OpenShift Dedicated 4

Monitoring

Monitoring projects in OpenShift Dedicated

Last Updated: 2026-03-31

OpenShift Dedicated 4 Monitoring

Monitoring projects in OpenShift Dedicated

Legal Notice

Abstract

This document provides information about monitoring projects in OpenShift Dedicated.

Table of Contents

CHAPTER 1. ABOUT MONITORING

1.1. ABOUT OPENSHIFT DEDICATED MONITORING

In OpenShift Dedicated, you can monitor your own projects in isolation from Red Hat Site Reliability Engineering (SRE) platform metrics. You can monitor your own projects without the need for an additional monitoring solution.

The OpenShift Dedicated monitoring stack is based on the Prometheus open source project and its wider ecosystem.

1.2. MONITORING STACK ARCHITECTURE

You can learn about the monitoring stack architecture, which includes default monitoring components and components for monitoring user-defined projects.

1.2.1. Understanding the monitoring stack

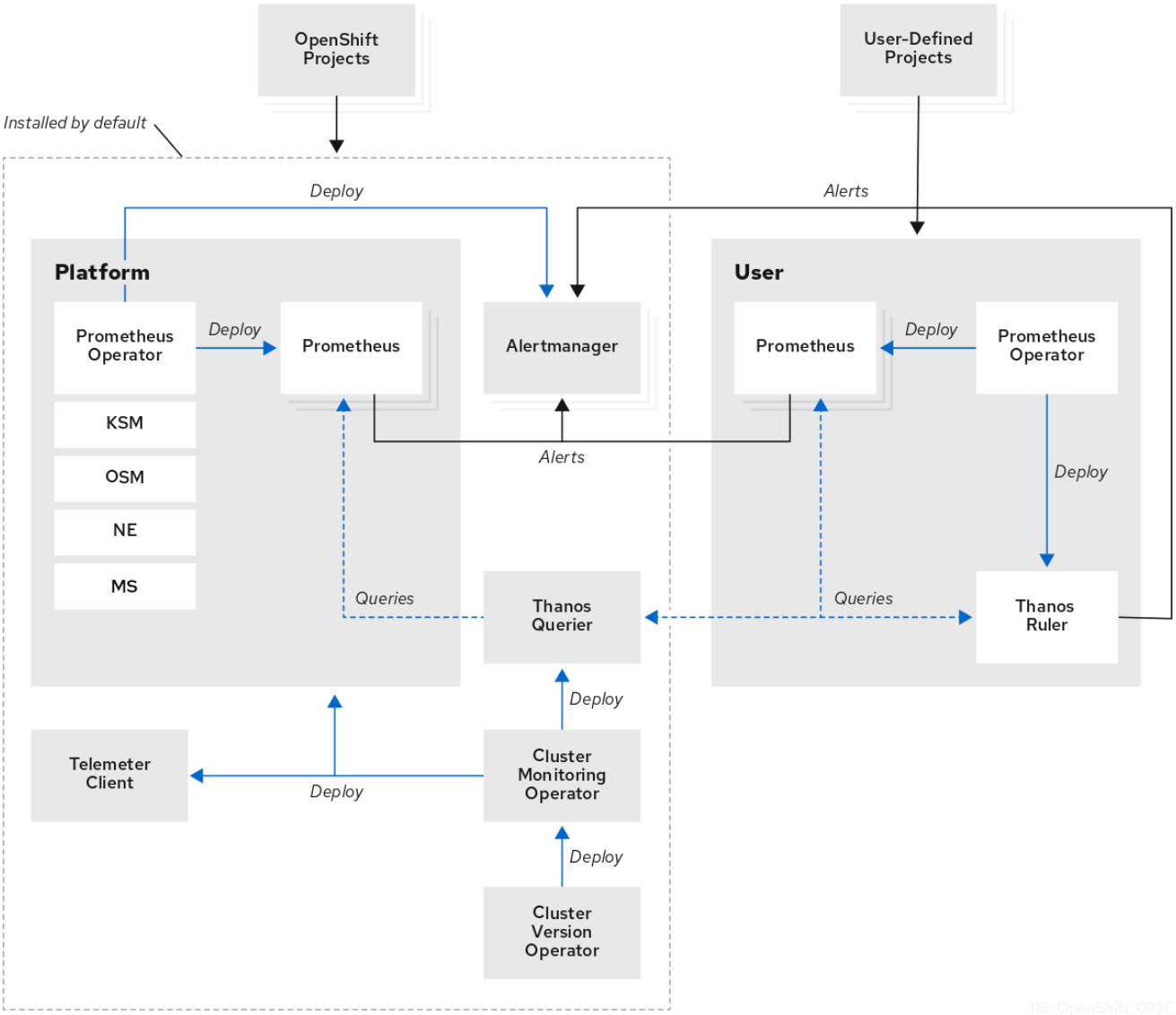
The monitoring stack includes the following components:

Default platform monitoring components

A set of platform monitoring components are installed in the **openshift-monitoring** project by default during a OpenShift Dedicated installation. Red Hat Site Reliability Engineers (SRE) use these components to monitor core cluster components including Kubernetes services. This includes critical metrics, such as CPU and memory, collected from all of the workloads in every namespace.
You can see these components in the **Installed by default** section in the following diagram.

Components for monitoring user-defined projects

A set of user-defined project monitoring components are installed in the **openshift-user-workload-monitoring** project by default during a OpenShift Dedicated installation. You can use these components to monitor services and pods in user-defined projects.
You can see these components in the **User** section in the following diagram.

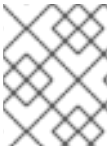


1.2.1.1. Default monitoring targets

The following are examples of targets monitored by Red Hat Site Reliability Engineers (SRE) in your OpenShift Dedicated cluster:

- CoreDNS
- etcd
- HAProxy
- Image registry
- Kubelets
- Kubernetes API server
- Kubernetes controller manager
- Kubernetes scheduler
- OpenShift API server

- OpenShift Controller Manager
- Operator Lifecycle Manager (OLM)



NOTE

The exact list of targets can vary depending on your cluster capabilities and installed components.

Additional resources

- Getting detailed information about a metrics target

1.2.2. Components for monitoring user-defined projects

OpenShift Dedicated includes an optional enhancement to the monitoring stack that helps you monitor services and pods in user-defined projects. This feature includes the following components:

Table 1.1. Components for monitoring user-defined projects

Component	Description
Prometheus Operator	The Prometheus Operator in the **openshift-user-workload-monitoring** project creates, configures, and manages Prometheus and Thanos Ruler instances in the same project.
Prometheus	Prometheus is the monitoring system that provides monitoring for user-defined projects. Prometheus sends alerts to Alertmanager for processing.
Thanos Ruler	The Thanos Ruler is a rule evaluation engine for Prometheus that is deployed as a separate process. In OpenShift Dedicated , Thanos Ruler provides rule and alerting evaluation for the monitoring of user-defined projects.
Alertmanager	The Alertmanager service handles alerts received from Prometheus and Thanos Ruler. Alertmanager is also responsible for sending user-defined alerts to external notification systems. Deploying this service is optional.

The monitoring stack monitors all components for user-defined projects. The components are automatically updated when OpenShift Dedicated is updated.

1.2.2.1. Monitoring targets for user-defined projects

Monitoring is enabled by default for OpenShift Dedicated user-defined projects. You can monitor:

- Metrics provided through service endpoints in user-defined projects.

- Pods running in user-defined projects.

1.2.3. The monitoring stack in high-availability clusters

By default, in multi-node clusters, the following components run in high-availability (HA) mode to prevent data loss and service interruption:

- Prometheus
- Alertmanager
- Thanos Ruler

The component is replicated across two pods, each running on a separate node. This means that the monitoring stack can tolerate the loss of one pod.

Prometheus in HA mode

- Both replicas independently scrape the same targets and evaluate the same rules.
- The replicas do not communicate with each other. Therefore, data might differ between the pods.

Alertmanager in HA mode

- The two replicas synchronize notification and silence states with each other. This ensures that each notification is sent at least once.
- If the replicas fail to communicate or if there is an issue on the receiving side, notifications are still sent, but they might be duplicated.

IMPORTANT

Prometheus, Alertmanager, and Thanos Ruler are stateful components. To ensure high availability, you must configure them with persistent storage.

Additional resources

- Configuring persistent storage
- Configuring performance and scalability

1.2.4. TLS security and rotation in the monitoring stack

Learn how TLS profiles and certificate rotation work in the OpenShift Dedicated monitoring stack to keep communication secure.

TLS security profiles for monitoring components

All components of the monitoring stack use the TLS security profile settings that are centrally configured by a cluster administrator. The monitoring stack component uses the TLS security profile settings that already exist in the **tlsSecurityProfile** field in the global OpenShift Dedicated **apiservers.config.openshift.io/cluster** resource.

TLS certificate rotation and automatic restarts

The Cluster Monitoring Operator manages the internal TLS certificate lifecycle for the monitoring components. These certificates secure the internal communication between the monitoring components.
During certificate rotation, the CMO updates secrets and config maps, which triggers automatic restarts of affected pods. This is an expected behavior, and the pods recover automatically.

The following example shows events that occur during certificate rotation:

```
$ oc get events -n openshift-monitoring

LAST SEEN   TYPE      REASON              OBJECT                                  MESSAGE
2h39m       Normal    SecretUpdated       deployment/cluster-monitoring-operator   Updated Secret/grpc-tls -n openshift-monitoring because it changed
2h39m       Normal    SecretCreated       deployment/cluster-monitoring-operator   Created Secret/prometheus-user-workload-grpc-tls -n openshift-user-workload-monitoring because it was missing
2h39m       Normal    SecretCreated       deployment/cluster-monitoring-operator   Created Secret/thanos-querier-grpc-tls -n openshift-monitoring because it was missing
2h39m       Normal    SecretCreated       deployment/cluster-monitoring-operator   Created Secret/thanos-ruler-grpc-tls -n openshift-user-workload-monitoring because it was missing
2h39m       Normal    SecretCreated       deployment/cluster-monitoring-operator   Created Secret/prometheus-k8s-grpc-tls -n openshift-monitoring because it was missing
2h38m       Warning   FailedMount         pod/prometheus-k8s-0                    MountVolume.SetUp failed for volume "secret-grpc-tls" : secret "prometheus-k8s-grpc-tls" not found
2h39m       Normal    Created             pod/prometheus-k8s-0                    Created container kube-rbac-proxy-thanos
2h39m       Normal    Started             pod/prometheus-k8s-0                    Started container kube-rbac-proxy-thanos
2h39m       Normal    SuccessfulDelete    statefulset/prometheus-k8s              delete Pod prometheus-k8s-0 in StatefulSet prometheus-k8s successful
2h39m       Normal    SuccessfulCreate    statefulset/prometheus-k8s              create Pod prometheus-k8s-0 in StatefulSet prometheus-k8s successful
```

1.2.5. Glossary of common terms for OpenShift Dedicated monitoring

This glossary defines common terms that are used in OpenShift Dedicated architecture.

Alertmanager

Alertmanager handles alerts received from Prometheus. Alertmanager is also responsible for sending the alerts to external notification systems.

Alerting rules

Alerting rules contain a set of conditions that outline a particular state within a cluster. Alerts are triggered when those conditions are true. An alerting rule can be assigned a severity that defines how the alerts are routed.

Cluster Monitoring Operator

The Cluster Monitoring Operator (CMO) is a central component of the monitoring stack. It deploys and manages Prometheus instances such as, the Thanos Querier, the Telemeter Client, and metrics targets to ensure that they are up to date. The CMO is deployed by the Cluster Version Operator (CVO).

Cluster Version Operator

The Cluster Version Operator (CVO) manages the lifecycle of cluster Operators, many of which are installed in OpenShift Dedicated by default.

config map

A config map provides a way to inject configuration data into pods. You can reference the data stored in a config map in a volume of type **ConfigMap**. Applications running in a pod can use this data.

Container

A container is a lightweight and executable image that includes software and all its dependencies. Containers virtualize the operating system. As a result, you can run containers anywhere from a data center to a public or private cloud as well as a developer's laptop.

custom resource (CR)

A CR is an extension of the Kubernetes API. You can create custom resources.

etcd

etcd is the key-value store for OpenShift Dedicated, which stores the state of all resource objects.

Kubelets

Runs on nodes and reads the container manifests. Ensures that the defined containers have started and are running.

Kubernetes API server

Kubernetes API server validates and configures data for the API objects.

Kubernetes controller manager

Kubernetes controller manager governs the state of the cluster.

Kubernetes scheduler

Kubernetes scheduler allocates pods to nodes.

labels

Labels are key-value pairs that you can use to organize and select subsets of objects such as a pod.

Metrics Server

The Metrics Server monitoring component collects resource metrics and exposes them in the **metrics.k8s.io** Metrics API service for use by other tools and APIs, which frees the core platform Prometheus stack from handling this functionality.

node

A compute machine in the OpenShift Dedicated cluster. A node is either a virtual machine (VM) or a physical machine.

Operator

The preferred method of packaging, deploying, and managing a Kubernetes application in your OpenShift Dedicated cluster. An Operator takes human operational knowledge and encodes it into software that is packaged and shared with customers.

Operator Lifecycle Manager (OLM)

OLM helps you install, update, and manage the lifecycle of Kubernetes native applications. OLM is an open source toolkit designed to manage Operators in an effective, automated, and scalable way.

Persistent storage

Stores the data even after the device is shut down. Kubernetes uses persistent volumes to store the application data.

Persistent volume claim (PVC)

You can use a PVC to mount a PersistentVolume into a Pod. You can access the storage without knowing the details of the cloud environment.

pod

The pod is the smallest logical unit in Kubernetes. A pod is comprised of one or more containers to run in a worker node.

Prometheus

Prometheus is the monitoring system on which the OpenShift Dedicated monitoring stack is based. Prometheus is a time-series database and a rule evaluation engine for metrics. Prometheus sends alerts to Alertmanager for processing.

Prometheus Operator

The Prometheus Operator in the **openshift-monitoring** project creates, configures, and manages platform Prometheus and Alertmanager instances. It also automatically generates monitoring target configurations based on Kubernetes label queries.

Silences

A silence can be applied to an alert to prevent notifications from being sent when the conditions for an alert are true. You can mute an alert after the initial notification, while you work on resolving the underlying issue.

storage

OpenShift Dedicated supports many types of storage on AWS and Google Cloud. You can manage container storage for persistent and non-persistent data in your OpenShift Dedicated cluster.

Thanos Ruler

The Thanos Ruler is a rule evaluation engine for Prometheus that is deployed as a separate process. In OpenShift Dedicated, Thanos Ruler provides rule and alerting evaluation for the monitoring of user-defined projects.

Vector

Vector is a log collector that deploys to each OpenShift Dedicated node. It collects log data from each node, transforms the data, and forwards it to configured outputs.

web console

A user interface (UI) to manage OpenShift Dedicated.

1.2.6. Additional resources

- About remote health monitoring

1.3. UNDERSTANDING THE MONITORING STACK - KEY CONCEPTS

Get familiar with the OpenShift Dedicated monitoring concepts and terms. Learn about how you can improve performance and scale of your cluster, store and record data, manage metrics and alerts, and more.

1.3.1. About performance and scalability

You can optimize the performance and scale of your clusters. You can configure the monitoring stack by performing any of the following actions:

- Control the placement and distribution of monitoring components:
  - Use node selectors to move components to specific nodes.
  - Assign tolerations to enable moving components to tainted nodes.
- Use pod topology spread constraints.
- Manage CPU and memory resources.

Additional resources

- Configuring performance and scalability for user workload monitoring

1.3.1.1. Using node selectors to move monitoring components

By using the **nodeSelector** constraint with labeled nodes, you can move any of the monitoring stack components to specific nodes. By doing so, you can control the placement and distribution of the monitoring components across a cluster.

By controlling placement and distribution of monitoring components, you can optimize system resource use, improve performance, and separate workloads based on specific requirements or policies.

How node selectors work with other constraints
If you move monitoring components by using node selector constraints, be aware that other constraints to control pod scheduling might exist for a cluster:

- Topology spread constraints might be in place to control pod placement.
- Hard anti-affinity rules are in place for Prometheus, Alertmanager, and other monitoring components to ensure that multiple pods for these components are always spread across different nodes and are therefore always highly available.

When scheduling pods onto nodes, the pod scheduler tries to satisfy all existing constraints when determining pod placement. That is, all constraints compound when the pod scheduler determines which pods will be placed on which nodes.

Therefore, if you configure a node selector constraint but existing constraints cannot all be satisfied, the pod scheduler cannot match all constraints and will not schedule a pod for placement onto a node.

To maintain resilience and high availability for monitoring components, ensure that enough nodes are available and match all constraints when you configure a node selector constraint to move a component.

1.3.1.2. About pod topology spread constraints for monitoring

You can use pod topology spread constraints to control how the monitoring pods are spread across a network topology when OpenShift Dedicated pods are deployed in multiple availability zones.

Pod topology spread constraints are suitable for controlling pod scheduling within hierarchical topologies in which nodes are spread across different infrastructure levels, such as regions and zones within those regions. Additionally, by being able to schedule pods in different zones, you can improve network latency in certain scenarios.

You can configure pod topology spread constraints for all the pods deployed by the Cluster Monitoring Operator to control how pod replicas are scheduled to nodes across zones. This ensures that the pods are highly available and run more efficiently, because workloads are spread across nodes in different data centers or hierarchical infrastructure zones.

1.3.1.3. About specifying limits and requests for monitoring components

You can configure resource limits and requests for the following components that monitor user-defined projects:

- Alertmanager
- Prometheus

- Thanos Ruler

By defining the resource limits, you limit a container's resource usage, which prevents the container from exceeding the specified maximum values for CPU and memory resources.

By defining the resource requests, you specify that a container can be scheduled only on a node that has enough CPU and memory resources available to match the requested resources.

1.3.2. About storing and recording data

You can store and record data to help you protect the data and use them for troubleshooting. You can configure the monitoring stack by performing any of the following actions:

- Configure persistent storage:
  - Protect your metrics and alerting data from data loss by storing them in a persistent volume (PV). As a result, they can survive pods being restarted or recreated.
  - Avoid getting duplicate notifications and losing silences for alerts when the Alertmanager pods are restarted.
- Modify the retention time and size for Prometheus and Thanos Ruler metrics data.
- Configure logging to help you troubleshoot issues with your cluster:
  - Set log levels for monitoring.
  - Enable the query logging for Prometheus and Thanos Querier.

Additional resources

- Storing and recording data for user workload monitoring

1.3.2.1. Retention time and size for Prometheus metrics

By default, Prometheus retains metrics data for the following durations:

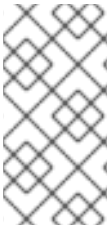
- **Core platform monitoring**: 15 days
- **Monitoring for user-defined projects**: 24 hours

You can modify the retention time for the Prometheus instance to change how soon the data is deleted. You can also set the maximum amount of disk space the retained metrics data uses. If the data reaches this size limit, Prometheus deletes the oldest data first until the disk space used is again below the limit.

Note the following behaviors of these data retention settings:

- The size-based retention policy applies to all data block directories in the /**prometheus** directory, including persistent blocks, write-ahead log (WAL) data, and m-mapped chunks.
- Data in the /**wal** and /**head_chunks** directories counts toward the retention size limit, but Prometheus never purges data from these directories based on size- or time-based retention policies. Thus, if you set a retention size limit lower than the maximum size set for the /**wal** and /**head_chunks** directories, you have configured the system not to retain any data blocks in the /**prometheus** data directories.

