. Elasticsearch status fields

Metric	Description
Cluster status	The cluster health status during the selected time period, using the Elasticsearch green, yellow, and red statuses: • 0 - Indicates that the Elasticsearch instance is in green status, which means that all shards are allocated. • 1 - Indicates that the Elasticsearch instance is in yellow status, which means that replica shards for at least one shard are not allocated. • 2 - Indicates that the Elasticsearch instance is in red status, which means that at least one primary shard and its replicas are not allocated.
Cluster nodes	The total number of Elasticsearch nodes in the cluster.
Cluster data nodes	The number of Elasticsearch data nodes in the cluster.
Cluster pending tasks	The number of cluster state changes that are not finished and are waiting in a cluster queue, for example, index creation, index deletion, or shard allocation. A growing trend indicates that the cluster is not able to keep up with changes.

Elasticsearch cluster index shard status

Each Elasticsearch index is a logical group of one or more shards, which are basic units of persisted data. There are two types of index shards: primary shards, and replica shards. When a document is indexed into an index, it is stored in one of its primary shards and copied into every replica of that shard. The number of primary shards is specified when the index is created, and the number cannot change during index lifetime. You can change the number of replica shards at any time.

The index shard can be in several states depending on its lifecycle phase or events occurring in the cluster. When the shard is able to perform search and indexing requests, the shard is active. If the shard cannot perform these requests, the shard is non–active. A shard might be non-active if the shard is initializing, reallocating, unassigned, and so forth.

Index shards consist of a number of smaller internal blocks, called index segments, which are physical representations of the data. An index segment is a relatively small, immutable Lucene index that is created when Lucene commits newly-indexed data. Lucene, a search library used by Elasticsearch, merges index segments into larger segments in the background to keep the total number of segments low. If the process of merging segments is slower than the speed at which new segments are created, it could indicate a problem.

When Lucene performs data operations, such as a search operation, Lucene performs the operation against the index segments in the relevant index. For that purpose, each segment contains specific data structures that are loaded in the memory and mapped. Index mapping can have a significant impact on the memory used by segment data structures.

The **Logging/Elasticsearch Nodes** dashboard contains the following charts about the Elasticsearch index shards.

Table 7.3. Elasticsearch cluster shard status charts

Metric	Description
Cluster active shards	The number of active primary shards and the total number of shards, including replicas, in the cluster. If the number of shards grows higher, the cluster performance can start degrading.
Cluster initializing shards	The number of non-active shards in the cluster. A non-active shard is one that is initializing, being reallocated to a different node, or is unassigned. A cluster typically has non–active shards for short periods. A growing number of non–active shards over longer periods could indicate a problem.
Cluster relocating shards	The number of shards that Elasticsearch is relocating to a new node. Elasticsearch relocates nodes for multiple reasons, such as high memory use on a node or after a new node is added to the cluster.
Cluster unassigned shards	The number of unassigned shards. Elasticsearch shards might be unassigned for reasons such as a new index being added or the failure of a node.

Elasticsearch node metrics

Each Elasticsearch node has a finite amount of resources that can be used to process tasks. When all the resources are being used and Elasticsearch attempts to perform a new task, Elasticsearch puts the tasks into a queue until some resources become available.

The **Logging/Elasticsearch Nodes** dashboard contains the following charts about resource usage for a selected node and the number of tasks waiting in the Elasticsearch queue.

Table 7.4. Elasticsearch node metric charts

Metric	Description
ThreadPool tasks	The number of waiting tasks in individual queues, shown by task type. A long–term accumulation of tasks in any queue could indicate node resource shortages or some other problem.
CPU usage	The amount of CPU being used by the selected Elasticsearch node as a percentage of the total CPU allocated to the host container.
Memory usage	The amount of memory being used by the selected Elasticsearch node.
Disk usage	The total disk space being used for index data and metadata on the selected Elasticsearch node.
Documents indexing rate	The rate that documents are indexed on the selected Elasticsearch node.
Indexing latency	The time taken to index the documents on the selected Elasticsearch node. Indexing latency can be affected by many factors, such as JVM Heap memory and overall load. A growing latency indicates a resource capacity shortage in the instance.
Search rate	The number of search requests run on the selected Elasticsearch node.
Search latency	The time taken to complete search requests on the selected Elasticsearch node. Search latency can be affected by many factors. A growing latency indicates a resource capacity shortage in the instance.
Documents count (with replicas)	The number of Elasticsearch documents stored on the selected Elasticsearch node, including documents stored in both the primary shards and replica shards that are allocated on the node.
Documents deleting rate	The number of Elasticsearch documents being deleted from any of the index shards that are allocated to the selected Elasticsearch node.

Metric	Description
Documents merging rate	The number of Elasticsearch documents being merged in any of index shards that are allocated to the selected Elasticsearch node.

Elasticsearch node fielddata

Fielddata is an Elasticsearch data structure that holds lists of terms in an index and is kept in the JVM Heap. Because fielddata building is an expensive operation, Elasticsearch caches the fielddata structures. Elasticsearch can evict a fielddata cache when the underlying index segment is deleted or merged, or if there is not enough JVM HEAP memory for all the fielddata caches.

The **Logging/Elasticsearch Nodes** dashboard contains the following charts about Elasticsearch fielddata.

Table 7.5. Elasticsearch node fielddata charts

Metric	Description
Fielddata memory size	The amount of JVM Heap used for the fielddata cache on the selected Elasticsearch node.
Fielddata evictions	The number of fielddata structures that were deleted from the selected Elasticsearch node.

Elasticsearch node query cache

If the data stored in the index does not change, search query results are cached in a node-level query cache for reuse by Elasticsearch.

The **Logging/Elasticsearch Nodes** dashboard contains the following charts about the Elasticsearch node query cache.

Table 7.6. Elasticsearch node query charts

Metric	Description
Query cache size	The total amount of memory used for the query cache for all the shards allocated to the selected Elasticsearch node.
Query cache evictions	The number of query cache evictions on the selected Elasticsearch node.
Query cache hits	The number of query cache hits on the selected Elasticsearch node.
Query cache misses	The number of query cache misses on the selected Elasticsearch node.

Elasticsearch index throttling

When indexing documents, Elasticsearch stores the documents in index segments, which are physical representations of the data. At the same time, Elasticsearch periodically merges smaller segments into a larger segment as a way to optimize resource use. If the indexing is faster then the ability to merge segments, the merge process does not complete quickly enough, which can lead to issues with searches and performance. To prevent this situation, Elasticsearch throttles indexing, typically by reducing the number of threads allocated to indexing down to a single thread.

The **Logging/Elasticsearch Nodes** dashboard contains the following charts about Elasticsearch index throttling.

Table 7.7. Index throttling charts

Metric	Description
Indexing throttling	The amount of time that Elasticsearch has been throttling the indexing operations on the selected Elasticsearch node.
Merging throttling	The amount of time that Elasticsearch has been throttling the segment merge operations on the selected Elasticsearch node.

Node JVM Heap statistics

The **Logging/Elasticsearch Nodes** dashboard contains the following charts about JVM Heap operations.

Table 7.8. JVM Heap statistic charts

Metric	Description
Heap used	The amount of the total allocated JVM Heap space that is used on the selected Elasticsearch node.
GC count	The number of garbage collection operations that have been run on the selected Elasticsearch node, by old and young garbage collection.
GC time	The amount of time that the JVM spent running garbage collection operations on the selected Elasticsearch node, by old and young garbage collection.

7.4. LOG VISUALIZATION WITH KIBANA

If you are using the ElasticSearch log store, you can use the Kibana console to visualize collected log data.

Using Kibana, you can do the following with your data:

- Search and browse the data using the **Discover** tab.

- Chart and map the data using the **Visualize** tab.
- Create and view custom dashboards using the **Dashboard** tab.

Use and configuration of the Kibana interface is beyond the scope of this documentation. For more information about using the interface, see the Kibana documentation.

> **NOTE**
>
> The audit logs are not stored in the internal Red Hat OpenShift Service on AWS Elasticsearch instance by default. To view the audit logs in Kibana, you must use the Log Forwarding API to configure a pipeline that uses the **default** output for audit logs.

7.4.1. Defining Kibana index patterns

An index pattern defines the Elasticsearch indices that you want to visualize. To explore and visualize data in Kibana, you must create an index pattern.

Prerequisites

- A user must have the **cluster-admin** role, the **cluster-reader** role, or both roles to view the **infra** and **audit** indices in Kibana. The default **kubeadmin** user has proper permissions to view these indices.
  If you can view the pods and logs in the **default**, **kube-** and **openshift-** projects, you should be able to access these indices. You can use the following command to check if the current user has appropriate permissions:

  ```
  $ oc auth can-i get pods --subresource log -n <project>
  ```

  Example output

  ```
  yes
  ```

  > **NOTE**
  >
  > The audit logs are not stored in the internal Red Hat OpenShift Service on AWS Elasticsearch instance by default. To view the audit logs in Kibana, you must use the Log Forwarding API to configure a pipeline that uses the **default** output for audit logs.

- Elasticsearch documents must be indexed before you can create index patterns. This is done automatically, but it might take a few minutes in a new or updated cluster.

Procedure

To define index patterns and create visualizations in Kibana:

1. In the Red Hat OpenShift Service on AWS console, click the Application Launcher and select **Logging**.
2. Create your Kibana index patterns by clicking **Management → Index Patterns → Create index pattern**:

- Each user must manually create index patterns when logging into Kibana the first time to see logs for their projects. Users must create an index pattern named **app** and use the **@timestamp** time field to view their container logs.
- Each admin user must create index patterns when logged into Kibana the first time for the **app**, **infra**, and **audit** indices using the **@timestamp** time field.

3. Create Kibana Visualizations from the new index patterns.

7.4.2. Viewing cluster logs in Kibana

You view cluster logs in the Kibana web console. The methods for viewing and visualizing your data in Kibana that are beyond the scope of this documentation. For more information, refer to the Kibana documentation.

Prerequisites

- The Red Hat OpenShift Logging and Elasticsearch Operators must be installed.
- Kibana index patterns must exist.
- A user must have the **cluster-admin** role, the **cluster-reader** role, or both roles to view the **infra** and **audit** indices in Kibana. The default **kubeadmin** user has proper permissions to view these indices.
  If you can view the pods and logs in the **default**, **kube-** and **openshift-** projects, you should be able to access these indices. You can use the following command to check if the current user has appropriate permissions:

  ```
  $ oc auth can-i get pods --subresource log -n <project>
  ```

  Example output

  ```
  yes
  ```

  NOTE

  The audit logs are not stored in the internal Red Hat OpenShift Service on AWS Elasticsearch instance by default. To view the audit logs in Kibana, you must use the Log Forwarding API to configure a pipeline that uses the **default** output for audit logs.

Procedure

To view logs in Kibana:

1. In the Red Hat OpenShift Service on AWS console, click the Application Launcher and select **Logging**.
2. Log in using the same credentials you use to log in to the Red Hat OpenShift Service on AWS console.
   The Kibana interface launches.
3. In Kibana, click **Discover**.

4. Select the index pattern you created from the drop-down menu in the top-left corner: **app**, **audit**, or **infra**.
   The log data displays as time-stamped documents.

5. Expand one of the time-stamped documents.

6. Click the **JSON** tab to display the log entry for that document.

Example 7.1. Sample infrastructure log entry in Kibana


```
{
  "_index": "infra-000001",
  "_type": "_doc",
  "_id": "YmJmYTBlNDkZTRmLTliMGQtMjE3NmFiOGUyOWM3",
  "_version": 1,
  "_score": null,
  "_source": {
    "docker": {
      "container_id": "f85fa55bbef7bb783f041066be1e7c267a6b88c4603dfce213e32c1"
    },
    "kubernetes": {
      "container_name": "registry-server",
      "namespace_name": "openshift-marketplace",
      "pod_name": "redhat-marketplace-n64gc",
      "container_image": "registry.redhat.io/redhat/redhat-marketplace-index:v4.7",
      "container_image_id": "registry.redhat.io/redhat/redhat-marketplace-
index@sha256:65fc0c45aabb95809e376feb065771ecda9e5e59cc8b3024c4545c168f",
      "pod_id": "8f594ea2-c866-4b5c-a1c8-a50756704b2a",
      "host": "ip-10-0-182-28.us-east-2.compute.internal",
      "master_url": "https://kubernetes.default.svc",
      "namespace_id": "3abab127-7669-4eb3-b9ef-44c04ad68d38",
      "namespace_labels": {
        "openshift_io/cluster-monitoring": "true"
      },
      "flat_labels": [
        "catalogsource_operators_coreos_com/update=redhat-marketplace"
      ]
    },
    "message": "time=\"2020-09-23T20:47:03Z\" level=info msg=\"serving registry\"
database=/database/index.db port=50051",
    "level": "unknown",
    "hostname": "ip-10-0-182-28.internal",
    "pipeline_metadata": {
      "collector": {
        "ipaddr4": "10.0.182.28",
        "inputname": "fluent-plugin-systemd",
        "name": "fluentd",
        "received_at": "2020-09-23T20:47:15.007583+00:00",
        "version": "1.7.4 1.6.0"
      }
    },
    "@timestamp": "2020-09-23T20:47:03.422465+00:00",
    "viaq_msg_id": "YmJmYTBlNDktMDMGQtMjE3NmFiOGUyOWM3",
    "openshift": {
      "labels": {
        "logging": "infra"
      }
```

```
      }
    },
    "fields": {
      "@timestamp": [
        "2020-09-23T20:47:03.422Z"
      ],
      "pipeline_metadata.collector.received_at": [
        "2020-09-23T20:47:15.007Z"
      ]
    },
    "sort": [
      1600894023422
    ]
  }
```

7.4.3. Configuring Kibana

You can configure using the Kibana console by modifying the **ClusterLogging** custom resource (CR).

7.4.3.1. Configuring CPU and memory limits

The logging components allow for adjustments to both the CPU and memory limits.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc -n openshift-logging edit ClusterLogging instance
```

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: openshift-logging

...

spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      resources: (1)
        limits:
          memory: 16Gi
        requests:
          cpu: 200m
          memory: 16Gi
      storage:
        storageClassName: "gp2"
        size: "200G"
      redundancyPolicy: "SingleRedundancy"
```

```
visualization:
  type: "kibana"
  kibana:
    resources: 2
      limits:
        memory: 1Gi
      requests:
        cpu: 500m
        memory: 1Gi
    proxy:
      resources: 3
        limits:
          memory: 100Mi
        requests:
          cpu: 100m
          memory: 100Mi
    replicas: 2
collection:
  logs:
    type: "fluentd"
    fluentd:
      resources: 4
        limits:
          memory: 736Mi
        requests:
          cpu: 200m
          memory: 736Mi
```

1. Specify the CPU and memory limits and requests for the log store as needed. For Elasticsearch, you must adjust both the request value and the limit value.

2 3. Specify the CPU and memory limits and requests for the log visualizer as needed.

4. Specify the CPU and memory limits and requests for the log collector as needed.

7.4.3.2. Scaling redundancy for the log visualizer nodes

You can scale the pod that hosts the log visualizer for redundancy.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc edit ClusterLogging instance
```

```
$ oc edit ClusterLogging instance

apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"

....
```

```
spec:
  visualization:
    type: "kibana"
    kibana:
      replicas: 1 (1)
```

(1) Specify the number of Kibana nodes.

CHAPTER 8. ACCESSING THE SERVICE LOGS FOR RED HAT OPENSHIFT SERVICE ON AWS CLUSTERS

You can view the service logs for your Red Hat OpenShift Service on AWS (ROSA) clusters by using the Red Hat OpenShift Cluster Manager. The service logs detail cluster events such as load balancer quota updates and scheduled maintenance upgrades. The logs also show cluster resource changes such as the addition or deletion of users, groups, and identity providers.

Additionally, you can add notification contacts for a ROSA cluster. Subscribed users receive emails about cluster events that require customer action, known cluster incidents, upgrade maintenance, and other topics.

8.1. VIEWING THE SERVICE LOGS BY USING OPENSHIFT CLUSTER MANAGER

You can view the service logs for a Red Hat OpenShift Service on AWS (ROSA) cluster by using Red Hat OpenShift Cluster Manager.

Prerequisites

- You have installed a ROSA cluster.

Procedure

1. Navigate to OpenShift Cluster Manager and select your cluster.
2. In the **Overview** page for your cluster, view the service logs in the **Cluster history** section.
3. Optional: Filter the cluster service logs by **Description** or **Severity** from the drop-down menu. You can filter further by entering a specific item in the search bar.
4. Optional: Click **Download history** to download the service logs for your cluster in JSON or CSV format.

8.2. ADDING CLUSTER NOTIFICATION CONTACTS

You can add notification contacts for your Red Hat OpenShift Service on AWS (ROSA) cluster. When an event occurs that triggers a cluster notification email, subscribed users are notified.

Procedure

1. Navigate to OpenShift Cluster Manager and select your cluster.
2. On the **Support** tab, under the **Notification contacts** heading, click **Add notification contact**.
3. Enter the Red Hat username or email of the contact you want to add.

NOTE

The username or email address must relate to a user account in the Red Hat organization where the cluster is deployed.

4. Click **Add contact**.

Verification

- You see a confirmation message when you have successfully added the contact. The user appears under the **Notification contacts** heading on the **Support** tab.

CHAPTER 9. VIEWING CLUSTER LOGS IN THE AWS CONSOLE

You can view forwarded cluster logs in the AWS console.

9.1. VIEWING FORWARDED LOGS

Logs that are being forwarded from Red Hat OpenShift Service on AWS are viewed in the Amazon Web Services (AWS) console.

Prerequisites

- The **cluster-logging-operator** add-on service is installed and **Cloudwatch** is enabled.

Procedure

1. Log in to the AWS console.
2. Select the region the cluster is deployed in.
3. Select the **CloudWatch** service.
4. Select **Logs** from the left column, and select **Log Groups**.
5. Select a log group to explore. You can view application, infrastructure, or audit logs, depending on which types were enabled during the add-on service installation. See the Amazon CloudWatch User Guide for more information.

CHAPTER 10. CONFIGURING YOUR LOGGING DEPLOYMENT

10.1. CONFIGURING CPU AND MEMORY LIMITS FOR LOGGING COMPONENTS

You can configure both the CPU and memory limits for each of the logging components as needed.

10.1.1. Configuring CPU and memory limits

The logging components allow for adjustments to both the CPU and memory limits.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc -n openshift-logging edit ClusterLogging instance
```

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: openshift-logging

...

spec:
  managementState: "Managed"
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      resources: (1)
        limits:
          memory: 16Gi
        requests:
          cpu: 200m
          memory: 16Gi
      storage:
        storageClassName: "gp2"
        size: "200G"
      redundancyPolicy: "SingleRedundancy"
  visualization:
    type: "kibana"
    kibana:
      resources: (2)
        limits:
          memory: 1Gi
        requests:
          cpu: 500m
          memory: 1Gi
      proxy:
        resources: (3)
          limits:
```

```
          memory: 100Mi
        requests:
          cpu: 100m
          memory: 100Mi
    replicas: 2
collection:
  logs:
    type: "fluentd"
    fluentd:
      resources: 4
        limits:
          memory: 736Mi
        requests:
          cpu: 200m
          memory: 736Mi
```

1. Specify the CPU and memory limits and requests for the log store as needed. For Elasticsearch, you must adjust both the request value and the limit value.

2 3. Specify the CPU and memory limits and requests for the log visualizer as needed.

4. Specify the CPU and memory limits and requests for the log collector as needed.

CHAPTER 11. LOG COLLECTION AND FORWARDING

11.1. ABOUT LOG COLLECTION AND FORWARDING

The Red Hat OpenShift Logging Operator deploys a collector based on the **ClusterLogForwarder** resource specification. There are two collector options supported by this Operator: the legacy Fluentd collector, and the Vector collector.

NOTE

Fluentd is deprecated and is planned to be removed in a future release. Red Hat provides bug fixes and support for this feature during the current release lifecycle, but this feature no longer receives enhancements. As an alternative to Fluentd, you can use Vector instead.

11.1.1. Log collection

The log collector is a daemon set that deploys pods to each Red Hat OpenShift Service on AWS node to collect container and node logs.

By default, the log collector uses the following sources:

- System and infrastructure logs generated by journald log messages from the operating system, the container runtime, and Red Hat OpenShift Service on AWS.
- **/var/log/containers/*.log** for all container logs.

If you configure the log collector to collect audit logs, it collects them from **/var/log/audit/audit.log**.

The log collector collects the logs from these sources and forwards them internally or externally depending on your logging configuration.

11.1.1.1. Log collector types

Vector is a log collector offered as an alternative to Fluentd for the logging.

You can configure which logging collector type your cluster uses by modifying the **ClusterLogging** custom resource (CR) **collection** spec:

Example ClusterLogging CR that configures Vector as the collector

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
spec:
  collection:
    logs:
      type: vector
      vector: {}
# ...
```

11.1.1.2. Log collection limitations

The container runtimes provide minimal information to identify the source of log messages: project, pod name, and container ID. This information is not sufficient to uniquely identify the source of the logs. If a pod with a given name and project is deleted before the log collector begins processing its logs, information from the API server, such as labels and annotations, might not be available. There might not be a way to distinguish the log messages from a similarly named pod and project or trace the logs to their source. This limitation means that log collection and normalization are considered *best effort*.

IMPORTANT

The available container runtimes provide minimal information to identify the source of log messages and do not guarantee unique individual log messages or that these messages can be traced to their source.

11.1.1.3. Log collector features by type

Table 11.1. Log Sources

Feature	Fluentd	Vector
App container logs	✓	✓
App-specific routing	✓	✓
App-specific routing by namespace	✓	✓
Infra container logs	✓	✓
Infra journal logs	✓	✓
Kube API audit logs	✓	✓
OpenShift API audit logs	✓	✓
Open Virtual Network (OVN) audit logs	✓	✓

Table 11.2. Authorization and Authentication

Feature	Fluentd	Vector
Elasticsearch certificates	✓	✓
Elasticsearch username / password	✓	✓
Amazon Cloudwatch keys	✓	✓

Feature	Fluentd	Vector
Amazon Cloudwatch STS	✓	✓
Kafka certificates	✓	✓
Kafka username / password	✓	✓
Kafka SASL	✓	✓
Loki bearer token	✓	✓

Table 11.3. Normalizations and Transformations

Feature	Fluentd	Vector
Viaq data model - app	✓	✓
Viaq data model - infra	✓	✓
Viaq data model - infra(journal)	✓	✓
Viaq data model - Linux audit	✓	✓
Viaq data model - kube-apiserver audit	✓	✓
Viaq data model - OpenShift API audit	✓	✓
Viaq data model - OVN	✓	✓
Loglevel Normalization	✓	✓
JSON parsing	✓	✓
Structured Index	✓	✓
Multiline error detection	✓	✓
Multicontainer / split indices	✓	✓
Flatten labels	✓	✓
CLF static labels	✓	✓

Table 11.4. Tuning

Feature	Fluentd	Vector
Fluentd readlinelimit	✓	
Fluentd buffer	✓	
- chunklimitsize	✓	
- totallimitsize	✓	
- overflowaction	✓	
- flushthreadcount	✓	
- flushmode	✓	
- flushinterval	✓	
- retrywait	✓	
- retrytype	✓	
- retrymaxinterval	✓	
- retrytimeout	✓	

Table 11.5. Visibility

Feature	Fluentd	Vector
Metrics	✓	✓
Dashboard	✓	✓
Alerts	✓	✓

Table 11.6. Miscellaneous

Feature	Fluentd	Vector
Global proxy support	✓	✓
x86 support	✓	✓
ARM support	✓	✓

Feature	Fluentd	Vector
IBM Power® support	✓	✓
IBM Z® support	✓	✓
IPv6 support	✓	✓
Log event buffering	✓	
Disconnected Cluster	✓	✓

11.1.1.4. Collector outputs

The following collector outputs are supported:

Table 11.7. Supported outputs

Feature	Fluentd	Vector
Elasticsearch v6-v8	✓	✓
Fluent forward	✓	
Syslog RFC3164	✓	✓ (Logging 5.7+)
Syslog RFC5424	✓	✓ (Logging 5.7+)
Kafka	✓	✓
Amazon Cloudwatch	✓	✓
Amazon Cloudwatch STS	✓	✓
Loki	✓	✓
HTTP	✓	✓ (Logging 5.7+)
Google Cloud Logging	✓	✓
Splunk		✓ (Logging 5.6+)

11.1.2. Log forwarding

Administrators can create **ClusterLogForwarder** resources that specify which logs are collected, how they are transformed, and where they are forwarded to.

ClusterLogForwarder resources can be used up to forward container, infrastructure, and audit logs to specific endpoints within or outside of a cluster. Transport Layer Security (TLS) is supported so that log forwarders can be configured to send logs securely.

Administrators can also authorize RBAC permissions that define which service accounts and users can access and forward which types of logs.

11.1.2.1. Log forwarding implementations

There are two log forwarding implementations available: the legacy implementation, and the multi log forwarder feature.

IMPORTANT

Only the Vector collector is supported for use with the multi log forwarder feature. The Fluentd collector can only be used with legacy implementations.

11.1.2.1.1. Legacy implementation

In legacy implementations, you can only use one log forwarder in your cluster. The **ClusterLogForwarder** resource in this mode must be named **instance**, and must be created in the **openshift-logging** namespace. The **ClusterLogForwarder** resource also requires a corresponding **ClusterLogging** resource named **instance** in the **openshift-logging** namespace.

11.1.2.1.2. Multi log forwarder feature

The multi log forwarder feature is available in logging 5.8 and later, and provides the following functionality:

- Administrators can control which users are allowed to define log collection and which logs they are allowed to collect.
- Users who have the required permissions are able to specify additional log collection configurations.
- Administrators who are migrating from the deprecated Fluentd collector to the Vector collector can deploy a new log forwarder separately from their existing deployment. The existing and new log forwarders can operate simultaneously while workloads are being migrated.

In multi log forwarder implementations, you are not required to create a corresponding **ClusterLogging** resource for your **ClusterLogForwarder** resource. You can create multiple **ClusterLogForwarder** resources using any name, in any namespace, with the following exceptions:

- You cannot create a **ClusterLogForwarder** resource named **instance** in the **openshift-logging** namespace, because this is reserved for a log forwarder that supports the legacy workflow using the Fluentd collector.
- You cannot create a **ClusterLogForwarder** resource named **collector** in the **openshift-logging** namespace, because this is reserved for the collector.

11.1.2.2. Enabling the multi log forwarder feature for a cluster

To use the multi log forwarder feature, you must create a service account and cluster role bindings for that service account. You can then reference the service account in the **ClusterLogForwarder** resource to control access permissions.

IMPORTANT

In order to support multi log forwarding in additional namespaces other than the **openshift-logging** namespace, you must update the Red Hat OpenShift Logging Operator to watch all namespaces. This functionality is supported by default in new Red Hat OpenShift Logging Operator version 5.8 installations.

11.1.2.2.1. Authorizing log collection RBAC permissions

In logging 5.8 and later, 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.

You can authorize RBAC permissions for log collection by binding the required cluster roles to a service account.

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>
   ```

Additional resources

- Using RBAC Authorization Kubernetes documentation

11.2. LOG OUTPUT TYPES

Outputs define the destination where logs are sent to from a log forwarder. You can configure multiple types of outputs in the **ClusterLogForwarder** custom resource (CR) to send logs to servers that support different protocols.

11.2.1. Supported log forwarding outputs

Outputs can be any of the following types:

Table 11.8. Supported log output types

Output type	Protocol	Tested with	Logging versions	Supported collector type
Elasticsearch v6	HTTP 1.1	6.8.1, 6.8.23	5.6+	Fluentd, Vector
Elasticsearch v7	HTTP 1.1	7.12.2, 7.17.7, 7.10.1	5.6+	Fluentd, Vector
Elasticsearch v8	HTTP 1.1	8.4.3, 8.6.1	5.6+	Fluentd [1], Vector
Fluent Forward	Fluentd forward v1	Fluentd 1.14.6, Logstash 7.10.1, Fluentd 1.14.5	5.4+	Fluentd
Google Cloud Logging	REST over HTTPS	Latest	5.7+	Vector
HTTP	HTTP 1.1	Fluentd 1.14.6, Vector 0.21	5.7+	Fluentd, Vector
Kafka	Kafka 0.11	Kafka 2.4.1, 2.7.0, 3.3.1	5.4+	Fluentd, Vector
Loki	REST over HTTP and HTTPS	2.3.0, 2.5.0, 2.7, 2.2.1	5.4+	Fluentd, Vector
Splunk	HEC	8.2.9, 9.0.0	5.7+	Vector
Syslog	RFC3164, RFC5424	Rsyslog 8.37.0-9.el7, rsyslog-8.39.0	5.4+	Fluentd, Vector [2]
Amazon CloudWatch	REST over HTTPS	Latest	5.4+	Fluentd, Vector

1. Fluentd does not support Elasticsearch 8 in the logging version 5.6.2.
2. Vector supports Syslog in the logging version 5.7 and higher.

11.2.2. Output type descriptions

default

The on-cluster, Red Hat managed log store. You are not required to configure the default output.

NOTE

If you configure a **default** output, you receive an error message, because the **default** output name is reserved for referencing the on-cluster, Red Hat managed log store.

loki

Loki, a horizontally scalable, highly available, multi-tenant log aggregation system.

kafka

A Kafka broker. The **kafka** output can use a TCP or TLS connection.

elasticsearch

An external Elasticsearch instance. The **elasticsearch** output can use a TLS connection.

fluentdForward

An external log aggregation solution that supports Fluentd. This option uses the Fluentd **forward** protocols. The **fluentForward** output can use a TCP or TLS connection and supports shared-key authentication by providing a **shared_key** field in a secret. Shared-key authentication can be used with or without TLS.

> **IMPORTANT**
>
> The **fluentdForward** output is only supported if you are using the Fluentd collector. It is not supported if you are using the Vector collector. If you are using the Vector collector, you can forward logs to Fluentd by using the **http** output.

syslog

An external log aggregation solution that supports the syslog RFC3164 or RFC5424 protocols. The **syslog** output can use a UDP, TCP, or TLS connection.

cloudwatch

Amazon CloudWatch, a monitoring and log storage service hosted by Amazon Web Services (AWS).

11.3. ENABLING JSON LOG FORWARDING

You can configure the Log Forwarding API to parse JSON strings into a structured object.

11.3.1. Parsing JSON logs

You can use a **ClusterLogForwarder** object to parse JSON logs into a structured object and forward them to a supported output.

To illustrate how this works, suppose that you have the following structured JSON log entry:

Example structured JSON log entry

```
{"level":"info","name":"fred","home":"bedrock"}
```

To enable parsing JSON log, you add **parse: json** to a pipeline in the **ClusterLogForwarder** CR, as shown in the following example:

Example snippet showing parse: json

```
pipelines:
- inputRefs: [ application ]
  outputRefs: myFluentd
  parse: json
```

When you enable parsing JSON logs by using **parse: json**, the CR copies the JSON-structured log entry in a **structured** field, as shown in the following example:

Example structured output containing the structured JSON log entry

```
{"structured": { "level": "info", "name": "fred", "home": "bedrock" },
 "more fields..."}
```

IMPORTANT

If the log entry does not contain valid structured JSON, the **structured** field is absent.

11.3.2. Configuring JSON log data for Elasticsearch

If your JSON logs follow more than one schema, storing them in a single index might cause type conflicts and cardinality problems. To avoid that, you must configure the **ClusterLogForwarder** custom resource (CR) to group each schema into a single output definition. This way, each schema is forwarded to a separate index.

IMPORTANT

If you forward JSON logs to the default Elasticsearch instance managed by OpenShift Logging, it generates new indices based on your configuration. To avoid performance issues associated with having too many indices, consider keeping the number of possible schemas low by standardizing to common schemas.

Structure types

You can use the following structure types in the **ClusterLogForwarder** CR to construct index names for the Elasticsearch log store:

- **structuredTypeKey** is the name of a message field. The value of that field is used to construct the index name.
  - **kubernetes.labels.<key>** is the Kubernetes pod label whose value is used to construct the index name.
  - **openshift.labels.<key>** is the **pipeline.label.<key>** element in the **ClusterLogForwarder** CR whose value is used to construct the index name.
  - **kubernetes.container_name** uses the container name to construct the index name.
- **structuredTypeName**: If the **structuredTypeKey** field is not set or its key is not present, the **structuredTypeName** value is used as the structured type. When you use both the **structuredTypeKey** field and the **structuredTypeName** field together, the **structuredTypeName** value provides a fallback index name if the key in the **structuredTypeKey** field is missing from the JSON log data.

NOTE

Although you can set the value of **structuredTypeKey** to any field shown in the "Log Record Fields" topic, the most useful fields are shown in the preceding list of structure types.

A structuredTypeKey: kubernetes.labels.<key> example

Suppose the following:

- Your cluster is running application pods that produce JSON logs in two different formats, "apache" and "google".
- The user labels these application pods with **logFormat=apache** and **logFormat=google**.
- You use the following snippet in your **ClusterLogForwarder** CR YAML file.

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
# ...
  outputDefaults:
    elasticsearch:
      structuredTypeKey: kubernetes.labels.logFormat (1)
      structuredTypeName: nologformat
  pipelines:
  - inputRefs:
    - application
    outputRefs:
    - default
    parse: json (2)
```

1. Uses the value of the key-value pair that is formed by the Kubernetes **logFormat** label.
2. Enables parsing JSON logs.

In that case, the following structured log record goes to the **app-apache-write** index:

```
{
  "structured":{"name":"fred","home":"bedrock"},
  "kubernetes":{"labels":{"logFormat": "apache", ...}}
}
```

And the following structured log record goes to the **app-google-write** index:

```
{
  "structured":{"name":"wilma","home":"bedrock"},
  "kubernetes":{"labels":{"logFormat": "google", ...}}
}
```

A structuredTypeKey: openshift.labels.<key> example

Suppose that you use the following snippet in your **ClusterLogForwarder** CR YAML file.

```
outputDefaults:
 elasticsearch:
    structuredTypeKey: openshift.labels.myLabel (1)
    structuredTypeName: nologformat
pipelines:
 - name: application-logs
   inputRefs:
   - application
```

```
- audit
outputRefs:
- elasticsearch-secure
- default
parse: json
labels:
  myLabel: myValue (2)
```

1. Uses the value of the key-value pair that is formed by the OpenShift **myLabel** label.
2. The **myLabel** element gives its string value, **myValue**, to the structured log record.

In that case, the following structured log record goes to the **app-myValue-write** index:

```
{
  "structured":{"name":"fred","home":"bedrock"},
  "openshift":{"labels":{"myLabel": "myValue", ...}}
}
```

Additional considerations

- The Elasticsearch *index* for structured records is formed by prepending "app-" to the structured type and appending "-write".
- Unstructured records are not sent to the structured index. They are indexed as usual in the application, infrastructure, or audit indices.
- If there is no non-empty structured type, forward an *unstructured* record with no **structured** field.

It is important not to overload Elasticsearch with too many indices. Only use distinct structured types for distinct log *formats*, **not** for each application or namespace. For example, most Apache applications use the same JSON log format and structured type, such as **LogApache**.

11.3.3. Forwarding JSON logs to the Elasticsearch log store

For an Elasticsearch log store, if your JSON log entries *follow different schemas*, configure the **ClusterLogForwarder** custom resource (CR) to group each JSON schema into a single output definition. This way, Elasticsearch uses a separate index for each schema.

IMPORTANT

Because forwarding different schemas to the same index can cause type conflicts and cardinality problems, you must perform this configuration before you forward data to the Elasticsearch store.

To avoid performance issues associated with having too many indices, consider keeping the number of possible schemas low by standardizing to common schemas.

Procedure

1. Add the following snippet to your **ClusterLogForwarder** CR YAML file.

```
outputDefaults:
```

```
 elasticsearch:
   structuredTypeKey: <log record field>
   structuredTypeName: <name>
pipelines:
- inputRefs:
  - application
  outputRefs: default
  parse: json
```

2. Use **structuredTypeKey** field to specify one of the log record fields.

3. Use **structuredTypeName** field to specify a name.

> **IMPORTANT**
>
> To parse JSON logs, you must set both the **structuredTypeKey** and **structuredTypeName** fields.

4. For **inputRefs**, specify which log types to forward by using that pipeline, such as **application, infrastructure**, or **audit**.

5. Add the **parse: json** element to pipelines.

6. Create the CR object:

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

The Red Hat OpenShift Logging Operator redeploys the collector pods. However, if they do not redeploy, delete the collector pods to force them to redeploy.

```
$ oc delete pod --selector logging-infra=collector
```

11.3.4. Forwarding JSON logs from containers in the same pod to separate indices

You can forward structured logs from different containers within the same pod to different indices. To use this feature, you must configure the pipeline with multi-container support and annotate the pods. Logs are written to indices with a prefix of **app-**. It is recommended that Elasticsearch be configured with aliases to accommodate this.

> **IMPORTANT**
>
> JSON formatting of logs varies by application. Because creating too many indices impacts performance, limit your use of this feature to creating indices for logs that have incompatible JSON formats. Use queries to separate logs from different namespaces, or applications with compatible JSON formats.

Prerequisites

- Logging for Red Hat OpenShift: 5.5

Procedure

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

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputDefaults:
    elasticsearch:
      structuredTypeKey: kubernetes.labels.logFormat (1)
      structuredTypeName: nologformat
      enableStructuredContainerLogs: true (2)
  pipelines:
  - inputRefs:
    - application
    name: application-logs
    outputRefs:
    - default
    parse: json
```

1. Uses the value of the key-value pair that is formed by the Kubernetes **logFormat** label.
2. Enables multi-container outputs.

2. Create or edit a YAML file that defines the **Pod** CR object:

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    containerType.logging.openshift.io/heavy: heavy (1)
    containerType.logging.openshift.io/low: low
spec:
  containers:
  - name: heavy (2)
    image: heavyimage
  - name: low
    image: lowimage
```

1. Format: **containerType.logging.openshift.io/<container-name>: <index>**
2. Annotation names must match container names

> **WARNING**
>
> This configuration might significantly increase the number of shards on the cluster.

Additional Resources

- Kubernetes Annotations

Additional resources

- About log forwarding

11.4. CONFIGURING LOG FORWARDING

In a logging deployment, container and infrastructure logs are forwarded to the internal log store defined in the **ClusterLogging** custom resource (CR) by default.

Audit logs are not forwarded to the internal log store by default because this does not provide secure storage. You are responsible for ensuring that the system to which you forward audit logs is compliant with your organizational and governmental regulations, and is properly secured.

If this default configuration meets your needs, you do not need to configure a **ClusterLogForwarder** CR. If a **ClusterLogForwarder** CR exists, logs are not forwarded to the internal log store unless a pipeline is defined that contains the **default** output.

11.4.1. About forwarding logs to third-party systems

To send logs to specific endpoints inside and outside your Red Hat OpenShift Service on AWS cluster, you specify a combination of *outputs* and *pipelines* in a **ClusterLogForwarder** custom resource (CR). You can also use *inputs* to forward the application logs associated with a specific project to an endpoint. Authentication is provided by a Kubernetes *Secret* object.

pipeline

Defines simple routing from one log type to one or more outputs, or which logs you want to send. The log types are one of the following:

- **application**. Container logs generated by user applications running in the cluster, except infrastructure container applications.
- **infrastructure**. Container logs from pods that run in the **openshift***, **kube***, or **default** projects and journal logs sourced from node file system.
- **audit**. Audit logs generated by the node audit system, **auditd**, Kubernetes API server, OpenShift API server, and OVN network.

You can add labels to outbound log messages by using **key:value** pairs in the pipeline. For example, you might add a label to messages that are forwarded to other data centers or label the logs by type. Labels that are added to objects are also forwarded with the log message.

input

Forwards the application logs associated with a specific project to a pipeline.
In the pipeline, you define which log types to forward using an **inputRef** parameter and where to forward the logs to using an **outputRef** parameter.

Secret

A **key:value map** that contains confidential data such as user credentials.

Note the following:

- If you do not define a pipeline for a log type, the logs of the undefined types are dropped. For example, if you specify a pipeline for the **application** and **audit** types, but do not specify a pipeline for the **infrastructure** type, **infrastructure** logs are dropped.
- You can use multiple types of outputs in the **ClusterLogForwarder** custom resource (CR) to send logs to servers that support different protocols.

The following example forwards the audit logs to a secure external Elasticsearch instance, the infrastructure logs to an insecure external Elasticsearch instance, the application logs to a Kafka broker, and the application logs from the **my-apps-logs** project to the internal Elasticsearch instance.

Sample log forwarding outputs and pipelines

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
   - name: elasticsearch-secure (4)
     type: "elasticsearch"
     url: https://elasticsearch.secure.com:9200
     secret:
        name: elasticsearch
   - name: elasticsearch-insecure (5)
     type: "elasticsearch"
     url: http://elasticsearch.insecure.com:9200
   - name: kafka-app (6)
     type: "kafka"
     url: tls://kafka.secure.com:9093/app-topic
  inputs: (7)
   - name: my-app-logs
     application:
        namespaces:
        - my-project
  pipelines:
   - name: audit-logs (8)
     inputRefs:
      - audit
     outputRefs:
      - elasticsearch-secure
      - default
     labels:
       secure: "true" (9)
       datacenter: "east"
   - name: infrastructure-logs (10)
     inputRefs:
      - infrastructure
     outputRefs:
      - elasticsearch-insecure
     labels:
       datacenter: "west"
   - name: my-app (11)
```

```
  inputRefs:
   - my-app-logs
  outputRefs:
   - default
- inputRefs: 12
   - application
  outputRefs:
   - kafka-app
  labels:
    datacenter: "south"
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.

2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.

3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.

4. Configuration for an secure Elasticsearch output using a secret with a secure URL.
   - A name to describe the output.
   - The type of output: **elasticsearch**.
   - The secure URL and port of the Elasticsearch instance as a valid absolute URL, including the prefix.
   - The secret required by the endpoint for TLS communication. The secret must exist in the **openshift-logging** project.

5. Configuration for an insecure Elasticsearch output:
   - A name to describe the output.
   - The type of output: **elasticsearch**.
   - The insecure URL and port of the Elasticsearch instance as a valid absolute URL, including the prefix.

6. Configuration for a Kafka output using a client-authenticated TLS communication over a secure URL:
   - A name to describe the output.
   - The type of output: **kafka**.
   - Specify the URL and port of the Kafka broker as a valid absolute URL, including the prefix.

7. Configuration for an input to filter application logs from the **my-project** namespace.

8. Configuration for a pipeline to send audit logs to the secure external Elasticsearch instance:
   - A name to describe the pipeline.
   - The **inputRefs** is the log type, in this example **audit**.

- The **outputRefs** is the name of the output to use, in this example **elasticsearch-secure** to forward to the secure Elasticsearch instance and **default** to forward to the internal Elasticsearch instance.
- Optional: Labels to add to the logs.

9 Optional: String. One or more labels to add to the logs. Quote values like "true" so they are recognized as string values, not as a boolean.

10 Configuration for a pipeline to send infrastructure logs to the insecure external Elasticsearch instance.

11 Configuration for a pipeline to send logs from the **my-project** project to the internal Elasticsearch instance.

- A name to describe the pipeline.
- The **inputRefs** is a specific input: **my-app-logs**.
- The **outputRefs** is **default**.
- Optional: String. One or more labels to add to the logs.

12 Configuration for a pipeline to send logs to the Kafka broker, with no pipeline name:

- The **inputRefs** is the log type, in this example **application**.
- The **outputRefs** is the name of the output to use.
- Optional: String. One or more labels to add to the logs.

Fluentd log handling when the external log aggregator is unavailable

If your external logging aggregator becomes unavailable and cannot receive logs, Fluentd continues to collect logs and stores them in a buffer. When the log aggregator becomes available, log forwarding resumes, including the buffered logs. If the buffer fills completely, Fluentd stops collecting logs. Red Hat OpenShift Service on AWS rotates the logs and deletes them. You cannot adjust the buffer size or add a persistent volume claim (PVC) to the Fluentd daemon set or pods.

Supported Authorization Keys

Common key types are provided here. Some output types support additional specialized keys, documented with the output-specific configuration field. All secret keys are optional. Enable the security features you want by setting the relevant keys. You are responsible for creating and maintaining any additional configurations that external destinations might require, such as keys and secrets, service accounts, port openings, or global proxy configuration. Open Shift Logging will not attempt to verify a mismatch between authorization combinations.

Transport Layer Security (TLS)

Using a TLS URL (**http://...** or **ssl://...**) without a secret enables basic TLS server-side authentication. Additional TLS features are enabled by including a secret and setting the following optional fields:

- **passphrase**: (string) Passphrase to decode an encoded TLS private key. Requires **tls.key**.
- **ca-bundle.crt**: (string) File name of a customer CA for server authentication.

Username and Password

- **username**: (string) Authentication user name. Requires **password**.

- **password**: (string) Authentication password. Requires **username**.

Simple Authentication Security Layer (SASL)

- **sasl.enable** (boolean) Explicitly enable or disable SASL. If missing, SASL is automatically enabled when any of the other **sasl.** keys are set.
- **sasl.mechanisms**: (array) List of allowed SASL mechanism names. If missing or empty, the system defaults are used.
- **sasl.allow-insecure**: (boolean) Allow mechanisms that send clear-text passwords. Defaults to false.

11.4.1.1. Creating a Secret

You can create a secret in the directory that contains your certificate and key files by using the following command:

```
$ oc create secret generic -n <namespace> <secret_name> \
  --from-file=ca-bundle.crt=<your_bundle_file> \
  --from-literal=username=<your_username> \
  --from-literal=password=<your_password>
```

NOTE

Generic or opaque secrets are recommended for best results.

11.4.2. Creating a log forwarder

To create a log forwarder, you must create a **ClusterLogForwarder** CR that specifies the log input types that the service account can collect. You can also specify which outputs the logs can be forwarded to. If you are using the multi log forwarder feature, you must also reference the service account in the **ClusterLogForwarder** CR.

If you are using the multi log forwarder feature on your cluster, you can create **ClusterLogForwarder** custom resources (CRs) in any namespace, using any name. If you are using a legacy implementation, the **ClusterLogForwarder** CR must be named **instance**, and must be created in the **openshift-logging** namespace.

IMPORTANT

You need administrator permissions for the namespace where you create the **ClusterLogForwarder** CR.

ClusterLogForwarder resource example

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
```

```
spec:
  serviceAccountName: <service_account_name> (3)
  pipelines:
   - inputRefs:
     - <log_type> (4)
     outputRefs:
     - <output_name> (5)
  outputs:
  - name: <output_name> (6)
    type: <output_type> (7)
    url: <log_output_url> (8)
# ...
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.
2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. 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.
5. 7. The type of output that you want to forward logs to. The value of this field can be **default**, **loki**, **kafka**, **elasticsearch**, **fluentdForward**, **syslog**, or **cloudwatch**.

> **NOTE**
>
> The **default** output type is not supported in mutli log forwarder implementations.

6. A name for the output that you want to forward logs to.
8. The URL of the output that you want to forward logs to.

11.4.3. Tuning log payloads and delivery

In logging 5.9 and newer versions, the **tuning** spec in the **ClusterLogForwarder** custom resource (CR) provides a means of configuring your deployment to prioritize either throughput or durability of logs.

For example, if you need to reduce the possibility of log loss when the collector restarts, or you require collected log messages to survive a collector restart to support regulatory mandates, you can tune your deployment to prioritize log durability. If you use outputs that have hard limitations on the size of batches they can receive, you may want to tune your deployment to prioritize log throughput.

> **IMPORTANT**
>
> To use this feature, your logging deployment must be configured to use the Vector collector. The **tuning** spec in the **ClusterLogForwarder** CR is not supported when using the Fluentd collector.

The following example shows the **ClusterLogForwarder** CR options that you can modify to tune log forwarder outputs:

Example ClusterLogForwarder CR tuning options

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  tuning:
    delivery: AtLeastOnce (1)
    compression: none (2)
    maxWrite: <integer> (3)
    minRetryDuration: 1s (4)
    maxRetryDuration: 1s (5)
# ...
```

1. Specify the delivery mode for log forwarding.
   - **AtLeastOnce** delivery means that if the log forwarder crashes or is restarted, any logs that were read before the crash but not sent to their destination are re-sent. It is possible that some logs are duplicated after a crash.
   - **AtMostOnce** delivery means that the log forwarder makes no effort to recover logs lost during a crash. This mode gives better throughput, but may result in greater log loss.
2. Specifying a **compression** configuration causes data to be compressed before it is sent over the network. Note that not all output types support compression, and if the specified compression type is not supported by the output, this results in an error. The possible values for this configuration are **none** for no compression, **gzip**, **snappy**, **zlib**, or **zstd**. **lz4** compression is also available if you are using a Kafka output. See the table "Supported compression types for tuning outputs" for more information.
3. Specifies a limit for the maximum payload of a single send operation to the output.
4. Specifies a minimum duration to wait between attempts before retrying delivery after a failure. This value is a string, and can be specified as milliseconds (**ms**), seconds (**s**), or minutes (**m**).
5. Specifies a maximum duration to wait between attempts before retrying delivery after a failure. This value is a string, and can be specified as milliseconds (**ms**), seconds (**s**), or minutes (**m**).

Table 11.9. Supported compression types for tuning outputs

Compression algorithm	Splunk	Amazon Cloudwatch	Elasticsearch 8	LokiStack	Apache Kafka	HTTP	Syslog	Google Cloud	Microsoft Azure Monitoring
gzip	X	X	X	X		X			

Compression algorithm	Splunk	Amazon Cloudwatch	Elasticsearch 8	LokiStack	Apache Kafka	HTTP	Syslog	Google Cloud	Microsoft Azure Monitoring
snappy		X		X	X	X			
zlib		X	X			X			
zstd		X			X	X			
lz4					X				

11.4.4. 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, with a value of **true**.

Example ClusterLogForwarder CR

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  pipelines:
```

```
    - name: my-app-logs
      inputRefs:
        - application
      outputRefs:
        - default
      detectMultilineErrors: true
```

11.4.4.1. 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.

Table 11.10. Supported languages per collector:

Language	Fluentd	Vector
Java	✓	✓
JS	✓	✓
Ruby	✓	✓
Python	✓	✓
Golang	✓	✓
PHP	✓	✓
Dart	✓	✓

11.4.4.2. Troubleshooting

When enabled, the collector configuration will include a new section with type: **detect_exceptions**

Example vector configuration section

```
[transforms.detect_exceptions_app-logs]
 type = "detect_exceptions"
 inputs = ["application"]
 languages = ["All"]
 group_by = ["kubernetes.namespace_name","kubernetes.pod_name","kubernetes.container_name"]
 expire_after_ms = 2000
 multiline_flush_interval_ms = 1000
```

Example fluentd config section

```
<label @MULTILINE_APP_LOGS>
  <match kubernetes.**>
    @type detect_exceptions
    remove_tag_prefix 'kubernetes'
```

```
    message message
    force_line_breaks true
    multiline_flush_interval .2
  </match>
</label>
```

11.4.5. Forwarding logs to Splunk

You can forward logs to the Splunk HTTP Event Collector (HEC) in addition to, or instead of, the internal default Red Hat OpenShift Service on AWS log store.

NOTE

Using this feature with Fluentd is not supported.

Prerequisites

- Red Hat OpenShift Logging Operator 5.6 or later
- A **ClusterLogging** instance with **vector** specified as the collector
- Base64 encoded Splunk HEC token

Procedure

1. Create a secret using your Base64 encoded Splunk HEC token.

```
$ oc -n openshift-logging create secret generic vector-splunk-secret --from-literal hecToken=
<HEC_Token>
```

2. Create or edit the **ClusterLogForwarder** Custom Resource (CR) using the template below:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
    - name: splunk-receiver (4)
      secret:
        name: vector-splunk-secret (5)
      type: splunk (6)
      url: <http://your.splunk.hec.url:8088> (7)
  pipelines: (8)
    - inputRefs:
        - application
        - infrastructure
      name: (9)
      outputRefs:
        - splunk-receiver (10)
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.
2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. Specify a name for the output.
5. Specify the name of the secret that contains your HEC token.
6. Specify the output type as **splunk**.
7. Specify the URL (including port) of your Splunk HEC.
8. Specify which log types to forward by using the pipeline: **application**, **infrastructure**, or **audit**.
9. Optional: Specify a name for the pipeline.
10. Specify the name of the output to use when forwarding logs with this pipeline.

11.4.6. Forwarding logs over HTTP

Forwarding logs over HTTP is supported for both the Fluentd and Vector log collectors. To enable, 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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
    - name: httpout-app
      type: http
      url: (4)
      http:
        headers: (5)
          h1: v1
          h2: v2
        method: POST
      secret:
        name: (6)
```

```
    tls:
      insecureSkipVerify: 7
pipelines:
  - name:
    inputRefs:
      - application
    outputRefs:
      - 8
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.
2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. Destination address for logs.
5. Additional headers to send with the log record.
6. Secret name for destination credentials.
7. Values are either **true** or **false**.
8. This value should be the same as the output name.

11.4.7. Forwarding to Azure Monitor Logs

With logging 5.9 and later, you can forward logs to Azure Monitor Logs in addition to, or instead of, the default log store. This functionality is provided by the Vector Azure Monitor Logs sink.

Prerequisites

- You are familiar with how to administer and create a **ClusterLogging** custom resource (CR) instance.
- You are familiar with how to administer and create a **ClusterLogForwarder** CR instance.
- You understand the **ClusterLogForwarder** CR specifications.
- You have basic familiarity with Azure services.
- You have an Azure account configured for Azure Portal or Azure CLI access.
- You have obtained your Azure Monitor Logs primary or the secondary security key.
- You have determined which log types to forward.

To enable log forwarding to Azure Monitor Logs via the HTTP Data Collector API:

Create a secret with your shared key:

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
  namespace: openshift-logging
type: Opaque
data:
  shared_key: <your_shared_key> (1)
```

1. Must contain a primary or secondary key for the Log Analytics workspace making the request.

To obtain a shared key, you can use this command in Azure CLI:

```
Get-AzOperationalInsightsWorkspaceSharedKey -ResourceGroupName "<resource_name>" -Name
"<workspace_name>"
```

Create or edit your **ClusterLogForwarder** CR using the template matching your log selection.

Forward all logs

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogForwarder"
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
  - name: azure-monitor
    type: azureMonitor
    azureMonitor:
      customerId: my-customer-id (1)
      logType: my_log_type (2)
    secret:
       name: my-secret
  pipelines:
  - name: app-pipeline
    inputRefs:
    - application
    outputRefs:
    - azure-monitor
```

1. Unique identifier for the Log Analytics workspace. Required field.
2. Azure record type of the data being submitted. May only contain letters, numbers, and underscores (_), and may not exceed 100 characters.

Forward application and infrastructure logs

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogForwarder"
metadata:
  name: instance
```

```
  namespace: openshift-logging
spec:
  outputs:
  - name: azure-monitor-app
    type: azureMonitor
    azureMonitor:
      customerId: my-customer-id
      logType: application_log (1)
    secret:
      name: my-secret
  - name: azure-monitor-infra
    type: azureMonitor
    azureMonitor:
      customerId: my-customer-id
      logType: infra_log #
    secret:
      name: my-secret
  pipelines:
    - name: app-pipeline
      inputRefs:
      - application
      outputRefs:
      - azure-monitor-app
    - name: infra-pipeline
      inputRefs:
      - infrastructure
      outputRefs:
      - azure-monitor-infra
```

1. Azure record type of the data being submitted. May only contain letters, numbers, and underscores (_), and may not exceed 100 characters.

Advanced configuration options

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogForwarder"
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
  - name: azure-monitor
    type: azureMonitor
    azureMonitor:
      customerId: my-customer-id
      logType: my_log_type
      azureResourceId: "/subscriptions/111111111" (1)
      host: "ods.opinsights.azure.com" (2)
    secret:
       name: my-secret
  pipelines:
   - name: app-pipeline
    inputRefs:
```

```
- application
outputRefs:
- azure-monitor
```

1. Resource ID of the Azure resource the data should be associated with. Optional field.
2. Alternative host for dedicated Azure regions. Optional field. Default value is **ods.opinsights.azure.com**. Default value for Azure Government is **ods.opinsights.azure.us**.

11.4.8. Forwarding application logs from specific projects

You can forward a copy of the application logs from specific projects to an external log aggregator, in addition to, or instead of, using the internal log store. You must also configure the external log aggregator to receive log data from Red Hat OpenShift Service on AWS.

To configure forwarding application logs from a project, you must create a **ClusterLogForwarder** custom resource (CR) with at least one input from a project, optional outputs for other log aggregators, and pipelines that use those inputs and outputs.

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:

Example ClusterLogForwarder CR

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance (1)
  namespace: openshift-logging (2)
spec:
  outputs:
   - name: fluentd-server-secure (3)
     type: fluentdForward (4)
     url: 'tls://fluentdserver.security.example.com:24224' (5)
     secret: (6)
        name: fluentd-secret
   - name: fluentd-server-insecure
     type: fluentdForward
     url: 'tcp://fluentdserver.home.example.com:24224'
  inputs: (7)
   - name: my-app-logs
     application:
        namespaces:
        - my-project (8)
  pipelines:
   - name: forward-to-fluentd-insecure (9)
     inputRefs: (10)
```

```
    - my-app-logs
    outputRefs: (11)
    - fluentd-server-insecure
    labels:
      project: "my-project" (12)
  - name: forward-to-fluentd-secure (13)
    inputRefs:
    - application (14)
    - audit
    - infrastructure
    outputRefs:
    - fluentd-server-secure
    - default
    labels:
      clusterId: "C1234"
```

1. The name of the **ClusterLogForwarder** CR must be **instance**.
2. The namespace for the **ClusterLogForwarder** CR must be **openshift-logging**.
3. The name of the output.
4. The output type: **elasticsearch**, **fluentdForward**, **syslog**, or **kafka**.
5. The URL and port of the external log aggregator as a valid absolute URL. If the cluster-wide proxy using the CIDR annotation is enabled, the output must be a server name or FQDN, not an IP address.
6. If using a **tls** prefix, you must specify the name of the secret required by the endpoint for TLS communication. The secret must exist in the **openshift-logging** project and have **tls.crt**, **tls.key**, and **ca-bundle.crt** keys that each point to the certificates they represent.
7. The configuration for an input to filter application logs from the specified projects.
8. If no namespace is specified, logs are collected from all namespaces.
9. The pipeline configuration directs logs from a named input to a named output. In this example, a pipeline named **forward-to-fluentd-insecure** forwards logs from an input named **my-app-logs** to an output named **fluentd-server-insecure**.
10. A list of inputs.
11. The name of the output to use.
12. Optional: String. One or more labels to add to the logs.
13. Configuration for a pipeline to send logs to other log aggregators.
    - Optional: Specify a name for the pipeline.
    - Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.
    - Specify the name of the output to use when forwarding logs with this pipeline.
    - Optional: Specify the **default** output to forward logs to the default log store.

- Optional: String. One or more labels to add to the logs.

14 Note that application logs from all namespaces are collected when using this configuration.

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

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

11.4.9. Forwarding application logs from specific pods

As a cluster administrator, you can use Kubernetes pod labels to gather log data from specific pods and forward it to a log collector.

Suppose that you have an application composed of pods running alongside other pods in various namespaces. If those pods have labels that identify the application, you can gather and output their log data to a specific log collector.

To specify the pod labels, you use one or more **matchLabels** key-value pairs. If you specify multiple key-value pairs, the pods must match all of them to be selected.

Procedure

1. Create or edit a YAML file that defines the **ClusterLogForwarder** CR object. In the file, specify the pod labels using simple equality-based selectors under **inputs[].name.application.selector.matchLabels**, as shown in the following example.

Example ClusterLogForwarder CR YAML file

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  pipelines:
    - inputRefs: [ myAppLogData ] (3)
      outputRefs: [ default ] (4)
  inputs: (5)
    - name: myAppLogData
      application:
        selector:
          matchLabels: (6)
            environment: production
            app: nginx
        namespaces: (7)
        - app1
        - app2
  outputs: (8)
    - <output_name>
    ...
```

1 In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.

2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. Specify one or more comma-separated values from **inputs[].name**.
4. Specify one or more comma-separated values from **outputs[]**.
5. Define a unique **inputs[].name** for each application that has a unique set of pod labels.
6. Specify the key-value pairs of pod labels whose log data you want to gather. You must specify both a key and value, not just a key. To be selected, the pods must match all the key-value pairs.
7. Optional: Specify one or more namespaces.
8. Specify one or more outputs to forward your log data to.

2. Optional: To restrict the gathering of log data to specific namespaces, use **inputs[].name.application.namespaces**, as shown in the preceding example.
3. Optional: You can send log data from additional applications that have different pod labels to the same pipeline.
   a. For each unique combination of pod labels, create an additional **inputs[].name** section similar to the one shown.
   b. Update the **selectors** to match the pod labels of this application.
   c. Add the new **inputs[].name** value to **inputRefs**. For example:

```
- inputRefs: [ myAppLogData, myOtherAppLogData ]
```

4. Create the CR object:

```
$ oc create -f <file-name>.yaml
```

Additional resources

- For more information on **matchLabels** in Kubernetes, see Resources that support set-based requirements.

11.4.10. 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, checking stops at the first match. How much data 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.

In logging 5.8 and later, 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, 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**, **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, a list of HTTP status code 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 Red Hat OpenShift Service on AWS 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

The 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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  pipelines:
    - name: my-pipeline
      inputRefs: audit (1)
      filterRefs: my-policy (2)
```

```
    outputRefs: default
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.

Additional resources

- Logging for egress firewall and network policy rules

11.4.11. Forwarding logs to an external Loki logging system

You can forward logs to an external Loki logging system in addition to, or instead of, the default log store.

To configure log forwarding to Loki, you must create a **ClusterLogForwarder** custom resource (CR) with an output to Loki, and a pipeline that uses the output. The output to Loki can use the HTTP (insecure) or HTTPS (secure HTTP) connection.

Prerequisites

- You must have a Loki logging system running at the URL you specify with the **url** field in the CR.

Procedure

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

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
  - name: loki-insecure (4)
    type: "loki" (5)
    url: http://loki.insecure.com:3100 (6)
    loki:
      tenantKey: kubernetes.namespace_name
      labelKeys:
```

```
    - kubernetes.labels.foo
  - name: loki-secure (7)
    type: "loki"
    url: https://loki.secure.com:3100
    secret:
      name: loki-secret (8)
    loki:
      tenantKey: kubernetes.namespace_name (9)
      labelKeys:
      - kubernetes.labels.foo (10)
pipelines:
  - name: application-logs (11)
    inputRefs: (12)
    - application
    - audit
    outputRefs: (13)
    - loki-secure
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.
2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. Specify a name for the output.
5. Specify the type as **"loki"**.
6. Specify the URL and port of the Loki system as a valid absolute URL. You can use the **http** (insecure) or **https** (secure HTTP) protocol. If the cluster-wide proxy using the CIDR annotation is enabled, the output must be a server name or FQDN, not an IP Address. Loki's default port for HTTP(S) communication is 3100.
7. For a secure connection, you can specify an **https** or **http** URL that you authenticate by specifying a **secret**.
8. For an **https** prefix, specify the name of the secret required by the endpoint for TLS communication. The secret must contain a **ca-bundle.crt** key that points to the certificates it represents. Otherwise, for **http** and **https** prefixes, you can specify a secret that contains a username and password. In legacy implementations, the secret must exist in the **openshift-logging** project. For more information, see the following "Example: Setting a secret that contains a username and password."
9. Optional: Specify a metadata key field to generate values for the **TenantID** field in Loki. For example, setting **tenantKey: kubernetes.namespace_name** uses the names of the Kubernetes namespaces as values for tenant IDs in Loki. To see which other log record fields you can specify, see the "Log Record Fields" link in the following "Additional resources" section.
10. Optional: Specify a list of metadata field keys to replace the default Loki labels. Loki label names must match the regular expression **[a-zA-Z_:][a-zA-Z0-9_:]***. Illegal characters in metadata keys are replaced with _ to form the label name. For example, the

kubernetes.labels.foo metadata key becomes Loki label **kubernetes_labels_foo**. If you do not set **labelKeys**, the default value is: **[log_type, kubernetes.namespace_name, kubernetes.pod_name, kubernetes_host]**. Keep the set of labels small because Loki limits the size and number of labels allowed. See Configuring Loki, limits_config. You can still query based on any log record field using query filters.