- The size-based retention policy is applied only when Prometheus cuts a new data block, which occurs every two hours after the WAL contains at least three hours of data.
- If you do not explicitly define values for either **retention** or **retentionSize**, retention time defaults to 15 days for core platform monitoring and 24 hours for user-defined project monitoring. Retention size is not set.
- If you define values for both **retention** and **retentionSize**, both values apply. If any data blocks exceed the defined retention time or the defined size limit, Prometheus purges these data blocks.
- If you define a value for **retentionSize** and do not define **retention**, only the **retentionSize** value applies.
- If you do not define a value for **retentionSize** and only define a value for **retention**, only the **retention** value applies.
- If you set the **retentionSize** or **retention** value to **0**, the default settings apply. The default settings set retention time to 15 days for core platform monitoring and 24 hours for user-defined project monitoring. By default, retention size is not set.



NOTE

Data compaction occurs every two hours. Therefore, a persistent volume (PV) might fill up before compaction, potentially exceeding the **retentionSize** limit. In such cases, the **KubePersistentVolumeFillingUp** alert fires until the space on a PV is lower than the **retentionSize** limit.

1.3.3. Understanding metrics

In OpenShift Dedicated, cluster components are monitored by scraping metrics exposed through service endpoints. You can also configure metrics collection for user-defined projects. Metrics enable you to monitor how cluster components and your own workloads are performing.

You can define the metrics that you want to provide for your own workloads by using Prometheus client libraries at the application level.

In OpenShift Dedicated, metrics are exposed through an HTTP service endpoint under the **/metrics** canonical name. You can list all available metrics for a service by running a **curl** query against **http://<endpoint>/metrics**. For instance, you can expose a route to the **prometheus-example-app** example application and then run the following to view all of its available metrics:

```
$ curl http://<example_app_endpoint>/metrics
```

Example output

```
# HELP http_requests_total Count of all HTTP requests
# TYPE http_requests_total counter
http_requests_total{code="200",method="get"} 4
http_requests_total{code="404",method="get"} 2
# HELP version Version information about this binary
# TYPE version gauge
version{version="v0.1.0"} 1
```

Additional resources

- Configuring metrics for user workload monitoring
- Accessing metrics as an administrator
- Accessing metrics as a developer

1.3.3.1. Controlling the impact of unbound metrics attributes in user-defined projects

Developers can create labels to define attributes for metrics in the form of key-value pairs. The number of potential key-value pairs corresponds to the number of possible values for an attribute. An attribute that has an unlimited number of potential values is called an unbound attribute. For example, a **customer_id** attribute is unbound because it has an infinite number of possible values.

Every assigned key-value pair has a unique time series. The use of many unbound attributes in labels can result in an exponential increase in the number of time series created. This can impact Prometheus performance and can consume a lot of disk space.

A **dedicated-admin** can use the following measures to control the impact of unbound metrics attributes in user-defined projects:

- Limit the number of samples that can be accepted per target scrape in user-defined projects
- Limit the number of scraped labels, the length of label names, and the length of label values
- Configure the intervals between consecutive scrapes and between Prometheus rule evaluations
- Create alerts that fire when a scrape sample threshold is reached or when the target cannot be scraped

NOTE

Limiting scrape samples can help prevent the issues caused by adding many unbound attributes to labels. Developers can also prevent the underlying cause by limiting the number of unbound attributes that they define for metrics. Using attributes that are bound to a limited set of possible values reduces the number of potential key-value pair combinations.

1.3.3.2. Adding cluster ID labels to metrics

If you manage multiple OpenShift Dedicated clusters and use the remote write feature to send metrics data from these clusters to an external storage location, you can add cluster ID labels to identify the metrics data coming from different clusters. You can then query these labels to identify the source cluster for a metric and distinguish that data from similar metrics data sent by other clusters.

This way, if you manage many clusters for multiple customers and send metrics data to a single centralized storage system, you can use cluster ID labels to query metrics for a particular cluster or customer.

Creating and using cluster ID labels involves three general steps:

- Configuring the write relabel settings for remote write storage.
- Adding cluster ID labels to the metrics.

- Querying these labels to identify the source cluster or customer for a metric.

1.3.4. About monitoring dashboards

OpenShift Dedicated provides a set of monitoring dashboards that help you understand the state of cluster components and user-defined workloads.

IMPORTANT

Starting with OpenShift Dedicated 4.19, the perspectives in the web console have unified. The **Developer** perspective is no longer enabled by default.

All users can interact with all OpenShift Dedicated web console features. However, if you are not the cluster owner, you might need to request permission to access certain features from the cluster owner.

You can still enable the **Developer** perspective. On the **Getting Started** pane in the web console, you can take a tour of the console, find information on setting up your cluster, view a quick start for enabling the **Developer** perspective, and follow links to explore new features and capabilities.

As an administrator, you can access dashboards for the core OpenShift Dedicated components, including the following items:

- API performance
- etcd
- Kubernetes compute resources
- Kubernetes network resources
- Prometheus
- USE method dashboards relating to cluster and node performance
- Node performance metrics

Additional resources

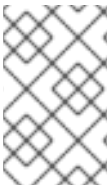
- Reviewing monitoring dashboards as a cluster administrator
- Reviewing monitoring dashboards as a developer

1.3.5. Managing alerts

In the OpenShift Dedicated, the Alerting UI enables you to manage alerts, silences, and alerting rules.

- **Alerting rules**. Alerting rules contain a set of conditions that outline a particular state within a cluster. Alerts are triggered when those conditions are true. An alerting rule can be assigned a severity that defines how the alerts are routed.
- **Alerts**. An alert is fired when the conditions defined in an alerting rule are true. Alerts provide a notification that a set of circumstances are apparent within your OpenShift Dedicated cluster.

- **Silences**. A silence can be applied to an alert to prevent notifications from being sent when the conditions for an alert are true. You can mute an alert after the initial notification, while you work on resolving the issue.



NOTE

The alerts, silences, and alerting rules that are available in the Alerting UI relate to the projects that you have access to. For example, if you are logged in as a user with the **cluster-admin** role, you can access all alerts, silences, and alerting rules.

Additional resources

- Configuring alerts and notifications for user workload monitoring
- Managing alerts as an Administrator
- Managing alerts as a Developer

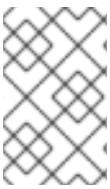
1.3.5.1. Managing silences

You can create a silence for an alert in the OpenShift Dedicated web console. After you create a silence, you will not receive notifications about an alert when the alert fires.

Creating silences is useful in scenarios where you have received an initial alert notification, and you do not want to receive further notifications during the time in which you resolve the underlying issue causing the alert to fire.

When creating a silence, you must specify whether it becomes active immediately or at a later time. You must also set a duration period after which the silence expires.

After you create silences, you can view, edit, and expire them.



NOTE

When you create silences, they are replicated across Alertmanager pods. However, if you do not configure persistent storage for Alertmanager, silences might be lost. This can happen, for example, if all Alertmanager pods restart at the same time.

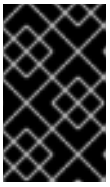
1.3.5.2. Creating alerting rules for user-defined projects

In OpenShift Dedicated, you can create alerting rules for user-defined projects. Those alerting rules will trigger alerts based on the values of the chosen metrics.

If you create alerting rules for a user-defined project, consider the following key behaviors and important limitations when you define the new rules:

- A user-defined alerting rule can include metrics exposed by its own project in addition to the default metrics from core platform monitoring. You cannot include metrics from another user-defined project.
  For example, an alerting rule for the **ns1** user-defined project can use metrics exposed by the **ns1** project in addition to core platform metrics, such as CPU and memory metrics. However, the rule cannot include metrics from a different **ns2** user-defined project.

- By default, when you create an alerting rule, the **namespace** label is enforced on it even if a rule with the same name exists in another project. To create alerting rules that are not bound to their project of origin, see "Creating cross-project alerting rules for user-defined projects".
- To reduce latency and to minimize the load on core platform monitoring components, you can add the **openshift.io/prometheus-rule-evaluation-scope: leaf-prometheus** label to a rule. This label forces only the Prometheus instance deployed in the **openshift-user-workload-monitoring** project to evaluate the alerting rule and prevents the Thanos Ruler instance from doing so.

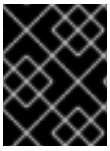


IMPORTANT

If an alerting rule has this label, your alerting rule can use only those metrics exposed by your user-defined project. Alerting rules you create based on default platform metrics might not trigger alerts.

1.3.5.3. Managing alerting rules for user-defined projects

In OpenShift Dedicated, you can view, edit, and remove alerting rules in user-defined projects.



IMPORTANT

Managing alerting rules for user-defined projects is only available in OpenShift Dedicated version 4.11 and later.

Alerting rule considerations

- The default alerting rules are used specifically for the OpenShift Dedicated cluster.
- Some alerting rules intentionally have identical names. They send alerts about the same event with different thresholds, different severity, or both.
- Inhibition rules prevent notifications for lower severity alerts that are firing when a higher severity alert is also firing.

1.3.5.4. Optimizing alerting for user-defined projects

You can optimize alerting for your own projects by considering the following recommendations when creating alerting rules:

- **Minimize the number of alerting rules that you create for your project**. Create alerting rules that notify you of conditions that impact you. It is more difficult to notice relevant alerts if you generate many alerts for conditions that do not impact you.
- **Create alerting rules for symptoms instead of causes**. Create alerting rules that notify you of conditions regardless of the underlying cause. The cause can then be investigated. You will need many more alerting rules if each relates only to a specific cause. Some causes are then likely to be missed.
- **Plan before you write your alerting rules**. Determine what symptoms are important to you and what actions you want to take if they occur. Then build an alerting rule for each symptom.
- **Provide clear alert messaging**. State the symptom and recommended actions in the alert message.

- **Include severity levels in your alerting rules** The severity of an alert depends on how you need to react if the reported symptom occurs. For example, a critical alert should be triggered if a symptom requires immediate attention by an individual or a critical response team.

1.3.5.5. Searching and filtering alerts, silences, and alerting rules

You can filter the alerts, silences, and alerting rules that are displayed in the Alerting UI. This section provides a description of each of the available filtering options.

1.3.5.5.1. Understanding alert filters

The **Alerts** page in the Alerting UI provides details about alerts relating to default OpenShift Dedicated and user-defined projects. The page includes a summary of severity, state, and source for each alert. The time at which an alert went into its current state is also shown.

You can filter by alert state, severity, and source. By default, only **Platform** alerts that are **Firing** are displayed. The following describes each alert filtering option:

- **State** filters:
  - **Firing**. The alert is firing because the alert condition is true and the optional **for** duration has passed. The alert continues to fire while the condition remains true.
  - **Pending**. The alert is active but is waiting for the duration that is specified in the alerting rule before it fires.
  - **Silenced**. The alert is now silenced for a defined time period. Silences temporarily mute alerts based on a set of label selectors that you define. Notifications are not sent for alerts that match all the listed values or regular expressions.
- **Severity** filters:
  - **Critical**. The condition that triggered the alert could have a critical impact. The alert requires immediate attention when fired and is typically paged to an individual or to a critical response team.
  - **Warning**. The alert provides a warning notification about something that might require attention to prevent a problem from occurring. Warnings are typically routed to a ticketing system for non-immediate review.
  - **Info**. The alert is provided for informational purposes only.
  - **None**. The alert has no defined severity.
  - You can also create custom severity definitions for alerts relating to user-defined projects.
- **Source** filters:
  - **Platform**. Platform-level alerts relate only to default OpenShift Dedicated projects. These projects provide core OpenShift Dedicated functionality.
  - **User**. User alerts relate to user-defined projects. These alerts are user-created and are customizable. User-defined workload monitoring can be enabled postinstallation to provide observability into your own workloads.

1.3.5.5.2. Understanding silence filters

The **Silences** page in the Alerting UI provides details about silences applied to alerts in default OpenShift Dedicated and user-defined projects. The page includes a summary of the state of each silence and the time at which a silence ends.

You can filter by silence state. By default, only **Active** and **Pending** silences are displayed. The following describes each silence state filter option:

- **State** filters:
    - **Active**. The silence is active and the alert will be muted until the silence is expired.
    - **Pending**. The silence has been scheduled and it is not yet active.
    - **Expired**. The silence has expired and notifications will be sent if the conditions for an alert are true.

1.3.5.5.3. Understanding alerting rule filters

The **Alerting rules** page in the Alerting UI provides details about alerting rules relating to default OpenShift Dedicated and user-defined projects. The page includes a summary of the state, severity, and source for each alerting rule.

You can filter alerting rules by alert state, severity, and source. By default, only **Platform** alerting rules are displayed. The following describes each alerting rule filtering option:

- **Alert state** filters:
    - **Firing**. The alert is firing because the alert condition is true and the optional **for** duration has passed. The alert continues to fire while the condition remains true.
    - **Pending**. The alert is active but is waiting for the duration that is specified in the alerting rule before it fires.
    - **Silenced**. The alert is now silenced for a defined time period. Silences temporarily mute alerts based on a set of label selectors that you define. Notifications are not sent for alerts that match all the listed values or regular expressions.
    - **Not Firing**. The alert is not firing.
- **Severity** filters:
    - **Critical**. The conditions defined in the alerting rule could have a critical impact. When true, these conditions require immediate attention. Alerts relating to the rule are typically paged to an individual or to a critical response team.
    - **Warning**. The conditions defined in the alerting rule might require attention to prevent a problem from occurring. Alerts relating to the rule are typically routed to a ticketing system for non-immediate review.
    - **Info**. The alerting rule provides informational alerts only.
    - **None**. The alerting rule has no defined severity.
    - You can also create custom severity definitions for alerting rules relating to user-defined projects.
- **Source** filters:

- **Platform**. Platform-level alerting rules relate only to default OpenShift Dedicated projects. These projects provide core OpenShift Dedicated functionality.
- **User**. User-defined workload alerting rules relate to user-defined projects. These alerting rules are user-created and are customizable. User-defined workload monitoring can be enabled postinstallation to provide observability into your own workloads.

1.3.6. Understanding alert routing for user-defined projects

As a **dedicated-admin**, you can enable alert routing for user-defined projects. With this feature, you can allow users with the **alert-routing-edit** cluster role to configure alert notification routing and receivers for user-defined projects. These notifications are routed by an Alertmanager instance dedicated to user-defined monitoring.

Users can then create and configure user-defined alert routing by creating or editing the **AlertmanagerConfig** objects for their user-defined projects without the help of an administrator.

After a user has defined alert routing for a user-defined project, user-defined alert notifications are routed to the **alertmanager-user-workload** pods in the **openshift-user-workload-monitoring** namespace.

NOTE

Review the following limitations of alert routing for user-defined projects:

- For user-defined alerting rules, user-defined routing is scoped to the namespace in which the resource is defined. For example, a routing configuration in namespace **ns1** only applies to **PrometheusRules** resources in the same namespace.
- When a namespace is excluded from user-defined monitoring, **AlertmanagerConfig** resources in the namespace cease to be part of the Alertmanager configuration.

Additional resources

- Enabling alert routing for user-defined projects

1.3.7. Sending notifications to external systems

In OpenShift Dedicated , firing alerts can be viewed in the Alerting UI. Alerts are not configured by default to be sent to any notification systems. You can configure OpenShift Dedicated to send alerts to the following receiver types:

- PagerDuty
- Webhook
- Email
- Slack
- Microsoft Teams

Routing alerts to receivers enables you to send timely notifications to the appropriate teams when

failures occur. For example, critical alerts require immediate attention and are typically paged to an individual or a critical response team. Alerts that provide non-critical warning notifications might instead be routed to a ticketing system for non-immediate review.

Checking that alerting is operational by using the watchdog alert

OpenShift Dedicated monitoring includes a watchdog alert that fires continuously. Alertmanager repeatedly sends watchdog alert notifications to configured notification providers. The provider is usually configured to notify an administrator when it stops receiving the watchdog alert. This mechanism helps you quickly identify any communication issues between Alertmanager and the notification provider.

Additional resources

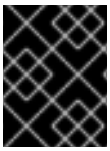
- Configuring alert notifications for user workload monitoring

CHAPTER 2. GETTING STARTED

2.1. MAINTENANCE AND SUPPORT FOR MONITORING

Not all configuration options for the monitoring stack are exposed. The only supported way of configuring OpenShift Dedicated monitoring is by configuring the Cluster Monitoring Operator (CMO) using the options described in the Config map reference for the Cluster Monitoring Operator . **Do not use other configurations, as they are unsupported.**

Configuration paradigms might change across Prometheus releases, and such cases can only be handled gracefully if all configuration possibilities are controlled. If you use configurations other than those described in the Config map reference for the Cluster Monitoring Operator , your changes will disappear because the CMO automatically reconciles any differences and resets any unsupported changes back to the originally defined state by default and by design.



IMPORTANT

Installing another Prometheus instance is not supported by the Red Hat Site Reliability Engineers (SRE).

2.1.1. Support considerations for monitoring

NOTE

Backward compatibility for metrics, recording rules, or alerting rules is not guaranteed.

The following modifications are explicitly not supported:

- **Installing custom Prometheus instances on OpenShift Dedicated.** A custom instance is a Prometheus custom resource (CR) managed by the Prometheus Operator.
- **Modifying the default platform monitoring components.** You should not modify any of the components defined in the **cluster-monitoring-config** config map. Red Hat SRE uses these components to monitor the core cluster components and Kubernetes services.

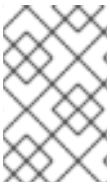
2.1.2. Support version matrix for monitoring components

The following matrix contains information about versions of monitoring components for OpenShift Dedicated 4.12 and later releases:

Table 2.1. OpenShift Dedicated and component versions

OpenShift Dedicated	Prometheus Operator	Prometheus	Metrics Server	Alertmanager	kube-state-metrics agent	monitoring-plugin	node-exporter agent	Thanos
4.20	0.85.0	3.5.0	0.8.0	0.28.1	2.16.0	1.0.0	1.9.1	0.39.2
4.19	0.81.0	3.2.1	0.7.2	0.28.1	2.15.0	1.0.0	1.9.1	0.37.2

OpenShift Dedicated	Prometheus Operator	Prometheus	Metrics Server	Alertmanager	kube-state-metrics agent	monitoring-plugin	node-exporter agent	Thanos
4.18	0.78.1	2.55.1	0.7.2	0.27.0	2.13.0	1.0.0	1.8.2	0.36.1
4.17	0.75.2	2.53.1	0.7.1	0.27.0	2.13.0	1.0.0	1.8.2	0.35.1
4.16	0.73.2	2.52.0	0.7.1	0.26.0	2.12.0	1.0.0	1.8.0	0.35.0
4.15	0.70.0	2.48.0	0.6.4	0.26.0	2.10.1	1.0.0	1.7.0	0.32.5
4.14	0.67.1	2.46.0	N/A	0.25.0	2.9.2	1.0.0	1.6.1	0.30.2
4.13	0.63.0	2.42.0	N/A	0.25.0	2.8.1	N/A	1.5.0	0.30.2
4.12	0.60.1	2.39.1	N/A	0.24.0	2.6.0	N/A	1.4.0	0.28.1



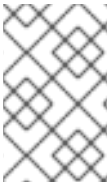
NOTE

The openshift-state-metrics agent and Telemeter Client are OpenShift-specific components. Therefore, their versions correspond with the versions of OpenShift Dedicated.