11 Optional: Specify a name for the pipeline.

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

13 Specify the name of the output to use when forwarding logs with this pipeline.

> **NOTE**
>
> Because Loki requires log streams to be correctly ordered by timestamp, **labelKeys** always includes the **kubernetes_host** label set, even if you do not specify it. This inclusion ensures that each stream originates from a single host, which prevents timestamps from becoming disordered due to clock differences on different hosts.

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

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

Additional resources

- Configuring Loki server

11.4.12. Forwarding logs to an external Elasticsearch instance

You can forward logs to an external Elasticsearch instance in addition to, or instead of, the internal log store. You are responsible for configuring the external log aggregator to receive log data from Red Hat OpenShift Service on AWS.

To configure log forwarding to an external Elasticsearch instance, you must create a **ClusterLogForwarder** custom resource (CR) with an output to that instance, and a pipeline that uses the output. The external Elasticsearch output can use the HTTP (insecure) or HTTPS (secure HTTP) connection.

To forward logs to both an external and the internal Elasticsearch instance, create outputs and pipelines to the external instance and a pipeline that uses the **default** output to forward logs to the internal instance.

> **NOTE**
>
> If you only want to forward logs to an internal Elasticsearch instance, you do not need to create a **ClusterLogForwarder** CR.

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:

Example ClusterLogForwarder CR

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
   - name: elasticsearch-example (4)
     type: elasticsearch (5)
     elasticsearch:
       version: 8 (6)
     url: http://elasticsearch.example.com:9200 (7)
     secret:
       name: es-secret (8)
  pipelines:
   - name: application-logs (9)
     inputRefs: (10)
     - application
     - audit
     outputRefs:
     - elasticsearch-example (11)
     - default (12)
     labels:
       myLabel: "myValue" (13)
# ...
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.
2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. Specify a name for the output.
5. Specify the **elasticsearch** type.
6. Specify the Elasticsearch version. This can be **6**, **7**, or **8**.
7. Specify the URL and port of the external Elasticsearch instance as a valid absolute URL. You can use the **http** (insecure) or **https** (secure HTTP) protocol. If the cluster-wide proxy using the CIDR annotation is enabled, the output must be a server name or FQDN, not an IP Address.

8. For an **https** prefix, specify the name of the secret required by the endpoint for TLS communication. The secret must contain a **ca-bundle.crt** key that points to the certificate
9. Optional: Specify a name for the pipeline.
10. Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.
11. Specify the name of the output to use when forwarding logs with this pipeline.
12. Optional: Specify the **default** output to send the logs to the internal Elasticsearch instance.
13. Optional: String. One or more labels to add to the logs.

2. Apply the **ClusterLogForwarder** CR:

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

Example: Setting a secret that contains a username and password

You can use a secret that contains a username and password to authenticate a secure connection to an external Elasticsearch instance.

For example, if you cannot use mutual TLS (mTLS) keys because a third party operates the Elasticsearch instance, you can use HTTP or HTTPS and set a secret that contains the username and password.

1. Create a **Secret** YAML file similar to the following example. Use base64-encoded values for the **username** and **password** fields. The secret type is opaque by default.

```
apiVersion: v1
kind: Secret
metadata:
  name: openshift-test-secret
data:
  username: <username>
  password: <password>
# ...
```

2. Create the secret:

```
$ oc create secret -n openshift-logging openshift-test-secret.yaml
```

3. Specify the name of the secret in the **ClusterLogForwarder** CR:

```
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
   - name: elasticsearch
     type: "elasticsearch"
     url: https://elasticsearch.secure.com:9200
```

```
    secret:
      name: openshift-test-secret
# ...
```

NOTE

In the value of the **url** field, the prefix can be **http** or **https**.

4. Apply the CR object:

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

11.4.13. Forwarding logs using the Fluentd forward protocol

You can use the Fluentd **forward** 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 to receive the logs from Red Hat OpenShift Service on AWS.

To configure log forwarding using the **forward** protocol, you must create a **ClusterLogForwarder** custom resource (CR) with one or more outputs to the Fluentd servers, and pipelines that use those outputs. The Fluentd output can use a TCP (insecure) or TLS (secure TCP) 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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance (1)
  namespace: openshift-logging (2)
spec:
  outputs:
   - name: fluentd-server-secure (3)
     type: fluentdForward (4)
     url: 'tls://fluentdserver.security.example.com:24224' (5)
     secret: (6)
       name: fluentd-secret
   - name: fluentd-server-insecure
     type: fluentdForward
     url: 'tcp://fluentdserver.home.example.com:24224'
  pipelines:
   - name: forward-to-fluentd-secure (7)
     inputRefs: (8)
     - application
     - audit
     outputRefs:
```

```
    - fluentd-server-secure (9)
    - default (10)
    labels:
      clusterId: "C1234" (11)
  - name: forward-to-fluentd-insecure (12)
    inputRefs:
    - infrastructure
    outputRefs:
    - fluentd-server-insecure
    labels:
      clusterId: "C1234"
```

1. The name of the **ClusterLogForwarder** CR must be **instance**.
2. The namespace for the **ClusterLogForwarder** CR must be **openshift-logging**.
3. Specify a name for the output.
4. Specify the **fluentdForward** type.
5. Specify the URL and port of the external Fluentd instance as a valid absolute URL. You can use the **tcp** (insecure) or **tls** (secure TCP) protocol. If the cluster-wide proxy using the CIDR annotation is enabled, the output must be a server name or FQDN, not an IP address.
6. If you are using a **tls** prefix, you must specify the name of the secret required by the endpoint for TLS communication. The secret must exist in the **openshift-logging** project and must contain a **ca-bundle.crt** key that points to the certificate it represents.
7. Optional: Specify a name for the pipeline.
8. Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.
9. Specify the name of the output to use when forwarding logs with this pipeline.
10. Optional: Specify the **default** output to forward logs to the internal Elasticsearch instance.
11. Optional: String. One or more labels to add to the logs.
12. Optional: Configure multiple outputs to forward logs to other external log aggregators of any supported type:
    - A name to describe the pipeline.
    - The **inputRefs** is the log type to forward by using the pipeline: **application, infrastructure**, or **audit**.
    - The **outputRefs** is the name of the output to use.
    - Optional: String. One or more labels to add to the logs.

2. Create the CR object:

```
$ oc create -f <file-name>.yaml
```

11.4.13.1. Enabling nanosecond precision for Logstash to ingest data from fluentd

For Logstash to ingest log data from fluentd, you must enable nanosecond precision in the Logstash configuration file.

Procedure

- In the Logstash configuration file, set **nanosecond_precision** to **true**.

Example Logstash configuration file

```
input { tcp { codec => fluent { nanosecond_precision => true } port => 24114 } }
filter { }
output { stdout { codec => rubydebug } }
```

11.4.14. 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 Red Hat OpenShift Service on AWS.

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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
   - name: rsyslog-east (4)
     type: syslog (5)
     syslog: (6)
       facility: local0
       rfc: RFC3164
       payloadKey: message
       severity: informational
     url: 'tls://rsyslogserver.east.example.com:514' (7)
     secret: (8)
        name: syslog-secret
```

```
  - name: rsyslog-west
    type: syslog
    syslog:
      appName: myapp
      facility: user
      msgID: mymsg
      procID: myproc
      rfc: RFC5424
      severity: debug
    url: 'tcp://rsyslogserver.west.example.com:514'
pipelines:
  - name: syslog-east 9
    inputRefs: 10
    - audit
    - application
    outputRefs: 11
    - rsyslog-east
    - default 12
    labels:
      secure: "true" 13
      syslog: "east"
  - name: syslog-west 14
    inputRefs:
    - infrastructure
    outputRefs:
    - rsyslog-west
    - default
    labels:
      syslog: "west"
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.
2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. Specify a name for the output.
5. Specify the **syslog** type.
6. Optional: Specify the syslog parameters, listed below.
7. Specify the URL and port of the external syslog instance. You can use the **udp** (insecure), **tcp** (insecure) or **tls** (secure TCP) protocol. If the cluster-wide proxy using the CIDR annotation is enabled, the output must be a server name or FQDN, not an IP address.
8. If using a **tls** prefix, you must specify the name of the secret required by the endpoint for TLS communication. The secret must contain a **ca-bundle.crt** key that points to the certificate it represents. In legacy implementations, the secret must exist in the **openshift-logging** project.
9. Optional: Specify a name for the pipeline.

10. Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.
11. Specify the name of the output to use when forwarding logs with this pipeline.
12. Optional: Specify the **default** output to forward logs to the internal Elasticsearch instance.
13. Optional: String. One or more labels to add to the logs. Quote values like "true" so they are recognized as string values, not as a boolean.
14. Optional: Configure multiple outputs to forward logs to other external log aggregators of any supported type:
    - A name to describe the pipeline.
    - The **inputRefs** is the log type to forward by using the pipeline: **application, infrastructure**, or **audit**.
    - The **outputRefs** is the name of the output to use.
    - Optional: String. One or more labels to add to the logs.

2. Create the CR object:

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

11.4.14.1. Adding log source information to message output

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

```
spec:
  outputs:
  - name: syslogout
    syslog:
      addLogSource: true
      facility: user
      payloadKey: message
      rfc: RFC3164
      severity: debug
      tag: mytag
    type: syslog
    url: tls://syslog-receiver.openshift-logging.svc:24224
  pipelines:
  - inputRefs:
    - application
    name: test-app
    outputRefs:
    - syslogout
```

NOTE

This configuration is compatible with both RFC3164 and RFC5424.

Example syslog message output without AddLogSource

```
<15>1 2020-11-15T17:06:14+00:00 fluentd-9hkb4 mytag - - - {"msgcontent"=>"Message Contents", "timestamp"=>"2020-11-15 17:06:09", "tag_key"=>"rec_tag", "index"=>56}
```

Example syslog message output with AddLogSource

```
<15>1 2020-11-16T10:49:37+00:00 crc-j55b9-master-0 mytag - - - namespace_name=clo-test-6327,pod_name=log-generator-ff9746c49-qxm7l,container_name=log-generator,message= {"msgcontent":"My life is my message", "timestamp":"2020-11-16 10:49:36", "tag_key":"rec_tag", "index":76}
```

11.4.14.2. Syslog parameters

You can configure the following for the **syslog** outputs. For more information, see the syslog RFC3164 or RFC5424 RFC.

- facility: The syslog facility. The value can be a decimal integer or a case-insensitive keyword:
  - **0** or **kern** for kernel messages
  - **1** or **user** for user-level messages, the default.
  - **2** or **mail** for the mail system
  - **3** or **daemon** for system daemons
  - **4** or **auth** for security/authentication messages
  - **5** or **syslog** for messages generated internally by syslogd
  - **6** or **lpr** for the line printer subsystem
  - **7** or **news** for the network news subsystem
  - **8** or **uucp** for the UUCP subsystem
  - **9** or **cron** for the clock daemon
  - **10** or **authpriv** for security authentication messages
  - **11** or **ftp** for the FTP daemon
  - **12** or **ntp** for the NTP subsystem
  - **13** or **security** for the syslog audit log
  - **14** or **console** for the syslog alert log
  - **15** or **solaris-cron** for the scheduling daemon
  - **16–23** or **local0** – **local7** for locally used facilities
- Optional: **payloadKey**: The record field to use as payload for the syslog message.

> **NOTE**
>
> Configuring the **payloadKey** parameter prevents other parameters from being forwarded to the syslog.

- rfc: The RFC to be used for sending logs using syslog. The default is RFC5424.
- severity: The syslog severity to set on outgoing syslog records. The value can be a decimal integer or a case-insensitive keyword:
  - **0** or **Emergency** for messages indicating the system is unusable
  - **1** or **Alert** for messages indicating action must be taken immediately
  - **2** or **Critical** for messages indicating critical conditions
  - **3** or **Error** for messages indicating error conditions
  - **4** or **Warning** for messages indicating warning conditions
  - **5** or **Notice** for messages indicating normal but significant conditions
  - **6** or **Informational** for messages indicating informational messages
  - **7** or **Debug** for messages indicating debug-level messages, the default
- tag: Tag specifies a record field to use as a tag on the syslog message.
- trimPrefix: Remove the specified prefix from the tag.

11.4.14.3. Additional RFC5424 syslog parameters

The following parameters apply to RFC5424:

- appName: The APP-NAME is a free-text string that identifies the application that sent the log. Must be specified for **RFC5424**.
- msgID: The MSGID is a free-text string that identifies the type of message. Must be specified for **RFC5424**.
- procID: The PROCID is a free-text string. A change in the value indicates a discontinuity in syslog reporting. Must be specified for **RFC5424**.

11.4.15. Forwarding logs to a Kafka broker

You can forward logs to an external Kafka broker in addition to, or instead of, the default log store.

To configure log forwarding to an external Kafka instance, you must create a **ClusterLogForwarder** custom resource (CR) with an output to that instance, and a pipeline that uses the output. You can include a specific Kafka topic in the output or use the default. The Kafka output can use a TCP (insecure) or TLS (secure TCP) connection.

Procedure

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

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
   - name: app-logs (4)
     type: kafka (5)
     url: tls://kafka.example.devlab.com:9093/app-topic (6)
     secret:
       name: kafka-secret (7)
   - name: infra-logs
     type: kafka
     url: tcp://kafka.devlab2.example.com:9093/infra-topic (8)
   - name: audit-logs
     type: kafka
     url: tls://kafka.qelab.example.com:9093/audit-topic
     secret:
        name: kafka-secret-qe
  pipelines:
   - name: app-topic (9)
     inputRefs: (10)
     - application
     outputRefs: (11)
     - app-logs
     labels:
       logType: "application" (12)
   - name: infra-topic (13)
     inputRefs:
     - infrastructure
     outputRefs:
     - infra-logs
     labels:
       logType: "infra"
   - name: audit-topic
     inputRefs:
     - audit
     outputRefs:
     - audit-logs
     labels:
       logType: "audit"
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.

2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.

3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.

4. Specify a name for the output.
5. Specify the **kafka** type.
6. Specify the URL and port of the Kafka broker as a valid absolute URL, optionally with a specific topic. You can use the **tcp** (insecure) or **tls** (secure TCP) protocol. If the cluster-wide proxy using the CIDR annotation is enabled, the output must be a server name or FQDN, not an IP address.
7. If you are using a **tls** prefix, you must specify the name of the secret required by the endpoint for TLS communication. The secret must contain a **ca-bundle.crt** key that points to the certificate it represents. In legacy implementations, the secret must exist in the **openshift-logging** project.
8. Optional: To send an insecure output, use a **tcp** prefix in front of the URL. Also omit the **secret** key and its **name** from this output.
9. Optional: Specify a name for the pipeline.
10. Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.
11. Specify the name of the output to use when forwarding logs with this pipeline.
12. Optional: String. One or more labels to add to the logs.
13. Optional: Configure multiple outputs to forward logs to other external log aggregators of any supported type:
    - A name to describe the pipeline.
    - The **inputRefs** is the log type to forward by using the pipeline: **application, infrastructure**, or **audit**.
    - The **outputRefs** is the name of the output to use.
    - Optional: String. One or more labels to add to the logs.

2. Optional: To forward a single output to multiple Kafka brokers, specify an array of Kafka brokers as shown in the following example:

```
# ...
spec:
  outputs:
  - name: app-logs
    type: kafka
    secret:
      name: kafka-secret-dev
    kafka: (1)
      brokers: (2)
        - tls://kafka-broker1.example.com:9093/
        - tls://kafka-broker2.example.com:9093/
      topic: app-topic (3)
# ...
```

1. Specify a **kafka** key that has a **brokers** and **topic** key.
2. Use the **brokers** key to specify an array of one or more brokers.
3. Use the **topic** key to specify the target topic that receives the logs.

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

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

11.4.16. Forwarding logs to Amazon CloudWatch

You can forward logs to Amazon CloudWatch, a monitoring and log storage service hosted by Amazon Web Services (AWS). You can forward logs to CloudWatch in addition to, or instead of, the default log store.

To configure log forwarding to CloudWatch, you must create a **ClusterLogForwarder** custom resource (CR) with an output for CloudWatch, and a pipeline that uses the output.

Procedure

1. Create a **Secret** YAML file that uses the **aws_access_key_id** and **aws_secret_access_key** fields to specify your base64-encoded AWS credentials. For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: cw-secret
  namespace: openshift-logging
data:
  aws_access_key_id: QUtJQUIPU0ZPRE5ON0VYQU1QTEUK
  aws_secret_access_key:
d0phbHJYVXRuRkVNSS9LN01ERU5HL2JQeFJmaUNZRVhBTVBMRUtFWQo=
```

2. Create the secret. For example:

```
$ oc apply -f cw-secret.yaml
```

3. Create or edit a YAML file that defines the **ClusterLogForwarder** CR object. In the file, specify the name of the secret. For example:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
   - name: cw (4)
     type: cloudwatch (5)
     cloudwatch:
       groupBy: logType (6)
```

```
      groupPrefix: <group prefix> (7)
      region: us-east-2 (8)
    secret:
      name: cw-secret (9)
  pipelines:
    - name: infra-logs (10)
      inputRefs: (11)
        - infrastructure
        - audit
        - application
      outputRefs:
        - cw (12)
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.
2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. Specify a name for the output.
5. Specify the **cloudwatch** type.
6. Optional: Specify how to group the logs:
   - **logType** creates log groups for each log type.
   - **namespaceName** creates a log group for each application name space. It also creates separate log groups for infrastructure and audit logs.
   - **namespaceUUID** creates a new log groups for each application namespace UUID. It also creates separate log groups for infrastructure and audit logs.
7. Optional: Specify a string to replace the default **infrastructureName** prefix in the names of the log groups.
8. Specify the AWS region.
9. Specify the name of the secret that contains your AWS credentials.
10. Optional: Specify a name for the pipeline.
11. Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.
12. Specify the name of the output to use when forwarding logs with this pipeline.

4. Create the CR object:

```
$ oc create -f <file-name>.yaml
```

Example: Using ClusterLogForwarder with Amazon CloudWatch

Here, you see an example **ClusterLogForwarder** custom resource (CR) and the log data that it outputs to Amazon CloudWatch.

Suppose that you are running a ROSA cluster named **mycluster**. The following command returns the cluster's **infrastructureName**, which you will use to compose **aws** commands later on:

```
$ oc get Infrastructure/cluster -ojson | jq .status.infrastructureName
"mycluster-7977k"
```

To generate log data for this example, you run a **busybox** pod in a namespace called **app**. The **busybox** pod writes a message to stdout every three seconds:

```
$ oc run busybox --image=busybox -- sh -c 'while true; do echo "My life is my message"; sleep 3;
done'
$ oc logs -f busybox
My life is my message
My life is my message
My life is my message
...
```

You can look up the UUID of the **app** namespace where the **busybox** pod runs:

```
$ oc get ns/app -ojson | jq .metadata.uid
"794e1e1a-b9f5-4958-a190-e76a9b53d7bf"
```

In your **ClusterLogForwarder** custom resource (CR), you configure the **infrastructure**, **audit**, and **application** log types as inputs to the **all-logs** pipeline. You also connect this pipeline to **cw** output, which forwards the logs to a CloudWatch instance in the **us-east-2** region:

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
   - name: cw
     type: cloudwatch
     cloudwatch:
       groupBy: logType
       region: us-east-2
     secret:
        name: cw-secret
  pipelines:
    - name: all-logs
      inputRefs:
        - infrastructure
        - audit
        - application
      outputRefs:
        - cw
```

Each region in CloudWatch contains three levels of objects:

- log group
  - log stream
    - log event

With **groupBy: logType** in the **ClusterLogForwarding** CR, the three log types in the **inputRefs** produce three log groups in Amazon Cloudwatch:

```
$ aws --output json logs describe-log-groups | jq .logGroups[].logGroupName
"mycluster-7977k.application"
"mycluster-7977k.audit"
"mycluster-7977k.infrastructure"
```

Each of the log groups contains log streams:

```
$ aws --output json logs describe-log-streams --log-group-name mycluster-7977k.application | jq
.logStreams[].logStreamName
"kubernetes.var.log.containers.busybox_app_busybox-
da085893053e20beddd6747acdbaf98e77c37718f85a7f6a4facf09ca195ad76.log"
```

```
$ aws --output json logs describe-log-streams --log-group-name mycluster-7977k.audit | jq
.logStreams[].logStreamName
"ip-10-0-131-228.us-east-2.compute.internal.k8s-audit.log"
"ip-10-0-131-228.us-east-2.compute.internal.linux-audit.log"
"ip-10-0-131-228.us-east-2.compute.internal.openshift-audit.log"
...
```

```
$ aws --output json logs describe-log-streams --log-group-name mycluster-7977k.infrastructure | jq
.logStreams[].logStreamName
"ip-10-0-131-228.us-east-2.compute.internal.kubernetes.var.log.containers.apiserver-69f9fd9b58-
zqzw5_openshift-oauth-apiserver_oauth-apiserver-
453c5c4ee026fe20a6139ba6b1cdd1bed25989c905bf5ac5ca211b7cbb5c3d7b.log"
"ip-10-0-131-228.us-east-2.compute.internal.kubernetes.var.log.containers.apiserver-797774f7c5-
lftrx_openshift-apiserver_openshift-apiserver-
ce51532df7d4e4d5f21c4f4be05f6575b93196336be0027067fd7d93d70f66a4.log"
"ip-10-0-131-228.us-east-2.compute.internal.kubernetes.var.log.containers.apiserver-797774f7c5-
lftrx_openshift-apiserver_openshift-apiserver-check-endpoints-
82a9096b5931b5c3b1d6dc4b66113252da4a6472c9fff48623baee761911a9ef.log"
...
```

Each log stream contains log events. To see a log event from the **busybox** Pod, you specify its log stream from the **application** log group:

```
$ aws logs get-log-events --log-group-name mycluster-7977k.application --log-stream-name
kubernetes.var.log.containers.busybox_app_busybox-
da085893053e20beddd6747acdbaf98e77c37718f85a7f6a4facf09ca195ad76.log
{
    "events": [
        {
            "timestamp": 1629422704178,
            "message": "{\"docker\":
{\"container_id\":\"da085893053e20beddd6747acdbaf98e77c37718f85a7f6a4facf09ca195ad76\"},\"kub
ernetes\":
```

```
{\"container_name\":\"busybox\",\"namespace_name\":\"app\",\"pod_name\":\"busybox\",\"container_image\":\"docker.io/library/busybox:latest\",\"container_image_id\":\"docker.io/library/busybox@sha256:0f354ec1728d9ff32edcd7d1b8bbdfc798277ad36120dc3dc683be44524c8b60\",\"pod_id\":\"870be234-90a3-4258-b73f-4f4d6e2777c7\",\"host\":\"ip-10-0-216-3.us-east-2.compute.internal\",\"labels\":{\"run\":\"busybox\"},\"master_url\":\"https://kubernetes.default.svc\",\"namespace_id\":\"794e1e1a-b9f5-4958-a190-e76a9b53d7bf\",\"namespace_labels\":{\"kubernetes_io/metadata_name\":\"app\"}},\"message\":\"My life is my message\",\"level\":\"unknown\",\"hostname\":\"ip-10-0-216-3.us-east-2.compute.internal\",\"pipeline_metadata\":{\"collector\":{\"ipaddr4\":\"10.0.216.3\",\"inputname\":\"fluent-plugin-systemd\",\"name\":\"fluentd\",\"received_at\":\"2021-08-20T01:25:08.085760+00:00\",\"version\":\"1.7.4 1.6.0\"}},\"@timestamp\":\"2021-08-20T01:25:04.178986+00:00\",\"viaq_index_name\":\"app-write\",\"viaq_msg_id\":\"NWRjZmUyMWQtZjgzNC00MjI4LTk3MjMtNTk3NmY3ZjU4NDk1\",\"log_type\":\"application\",\"time\":\"2021-08-20T01:25:04+00:00\"}",
            "ingestionTime": 1629422744016
        },
...
```

Example: Customizing the prefix in log group names

In the log group names, you can replace the default **infrastructureName** prefix, **mycluster-7977k**, with an arbitrary string like **demo-group-prefix**. To make this change, you update the **groupPrefix** field in the **ClusterLogForwarding** CR:

```
cloudwatch:
  groupBy: logType
  groupPrefix: demo-group-prefix
  region: us-east-2
```

The value of **groupPrefix** replaces the default **infrastructureName** prefix:

```
$ aws --output json logs describe-log-groups | jq .logGroups[].logGroupName
"demo-group-prefix.application"
"demo-group-prefix.audit"
"demo-group-prefix.infrastructure"
```

Example: Naming log groups after application namespace names

For each application namespace in your cluster, you can create a log group in CloudWatch whose name is based on the name of the application namespace.

If you delete an application namespace object and create a new one that has the same name, CloudWatch continues using the same log group as before.

If you consider successive application namespace objects that have the same name as equivalent to each other, use the approach described in this example. Otherwise, if you need to distinguish the resulting log groups from each other, see the following "Naming log groups for application namespace UUIDs" section instead.

To create application log groups whose names are based on the names of the application namespaces, you set the value of the **groupBy** field to **namespaceName** in the **ClusterLogForwarder** CR:

```
cloudwatch:
  groupBy: namespaceName
  region: us-east-2
```

- Setting **groupBy** to **namespaceName** affects the application log group only. It does not affect the **audit** and **infrastructure** log groups.

In Amazon Cloudwatch, the namespace name appears at the end of each log group name. Because there is a single application namespace, "app", the following output shows a new **mycluster-7977k.app** log group instead of **mycluster-7977k.application**:

```
$ aws --output json logs describe-log-groups | jq .logGroups[].logGroupName
"mycluster-7977k.app"
"mycluster-7977k.audit"
"mycluster-7977k.infrastructure"
```

If the cluster in this example had contained multiple application namespaces, the output would show multiple log groups, one for each namespace.

The **groupBy** field affects the application log group only. It does not affect the **audit** and **infrastructure** log groups.

Example: Naming log groups after application namespace UUIDs

For each application namespace in your cluster, you can create a log group in CloudWatch whose name is based on the UUID of the application namespace.

If you delete an application namespace object and create a new one, CloudWatch creates a new log group.

If you consider successive application namespace objects with the same name as different from each other, use the approach described in this example. Otherwise, see the preceding "Example: Naming log groups for application namespace names" section instead.

To name log groups after application namespace UUIDs, you set the value of the **groupBy** field to **namespaceUUID** in the **ClusterLogForwarder** CR:

```
cloudwatch:
  groupBy: namespaceUUID
  region: us-east-2
```

In Amazon Cloudwatch, the namespace UUID appears at the end of each log group name. Because there is a single application namespace, "app", the following output shows a new **mycluster-7977k.794e1e1a-b9f5-4958-a190-e76a9b53d7bf** log group instead of **mycluster-7977k.application**:

```
$ aws --output json logs describe-log-groups | jq .logGroups[].logGroupName
"mycluster-7977k.794e1e1a-b9f5-4958-a190-e76a9b53d7bf" // uid of the "app" namespace
"mycluster-7977k.audit"
"mycluster-7977k.infrastructure"
```

The **groupBy** field affects the application log group only. It does not affect the **audit** and **infrastructure** log groups.

11.4.17. Creating a secret for AWS CloudWatch with an existing AWS role

If you have an existing role for AWS, you can create a secret for AWS with STS using the **oc create secret --from-literal** command.

Procedure

Procedure

- In the CLI, enter the following to generate a secret for AWS:

```
$ oc create secret generic cw-sts-secret -n openshift-logging --from-literal=role_arn=arn:aws:iam::123456789012:role/my-role_with-permissions
```

Example Secret

```
apiVersion: v1
kind: Secret
metadata:
  namespace: openshift-logging
  name: my-secret-name
stringData:
  role_arn: arn:aws:iam::123456789012:role/my-role_with-permissions
```

11.4.18. Forwarding logs to Amazon CloudWatch from STS enabled clusters

For clusters with AWS Security Token Service (STS) enabled, you must create the AWS IAM roles and policies that enable log forwarding, and a **ClusterLogForwarder** custom resource (CR) with an output for CloudWatch.

Prerequisites

- Logging for Red Hat OpenShift: 5.5 and later

Procedure

1. Prepare the AWS account:

   a. Create an IAM policy JSON file with the following content:

```
{
"Version": "2012-10-17",
"Statement": [
    {
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:DescribeLogGroups",
      "logs:DescribeLogStreams",
      "logs:PutLogEvents",
      "logs:PutRetentionPolicy"
    ],
    "Resource": "arn:aws:logs:*:*:*"
    }
  ]
}
```

   b. Create an IAM trust JSON file with the following content:

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<your_aws_account_id>:oidc-
provider/<openshift_oidc_provider>" (1)
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "<openshift_oidc_provider>:sub": "system:serviceaccount:openshift-
logging:logcollector" (2)
        }
      }
    }
  ]
}
```

1. Specify your AWS account ID and the OpenShift OIDC provider endpoint. Obtain the endpoint by running the following command:

```
$ rosa describe cluster \
  -c $(oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}') \
  -o yaml | awk '/oidc_endpoint_url/ {print $2}' | cut -d '/' -f 3,4
```

2. Specify the OpenShift OIDC endpoint again.

c. Create the IAM role:

```
$ aws iam create-role
  --role-name “<your_rosa_cluster_name>-RosaCloudWatch” \
  --assume-role-policy-document file://<your_trust_file_name>.json \
  --query Role.Arn \
  --output text
```

Save the output. You will use it in the next steps.

d. Create the IAM policy:

```
$ aws iam create-policy \
--policy-name "RosaCloudWatch" \
--policy-document file:///<your_policy_file_name>.json \
--query Policy.Arn \
--output text
```

Save the output. You will use it in the next steps.

e. Attach the IAM policy to the IAM role:

```
$ aws iam attach-role-policy \
 --role-name “<your_rosa_cluster_name>-RosaCloudWatch” \
 --policy-arn <policy_ARN> (1)
```

1. Replace **policy_ARN** with the output you saved while creating the policy.

2. Create a **Secret** YAML file for the Red Hat OpenShift Logging Operator:

```
apiVersion: v1
kind: Secret
metadata:
  name: cloudwatch-credentials
  namespace: openshift-logging
stringData:
  credentials: |-
    [default]
    sts_regional_endpoints = regional
    role_arn: <role_ARN> (1)
    web_identity_token_file = /var/run/secrets/openshift/serviceaccount/token
```

1. Replace **role_ARN** with the output you saved while creating the role.

3. Create the secret:

```
$ oc apply -f cloudwatch-credentials.yaml
```

4. Create or edit a **ClusterLogForwarder** custom resource:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: <log_forwarder_name> (1)
  namespace: <log_forwarder_namespace> (2)
spec:
  serviceAccountName: <service_account_name> (3)
  outputs:
   - name: cw (4)
     type: cloudwatch (5)
     cloudwatch:
       groupBy: logType (6)
       groupPrefix: <group prefix> (7)
       region: us-east-2 (8)
     secret:
        name: <your_secret_name> (9)
  pipelines:
    - name: to-cloudwatch (10)
      inputRefs: (11)
        - infrastructure
        - audit
        - application
      outputRefs:
        - cw (12)
```

1. In legacy implementations, the CR name must be **instance**. In multi log forwarder implementations, you can use any name.

2. In legacy implementations, the CR namespace must be **openshift-logging**. In multi log forwarder implementations, you can use any namespace.
3. The name of your service account. The service account is only required in multi log forwarder implementations if the log forwarder is not deployed in the **openshift-logging** namespace.
4. Specify a name for the output.
5. Specify the **cloudwatch** type.
6. Optional: Specify how to group the logs:
   - **logType** creates log groups for each log type
   - **namespaceName** creates a log group for each application name space. Infrastructure and audit logs are unaffected, remaining grouped by **logType**.
   - **namespaceUUID** creates a new log groups for each application namespace UUID. It also creates separate log groups for infrastructure and audit logs.
7. Optional: Specify a string to replace the default **infrastructureName** prefix in the names of the log groups.
8. Specify the AWS region.
9. Specify the name of the secret you created previously.
10. Optional: Specify a name for the pipeline.
11. Specify which log types to forward by using the pipeline: **application, infrastructure**, or **audit**.
12. Specify the name of the output to use when forwarding logs with this pipeline.

Additional resources

- AWS STS API Reference

11.5. CONFIGURING THE LOGGING COLLECTOR

Logging for Red Hat OpenShift collects operations and application logs from your cluster and enriches the data with Kubernetes pod and project metadata. All supported modifications to the log collector can be performed though the **spec.collection** stanza in the **ClusterLogging** custom resource (CR).

11.5.1. Configuring the log collector

You can configure which log collector type your logging uses by modifying the **ClusterLogging** custom resource (CR).

NOTE

Fluentd is deprecated and is planned to be removed in a future release. Red Hat provides bug fixes and support for this feature during the current release lifecycle, but this feature no longer receives enhancements. As an alternative to Fluentd, you can use Vector instead.

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator.
- You have created a **ClusterLogging** CR.

Procedure

1. Modify the **ClusterLogging** CR **collection** spec:

   ClusterLogging CR example

   ```
   apiVersion: logging.openshift.io/v1
   kind: ClusterLogging
   metadata:
   # ...
   spec:
   # ...
     collection:
       type: <log_collector_type> (1)
       resources: {}
       tolerations: {}
   # ...
   ```

   (1) The log collector type you want to use for the logging. This can be **vector** or **fluentd**.

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

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

11.5.2. Creating a LogFileMetricExporter resource

In logging version 5.8 and newer versions, the LogFileMetricExporter is no longer deployed with the collector by default. You must manually create a **LogFileMetricExporter** custom resource (CR) to generate metrics from the logs produced by running containers.

If you do not create the **LogFileMetricExporter** CR, you may see a **No datapoints found** message in the Red Hat OpenShift Service on AWS web console dashboard for **Produced Logs**.

Prerequisites

- You have administrator permissions.

- You have installed the Red Hat OpenShift Logging Operator.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **LogFileMetricExporter** CR as a YAML file:

   Example LogFileMetricExporter CR

```
apiVersion: logging.openshift.io/v1alpha1
kind: LogFileMetricExporter
metadata:
  name: instance
  namespace: openshift-logging
spec:
  nodeSelector: {} (1)
  resources: (2)
    limits:
      cpu: 500m
      memory: 256Mi
    requests:
      cpu: 200m
      memory: 128Mi
  tolerations: [] (3)
# ...
```

   (1) Optional: The **nodeSelector** stanza defines which nodes the pods are scheduled on.

   (2) The **resources** stanza defines resource requirements for the **LogFileMetricExporter** CR.

   (3) Optional: The **tolerations** stanza defines the tolerations that the pods accept.

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

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

Verification

A **logfilesmetricexporter** pod runs concurrently with a **collector** pod on each node.

- Verify that the **logfilesmetricexporter** pods are running in the namespace where you have created the **LogFileMetricExporter** CR, by running the following command and observing the output:

```
$ oc get pods -l app.kubernetes.io/component=logfilesmetricexporter -n openshift-logging
```

   Example output

```
NAME                                READY   STATUS    RESTARTS   AGE
logfilesmetricexporter-9qbjj   1/1     Running   0          2m46s
logfilesmetricexporter-cbc4v   1/1     Running   0          2m46s
```

11.5.3. Configure log collector CPU and memory limits

The log collector allows for adjustments to both the CPU and memory limits.

Procedure

- Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc -n openshift-logging edit ClusterLogging instance
```

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
spec:
  collection:
    type: fluentd
    resources:
      limits: (1)
        memory: 736Mi
        requests:
          cpu: 100m
          memory: 736Mi
# ...
```

(1) Specify the CPU and memory limits and requests as needed. The values shown are the default values.

11.5.4. Configuring input receivers

The Red Hat OpenShift Logging Operator deploys a service for each configured input receiver so that clients can write to the collector. This service exposes the port specified for the input receiver. The service name is generated based on the following:

- For multi log forwarder **ClusterLogForwarder** CR deployments, the service name is in the format **<ClusterLogForwarder_CR_name>-<input_name>**. For example, **example-http-receiver**.
- For legacy **ClusterLogForwarder** CR deployments, meaning those named **instance** and located in the **openshift-logging** namespace, the service name is in the format **collector-<input_name>**. For example, **collector-http-receiver**.

11.5.4.1. Configuring the collector to receive audit logs as an HTTP server

You can configure your log collector to listen for HTTP connections and receive audit logs as an HTTP server by specifying **http** as a receiver input in the **ClusterLogForwarder** custom resource (CR). This enables you to use a common log store for audit logs that are collected from both inside and outside of your Red Hat OpenShift Service on AWS cluster.

Prerequisites

- You have administrator permissions.

- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator.
- You have created a **ClusterLogForwarder** CR.

Procedure

1. Modify the **ClusterLogForwarder** CR to add configuration for the **http** receiver input:

Example ClusterLogForwarder CR if you are using a multi log forwarder deployment

```
apiVersion: logging.openshift.io/v1beta1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  serviceAccountName: <service_account_name>
  inputs:
    - name: http-receiver (1)
      receiver:
        type: http (2)
        http:
          format: kubeAPIAudit (3)
          port: 8443 (4)
  pipelines: (5)
    - name: http-pipeline
      inputRefs:
        - http-receiver
# ...
```

(1) Specify a name for your input receiver.

(2) Specify the input receiver type as **http**.

(3) Currently, only the **kube-apiserver** webhook format is supported for **http** input receivers.

(4) Optional: Specify the port that the input receiver listens on. This must be a value between **1024** and **65535**. The default value is **8443** if this is not specified.

(5) Configure a pipeline for your input receiver.

Example ClusterLogForwarder CR if you are using a legacy deployment

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  inputs:
    - name: http-receiver (1)
      receiver:
```

```
      type: http (2)
      http:
        format: kubeAPIAudit (3)
        port: 8443 (4)
  pipelines: (5)
  - inputRefs:
    - http-receiver
    name: http-pipeline
# ...
```

1. Specify a name for your input receiver.
2. Specify the input receiver type as **http**.
3. Currently, only the **kube-apiserver** webhook format is supported for **http** input receivers.
4. Optional: Specify the port that the input receiver listens on. This must be a value between **1024** and **65535**. The default value is **8443** if this is not specified.
5. Configure a pipeline for your input receiver.

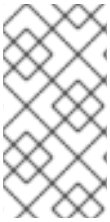
2. Apply the changes to the **ClusterLogForwarder** CR by running the following command:

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

Additional resources

- Overview of API audit filter

11.5.5. Advanced configuration for the Fluentd log forwarder



NOTE

Fluentd is deprecated and is planned to be removed in a future release. Red Hat provides bug fixes and support for this feature during the current release lifecycle, but this feature no longer receives enhancements. As an alternative to Fluentd, you can use Vector instead.

Logging includes multiple Fluentd parameters that you can use for tuning the performance of the Fluentd log forwarder. With these parameters, you can change the following Fluentd behaviors:

- Chunk and chunk buffer sizes
- Chunk flushing behavior
- Chunk forwarding retry behavior

Fluentd collects log data in a single blob called a *chunk*. When Fluentd creates a chunk, the chunk is considered to be in the *stage*, where the chunk gets filled with data. When the chunk is full, Fluentd moves the chunk to the *queue*, where chunks are held before being flushed, or written out to their destination. Fluentd can fail to flush a chunk for a number of reasons, such as network issues or capacity issues at the destination. If a chunk cannot be flushed, Fluentd retries flushing as configured.

By default in Red Hat OpenShift Service on AWS, Fluentd uses the *exponential backoff* method to retry flushing, where Fluentd doubles the time it waits between attempts to retry flushing again, which helps reduce connection requests to the destination. You can disable exponential backoff and use the *periodic* retry method instead, which retries flushing the chunks at a specified interval.

These parameters can help you determine the trade-offs between latency and throughput.

- To optimize Fluentd for throughput, you could use these parameters to reduce network packet count by configuring larger buffers and queues, delaying flushes, and setting longer times between retries. Be aware that larger buffers require more space on the node file system.
- To optimize for low latency, you could use the parameters to send data as soon as possible, avoid the build-up of batches, have shorter queues and buffers, and use more frequent flush and retries.

You can configure the chunking and flushing behavior using the following parameters in the **ClusterLogging** custom resource (CR). The parameters are then automatically added to the Fluentd config map for use by Fluentd.

NOTE

These parameters are:

- Not relevant to most users. The default settings should give good general performance.
- Only for advanced users with detailed knowledge of Fluentd configuration and performance.
- Only for performance tuning. They have no effect on functional aspects of logging.

Table 11.11. Advanced Fluentd Configuration Parameters

Parameter	Description	Default
chunkLimitSize	The maximum size of each chunk. Fluentd stops writing data to a chunk when it reaches this size. Then, Fluentd sends the chunk to the queue and opens a new chunk.	**8m**
totalLimitSize	The maximum size of the buffer, which is the total size of the stage and the queue. If the buffer size exceeds this value, Fluentd stops adding data to chunks and fails with an error. All data not in chunks is lost.	Approximately 15% of the node disk distributed across all outputs.

Parameter	Description	Default
flushInterval	The interval between chunk flushes. You can use **s** (seconds), **m** (minutes), **h** (hours), or **d** (days).	**1s**
flushMode	The method to perform flushes: • **lazy**: Flush chunks based on the **timekey** parameter. You cannot modify the **timekey** parameter. • **interval**: Flush chunks based on the **flushInterval** parameter. • **immediate**: Flush chunks immediately after data is added to a chunk.	**interval**
flushThreadCount	The number of threads that perform chunk flushing. Increasing the number of threads improves the flush throughput, which hides network latency.	**2**
overflowAction	The chunking behavior when the queue is full: • **throw_exception**: Raise an exception to show in the log. • **block**: Stop data chunking until the full buffer issue is resolved. • **drop_oldest_chunk**: Drop the oldest chunk to accept new incoming chunks. Older chunks have less value than newer chunks.	**block**
retryMaxInterval	The maximum time in seconds for the **exponential_backoff** retry method.	**300s**

Parameter	Description	Default
retryType	The retry method when flushing fails: • **exponential_backoff**: Increase the time between flush retries. Fluentd doubles the time it waits until the next retry until the **retry_max_interval** parameter is reached. • **periodic**: Retries flushes periodically, based on the **retryWait** parameter.	**exponential_backoff**
retryTimeOut	The maximum time interval to attempt retries before the record is discarded.	**60m**
retryWait	The time in seconds before the next chunk flush.	**1s**

For more information on the Fluentd chunk lifecycle, see Buffer Plugins in the Fluentd documentation.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc edit ClusterLogging instance
```

2. Add or modify any of the following parameters:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: instance
  namespace: openshift-logging
spec:
  collection:
    fluentd:
      buffer:
        chunkLimitSize: 8m (1)
        flushInterval: 5s (2)
        flushMode: interval (3)
        flushThreadCount: 3 (4)
        overflowAction: throw_exception (5)
        retryMaxInterval: "300s" (6)
        retryType: periodic (7)
```

```
        retryWait: 1s (8)
        totalLimitSize: 32m (9)
# ...
```

1. Specify the maximum size of each chunk before it is queued for flushing.
2. Specify the interval between chunk flushes.
3. Specify the method to perform chunk flushes: **lazy**, **interval**, or **immediate**.
4. Specify the number of threads to use for chunk flushes.
5. Specify the chunking behavior when the queue is full: **throw_exception**, **block**, or **drop_oldest_chunk**.
6. Specify the maximum interval in seconds for the **exponential_backoff** chunk flushing method.
7. Specify the retry type when chunk flushing fails: **exponential_backoff** or **periodic**.
8. Specify the time in seconds before the next chunk flush.
9. Specify the maximum size of the chunk buffer.

3. Verify that the Fluentd pods are redeployed:

```
$ oc get pods -l component=collector -n openshift-logging
```

4. Check that the new values are in the **fluentd** config map:

```
$ oc extract configmap/collector-config --confirm
```

Example fluentd.conf

```
<buffer>
  @type file
  path '/var/lib/fluentd/default'
  flush_mode interval
  flush_interval 5s
  flush_thread_count 3
  retry_type periodic
  retry_wait 1s
  retry_max_interval 300s
  retry_timeout 60m
  queued_chunks_limit_size "#{ENV['BUFFER_QUEUE_LIMIT'] || '32'}"
  total_limit_size "#{ENV['TOTAL_LIMIT_SIZE_PER_BUFFER'] || '8589934592'}"
  chunk_limit_size 8m
  overflow_action throw_exception
  disable_chunk_backup true
</buffer>
```

11.6. COLLECTING AND STORING KUBERNETES EVENTS

The Red Hat OpenShift Service on AWS Event Router is a pod that watches Kubernetes events and logs them for collection by the logging. You must manually deploy the Event Router.

The Event Router collects events from all projects and writes them to **STDOUT**. The collector then forwards those events to the store defined in the **ClusterLogForwarder** custom resource (CR).

IMPORTANT

The Event Router adds additional load to Fluentd and can impact the number of other log messages that can be processed.

11.6.1. Deploying and configuring the Event Router

Use the following steps to deploy the Event Router into your cluster. You should always deploy the Event Router to the **openshift-logging** project to ensure it collects events from across the cluster.

NOTE

The Event Router image is not a part of the Red Hat OpenShift Logging Operator and must be downloaded separately.

The following **Template** object creates the service account, cluster role, and cluster role binding required for the Event Router. The template also configures and deploys the Event Router pod. You can either use this template without making changes or edit the template to change the deployment object CPU and memory requests.

Prerequisites

- You need proper permissions to create service accounts and update cluster role bindings. For example, you can run the following template with a user that has the **cluster-admin** role.
- The Red Hat OpenShift Logging Operator must be installed.

Procedure

1. Create a template for the Event Router:

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: eventrouter-template
  annotations:
    description: "A pod forwarding kubernetes events to OpenShift Logging stack."
    tags: "events,EFK,logging,cluster-logging"
objects:
  - kind: ServiceAccount (1)
    apiVersion: v1
    metadata:
      name: eventrouter
      namespace: ${NAMESPACE}
  - kind: ClusterRole (2)
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: event-reader
```

```
    rules:
    - apiGroups: [""]
      resources: ["events"]
      verbs: ["get", "watch", "list"]
  - kind: ClusterRoleBinding (3)
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: event-reader-binding
    subjects:
    - kind: ServiceAccount
      name: eventrouter
      namespace: ${NAMESPACE}
    roleRef:
      kind: ClusterRole
      name: event-reader
  - kind: ConfigMap (4)
    apiVersion: v1
    metadata:
      name: eventrouter
      namespace: ${NAMESPACE}
    data:
      config.json: |-
        {
          "sink": "stdout"
        }
  - kind: Deployment (5)
    apiVersion: apps/v1
    metadata:
      name: eventrouter
      namespace: ${NAMESPACE}
      labels:
        component: "eventrouter"
        logging-infra: "eventrouter"
        provider: "openshift"
    spec:
      selector:
        matchLabels:
          component: "eventrouter"
          logging-infra: "eventrouter"
          provider: "openshift"
      replicas: 1
      template:
        metadata:
          labels:
            component: "eventrouter"
            logging-infra: "eventrouter"
            provider: "openshift"
          name: eventrouter
        spec:
          serviceAccount: eventrouter
          containers:
            - name: kube-eventrouter
              image: ${IMAGE}
              imagePullPolicy: IfNotPresent
              resources:
                requests:
```

```
                cpu: ${CPU}
                memory: ${MEMORY}
              volumeMounts:
              - name: config-volume
                mountPath: /etc/eventrouter
              securityContext:
                allowPrivilegeEscalation: false
                capabilities:
                  drop: ["ALL"]
            securityContext:
              runAsNonRoot: true
              seccompProfile:
                type: RuntimeDefault
            volumes:
            - name: config-volume
              configMap:
                name: eventrouter
parameters:
  - name: IMAGE 6
    displayName: Image
    value: "registry.redhat.io/openshift-logging/eventrouter-rhel9:v0.4"
  - name: CPU 7
    displayName: CPU
    value: "100m"
  - name: MEMORY 8
    displayName: Memory
    value: "128Mi"
  - name: NAMESPACE
    displayName: Namespace
    value: "openshift-logging" 9
```

1. Creates a Service Account in the **openshift-logging** project for the Event Router.
2. Creates a ClusterRole to monitor for events in the cluster.
3. Creates a ClusterRoleBinding to bind the ClusterRole to the service account.
4. Creates a config map in the **openshift-logging** project to generate the required **config.json** file.
5. Creates a deployment in the **openshift-logging** project to generate and configure the Event Router pod.
6. Specifies the image, identified by a tag such as **v0.4**.
7. Specifies the minimum amount of CPU to allocate to the Event Router pod. Defaults to **100m**.
8. Specifies the minimum amount of memory to allocate to the Event Router pod. Defaults to **128Mi**.
9. Specifies the **openshift-logging** project to install objects in.

2. Use the following command to process and apply the template:

```
$ oc process -f <templatefile> | oc apply -n openshift-logging -f -
```

For example:

```
$ oc process -f eventrouter.yaml | oc apply -n openshift-logging -f -
```

Example output

```
serviceaccount/eventrouter created
clusterrole.rbac.authorization.k8s.io/event-reader created
clusterrolebinding.rbac.authorization.k8s.io/event-reader-binding created
configmap/eventrouter created
deployment.apps/eventrouter created
```

3. Validate that the Event Router installed in the **openshift-logging** project:

   a. View the new Event Router pod:

   ```
   $ oc get pods --selector  component=eventrouter -o name -n openshift-logging
   ```

   Example output

   ```
   pod/cluster-logging-eventrouter-d649f97c8-qvv8r
   ```

   b. View the events collected by the Event Router:

   ```
   $ oc logs <cluster_logging_eventrouter_pod> -n openshift-logging
   ```

   For example:

   ```
   $ oc logs cluster-logging-eventrouter-d649f97c8-qvv8r -n openshift-logging
   ```

   Example output

   ```
   {"verb":"ADDED","event":{"metadata":{"name":"openshift-service-catalog-controller-
   manager-remover.1632d931e88fcd8f","namespace":"openshift-service-catalog-
   removed","selfLink":"/api/v1/namespaces/openshift-service-catalog-
   removed/events/openshift-service-catalog-controller-manager-
   remover.1632d931e88fcd8f","uid":"787d7b26-3d2f-4017-b0b0-
   420db4ae62c0","resourceVersion":"21399","creationTimestamp":"2020-09-
   08T15:40:26Z"},"involvedObject":{"kind":"Job","namespace":"openshift-service-catalog-
   removed","name":"openshift-service-catalog-controller-manager-
   remover","uid":"fac9f479-4ad5-4a57-8adc-
   cb25d3d9cf8f","apiVersion":"batch/v1","resourceVersion":"21280"},"reason":"Completed","
   message":"Job completed","source":{"component":"job-
   controller"},"firstTimestamp":"2020-09-08T15:40:26Z","lastTimestamp":"2020-09-
   08T15:40:26Z","count":1,"type":"Normal"}}
   ```

   You can also use Kibana to view events by creating an index pattern using the Elasticsearch **infra** index.

CHAPTER 12. LOG STORAGE

12.1. ABOUT LOG STORAGE

You can use an internal Loki or Elasticsearch log store on your cluster for storing logs, or you can use a **ClusterLogForwarder** custom resource (CR) to forward logs to an external store.

12.1.1. Log storage types

Loki is a horizontally scalable, highly available, multi-tenant log aggregation system offered as an alternative to Elasticsearch as a log store for the logging.

Elasticsearch indexes incoming log records completely during ingestion. Loki only indexes a few fixed labels during ingestion and defers more complex parsing until after the logs have been stored. This means Loki can collect logs more quickly.

12.1.1.1. About the Elasticsearch log store

The logging Elasticsearch instance is optimized and tested for short term storage, approximately seven days. If you want to retain your logs over a longer term, it is recommended you move the data to a third-party storage system.

Elasticsearch organizes the log data from Fluentd into datastores, or *indices*, then subdivides each index into multiple pieces called *shards*, which it spreads across a set of Elasticsearch nodes in an Elasticsearch cluster. You can configure Elasticsearch to make copies of the shards, called *replicas*, which Elasticsearch also spreads across the Elasticsearch nodes. The **ClusterLogging** custom resource (CR) allows you to specify how the shards are replicated to provide data redundancy and resilience to failure. You can also specify how long the different types of logs are retained using a retention policy in the **ClusterLogging** CR.

NOTE

The number of primary shards for the index templates is equal to the number of Elasticsearch data nodes.

The Red Hat OpenShift Logging Operator and companion OpenShift Elasticsearch Operator ensure that each Elasticsearch node is deployed using a unique deployment that includes its own storage volume. You can use a **ClusterLogging** custom resource (CR) to increase the number of Elasticsearch nodes, as needed. See the Elasticsearch documentation for considerations involved in configuring storage.

NOTE

A highly-available Elasticsearch environment requires at least three Elasticsearch nodes, each on a different host.

Role-based access control (RBAC) applied on the Elasticsearch indices enables the controlled access of the logs to the developers. Administrators can access all logs and developers can access only the logs in their projects.

12.1.2. Querying log stores

You can query Loki by using the LogQL log query language.

12.1.3. Additional resources

- Loki components documentation
- Loki Object Storage documentation

12.2. INSTALLING LOG STORAGE

You can use the OpenShift CLI (**oc**) or the Red Hat OpenShift Service on AWS web console to deploy a log store on your Red Hat OpenShift Service on AWS cluster.

NOTE

The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

12.2.1. Deploying a Loki log store

You can use the Loki Operator to deploy an internal Loki log store on your Red Hat OpenShift Service on AWS cluster. After install the Loki Operator, you must configure Loki object storage by creating a secret, and create a **LokiStack** custom resource (CR).

12.2.1.1. 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.

IMPORTANT

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

Table 12.1. Loki sizing

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

	1x.demo	1x.extra-small	1x.small	1x.medium
Total CPU requests if using the ruler	None	16 vCPUs	42 vCPUs	70 vCPUs
Total memory requests	None	31Gi	67Gi	139Gi
Total memory requests if using the ruler	None	35Gi	83Gi	171Gi
Total disk requests	40Gi	430Gi	430Gi	590Gi
Total disk requests if using the ruler	80Gi	750Gi	750Gi	910Gi

12.2.1.2. Installing the Loki Operator by using the Red Hat OpenShift Service on AWS web console

To install and configure logging on your Red Hat OpenShift Service on AWS cluster, additional Operators must be installed. This can be done from the Operator Hub within the web console.

Red Hat OpenShift Service on AWS Operators use custom resources (CR) to manage applications and their components. High-level configuration and settings are provided by the user within a CR. The Operator translates high-level directives into low-level actions, based on best practices embedded within the Operator's logic. 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, which are then used to generate CRs.

Prerequisites

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

Procedure

1. In the Red Hat OpenShift Service on AWS 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** or **stable-x.y** as the **Update channel**.

NOTE

The **stable** channel only provides updates to the most recent release of logging. To continue receiving updates for prior releases, you must change your subscription channel to **stable-x.y**, where **x.y** represents the major and minor version of logging you have installed. For example, **stable-5.7**.

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 is 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.

Verification

1. Go to **Operators → Installed Operators**.
2. Make sure the **openshift-logging** project is selected.
3. In the **Status** column, verify that you see green checkmarks with **InstallSucceeded** and the text **Up to date**.

NOTE

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

12.2.1.3. Creating a secret for Loki object storage by using the web console

To configure Loki object storage, you must create a secret. You can create a secret by using the Red Hat OpenShift Service on AWS web console.

Prerequisites

- You have administrator permissions.
- You have access to the Red Hat OpenShift Service on AWS web console.
- You installed the Loki Operator.

Procedure

1. Go to **Workloads → Secrets** in the **Administrator** perspective of the Red Hat OpenShift Service on AWS web console.
2. From the **Create** drop-down list, select **From YAML**.
3. Create a secret that uses the **access_key_id** and **access_key_secret** fields to specify your credentials and the **bucketnames**, **endpoint**, and **region** fields to define the object storage location. AWS is used in the following example:

   Example Secret object

```
apiVersion: v1
kind: Secret
metadata:
  name: logging-loki-s3
  namespace: openshift-logging
stringData:
  access_key_id: AKIAIOSFODNN7EXAMPLE
  access_key_secret: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
  bucketnames: s3-bucket-name
  endpoint: https://s3.eu-central-1.amazonaws.com
  region: eu-central-1
```

Additional resources

- Loki object storage

12.2.1.4. Workload identity federation

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

Prerequisites

- Red Hat OpenShift Service on AWS 4.14 and later
- Logging 5.9 and later

Procedure

- If you use the Red Hat OpenShift Service on AWS web console to install the Loki Operator, STS clusters 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.

Azure sample subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat
spec:
  channel: "stable-5.9"
  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>
```

AWS sample subscription

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: loki-operator
  namespace: openshift-operators-redhat
spec:
  channel: "stable-5.9"
  installPlanApproval: Manual
  name: loki-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    env:
    - name: ROLEARN
      value: <role_ARN>
```

12.2.1.5. Creating a LokiStack custom resource by using the web console

You can create a **LokiStack** custom resource (CR) by using the Red Hat OpenShift Service on AWS web console.

Prerequisites

- You have administrator permissions.
- You have access to the Red Hat OpenShift Service on AWS web console.
- You installed the Loki Operator.

Procedure

1. Go to the **Operators → Installed Operators** page. Click the **All instances** tab.

2. From the **Create new** drop-down list, select **LokiStack**.

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

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki (1)
  namespace: openshift-logging
spec:
  size: 1x.small (2)
  storage:
    schemas:
      - effectiveDate: '2023-10-15'
        version: v13
    secret:
      name: logging-loki-s3 (3)
      type: s3 (4)
      credentialMode: (5)
  storageClassName: <storage_class_name> (6)
  tenants:
    mode: openshift-logging
```

1. Use the name **logging-loki**.
2. Specify the deployment size. In the logging 5.8 and later versions, the supported size options for production instances of Loki are **1x.extra-small**, **1x.small**, or **1x.medium**.
3. Specify the secret used for your log storage.
4. Specify the corresponding storage type.
5. Optional field, logging 5.9 and later. Supported user configured values are as follows: **static** is the default authentication mode available for all supported object storage types using credentials stored in a Secret. **token** for short-lived tokens retrieved from a credential source. In this mode the static configuration does not contain credentials needed for the object storage. Instead, they are generated during runtime using a service, which allows for shorter-lived credentials and much more granular control. This authentication mode is not supported for all object storage types. **token-cco** is the default value when Loki is running on managed STS mode and using CCO on STS/WIF clusters.
6. Enter the name of a storage class for temporary storage. For best performance, specify a storage class that allocates block storage. Available storage classes for your cluster can be listed by using the **oc get storageclasses** command.

12.2.1.6. Installing Loki Operator by using the CLI

To install and configure logging on your Red Hat OpenShift Service on AWS cluster, additional Operators must be installed. This can be done from the Red Hat OpenShift Service on AWS CLI.

Red Hat OpenShift Service on AWS Operators use custom resources (CR) to manage applications and their components. High-level configuration and settings are provided by the user within a CR. The Operator translates high-level directives into low-level actions, based on best practices embedded

within the Operator's logic. 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, which are then used to generate CRs.

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 **Subscription** object:

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

   1. You must specify the **openshift-operators-redhat** namespace.
   2. Specify **stable**, or **stable-5.<y>** as the channel.
   3. Specify **redhat-operators**. If your Red Hat OpenShift Service on AWS cluster is installed on a restricted network, also known as a disconnected cluster, specify the name of the **CatalogSource** object you created when you configured the Operator Lifecycle Manager (OLM).

2. Apply the **Subscription** object:

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

12.2.1.7. Creating a secret for Loki object storage by using the CLI

To configure Loki object storage, you must create a secret. You can do this by using the OpenShift CLI (**oc**).

Prerequisites

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

Procedure

- Create a secret in the directory that contains your certificate and key files by running the following command:

```
$ oc create secret generic -n openshift-logging <your_secret_name> \
 --from-file=tls.key=<your_key_file>
 --from-file=tls.crt=<your_crt_file>
 --from-file=ca-bundle.crt=<your_bundle_file>
 --from-literal=username=<your_username>
 --from-literal=password=<your_password>
```

> **NOTE**
>
> Use generic or opaque secrets for best results.

Verification

- Verify that a secret was created by running the following command:

```
$ oc get secrets
```

Additional resources

- Loki object storage

12.2.1.8. Creating a LokiStack custom resource by using the CLI

You can create a **LokiStack** custom resource (CR) by using the OpenShift CLI (**oc**).