2.2. ACCESSING MONITORING FOR USER-DEFINED PROJECTS

When you install a OpenShift Dedicated cluster, monitoring for user-defined projects is enabled by default. With monitoring for user-defined projects enabled, you can monitor your own OpenShift Dedicated projects without the need for an additional monitoring solution.

The **dedicated-admin** user has default permissions to configure and access monitoring for user-defined projects.



NOTE

Custom Prometheus instances and the Prometheus Operator installed through Operator Lifecycle Manager (OLM) can cause issues with user-defined project monitoring if it is enabled. Custom Prometheus instances are not supported.

Optionally, you can disable monitoring for user-defined projects during or after a cluster installation.

2.3. DISABLING MONITORING FOR USER-DEFINED PROJECTS

As a **dedicated-admin**, you can disable monitoring for user-defined projects. You can also exclude individual projects from user workload monitoring.

2.3.1. Disabling monitoring for user-defined projects

By default, monitoring for user-defined projects is enabled. If you do not want to use the built-in monitoring stack to monitor user-defined projects, you can disable it.

Prerequisites

- You logged in to OpenShift Cluster Manager.

Procedure

1. From the OpenShift Cluster Manager Hybrid Cloud Console, select a cluster.
2. Click the **Settings** tab.
3. Click the **Enable user workload monitoring** check box to unselect the option, and then click **Save**.
   User workload monitoring is disabled. The Prometheus, Prometheus Operator, and Thanos Ruler components are stopped in the **openshift-user-workload-monitoring** project.

2.3.2. Excluding a user-defined project from monitoring

Individual user-defined projects can be excluded from user workload monitoring. To do so, add the **openshift.io/user-monitoring** label to the project's namespace with a value of **false**.

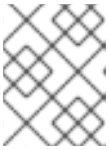
Procedure

1. Add the label to the project namespace:

```
$ oc label namespace my-project 'openshift.io/user-monitoring=false'
```

2. To re-enable monitoring, remove the label from the namespace:

```
$ oc label namespace my-project 'openshift.io/user-monitoring-'
```



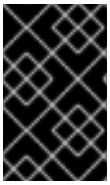
NOTE

If there were any active monitoring targets for the project, it may take a few minutes for Prometheus to stop scraping them after adding the label.

CHAPTER 3. CONFIGURING USER WORKLOAD MONITORING

3.1. PREPARING TO CONFIGURE THE USER WORKLOAD MONITORING STACK

This section explains which user-defined monitoring components can be configured and how to prepare for configuring the user workload monitoring stack.



IMPORTANT

- Not all configuration parameters for the monitoring stack are exposed. Only the parameters and fields listed in the Config map reference for the Cluster Monitoring Operator are supported for configuration.

3.1.1. Configurable monitoring components

This table shows the monitoring components you can configure and the keys used to specify the components in the **user-workload-monitoring-config** config map.

WARNING

Do not modify the monitoring components in the **cluster-monitoring-config ConfigMap** object. Red Hat Site Reliability Engineers (SRE) use these components to monitor the core cluster components and Kubernetes services.

Table 3.1. Configurable monitoring components for user-defined projects

Component	user-workload-monitoring-config config map key
Prometheus Operator	**prometheusOperator**
Prometheus	**prometheus**
Alertmanager	**alertmanager**
Thanos Ruler	**thanosRuler**

3.1.2. Enabling alert routing for user-defined projects

In OpenShift Dedicated, an administrator can enable alert routing for user-defined projects. This process consists of the following steps:

- Enable alert routing for user-defined projects to use a separate Alertmanager instance.
- Grant users permission to configure alert routing for user-defined projects.

After you complete these steps, developers and other users can configure custom alerts and alert routing for their user-defined projects.

Additional resources

- Understanding alert routing for user-defined projects

3.1.2.1. Enabling a separate Alertmanager instance for user-defined alert routing

In OpenShift Dedicated, you may want to deploy a dedicated Alertmanager instance for user-defined projects, which provides user-defined alerts separate from default platform alerts. In these cases, you can optionally enable a separate instance of Alertmanager to send alerts for user-defined projects only.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config ConfigMap** object:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add **enabled: true** and **enableAlertmanagerConfig: true** in the **alertmanager** section under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    alertmanager:
      enabled: true 1
      enableAlertmanagerConfig: true 2
```

1. Set the **enabled** value to **true** to enable a dedicated instance of the Alertmanager for user-defined projects in a cluster. Set the value to **false** or omit the key entirely to disable the Alertmanager for user-defined projects. If you set this value to **false** or if the key is omitted, user-defined alerts are routed to the default platform Alertmanager instance.

2. Set the **enableAlertmanagerConfig** value to **true** to enable users to define their own alert routing configurations with **AlertmanagerConfig** objects.

3. Save the file to apply the changes. The dedicated instance of Alertmanager for user-defined projects starts automatically.

Verification

- Verify that the **alert-manager-user-workload** pods are running:

```
$ oc -n openshift-user-workload-monitoring get pods
```

Example output

```
NAME                                  READY   STATUS    RESTARTS   AGE
alertmanager-user-workload-0          6/6     Running   0          38s
alertmanager-user-workload-1          6/6     Running   0          38s
...
```

3.1.2.2. Granting users permission to configure alert routing for user-defined projects

You can grant users permission to configure alert routing for user-defined projects.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- The user account that you are assigning the role to already exists.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Assign the **alert-routing-edit** cluster role to a user in the user-defined project:

```
$ oc -n <namespace> adm policy add-role-to-user alert-routing-edit <user> 1
```

1 For **<namespace>**, substitute the namespace for the user-defined project, such as **ns1**. For **<user>**, substitute the username for the account to which you want to assign the role.

Additional resources

- Configuring alert notifications

3.2. CONFIGURING PERFORMANCE AND SCALABILITY FOR USER WORKLOAD MONITORING

You can configure the monitoring stack to optimize the performance and scale of your clusters. The following documentation provides information about how to distribute the monitoring components and control the impact of the monitoring stack on CPU and memory resources.

3.2.1. Controlling the placement and distribution of monitoring components

You can move the monitoring stack components to specific nodes:

- Use the **nodeSelector** constraint with labeled nodes to move any of the monitoring stack components to specific nodes.
- Assign tolerations to enable moving components to tainted nodes.

By doing so, you control the placement and distribution of the monitoring components across a cluster.

By controlling placement and distribution of monitoring components, you can optimize system resource use, improve performance, and separate workloads based on specific requirements or policies.

Additional resources

- Using node selectors to move monitoring components

3.2.1.1. Moving monitoring components to different nodes

You can move any of the components that monitor workloads for user-defined projects to specific worker nodes.

WARNING

It is not permitted to move components to control plane or infrastructure nodes.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. If you have not done so yet, add a label to the nodes on which you want to run the monitoring components:

```
$ oc label nodes <node_name> <node_label> 1
```

1 Replace **<node_name>** with the name of the node where you want to add the label. Replace **<node_label>** with the name of the wanted label.

2. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

3. Specify the node labels for the **nodeSelector** constraint for the component under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    # ...
    <component>: 1
      nodeSelector:
        <node_label_1> 2
        <node_label_2> 3
    # ...
```

1. Substitute **<component>** with the appropriate monitoring stack component name.
2. Substitute **<node_label_1>** with the label you added to the node.
3. Optional: Specify additional labels. If you specify additional labels, the pods for the component are only scheduled on the nodes that contain all of the specified labels.

NOTE

If monitoring components remain in a **Pending** state after configuring the **nodeSelector** constraint, check the pod events for errors relating to taints and tolerations.

4. Save the file to apply the changes. The components specified in the new configuration are automatically moved to the new nodes, and the pods affected by the new configuration are redeployed.

Additional resources

- **nodeSelector** (Kubernetes documentation)

3.2.1.2. Assigning tolerations to monitoring components

You can assign tolerations to the components that monitor user-defined projects, to enable moving them to tainted worker nodes. Scheduling is not permitted on control plane or infrastructure nodes.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists in the **openshift-user-workload-monitoring** namespace. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Specify **tolerations** for the component:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>:
      tolerations:
        <toleration_specification>
```

Substitute **<component>** and **<toleration_specification>** accordingly.

For example, **oc adm taint nodes node1 key1=value1:NoSchedule** adds a taint to **node1** with the key **key1** and the value **value1**. This prevents monitoring components from deploying pods on **node1** unless a toleration is configured for that taint. The following example configures the **thanosRuler** component to tolerate the example taint:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      tolerations:
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoSchedule"
```

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

Additional resources

- Taints and tolerations (Kubernetes documentation)

3.2.2. Managing CPU and memory resources for monitoring components

You can ensure that the containers that run monitoring components have enough CPU and memory resources by specifying values for resource limits and requests for those components.

You can configure these limits and requests for monitoring components that monitor user-defined projects in the **openshift-user-workload-monitoring** namespace.

3.2.2.1. Specifying limits and requests

To configure CPU and memory resources, specify values for resource limits and requests in the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** namespace.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role, or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add values to define resource limits and requests for each component you want to configure.

IMPORTANT

Ensure that the value set for a limit is always higher than the value set for a request. Otherwise, an error will occur, and the container will not run.

Example of setting resource limits and requests

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    alertmanager:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    prometheus:
      resources:
        limits:
          cpu: 500m
          memory: 3Gi
        requests:
```

```
      cpu: 200m
      memory: 500Mi
  thanosRuler:
    resources:
      limits:
        cpu: 500m
        memory: 1Gi
      requests:
        cpu: 200m
        memory: 500Mi
```

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

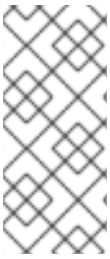
Additional resources

- About specifying limits and requests for monitoring components
- Requests and limits (Kubernetes documentation)

3.2.3. Controlling the impact of unbound metrics attributes in user-defined projects

A **dedicated-admin** can use the following measures to control the impact of unbound metrics attributes in user-defined projects:

- Limit the number of samples that can be accepted per target scrape in user-defined projects
- Limit the number of scraped labels, the length of label names, and the length of label values
- Configure the intervals between consecutive scrapes and between Prometheus rule evaluations



NOTE

Limiting scrape samples can help prevent the issues caused by adding many unbound attributes to labels. Developers can also prevent the underlying cause by limiting the number of unbound attributes that they define for metrics. Using attributes that are bound to a limited set of possible values reduces the number of potential key-value pair combinations.

Additional resources

- Controlling the impact of unbound metrics attributes in user-defined projects
- Determining why Prometheus is consuming a lot of disk space

3.2.3.1. Setting scrape intervals, evaluation intervals, and enforced limits for user-defined projects

You can set the following scrape and label limits for user-defined projects:

- Limit the number of samples that can be accepted per target scrape
- Limit the number of scraped labels
- Limit the length of label names and label values

You can also set an interval between consecutive scrapes and between Prometheus rule evaluations.

WARNING

If you set sample or label limits, no further sample data is ingested for that target scrape after the limit is reached.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add the enforced limit and time interval configurations to **data/config.yaml**:

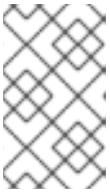
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      enforcedSampleLimit: 50000 1
      enforcedLabelLimit: 500 2
      enforcedLabelNameLengthLimit: 50 3
      enforcedLabelValueLengthLimit: 600 4
      scrapeInterval: 1m30s 5
      evaluationInterval: 1m15s 6
```

1. A value is required if this parameter is specified. This **enforcedSampleLimit** example limits the number of samples that can be accepted per target scrape in user-defined projects to 50,000.
2. Specifies the maximum number of labels per scrape. The default value is **0**, which specifies no limit.
3. Specifies the maximum character length for a label name. The default value is **0**, which specifies no limit.

4 Specifies the maximum character length for a label value. The default value is **0**, which specifies no limit.

5 Specifies the interval between consecutive scrapes. The interval must be set between 5 seconds and 5 minutes. The default value is **30s**.

6 Specifies the interval between Prometheus rule evaluations. The interval must be set between 5 seconds and 5 minutes. The default value for Prometheus is **30s**.



NOTE

You can also configure the **evaluationInterval** property for Thanos Ruler through the **data/config.yaml/thanosRuler** field. The default value for Thanos Ruler is **15s**.

3. Save the file to apply the changes. The limits are applied automatically.

3.2.4. Configuring pod topology spread constraints

You can configure pod topology spread constraints for all the pods for user-defined monitoring to control how pod replicas are scheduled to nodes across zones. This ensures that the pods are highly available and run more efficiently, because workloads are spread across nodes in different data centers or hierarchical infrastructure zones.

You can configure pod topology spread constraints for monitoring pods by using the **user-workload-monitoring-config** config map.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add the following settings under the **data/config.yaml** field to configure pod topology spread constraints:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

```
<component>: 1
  topologySpreadConstraints:
  - maxSkew: <n> 2
    topologyKey: <key> 3
    whenUnsatisfiable: <value> 4
    labelSelector: 5
      <match_option>
```

1. Specify a name of the component for which you want to set up pod topology spread constraints.
2. Specify a numeric value for **maxSkew**, which defines the degree to which pods are allowed to be unevenly distributed.
3. Specify a key of node labels for **topologyKey**. Nodes that have a label with this key and identical values are considered to be in the same topology. The scheduler tries to put a balanced number of pods into each domain.
4. Specify a value for **whenUnsatisfiable**. Available options are **DoNotSchedule** and **ScheduleAnyway**. Specify **DoNotSchedule** if you want the **maxSkew** value to define the maximum difference allowed between the number of matching pods in the target topology and the global minimum. Specify **ScheduleAnyway** if you want the scheduler to still schedule the pod but to give higher priority to nodes that might reduce the skew.
5. Specify **labelSelector** to find matching pods. Pods that match this label selector are counted to determine the number of pods in their corresponding topology domain.

Example configuration for Thanos Ruler

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      topologySpreadConstraints:
      - maxSkew: 1
        topologyKey: monitoring
        whenUnsatisfiable: ScheduleAnyway
        labelSelector:
          matchLabels:
            app.kubernetes.io/name: thanos-ruler
```

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

Additional resources

- About pod topology spread constraints for monitoring
- Pod topology spread constraints (Kubernetes documentation)

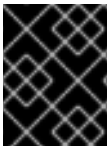
3.3. STORING AND RECORDING DATA FOR USER WORKLOAD MONITORING

Store and record your metrics and alerting data, configure logs to specify which activities are recorded, control how long Prometheus retains stored data, and set the maximum amount of disk space for the data. These actions help you protect your data and use them for troubleshooting.

3.3.1. Configuring persistent storage

Run cluster monitoring with persistent storage to gain the following benefits:

- Protect your metrics and alerting data from data loss by storing them in a persistent volume (PV). As a result, they can survive pods being restarted or recreated.
- Avoid getting duplicate notifications and losing silences for alerts when the Alertmanager pods are restarted.



IMPORTANT

In multi-node clusters, you must configure persistent storage for Prometheus, Alertmanager, and Thanos Ruler to ensure high availability.

NOTE

For production environments, it is highly recommended to configure persistent storage.

3.3.1.1. Persistent storage prerequisites

- Use the block type of storage.

3.3.1.2. Configuring a persistent volume claim

To use a persistent volume (PV) for monitoring components, you must configure a persistent volume claim (PVC).

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add your PVC configuration for the component under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>: 1
      volumeClaimTemplate:
        spec:
          storageClassName: <storage_class> 2
          resources:
            requests:
              storage: <amount_of_storage> 3
```

1. Specify the monitoring component for which you want to configure the PVC.
2. Specify an existing storage class. If a storage class is not specified, the default storage class is used.
3. Specify the amount of required storage.

The following example configures a PVC that claims persistent storage for Thanos Ruler:

Example PVC configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      volumeClaimTemplate:
        spec:
          storageClassName: my-storage-class
          resources:
            requests:
              storage: 10Gi
```

NOTE

Storage requirements for the **thanosRuler** component depend on the number of rules that are evaluated and how many samples each rule generates.

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed and the new storage configuration is applied.

WARNING

When you update the config map with a PVC configuration, the affected **StatefulSet** object is recreated, resulting in a temporary service outage.

Additional resources

- Understanding persistent storage
- PersistentVolumeClaims (Kubernetes documentation)

3.3.2. Modifying retention time and size for Prometheus metrics data

By default, Prometheus retains metrics data for 24 hours for monitoring for user-defined projects. You can modify the retention time for the Prometheus instance to change when the data is deleted. You can also set the maximum amount of disk space the retained metrics data uses.

NOTE

Data compaction occurs every two hours. Therefore, a persistent volume (PV) might fill up before compaction, potentially exceeding the **retentionSize** limit. In such cases, the **KubePersistentVolumeFillingUp** alert fires until the space on a PV is lower than the **retentionSize** limit.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add the retention time and size configuration under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

```
prometheus:
  retention: <time_specification> 1
  retentionSize: <size_specification> 2
```

1. The retention time: a number directly followed by **ms** (milliseconds), **s** (seconds), **m** (minutes), **h** (hours), **d** (days), **w** (weeks), or **y** (years). You can also combine time values for specific times, such as **1h30m15s**.
2. The retention size: a number directly followed by **B** (bytes), **KB** (kilobytes), **MB** (megabytes), **GB** (gigabytes), **TB** (terabytes), **PB** (petabytes), and **EB** (exabytes).

The following example sets the retention time to 24 hours and the retention size to 10 gigabytes for the Prometheus instance:

Example of setting retention time for Prometheus

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      retention: 24h
      retentionSize: 10GB
```

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

3.3.2.1. Modifying the retention time for Thanos Ruler metrics data

By default, for user-defined projects, Thanos Ruler automatically retains metrics data for 24 hours. You can modify the retention time to change how long this data is retained by specifying a time value in the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** namespace.

NOTE

If you configure retention for user-workload Prometheus, Thanos Ruler automatically inherits the same retention time, unless explicitly configured otherwise.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add the retention time configuration under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      retention: <time_specification> 1
```

1 Specify the retention time in the following format: a number directly followed by **ms** (milliseconds), **s** (seconds), **m** (minutes), **h** (hours), **d** (days), **w** (weeks), or **y** (years). You can also combine time values for specific times, such as **1h30m15s**. The default is **24h**.

The following example sets the retention time to 10 days for Thanos Ruler data:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      retention: 10d
```

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

Additional resources

- Retention time and size for Prometheus metrics
- Understanding persistent storage

3.3.3. Setting log levels for monitoring components

You can configure the log level for Alertmanager, Prometheus Operator, Prometheus, and Thanos Ruler. You can use these settings for troubleshooting and to gain better insight into how the components are functioning.

The following log levels can be applied to the relevant component in the **user-workload-monitoring-config ConfigMap** object:

- **debug**. Log debug, informational, warning, and error messages.