Prerequisites

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

Procedure

1. Create a **LokiStack** CR:

   Example LokiStack CR

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: logging-loki (1)
  namespace: openshift-logging
spec:
  size: 1x.small (2)
  storage:
    schemas:
```

```
    - effectiveDate: '2023-10-15'
      version: v13
  secret:
    name: logging-loki-s3 (3)
    type: s3 (4)
    credentialMode: (5)
storageClassName: <storage_class_name> (6)
tenants:
  mode: openshift-logging
```

1. Use the name **logging-loki**.
2. Specify the deployment size. In the logging 5.8 and later versions, the supported size options for production instances of Loki are **1x.extra-small**, **1x.small**, or **1x.medium**.
3. Specify the secret used for your log storage.
4. Specify the corresponding storage type.
5. Optional field, logging 5.9 and later. Supported user configured values are as follows: **static** is the default authentication mode available for all supported object storage types using credentials stored in a Secret. **token** for short-lived tokens retrieved from a credential source. In this mode the static configuration does not contain credentials needed for the object storage. Instead, they are generated during runtime using a service, which allows for shorter-lived credentials and much more granular control. This authentication mode is not supported for all object storage types. **token-cco** is the default value when Loki is running on managed STS mode and using CCO on STS/WIF clusters.
6. Enter the name of a storage class for temporary storage. For best performance, specify a storage class that allocates block storage. Available storage classes for your cluster can be listed by using the **oc get storageclasses** command.

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

Verification

- Verify the installation by listing the pods in the **openshift-logging** project by running the following command and observing the output:

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

Confirm that you see several pods for components of the logging, similar to the following list:

Example output

```
NAME                                          READY   STATUS    RESTARTS   AGE
cluster-logging-operator-78fddc697-mnl82      1/1     Running   0          14m
collector-6cglq                               2/2     Running   0          45s
collector-8r664                               2/2     Running   0          45s
collector-8z7px                               2/2     Running   0          45s
collector-pdxl9                               2/2     Running   0          45s
collector-tc9dx                               2/2     Running   0          45s
collector-xkd76                               2/2     Running   0          45s
logging-loki-compactor-0                      1/1     Running   0          8m2s
```

```
logging-loki-distributor-b85b7d9fd-25j9g      1/1     Running   0          8m2s
logging-loki-distributor-b85b7d9fd-xwjs6      1/1     Running   0          8m2s
logging-loki-gateway-7bb86fd855-hjhl4         2/2     Running   0          8m2s
logging-loki-gateway-7bb86fd855-qjtlb         2/2     Running   0          8m2s
logging-loki-index-gateway-0                  1/1     Running   0          8m2s
logging-loki-index-gateway-1                  1/1     Running   0          7m29s
logging-loki-ingester-0                       1/1     Running   0          8m2s
logging-loki-ingester-1                       1/1     Running   0          6m46s
logging-loki-querier-f5cf9cb87-9fdjd          1/1     Running   0          8m2s
logging-loki-querier-f5cf9cb87-fp9v5          1/1     Running   0          8m2s
logging-loki-query-frontend-58c579fcb7-lfvbc  1/1     Running   0          8m2s
logging-loki-query-frontend-58c579fcb7-tjf9k  1/1     Running   0          8m2s
logging-view-plugin-79448d8df6-ckgmx          1/1     Running   0          46s
```

12.2.2. Loki object storage

The Loki Operator supports AWS S3, as well as other S3 compatible object stores such as Minio and OpenShift Data Foundation. Azure, GCS, and Swift are also supported.

The recommended nomenclature for Loki storage is **logging-loki-*<your_storage_provider>***.

The following table shows the **type** values within the **LokiStack** custom resource (CR) for each storage provider. For more information, see the section on your storage provider.

Table 12.2. Secret type quick reference

Storage provider	Secret **type** value
AWS	s3
Azure	azure
Google Cloud	gcs
Minio	s3
OpenShift Data Foundation	s3
Swift	swift

12.2.2.1. AWS storage

Prerequisites

- You installed the Loki Operator.
- You installed the OpenShift CLI (**oc**).
- You created a bucket on AWS.
- You created an AWS IAM Policy and IAM User .

Procedure

- Create an object storage secret with the name **logging-loki-aws** by running the following command:

```
$ oc create secret generic logging-loki-aws \
  --from-literal=bucketnames="<bucket_name>" \
  --from-literal=endpoint="<aws_bucket_endpoint>" \
  --from-literal=access_key_id="<aws_access_key_id>" \
  --from-literal=access_key_secret="<aws_access_key_secret>" \
  --from-literal=region="<aws_region_of_your_bucket>"
```

12.2.2.1.1. AWS storage for STS enabled clusters

If your cluster has STS enabled, the Cloud Credential Operator (CCO) supports short-term authentication using AWS tokens.

You can create the Loki object storage secret manually by running the following command:

```
$ oc -n openshift-logging create secret generic "logging-loki-aws" \
--from-literal=bucketnames="<s3_bucket_name>" \
--from-literal=region="<bucket_region>" \
--from-literal=audience="<oidc_audience>" (1)
```

(1) Optional annotation, default value is **openshift**.

12.2.2.2. Azure storage

Prerequisites

- You installed the Loki Operator.
- You installed the OpenShift CLI (**oc**).
- You created a bucket on Azure.

Procedure

- Create an object storage secret with the name **logging-loki-azure** by running the following command:

```
$ oc create secret generic logging-loki-azure \
  --from-literal=container="<azure_container_name>" \
  --from-literal=environment="<azure_environment>" \ (1)
  --from-literal=account_name="<azure_account_name>" \
  --from-literal=account_key="<azure_account_key>"
```

(1) Supported environment values are **AzureGlobal**, **AzureChinaCloud**, **AzureGermanCloud**, or **AzureUSGovernment**.

12.2.2.2.1. Azure storage for STS enabled clusters

If your cluster has STS enabled, the Cloud Credential Operator (CCO) supports short-term authentication using Azure AD Workload Identity.

You can create the Loki object storage secret manually by running the following command:

```
$ oc -n openshift-logging create secret generic logging-loki-azure \
--from-literal=environment="<azure_environment>" \
--from-literal=account_name="<storage_account_name>" \
--from-literal=container="<container_name>"
```

12.2.2.3. Google Cloud Platform storage

Prerequisites

- You installed the Loki Operator.
- You installed the OpenShift CLI (**oc**).
- You created a project on Google Cloud Platform (GCP).
- You created a bucket in the same project.
- You created a service account in the same project for GCP authentication.

Procedure

1. Copy the service account credentials received from GCP into a file called **key.json**.
2. Create an object storage secret with the name **logging-loki-gcs** by running the following command:

```
$ oc create secret generic logging-loki-gcs \
  --from-literal=bucketname="<bucket_name>" \
  --from-file=key.json="<path/to/key.json>"
```

12.2.2.4. Minio storage

Prerequisites

- You installed the Loki Operator.
- You installed the OpenShift CLI (**oc**).
- You have Minio deployed on your cluster.
- You created a bucket on Minio.

Procedure

- Create an object storage secret with the name **logging-loki-minio** by running the following command:

```
$ oc create secret generic logging-loki-minio \
  --from-literal=bucketnames="<bucket_name>" \
```

```
  --from-literal=endpoint="<minio_bucket_endpoint>" \
  --from-literal=access_key_id="<minio_access_key_id>" \
  --from-literal=access_key_secret="<minio_access_key_secret>"
```

12.2.2.5. OpenShift Data Foundation storage

Prerequisites

- You installed the Loki Operator.
- You installed the OpenShift CLI (**oc**).
- You deployed OpenShift Data Foundation.
- You configured your OpenShift Data Foundation cluster for object storage.

Procedure

1. Create an **ObjectBucketClaim** custom resource in the **openshift-logging** namespace:

```
apiVersion: objectbucket.io/v1alpha1
kind: ObjectBucketClaim
metadata:
  name: loki-bucket-odf
  namespace: openshift-logging
spec:
  generateBucketName: loki-bucket-odf
  storageClassName: openshift-storage.noobaa.io
```

2. Get bucket properties from the associated **ConfigMap** object by running the following command:

```
BUCKET_HOST=$(oc get -n openshift-logging configmap loki-bucket-odf -o
jsonpath='{.data.BUCKET_HOST}')
BUCKET_NAME=$(oc get -n openshift-logging configmap loki-bucket-odf -o
jsonpath='{.data.BUCKET_NAME}')
BUCKET_PORT=$(oc get -n openshift-logging configmap loki-bucket-odf -o
jsonpath='{.data.BUCKET_PORT}')
```

3. Get bucket access key from the associated secret by running the following command:

```
ACCESS_KEY_ID=$(oc get -n openshift-logging secret loki-bucket-odf -o
jsonpath='{.data.AWS_ACCESS_KEY_ID}' | base64 -d)
SECRET_ACCESS_KEY=$(oc get -n openshift-logging secret loki-bucket-odf -o
jsonpath='{.data.AWS_SECRET_ACCESS_KEY}' | base64 -d)
```

4. Create an object storage secret with the name **logging-loki-odf** by running the following command:

```
$ oc create -n openshift-logging secret generic logging-loki-odf \
--from-literal=access_key_id="<access_key_id>" \
--from-literal=access_key_secret="<secret_access_key>" \
--from-literal=bucketnames="<bucket_name>" \
--from-literal=endpoint="https://<bucket_host>:<bucket_port>"
```

-

12.2.2.6. Swift storage

Prerequisites

- You installed the Loki Operator.
- You installed the OpenShift CLI (**oc**).
- You created a bucket on Swift.

Procedure

- Create an object storage secret with the name **logging-loki-swift** by running the following command:

```
$ oc create secret generic logging-loki-swift \
  --from-literal=auth_url="<swift_auth_url>" \
  --from-literal=username="<swift_usernameclaim>" \
  --from-literal=user_domain_name="<swift_user_domain_name>" \
  --from-literal=user_domain_id="<swift_user_domain_id>" \
  --from-literal=user_id="<swift_user_id>" \
  --from-literal=password="<swift_password>" \
  --from-literal=domain_id="<swift_domain_id>" \
  --from-literal=domain_name="<swift_domain_name>" \
  --from-literal=container_name="<swift_container_name>"
```

- You can optionally provide project-specific data, region, or both by running the following command:

```
$ oc create secret generic logging-loki-swift \
  --from-literal=auth_url="<swift_auth_url>" \
  --from-literal=username="<swift_usernameclaim>" \
  --from-literal=user_domain_name="<swift_user_domain_name>" \
  --from-literal=user_domain_id="<swift_user_domain_id>" \
  --from-literal=user_id="<swift_user_id>" \
  --from-literal=password="<swift_password>" \
  --from-literal=domain_id="<swift_domain_id>" \
  --from-literal=domain_name="<swift_domain_name>" \
  --from-literal=container_name="<swift_container_name>" \
  --from-literal=project_id="<swift_project_id>" \
  --from-literal=project_name="<swift_project_name>" \
  --from-literal=project_domain_id="<swift_project_domain_id>" \
  --from-literal=project_domain_name="<swift_project_domain_name>" \
  --from-literal=region="<swift_region>"
```

12.2.3. Deploying an Elasticsearch log store

You can use the OpenShift Elasticsearch Operator to deploy an internal Elasticsearch log store on your Red Hat OpenShift Service on AWS cluster.

NOTE

The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

12.2.3.1. Storage considerations for Elasticsearch

A persistent volume is required for each Elasticsearch deployment configuration. On Red Hat OpenShift Service on AWS this is achieved using persistent volume claims (PVCs).

NOTE

If you use a local volume for persistent storage, do not use a raw block volume, which is described with **volumeMode: block** in the **LocalVolume** object. Elasticsearch cannot use raw block volumes.

The OpenShift Elasticsearch Operator names the PVCs using the Elasticsearch resource name.

Fluentd ships any logs from **systemd journal** and **/var/log/containers/*.log** to Elasticsearch.

Elasticsearch requires sufficient memory to perform large merge operations. If it does not have enough memory, it becomes unresponsive. To avoid this problem, evaluate how much application log data you need, and allocate approximately double that amount of free storage capacity.

By default, when storage capacity is 85% full, Elasticsearch stops allocating new data to the node. At 90%, Elasticsearch attempts to relocate existing shards from that node to other nodes if possible. But if no nodes have a free capacity below 85%, Elasticsearch effectively rejects creating new indices and becomes RED.

NOTE

These low and high watermark values are Elasticsearch defaults in the current release. You can modify these default values. Although the alerts use the same default values, you cannot change these values in the alerts.

12.2.3.2. Installing the OpenShift Elasticsearch Operator by using the web console

The OpenShift Elasticsearch Operator creates and manages the Elasticsearch cluster used by OpenShift Logging.

Prerequisites

- Elasticsearch is a memory-intensive application. Each Elasticsearch node needs at least 16GB of memory for both memory requests and limits, unless you specify otherwise in the **ClusterLogging** custom resource.
  The initial set of Red Hat OpenShift Service on AWS nodes might not be large enough to support the Elasticsearch cluster. You must add additional nodes to the Red Hat OpenShift Service on AWS cluster to run with the recommended or higher memory, up to a maximum of 64GB for each Elasticsearch node.

Elasticsearch nodes can operate with a lower memory setting, though this is not recommended for production environments.

- Ensure that you have the necessary persistent storage for Elasticsearch. Note that each Elasticsearch node requires its own storage volume.

> **NOTE**
>
> If you use a local volume for persistent storage, do not use a raw block volume, which is described with **volumeMode: block** in the **LocalVolume** object. Elasticsearch cannot use raw block volumes.

Procedure

1. In the Red Hat OpenShift Service on AWS web console, click **Operators → OperatorHub**.
2. Click **OpenShift Elasticsearch Operator** from the list of available Operators, and click **Install**.
3. Ensure that the **All namespaces on the cluster** is selected under **Installation mode**.
4. Ensure that **openshift-operators-redhat** is selected under **Installed Namespace**.
   You must specify the **openshift-operators-redhat** namespace. The **openshift-operators** namespace might contain Community Operators, which are untrusted and could publish a metric with the same name as Red Hat OpenShift Service on AWS metric, which would cause conflicts.
5. 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.
6. Select **stable-5.x** as the **Update channel**.
7. Select an **Update approval** strategy:
   - The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.
   - The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.
8. Click **Install**.

Verification

1. Verify that the OpenShift Elasticsearch Operator installed by switching to the **Operators → Installed Operators** page.
2. Ensure that **OpenShift Elasticsearch Operator** is listed in all projects with a **Status** of **Succeeded**.

12.2.3.3. Installing the OpenShift Elasticsearch Operator by using the CLI

You can use the OpenShift CLI (**oc**) to install the OpenShift Elasticsearch Operator.

Prerequisites

- Ensure that you have the necessary persistent storage for Elasticsearch. Note that each Elasticsearch node requires its own storage volume.

  NOTE

  If you use a local volume for persistent storage, do not use a raw block volume, which is described with **volumeMode: block** in the **LocalVolume** object. Elasticsearch cannot use raw block volumes.

  Elasticsearch is a memory-intensive application. By default, Red Hat OpenShift Service on AWS installs three Elasticsearch nodes with memory requests and limits of 16 GB. This initial set of three Red Hat OpenShift Service on AWS nodes might not have enough memory to run Elasticsearch within your cluster. If you experience memory issues that are related to Elasticsearch, add more Elasticsearch nodes to your cluster rather than increasing the memory on existing nodes.

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **Namespace** object as a YAML file:

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

(1) You must specify the **openshift-operators-redhat** namespace. To prevent possible conflicts with metrics, configure the Prometheus Cluster Monitoring stack 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 a ROSA metric, which would cause conflicts.

(2) String. You must specify this 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 as a YAML file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
```

```
  name: openshift-operators-redhat
  namespace: openshift-operators-redhat (1)
spec: {}
```

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

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

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

5. Create a **Subscription** object to subscribe the namespace to the OpenShift Elasticsearch Operator:

Example Subscription

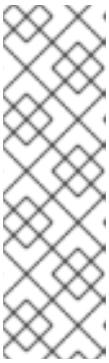
```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: elasticsearch-operator
  namespace: openshift-operators-redhat (1)
spec:
  channel: stable-x.y (2)
  installPlanApproval: Automatic (3)
  source: redhat-operators (4)
  sourceNamespace: openshift-marketplace
  name: elasticsearch-operator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

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

(2) Specify **stable**, or **stable-x.y** as the channel. See the following note.

(3) **Automatic** allows the Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available. **Manual** requires a user with appropriate credentials to approve the Operator update.

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



NOTE

Specifying **stable** installs the current version of the latest stable release. Using **stable** with **installPlanApproval: "Automatic"** automatically upgrades your Operators to the latest stable major and minor release.

Specifying **stable-x.y** installs the current minor version of a specific major release. Using **stable-x.y** with **installPlanApproval: "Automatic"** automatically upgrades your Operators to the latest stable minor release within the major release.

6. Apply the subscription by running the following command:

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

The OpenShift Elasticsearch Operator is installed to the **openshift-operators-redhat** namespace and copied to each project in the cluster.

Verification

1. Run the following command:

```
$ oc get csv -n --all-namespaces
```

2. Observe the output and confirm that pods for the OpenShift Elasticsearch Operator exist in each namespace

Example output

```
NAMESPACE                                NAME                        DISPLAY
VERSION         REPLACES                    PHASE
default                                  elasticsearch-operator.v5.8.1   OpenShift Elasticsearch
Operator   5.8.1          elasticsearch-operator.v5.8.0   Succeeded
kube-node-lease                            elasticsearch-operator.v5.8.1   OpenShift
Elasticsearch Operator   5.8.1          elasticsearch-operator.v5.8.0   Succeeded
kube-public                              elasticsearch-operator.v5.8.1   OpenShift Elasticsearch
Operator   5.8.1          elasticsearch-operator.v5.8.0   Succeeded
kube-system                              elasticsearch-operator.v5.8.1   OpenShift Elasticsearch
Operator   5.8.1          elasticsearch-operator.v5.8.0   Succeeded
non-destructive-test                       elasticsearch-operator.v5.8.1   OpenShift
Elasticsearch Operator   5.8.1          elasticsearch-operator.v5.8.0   Succeeded
openshift-apiserver-operator                 elasticsearch-operator.v5.8.1   OpenShift
Elasticsearch Operator   5.8.1          elasticsearch-operator.v5.8.0   Succeeded
openshift-apiserver                        elasticsearch-operator.v5.8.1   OpenShift
Elasticsearch Operator   5.8.1          elasticsearch-operator.v5.8.0   Succeeded
...
```

12.2.4. Configuring log storage

You can configure which log storage type your logging uses by modifying the **ClusterLogging** custom resource (CR).

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator and an internal log store that is either the LokiStack or Elasticsearch.
- You have created a **ClusterLogging** CR.

NOTE

The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

Procedure

1. Modify the **ClusterLogging** CR **logStore** spec:

ClusterLogging CR example

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
# ...
spec:
# ...
  logStore:
    type: <log_store_type> (1)
    elasticsearch: (2)
      nodeCount: <integer>
      resources: {}
      storage: {}
      redundancyPolicy: <redundancy_type> (3)
    lokistack: (4)
      name: {}
# ...
```

1. Specify the log store type. This can be either **lokistack** or **elasticsearch**.
2. Optional configuration options for the Elasticsearch log store.
3. Specify the redundancy type. This value can be **ZeroRedundancy**, **SingleRedundancy**, **MultipleRedundancy**, or **FullRedundancy**.
4. Optional configuration options for LokiStack.

Example ClusterLogging CR to specify LokiStack as the log store

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

```
    lokistack:
      name: logging-loki
# ...
```

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

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

12.3. CONFIGURING THE LOKISTACK LOG STORE

In logging documentation, *LokiStack* refers to the logging supported combination of Loki and web proxy with Red Hat OpenShift Service on AWS authentication integration. LokiStack's proxy uses Red Hat OpenShift Service on AWS authentication to enforce multi-tenancy. *Loki* refers to the log store as either the individual component or an external store.

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

IMPORTANT

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

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

Procedure

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

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

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

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

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

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

12.3.2. LokiStack behavior during cluster restarts

In logging version 5.8 and newer versions, when an Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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.

Additional resources

- Pod disruption budgets Kubernetes documentation

12.3.3. Configuring Loki to tolerate node failure

In the logging 5.8 and later versions, 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 Red Hat OpenShift Service on AWS, *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.

Additional resources

- **PodAntiAffinity** v1 core Kubernetes documentation
- Assigning Pods to Nodes Kubernetes documentation
- Placing pods relative to other pods using affinity and anti-affinity rules

12.3.4. Zone aware data replication

In the logging 5.8 and later versions, 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.

12.3.4.1. Recovering Loki pods from failed zones

In Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS cluster isn't 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

- Logging version 5.8 or later.
- 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. Then 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.

12.3.4.1.1. 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.

1. 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
```

Additional resources

- Topology spread constraints Kubernetes documentation
- Kubernetes storage documentation .

12.3.5. Fine grained access for Loki logs

In logging 5.8 and later, the Red Hat OpenShift Logging Operator does not grant all users access to logs by default. As an administrator, you must configure your users' access unless the Operator was upgraded and prior configurations are in place. Depending on your configuration and need, you can configure fine grain access to logs using the following:

- Cluster wide policies
- Namespace scoped policies
- Creation of custom admin groups

As an administrator, you need to create the role bindings and cluster role bindings appropriate for your deployment. The Red Hat OpenShift Logging Operator provides the following cluster roles:

- **cluster-logging-application-view** grants permission to read application logs.
- **cluster-logging-infrastructure-view** grants permission to read infrastructure logs.
- **cluster-logging-audit-view** grants permission to read audit logs.

If you have upgraded from a prior version, an additional cluster role **logging-application-logs-reader** and associated cluster role binding **logging-all-authenticated-application-logs-reader** provide backward compatibility, allowing any authenticated user read access in their namespaces.

NOTE

Users with access by namespace must provide a namespace when querying application logs.

12.3.5.1. Cluster wide access

Cluster role binding resources reference cluster roles, and set permissions cluster wide.

Example ClusterRoleBinding

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: logging-all-application-logs-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-logging-application-view (1)
subjects: (2)
- kind: Group
  name: system:authenticated
  apiGroup: rbac.authorization.k8s.io
```

1. Additional **ClusterRoles** are **cluster-logging-infrastructure-view**, and **cluster-logging-audit-view**.
2. Specifies the users or groups this object applies to.

12.3.5.2. Namespaced access

RoleBinding resources can be used with **ClusterRole** objects to define the namespace a user or group has access to logs for.

Example RoleBinding

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: allow-read-logs
  namespace: log-test-0 (1)
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-logging-application-view
subjects:
- kind: User
  apiGroup: rbac.authorization.k8s.io
  name: testuser-0
```

1. Specifies the namespace this **RoleBinding** applies to.

12.3.5.3. Custom admin group access

If you have a large deployment with a number of users who require broader permissions, you can create a custom group using the **adminGroup** field. Users who are members of any group specified in the **adminGroups** field of the **LokiStack** CR are considered admins. Admin users have access to all application logs in all namespaces, if they also get assigned the **cluster-logging-application-view** role.

Example LokiStack CR

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

1. Custom admin groups are only available in this mode.
2. Entering an empty list **[]** value for this field disables admin groups.
3. Overrides the default groups (**system:cluster-admins**, **cluster-admin**, **dedicated-admin**)

12.3.6. Enabling stream-based retention with Loki

Additional resources

With Logging version 5.6 and higher, 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.

1. To enable stream-based retention, create a **LokiStack** custom resource (CR):

   Example global stream-based retention

```
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: v11
    secret:
      name: logging-loki-s3
      type: aws
  storageClassName: standard
  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.

Example per-tenant stream-based retention

```
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: v11
    secret:
      name: logging-loki-s3
      type: aws
  storageClassName: standard
  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.

12.3.7. 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.

12.3.8. Configuring Loki to tolerate memberlist creation failure

In an OpenShift 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 CR to use the **podIP** 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
# ...
```

12.3.9. Additional Resources

- Loki components documentation
- Loki Query Language (LogQL) documentation
- Grafana Dashboard documentation
- Loki Object Storage documentation
- Loki Operator **IngestionLimitSpec** documentation
- Loki Storage Schema documentation

12.4. CONFIGURING THE ELASTICSEARCH LOG STORE

You can use Elasticsearch 6 to store and organize log data.

You can make modifications to your log store, including:

- Storage for your Elasticsearch cluster
- Shard replication across data nodes in the cluster, from full replication to no replication
- External access to Elasticsearch data

12.4.1. Configuring log storage

You can configure which log storage type your logging uses by modifying the **ClusterLogging** custom resource (CR).

Prerequisites

- You have administrator permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed the Red Hat OpenShift Logging Operator and an internal log store that is either the LokiStack or Elasticsearch.
- You have created a **ClusterLogging** CR.

NOTE

The Logging 5.9 release does not contain an updated version of the OpenShift Elasticsearch Operator. If you currently use the OpenShift Elasticsearch Operator released with Logging 5.8, it will continue to work with Logging until the EOL of Logging 5.8. As an alternative to using the OpenShift Elasticsearch Operator to manage the default log storage, you can use the Loki Operator. For more information on the Logging lifecycle dates, see Platform Agnostic Operators.

Procedure

1. Modify the **ClusterLogging** CR **logStore** spec:

ClusterLogging CR example

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
# ...
spec:
# ...
  logStore:
    type: <log_store_type> 1
    elasticsearch: 2
      nodeCount: <integer>
      resources: {}
      storage: {}
      redundancyPolicy: <redundancy_type> 3
```

```
    lokistack: 4
      name: {}
# ...
```

1. Specify the log store type. This can be either **lokistack** or **elasticsearch**.
2. Optional configuration options for the Elasticsearch log store.
3. Specify the redundancy type. This value can be **ZeroRedundancy**, **SingleRedundancy**, **MultipleRedundancy**, or **FullRedundancy**.
4. Optional configuration options for LokiStack.

Example ClusterLogging CR to specify LokiStack as the log store

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

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

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

12.4.2. Forwarding audit logs to the log store

In a logging deployment, container and infrastructure logs are forwarded to the internal log store defined in the **ClusterLogging** custom resource (CR) by default.

Audit logs are not forwarded to the internal log store by default because this does not provide secure storage. You are responsible for ensuring that the system to which you forward audit logs is compliant with your organizational and governmental regulations, and is properly secured.

If this default configuration meets your needs, you do not need to configure a **ClusterLogForwarder** CR. If a **ClusterLogForwarder** CR exists, logs are not forwarded to the internal log store unless a pipeline is defined that contains the **default** output.

Procedure

To use the Log Forward API to forward audit logs to the internal Elasticsearch instance:

1. Create or edit a YAML file that defines the **ClusterLogForwarder** CR object:
   - Create a CR to send all log types to the internal Elasticsearch instance. You can use the following example without making any changes:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  pipelines: (1)
  - name: all-to-default
    inputRefs:
    - infrastructure
    - application
    - audit
    outputRefs:
    - default
```

(1) A pipeline defines the type of logs to forward using the specified output. The default output forwards logs to the internal Elasticsearch instance.

NOTE

You must specify all three types of logs in the pipeline: application, infrastructure, and audit. If you do not specify a log type, those logs are not stored and will be lost.

- If you have an existing **ClusterLogForwarder** CR, add a pipeline to the default output for the audit logs. You do not need to define the default output. For example:

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
metadata:
  name: instance
  namespace: openshift-logging
spec:
  outputs:
   - name: elasticsearch-insecure
     type: "elasticsearch"
     url: http://elasticsearch-insecure.messaging.svc.cluster.local
     insecure: true
   - name: elasticsearch-secure
     type: "elasticsearch"
     url: https://elasticsearch-secure.messaging.svc.cluster.local
     secret:
       name: es-audit
   - name: secureforward-offcluster
     type: "fluentdForward"
     url: https://secureforward.offcluster.com:24224
     secret:
       name: secureforward
  pipelines:
   - name: container-logs
     inputRefs:
     - application
     outputRefs:
     - secureforward-offcluster
```

```
- name: infra-logs
  inputRefs:
  - infrastructure
  outputRefs:
  - elasticsearch-insecure
- name: audit-logs
  inputRefs:
  - audit
  outputRefs:
  - elasticsearch-secure
  - default (1)
```

(1) This pipeline sends the audit logs to the internal Elasticsearch instance in addition to an external instance.

Additional resources

- About log collection and forwarding

12.4.3. Configuring log retention time

You can configure a *retention policy* that specifies how long the default Elasticsearch log store keeps indices for each of the three log sources: infrastructure logs, application logs, and audit logs.

To configure the retention policy, you set a **maxAge** parameter for each log source in the **ClusterLogging** custom resource (CR). The CR applies these values to the Elasticsearch rollover schedule, which determines when Elasticsearch deletes the rolled-over indices.

Elasticsearch rolls over an index, moving the current index and creating a new index, when an index matches any of the following conditions:

- The index is older than the **rollover.maxAge** value in the **Elasticsearch** CR.
- The index size is greater than 40 GB × the number of primary shards.
- The index doc count is greater than 40960 KB × the number of primary shards.

Elasticsearch deletes the rolled-over indices based on the retention policy you configure. If you do not create a retention policy for any log sources, logs are deleted after seven days by default.

Prerequisites

- The Red Hat OpenShift Logging Operator and the OpenShift Elasticsearch Operator must be installed.

Procedure

To configure the log retention time:

1. Edit the **ClusterLogging** CR to add or modify the **retentionPolicy** parameter:

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

```
managementState: "Managed"
logStore:
  type: "elasticsearch"
  retentionPolicy: (1)
    application:
      maxAge: 1d
    infra:
      maxAge: 7d
    audit:
      maxAge: 7d
  elasticsearch:
    nodeCount: 3
...
```

1. Specify the time that Elasticsearch should retain each log source. Enter an integer and a time designation: weeks(w), hours(h/H), minutes(m) and seconds(s). For example, **1d** for one day. Logs older than the **maxAge** are deleted. By default, logs are retained for seven days.

2. You can verify the settings in the **Elasticsearch** custom resource (CR).
   For example, the Red Hat OpenShift Logging Operator updated the following **Elasticsearch** CR to configure a retention policy that includes settings to roll over active indices for the infrastructure logs every eight hours and the rolled-over indices are deleted seven days after rollover. Red Hat OpenShift Service on AWS checks every 15 minutes to determine if the indices need to be rolled over.

```
apiVersion: "logging.openshift.io/v1"
kind: "Elasticsearch"
metadata:
  name: "elasticsearch"
spec:
...
  indexManagement:
    policies: (1)
      - name: infra-policy
        phases:
          delete:
            minAge: 7d (2)
          hot:
            actions:
              rollover:
                maxAge: 8h (3)
        pollInterval: 15m (4)
...
```

1. For each log source, the retention policy indicates when to delete and roll over logs for that source.

2. When Red Hat OpenShift Service on AWS deletes the rolled-over indices. This setting is the **maxAge** you set in the **ClusterLogging** CR.

3. The index age for Red Hat OpenShift Service on AWS to consider when rolling over the indices. This value is determined from the **maxAge** you set in the **ClusterLogging** CR.

4 When Red Hat OpenShift Service on AWS checks if the indices should be rolled over. This setting is the default and cannot be changed.

> **NOTE**
>
> Modifying the **Elasticsearch** CR is not supported. All changes to the retention policies must be made in the **ClusterLogging** CR.

The OpenShift Elasticsearch Operator deploys a cron job to roll over indices for each mapping using the defined policy, scheduled using the **pollInterval**.

```
$ oc get cronjob
```

Example output

```
NAME                    SCHEDULE      SUSPEND   ACTIVE   LAST SCHEDULE   AGE
elasticsearch-im-app     */15 * * * *   False     0       <none>          4s
elasticsearch-im-audit   */15 * * * *   False     0       <none>          4s
elasticsearch-im-infra   */15 * * * *   False     0       <none>          4s
```

12.4.4. Configuring CPU and memory requests for the log store

Each component specification allows for adjustments to both the CPU and memory requests. You should not have to manually adjust these values as the OpenShift Elasticsearch Operator sets values sufficient for your environment.

> **NOTE**
>
> In large-scale clusters, the default memory limit for the Elasticsearch proxy container might not be sufficient, causing the proxy container to be OOMKilled. If you experience this issue, increase the memory requests and limits for the Elasticsearch proxy.

Each Elasticsearch node can operate with a lower memory setting though this is **not** recommended for production deployments. For production use, you should have no less than the default 16Gi allocated to each pod. Preferably you should allocate as much as possible, up to 64Gi per pod.

Prerequisites

- The Red Hat OpenShift Logging and Elasticsearch Operators must be installed.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc edit ClusterLogging instance
```

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
....
```

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch: (1)
      resources:
        limits: (2)
          memory: "32Gi"
        requests: (3)
          cpu: "1"
          memory: "16Gi"
      proxy: (4)
        resources:
          limits:
            memory: 100Mi
          requests:
            memory: 100Mi
```

1. Specify the CPU and memory requests for Elasticsearch as needed. If you leave these values blank, the OpenShift Elasticsearch Operator sets default values that should be sufficient for most deployments. The default values are **16Gi** for the memory request and **1** for the CPU request.
2. The maximum amount of resources a pod can use.
3. The minimum resources required to schedule a pod.
4. Specify the CPU and memory requests for the Elasticsearch proxy as needed. If you leave these values blank, the OpenShift Elasticsearch Operator sets default values that are sufficient for most deployments. The default values are **256Mi** for the memory request and **100m** for the CPU request.

When adjusting the amount of Elasticsearch memory, the same value should be used for both **requests** and **limits**.

For example:

```
resources:
  limits: (1)
    memory: "32Gi"
  requests: (2)
    cpu: "8"
    memory: "32Gi"
```

1. The maximum amount of the resource.
2. The minimum amount required.

Kubernetes generally adheres the node configuration and does not allow Elasticsearch to use the specified limits. Setting the same value for the **requests** and **limits** ensures that Elasticsearch can use the memory you want, assuming the node has the memory available.

12.4.5. Configuring replication policy for the log store

You can define how Elasticsearch shards are replicated across data nodes in the cluster.

Prerequisites

- The Red Hat OpenShift Logging and Elasticsearch Operators must be installed.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc edit clusterlogging instance
```

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"

....

spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      redundancyPolicy: "SingleRedundancy" (1)
```

(1) Specify a redundancy policy for the shards. The change is applied upon saving the changes.

- **FullRedundancy**. Elasticsearch fully replicates the primary shards for each index to every data node. This provides the highest safety, but at the cost of the highest amount of disk required and the poorest performance.
- **MultipleRedundancy**. Elasticsearch fully replicates the primary shards for each index to half of the data nodes. This provides a good tradeoff between safety and performance.
- **SingleRedundancy**. Elasticsearch makes one copy of the primary shards for each index. Logs are always available and recoverable as long as at least two data nodes exist. Better performance than MultipleRedundancy, when using 5 or more nodes. You cannot apply this policy on deployments of single Elasticsearch node.
- **ZeroRedundancy**. Elasticsearch does not make copies of the primary shards. Logs might be unavailable or lost in the event a node is down or fails. Use this mode when you are more concerned with performance than safety, or have implemented your own disk/PVC backup/restore strategy.

NOTE

The number of primary shards for the index templates is equal to the number of Elasticsearch data nodes.

12.4.6. Scaling down Elasticsearch pods

Reducing the number of Elasticsearch pods in your cluster can result in data loss or Elasticsearch performance degradation.

If you scale down, you should scale down by one pod at a time and allow the cluster to re-balance the shards and replicas. After the Elasticsearch health status returns to **green**, you can scale down by another pod.

NOTE

If your Elasticsearch cluster is set to **ZeroRedundancy**, you should not scale down your Elasticsearch pods.

12.4.7. Configuring persistent storage for the log store

Elasticsearch requires persistent storage. The faster the storage, the faster the Elasticsearch performance.

WARNING

Using NFS storage as a volume or a persistent volume (or via NAS such as Gluster) is not supported for Elasticsearch storage, as Lucene relies on file system behavior that NFS does not supply. Data corruption and other problems can occur.

Prerequisites

- The Red Hat OpenShift Logging and Elasticsearch Operators must be installed.

Procedure

1. Edit the **ClusterLogging** CR to specify that each data node in the cluster is bound to a Persistent Volume Claim.

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
# ...
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage:
        storageClassName: "gp2"
        size: "200G"
```

This example specifies each data node in the cluster is bound to a Persistent Volume Claim that requests "200G" of AWS General Purpose SSD (gp2) storage.

NOTE

If you use a local volume for persistent storage, do not use a raw block volume, which is described with **volumeMode: block** in the **LocalVolume** object. Elasticsearch cannot use raw block volumes.

12.4.8. Configuring the log store for emptyDir storage

You can use emptyDir with your log store, which creates an ephemeral deployment in which all of a pod's data is lost upon restart.

NOTE

When using emptyDir, if log storage is restarted or redeployed, you will lose data.

Prerequisites

- The Red Hat OpenShift Logging and Elasticsearch Operators must be installed.

Procedure

1. Edit the **ClusterLogging** CR to specify emptyDir:

```
spec:
  logStore:
    type: "elasticsearch"
    elasticsearch:
      nodeCount: 3
      storage: {}
```

12.4.9. Performing an Elasticsearch rolling cluster restart

Perform a rolling restart when you change the **elasticsearch** config map or any of the **elasticsearch-*** deployment configurations.

Also, a rolling restart is recommended if the nodes on which an Elasticsearch pod runs requires a reboot.

Prerequisites

- The Red Hat OpenShift Logging and Elasticsearch Operators must be installed.

Procedure

To perform a rolling cluster restart:

1. Change to the **openshift-logging** project:

```
$ oc project openshift-logging
```

2. Get the names of the Elasticsearch pods:

```
$ oc get pods -l component=elasticsearch
```

3. Scale down the collector pods so they stop sending new logs to Elasticsearch:

```
$ oc -n openshift-logging patch daemonset/collector -p '{"spec":{"template":{"spec":
{"nodeSelector":{"logging-infra-collector": "false"}}}}}'
```

4. Perform a shard synced flush using the Red Hat OpenShift Service on AWS **es_util** tool to ensure there are no pending operations waiting to be written to disk prior to shutting down:

```
$ oc exec <any_es_pod_in_the_cluster> -c elasticsearch -- es_util --query="_flush/synced" -
XPOST
```

For example:

```
$ oc exec -c elasticsearch-cdm-5ceex6ts-1-dcd6c4c7c-jpw6  -c elasticsearch -- es_util --
query="_flush/synced" -XPOST
```

Example output

```
{"_shards":{"total":4,"successful":4,"failed":0},".security":
{"total":2,"successful":2,"failed":0},".kibana_1":{"total":2,"successful":2,"failed":0}}
```

5. Prevent shard balancing when purposely bringing down nodes using the Red Hat OpenShift Service on AWS es_util tool:

```
$ oc exec <any_es_pod_in_the_cluster> -c elasticsearch -- es_util --
query="_cluster/settings" -XPUT -d '{ "persistent": { "cluster.routing.allocation.enable" :
"primaries" } }'
```

For example:

```
$ oc exec elasticsearch-cdm-5ceex6ts-1-dcd6c4c7c-jpw6 -c elasticsearch -- es_util --
query="_cluster/settings" -XPUT -d '{ "persistent": { "cluster.routing.allocation.enable" :
"primaries" } }'
```

Example output

```
{"acknowledged":true,"persistent":{"cluster":{"routing":{"allocation":
{"enable":"primaries"}}}},"transient":
```

6. After the command is complete, for each deployment you have for an ES cluster:

   a. By default, the Red Hat OpenShift Service on AWS Elasticsearch cluster blocks rollouts to their nodes. Use the following command to allow rollouts and allow the pod to pick up the changes:

   ```
   $ oc rollout resume deployment/<deployment-name>
   ```

   For example:

   ```
   $ oc rollout resume deployment/elasticsearch-cdm-0-1
   ```

   Example output

   ```
   deployment.extensions/elasticsearch-cdm-0-1 resumed
   ```

A new pod is deployed. After the pod has a ready container, you can move on to the next deployment.

```
$ oc get pods -l component=elasticsearch-
```

Example output

```
NAME                                            READY   STATUS    RESTARTS   AGE
elasticsearch-cdm-5ceex6ts-1-dcd6c4c7c-jpw6k    2/2     Running   0          22h
elasticsearch-cdm-5ceex6ts-2-f799564cb-l9mj7    2/2     Running   0          22h
elasticsearch-cdm-5ceex6ts-3-585968dc68-k7kjr   2/2     Running   0          22h
```

b. After the deployments are complete, reset the pod to disallow rollouts:

```
$ oc rollout pause deployment/<deployment-name>
```

For example:

```
$ oc rollout pause deployment/elasticsearch-cdm-0-1
```

Example output

```
deployment.extensions/elasticsearch-cdm-0-1 paused
```

c. Check that the Elasticsearch cluster is in a **green** or **yellow** state:

```
$ oc exec <any_es_pod_in_the_cluster> -c elasticsearch -- es_util --
query=_cluster/health?pretty=true
```

NOTE

If you performed a rollout on the Elasticsearch pod you used in the previous commands, the pod no longer exists and you need a new pod name here.

For example:

```
$ oc exec elasticsearch-cdm-5ceex6ts-1-dcd6c4c7c-jpw6 -c elasticsearch -- es_util --
query=_cluster/health?pretty=true
```

```
{
  "cluster_name" : "elasticsearch",
  "status" : "yellow", (1)
  "timed_out" : false,
  "number_of_nodes" : 3,
  "number_of_data_nodes" : 3,
  "active_primary_shards" : 8,
  "active_shards" : 16,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 1,
```

```
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 100.0
}
```

1. Make sure this parameter value is **green** or **yellow** before proceeding.

7. If you changed the Elasticsearch configuration map, repeat these steps for each Elasticsearch pod.

8. After all the deployments for the cluster have been rolled out, re-enable shard balancing:

```
$ oc exec <any_es_pod_in_the_cluster> -c elasticsearch -- es_util --
query="_cluster/settings" -XPUT -d '{ "persistent": { "cluster.routing.allocation.enable" : "all" }
}'
```

For example:

```
$ oc exec elasticsearch-cdm-5ceex6ts-1-dcd6c4c7c-jpw6 -c elasticsearch -- es_util --
query="_cluster/settings" -XPUT -d '{ "persistent": { "cluster.routing.allocation.enable" : "all" }
}'
```

Example output

```
{
  "acknowledged" : true,
  "persistent" : { },
  "transient" : {
    "cluster" : {
      "routing" : {
        "allocation" : {
          "enable" : "all"
        }
      }
    }
  }
}
```

9. Scale up the collector pods so they send new logs to Elasticsearch.

```
$ oc -n openshift-logging patch daemonset/collector -p '{"spec":{"template":{"spec":
{"nodeSelector":{"logging-infra-collector": "true"}}}}}'
```

12.4.10. Exposing the log store service as a route

By default, the log store that is deployed with logging is not accessible from outside the logging cluster. You can enable a route with re-encryption termination for external access to the log store service for those tools that access its data.

Externally, you can access the log store by creating a reencrypt route, your Red Hat OpenShift Service on AWS token and the installed log store CA certificate. Then, access a node that hosts the log store service with a cURL request that contains:

- The **Authorization: Bearer ${token}**
- The Elasticsearch reencrypt route and an Elasticsearch API request.

Internally, you can access the log store service using the log store cluster IP, which you can get by using either of the following commands:

```
$ oc get service elasticsearch -o jsonpath={.spec.clusterIP} -n openshift-logging
```

Example output

```
172.30.183.229
```

```
$ oc get service elasticsearch -n openshift-logging
```

Example output

```
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
elasticsearch   ClusterIP   172.30.183.229   <none>        9200/TCP   22h
```

You can check the cluster IP address with a command similar to the following:

```
$ oc exec elasticsearch-cdm-oplnhinv-1-5746475887-fj2f8 -n openshift-logging -- curl -tlsv1.2 --
insecure -H "Authorization: Bearer ${token}" "https://172.30.183.229:9200/_cat/health"
```

Example output

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    29  100    29    0     0    108      0 --:--:-- --:--:-- --:--:--   108
```

Prerequisites

- The Red Hat OpenShift Logging and Elasticsearch Operators must be installed.
- You must have access to the project to be able to access to the logs.

Procedure

To expose the log store externally:

1. Change to the **openshift-logging** project:

   ```
   $ oc project openshift-logging
   ```

2. Extract the CA certificate from the log store and write to the ***admin-ca*** file:

   ```
   $ oc extract secret/elasticsearch --to=. --keys=admin-ca
   ```

Example output

```
admin-ca
```

3. Create the route for the log store service as a YAML file:

   a. Create a YAML file with the following:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: elasticsearch
  namespace: openshift-logging
spec:
  host:
  to:
    kind: Service
    name: elasticsearch
  tls:
    termination: reencrypt
    destinationCACertificate: | (1)
```

   (1) Add the log store CA certifcate or use the command in the next step. You do not have to set the **spec.tls.key**, **spec.tls.certificate**, and **spec.tls.caCertificate** parameters required by some reencrypt routes.

   b. Run the following command to add the log store CA certificate to the route YAML you created in the previous step:

```
$ cat ./admin-ca | sed -e "s/^/      /" >> <file-name>.yaml
```

   c. Create the route:

```
$ oc create -f <file-name>.yaml
```

   Example output

```
route.route.openshift.io/elasticsearch created
```

4. Check that the Elasticsearch service is exposed:

   a. Get the token of this service account to be used in the request:

```
$ token=$(oc whoami -t)
```

   b. Set the **elasticsearch** route you created as an environment variable.

```
$ routeES=`oc get route elasticsearch -o jsonpath={.spec.host}`
```

   c. To verify the route was successfully created, run the following command that accesses Elasticsearch through the exposed route:

```
curl -tlsv1.2 --insecure -H "Authorization: Bearer ${token}" "https://${routeES}"
```

The response appears similar to the following:

Example output

```
{
  "name" : "elasticsearch-cdm-i40ktba0-1",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "0eY-tJzcR3KOdpgeMJo-MQ",
  "version" : {
  "number" : "6.8.1",
  "build_flavor" : "oss",
  "build_type" : "zip",
  "build_hash" : "Unknown",
  "build_date" : "Unknown",
  "build_snapshot" : true,
  "lucene_version" : "7.7.0",
  "minimum_wire_compatibility_version" : "5.6.0",
  "minimum_index_compatibility_version" : "5.0.0"
},
  "<tagline>" : "<for search>"
}
```

12.4.11. Removing unused components if you do not use the default Elasticsearch log store

As an administrator, in the rare case that you forward logs to a third-party log store and do not use the default Elasticsearch log store, you can remove several unused components from your logging cluster.

In other words, if you do not use the default Elasticsearch log store, you can remove the internal Elasticsearch **logStore** and Kibana **visualization** components from the **ClusterLogging** custom resource (CR). Removing these components is optional but saves resources.

Prerequisites

- Verify that your log forwarder does not send log data to the default internal Elasticsearch cluster. Inspect the **ClusterLogForwarder** CR YAML file that you used to configure log forwarding. Verify that it *does not* have an **outputRefs** element that specifies **default**. For example:

  ```
  outputRefs:
  - default
  ```

> **WARNING**
>
> Suppose the **ClusterLogForwarder** CR forwards log data to the internal Elasticsearch cluster, and you remove the **logStore** component from the **ClusterLogging** CR. In that case, the internal Elasticsearch cluster will not be present to store the log data. This absence can cause data loss.

Procedure

1. Edit the **ClusterLogging** custom resource (CR) in the **openshift-logging** project:

```
$ oc edit ClusterLogging instance
```

2. If they are present, remove the **logStore** and **visualization** stanzas from the **ClusterLogging** CR.

3. Preserve the **collection** stanza of the **ClusterLogging** CR. The result should look similar to the following example:

```
apiVersion: "logging.openshift.io/v1"
kind: "ClusterLogging"
metadata:
  name: "instance"
  namespace: "openshift-logging"
spec:
  managementState: "Managed"
  collection:
    type: "fluentd"
    fluentd: {}
```

4. Verify that the collector pods are redeployed:

```
$ oc get pods -l component=collector -n openshift-logging
```

CHAPTER 13. LOGGING ALERTS

13.1. DEFAULT LOGGING ALERTS

Logging alerts are installed as part of the Red Hat OpenShift Logging Operator installation. Alerts depend on metrics exported by the log collection and log storage backends. These metrics are enabled if you selected the option to **Enable operator recommended cluster monitoring on this namespace** when installing the Red Hat OpenShift Logging Operator. For more information about installing logging Operators, see Installing logging using the web console .

Default logging alerts are sent to the Red Hat OpenShift Service on AWS monitoring stack Alertmanager in the **openshift-monitoring** namespace, unless you have disabled the local Alertmanager instance.

13.1.1. Accessing the Alerting UI in the Administrator and Developer perspectives

The Alerting UI is accessible through the **Administrator** perspective and the **Developer** perspective of the Red Hat OpenShift Service on AWS web console.

- In the **Administrator** perspective, go to **Observe → Alerting**. The three main pages in the Alerting UI in this perspective are the **Alerts**, **Silences**, and **Alerting rules** pages.
- In the **Developer** perspective, go to **Observe → <project_name> → Alerts**. In this perspective, alerts, silences, and alerting rules are all managed from the **Alerts** page. The results shown in the **Alerts** page are specific to the selected project.

NOTE

In the **Developer** perspective, you can select from core Red Hat OpenShift Service on AWS and user-defined projects that you have access to in the **Project: <project_name>** list. However, alerts, silences, and alerting rules relating to core Red Hat OpenShift Service on AWS projects are not displayed if you are not logged in as a cluster administrator.

13.1.2. Vector collector alerts

In logging 5.7 and later versions, the following alerts are generated by the Vector collector. You can view these alerts in the Red Hat OpenShift Service on AWS web console.

Table 13.1. Vector collector alerts

Alert	Message	Description	Severity
CollectorHighErrorRate	**<value> of records have resulted in an error by vector <instance>.**	The number of vector output errors is high, by default more than 10 in the previous 15 minutes.	Warning
CollectorNodeDown	**Prometheus could not scrape vector <instance> for more than 10m.**	Vector is reporting that Prometheus could not scrape a specific Vector instance.	Critical

Alert	Message	Description	Severity
CollectorVeryHighErrorRate	**<value> of records have resulted in an error by vector <instance>.**	The number of Vector component errors are very high, by default more than 25 in the previous 15 minutes.	Critical
FluentdQueueLengthIncreasing	**In the last 1h, fluentd <instance> buffer queue length constantly increased more than 1. Current value is <value>.**	Fluentd is reporting that the queue size is increasing.	Warning

13.1.3. Fluentd collector alerts

The following alerts are generated by the legacy Fluentd log collector. You can view these alerts in the Red Hat OpenShift Service on AWS web console.

Table 13.2. Fluentd collector alerts

Alert	Message	Description	Severity
FluentDHighErrorRate	**<value> of records have resulted in an error by fluentd <instance>.**	The number of FluentD output errors is high, by default more than 10 in the previous 15 minutes.	Warning
FluentdNodeDown	**Prometheus could not scrape fluentd <instance> for more than 10m.**	Fluentd is reporting that Prometheus could not scrape a specific Fluentd instance.	Critical
FluentdQueueLengthIncreasing	**In the last 1h, fluentd <instance> buffer queue length constantly increased more than 1. Current value is <value>.**	Fluentd is reporting that the queue size is increasing.	Warning
FluentDVeryHighErrorRate	**<value> of records have resulted in an error by fluentd <instance>.**	The number of FluentD output errors is very high, by default more than 25 in the previous 15 minutes.	Critical

13.1.4. Elasticsearch alerting rules

You can view these alerting rules in the Red Hat OpenShift Service on AWS web console.

Table 13.3. Alerting rules

Alert	Description	Severity
ElasticsearchClusterNotHealthy	The cluster health status has been RED for at least 2 minutes. The cluster does not accept writes, shards may be missing, or the master node hasn't been elected yet.	Critical
ElasticsearchClusterNotHealthy	The cluster health status has been YELLOW for at least 20 minutes. Some shard replicas are not allocated.	Warning
ElasticsearchDiskSpaceRunningLow	The cluster is expected to be out of disk space within the next 6 hours.	Critical
ElasticsearchHighFileDescriptorUsage	The cluster is predicted to be out of file descriptors within the next hour.	Warning
ElasticsearchJVMHeapUseHigh	The JVM Heap usage on the specified node is high.	Alert
ElasticsearchNodeDiskWatermarkReached	The specified node has hit the low watermark due to low free disk space. Shards can not be allocated to this node anymore. You should consider adding more disk space to the node.	Info
ElasticsearchNodeDiskWatermarkReached	The specified node has hit the high watermark due to low free disk space. Some shards will be re-allocated to different nodes if possible. Make sure more disk space is added to the node or drop old indices allocated to this node.	Warning
ElasticsearchNodeDiskWatermarkReached	The specified node has hit the flood watermark due to low free disk space. Every index that has a shard allocated on this node is enforced a read-only block. The index block must be manually released when the disk use falls below the high watermark.	Critical
ElasticsearchJVMHeapUseHigh	The JVM Heap usage on the specified node is too high.	Alert
ElasticsearchWriteRequestsRejectionJumps	Elasticsearch is experiencing an increase in write rejections on the specified node. This node might not be keeping up with the indexing speed.	Warning
AggregatedLoggingSystemCPUHigh	The CPU used by the system on the specified node is too high.	Alert
ElasticsearchProcessCPUHigh	The CPU used by Elasticsearch on the specified node is too high.	Alert

13.1.5. Additional resources

- Modifying core platform alerting rules

13.2. CUSTOM LOGGING ALERTS

In logging 5.7 and later versions, users can configure the LokiStack deployment to produce customized alerts and recorded metrics. If you want to use customized alerting and recording rules, you must enable the LokiStack ruler component.

LokiStack log-based alerts and recorded metrics are triggered by providing LogQL expressions to the ruler component. The Loki Operator manages a ruler that is optimized for the selected LokiStack size, which can be **1x.extra-small**, **1x.small**, or **1x.medium**.

To provide these expressions, you must create an **AlertingRule** custom resource (CR) containing Prometheus-compatible alerting rules, or a **RecordingRule** CR containing Prometheus-compatible recording rules.

Administrators can configure log-based alerts or recorded metrics for **application**, **audit**, or **infrastructure** tenants. Users without administrator permissions can configure log-based alerts or recorded metrics for **application** tenants of the applications that they have access to.

Application, audit, and infrastructure alerts are sent by default to the Red Hat OpenShift Service on AWS monitoring stack Alertmanager in the **openshift-monitoring** namespace, unless you have disabled the local Alertmanager instance. If the Alertmanager that is used to monitor user-defined projects in the **openshift-user-workload-monitoring** namespace is enabled, application alerts are sent to the Alertmanager in this namespace by default.

13.2.1. Configuring the ruler

When the LokiStack ruler component is enabled, users can define a group of LogQL expressions that trigger logging alerts or recorded metrics.

Administrators can enable the ruler by modifying the **LokiStack** custom resource (CR).

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator and the Loki Operator.
- You have created a **LokiStack** CR.
- You have administrator permissions.

Procedure

- Enable the ruler by ensuring that the **LokiStack** CR contains the following spec configuration:

```
apiVersion: loki.grafana.com/v1
kind: LokiStack
metadata:
  name: <name>
  namespace: <namespace>
spec:
# ...
  rules:
    enabled: true (1)
    selector:
      matchLabels:
        openshift.io/<label_name>: "true" (2)
```

```
namespaceSelector:
  matchLabels:
    openshift.io/<label_name>: "true" (3)
```

1. Enable Loki alerting and recording rules in your cluster.
2. Add a custom label that can be added to namespaces where you want to enable the use of logging alerts and metrics.
3. Add a custom label that can be added to namespaces where you want to enable the use of logging alerts and metrics.

13.2.2. 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.

In logging 5.8 and later, 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.

13.2.2.1. 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>
```

13.2.3. 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 above applies, an alerting rule is considered valid.

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

Prerequisites

- Red Hat OpenShift Logging Operator 5.7 and later
- Red Hat OpenShift Service on AWS 4.13 and later

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
```

13.2.4. Additional resources

- About Red Hat OpenShift Service on AWS monitoring

CHAPTER 14. PERFORMANCE AND RELIABILITY TUNING

14.1. FLOW CONTROL MECHANISMS

If logs are produced faster than they can be collected, it can be difficult to predict or control the volume of logs being sent to an output. Not being able to predict or control the volume of logs being sent to an output can result in logs being lost. If there is a system outage and log buffers are accumulated without user control, this can also cause long recovery times and high latency when the connection is restored.

As an administrator, you can limit logging rates by configuring flow control mechanisms for your logging.

14.1.1. Benefits of flow control mechanisms

- The cost and volume of logging can be predicted more accurately in advance.
- Noisy containers cannot produce unbounded log traffic that drowns out other containers.
- Ignoring low-value logs reduces the load on the logging infrastructure.
- High-value logs can be preferred over low-value logs by assigning higher rate limits.

14.1.2. Configuring rate limits

Rate limits are configured per collector, which means that the maximum rate of log collection is the number of collector instances multiplied by the rate limit.

Because logs are collected from each node's file system, a collector is deployed on each cluster node. For example, in a 3-node cluster, with a maximum rate limit of 10 records per second per collector, the maximum rate of log collection is 30 records per second.

Because the exact byte size of a record as written to an output can vary due to transformations, different encodings, or other factors, rate limits are set in number of records instead of bytes.

You can configure rate limits in the **ClusterLogForwarder** custom resource (CR) in two ways:

Output rate limit

Limit the rate of outbound logs to selected outputs, for example, to match the network or storage capacity of an output. The output rate limit controls the aggregated per-output rate.

Input rate limit

Limit the per-container rate of log collection for selected containers.

14.1.3. Configuring log forwarder output rate limits

You can limit the rate of outbound logs to a specified output by configuring the **ClusterLogForwarder** custom resource (CR).

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.

Procedure

1. Add a **maxRecordsPerSecond** limit value to the **ClusterLogForwarder** CR for a specified output.
   The following example shows how to configure a per collector output rate limit for a Kafka broker output named **kafka-example**:

   Example ClusterLogForwarder CR

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
# ...
  outputs:
    - name: kafka-example (1)
      type: kafka (2)
      limit:
        maxRecordsPerSecond: 1000000 (3)
# ...
```

   (1) The output name.

   (2) The type of output.

   (3) The log output rate limit. This value sets the maximum Quantity of logs that can be sent to the Kafka broker per second. This value is not set by default. The default behavior is best effort, and records are dropped if the log forwarder cannot keep up. If this value is **0**, no logs are forwarded.

2. Apply the **ClusterLogForwarder** CR:

   Example command

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

Additional resources

- Log output types

14.1.4. Configuring log forwarder input rate limits

You can limit the rate of incoming logs that are collected by configuring the **ClusterLogForwarder** custom resource (CR). You can set input limits on a per-container or per-namespace basis.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.

Procedure

1. Add a **maxRecordsPerSecond** limit value to the **ClusterLogForwarder** CR for a specified input.
   The following examples show how to configure input rate limits for different scenarios:

Example ClusterLogForwarder CR that sets a per-container limit for containers with certain labels

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
# ...
  inputs:
    - name: <input_name> (1)
      application:
        selector:
          matchLabels: { example: label } (2)
      containerLimit:
          maxRecordsPerSecond: 0 (3)
# ...
```

(1) The input name.

(2) A list of labels. If these labels match labels that are applied to a pod, the per-container limit specified in the **maxRecordsPerSecond** field is applied to those containers.

(3) Configures the rate limit. Setting the **maxRecordsPerSecond** field to **0** means that no logs are collected for the container. Setting the **maxRecordsPerSecond** field to some other value means that a maximum of that number of records per second are collected for the container.

Example ClusterLogForwarder CR that sets a per-container limit for containers in selected namespaces

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
# ...
  inputs:
    - name: <input_name> (1)
      application:
        namespaces: [ example-ns-1, example-ns-2 ] (2)
      containerLimit:
        maxRecordsPerSecond: 10 (3)
    - name: <input_name>
      application:
        namespaces: [ test ]
      containerLimit:
          maxRecordsPerSecond: 1000
# ...
```

1. The input name.
2. A list of namespaces. The per-container limit specified in the **maxRecordsPerSecond** field is applied to all containers in the namespaces listed.
3. Configures the rate limit. Setting the **maxRecordsPerSecond** field to **10** means that a maximum of 10 records per second are collected for each container in the namespaces listed.

2. Apply the **ClusterLogForwarder** CR:

Example command

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

14.2. FILTERING LOGS BY CONTENT

Collecting all logs from a cluster might produce a large amount of data, which can be expensive to transport and store.

You can reduce the volume of your log data by filtering out low priority data that does not need to be stored. Logging provides content filters that you can use to reduce the volume of log data.

NOTE

Content filters are distinct from **input** selectors. **input** selectors select or ignore entire log streams based on source metadata. Content filters edit log streams to remove and modify records based on the record content.

Log data volume can be reduced by using one of the following methods:

- Configuring content filters to drop unwanted log records
- Configuring content filters to prune log records

14.2.1. 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.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.
- You have created a **ClusterLogForwarder** custom resource (CR).

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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  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"
# ...
```

14.2.2. 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.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.
- You have created a **ClusterLogForwarder** custom resource (CR).

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: logging.openshift.io/v1
kind: ClusterLogForwarder
metadata:
# ...
spec:
  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.

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

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

14.2.3. Additional resources

- About forwarding logs to third-party systems

14.3. FILTERING LOGS BY METADATA

You can filter logs in the **ClusterLogForwarder** CR to select or ignore an entire log stream based on the metadata by using the **input** selector. As an administrator or developer, you can include or exclude the log collection to reduce the memory and CPU load on the collector.

IMPORTANT

You can use this feature only if the Vector collector is set up in your logging deployment.

NOTE

input spec filtering is different from content filtering. **input** selectors select or ignore entire log streams based on the source metadata. Content filters edit the log streams to remove and modify the records based on the record content.

14.3.1. 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.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.
- You have created a **ClusterLogForwarder** custom resource (CR).

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: "logging.openshift.io/v1"
kind: ClusterLogForwarder
# ...
spec:
  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.

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

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

The **excludes** option takes precedence over **includes**.

14.3.2. Filtering application logs at input ny 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.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.
- You have created a **ClusterLogForwarder** custom resource (CR).

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: "logging.openshift.io/v1"
kind: ClusterLogForwarder
# ...
spec:
  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
# ...
```

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
```

14.3.3. 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.

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator.
- You have administrator permissions.
- You have created a **ClusterLogForwarder** custom resource (CR).

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 **aduit** and **infrastructure** sources:

   Example ClusterLogForwarder CR

```
apiVersion: "logging.openshift.io/v1"
kind: ClusterLogForwarder
# ...
spec:
  inputs:
    - name: mylogs1
```

```
      infrastructure:
        sources: (1)
          - node
    - name: mylogs2
      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
```

CHAPTER 15. SCHEDULING RESOURCES

15.1. USING NODE SELECTORS TO MOVE LOGGING RESOURCES

A *node selector* specifies a map of key/value pairs that are defined using custom labels on nodes and selectors specified in pods.

For the pod to be eligible to run on a node, the pod must have the same key/value node selector as the label on the node.

15.1.1. About node selectors

You can use node selectors on pods and labels on nodes to control where the pod is scheduled. With node selectors, Red Hat OpenShift Service on AWS schedules the pods on nodes that contain matching labels.

You can use a node selector to place specific pods on specific nodes, cluster-wide node selectors to place new pods on specific nodes anywhere in the cluster, and project node selectors to place new pods in a project on specific nodes.

For example, as a cluster administrator, you can create an infrastructure where application developers can deploy pods only onto the nodes closest to their geographical location by including a node selector in every pod they create. In this example, the cluster consists of five data centers spread across two regions. In the U.S., label the nodes as **us-east**, **us-central**, or **us-west**. In the Asia-Pacific region (APAC), label the nodes as **apac-east** or **apac-west**. The developers can add a node selector to the pods they create to ensure the pods get scheduled on those nodes.

A pod is not scheduled if the **Pod** object contains a node selector, but no node has a matching label.

IMPORTANT

If you are using node selectors and node affinity in the same pod configuration, the following rules control pod placement onto nodes:

- If you configure both **nodeSelector** and **nodeAffinity**, both conditions must be satisfied for the pod to be scheduled onto a candidate node.
- If you specify multiple **nodeSelectorTerms** associated with **nodeAffinity** types, then the pod can be scheduled onto a node if one of the **nodeSelectorTerms** is satisfied.
- If you specify multiple **matchExpressions** associated with **nodeSelectorTerms**, then the pod can be scheduled onto a node only if all **matchExpressions** are satisfied.

Node selectors on specific pods and nodes

You can control which node a specific pod is scheduled on by using node selectors and labels.
To use node selectors and labels, first label the node to avoid pods being descheduled, then add the node selector to the pod.

NOTE

You cannot add a node selector directly to an existing scheduled pod. You must label the object that controls the pod, such as deployment config.

For example, the following **Node** object has the **region: east** label:

Sample Node object with a label

```
kind: Node
apiVersion: v1
metadata:
  name: ip-10-0-131-14.ec2.internal
  selfLink: /api/v1/nodes/ip-10-0-131-14.ec2.internal
  uid: 7bc2580a-8b8e-11e9-8e01-021ab4174c74
  resourceVersion: '478704'
  creationTimestamp: '2019-06-10T14:46:08Z'
  labels:
    kubernetes.io/os: linux
    topology.kubernetes.io/zone: us-east-1a
    node.openshift.io/os_version: '4.5'
    node-role.kubernetes.io/worker: ''
    topology.kubernetes.io/region: us-east-1
    node.openshift.io/os_id: rhcos
    node.kubernetes.io/instance-type: m4.large
    kubernetes.io/hostname: ip-10-0-131-14
    kubernetes.io/arch: amd64
    region: east (1)
    type: user-node
#...
```

1. Labels to match the pod node selector.

A pod has the **type: user-node,region: east** node selector:

Sample Pod object with node selectors

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector: (1)
    region: east
    type: user-node
#...
```

1. Node selectors to match the node label. The node must have a label for each node selector.

When you create the pod using the example pod spec, it can be scheduled on the example node.

Default cluster-wide node selectors

With default cluster-wide node selectors, when you create a pod in that cluster, Red Hat OpenShift Service on AWS adds the default node selectors to the pod and schedules the pod on nodes with matching labels.

For example, the following **Scheduler** object has the default cluster-wide **region=east** and **type=user-node** node selectors:

Example Scheduler Operator Custom Resource

```
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  name: cluster
#...
spec:
  defaultNodeSelector: type=user-node,region=east
#...
```

A node in that cluster has the **type=user-node,region=east** labels:

Example **Node** object

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
  labels:
    region: east
    type: user-node
#...
```

Example **Pod** object with a node selector

```
apiVersion: v1
kind: Pod
metadata:
  name: s1
#...
spec:
  nodeSelector:
    region: east
#...
```

When you create the pod using the example pod spec in the example cluster, the pod is created with the cluster-wide node selector and is scheduled on the labeled node:

Example pod list with the pod on the labeled node

```
NAME     READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
pod-s1   1/1     Running   0          20s   10.131.2.6   ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>           <none>
```

NOTE

If the project where you create the pod has a project node selector, that selector takes preference over a cluster-wide node selector. Your pod is not created or scheduled if the pod does not have the project node selector.

Project node selectors

With project node selectors, when you create a pod in this project, Red Hat OpenShift Service on AWS adds the node selectors to the pod and schedules the pods on a node with matching labels. If there is a cluster-wide default node selector, a project node selector takes preference.
For example, the following project has the **region=east** node selector:

Example Namespace object

```
apiVersion: v1
kind: Namespace
metadata:
  name: east-region
  annotations:
    openshift.io/node-selector: "region=east"
#...
```

The following node has the **type=user-node,region=east** labels:

Example Node object

```
apiVersion: v1
kind: Node
metadata:
  name: ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
#...
  labels:
    region: east
    type: user-node
#...
```

When you create the pod using the example pod spec in this example project, the pod is created with the project node selectors and is scheduled on the labeled node:

Example Pod object

```
apiVersion: v1
kind: Pod
metadata:
  namespace: east-region
#...
spec:
  nodeSelector:
    region: east
    type: user-node
#...
```

Example pod list with the pod on the labeled node

```
NAME     READY   STATUS    RESTARTS   AGE   IP          NODE
NOMINATED NODE   READINESS GATES
pod-s1   1/1     Running   0          20s   10.131.2.6   ci-ln-qg1il3k-f76d1-hlmhl-worker-b-df2s4
<none>           <none>
```

A pod in the project is not created or scheduled if the pod contains different node selectors. For example, if you deploy the following pod into the example project, it is not be created:

Example Pod object with an invalid node selector

```
apiVersion: v1
kind: Pod
metadata:
  name: west-region
#...
spec:
  nodeSelector:
    region: west
#...
```

15.1.2. 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.

In the previous example configuration, all Loki pods are moved to nodes containing the **node-role.kubernetes.io/infra: ""** 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.
...
```

15.1.3. Configuring resources and scheduling for logging collectors

Administrators can modify the resources or scheduling of the collector by creating a **ClusterLogging** custom resource (CR) that is in the same namespace and has the same name as the **ClusterLogForwarder** CR that it supports.