- **info** (default). Log informational, warning, and error messages.
- **warn**. Log warning and error messages only.
- **error**. Log error messages only.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add log configuration for a component under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>: 1
      logLevel: <log_level> 2
    # ...
```

1. Specify the monitoring stack component for which you are setting a log level. Available component values are **prometheus**, **alertmanager**, **prometheusOperator**, and **thanosRuler**.
2. Specify the log level for the component. The available values are **error**, **warn**, **info**, and **debug**. The default value is **info**.

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.
4. Verify that the log configuration is applied by reviewing the deployment or pod configuration in the related project.
   - The following example checks the log level for the **prometheus-operator** deployment:

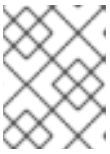
```
$ oc -n openshift-user-workload-monitoring get deploy prometheus-operator -o yaml |
grep "log-level"
```

Example output

```
- --log-level=debug
```

5. Verify that the pods for the component are running:

```
$ oc -n openshift-user-workload-monitoring get pods
```

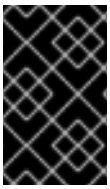


NOTE

If an unrecognized **logLevel** value is included in the **ConfigMap** object, the pods for the component might not restart successfully.

3.3.4. Enabling the query log file for Prometheus

You can configure Prometheus to write all queries that have been run by the engine to a log file.



IMPORTANT

Because log rotation is not supported, only enable this feature temporarily when you need to troubleshoot an issue. After you finish troubleshooting, disable query logging by reverting the changes you made to the **ConfigMap** object to enable the feature.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add the **queryLogFile** parameter for Prometheus under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      queryLogFile: <path> 1
```

1 Add the full path to the file in which queries will be logged.

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

4. Verify that the pods for the component are running. The following sample command lists the status of pods:

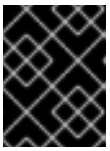
```
$ oc -n openshift-user-workload-monitoring get pods
```

Example output

```
...
prometheus-operator-776fcbbd56-2nbfm   2/2     Running   0          132m
prometheus-user-workload-0             5/5     Running   1          132m
prometheus-user-workload-1             5/5     Running   1          132m
thanos-ruler-user-workload-0           3/3     Running   0          132m
thanos-ruler-user-workload-1           3/3     Running   0          132m
...
```

5. Read the query log:

```
$ oc -n openshift-user-workload-monitoring exec prometheus-user-workload-0 -- cat <path>
```



IMPORTANT

Revert the setting in the config map after you have examined the logged query information.

3.4. CONFIGURING METRICS FOR USER WORKLOAD MONITORING

Configure the collection of metrics to monitor how cluster components and your own workloads are performing.

You can send ingested metrics to remote systems for long-term storage and add cluster ID labels to the metrics to identify the data coming from different clusters.

Additional resources

- Understanding metrics

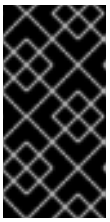
3.4.1. Configuring remote write storage

You can configure remote write storage to enable Prometheus to send ingested metrics to remote systems for long-term storage. Doing so has no impact on how or for how long Prometheus stores metrics.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

- You have set up a remote write compatible endpoint (such as Thanos) and know the endpoint URL. See the Prometheus remote endpoints and storage documentation for information about endpoints that are compatible with the remote write feature.



IMPORTANT

Red Hat only provides information for configuring remote write senders and does not offer guidance on configuring receiver endpoints. Customers are responsible for setting up their own endpoints that are remote-write compatible. Issues with endpoint receiver configurations are not included in Red Hat production support.

- You have set up authentication credentials in a **Secret** object for the remote write endpoint. You must create the secret in the **openshift-user-workload-monitoring** namespace.

WARNING

To reduce security risks, use HTTPS and authentication to send metrics to an endpoint.

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add a **remoteWrite:** section under **data/config.yaml/prometheus**, as shown in the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com" 1
        <endpoint_authentication_credentials> 2
```

1. The URL of the remote write endpoint.
2. The authentication method and credentials for the endpoint. Currently supported authentication methods are AWS Signature Version 4, authentication using HTTP in an **Authorization** request header, Basic authentication, OAuth 2.0, and TLS client. See *Supported remote write authentication settings* for sample configurations of supported authentication methods.

3. Add write relabel configuration values after the authentication credentials:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        <endpoint_authentication_credentials>
        writeRelabelConfigs:
        - <your_write_relabel_configs> 1
```

1 Add configuration for metrics that you want to send to the remote endpoint.

Example of forwarding a single metric called my_metric

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        writeRelabelConfigs:
        - sourceLabels: [__name__]
          regex: 'my_metric'
          action: keep
```

Example of forwarding metrics called my_metric_1 and my_metric_2 in my_namespace namespace

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        writeRelabelConfigs:
        - sourceLabels: [__name__,namespace]
          regex: '(my_metric_1|my_metric_2);my_namespace'
          action: keep
```

4. Save the file to apply the changes. The new configuration is applied automatically.

Additional resources

- **relabel_config** (Prometheus documentation)

3.4.1.1. Supported remote write authentication settings

You can use different methods to authenticate with a remote write endpoint. Currently supported authentication methods are AWS Signature Version 4, basic authentication, authorization, OAuth 2.0, and TLS client. The following table provides details about supported authentication methods for use with remote write.

Authentication method	Config map field	Description
AWS Signature Version 4	**sigv4**	This method uses AWS Signature Version 4 authentication to sign requests. You cannot use this method simultaneously with authorization, OAuth 2.0, or Basic authentication.
Basic authentication	**basicAuth**	Basic authentication sets the authorization header on every remote write request with the configured username and password.
authorization	**authorization**	Authorization sets the **Authorization** header on every remote write request using the configured token.
OAuth 2.0	**oauth2**	An OAuth 2.0 configuration uses the client credentials grant type. Prometheus fetches an access token from **tokenUrl** with the specified client ID and client secret to access the remote write endpoint. You cannot use this method simultaneously with authorization, AWS Signature Version 4, or Basic authentication.

Authentication method	Config map field	Description
TLS client	**tlsConfig**	A TLS client configuration specifies the CA certificate, the client certificate, and the client key file information used to authenticate with the remote write endpoint server using TLS. The sample configuration assumes that you have already created a CA certificate file, a client certificate file, and a client key file.

3.4.1.2. Example remote write authentication settings

The following samples show different authentication settings you can use to connect to a remote write endpoint. Each sample also shows how to configure a corresponding **Secret** object that contains authentication credentials and other relevant settings. Each sample configures authentication for use with monitoring for user-defined projects in the **openshift-user-workload-monitoring** namespace.

3.4.1.2.1. Sample YAML for AWS Signature Version 4 authentication

The following shows the settings for a **sigv4** secret named **sigv4-credentials** in the **openshift-user-workload-monitoring** namespace.

```
apiVersion: v1
kind: Secret
metadata:
  name: sigv4-credentials
  namespace: openshift-user-workload-monitoring
stringData:
  accessKey: <AWS_access_key> 1
  secretKey: <AWS_secret_key> 2
type: Opaque
```

1. The AWS API access key.
2. The AWS API secret key.

The following shows sample AWS Signature Version 4 remote write authentication settings that use a **Secret** object named **sigv4-credentials** in the **openshift-user-workload-monitoring** namespace:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

```
prometheus:
  remoteWrite:
  - url: "https://authorization.example.com/api/write"
    sigv4:
      region: <AWS_region> 1
      accessKey:
        name: sigv4-credentials 2
        key: accessKey 3
      secretKey:
        name: sigv4-credentials 4
        key: secretKey 5
      profile: <AWS_profile_name> 6
      roleArn: <AWS_role_arn> 7
```

1. The AWS region.
2. 4 The name of the **Secret** object containing the AWS API access credentials.
3. The key that contains the AWS API access key in the specified **Secret** object.
5. The key that contains the AWS API secret key in the specified **Secret** object.
6. The name of the AWS profile that is being used to authenticate.
7. The unique identifier for the Amazon Resource Name (ARN) assigned to your role.

3.4.1.2.2. Sample YAML for Basic authentication

The following shows sample Basic authentication settings for a **Secret** object named **rw-basic-auth** in the **openshift-user-workload-monitoring** namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: rw-basic-auth
  namespace: openshift-user-workload-monitoring
stringData:
  user: <basic_username> 1
  password: <basic_password> 2
type: Opaque
```

1. The username.
2. The password.

The following sample shows a **basicAuth** remote write configuration that uses a **Secret** object named **rw-basic-auth** in the **openshift-user-workload-monitoring** namespace. It assumes that you have already set up authentication credentials for the endpoint.

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://basicauth.example.com/api/write"
        basicAuth:
          username:
            name: rw-basic-auth (1)
            key: user (2)
          password:
            name: rw-basic-auth (3)
            key: password (4)
```

(1) (3) The name of the **Secret** object that contains the authentication credentials.

(2) The key that contains the username in the specified **Secret** object.

(4) The key that contains the password in the specified **Secret** object.

3.4.1.2.3. Sample YAML for authentication with a bearer token using a Secret Object

The following shows bearer token settings for a **Secret** object named **rw-bearer-auth** in the **openshift-user-workload-monitoring** namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: rw-bearer-auth
  namespace: openshift-user-workload-monitoring
stringData:
  token: <authentication_token> (1)
type: Opaque
```

(1) The authentication token.

The following shows sample bearer token config map settings that use a **Secret** object named **rw-bearer-auth** in the **openshift-user-workload-monitoring** namespace:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://authorization.example.com/api/write"
        authorization:
          type: Bearer (1)
```

```
      credentials:
        name: rw-bearer-auth 2
        key: token 3
```

1 The authentication type of the request. The default value is **Bearer**.

2 The name of the **Secret** object that contains the authentication credentials.

3 The key that contains the authentication token in the specified **Secret** object.

3.4.1.2.4. Sample YAML for OAuth 2.0 authentication

The following shows sample OAuth 2.0 settings for a **Secret** object named **oauth2-credentials** in the **openshift-user-workload-monitoring** namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: oauth2-credentials
  namespace: openshift-user-workload-monitoring
stringData:
  id: <oauth2_id> 1
  secret: <oauth2_secret> 2
type: Opaque
```

1 The Oauth 2.0 ID.

2 The OAuth 2.0 secret.

The following shows an **oauth2** remote write authentication sample configuration that uses a **Secret** object named **oauth2-credentials** in the **openshift-user-workload-monitoring** namespace:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://test.example.com/api/write"
        oauth2:
          clientId:
            secret:
              name: oauth2-credentials 1
              key: id 2
          clientSecret:
            name: oauth2-credentials 3
            key: secret 4
          tokenUrl: https://example.com/oauth2/token 5
          scopes: 6
```

```
      - <scope_1>
      - <scope_2>
      endpointParams: 7
        param1: <parameter_1>
        param2: <parameter_2>
```

1 3 The name of the corresponding **Secret** object. Note that **ClientId** can alternatively refer to a **ConfigMap** object, although **clientSecret** must refer to a **Secret** object.

2 4 The key that contains the OAuth 2.0 credentials in the specified **Secret** object.

5 The URL used to fetch a token with the specified **clientId** and **clientSecret**.

6 The OAuth 2.0 scopes for the authorization request. These scopes limit what data the tokens can access.

7 The OAuth 2.0 authorization request parameters required for the authorization server.

3.4.1.2.5. Sample YAML for TLS client authentication

The following shows sample TLS client settings for a **tls Secret** object named **mtls-bundle** in the **openshift-user-workload-monitoring** namespace.

```
apiVersion: v1
kind: Secret
metadata:
  name: mtls-bundle
  namespace: openshift-user-workload-monitoring
data:
  ca.crt: <ca_cert> 1
  client.crt: <client_cert> 2
  client.key: <client_key> 3
type: tls
```

1 The CA certificate in the Prometheus container with which to validate the server certificate.

2 The client certificate for authentication with the server.

3 The client key.

The following sample shows a **tlsConfig** remote write authentication configuration that uses a TLS **Secret** object named **mtls-bundle**.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
```

```
    tlsConfig:
      ca:
        secret:
          name: mtls-bundle (1)
          key: ca.crt (2)
      cert:
        secret:
          name: mtls-bundle (3)
          key: client.crt (4)
      keySecret:
        name: mtls-bundle (5)
        key: client.key (6)
```

(1) (3) (5) The name of the corresponding **Secret** object that contains the TLS authentication credentials. Note that **ca** and **cert** can alternatively refer to a **ConfigMap** object, though **keySecret** must refer to a **Secret** object.

(2) The key in the specified **Secret** object that contains the CA certificate for the endpoint.

(4) The key in the specified **Secret** object that contains the client certificate for the endpoint.

(6) The key in the specified **Secret** object that contains the client key secret.

3.4.1.3. Example remote write queue configuration

You can use the **queueConfig** object for remote write to tune the remote write queue parameters. The following example shows the queue parameters with their default values for monitoring for user-defined projects in the **openshift-user-workload-monitoring** namespace.

Example configuration of remote write parameters with default values

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        <endpoint_authentication_credentials>
        queueConfig:
          capacity: 10000 (1)
          minShards: 1 (2)
          maxShards: 50 (3)
          maxSamplesPerSend: 2000 (4)
          batchSendDeadline: 5s (5)
          minBackoff: 30ms (6)
          maxBackoff: 5s (7)
          retryOnRateLimit: false (8)
          sampleAgeLimit: 0s (9)
```

1. The number of samples to buffer per shard before they are dropped from the queue.
2. The minimum number of shards.
3. The maximum number of shards.
4. The maximum number of samples per send.
5. The maximum time for a sample to wait in buffer.
6. The initial time to wait before retrying a failed request. The time gets doubled for every retry up to the **maxbackoff** time.
7. The maximum time to wait before retrying a failed request.
8. Set this parameter to **true** to retry a request after receiving a 429 status code from the remote write storage.
9. The samples that are older than the **sampleAgeLimit** limit are dropped from the queue. If the value is undefined or set to **0s**, the parameter is ignored.

Additional resources

- Remote write compatible endpoints (Prometheus documentation)
- Remote write tuning (Prometheus documentation)
- Understanding secrets

3.4.1.4. Table of remote write metrics

The following table contains remote write and remote write-adjacent metrics with further description to help solve issues during remote write configuration.

Metric	Description
prometheus_remote_storage_highest_timestamp_in_seconds	Shows the newest timestamp that Prometheus stored in the write-ahead log (WAL) for any sample.
prometheus_remote_storage_queue_highest_sent_timestamp_seconds	Shows the newest timestamp that the remote write queue successfully sent.
prometheus_remote_storage_samples_retried_total	The number of samples that remote write failed to send and had to resend to remote storage. A steady high rate for this metric indicates problems with the network or remote storage endpoint.
prometheus_remote_storage_shards	Shows how many shards are currently running for each remote endpoint.

Metric	Description
prometheus_remote_storage_shards_desired	Shows the calculated needed number of shards based on the current write throughput and the rate of incoming versus sent samples.
prometheus_remote_storage_shards_max	Shows the maximum number of shards based on the current configuration.
prometheus_remote_storage_shards_min	Shows the minimum number of shards based on the current configuration.
prometheus_tsdb_wal_segment_current	The WAL segment file that Prometheus is currently writing new data to.
prometheus_wal_watcher_current_segment	The WAL segment file that each remote write instance is currently reading from.

3.4.2. Creating cluster ID labels for metrics

You can create cluster ID labels for metrics by adding the **write_relabel** settings for remote write storage in the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** namespace.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config** ConfigMap object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).
- You have configured remote write storage.

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. In the **writeRelabelConfigs:** section under **data/config.yaml/prometheus/remoteWrite**, add cluster ID relabel configuration values:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
```

```
config.yaml: |
  prometheus:
    remoteWrite:
    - url: "https://remote-write-endpoint.example.com"
      <endpoint_authentication_credentials>
      writeRelabelConfigs: (1)
        - <relabel_config> (2)
```

(1) Add a list of write relabel configurations for metrics that you want to send to the remote endpoint.

(2) Substitute the label configuration for the metrics sent to the remote write endpoint.

The following sample shows how to forward a metric with the cluster ID label **cluster_id**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
      - url: "https://remote-write-endpoint.example.com"
        writeRelabelConfigs:
        - sourceLabels:
          - __tmp_openshift_cluster_id__ (1)
          targetLabel: cluster_id (2)
          action: replace (3)
```

(1) The system initially applies a temporary cluster ID source label named **__tmp_openshift_cluster_id__**. This temporary label gets replaced by the cluster ID label name that you specify.

(2) Specify the name of the cluster ID label for metrics sent to remote write storage. If you use a label name that already exists for a metric, that value is overwritten with the name of this cluster ID label. For the label name, do not use **__tmp_openshift_cluster_id__**. The final relabeling step removes labels that use this name.

(3) The **replace** write relabel action replaces the temporary label with the target label for outgoing metrics. This action is the default and is applied if no action is specified.

3. Save the file to apply the changes. The new configuration is applied automatically.

Additional resources

- Adding cluster ID labels to metrics
- Obtaining your cluster ID

3.4.3. Setting up metrics collection for user-defined projects

You can create a **ServiceMonitor** resource to scrape metrics from a service endpoint in a user-defined project. This assumes that your application uses a Prometheus client library to expose metrics to the /**metrics** canonical name.

This section describes how to deploy a sample service in a user-defined project and then create a **ServiceMonitor** resource that defines how that service should be monitored.

3.4.3.1. Deploying a sample service

To test monitoring of a service in a user-defined project, you can deploy a sample service.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role or as a user with administrative permissions for the namespace.

Procedure

1. Create a YAML file for the service configuration. In this example, it is called **prometheus-example-app.yaml**.
2. Add the following deployment and service configuration details to the file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus-example-app
  name: prometheus-example-app
  namespace: ns1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-example-app
  template:
    metadata:
      labels:
        app: prometheus-example-app
    spec:
      containers:
      - image: ghcr.io/rhobs/prometheus-example-app:0.4.2
        imagePullPolicy: IfNotPresent
        name: prometheus-example-app
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: prometheus-example-app
```

```
    name: prometheus-example-app
    namespace: ns1
  spec:
    ports:
    - port: 8080
      protocol: TCP
      targetPort: 8080
      name: web
    selector:
      app: prometheus-example-app
    type: ClusterIP
```

This configuration deploys a service named **prometheus-example-app** in the user-defined **ns1** project. This service exposes the custom **version** metric.

3. Apply the configuration to the cluster:

```
$ oc apply -f prometheus-example-app.yaml
```

It takes some time to deploy the service.

4. You can check that the pod is running:

```
$ oc -n ns1 get pod
```

Example output

```
NAME                                  READY   STATUS    RESTARTS   AGE
prometheus-example-app-7857545cb7-sbgwq   1/1     Running   0          81m
```

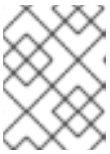
3.4.3.2. Specifying how a service is monitored

To use the metrics exposed by your service, you must configure OpenShift Dedicated monitoring to scrape metrics from the **/metrics** endpoint. You can do this using a **ServiceMonitor** custom resource definition (CRD) that specifies how a service should be monitored, or a **PodMonitor** CRD that specifies how a pod should be monitored. The former requires a **Service** object, while the latter does not, allowing Prometheus to directly scrape metrics from the metrics endpoint exposed by a pod.

This procedure shows you how to create a **ServiceMonitor** resource for a service in a user-defined project.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role or the **monitoring-edit** role.
- For this example, you have deployed the **prometheus-example-app** sample service in the **ns1** project.



NOTE

The **prometheus-example-app** sample service does not support TLS authentication.

Procedure

1. Create a new YAML configuration file named **example-app-service-monitor.yaml**.

2. Add a **ServiceMonitor** resource to the YAML file. The following example creates a service monitor named **prometheus-example-monitor** to scrape metrics exposed by the **prometheus-example-app** service in the **ns1** namespace:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1 (1)
spec:
  endpoints:
  - interval: 30s
    port: web (2)
    scheme: http
  selector: (3)
    matchLabels:
      app: prometheus-example-app
```

(1) Specify a user-defined namespace where your service runs.

(2) Specify endpoint ports to be scraped by Prometheus.

(3) Configure a selector to match your service based on its metadata labels.

NOTE

A **ServiceMonitor** resource in a user-defined namespace can only discover services in the same namespace. That is, the **namespaceSelector** field of the **ServiceMonitor** resource is always ignored.

3. Apply the configuration to the cluster:

```
$ oc apply -f example-app-service-monitor.yaml
```

It takes some time to deploy the **ServiceMonitor** resource.

4. Verify that the **ServiceMonitor** resource is running:

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

Example output

```
NAME                         AGE
prometheus-example-monitor   81m
```

3.4.3.3. Example service endpoint authentication settings

You can configure authentication for service endpoints for user-defined project monitoring by using **ServiceMonitor** and **PodMonitor** custom resource definitions (CRDs).

The following samples show different authentication settings for a **ServiceMonitor** resource. Each sample shows how to configure a corresponding **Secret** object that contains authentication credentials and other relevant settings.

3.4.3.3.1. Sample YAML authentication with a bearer token

The following sample shows bearer token settings for a **Secret** object named **example-bearer-auth** in the **ns1** namespace:

Example bearer token secret

```
apiVersion: v1
kind: Secret
metadata:
  name: example-bearer-auth
  namespace: ns1
stringData:
  token: <authentication_token> 1
```

1 Specify an authentication token.

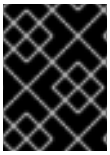
The following sample shows bearer token authentication settings for a **ServiceMonitor** CRD. The example uses a **Secret** object named **example-bearer-auth**:

Example bearer token authentication settings

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - authorization:
      credentials:
        key: token 1
        name: example-bearer-auth 2
    port: web
  selector:
    matchLabels:
      app: prometheus-example-app
```

1 The key that contains the authentication token in the specified **Secret** object.

2 The name of the **Secret** object that contains the authentication credentials.



IMPORTANT

Do not use **bearerTokenFile** to configure bearer token. If you use the **bearerTokenFile** configuration, the **ServiceMonitor** resource is rejected.

3.4.3.3.2. Sample YAML for Basic authentication

The following sample shows Basic authentication settings for a **Secret** object named **example-basic-auth** in the **ns1** namespace:

Example Basic authentication secret

```
apiVersion: v1
kind: Secret
metadata:
  name: example-basic-auth
  namespace: ns1
stringData:
  user: <basic_username> 1
  password: <basic_password> 2
```

1 Specify a username for authentication.

2 Specify a password for authentication.

The following sample shows Basic authentication settings for a **ServiceMonitor** CRD. The example uses a **Secret** object named **example-basic-auth**:

Example Basic authentication settings

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - basicAuth:
      username:
        key: user 1
        name: example-basic-auth 2
      password:
        key: password 3
        name: example-basic-auth 4
    port: web
  selector:
    matchLabels:
      app: prometheus-example-app
```

1 The key that contains the username in the specified **Secret** object.

2 4 The name of the **Secret** object that contains the Basic authentication.

3 The key that contains the password in the specified **Secret** object.

3.4.3.3.3. Sample YAML authentication with OAuth 2.0

The following sample shows OAuth 2.0 settings for a **Secret** object named **example-oauth2** in the **ns1** namespace:

Example OAuth 2.0 secret

```
apiVersion: v1
kind: Secret
metadata:
  name: example-oauth2
  namespace: ns1
stringData:
  id: <oauth2_id> 1
  secret: <oauth2_secret> 2
```

1 Specify an Oauth 2.0 ID.

2 Specify an Oauth 2.0 secret.

The following sample shows OAuth 2.0 authentication settings for a **ServiceMonitor** CRD. The example uses a **Secret** object named **example-oauth2**:

Example OAuth 2.0 authentication settings

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
  - oauth2:
      clientId:
        secret:
          key: id 1
          name: example-oauth2 2
      clientSecret:
        key: secret 3
        name: example-oauth2 4
      tokenUrl: https://example.com/oauth2/token 5
    port: web
  selector:
    matchLabels:
      app: prometheus-example-app
```

1 The key that contains the OAuth 2.0 ID in the specified **Secret** object.

2 4 The name of the **Secret** object that contains the OAuth 2.0 credentials.

3 The key that contains the OAuth 2.0 secret in the specified **Secret** object.

5 The URL used to fetch a token with the specified **clientId** and **clientSecret**.

Additional resources

- Scrape Prometheus metrics using TLS in ServiceMonitor configuration (Red Hat Customer Portal)

3.5. CONFIGURING ALERTS AND NOTIFICATIONS FOR USER WORKLOAD MONITORING

You can configure a local or external Alertmanager instance to route alerts from Prometheus to endpoint receivers. You can also attach custom labels to all time series and alerts to add useful metadata information.

3.5.1. Configuring external Alertmanager instances

The OpenShift Dedicated monitoring stack includes a local Alertmanager instance that routes alerts from Prometheus.

You can add external Alertmanager instances to route alerts for user-defined projects.

If you add the same external Alertmanager configuration for multiple clusters and disable the local instance for each cluster, you can then manage alert routing for multiple clusters by using a single external Alertmanager instance.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add an **additionalAlertmanagerConfigs** section with configuration details under **data/config.yaml/<component>**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

```
<component>: 1
  additionalAlertmanagerConfigs:
  - <alertmanager_specification> 2
```

2 Substitute **<alertmanager_specification>** with authentication and other configuration details for additional Alertmanager instances. Currently supported authentication methods are bearer token (**bearerToken**) and client TLS (**tlsConfig**).

1 Substitute **<component>** for one of two supported external Alertmanager components: **prometheus** or **thanosRuler**.

The following sample config map configures an additional Alertmanager for Thanos Ruler by using a bearer token with client TLS authentication:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      additionalAlertmanagerConfigs:
      - scheme: https
        pathPrefix: /
        timeout: "30s"
        apiVersion: v1
        bearerToken:
          name: alertmanager-bearer-token
          key: token
        tlsConfig:
          key:
            name: alertmanager-tls
            key: tls.key
          cert:
            name: alertmanager-tls
            key: tls.crt
          ca:
            name: alertmanager-tls
            key: tls.ca
        staticConfigs:
        - external-alertmanager1-remote.com
        - external-alertmanager1-remote2.com
```

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

3.5.2. Configuring secrets for Alertmanager

The OpenShift Dedicated monitoring stack includes Alertmanager, which routes alerts from Prometheus to endpoint receivers. If you need to authenticate with a receiver so that Alertmanager can send alerts to it, you can configure Alertmanager to use a secret that contains authentication credentials for the receiver.

For example, you can configure Alertmanager to use a secret to authenticate with an endpoint receiver that requires a certificate issued by a private Certificate Authority (CA). You can also configure Alertmanager to use a secret to authenticate with a receiver that requires a password file for Basic HTTP authentication. In either case, authentication details are contained in the **Secret** object rather than in the **ConfigMap** object.

3.5.2.1. Adding a secret to the Alertmanager configuration

You can add secrets to the Alertmanager configuration by editing the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project.

After you add a secret to the config map, the secret is mounted as a volume at **/etc/alertmanager/secrets/<secret_name>** within the **alertmanager** container for the Alertmanager pods.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have created the secret to be configured in Alertmanager in the **openshift-user-workload-monitoring** project.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add a **secrets:** section under **data/config.yaml/alertmanager** with the following configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    alertmanager:
      secrets: 1
      - <secret_name_1> 2
      - <secret_name_2>
```

1. This section contains the secrets to be mounted into Alertmanager. The secrets must be located within the same namespace as the Alertmanager object.
2. The name of the **Secret** object that contains authentication credentials for the receiver. If you add multiple secrets, place each one on a new line.

The following sample config map settings configure Alertmanager to use two **Secret** objects named **test-secret-basic-auth** and **test-secret-api-token**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    alertmanager:
      secrets:
      - test-secret-basic-auth
      - test-secret-api-token
```

3. Save the file to apply the changes. The new configuration is applied automatically.

3.5.3. Attaching additional labels to your time series and alerts

You can attach custom labels to all time series and alerts leaving Prometheus by using the external labels feature of Prometheus.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Define labels you want to add for every metric under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      externalLabels:
        <key>: <value> 1
```

1 Substitute **<key>: <value>** with key-value pairs where **<key>** is a unique name for the new label and **<value>** is its value.

WARNING

- Do not use **prometheus** or **prometheus_replica** as key names, because they are reserved and will be overwritten.
- Do not use **cluster** as a key name. Using it can cause issues where you are unable to see data in the developer dashboards.

NOTE

In the **openshift-user-workload-monitoring** project, Prometheus handles metrics and Thanos Ruler handles alerting and recording rules. Setting **externalLabels** for **prometheus** in the **user-workload-monitoring-config ConfigMap** object will only configure external labels for metrics and not for any rules.

For example, to add metadata about the region and environment to all time series and alerts, use the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      externalLabels:
        region: eu
        environment: prod
```

3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

3.5.4. Configuring alert notifications

In OpenShift Dedicated, the **dedicated-admin** user can enable alert routing for user-defined projects by using a separate Alertmanager instance for user-defined projects.

Developers and other users with the **alert-routing-edit** cluster role can configure custom alert notifications for their user-defined projects by configuring alert receivers.

NOTE

Review the following limitations of alert routing for user-defined projects:

- User-defined alert routing is scoped to the namespace in which the resource is defined. For example, a routing configuration in namespace **ns1** only applies to **PrometheusRules** resources in the same namespace.
- When a namespace is excluded from user-defined monitoring, **AlertmanagerConfig** resources in the namespace cease to be part of the Alertmanager configuration.

Additional resources

- Understanding alert routing for user-defined projects
- Sending notifications to external systems
- PagerDuty website
- Prometheus Integration Guide (PagerDuty documentation)
- Support version matrix for monitoring components
- Enabling alert routing for user-defined projects

3.5.4.1. Configuring alert routing for user-defined projects

If you are a non-administrator user who has been given the **alert-routing-edit** cluster role, you can create or edit alert routing for user-defined projects.

Prerequisites

- Alert routing has been enabled for user-defined projects.
- You are logged in as a user that has the **alert-routing-edit** cluster role for the project for which you want to create alert routing.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for alert routing. The example in this procedure uses a file called **example-app-alert-routing.yaml**.
2. Add an **AlertmanagerConfig** YAML definition to the file. For example:

```
apiVersion: monitoring.coreos.com/v1beta1
kind: AlertmanagerConfig
metadata:
  name: example-routing
  namespace: ns1
spec:
  route:
    receiver: default
```

```
    groupBy: [job]
  receivers:
  - name: default
    webhookConfigs:
    - url: https://example.org/post
```

3. Save the file.

4. Apply the resource to the cluster:

```
$ oc apply -f example-app-alert-routing.yaml
```

The configuration is automatically applied to the Alertmanager pods.

Additional resources

- Send test alerts to Alertmanager in OpenShift 4 (Red Hat Customer Portal)

3.5.4.2. Configuring alert routing for user-defined projects with the Alertmanager secret

If you have enabled a separate instance of Alertmanager that is dedicated to user-defined alert routing, you can customize where and how the instance sends notifications by editing the **alertmanager-user-workload** secret in the **openshift-user-workload-monitoring** namespace.

NOTE

All features of a supported version of upstream Alertmanager are also supported in an OpenShift Dedicated Alertmanager configuration. To check all the configuration options of a supported version of upstream Alertmanager, see Alertmanager configuration (Prometheus documentation).

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Print the currently active Alertmanager configuration into the file **alertmanager.yaml**:

```
$ oc -n openshift-user-workload-monitoring get secret alertmanager-user-workload --
template='{{ index .data "alertmanager.yaml" }}' | base64 --decode > alertmanager.yaml
```

2. Edit the configuration in **alertmanager.yaml**:

```
global:
  http_config:
    proxy_from_environment: true (1)
route:
  receiver: Default
  group_by:
  - name: Default
  routes:
```

```
  - matchers:
    - "service = prometheus-example-monitor" 2
    receiver: <receiver> 3
receivers:
- name: Default
- name: <receiver>
  <receiver_configuration> 4
```

1. If you configured an HTTP cluster-wide proxy, set the **proxy_from_environment** parameter to **true** to enable proxying for all alert receivers.
2. Specify labels to match your alerts. This example targets all alerts that have the **service="prometheus-example-monitor"** label.
3. Specify the name of the receiver to use for the alerts group.
4. Specify the receiver configuration.

3. Apply the new configuration in the file:

```
$ oc -n openshift-user-workload-monitoring create secret generic alertmanager-user-workload --from-file=alertmanager.yaml --dry-run=client -o=yaml |  oc -n openshift-user-workload-monitoring replace secret --filename=-
```

Additional resources

- Send test alerts to Alertmanager in OpenShift 4 (Red Hat Customer Portal)

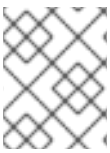
3.5.4.3. Configuring different alert receivers for default platform alerts and user-defined alerts

You can configure different alert receivers for default platform alerts and user-defined alerts to ensure the following results:

- All default platform alerts are sent to a receiver owned by the team in charge of these alerts.
- All user-defined alerts are sent to another receiver so that the team can focus only on platform alerts.

You can achieve this by using the **openshift_io_alert_source="platform"** label that is added by the Cluster Monitoring Operator to all platform alerts:

- Use the **openshift_io_alert_source="platform"** matcher to match default platform alerts.
- Use the **openshift_io_alert_source!="platform"** or **'openshift_io_alert_source=""'** matcher to match user-defined alerts.



NOTE

This configuration does not apply if you have enabled a separate instance of Alertmanager dedicated to user-defined alerts.

CHAPTER 4. ACCESSING METRICS

4.1. ACCESSING METRICS AS AN ADMINISTRATOR

You can access metrics to monitor the performance of cluster components and your workloads.

Additional resources

- Understanding metrics

4.1.1. Querying metrics for all projects with the OpenShift Dedicated web console

Monitor the state of a cluster and any user-defined workloads by using the OpenShift Dedicated metrics query browser. The query browser uses Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot.

As a **dedicated-admin** or as a user with view permissions for all projects, you can access metrics for all default OpenShift Dedicated and user-defined projects in the Metrics UI.

NOTE

Only dedicated administrators have access to the third-party UIs provided with OpenShift Dedicated monitoring.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role or with view permissions for all projects.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In the OpenShift Dedicated web console, click **Observe → Metrics**.
2. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the **Select query** drop-down list, select an existing query.

Option	Description
Create a custom query.	Add your Prometheus Query Language (PromQL) query to the **Expression** field. As you type a PromQL expression, autocomplete suggestions appear in a drop-down list. These suggestions include functions, metrics, labels, and time tokens. Use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. Move your mouse pointer over a suggested item to view a brief description of that item.
Add multiple queries.	Click **Add query**.
Duplicate an existing query.	Click the options menu ⋮ next to the query, then choose **Duplicate query**.
Disable a query from being run.	Click the options menu ⋮ next to the query and choose **Disable query**.

3. To run queries that you created, click **Run queries**. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.

NOTE

- When drawing time series graphs, queries that operate on large amounts of data might time out or overload the browser. To avoid this, click **Hide graph** and calibrate your query by using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.
- By default, the query table shows an expanded view that lists every metric and its current value. Click the ˅ down arrowhead to minimize the expanded view for a query.

4. Optional: Save the page URL to use this set of queries again in the future.

5. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot. Select which metrics are shown by performing any of the following actions:

Option	Description
Hide all metrics from a query.	Click the options menu ⋮ for the query and click **Hide all series**.

Option	Description
Hide a specific metric.	Go to the query table and click the colored square near the metric name.
Zoom into the plot and change the time range.	Perform one of the following actions: • Visually select the time range by clicking and dragging on the plot horizontally. • Use the menu to select the time range.
Reset the time range.	Click **Reset zoom**.
Display outputs for all queries at a specific point in time.	Hover over the plot at the point you are interested in. The query outputs appear in a pop-up box.
Hide the plot.	Click **Hide graph**.

Additional resources

- Querying Prometheus (Prometheus documentation)

4.1.2. Getting detailed information about a metrics target

You can use the OpenShift Dedicated web console to view, search, and filter the endpoints that are currently targeted for scraping, which helps you to identify and troubleshoot problems. For example, you can view the current status of targeted endpoints to see when OpenShift Dedicated monitoring is not able to scrape metrics from a targeted component.

The **Metrics targets** page shows targets for user-defined projects.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.

Procedure

1. In the OpenShift Dedicated web console, go to **Observe → Targets**. The **Metrics targets** page opens with a list of all service endpoint targets that are being scraped for metrics.
   This page shows details about targets for default OpenShift Dedicated and user-defined projects. This page lists the following information for each target:
   - Service endpoint URL being scraped
   - The **ServiceMonitor** resource being monitored
   - The **up** or **down** status of the target
   - Namespace

- Last scrape time
- Duration of the last scrape

2. Optional: To find a specific target, perform any of the following actions:

Option	Description

Filter the targets by status and source.	Choose filters in the **Filter** list.
	
	The following filtering options are available:
	
	• **Status** filters:
	  ◦ **Up**. The target is currently up and being actively scraped for metrics.
	  ◦ **Down**. The target is currently down and not being scraped for metrics.
	• **Source** filters:
	  ◦ **Platform**. Platform-level targets relate only to default Red Hat OpenShift Service on AWS projects. These projects provide core Red Hat OpenShift Service on AWS functionality.
	  ◦ **User**. User targets relate to user-defined projects. These projects are user-created and can be customized.

Search for a target by name or label.	Enter a search term in the **Text** or **Label** field next to the search box.
Sort the targets.	Click one or more of the **Endpoint Status**, **Namespace**, **Last Scrape**, and **Scrape Duration** column headers.

3. Click the URL in the **Endpoint** column for a target to go to its **Target details** page. This page provides information about the target, including the following information:

- The endpoint URL being scraped for metrics
- The current **Up** or **Down** status of the target
- A link to the namespace
- A link to the **ServiceMonitor** resource details
- Labels attached to the target
- The most recent time that the target was scraped for metrics

4.1.3. Reviewing monitoring dashboards as a cluster administrator

As an administrator, you can view dashboards relating to core OpenShift Dedicated cluster components.

IMPORTANT

Starting with OpenShift Dedicated 4.19, the perspectives in the web console have unified. The **Developer** perspective is no longer enabled by default.

All users can interact with all OpenShift Dedicated web console features. However, if you are not the cluster owner, you might need to request permission to access certain features from the cluster owner.

You can still enable the **Developer** perspective. On the **Getting Started** pane in the web console, you can take a tour of the console, find information on setting up your cluster, view a quick start for enabling the **Developer** perspective, and follow links to explore new features and capabilities.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.

Procedure

1. In the OpenShift Dedicated web console, go to **Observe → Dashboards**.
2. Choose a dashboard in the **Dashboard** list. Some dashboards, such as **etcd** and **Prometheus** dashboards, produce additional sub-menus when selected.
3. Optional: Select a time range for the graphs in the **Time range** list.
   - Select a predefined time period.
   - Set a custom time range by clicking **Custom time range** in the **Time range** list.
     a. Input or select the **From** and **To** dates and times.
     b. Click **Save** to save the custom time range.
4. Optional: Select a **Refresh interval**.
5. Hover over each of the graphs within a dashboard to display detailed information about specific items.

Additional resources

- About monitoring dashboards

4.2. ACCESSING METRICS AS A DEVELOPER

You can access metrics to monitor the performance of your cluster workloads.

Additional resources

- Understanding metrics

4.2.1. Querying metrics for user-defined projects with the OpenShift Dedicated web console

Monitor user-defined workloads by using the OpenShift Dedicated metrics query browser. The query browser uses Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot.

As a developer, you must specify a project name when querying metrics. You must have the required privileges to view metrics for the selected project.