The applicable stanzas for the **ClusterLogging** CR when using multiple log forwarders in a deployment are **managementState** and **collection**. All other stanzas are ignored.

Prerequisites

- You have administrator permissions.
- You have installed the Red Hat OpenShift Logging Operator version 5.8 or newer.
- You have created a **ClusterLogForwarder** CR.

Procedure

1. Create a **ClusterLogging** CR that supports your existing **ClusterLogForwarder** CR:

Example ClusterLogging CR YAML

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: <name> (1)
  namespace: <namespace> (2)
spec:
  managementState: "Managed"
  collection:
    type: "vector"
    tolerations:
    - key: "logging"
      operator: "Exists"
      effect: "NoExecute"
      tolerationSeconds: 6000
    resources:
      limits:
        memory: 1Gi
      requests:
        cpu: 100m
        memory: 1Gi
    nodeSelector:
      collector: needed
# ...
```

(1) The name must be the same name as the **ClusterLogForwarder** CR.

(2) The namespace must be the same namespace as the **ClusterLogForwarder** CR.

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

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

15.1.4. Viewing logging collector pods

You can view the logging collector pods and the corresponding nodes that they are running on.

Procedure

- Run the following command in a project to view the logging collector pods and their details:

```
$ oc get pods --selector component=collector -o wide -n <project_name>
```

Example output

```
NAME          READY  STATUS    RESTARTS   AGE     IP           NODE
NOMINATED NODE   READINESS GATES
collector-8d69v  1/1    Running   0          134m    10.130.2.30   master1.example.com
<none>           <none>
collector-bd225  1/1    Running   0          134m    10.131.1.11   master2.example.com
<none>           <none>
collector-cvrzs  1/1    Running   0          134m    10.130.0.21   master3.example.com   <none>
<none>
collector-gpqg2  1/1    Running   0          134m    10.128.2.27   worker1.example.com
<none>           <none>
collector-l9j7j  1/1    Running   0          134m    10.129.2.31   worker2.example.com   <none>
<none>
```

15.1.5. Additional resources

- Placing pods on specific nodes using node selectors

15.2. USING TAINTS AND TOLERATIONS TO CONTROL LOGGING POD PLACEMENT

Taints and tolerations allow the node to control which pods should (or should not) be scheduled on them.

15.2.1. Understanding taints and tolerations

A *taint* allows a node to refuse a pod to be scheduled unless that pod has a matching *toleration*.

You apply taints to a node through the **Node** specification (**NodeSpec**) and apply tolerations to a pod through the **Pod** specification (**PodSpec**). When you apply a taint a node, the scheduler cannot place a pod on that node unless the pod can tolerate the taint.

Example taint in a node specification

```
apiVersion: v1
kind: Node
metadata:
```

```
  name: my-node
#...
spec:
  taints:
  - effect: NoExecute
    key: key1
    value: value1
#...
```

Example toleration in a **Pod** spec

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
#...
spec:
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoExecute"
    tolerationSeconds: 3600
#...
```

Taints and tolerations consist of a key, value, and effect.

Table 15.1. Taint and toleration components

Parameter	Description
key	The **key** is any string, up to 253 characters. The key must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.
value	The **value** is any string, up to 63 characters. The value must begin with a letter or number, and may contain letters, numbers, hyphens, dots, and underscores.

Parameter	Description
effect	The effect is one of the following: **NoSchedule** [1] — • New pods that do not match the taint are not scheduled onto that node. • Existing pods on the node remain. **PreferNoSchedule** — • New pods that do not match the taint might be scheduled onto that node, but the scheduler tries not to. • Existing pods on the node remain. **NoExecute** — • New pods that do not match the taint cannot be scheduled onto that node. • Existing pods on the node that do not have a matching toleration are removed.
operator	**Equal** — The **key**/**value**/**effect** parameters must match. This is the default. **Exists** — The **key**/**effect** parameters must match. You must leave a blank **value** parameter, which matches any.

1. If you add a **NoSchedule** taint to a control plane node, the node must have the **node-role.kubernetes.io/master=:NoSchedule** taint, which is added by default.
   For example:

```
apiVersion: v1
kind: Node
metadata:
  annotations:
    machine.openshift.io/machine: openshift-machine-api/ci-ln-62s7gtb-f76d1-v8jxv-master-0
    machineconfiguration.openshift.io/currentConfig: rendered-master-
cdc1ab7da414629332cc4c3926e6e59c
  name: my-node
#...
spec:
```

```
  taints:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
#...
```

A toleration matches a taint:

- If the **operator** parameter is set to **Equal**:
  - the **key** parameters are the same;
  - the **value** parameters are the same;
  - the **effect** parameters are the same.
- If the **operator** parameter is set to **Exists**:
  - the **key** parameters are the same;
  - the **effect** parameters are the same.

The following taints are built into Red Hat OpenShift Service on AWS:

- **node.kubernetes.io/not-ready**: The node is not ready. This corresponds to the node condition **Ready=False**.
- **node.kubernetes.io/unreachable**: The node is unreachable from the node controller. This corresponds to the node condition **Ready=Unknown**.
- **node.kubernetes.io/memory-pressure**: The node has memory pressure issues. This corresponds to the node condition **MemoryPressure=True**.
- **node.kubernetes.io/disk-pressure**: The node has disk pressure issues. This corresponds to the node condition **DiskPressure=True**.
- **node.kubernetes.io/network-unavailable**: The node network is unavailable.
- **node.kubernetes.io/unschedulable**: The node is unschedulable.
- **node.cloudprovider.kubernetes.io/uninitialized**: When the node controller is started with an external cloud provider, this taint is set on a node to mark it as unusable. After a controller from the cloud-controller-manager initializes this node, the kubelet removes this taint.
- **node.kubernetes.io/pid-pressure**: The node has pid pressure. This corresponds to the node condition **PIDPressure=True**.

IMPORTANT

Red Hat OpenShift Service on AWS does not set a default pid.available **evictionHard**.

15.2.2. 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.

In the previous example configuration, all Loki pods are moved to nodes containing the **node-role.kubernetes.io/infra: ""** 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.
...
```

15.2.3. Using tolerations to control log collector pod placement

By default, log collector pods have the following **tolerations** configuration:

```
apiVersion: v1
kind: Pod
metadata:
  name: collector-example
  namespace: openshift-logging
spec:
# ...
  collection:
    type: vector
    tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
      operator: Exists
    - effect: NoSchedule
      key: node.kubernetes.io/disk-pressure
      operator: Exists
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
    - effect: NoSchedule
```

```
      key: node.kubernetes.io/memory-pressure
      operator: Exists
    - effect: NoSchedule
      key: node.kubernetes.io/pid-pressure
      operator: Exists
    - effect: NoSchedule
      key: node.kubernetes.io/unschedulable
      operator: Exists
# ...
```

Prerequisites

- You have installed the Red Hat OpenShift Logging Operator and OpenShift CLI (**oc**).

Procedure

1. Add a taint to a node where you want logging collector pods to schedule logging collector pods by running the following command:

```
$ oc adm taint nodes <node_name> <key>=<value>:<effect>
```

Example command

```
$ oc adm taint nodes node1 collector=node:NoExecute
```

This example places a taint on **node1** that has key **collector**, value **node**, and taint effect **NoExecute**. You must use the **NoExecute** taint effect. **NoExecute** schedules only pods that match the taint and removes existing pods that do not match.

2. Edit the **collection** stanza of the **ClusterLogging** custom resource (CR) to configure a toleration for the logging collector pods:

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
# ...
spec:
# ...
  collection:
    type: vector
    tolerations:
    - key: collector (1)
      operator: Exists (2)
      effect: NoExecute (3)
      tolerationSeconds: 6000 (4)
    resources:
      limits:
        memory: 2Gi
      requests:
        cpu: 100m
        memory: 1Gi
# ...
```

1. Specify the key that you added to the node.
2. Specify the **Exists** operator to require the **key**/**value**/**effect** parameters to match.
3. Specify the **NoExecute** effect.
4. Optionally, specify the **tolerationSeconds** parameter to set how long a pod can remain bound to a node before being evicted.

This toleration matches the taint created by the **oc adm taint** command. A pod with this toleration can be scheduled onto **node1**.

15.2.4. Configuring resources and scheduling for logging collectors

Administrators can modify the resources or scheduling of the collector by creating a **ClusterLogging** custom resource (CR) that is in the same namespace and has the same name as the **ClusterLogForwarder** CR that it supports.

The applicable stanzas for the **ClusterLogging** CR when using multiple log forwarders in a deployment are **managementState** and **collection**. All other stanzas are ignored.

Prerequisites

- You have administrator permissions.
- You have installed the Red Hat OpenShift Logging Operator version 5.8 or newer.
- You have created a **ClusterLogForwarder** CR.

Procedure

1. Create a **ClusterLogging** CR that supports your existing **ClusterLogForwarder** CR:

Example ClusterLogging CR YAML

```
apiVersion: logging.openshift.io/v1
kind: ClusterLogging
metadata:
  name: <name> (1)
  namespace: <namespace> (2)
spec:
  managementState: "Managed"
  collection:
    type: "vector"
    tolerations:
    - key: "logging"
      operator: "Exists"
      effect: "NoExecute"
      tolerationSeconds: 6000
    resources:
      limits:
        memory: 1Gi
      requests:
        cpu: 100m
```

```
        memory: 1Gi
    nodeSelector:
      collector: needed
# ...
```

1. The name must be the same name as the **ClusterLogForwarder** CR.
2. The namespace must be the same namespace as the **ClusterLogForwarder** CR.

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

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

15.2.5. Viewing logging collector pods

You can view the logging collector pods and the corresponding nodes that they are running on.

Procedure

- Run the following command in a project to view the logging collector pods and their details:

```
$ oc get pods --selector component=collector -o wide -n <project_name>
```

Example output

```
NAME          READY  STATUS    RESTARTS   AGE     IP           NODE
NOMINATED NODE   READINESS GATES
collector-8d69v  1/1    Running   0          134m    10.130.2.30   master1.example.com
<none>           <none>
collector-bd225  1/1    Running   0          134m    10.131.1.11   master2.example.com
<none>           <none>
collector-cvrzs  1/1    Running   0          134m    10.130.0.21   master3.example.com   <none>
<none>
collector-gpqg2  1/1    Running   0          134m    10.128.2.27   worker1.example.com
<none>           <none>
collector-l9j7j  1/1    Running   0          134m    10.129.2.31   worker2.example.com   <none>
<none>
```

15.2.6. Additional resources

- Controlling pod placement using node taints

CHAPTER 16. UNINSTALLING LOGGING

You can remove logging from your Red Hat OpenShift Service on AWS cluster by removing installed Operators and related custom resources (CRs).

16.1. UNINSTALLING THE LOGGING

You can stop aggregating logs by deleting the Red Hat OpenShift Logging Operator and the **ClusterLogging** custom resource (CR).

Prerequisites

- You have administrator permissions.
- You have access to the **Administrator** perspective of the Red Hat OpenShift Service on AWS web console.

Procedure

1. Go to the **Administration** → **Custom Resource Definitions** page, and click **ClusterLogging**.
2. On the **Custom Resource Definition Details** page, click **Instances**.
3. Click the options menu ⋮ next to the instance, and click **Delete ClusterLogging**.
4. Go to the **Administration** → **Custom Resource Definitions** page.
5. Click the options menu ⋮ next to **ClusterLogging**, and select **Delete Custom Resource Definition**.

> **WARNING**
>
> Deleting the **ClusterLogging** CR does not remove the persistent volume claims (PVCs). To delete the remaining PVCs, persistent volumes (PVs), and associated data, you must take further action. Releasing or deleting PVCs can delete PVs and cause data loss.

6. If you have created a **ClusterLogForwarder** CR, click the options menu ⋮ next to **ClusterLogForwarder**, and then click **Delete Custom Resource Definition**
7. Go to the **Operators** → **Installed Operators** page.

8. Click the options menu ⋮ next to the Red Hat OpenShift Logging Operator, and then click **Uninstall Operator**.

9. Optional: Delete the **openshift-logging** project.

> **WARNING**
>
> Deleting the **openshift-logging** project deletes everything in that namespace, including any persistent volume claims (PVCs). If you want to preserve logging data, do not delete the **openshift-logging** project.

   a. Go to the **Home → Projects** page.

   b. Click the options menu ⋮ next to the **openshift-logging** project, and then click **Delete Project**.

   c. Confirm the deletion by typing **openshift-logging** in the dialog box, and then click **Delete**.

16.2. DELETING LOGGING PVCS

To keep persistent volume claims (PVCs) for reuse with other pods, keep the labels or PVC names that you need to reclaim the PVCs. If you do not want to keep the PVCs, you can delete them. If you want to recover storage space, you can also delete the persistent volumes (PVs).

Prerequisites

- You have administrator permissions.
- You have access to the **Administrator** perspective of the Red Hat OpenShift Service on AWS web console.

Procedure

1. Go to the **Storage → Persistent Volume Claims**page.

2. Click the options menu ⋮ next to each PVC, and select **Delete Persistent Volume Claim**

16.3. UNINSTALLING LOKI

Prerequisites

- You have administrator permissions.

- You have access to the **Administrator** perspective of the Red Hat OpenShift Service on AWS web console.
- If you have not already removed the Red Hat OpenShift Logging Operator and related resources, you have removed references to LokiStack from the **ClusterLogging** custom resource.

Procedure

1. Go to the **Administration** → **Custom Resource Definitions** page, and click **LokiStack**.
2. On the **Custom Resource Definition Details** page, click **Instances**.
3. Click the options menu ⋮ next to the instance, and then click **Delete LokiStack**.
4. Go to the **Administration** → **Custom Resource Definitions** page.
5. Click the options menu ⋮ next to **LokiStack**, and select **Delete Custom Resource Definition**.
6. Delete the object storage secret.
7. Go to the **Operators** → **Installed Operators** page.
8. Click the options menu ⋮ next to the Loki Operator, and then click **Uninstall Operator**.
9. Optional: Delete the **openshift-operators-redhat** project.

   IMPORTANT

   Do not delete the **openshift-operators-redhat** project if other global Operators are installed in this namespace.

   a. Go to the **Home** → **Projects** page.
   b. Click the options menu ⋮ next to the **openshift-operators-redhat** project, and then click **Delete Project**.
   c. Confirm the deletion by typing **openshift-operators-redhat** in the dialog box, and then click **Delete**.

16.4. UNINSTALLING ELASTICSEARCH

Prerequisites

- You have administrator permissions.

- You have access to the **Administrator** perspective of the Red Hat OpenShift Service on AWS web console.
- If you have not already removed the Red Hat OpenShift Logging Operator and related resources, you must remove references to Elasticsearch from the **ClusterLogging** custom resource.

Procedure

1. Go to the **Administration** → **Custom Resource Definitions** page, and click **Elasticsearch**.
2. On the **Custom Resource Definition Details** page, click **Instances**.
3. Click the options menu ⋮ next to the instance, and then click **Delete Elasticsearch**.
4. Go to the **Administration** → **Custom Resource Definitions** page.
5. Click the options menu ⋮ next to **Elasticsearch**, and select **Delete Custom Resource Definition**.
6. Delete the object storage secret.
7. Go to the **Operators** → **Installed Operators** page.
8. Click the options menu ⋮ next to the OpenShift Elasticsearch Operator, and then click **Uninstall Operator**.
9. Optional: Delete the **openshift-operators-redhat** project.

   IMPORTANT

   Do not delete the **openshift-operators-redhat** project if other global Operators are installed in this namespace.

   a. Go to the **Home** → **Projects** page.
   b. Click the options menu ⋮ next to the **openshift-operators-redhat** project, and then click **Delete Project**.
   c. Confirm the deletion by typing **openshift-operators-redhat** in the dialog box, and then click **Delete**.

16.5. DELETING OPERATORS FROM A CLUSTER USING THE CLI

Cluster administrators can delete installed Operators from a selected namespace by using the CLI.

Prerequisites

- You have access to an Red Hat OpenShift Service on AWS cluster using an account with **dedicated-admin** permissions.
- The OpenShift CLI (**oc**) is installed on your workstation.

Procedure

1. Ensure the latest version of the subscribed operator (for example, **serverless-operator**) is identified in the **currentCSV** field.

```
$ oc get subscription.operators.coreos.com serverless-operator -n openshift-serverless -o yaml | grep currentCSV
```

Example output

```
  currentCSV: serverless-operator.v1.28.0
```

2. Delete the subscription (for example, **serverless-operator**):

```
$ oc delete subscription.operators.coreos.com serverless-operator -n openshift-serverless
```

Example output

```
subscription.operators.coreos.com "serverless-operator" deleted
```

3. Delete the CSV for the Operator in the target namespace using the **currentCSV** value from the previous step:

```
$ oc delete clusterserviceversion serverless-operator.v1.28.0 -n openshift-serverless
```

Example output

```
clusterserviceversion.operators.coreos.com "serverless-operator.v1.28.0" deleted
```

Additional resources

- Reclaiming a persistent volume manually

CHAPTER 17. LOG RECORD FIELDS

The following fields can be present in log records exported by the logging. Although log records are typically formatted as JSON objects, the same data model can be applied to other encodings.

To search these fields from Elasticsearch and Kibana, use the full dotted field name when searching. For example, with an Elasticsearch **/_search URL**, to look for a Kubernetes pod name, use **/_search/q=kubernetes.pod_name:name-of-my-pod**.

The top level fields may be present in every record.

MESSAGE

The original log entry text, UTF-8 encoded. This field may be absent or empty if a non-empty **structured** field is present. See the description of **structured** for more.

Data type	text
Example value	**HAPPY**

STRUCTURED

Original log entry as a structured object. This field may be present if the forwarder was configured to parse structured JSON logs. If the original log entry was a valid structured log, this field will contain an equivalent JSON structure. Otherwise this field will be empty or absent, and the **message** field will contain the original log message. The **structured** field can have any subfields that are included in the log message, there are no restrictions defined here.

Data type	group
Example value	map[message:starting fluentd worker pid=21631 ppid=21618 worker=0 pid:21631 ppid:21618 worker:0]

@TIMESTAMP

A UTC value that marks when the log payload was created or, if the creation time is not known, when the log payload was first collected. The “@” prefix denotes a field that is reserved for a particular use. By default, most tools look for “@timestamp” with ElasticSearch.

Data type	date
Example value	**2015-01-24 14:06:05.071000000 Z**

HOSTNAME

The name of the host where this log message originated. In a Kubernetes cluster, this is the same as **kubernetes.host**.

Data type	keyword

IPADDR4

The IPv4 address of the source server. Can be an array.

Data type	ip

IPADDR6

The IPv6 address of the source server, if available. Can be an array.

Data type	ip

LEVEL

The logging level from various sources, including **rsyslog(severitytext property)**, a Python logging module, and others.

The following values come from **syslog.h**, and are preceded by their numeric equivalents:

- **0** = **emerg**, system is unusable.
- **1** = **alert**, action must be taken immediately.
- **2** = **crit**, critical conditions.
- **3** = **err**, error conditions.
- **4** = **warn**, warning conditions.
- **5** = **notice**, normal but significant condition.
- **6** = **info**, informational.
- **7** = **debug**, debug-level messages.

The two following values are not part of **syslog.h** but are widely used:

- **8** = **trace**, trace-level messages, which are more verbose than **debug** messages.
- **9** = **unknown**, when the logging system gets a value it doesn't recognize.

Map the log levels or priorities of other logging systems to their nearest match in the preceding list. For example, from python logging, you can match **CRITICAL** with **crit**, **ERROR** with **err**, and so on.

Data type	keyword
Example value	**info**

PID

The process ID of the logging entity, if available.

Data type	keyword

SERVICE

The name of the service associated with the logging entity, if available. For example, syslog's **APP-NAME** and rsyslog's **programname** properties are mapped to the service field.

Data type	keyword

CHAPTER 18. TAGS

Optional. An operator-defined list of tags placed on each log by the collector or normalizer. The payload can be a string with whitespace-delimited string tokens or a JSON list of string tokens.

Data type	text

FILE

The path to the log file from which the collector reads this log entry. Normally, this is a path in the **/var/log** file system of a cluster node.

Data type	text

OFFSET

The offset value. Can represent bytes to the start of the log line in the file (zero- or one-based), or log line numbers (zero- or one-based), so long as the values are strictly monotonically increasing in the context of a single log file. The values are allowed to wrap, representing a new version of the log file (rotation).

Data type	long

CHAPTER 19. KUBERNETES

The namespace for Kubernetes-specific metadata

Data type	group

19.1. KUBERNETES.POD_NAME

The name of the pod

Data type	keyword

19.2. KUBERNETES.POD_ID

The Kubernetes ID of the pod

Data type	keyword

19.3. KUBERNETES.NAMESPACE_NAME

The name of the namespace in Kubernetes

Data type	keyword

19.4. KUBERNETES.NAMESPACE_ID

The ID of the namespace in Kubernetes

Data type	keyword

19.5. KUBERNETES.HOST

The Kubernetes node name

Data type	keyword

19.6. KUBERNETES.CONTAINER_NAME

The name of the container in Kubernetes

Data type	keyword

19.7. KUBERNETES.ANNOTATIONS

Annotations associated with the Kubernetes object

Data type	group

19.8. KUBERNETES.LABELS

Labels present on the original Kubernetes Pod

Data type	group

19.9. KUBERNETES.EVENT

The Kubernetes event obtained from the Kubernetes master API. This event description loosely follows **type Event** in Event v1 core.

Data type	group

19.9.1. kubernetes.event.verb

The type of event, **ADDED**, **MODIFIED**, or **DELETED**

Data type	keyword
Example value	**ADDED**

19.9.2. kubernetes.event.metadata

Information related to the location and time of the event creation

Data type	group

19.9.2.1. kubernetes.event.metadata.name

The name of the object that triggered the event creation

Data type	keyword
Example value	**java-mainclass-1.14d888a4cfc24890**

19.9.2.2. kubernetes.event.metadata.namespace

The name of the namespace where the event originally occurred. Note that it differs from **kubernetes.namespace_name**, which is the namespace where the **eventrouter** application is deployed.

Data type	keyword
Example value	**default**

19.9.2.3. kubernetes.event.metadata.selfLink

A link to the event

Data type	keyword
Example value	**/api/v1/namespaces/javaj/events/java-mainclass-1.14d888a4cfc24890**

19.9.2.4. kubernetes.event.metadata.uid

The unique ID of the event

Data type	keyword
Example value	**d828ac69-7b58-11e7-9cf5-5254002f560c**

19.9.2.5. kubernetes.event.metadata.resourceVersion

A string that identifies the server's internal version of the event. Clients can use this string to determine when objects have changed.

Data type	integer
Example value	**311987**

19.9.3. kubernetes.event.involvedObject

The object that the event is about.

Data type	group

19.9.3.1. kubernetes.event.involvedObject.kind

The type of object

Data type	keyword
Example value	**ReplicationController**

19.9.3.2. kubernetes.event.involvedObject.namespace

The namespace name of the involved object. Note that it may differ from **kubernetes.namespace_name**, which is the namespace where the **eventrouter** application is deployed.

Data type	keyword
Example value	**default**

19.9.3.3. kubernetes.event.involvedObject.name

The name of the object that triggered the event

Data type	keyword
Example value	**java-mainclass-1**

19.9.3.4. kubernetes.event.involvedObject.uid

The unique ID of the object

Data type	keyword
Example value	**e6bff941-76a8-11e7-8193-5254002f560c**

19.9.3.5. kubernetes.event.involvedObject.apiVersion

The version of kubernetes master API

Data type	keyword
Example value	**v1**

19.9.3.6. kubernetes.event.involvedObject.resourceVersion

A string that identifies the server's internal version of the pod that triggered the event. Clients can use this string to determine when objects have changed.

Data type	keyword
Example value	**308882**

19.9.4. kubernetes.event.reason

A short machine-understandable string that gives the reason for generating this event

Data type	keyword
Example value	**SuccessfulCreate**

19.9.5. kubernetes.event.source_component

The component that reported this event

Data type	keyword
Example value	**replication-controller**

19.9.6. kubernetes.event.firstTimestamp

The time at which the event was first recorded

Data type	date
Example value	**2017-08-07 10:11:57.000000000 Z**

19.9.7. kubernetes.event.count

The number of times this event has occurred

Data type	integer
Example value	**1**

19.9.8. kubernetes.event.type

The type of event, **Normal** or **Warning**. New types could be added in the future.

Data type	keyword
Example value	**Normal**

CHAPTER 20. OPENSHIFT

The namespace for openshift-logging specific metadata

Data type	group

20.1. OPENSHIFT.LABELS

Labels added by the Cluster Log Forwarder configuration

Data type	group

CHAPTER 21. API REFERENCE

21.1. 5.6 LOGGING API REFERENCE

21.1.1. Logging 5.6 API reference

21.1.1.1. ClusterLogForwarder

ClusterLogForwarder is an API to configure forwarding logs.

You configure forwarding by specifying a list of **pipelines**, which forward from a set of named inputs to a set of named outputs.

There are built-in input names for common log categories, and you can define custom inputs to do additional filtering.

There is a built-in output name for the default openshift log store, but you can define your own outputs with a URL and other connection information to forward logs to other stores or processors, inside or outside the cluster.

For more details see the documentation on the API fields.

Property	Type	Description
spec	object	Specification of the desired behavior of ClusterLogForwarder
status	object	Status of the ClusterLogForwarder

21.1.1.1.1. .spec

21.1.1.1.1.1. Description

ClusterLogForwarderSpec defines how logs should be forwarded to remote targets.

21.1.1.1.1.1.1. Type

- object

Property	Type	Description
inputs	array	**(optional)** Inputs are named filters for log messages to be forwarded.

Property	Type	Description
outputDefaults	object	**(optional)** DEPRECATED OutputDefaults specify forwarder config explicitly for the default store.
outputs	array	**(optional)** Outputs are named destinations for log messages.
pipelines	array	Pipelines forward the messages selected by a set of inputs to a set of outputs.

21.1.1.1.2. .spec.inputs[]

21.1.1.1.2.1. Description

InputSpec defines a selector of log messages.

21.1.1.1.2.1.1. Type

- array

Property	Type	Description
application	object	**(optional)** Application, if present, enables named set of **application** logs that
name	string	Name used to refer to the input of a **pipeline**.

21.1.1.1.3. .spec.inputs[].application

21.1.1.1.3.1. Description

Application log selector. All conditions in the selector must be satisfied (logical AND) to select logs.

21.1.1.1.3.1.1. Type

- object

Property	Type	Description
namespaces	array	**(optional)** Namespaces from which to collect application logs.

Property	Type	Description
selector	object	**(optional)** Selector for logs from pods with matching labels.

21.1.1.1.4. .spec.inputs[].application.namespaces[]

21.1.1.1.4.1. Description

21.1.1.1.4.1.1. Type

- array

21.1.1.1.5. .spec.inputs[].application.selector

21.1.1.1.5.1. Description

A label selector is a label query over a set of resources.

21.1.1.1.5.1.1. Type

- object

Property	Type	Description
matchLabels	object	**(optional)** matchLabels is a map of {key,value} pairs. A single {key,value} in the matchLabels

21.1.1.1.6. .spec.inputs[].application.selector.matchLabels

21.1.1.1.6.1. Description

21.1.1.1.6.1.1. Type

- object

21.1.1.1.7. .spec.outputDefaults

21.1.1.1.7.1. Description

21.1.1.1.7.1.1. Type

- object

Property	Type	Description
elasticsearch	object	**(optional)** Elasticsearch OutputSpec default values

21.1.1.1.8. .spec.outputDefaults.elasticsearch

21.1.1.1.8.1. Description

ElasticsearchStructuredSpec is spec related to structured log changes to determine the elasticsearch index

21.1.1.1.8.1.1. Type

- object

Property	Type	Description
enableStructuredContainerLogs	bool	**(optional)** EnableStructuredContainerLogs enables multi-container structured logs to allow
structuredTypeKey	string	**(optional)** StructuredTypeKey specifies the metadata key to be used as name of elasticsearch index
structuredTypeName	string	**(optional)** StructuredTypeName specifies the name of elasticsearch schema

21.1.1.1.9. .spec.outputs[]

21.1.1.1.9.1. Description

Output defines a destination for log messages.

21.1.1.1.9.1.1. Type

- array

Property	Type	Description
syslog	object	**(optional)**
fluentdForward	object	**(optional)**

Property	Type	Description
elasticsearch	object	**(optional)**
kafka	object	**(optional)**
cloudwatch	object	**(optional)**
loki	object	**(optional)**
googleCloudLogging	object	**(optional)**
splunk	object	**(optional)**
name	string	Name used to refer to the output from a **pipeline**.
secret	object	**(optional)** Secret for authentication.
tls	object	TLS contains settings for controlling options on TLS client connections.
type	string	Type of output plugin.
url	string	**(optional)** URL to send log records to.

21.1.1.1.10. .spec.outputs[].secret

21.1.1.1.10.1. Description

OutputSecretSpec is a secret reference containing name only, no namespace.

21.1.1.1.10.1.1. Type

- object

Property	Type	Description
name	string	Name of a secret in the namespace configured for log forwarder secrets.

21.1.1.1.11. .spec.outputs[].tls

21.1.1.1.11.1. Description

OutputTLSSpec contains options for TLS connections that are agnostic to the output type.

21.1.1.1.11.1.1. Type

- object

Property	Type	Description
insecureSkipVerify	bool	If InsecureSkipVerify is true, then the TLS client will be configured to ignore errors with certificates.

21.1.1.1.12. .spec.pipelines[]

21.1.1.1.12.1. Description

PipelinesSpec link a set of inputs to a set of outputs.

21.1.1.1.12.1.1. Type

- array

Property	Type	Description
detectMultilineErrors	bool	**(optional)** DetectMultilineErrors enables multiline error detection of container logs
inputRefs	array	InputRefs lists the names (**input.name**) of inputs to this pipeline.
labels	object	**(optional)** Labels applied to log records passing through this pipeline.
name	string	**(optional)** Name is optional, but must be unique in the **pipelines** list if provided.
outputRefs	array	OutputRefs lists the names (**output.name**) of outputs from this pipeline.
parse	string	**(optional)** Parse enables parsing of log entries into structured logs

21.1.1.1.13. .spec.pipelines[].inputRefs[]

21.1.1.1.13.1. Description

21.1.1.1.13.1.1. Type

- array

21.1.1.1.14. .spec.pipelines[].labels

21.1.1.1.14.1. Description

21.1.1.1.14.1.1. Type

- object

21.1.1.1.15. .spec.pipelines[].outputRefs[]

21.1.1.1.15.1. Description

21.1.1.1.15.1.1. Type

- array

21.1.1.1.16. .status

21.1.1.1.16.1. Description

ClusterLogForwarderStatus defines the observed state of ClusterLogForwarder

21.1.1.1.16.1.1. Type

- object

Property	Type	Description
conditions	object	Conditions of the log forwarder.
inputs	Conditions	Inputs maps input name to condition of the input.
outputs	Conditions	Outputs maps output name to condition of the output.
pipelines	Conditions	Pipelines maps pipeline name to condition of the pipeline.