NOTE

Developers cannot access the third-party UIs provided with OpenShift Dedicated monitoring.

Prerequisites

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.
- You have enabled monitoring for user-defined projects.
- You have deployed a service in a user-defined project.
- You have created a **ServiceMonitor** custom resource definition (CRD) for the service to define how the service is monitored.

Procedure

1. In the OpenShift Dedicated web console, click **Observe → Metrics**.
2. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the **Select query** drop-down list, select an existing query.
Create a custom query.	Add your Prometheus Query Language (PromQL) query to the **Expression** field. As you type a PromQL expression, autocomplete suggestions appear in a drop-down list. These suggestions include functions, metrics, labels, and time tokens. Use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. Move your mouse pointer over a suggested item to view a brief description of that item.
Add multiple queries.	Click **Add query**.

Option	Description
Duplicate an existing query.	Click the options menu ⋮ next to the query, then choose **Duplicate query**.
Disable a query from being run.	Click the options menu ⋮ next to the query and choose **Disable query**.

3. To run queries that you created, click **Run queries**. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.

NOTE

- When drawing time series graphs, queries that operate on large amounts of data might time out or overload the browser. To avoid this, click **Hide graph** and calibrate your query by using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.
- By default, the query table shows an expanded view that lists every metric and its current value. Click the ˅ down arrowhead to minimize the expanded view for a query.

4. Optional: Save the page URL to use this set of queries again in the future.

5. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot. Select which metrics are shown by performing any of the following actions:

Option	Description
Hide all metrics from a query.	Click the options menu ⋮ for the query and click **Hide all series**.
Hide a specific metric.	Go to the query table and click the colored square near the metric name.
Zoom into the plot and change the time range.	Perform one of the following actions: • Visually select the time range by clicking and dragging on the plot horizontally. • Use the menu to select the time range.
Reset the time range.	Click **Reset zoom**.

Option	Description
Display outputs for all queries at a specific point in time.	Hover over the plot at the point you are interested in. The query outputs appear in a pop-up box.
Hide the plot.	Click **Hide graph**.

Additional resources

- Querying Prometheus (Prometheus documentation)

4.2.2. Reviewing monitoring dashboards as a developer

As a developer, you can view dashboards relating to projects you have permissions for.

IMPORTANT

Starting with OpenShift Dedicated 4.19, the perspectives in the web console have unified. The **Developer** perspective is no longer enabled by default.

All users can interact with all OpenShift Dedicated web console features. However, if you are not the cluster owner, you might need to request permission to access certain features from the cluster owner.

You can still enable the **Developer** perspective. On the **Getting Started** pane in the web console, you can take a tour of the console, find information on setting up your cluster, view a quick start for enabling the **Developer** perspective, and follow links to explore new features and capabilities.

Prerequisites

- You have access to the cluster as a developer or as a user.
- You have view permissions for the project that you are viewing the dashboard for.
- A cluster administrator has enabled the **Developer** perspective in the web console.

Procedure

1. In the **Developer** perspective of the OpenShift Dedicated web console, click **Observe** and go to the **Dashboards** tab.
2. Select a project from the **Project:** drop-down list.
3. Select a dashboard from the **Dashboard** drop-down list to see the filtered metrics.
4. Optional: Select a time range for the graphs in the **Time range** list.
   - Select a predefined time period.
   - Set a custom time range by clicking **Custom time range** in the **Time range** list.
     a. Input or select the **From** and **To** dates and times.

b. Click **Save** to save the custom time range.

5. Optional: Select a **Refresh interval**.

6. Hover over each of the graphs within a dashboard to display detailed information about specific items.

Additional resources

- About monitoring dashboards
- Monitoring project and application metrics using the Developer perspective

4.3. ACCESSING MONITORING APIS BY USING THE CLI

In OpenShift Dedicated, you can access web service APIs for some monitoring components from the command-line interface (CLI).

IMPORTANT

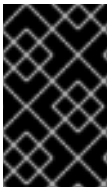
In certain situations, accessing API endpoints can degrade the performance and scalability of your cluster, especially if you use endpoints to retrieve, send, or query large amounts of metrics data.

To avoid these issues, consider the following recommendations:

- Avoid querying endpoints frequently. Limit queries to a maximum of one every 30 seconds.
- Do not retrieve all metrics data through the **/federate** endpoint for Prometheus. Query the endpoint only when you want to retrieve a limited, aggregated data set. For example, retrieving fewer than 1,000 samples for each request helps minimize the risk of performance degradation.

4.3.1. About accessing monitoring web service APIs

To interact with the monitoring stack by using the command line, you can access web service API endpoints for Prometheus, Alertmanager, Thanos Ruler, and Thanos Querier. Direct API access requires bearer token authentication and the correct namespace permissions.



IMPORTANT

To access Thanos Ruler and Thanos Querier service APIs, the requesting account must have **get** permission on the namespaces resource, which can be granted by binding the **cluster-monitoring-view** cluster role to the account.

When you access web service API endpoints for monitoring components, be aware of the following limitations:

- You can only use bearer token authentication to access API endpoints.
- You can only access endpoints in the **/api** path for a route. If you try to access an API endpoint in a web browser, an **Application is not available** error occurs. To access monitoring features in a web browser, use the OpenShift Dedicated web console to review monitoring dashboards.

Additional resources

- Reviewing monitoring dashboards as a cluster administrator
- Reviewing monitoring dashboards as a developer

4.3.2. Accessing a monitoring web service API

The following example shows how to query the service API receivers for the Alertmanager service used in core platform monitoring. You can use a similar method to access the **prometheus-k8s** service for core platform Prometheus and the **thanos-ruler** service for Thanos Ruler.

Prerequisites

- You are logged in to an account that is bound against the **monitoring-alertmanager-edit** role in the **openshift-monitoring** namespace.
- You are logged in to an account that has permission to get the Alertmanager API route.

NOTE

If your account does not have permission to get the Alertmanager API route, a cluster administrator can provide the URL for the route.

Procedure

1. Extract an authentication token by running the following command:

```
$ TOKEN=$(oc whoami -t)
```

2. Extract the **alertmanager-main** API route URL by running the following command:

```
$ HOST=$(oc -n openshift-monitoring get route alertmanager-main -
ojsonpath='{.status.ingress[].host}')
```

3. Query the service API receivers for Alertmanager by running the following command:

```
$ curl -H "Authorization: Bearer $TOKEN" -k "https://$HOST/api/v2/receivers"
```

4.3.3. Querying metrics by using the federation endpoint for Prometheus

You can use the federation endpoint for Prometheus to scrape platform and user-defined metrics from a network location outside the cluster. To do so, access the Prometheus **/federate** endpoint for the cluster via an OpenShift Dedicated route.

IMPORTANT

A delay in retrieving metrics data occurs when you use federation. This delay can affect the accuracy and timeliness of the scraped metrics.

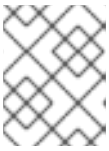
Using the federation endpoint can also degrade the performance and scalability of your cluster, especially if you use the federation endpoint to retrieve large amounts of metrics data. To avoid these issues, follow these recommendations:

- Do not try to retrieve all metrics data via the federation endpoint for Prometheus. Query it only when you want to retrieve a limited, aggregated data set. For example, retrieving fewer than 1,000 samples for each request helps minimize the risk of performance degradation.
- Avoid frequent querying of the federation endpoint for Prometheus. Limit queries to a maximum of one every 30 seconds.

If you need to forward large amounts of data outside the cluster, use remote write instead. For more information, see the *Configuring remote write storage* section.

Prerequisites

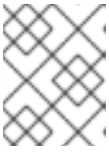
- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-monitoring-view** cluster role or have obtained a bearer token with **get** permission on the **namespaces** resource.



NOTE

You can only use bearer token authentication to access the Prometheus federation endpoint.

- You are logged in to an account that has permission to get the Prometheus federation route.



NOTE

If your account does not have permission to get the Prometheus federation route, a cluster administrator can provide the URL for the route.

Procedure

1. Retrieve the bearer token by running the following the command:

```
$ TOKEN=$(oc whoami -t)
```

2. Get the Prometheus federation route URL by running the following command:

```
$ HOST=$(oc -n openshift-monitoring get route prometheus-k8s-federate -
ojsonpath='{.status.ingress[].host}')
```

3. Query metrics from the /**federate** route. The following example command queries **up** metrics:

```
$ curl -G -k -H "Authorization: Bearer $TOKEN" https://$HOST/federate --data-urlencode
'match[]=up'
```

-

Example output

```
# TYPE up untyped
up{apiserver="kube-
apiserver",endpoint="https",instance="10.0.143.148:6443",job="apiserver",namespace="default
",service="kubernetes",prometheus="openshift-
monitoring/k8s",prometheus_replica="prometheus-k8s-0"} 1 1657035322214
up{apiserver="kube-
apiserver",endpoint="https",instance="10.0.148.166:6443",job="apiserver",namespace="default
",service="kubernetes",prometheus="openshift-
monitoring/k8s",prometheus_replica="prometheus-k8s-0"} 1 1657035338597
up{apiserver="kube-
apiserver",endpoint="https",instance="10.0.173.16:6443",job="apiserver",namespace="default",
service="kubernetes",prometheus="openshift-
monitoring/k8s",prometheus_replica="prometheus-k8s-0"} 1 1657035343834
...
```

4.3.4. Accessing metrics from outside the cluster for custom applications

You can query Prometheus metrics from outside the cluster when monitoring your own services with user-defined projects. Access this data from outside the cluster by using the **thanos-querier** route.

This access only supports using a bearer token for authentication.

Prerequisites

- You have deployed your own service, following the "Enabling monitoring for user-defined projects" procedure.
- You are logged in to an account with the **cluster-monitoring-view** cluster role, which provides permission to access the Thanos Querier API.
- You are logged in to an account that has permission to get the Thanos Querier API route.

NOTE

If your account does not have permission to get the Thanos Querier API route, a cluster administrator can provide the URL for the route.

Procedure

1. Extract an authentication token to connect to Prometheus by running the following command:

```
$ TOKEN=$(oc whoami -t)
```

2. Extract the **thanos-querier** API route URL by running the following command:

```
$ HOST=$(oc -n openshift-monitoring get route thanos-querier -
ojsonpath='{.status.ingress[].host}')
```

3. Set the namespace to the namespace in which your service is running by using the following command:

-

```
$ NAMESPACE=ns1
```

4. Query the metrics of your own services in the command line by running the following command:

```
$ curl -H "Authorization: Bearer $TOKEN" -k "https://$HOST/api/v1/query?" --data-urlencode
"query=up{namespace='$NAMESPACE'}"
```

The output shows the status for each application pod that Prometheus is scraping:

The formatted example output


```
{
  "status": "success",
  "data": {
    "resultType": "vector",
    "result": [
      {
        "metric": {
          "__name__": "up",
          "endpoint": "web",
          "instance": "10.129.0.46:8080",
          "job": "prometheus-example-app",
          "namespace": "ns1",
          "pod": "prometheus-example-app-68d47c4fb6-jztp2",
          "service": "prometheus-example-app"
        },
        "value": [
          1591881154.748,
          "1"
        ]
      }
    ],
  }
}
```


NOTE

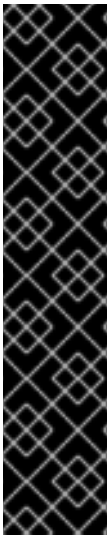
- The formatted example output uses a filtering tool, such as **jq**, to provide the formatted indented JSON. See the jq Manual (jq documentation) for more information about using **jq**.
- The command requests an instant query endpoint of the Thanos Querier service, which evaluates selectors at one point in time.

4.3.5. Resources reference for the Cluster Monitoring Operator

This document describes the following resources deployed and managed by the Cluster Monitoring Operator (CMO):

- Routes
- Services

Use this information when you want to configure API endpoint connections to retrieve, send, or query metrics data.



IMPORTANT

In certain situations, accessing endpoints can degrade the performance and scalability of your cluster, especially if you use endpoints to retrieve, send, or query large amounts of metrics data.

To avoid these issues, follow these recommendations:

- Avoid querying endpoints frequently. Limit queries to a maximum of one every 30 seconds.
- Do not try to retrieve all metrics data via the **/federate** endpoint. Query it only when you want to retrieve a limited, aggregated data set. For example, retrieving fewer than 1,000 samples for each request helps minimize the risk of performance degradation.

4.3.5.1. CMO routes resources

4.3.5.1.1. openshift-monitoring/alertmanager-main

Expose the /**api** endpoints of the **alertmanager-main** service via a router.

4.3.5.1.2. openshift-monitoring/prometheus-k8s

Expose the /**api** endpoints of the **prometheus-k8s** service via a router.

4.3.5.1.3. openshift-monitoring/prometheus-k8s-federate

Expose the /**federate** endpoint of the **prometheus-k8s** service via a router.

4.3.5.1.4. openshift-user-workload-monitoring/federate

Expose the /**federate** endpoint of the **prometheus-user-workload** service via a router.

4.3.5.1.5. openshift-monitoring/thanos-querier

Expose the /**api** endpoints of the **thanos-querier** service via a router.

4.3.5.1.6. openshift-user-workload-monitoring/thanos-ruler

Expose the /**api** endpoints of the **thanos-ruler** service via a router.

4.3.5.2. CMO services resources

4.3.5.2.1. openshift-monitoring/prometheus-operator-admission-webhook

Expose the admission webhook service which validates **PrometheusRules** and **AlertmanagerConfig** custom resources on port 8443.

4.3.5.2.2. openshift-user-workload-monitoring/alertmanager-user-workload

Expose the user-defined Alertmanager web server within the cluster on the following ports:

- Port 9095 provides access to the Alertmanager endpoints. Granting access requires binding a user to the **monitoring-alertmanager-api-reader** role (for read-only operations) or the **monitoring-alertmanager-api-writer** role in the **openshift-user-workload-monitoring** project.
- Port 9092 provides access to the Alertmanager endpoints restricted to a given project. Granting access requires binding a user to the **monitoring-rules-edit** cluster role or **monitoring-edit** cluster role in the project.
- Port 9097 provides access to the /**metrics** endpoint only. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.3. openshift-monitoring/alertmanager-main

Expose the Alertmanager web server within the cluster on the following ports:

- Port 9094 provides access to all the Alertmanager endpoints. Granting access requires binding a user to the **monitoring-alertmanager-view** role (for read-only operations) or the **monitoring-alertmanager-edit** role in the **openshift-monitoring** project.

  Example monitoring-alertmanager-view permissions

  The following example exercises permissions granted by the **monitoring-alertmanager-view** role. The binding commands must be run by a user with the necessary privileges.

  1. Create a test namespace and a service account.

     ```
     $ oc create namespace test-alertmanager-web-monitoring-alertmanager-view
     ```

     ```
     $ oc create serviceaccount am-client --namespace=test-alertmanager-web-monitoring-alertmanager-view
     ```

  2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

     ```
     $ oc create rolebinding test-alertmanager-web-monitoring-alertmanager-view \
       --namespace=openshift-monitoring \
       --role=monitoring-alertmanager-view \
       --serviceaccount=test-alertmanager-web-monitoring-alertmanager-view:am-client
     ```

  3. Generate a token to access the endpoints.

     ```
     $ TOKEN=$(oc create token am-client --namespace=test-alertmanager-web-monitoring-alertmanager-view)
     ```

  4. Access Alertmanager endpoints externally.

     ```
     $ ROUTE=$(oc get route alertmanager-main --namespace=openshift-monitoring -ojsonpath={.spec.host})
     ```

     ```
     $ curl -k -H "Authorization: Bearer $TOKEN" "https://$ROUTE/api/v2/alerts?filter=alertname=Watchdog"
     ```

5. Access Alertmanager endpoints from within the cluster.

```
$ curl -k -H "Authorization: Bearer $TOKEN" "https://alertmanager-main.openshift-monitoring:9094/api/v2/alerts?filter=alertname=Watchdog"
```

Example monitoring-alertmanager-edit permissions

The following example exercises permissions granted by the **monitoring-alertmanager-edit** role. The binding commands must be run by a user with the necessary privileges.

1. Create a test namespace and a service account.

```
$ oc create namespace test-alertmanager-web-monitoring-alertmanager-edit
```

```
$ oc create serviceaccount am-client --namespace=test-alertmanager-web-monitoring-alertmanager-edit
```

2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-alertmanager-web-monitoring-alertmanager-edit \
  --namespace=openshift-monitoring \
  --role=monitoring-alertmanager-edit \
  --serviceaccount=test-alertmanager-web-monitoring-alertmanager-edit:am-client
```

3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token am-client --namespace=test-alertmanager-web-monitoring-alertmanager-edit)
```

4. Access Alertmanager endpoints externally.

```
$ ROUTE=$(oc get route alertmanager-main --namespace=openshift-monitoring -ojsonpath={.spec.host})
```

```
$ curl -k -X POST  "https://$ROUTE/api/v2/silences" \
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" \
  -d '{
    "matchers": [
      {
        "name": "alertname",
        "value": "MyTestAlert1",
        "isRegex": false
      }
    ],
    "startsAt": "2044-01-01T00:00:00Z",
    "endsAt": "2044-01-01T00:00:01Z",
    "createdBy": "test-alertmanager-web-monitoring-alertmanager-edit/am-client",
    "comment": "Silence test"
  }'
```

5. Access Alertmanager endpoints from within the cluster.

```
$ curl -k -X POST  "https://alertmanager-main.openshift-
monitoring:9094/api/v2/silences" \
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" \
  -d '{
    "matchers": [
      {
        "name": "alertname",
        "value": "MyTestAlert2",
        "isRegex": false
      }
    ],
    "startsAt": "2044-01-01T00:00:00Z",
    "endsAt": "2044-01-01T00:00:01Z",
    "createdBy": "test-alertmanager-web-monitoring-alertmanager-edit/am-client",
    "comment": "Silence test"
  }'
```

- Port 9092 provides access to the Alertmanager endpoints restricted to a given project. Granting access requires binding a user to the **monitoring-rules-edit** cluster role or **monitoring-edit** cluster role in the project.

  Example monitoring-rules-edit permissions

  The following example exercises permissions granted by the **monitoring-rules-edit** cluster role. The binding commands must be run by a user with the necessary privileges.

  1. Create a test namespace and a service account.

     ```
     $ oc create namespace test-alertmanager-tenancy-monitoring-rules-edit
     ```

     ```
     $ oc create serviceaccount am-client --namespace=test-alertmanager-tenancy-
     monitoring-rules-edit
     ```

  2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

     ```
     $ oc create rolebinding test-alertmanager-tenancy-monitoring-rules-edit \
       --namespace=test-alertmanager-tenancy-monitoring-rules-edit \
       --clusterrole=monitoring-rules-edit \
       --serviceaccount=test-alertmanager-tenancy-monitoring-rules-edit:am-client
     ```

  3. Generate a token to access the endpoints.

     ```
     $ TOKEN=$(oc create token am-client --namespace=test-alertmanager-tenancy-
     monitoring-rules-edit)
     ```

  4. Access Alertmanager endpoints from within the cluster. The port is not exposed externally by default.

     ```
     $ curl -k -f -H "Authorization: Bearer $TOKEN" "https://alertmanager-main.openshift-
     monitoring:9092/api/v2/alerts?namespace=test-alertmanager-tenancy-monitoring-
     rules-edit"
     ```

```
$ curl -k -X POST -f "https://alertmanager-main.openshift-monitoring:9092/api/v2/silences?namespace=test-alertmanager-tenancy-monitoring-rules-edit" \
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" \
  -d '{
    "matchers": [
      {
        "name": "alertname",
        "value": "MyTestAlert",
        "isRegex": false
      }
    ],
    "startsAt": "2044-01-01T00:00:00Z",
    "endsAt": "2044-01-01T00:00:01Z",
    "createdBy": "test-alertmanager-tenancy-monitoring-rules-edit/am-client",
    "comment": "Silence test"
  }'
```

Example monitoring-edit permissions

The following example exercises permissions granted by the **monitoring-edit** cluster role. The binding commands must be run by a user with the necessary privileges.

1. Create a test namespace and a service account.

```
$ oc create namespace test-alertmanager-tenancy-monitoring-edit
```

```
$ oc create serviceaccount am-client --namespace=test-alertmanager-tenancy-monitoring-edit
```

2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-alertmanager-tenancy-monitoring-edit \
  --namespace=test-alertmanager-tenancy-monitoring-edit \
  --clusterrole=monitoring-edit \
  --serviceaccount=test-alertmanager-tenancy-monitoring-edit:am-client
```

3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token am-client --namespace=test-alertmanager-tenancy-monitoring-edit)
```

4. Access Alertmanager endpoints from within the cluster. The port is not exposed externally by default.

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://alertmanager-main.openshift-monitoring:9092/api/v2/alerts?namespace=test-alertmanager-tenancy-monitoring-edit"
```

```
$ curl -k -X POST -f "https://alertmanager-main.openshift-monitoring:9092/api/v2/silences?namespace=test-alertmanager-tenancy-monitoring-edit" \
  -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" \
```

```
-d '{
  "matchers": [
    {
      "name": "alertname",
      "value": "MyTestAlert",
      "isRegex": false
    }
  ],
  "startsAt": "2044-01-01T00:00:00Z",
  "endsAt": "2044-01-01T00:00:01Z",
  "createdBy": "test-alertmanager-tenancy-monitoring-edit/am-client",
  "comment": "Silence test"
}'
```

- Port 9097 provides access to the /**metrics** endpoint only. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.4. openshift-monitoring/kube-state-metrics

Expose kube-state-metrics /**metrics** endpoints within the cluster on the following ports:

- Port 8443 provides access to the Kubernetes resource metrics. This port is for internal use, and no other usage is guaranteed.
- Port 9443 provides access to the internal kube-state-metrics metrics. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.5. openshift-monitoring/metrics-server

Expose the metrics-server web server on port 443. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.6. openshift-monitoring/monitoring-plugin

Expose the monitoring plugin service on port 9443. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.7. openshift-monitoring/node-exporter

Expose the /**metrics** endpoint on port 9100. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.8. openshift-monitoring/openshift-state-metrics

Expose openshift-state-metrics /**metrics** endpoints within the cluster on the following ports:

- Port 8443 provides access to the OpenShift resource metrics. This port is for internal use, and no other usage is guaranteed.
- Port 9443 provides access to the internal **openshift-state-metrics** metrics. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.9. openshift-monitoring/prometheus-k8s

Expose the Prometheus web server within the cluster on the following ports:

- Port 9091 provides access to all the Prometheus endpoints. Granting access requires binding a user to the **cluster-monitoring-view** cluster role or **cluster-monitoring-metrics-api** cluster role in the **openshift-monitoring** project.

  Example cluster-monitoring-view permissions

  The following example exercises permissions granted by the **cluster-monitoring-view** cluster role. The binding commands must be run by a user with the necessary privileges.

  1. Create a test namespace and a service account.

     ```
     $ oc create namespace test-prometheus-web-cluster-monitoring-view
     ```

     ```
     $ oc create serviceaccount prom-client --namespace=test-prometheus-web-cluster-monitoring-view
     ```

  2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

     ```
     $ oc create rolebinding test-prometheus-web-cluster-monitoring-view \
       --namespace=openshift-monitoring \
       --clusterrole=cluster-monitoring-view \
       --serviceaccount=test-prometheus-web-cluster-monitoring-view:prom-client
     ```

  3. Generate a token to access the endpoints.

     ```
     $ TOKEN=$(oc create token prom-client --namespace=test-prometheus-web-cluster-monitoring-view)
     ```

  4. Access Prometheus endpoints externally.

     ```
     $ ROUTE=$(oc get route prometheus-k8s --namespace=openshift-monitoring -ojsonpath={.spec.host})
     ```

     ```
     $ curl -k -H "Authorization: Bearer $TOKEN" "https://$ROUTE/api/v1/query?query=up"
     ```

  5. Access Prometheus endpoints from within the cluster.

     ```
     $ curl -k -H "Authorization: Bearer $TOKEN" "https://prometheus-k8s.openshift-monitoring:9091/api/v1/query?query=up"
     ```

  Example cluster-monitoring-metrics-api permissions

  The following example exercises permissions granted by the **cluster-monitoring-metrics-api** role. The binding commands must be run by a user with the necessary privileges.

  1. Create a test namespace and a service account.

     ```
     $ oc create namespace test-prometheus-web-cluster-monitoring-metrics-api
     ```

```
$ oc create serviceaccount prom-client --namespace=test-prometheus-web-cluster-monitoring-metrics-api
```

2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-prometheus-web-cluster-monitoring-metrics-api \
  --namespace=openshift-monitoring \
  --role=cluster-monitoring-metrics-api  \
  --serviceaccount=test-prometheus-web-cluster-monitoring-metrics-api:prom-client
```

3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token prom-client --namespace=test-prometheus-web-cluster-monitoring-metrics-api)
```

4. Access Prometheus endpoints externally.

```
$ ROUTE=$(oc get route prometheus-k8s --namespace=openshift-monitoring -ojsonpath={.spec.host})
```

```
$ curl -k -H "Authorization: Bearer $TOKEN" "https://$ROUTE/api/v1/query?query=up"
```

5. Access Prometheus endpoints from within the cluster.

```
$ curl -k -H "Authorization: Bearer $TOKEN" "https://prometheus-k8s.openshift-monitoring:9091/api/v1/query?query=up"
```

- Port 9092 provides access to the **/metrics** and **/federate** endpoints only. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.10. openshift-user-workload-monitoring/prometheus-operator

Expose the **/metrics** endpoint on port 8443. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.11. openshift-monitoring/prometheus-operator

Expose the **/metrics** endpoint on port 8443. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.12. openshift-user-workload-monitoring/prometheus-user-workload

Expose the Prometheus web server within the cluster on the following ports:

- Port 9091 provides access to the **/metrics** endpoint only. This port is for internal use, and no other usage is guaranteed.
- Port 9092 provides access to the **/federate** endpoint only. Granting access requires binding a user to the **cluster-monitoring-view** cluster role.

This also exposes the /**metrics** endpoint of the Thanos sidecar web server on port 10902. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.13. openshift-monitoring/telemeter-client

Expose the /**metrics** endpoint on port 8443. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.14. openshift-monitoring/thanos-querier

Expose the Thanos Querier web server within the cluster on the following ports:

- Port 9091 provides access to all the Thanos Querier endpoints. Granting access requires binding a user to the **cluster-monitoring-view** cluster role or **cluster-monitoring-metrics-api** cluster role in the **openshift-monitoring** project.

  Example cluster-monitoring-view permissions

  The following example exercises permissions granted by the **cluster-monitoring-view** cluster role. The binding commands must be run by a user with the necessary privileges.

  1. Create a test namespace and a service account.

     ```
     $ oc create namespace test-thanos-querier-web-cluster-monitoring-view
     ```

     ```
     $ oc create serviceaccount thanos-client --namespace=test-thanos-querier-web-cluster-monitoring-view
     ```

  2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

     ```
     $ oc create rolebinding test-thanos-querier-web-cluster-monitoring-view \
       --namespace=openshift-monitoring \
       --clusterrole=cluster-monitoring-view \
       --serviceaccount=test-thanos-querier-web-cluster-monitoring-view:thanos-client
     ```

  3. Generate a token to access the endpoints.

     ```
     $ TOKEN=$(oc create token thanos-client --namespace=test-thanos-querier-web-cluster-monitoring-view)
     ```

  4. Access Thanos Querier endpoints externally.

     ```
     $ ROUTE=$(oc get route thanos-querier --namespace=openshift-monitoring -ojsonpath={.spec.host})
     ```

     ```
     $ curl -k -H "Authorization: Bearer $TOKEN" "https://$ROUTE/api/v1/query?query=up"
     ```

  5. Access Thanos Querier endpoints from within the cluster.

     ```
     $ curl -k -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9091/api/v1/query?query=up"
     ```

Example cluster-monitoring-metrics-api permissions

The following example exercises permissions granted by the **cluster-monitoring-metrics-api** role. The binding commands must be run by a user with the necessary privileges.

1. Create a test namespace and a service account.

```
$ oc create namespace test-thanos-querier-web-cluster-monitoring-metrics-api
```

```
$ oc create serviceaccount thanos-client --namespace=test-thanos-querier-web-cluster-monitoring-metrics-api
```

2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-thanos-querier-web-cluster-monitoring-metrics-api \
  --namespace=openshift-monitoring \
  --role=cluster-monitoring-metrics-api  \
  --serviceaccount=test-thanos-querier-web-cluster-monitoring-metrics-api:thanos-client
```

3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token thanos-client --namespace=test-thanos-querier-web-cluster-monitoring-metrics-api)
```

4. Access Thanos Querier endpoints externally.

```
$ ROUTE=$(oc get route thanos-querier --namespace=openshift-monitoring -ojsonpath={.spec.host})
```

```
$ curl -k -H "Authorization: Bearer $TOKEN" "https://$ROUTE/api/v1/query?query=up"
```

5. Access Thanos Querier endpoints from within the cluster.

```
$ curl -k -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9091/api/v1/query?query=up"
```

- Port 9092 provides access to the **/api/v1/query**, **/api/v1/query_range/**, **/api/v1/labels**, **/api/v1/label/*/values**, and **/api/v1/series** endpoints restricted to a given project. Granting access requires binding a user to the **view** cluster role in the project.

Example view permissions

The following example exercises permissions granted by the **view** cluster role. The binding commands must be run by a user with the necessary privileges.

1. Create a test namespace and a service account.

```
$ oc create namespace test-thanos-querier-tenancy-view
```

```
$ oc create serviceaccount thanos-client --namespace=test-thanos-querier-tenancy-view
```

2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-thanos-querier-tenancy-view \
  --namespace=test-thanos-querier-tenancy-view \
  --clusterrole=view \
  --serviceaccount=test-thanos-querier-tenancy-view:thanos-client
```

3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token thanos-client --namespace=test-thanos-querier-tenancy-view)
```

4. Access Thanos Querier endpoints from within the cluster. The port is not exposed externally by default.

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9092/api/v1/query?query=up&namespace=test-thanos-querier-tenancy-view"
```

- Port 9093 provides access to the **/api/v1/alerts**, and **/api/v1/rules** endpoints restricted to a given project. Granting access requires binding a user to the **monitoring-rules-edit**, **monitoring-edit**, or **monitoring-rules-view** cluster role in the project.

  Example monitoring-rules-edit permissions

  The following example exercises permissions granted by the **monitoring-rules-edit** cluster role. The binding commands must be run by a user with the necessary privileges.

  1. Create a test namespace and a service account.

```
$ oc create namespace test-thanos-querier-tenancy-rules-monitoring-rules-edit
```

```
$ oc create serviceaccount thanos-client --namespace=test-thanos-querier-tenancy-rules-monitoring-rules-edit
```

  2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-thanos-querier-tenancy-rules-monitoring-rules-edit \
  --namespace=test-thanos-querier-tenancy-rules-monitoring-rules-edit \
  --clusterrole=monitoring-rules-edit \
  --serviceaccount=test-thanos-querier-tenancy-rules-monitoring-rules-edit:thanos-client
```

  3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token thanos-client --namespace=test-thanos-querier-tenancy-rules-monitoring-rules-edit)
```

4. Access Thanos Querier endpoints from within the cluster. The port is not exposed externally by default.

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9093/api/v1/rules?namespace=test-thanos-querier-tenancy-rules-monitoring-rules-edit"
```

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9093/api/v1/alerts?namespace=test-thanos-querier-tenancy-rules-monitoring-rules-edit"
```

Example monitoring-edit permissions

The following example exercises permissions granted by the **monitoring-edit** cluster role. The binding commands must be run by a user with the necessary privileges.

1. Create a test namespace and a service account.

```
$ oc create namespace test-thanos-querier-tenancy-rules-monitoring-edit
```

```
$ oc create serviceaccount thanos-client --namespace=test-thanos-querier-tenancy-rules-monitoring-edit
```

2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-thanos-querier-tenancy-rules-monitoring-edit \
  --namespace=test-thanos-querier-tenancy-rules-monitoring-edit \
  --clusterrole=monitoring-edit \
  --serviceaccount=test-thanos-querier-tenancy-rules-monitoring-edit:thanos-client
```

3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token thanos-client --namespace=test-thanos-querier-tenancy-rules-monitoring-edit)
```

4. Access Thanos Querier endpoints from within the cluster. The port is not exposed externally by default.

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9093/api/v1/rules?namespace=test-thanos-querier-tenancy-rules-monitoring-edit"
```

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9093/api/v1/alerts?namespace=test-thanos-querier-tenancy-rules-monitoring-edit"
```

Example monitoring-rules-view permissions

The following example exercises permissions granted by the **monitoring-rules-view** cluster role. The binding commands must be run by a user with the necessary privileges.

1. Create a test namespace and a service account.

```
$ oc create namespace test-thanos-querier-tenancy-rules-monitoring-rules-view
```

```
$ oc create serviceaccount thanos-client --namespace=test-thanos-querier-tenancy-rules-monitoring-rules-view
```

2. Bind the role to the service account. The binding in this example is applied to a service account but can also be applied to any user.

```
$ oc create rolebinding test-thanos-querier-tenancy-rules-monitoring-rules-view \
  --namespace=test-thanos-querier-tenancy-rules-monitoring-rules-view \
  --clusterrole=monitoring-rules-view \
  --serviceaccount=test-thanos-querier-tenancy-rules-monitoring-rules-view:thanos-client
```

3. Generate a token to access the endpoints.

```
$ TOKEN=$(oc create token thanos-client --namespace=test-thanos-querier-tenancy-rules-monitoring-rules-view)
```

4. Access Thanos Querier endpoints from within the cluster. The port is not exposed externally by default.

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9093/api/v1/rules?namespace=test-thanos-querier-tenancy-rules-monitoring-rules-view"
```

```
$ curl -k -f -H "Authorization: Bearer $TOKEN" "https://thanos-querier.openshift-monitoring:9093/api/v1/alerts?namespace=test-thanos-querier-tenancy-rules-monitoring-rules-view"
```

- Port 9094 provides access to the /**metrics** endpoint only. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.15. openshift-user-workload-monitoring/thanos-ruler

Expose the Thanos Ruler web server within the cluster on the following ports:

- Port 9091 provides access to all Thanos Ruler endpoints. Granting access requires binding a user to the **cluster-monitoring-view** cluster role.
- Port 9092 provides access to the /**metrics** endpoint only. This port is for internal use, and no other usage is guaranteed.

This also exposes the gRPC endpoints on port 10901. This port is for internal use, and no other usage is guaranteed.

4.3.5.2.16. openshift-monitoring/cluster-monitoring-operator

Expose the /**metrics** and /**validate-webhook** endpoints on port 8443. This port is for internal use, and no other usage is guaranteed.

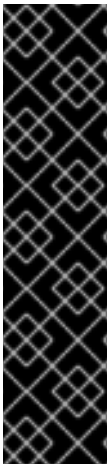
4.3.6. Additional resources

- Configuring remote write storage for monitoring of user-defined projects
- Accessing metrics as an administrator
- Accessing metrics as a developer
- Managing alerts as an Administrator
- Managing alerts as a Developer

CHAPTER 5. MANAGING ALERTS

5.1. MANAGING ALERTS AS AN ADMINISTRATOR

In OpenShift Dedicated, the Alerting UI enables you to manage alerts, silences, and alerting rules.

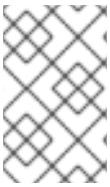


IMPORTANT

Starting with OpenShift Dedicated 4.19, the perspectives in the web console have unified. The **Developer** perspective is no longer enabled by default.

All users can interact with all OpenShift Dedicated web console features. However, if you are not the cluster owner, you might need to request permission to access certain features from the cluster owner.

You can still enable the **Developer** perspective. On the **Getting Started** pane in the web console, you can take a tour of the console, find information on setting up your cluster, view a quick start for enabling the **Developer** perspective, and follow links to explore new features and capabilities.



NOTE

The alerts, silences, and alerting rules that are available in the Alerting UI relate to the projects that you have access to. For example, if you are logged in as an administrator, you can access all alerts, silences, and alerting rules.

5.1.1. Accessing the Alerting UI

The Alerting UI is accessible in the OpenShift Dedicated web console.

- In the OpenShift Dedicated web console, go to **Observe → Alerting**. The three main pages in the Alerting UI in this perspective are the **Alerts**, **Silences**, and **Alerting rules** pages.

Additional resources

- Searching and filtering alerts, silences, and alerting rules

5.1.2. Getting information about alerts, silences, and alerting rules

The Alerting UI provides detailed information about alerts and their governing alerting rules and silences.

Prerequisites

- You have access to the cluster as a user with view permissions for the project that you are viewing alerts for.

Procedure

To obtain information about alerts:

1. In the OpenShift Dedicated web console, go to the **Observe → Alerting → Alerts** page.
2. Optional: Search for alerts by name by using the **Name** field in the search list.

3. Optional: Filter alerts by state, severity, and source by selecting filters in the **Filter** list.
4. Optional: Sort the alerts by clicking one or more of the **Name**, **Severity**, **State**, and **Source** column headers.
5. Click the name of an alert to view its **Alert details** page. The page includes a graph that illustrates alert time series data. It also provides the following information about the alert:
    - A description of the alert
    - Messages associated with the alert
    - A link to the runbook page on GitHub for the alert, if the page exists
    - Labels attached to the alert
    - A link to its governing alerting rule
    - Silences for the alert, if any exist

To obtain information about silences:

1. In the OpenShift Dedicated web console, go to the **Observe → Alerting → Silences** page.
2. Optional: Filter the silences by name using the **Search by name** field.
3. Optional: Filter silences by state by selecting filters in the **Filter** list. By default, **Active** and **Pending** filters are applied.
4. Optional: Sort the silences by clicking one or more of the **Name**, **Firing alerts**, **State**, and **Creator** column headers.
5. Select the name of a silence to view its **Silence details** page. The page includes the following details:
    - Alert specification
    - Start time
    - End time
    - Silence state
    - Number and list of firing alerts

To obtain information about alerting rules:

1. In the OpenShift Dedicated web console, go to the **Observe → Alerting → Alerting rules** page.
2. Optional: Filter alerting rules by state, severity, and source by selecting filters in the **Filter** list.
3. Optional: Sort the alerting rules by clicking one or more of the **Name**, **Severity**, **Alert state**, and **Source** column headers.
4. Select the name of an alerting rule to view its **Alerting rule details** page. The page provides the following details about the alerting rule:
    - Alerting rule name, severity, and description.

- The expression that defines the condition for firing the alert.
- The time for which the condition should be true for an alert to fire.
- A graph for each alert governed by the alerting rule, showing the value with which the alert is firing.
- A table of all alerts governed by the alerting rule.

Additional resources

- GitHub Cluster Monitoring Operator runbooks repository

5.1.3. Managing silences

You can create a silence for an alert in the OpenShift Dedicated web console. After you create silences, you can view, edit, and expire them. You also do not receive notifications about a silenced alert when the alert fires.

NOTE

When you create silences, they are replicated across Alertmanager pods. However, if you do not configure persistent storage for Alertmanager, silences might be lost. This can happen, for example, if all Alertmanager pods restart at the same time.

Additional resources

- Managing silences
- Configuring persistent storage

5.1.3.1. Silencing alerts

You can silence a specific alert or silence alerts that match a specification that you define.

Prerequisites

- If you are a cluster administrator, you have access to the cluster as a user with the **dedicated-admin** role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **cluster-monitoring-view** cluster role, which allows you to access Alertmanager.
  - The **monitoring-alertmanager-edit** role, which permits you to create and silence alerts.

Procedure

To silence a specific alert:

1. In the OpenShift Dedicated web console, go to **Observe → Alerting → Alerts**.

2. For the alert that you want to silence, click  and select **Silence alert** to open the **Silence alert** page with a default configuration for the chosen alert.

3. Optional: Change the default configuration details for the silence.

NOTE

You must add a comment before saving a silence.

4. To save the silence, click **Silence**.

To silence a set of alerts:

1. In the OpenShift Dedicated web console, go to **Observe → Alerting → Silences**.

2. Click **Create silence**.

3. On the **Create silence** page, set the schedule, duration, and label details for an alert.

NOTE

You must add a comment before saving a silence.

4. To create silences for alerts that match the labels that you entered, click **Silence**.

5.1.3.2. Editing silences

You can edit a silence, which expires the existing silence and creates a new one with the changed configuration.

Prerequisites

- If you are a cluster administrator, you have access to the cluster as a user with the **dedicated-admin** role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **cluster-monitoring-view** cluster role, which allows you to access Alertmanager.
  - The **monitoring-alertmanager-edit** role, which permits you to create and silence alerts.

Procedure

1. In the OpenShift Dedicated web console, go to **Observe → Alerting → Silences**.

2. For the silence you want to modify, click  and select **Edit silence**.
Alternatively, you can click **Actions** and select **Edit silence** on the **Silence details** page for a silence.

3. On the **Edit silence** page, make changes and click **Silence**. Doing so expires the existing silence and creates one with the updated configuration.

5.1.3.3. Expiring silences

You can expire a single silence or multiple silences. Expiring a silence deactivates it permanently.

NOTE

You cannot delete expired, silenced alerts. Expired silences older than 120 hours are garbage collected.

Prerequisites

- If you are a cluster administrator, you have access to the cluster as a user with the **dedicated-admin** role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **cluster-monitoring-view** cluster role, which allows you to access Alertmanager.
  - The **monitoring-alertmanager-edit** role, which permits you to create and silence alerts.

Procedure

1. Go to **Observe → Alerting → Silences**.
2. For the silence or silences you want to expire, select the checkbox in the corresponding row.
3. Click **Expire 1 silence** to expire a single selected silence or **Expire *<n>* silences** to expire multiple selected silences, where *<n>* is the number of silences you selected.
   Alternatively, to expire a single silence you can click **Actions** and select **Expire silence** on the **Silence details** page for a silence.

5.1.4. Managing alerting rules for user-defined projects

In OpenShift Dedicated, you can create, view, edit, and remove alerting rules for user-defined projects. Those alerting rules will trigger alerts based on the values of the chosen metrics.

Additional resources

- Creating alerting rules for user-defined projects
- Managing alerting rules for user-defined projects
- Optimizing alerting for user-defined projects

5.1.4.1. Creating alerting rules for user-defined projects

You can create alerting rules for user-defined projects. Those alerting rules will trigger alerts based on the values of the chosen metrics.

NOTE

To help users understand the impact and cause of the alert, ensure that your alerting rule contains an alert message and severity value.

Prerequisites

- You have enabled monitoring for user-defined projects.
- You are logged in as a cluster administrator or as a user that has the **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for alerting rules. In this example, it is called **example-app-alerting-rule.yaml**.
2. Add an alerting rule configuration to the YAML file. The following example creates a new alerting rule named **example-alert**. The alerting rule fires an alert when the **version** metric exposed by the sample service becomes **0**:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-alert
  namespace: ns1
spec:
  groups:
  - name: example
    rules:
    - alert: VersionAlert (1)
      for: 1m (2)
      expr: version{job="prometheus-example-app"} == 0 (3)
      labels:
        severity: warning (4)
      annotations:
        message: This is an example alert. (5)
```