21.1.1.1.17. .status.conditions

21.1.1.1.17.1. Description

21.1.1.1.17.1.1. Type

- object

21.1.1.1.18. .status.inputs

21.1.1.1.18.1. Description

21.1.1.1.18.1.1. Type

- Conditions

21.1.1.1.19. .status.outputs

21.1.1.1.19.1. Description

21.1.1.1.19.1.1. Type

- Conditions

21.1.1.1.20. .status.pipelines

21.1.1.1.20.1. Description

21.1.1.1.20.1.1. Type

- Conditions== ClusterLogging A Red Hat OpenShift Logging instance. ClusterLogging is the Schema for the clusterloggings API

Property	Type	Description
spec	object	Specification of the desired behavior of ClusterLogging
status	object	Status defines the observed state of ClusterLogging

21.1.1.1.21. .spec

21.1.1.1.21.1. Description

ClusterLoggingSpec defines the desired state of ClusterLogging

21.1.1.1.21.1.1. Type

- object

Property	Type	Description
collection	object	Specification of the Collection component for the cluster
curation	object	**(DEPRECATED) (optional)** Deprecated. Specification of the Curation component for the cluster
forwarder	object	**(DEPRECATED) (optional)** Deprecated. Specification for Forwarder component for the cluster
logStore	object	**(optional)** Specification of the Log Storage component for the cluster
managementState	string	**(optional)** Indicator if the resource is 'Managed' or 'Unmanaged' by the operator
visualization	object	**(optional)** Specification of the Visualization component for the cluster

21.1.1.1.22. .spec.collection

21.1.1.1.22.1. Description

This is the struct that will contain information pertinent to Log and event collection

21.1.1.1.22.1.1. Type

- object

Property	Type	Description
resources	object	**(optional)** The resource requirements for the collector
nodeSelector	object	**(optional)** Define which Nodes the Pods are scheduled on.
tolerations	array	**(optional)** Define the tolerations the Pods will accept

Property	Type	Description
fluentd	object	**(optional)** Fluentd represents the configuration for forwarders of type fluentd.
logs	object	**(DEPRECATED) (optional)** Deprecated. Specification of Log Collection for the cluster
type	string	**(optional)** The type of Log Collection to configure

21.1.1.1.23. .spec.collection.fluentd

21.1.1.1.23.1. Description

FluentdForwarderSpec represents the configuration for forwarders of type fluentd.

21.1.1.1.23.1.1. Type

- object

Property	Type	Description
buffer	object	
inFile	object	

21.1.1.1.24. .spec.collection.fluentd.buffer

21.1.1.1.24.1. Description

FluentdBufferSpec represents a subset of fluentd buffer parameters to tune the buffer configuration for all fluentd outputs. It supports a subset of parameters to configure buffer and queue sizing, flush operations and retry flushing.

For general parameters refer to: https://docs.fluentd.org/configuration/buffer-section#buffering-parameters

For flush parameters refer to: https://docs.fluentd.org/configuration/buffer-section#flushing-parameters

For retry parameters refer to: https://docs.fluentd.org/configuration/buffer-section#retries-parameters

21.1.1.1.24.1.1. Type

- object

Property	Type	Description
chunkLimitSize	string	**(optional)** ChunkLimitSize represents the maximum size of each chunk. Events will be
flushInterval	string	**(optional)** FlushInterval represents the time duration to wait between two consecutive flush
flushMode	string	**(optional)** FlushMode represents the mode of the flushing thread to write chunks. The mode
flushThreadCount	int	**(optional)** FlushThreadCount reprents the number of threads used by the fluentd buffer
overflowAction	string	**(optional)** OverflowAction represents the action for the fluentd buffer plugin to
retryMaxInterval	string	**(optional)** RetryMaxInterval represents the maximum time interval for exponential backoff
retryTimeout	string	**(optional)** RetryTimeout represents the maximum time interval to attempt retries before giving up
retryType	string	**(optional)** RetryType represents the type of retrying flush operations. Flush operations can
retryWait	string	**(optional)** RetryWait represents the time duration between two consecutive retries to flush
totalLimitSize	string	**(optional)** TotalLimitSize represents the threshold of node space allowed per fluentd

21.1.1.1.25. .spec.collection.fluentd.inFile

21.1.1.1.25.1. Description

FluentdInFileSpec represents a subset of fluentd in-tail plugin parameters to tune the configuration for all fluentd in-tail inputs.

For general parameters refer to: https://docs.fluentd.org/input/tail#parameters

21.1.1.1.25.1.1. Type

- object

Property	Type	Description
readLinesLimit	int	**(optional)** ReadLinesLimit represents the number of lines to read with each I/O operation

21.1.1.1.26. .spec.collection.logs

21.1.1.1.26.1. Description

21.1.1.1.26.1.1. Type

- object

Property	Type	Description
fluentd	object	Specification of the Fluentd Log Collection component
type	string	The type of Log Collection to configure

21.1.1.1.27. .spec.collection.logs.fluentd

21.1.1.1.27.1. Description

CollectorSpec is spec to define scheduling and resources for a collector

21.1.1.1.27.1.1. Type

- object

Property	Type	Description
nodeSelector	object	**(optional)** Define which Nodes the Pods are scheduled on.
resources	object	**(optional)** The resource requirements for the collector
tolerations	array	**(optional)** Define the tolerations the Pods will accept

21.1.1.1.28. .spec.collection.logs.fluentd.nodeSelector

21.1.1.1.28.1. Description

21.1.1.1.28.1.1. Type

- object

21.1.1.1.29. .spec.collection.logs.fluentd.resources

21.1.1.1.29.1. Description

21.1.1.1.29.1.1. Type

- object

Property	Type	Description
limits	object	**(optional)** Limits describes the maximum amount of compute resources allowed.
requests	object	**(optional)** Requests describes the minimum amount of compute resources required.

21.1.1.1.30. .spec.collection.logs.fluentd.resources.limits

21.1.1.1.30.1. Description

21.1.1.1.30.1.1. Type

- object

21.1.1.1.31. .spec.collection.logs.fluentd.resources.requests

21.1.1.1.31.1. Description

21.1.1.1.31.1.1. Type

- object

21.1.1.1.32. .spec.collection.logs.fluentd.tolerations[]

21.1.1.1.32.1. Description

21.1.1.1.32.1.1. Type

- array

Property	Type	Description
effect	string	**(optional)** Effect indicates the taint effect to match. Empty means match all taint effects.
key	string	**(optional)** Key is the taint key that the toleration applies to. Empty means match all taint keys.
operator	string	**(optional)** Operator represents a key's relationship to the value.
tolerationSeconds	int	**(optional)** TolerationSeconds represents the period of time the toleration (which must be
value	string	**(optional)** Value is the taint value the toleration matches to.

21.1.1.1.33. .spec.collection.logs.fluentd.tolerations[].tolerationSeconds

21.1.1.1.33.1. Description

21.1.1.1.33.1.1. Type

- int

21.1.1.1.34. .spec.curation

21.1.1.1.34.1. Description

This is the struct that will contain information pertinent to Log curation (Curator)

21.1.1.1.34.1.1. Type

- object

Property	Type	Description
curator	object	The specification of curation to configure
type	string	The kind of curation to configure

21.1.1.1.35. .spec.curation.curator

21.1.1.1.35.1. Description

21.1.1.1.35.1.1. Type

- object

Property	Type	Description
nodeSelector	object	Define which Nodes the Pods are scheduled on.
resources	object	**(optional)** The resource requirements for Curator
schedule	string	The cron schedule that the Curator job is run. Defaults to "30 3 * * *"
tolerations	array	

21.1.1.1.36. .spec.curation.curator.nodeSelector

21.1.1.1.36.1. Description

21.1.1.1.36.1.1. Type

- object

21.1.1.1.37. .spec.curation.curator.resources

21.1.1.1.37.1. Description

21.1.1.1.37.1.1. Type

- object

Property	Type	Description
limits	object	**(optional)** Limits describes the maximum amount of compute resources allowed.
requests	object	**(optional)** Requests describes the minimum amount of compute resources required.

21.1.1.1.38. .spec.curation.curator.resources.limits

21.1.1.1.38.1. Description

21.1.1.1.38.1.1. Type

- object

21.1.1.1.39. .spec.curation.curator.resources.requests

21.1.1.1.39.1. Description

21.1.1.1.39.1.1. Type

- object

21.1.1.1.40. .spec.curation.curator.tolerations[]

21.1.1.1.40.1. Description

21.1.1.1.40.1.1. Type

- array

Property	Type	Description
effect	string	**(optional)** Effect indicates the taint effect to match. Empty means match all taint effects.
key	string	**(optional)** Key is the taint key that the toleration applies to. Empty means match all taint keys.
operator	string	**(optional)** Operator represents a key's relationship to the value.
tolerationSeconds	int	**(optional)** TolerationSeconds represents the period of time the toleration (which must be
value	string	**(optional)** Value is the taint value the toleration matches to.

21.1.1.1.41. .spec.curation.curator.tolerations[].tolerationSeconds

21.1.1.1.41.1. Description

21.1.1.1.41.1.1. Type

- int

21.1.1.1.42. .spec.forwarder

21.1.1.1.42.1. Description

ForwarderSpec contains global tuning parameters for specific forwarder implementations. This field is not required for general use, it allows performance tuning by users familiar with the underlying forwarder technology. Currently supported: **fluentd**.

21.1.1.1.42.1.1. Type

- object

Property	Type	Description
fluentd	object	

21.1.1.1.43. .spec.forwarder.fluentd

21.1.1.1.43.1. Description

FluentdForwarderSpec represents the configuration for forwarders of type fluentd.

21.1.1.1.43.1.1. Type

- object

Property	Type	Description
buffer	object	
inFile	object	

21.1.1.1.44. .spec.forwarder.fluentd.buffer

21.1.1.1.44.1. Description

FluentdBufferSpec represents a subset of fluentd buffer parameters to tune the buffer configuration for all fluentd outputs. It supports a subset of parameters to configure buffer and queue sizing, flush operations and retry flushing.

For general parameters refer to: https://docs.fluentd.org/configuration/buffer-section#buffering-parameters

For flush parameters refer to: https://docs.fluentd.org/configuration/buffer-section#flushing-parameters

For retry parameters refer to: https://docs.fluentd.org/configuration/buffer-section#retries-parameters

21.1.1.1.44.1.1. Type

- object

Property	Type	Description
chunkLimitSize	string	**(optional)** ChunkLimitSize represents the maximum size of each chunk. Events will be
flushInterval	string	**(optional)** FlushInterval represents the time duration to wait between two consecutive flush
flushMode	string	**(optional)** FlushMode represents the mode of the flushing thread to write chunks. The mode
flushThreadCount	int	**(optional)** FlushThreadCount reprents the number of threads used by the fluentd buffer
overflowAction	string	**(optional)** OverflowAction represents the action for the fluentd buffer plugin to
retryMaxInterval	string	**(optional)** RetryMaxInterval represents the maximum time interval for exponential backoff
retryTimeout	string	**(optional)** RetryTimeout represents the maximum time interval to attempt retries before giving up
retryType	string	**(optional)** RetryType represents the type of retrying flush operations. Flush operations can
retryWait	string	**(optional)** RetryWait represents the time duration between two consecutive retries to flush
totalLimitSize	string	**(optional)** TotalLimitSize represents the threshold of node space allowed per fluentd

21.1.1.1.45. .spec.forwarder.fluentd.inFile

21.1.1.1.45.1. Description

FluentdInFileSpec represents a subset of fluentd in-tail plugin parameters to tune the configuration for all fluentd in-tail inputs.

For general parameters refer to: https://docs.fluentd.org/input/tail#parameters

21.1.1.1.45.1.1. Type

- object

Property	Type	Description
readLinesLimit	int	**(optional)** ReadLinesLimit represents the number of lines to read with each I/O operation

21.1.1.1.46. .spec.logStore

21.1.1.1.46.1. Description

The LogStoreSpec contains information about how logs are stored.

21.1.1.1.46.1.1. Type

- object

Property	Type	Description
elasticsearch	object	Specification of the Elasticsearch Log Store component
lokistack	object	LokiStack contains information about which LokiStack to use for log storage if Type is set to LogStoreTypeLokiStack.
retentionPolicy	object	**(optional)** Retention policy defines the maximum age for an index after which it should be deleted
type	string	The Type of Log Storage to configure. The operator currently supports either using ElasticSearch

21.1.1.1.47. .spec.logStore.elasticsearch

21.1.1.1.47.1. Description

21.1.1.1.47.1.1. Type

- object

Property	Type	Description
nodeCount	int	Number of nodes to deploy for Elasticsearch
nodeSelector	object	Define which Nodes the Pods are scheduled on.
proxy	object	Specification of the Elasticsearch Proxy component
redundancyPolicy	string	**(optional)**
resources	object	**(optional)** The resource requirements for Elasticsearch
storage	object	**(optional)** The storage specification for Elasticsearch data nodes
tolerations	array	

21.1.1.1.48. .spec.logStore.elasticsearch.nodeSelector

21.1.1.1.48.1. Description

21.1.1.1.48.1.1. Type

- object

21.1.1.1.49. .spec.logStore.elasticsearch.proxy

21.1.1.1.49.1. Description

21.1.1.1.49.1.1. Type

- object

Property	Type	Description
resources	object	

21.1.1.1.50. .spec.logStore.elasticsearch.proxy.resources

21.1.1.1.50.1. Description

21.1.1.1.50.1.1. Type

- object

Property	Type	Description
limits	object	**(optional)** Limits describes the maximum amount of compute resources allowed.
requests	object	**(optional)** Requests describes the minimum amount of compute resources required.

21.1.1.1.51. .spec.logStore.elasticsearch.proxy.resources.limits

21.1.1.1.51.1. Description

21.1.1.1.51.1.1. Type

- object

21.1.1.1.52. .spec.logStore.elasticsearch.proxy.resources.requests

21.1.1.1.52.1. Description

21.1.1.1.52.1.1. Type

- object

21.1.1.1.53. .spec.logStore.elasticsearch.resources

21.1.1.1.53.1. Description

21.1.1.1.53.1.1. Type

- object

Property	Type	Description
limits	object	**(optional)** Limits describes the maximum amount of compute resources allowed.
requests	object	**(optional)** Requests describes the minimum amount of compute resources required.

21.1.1.1.54. .spec.logStore.elasticsearch.resources.limits

21.1.1.1.54.1. Description

21.1.1.1.54.1.1. Type

- object

21.1.1.1.55. .spec.logStore.elasticsearch.resources.requests

21.1.1.1.55.1. Description

21.1.1.1.55.1.1. Type

- object

21.1.1.1.56. .spec.logStore.elasticsearch.storage

21.1.1.1.56.1. Description

21.1.1.1.56.1.1. Type

- object

Property	Type	Description
size	object	The max storage capacity for the node to provision.
storageClassName	string	**(optional)** The name of the storage class to use with creating the node's PVC.

21.1.1.1.57. .spec.logStore.elasticsearch.storage.size

21.1.1.1.57.1. Description

21.1.1.1.57.1.1. Type

- object

Property	Type	Description
Format	string	Change Format at will. See the comment for Canonicalize for
d	object	d is the quantity in inf.Dec form if d.Dec != nil

Property	Type	Description
i	int	i is the quantity in int64 scaled form, if d.Dec == nil
s	string	s is the generated value of this quantity to avoid recalculation

21.1.1.1.58. .spec.logStore.elasticsearch.storage.size.d

21.1.1.1.58.1. Description

21.1.1.1.58.1.1. Type

- object

Property	Type	Description
Dec	object	

21.1.1.1.59. .spec.logStore.elasticsearch.storage.size.d.Dec

21.1.1.1.59.1. Description

21.1.1.1.59.1.1. Type

- object

Property	Type	Description
scale	int	
unscaled	object	

21.1.1.1.60. .spec.logStore.elasticsearch.storage.size.d.Dec.unscaled

21.1.1.1.60.1. Description

21.1.1.1.60.1.1. Type

- object

Property	Type	Description
abs	Word	sign

Property	Type	Description
neg	bool	

21.1.1.1.61. .spec.logStore.elasticsearch.storage.size.d.Dec.unscaled.abs

21.1.1.1.61.1. Description

21.1.1.1.61.1.1. Type

- Word

21.1.1.1.62. .spec.logStore.elasticsearch.storage.size.i

21.1.1.1.62.1. Description

21.1.1.1.62.1.1. Type

- int

Property	Type	Description
scale	int	
value	int	

21.1.1.1.63. .spec.logStore.elasticsearch.tolerations[]

21.1.1.1.63.1. Description

21.1.1.1.63.1.1. Type

- array

Property	Type	Description
effect	string	**(optional)** Effect indicates the taint effect to match. Empty means match all taint effects.
key	string	**(optional)** Key is the taint key that the toleration applies to. Empty means match all taint keys.
operator	string	**(optional)** Operator represents a key's relationship to the value.

Property	Type	Description
tolerationSeconds	int	**(optional)** TolerationSeconds represents the period of time the toleration (which must be
value	string	**(optional)** Value is the taint value the toleration matches to.

21.1.1.1.64. .spec.logStore.elasticsearch.tolerations[].tolerationSeconds

21.1.1.1.64.1. Description

21.1.1.1.64.1.1. Type

- int

21.1.1.1.65. .spec.logStore.lokistack

21.1.1.1.65.1. Description

LokiStackStoreSpec is used to set up cluster-logging to use a LokiStack as logging storage. It points to an existing LokiStack in the same namespace.

21.1.1.1.65.1.1. Type

- object

Property	Type	Description
name	string	Name of the LokiStack resource.

21.1.1.1.66. .spec.logStore.retentionPolicy

21.1.1.1.66.1. Description

21.1.1.1.66.1.1. Type

- object

Property	Type	Description
application	object	
audit	object	

Property	Type	Description
infra	object	

21.1.1.1.67. .spec.logStore.retentionPolicy.application

21.1.1.1.67.1. Description

21.1.1.1.67.1.1. Type

- object

Property	Type	Description
diskThresholdPercent	int	**(optional)** The threshold percentage of ES disk usage that when reached, old indices should be deleted (e.g. 75)
maxAge	string	**(optional)**
namespaceSpec	array	**(optional)** The per namespace specification to delete documents older than a given minimum age
pruneNamespacesInterval	string	**(optional)** How often to run a new prune-namespaces job

21.1.1.1.68. .spec.logStore.retentionPolicy.application.namespaceSpec[]

21.1.1.1.68.1. Description

21.1.1.1.68.1.1. Type

- array

Property	Type	Description
minAge	string	**(optional)** Delete the records matching the namespaces which are older than this MinAge (e.g. 1d)
namespace	string	Target Namespace to delete logs older than MinAge (defaults to 7d)

21.1.1.1.69. .spec.logStore.retentionPolicy.audit

21.1.1.1.69.1. Description

21.1.1.1.69.1.1. Type

- object

Property	Type	Description
diskThresholdPercent	int	**(optional)** The threshold percentage of ES disk usage that when reached, old indices should be deleted (e.g. 75)
maxAge	string	**(optional)**
namespaceSpec	array	**(optional)** The per namespace specification to delete documents older than a given minimum age
pruneNamespacesInterval	string	**(optional)** How often to run a new prune-namespaces job

21.1.1.1.70. .spec.logStore.retentionPolicy.audit.namespaceSpec[]

21.1.1.1.70.1. Description

21.1.1.1.70.1.1. Type

- array

Property	Type	Description
minAge	string	**(optional)** Delete the records matching the namespaces which are older than this MinAge (e.g. 1d)
namespace	string	Target Namespace to delete logs older than MinAge (defaults to 7d)

21.1.1.1.71. .spec.logStore.retentionPolicy.infra

21.1.1.1.71.1. Description

21.1.1.1.71.1.1. Type

- object

Property	Type	Description
diskThresholdPercent	int	**(optional)** The threshold percentage of ES disk usage that when reached, old indices should be deleted (e.g. 75)
maxAge	string	**(optional)**
namespaceSpec	array	**(optional)** The per namespace specification to delete documents older than a given minimum age
pruneNamespacesInterval	string	**(optional)** How often to run a new prune-namespaces job

21.1.1.1.72. .spec.logStore.retentionPolicy.infra.namespaceSpec[]

21.1.1.1.72.1. Description

21.1.1.1.72.1.1. Type

- array

Property	Type	Description
minAge	string	**(optional)** Delete the records matching the namespaces which are older than this MinAge (e.g. 1d)
namespace	string	Target Namespace to delete logs older than MinAge (defaults to 7d)

21.1.1.1.73. .spec.visualization

21.1.1.1.73.1. Description

This is the struct that will contain information pertinent to Log visualization (Kibana)

21.1.1.1.73.1.1. Type

- object

Property	Type	Description
kibana	object	Specification of the Kibana Visualization component
type	string	The type of Visualization to configure

21.1.1.1.74. .spec.visualization.kibana

21.1.1.1.74.1. Description

21.1.1.1.74.1.1. Type

- object

Property	Type	Description
nodeSelector	object	Define which Nodes the Pods are scheduled on.
proxy	object	Specification of the Kibana Proxy component
replicas	int	Number of instances to deploy for a Kibana deployment
resources	object	**(optional)** The resource requirements for Kibana
tolerations	array	

21.1.1.1.75. .spec.visualization.kibana.nodeSelector

21.1.1.1.75.1. Description

21.1.1.1.75.1.1. Type

- object

21.1.1.1.76. .spec.visualization.kibana.proxy

21.1.1.1.76.1. Description

21.1.1.1.76.1.1. Type

- object

Property	Type	Description
resources	object	

21.1.1.1.77. .spec.visualization.kibana.proxy.resources

21.1.1.1.77.1. Description

21.1.1.1.77.1.1. Type

- object

Property	Type	Description
limits	object	**(optional)** Limits describes the maximum amount of compute resources allowed.
requests	object	**(optional)** Requests describes the minimum amount of compute resources required.

21.1.1.1.78. .spec.visualization.kibana.proxy.resources.limits

21.1.1.1.78.1. Description

21.1.1.1.78.1.1. Type

- object

21.1.1.1.79. .spec.visualization.kibana.proxy.resources.requests

21.1.1.1.79.1. Description

21.1.1.1.79.1.1. Type

- object

21.1.1.1.80. .spec.visualization.kibana.replicas

21.1.1.1.80.1. Description

21.1.1.1.80.1.1. Type

- int

21.1.1.1.81. .spec.visualization.kibana.resources

21.1.1.1.81.1. Description

21.1.1.1.81.1.1. Type

- object

Property	Type	Description
limits	object	**(optional)** Limits describes the maximum amount of compute resources allowed.
requests	object	**(optional)** Requests describes the minimum amount of compute resources required.

21.1.1.1.82. .spec.visualization.kibana.resources.limits

21.1.1.1.82.1. Description

21.1.1.1.82.1.1. Type

- object

21.1.1.1.83. .spec.visualization.kibana.resources.requests

21.1.1.1.83.1. Description

21.1.1.1.83.1.1. Type

- object

21.1.1.1.84. .spec.visualization.kibana.tolerations[]

21.1.1.1.84.1. Description

21.1.1.1.84.1.1. Type

- array

Property	Type	Description
effect	string	**(optional)** Effect indicates the taint effect to match. Empty means match all taint effects.

Property	Type	Description
key	string	**(optional)** Key is the taint key that the toleration applies to. Empty means match all taint keys.
operator	string	**(optional)** Operator represents a key's relationship to the value.
tolerationSeconds	int	**(optional)** TolerationSeconds represents the period of time the toleration (which must be
value	string	**(optional)** Value is the taint value the toleration matches to.

21.1.1.1.85. .spec.visualization.kibana.tolerations[].tolerationSeconds

21.1.1.1.85.1. Description

21.1.1.1.85.1.1. Type

- int

21.1.1.1.86. .status

21.1.1.1.86.1. Description

ClusterLoggingStatus defines the observed state of ClusterLogging

21.1.1.1.86.1.1. Type

- object

Property	Type	Description
collection	object	**(optional)**
conditions	object	**(optional)**
curation	object	**(optional)**
logStore	object	**(optional)**
visualization	object	**(optional)**

21.1.1.1.87. .status.collection

21.1.1.1.87.1. Description

21.1.1.1.87.1.1. Type

- object

Property	Type	Description
logs	object	**(optional)**

21.1.1.1.88. .status.collection.logs

21.1.1.1.88.1. Description

21.1.1.1.88.1.1. Type

- object

Property	Type	Description
fluentdStatus	object	**(optional)**

21.1.1.1.89. .status.collection.logs.fluentdStatus

21.1.1.1.89.1. Description

21.1.1.1.89.1.1. Type

- object

Property	Type	Description
clusterCondition	object	**(optional)**
daemonSet	string	**(optional)**
nodes	object	**(optional)**
pods	string	**(optional)**

21.1.1.1.90. .status.collection.logs.fluentdStatus.clusterCondition

21.1.1.1.90.1. Description

operator-sdk generate crds does not allow map-of-slice, must use a named type.

21.1.1.1.90.1.1. Type

- object

21.1.1.1.91. .status.collection.logs.fluentdStatus.nodes

21.1.1.1.91.1. Description

21.1.1.1.91.1.1. Type

- object

21.1.1.1.92. .status.conditions

21.1.1.1.92.1. Description

21.1.1.1.92.1.1. Type

- object

21.1.1.1.93. .status.curation

21.1.1.1.93.1. Description

21.1.1.1.93.1.1. Type

- object

Property	Type	Description
curatorStatus	array	**(optional)**

21.1.1.1.94. .status.curation.curatorStatus[]

21.1.1.1.94.1. Description

21.1.1.1.94.1.1. Type

- array

Property	Type	Description
clusterCondition	object	**(optional)**
cronJobs	string	**(optional)**
schedules	string	**(optional)**

Property	Type	Description
suspended	bool	**(optional)**

21.1.1.1.95. .status.curation.curatorStatus[].clusterCondition

21.1.1.1.95.1. Description

operator-sdk generate crds does not allow map-of-slice, must use a named type.

21.1.1.1.95.1.1. Type

- object

21.1.1.1.96. .status.logStore

21.1.1.1.96.1. Description

21.1.1.1.96.1.1. Type

- object

Property	Type	Description
elasticsearchStatus	array	**(optional)**

21.1.1.1.97. .status.logStore.elasticsearchStatus[]

21.1.1.1.97.1. Description

21.1.1.1.97.1.1. Type

- array

Property	Type	Description
cluster	object	**(optional)**
clusterConditions	object	**(optional)**
clusterHealth	string	**(optional)**
clusterName	string	**(optional)**
deployments	array	**(optional)**

Property	Type	Description
nodeConditions	object	**(optional)**
nodeCount	int	**(optional)**
pods	object	**(optional)**
replicaSets	array	**(optional)**
shardAllocationEnabled	string	**(optional)**
statefulSets	array	**(optional)**

21.1.1.1.98. .status.logStore.elasticsearchStatus[].cluster

21.1.1.1.98.1. Description

21.1.1.1.98.1.1. Type

- object

Property	Type	Description
activePrimaryShards	int	The number of Active Primary Shards for the Elasticsearch Cluster
activeShards	int	The number of Active Shards for the Elasticsearch Cluster
initializingShards	int	The number of Initializing Shards for the Elasticsearch Cluster
numDataNodes	int	The number of Data Nodes for the Elasticsearch Cluster
numNodes	int	The number of Nodes for the Elasticsearch Cluster
pendingTasks	int	
relocatingShards	int	The number of Relocating Shards for the Elasticsearch Cluster
status	string	The current Status of the Elasticsearch Cluster

Property	Type	Description
unassignedShards	int	The number of Unassigned Shards for the Elasticsearch Cluster

21.1.1.1.99. .status.logStore.elasticsearchStatus[].clusterConditions

21.1.1.1.99.1. Description

21.1.1.1.99.1.1. Type

- object

21.1.1.1.100. .status.logStore.elasticsearchStatus[].deployments[]

21.1.1.1.100.1. Description

21.1.1.1.100.1.1. Type

- array

21.1.1.1.101. .status.logStore.elasticsearchStatus[].nodeConditions

21.1.1.1.101.1. Description

21.1.1.1.101.1.1. Type

- object

21.1.1.1.102. .status.logStore.elasticsearchStatus[].pods

21.1.1.1.102.1. Description

21.1.1.1.102.1.1. Type

- object

21.1.1.1.103. .status.logStore.elasticsearchStatus[].replicaSets[]

21.1.1.1.103.1. Description

21.1.1.1.103.1.1. Type

- array

21.1.1.1.104. .status.logStore.elasticsearchStatus[].statefulSets[]

21.1.1.1.104.1. Description

21.1.1.1.104.1.1. Type

- array

21.1.1.1.105. .status.visualization

21.1.1.1.105.1. Description

21.1.1.1.105.1.1. Type

- object

Property	Type	Description
kibanaStatus	array	**(optional)**

21.1.1.1.106. .status.visualization.kibanaStatus[]

21.1.1.1.106.1. Description

21.1.1.1.106.1.1. Type

- array

Property	Type	Description
clusterCondition	object	**(optional)**
deployment	string	**(optional)**
pods	string	**(optional)** The status for each of the Kibana pods for the Visualization component
replicaSets	array	**(optional)**
replicas	int	**(optional)**

21.1.1.1.107. .status.visualization.kibanaStatus[].clusterCondition

21.1.1.1.107.1. Description

21.1.1.1.107.1.1. Type

- object

21.1.1.1.108. .status.visualization.kibanaStatus[].replicaSets[]

21.1.1.1.108.1. Description

21.1.1.1.108.1.1. Type

- array

CHAPTER 22. GLOSSARY

This glossary defines common terms that are used in the logging documentation.

Annotation

You can use annotations to attach metadata to objects.

Red Hat OpenShift Logging Operator

The Red Hat OpenShift Logging Operator provides a set of APIs to control the collection and forwarding of application, infrastructure, and audit logs.

Custom resource (CR)

A CR is an extension of the Kubernetes API. To configure the logging and log forwarding, you can customize the **ClusterLogging** and the **ClusterLogForwarder** custom resources.

Event router

The event router is a pod that watches Red Hat OpenShift Service on AWS events. It collects logs by using the logging.

Fluentd

Fluentd is a log collector that resides on each Red Hat OpenShift Service on AWS node. It gathers application, infrastructure, and audit logs and forwards them to different outputs.

Garbage collection

Garbage collection is the process of cleaning up cluster resources, such as terminated containers and images that are not referenced by any running pods.

Elasticsearch

Elasticsearch is a distributed search and analytics engine. Red Hat OpenShift Service on AWS uses Elasticsearch as a default log store for the logging.

OpenShift Elasticsearch Operator

The OpenShift Elasticsearch Operator is used to run an Elasticsearch cluster on Red Hat OpenShift Service on AWS. The OpenShift Elasticsearch Operator provides self-service for the Elasticsearch cluster operations and is used by the logging.

Indexing

Indexing is a data structure technique that is used to quickly locate and access data. Indexing optimizes the performance by minimizing the amount of disk access required when a query is processed.

JSON logging

The Log Forwarding API enables you to parse JSON logs into a structured object and forward them to either the logging managed Elasticsearch or any other third-party system supported by the Log Forwarding API.

Kibana

Kibana is a browser-based console interface to query, discover, and visualize your Elasticsearch data through histograms, line graphs, and pie charts.

Kubernetes API server

Kubernetes API server validates and configures data for the API objects.

Labels

Labels are key-value pairs that you can use to organize and select subsets of objects, such as a pod.

Logging

With the logging, you can aggregate application, infrastructure, and audit logs throughout your cluster. You can also store them to a default log store, forward them to third party systems, and query and visualize the stored logs in the default log store.

Logging collector

A logging collector collects logs from the cluster, formats them, and forwards them to the log store or third party systems.

Log store

A log store is used to store aggregated logs. You can use an internal log store or forward logs to external log stores.

Log visualizer

Log visualizer is the user interface (UI) component you can use to view information such as logs, graphs, charts, and other metrics.

Node

A node is a worker machine in the Red Hat OpenShift Service on AWS cluster. A node is either a virtual machine (VM) or a physical machine.

Operators

Operators are the preferred method of packaging, deploying, and managing a Kubernetes application in an Red Hat OpenShift Service on AWS cluster. An Operator takes human operational knowledge and encodes it into software that is packaged and shared with customers.

Pod

A pod is the smallest logical unit in Kubernetes. A pod consists of one or more containers and runs on a worker node.

Role-based access control (RBAC)

RBAC is a key security control to ensure that cluster users and workloads have access only to resources required to execute their roles.

Shards

Elasticsearch organizes log data from Fluentd into datastores, or indices, then subdivides each index into multiple pieces called shards.

Taint

Taints ensure that pods are scheduled onto appropriate nodes. You can apply one or more taints on a node.

Toleration

You can apply tolerations to pods. Tolerations allow the scheduler to schedule pods with matching taints.

Web console

A user interface (UI) to manage Red Hat OpenShift Service on AWS. The web console for Red Hat OpenShift Service on AWS can be found at https://console.redhat.com/openshift.