(1) The name of the alerting rule you want to create.

(2) The duration for which the condition should be true before an alert is fired.

(3) The PromQL query expression that defines the new rule.

(4) The severity that alerting rule assigns to the alert.

(5) The message associated with the alert.

3. Apply the configuration file to the cluster:

```
$ oc apply -f example-app-alerting-rule.yaml
```

5.1.4.2. Creating cross-project alerting rules for user-defined projects

You can create alerting rules that are not bound to their project of origin by configuring a project in the **user-workload-monitoring-config** config map. The **PrometheusRule** objects created in these projects are then applicable to all projects.

Therefore, you can have generic alerting rules that apply to multiple user-defined projects instead of having individual **PrometheusRule** objects in each user project. You can filter which projects are included or excluded from the alerting rule by using PromQL queries in the **PrometheusRule** object.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.

NOTE

If you are a non-administrator user, you can still create cross-project alerting rules if you have the **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule. However, that project needs to be configured in the **user-workload-monitoring-config** config map under the **namespacesWithoutLabelEnforcement** property, which can be done only by cluster administrators.

- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Configure projects in which you want to create alerting rules that are not bound to a specific project:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    namespacesWithoutLabelEnforcement: [ <namespace1>, <namespace2> ] 1
    # ...
```

1 Specify one or more projects in which you want to create cross-project alerting rules. Prometheus and Thanos Ruler for user-defined monitoring do not enforce the **namespace** label in **PrometheusRule** objects created in these projects, making the **PrometheusRule** objects applicable to all projects.

3. Create a YAML file for alerting rules. In this example, it is called **example-cross-project-alerting-rule.yaml**.

4. Add an alerting rule configuration to the YAML file. The following example creates a new cross-project alerting rule called **example-security**. The alerting rule fires when a user project does not enforce the restricted pod security policy:

Example cross-project alerting rule

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-security
  namespace: ns1 (1)
spec:
  groups:
    - name: pod-security-policy
      rules:
        - alert: "ProjectNotEnforcingRestrictedPolicy" (2)
          for: 5m (3)
          expr: kube_namespace_labels{namespace!~"
(openshift|kube).*|default",label_pod_security_kubernetes_io_enforce!="restricted"} (4)
          annotations:
            message: "Restricted policy not enforced. Project {{ $labels.namespace }} does not
enforce the restricted pod security policy." (5)
          labels:
            severity: warning (6)
```

(1) Ensure that you specify the project that you defined in the **namespacesWithoutLabelEnforcement** field.

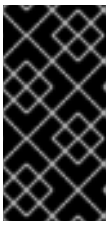
(2) The name of the alerting rule you want to create.

(3) The duration for which the condition should be true before an alert is fired.

(4) The PromQL query expression that defines the new rule. You can use label matchers on the **namespace** label to filter which projects are included or excluded from the alerting rule.

(5) The message associated with the alert.

(6) The severity that alerting rule assigns to the alert.



IMPORTANT

Ensure that you create a specific cross-project alerting rule in only one of the projects that you specified in the **namespacesWithoutLabelEnforcement** field. If you create the same cross-project alerting rule in multiple projects, it results in repeated alerts.

5. Apply the configuration file to the cluster:

```
$ oc apply -f example-cross-project-alerting-rule.yaml
```

Additional resources

- Monitoring stack architecture
- Alerting (Prometheus documentation)

5.1.4.3. Listing alerting rules for all projects in a single view

As a **dedicated-admin**, you can list alerting rules for core OpenShift Dedicated and user-defined projects together in a single view.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In the OpenShift Dedicated web console, go to **Observe → Alerting → Alerting rules**.
2. Select the **Platform** and **User** sources in the **Filter** drop-down menu.

NOTE

The **Platform** source is selected by default.

5.1.4.4. Removing alerting rules for user-defined projects

You can remove alerting rules for user-defined projects.

Prerequisites

- You have enabled monitoring for user-defined projects.
- You are logged in as a cluster administrator or as a user that has the **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule.
- You have installed the OpenShift CLI (**oc**).

Procedure

- To remove rule **<alerting_rule>** in **<namespace>**, run the following:

```
$ oc -n <namespace> delete prometheusrule <alerting_rule>
```

5.1.4.5. Disabling cross-project alerting rules for user-defined projects

Creating cross-project alerting rules for user-defined projects is enabled by default. Cluster administrators can disable the capability in the **cluster-monitoring-config** config map for the following reasons:

- To prevent user-defined monitoring from overloading the cluster monitoring stack.

- To prevent buggy alerting rules from being applied to the cluster without having to identify the rule that causes the issue.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **cluster-monitoring-config** config map in the **openshift-monitoring** project:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. In the **cluster-monitoring-config** config map, disable the option to create cross-project alerting rules by setting the **rulesWithoutLabelEnforcementAllowed** value under **data/config.yaml/userWorkload** to **false**:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    userWorkload:
      rulesWithoutLabelEnforcementAllowed: false
    # ...
```

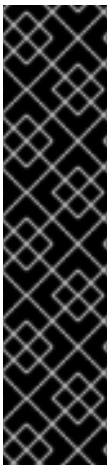
3. Save the file to apply the changes.

Additional resources

- Alertmanager (Prometheus documentation)

5.2. MANAGING ALERTS AS A DEVELOPER

In OpenShift Dedicated, the Alerting UI enables you to manage alerts, silences, and alerting rules.



IMPORTANT

Starting with OpenShift Dedicated 4.19, the perspectives in the web console have unified. The **Developer** perspective is no longer enabled by default.

All users can interact with all OpenShift Dedicated web console features. However, if you are not the cluster owner, you might need to request permission to access certain features from the cluster owner.

You can still enable the **Developer** perspective. On the **Getting Started** pane in the web console, you can take a tour of the console, find information on setting up your cluster, view a quick start for enabling the **Developer** perspective, and follow links to explore new features and capabilities.

NOTE

The alerts, silences, and alerting rules that are available in the Alerting UI relate to the projects that you have access to.

5.2.1. Accessing the Alerting UI

The Alerting UI is accessible in the OpenShift Dedicated web console.

- In the OpenShift Dedicated web console, go to **Observe → Alerting**. The three main pages in the Alerting UI in this perspective are the **Alerts**, **Silences**, and **Alerting rules** pages.

Additional resources

- Searching and filtering alerts, silences, and alerting rules

5.2.2. Getting information about alerts, silences, and alerting rules

The Alerting UI provides detailed information about alerts and their governing alerting rules and silences.

Prerequisites

- You have access to the cluster as a user with view permissions for the project that you are viewing alerts for.

Procedure

To obtain information about alerts:

1. In the OpenShift Dedicated web console, go to the **Observe → Alerting → Alerts** page.
2. Optional: Search for alerts by name by using the **Name** field in the search list.
3. Optional: Filter alerts by state, severity, and source by selecting filters in the **Filter** list.
4. Optional: Sort the alerts by clicking one or more of the **Name**, **Severity**, **State**, and **Source** column headers.
5. Click the name of an alert to view its **Alert details** page. The page includes a graph that illustrates alert time series data. It also provides the following information about the alert:
   - A description of the alert
   - Messages associated with the alert
   - A link to the runbook page on GitHub for the alert, if the page exists
   - Labels attached to the alert
   - A link to its governing alerting rule
   - Silences for the alert, if any exist

To obtain information about silences:

1. In the OpenShift Dedicated web console, go to the **Observe → Alerting → Silences** page.

2. Optional: Filter the silences by name using the **Search by name** field.
3. Optional: Filter silences by state by selecting filters in the **Filter** list. By default, **Active** and **Pending** filters are applied.
4. Optional: Sort the silences by clicking one or more of the **Name**, **Firing alerts**, **State**, and **Creator** column headers.
5. Select the name of a silence to view its **Silence details** page. The page includes the following details:
   - Alert specification
   - Start time
   - End time
   - Silence state
   - Number and list of firing alerts

To obtain information about alerting rules:

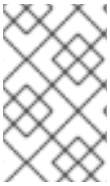
1. In the OpenShift Dedicated web console, go to the **Observe → Alerting → Alerting rules** page.
2. Optional: Filter alerting rules by state, severity, and source by selecting filters in the **Filter** list.
3. Optional: Sort the alerting rules by clicking one or more of the **Name**, **Severity**, **Alert state**, and **Source** column headers.
4. Select the name of an alerting rule to view its **Alerting rule details** page. The page provides the following details about the alerting rule:
   - Alerting rule name, severity, and description.
   - The expression that defines the condition for firing the alert.
   - The time for which the condition should be true for an alert to fire.
   - A graph for each alert governed by the alerting rule, showing the value with which the alert is firing.
   - A table of all alerts governed by the alerting rule.

Additional resources

- GitHub Cluster Monitoring Operator runbooks repository

5.2.3. Managing silences

You can create a silence for an alert in the OpenShift Dedicated web console. After you create silences, you can view, edit, and expire them. You also do not receive notifications about a silenced alert when the alert fires.



NOTE

When you create silences, they are replicated across Alertmanager pods. However, if you do not configure persistent storage for Alertmanager, silences might be lost. This can happen, for example, if all Alertmanager pods restart at the same time.

Additional resources

- Managing silences
- Configuring persistent storage

5.2.3.1. Silencing alerts

You can silence a specific alert or silence alerts that match a specification that you define.

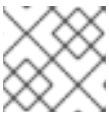
Prerequisites

- If you are a cluster administrator, you have access to the cluster as a user with the **dedicated-admin** role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **cluster-monitoring-view** cluster role, which allows you to access Alertmanager.
  - The **monitoring-alertmanager-edit** role, which permits you to create and silence alerts.

Procedure

To silence a specific alert:

1. In the OpenShift Dedicated web console, go to **Observe → Alerting → Alerts**.
2. For the alert that you want to silence, click  and select **Silence alert** to open the **Silence alert** page with a default configuration for the chosen alert.
3. Optional: Change the default configuration details for the silence.



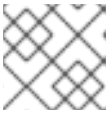
NOTE

You must add a comment before saving a silence.

4. To save the silence, click **Silence**.

To silence a set of alerts:

1. In the OpenShift Dedicated web console, go to **Observe → Alerting → Silences**.
2. Click **Create silence**.
3. On the **Create silence** page, set the schedule, duration, and label details for an alert.



NOTE

You must add a comment before saving a silence.

4. To create silences for alerts that match the labels that you entered, click **Silence**.

5.2.3.2. Editing silences

You can edit a silence, which expires the existing silence and creates a new one with the changed configuration.

Prerequisites

- If you are a cluster administrator, you have access to the cluster as a user with the **dedicated-admin** role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **cluster-monitoring-view** cluster role, which allows you to access Alertmanager.
  - The **monitoring-alertmanager-edit** role, which permits you to create and silence alerts.

Procedure

1. In the OpenShift Dedicated web console, go to **Observe → Alerting → Silences**.
2. For the silence you want to modify, click  and select **Edit silence**. Alternatively, you can click **Actions** and select **Edit silence** on the **Silence details** page for a silence.
3. On the **Edit silence** page, make changes and click **Silence**. Doing so expires the existing silence and creates one with the updated configuration.

5.2.3.3. Expiring silences

You can expire a single silence or multiple silences. Expiring a silence deactivates it permanently.

NOTE

You cannot delete expired, silenced alerts. Expired silences older than 120 hours are garbage collected.

Prerequisites

- If you are a cluster administrator, you have access to the cluster as a user with the **dedicated-admin** role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **cluster-monitoring-view** cluster role, which allows you to access Alertmanager.

- The **monitoring-alertmanager-edit** role, which permits you to create and silence alerts.

Procedure

1. Go to **Observe → Alerting → Silences**.
2. For the silence or silences you want to expire, select the checkbox in the corresponding row.
3. Click **Expire 1 silence** to expire a single selected silence or **Expire *<n>* silences** to expire multiple selected silences, where *<n>* is the number of silences you selected.
   Alternatively, to expire a single silence you can click **Actions** and select **Expire silence** on the **Silence details** page for a silence.

5.2.4. Managing alerting rules for user-defined projects

In OpenShift Dedicated, you can create, view, edit, and remove alerting rules for user-defined projects. Those alerting rules will trigger alerts based on the values of the chosen metrics.

Additional resources

- Creating alerting rules for user-defined projects
- Managing alerting rules for user-defined projects
- Optimizing alerting for user-defined projects

5.2.4.1. Creating alerting rules for user-defined projects

You can create alerting rules for user-defined projects. Those alerting rules will trigger alerts based on the values of the chosen metrics.

NOTE

To help users understand the impact and cause of the alert, ensure that your alerting rule contains an alert message and severity value.

Prerequisites

- You have enabled monitoring for user-defined projects.
- You are logged in as a cluster administrator or as a user that has the **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for alerting rules. In this example, it is called **example-app-alerting-rule.yaml**.
2. Add an alerting rule configuration to the YAML file. The following example creates a new alerting rule named **example-alert**. The alerting rule fires an alert when the **version** metric exposed by the sample service becomes **0**:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-alert
  namespace: ns1
spec:
  groups:
  - name: example
    rules:
    - alert: VersionAlert (1)
      for: 1m (2)
      expr: version{job="prometheus-example-app"} == 0 (3)
      labels:
        severity: warning (4)
      annotations:
        message: This is an example alert. (5)
```

1. The name of the alerting rule you want to create.
2. The duration for which the condition should be true before an alert is fired.
3. The PromQL query expression that defines the new rule.
4. The severity that alerting rule assigns to the alert.
5. The message associated with the alert.

3. Apply the configuration file to the cluster:

```
$ oc apply -f example-app-alerting-rule.yaml
```

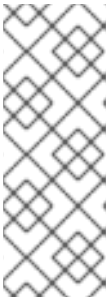
5.2.4.2. Creating cross-project alerting rules for user-defined projects

You can create alerting rules that are not bound to their project of origin by configuring a project in the **user-workload-monitoring-config** config map. The **PrometheusRule** objects created in these projects are then applicable to all projects.

Therefore, you can have generic alerting rules that apply to multiple user-defined projects instead of having individual **PrometheusRule** objects in each user project. You can filter which projects are included or excluded from the alerting rule by using PromQL queries in the **PrometheusRule** object.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.



NOTE

If you are a non-administrator user, you can still create cross-project alerting rules if you have the **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule. However, that project needs to be configured in the **user-workload-monitoring-config** config map under the **namespacesWithoutLabelEnforcement** property, which can be done only by cluster administrators.

- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Configure projects in which you want to create alerting rules that are not bound to a specific project:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    namespacesWithoutLabelEnforcement: [ <namespace1>, <namespace2> ] 1
    # ...
```

1 Specify one or more projects in which you want to create cross-project alerting rules. Prometheus and Thanos Ruler for user-defined monitoring do not enforce the **namespace** label in **PrometheusRule** objects created in these projects, making the **PrometheusRule** objects applicable to all projects.

3. Create a YAML file for alerting rules. In this example, it is called **example-cross-project-alerting-rule.yaml**.

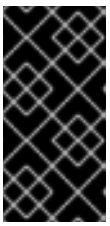
4. Add an alerting rule configuration to the YAML file. The following example creates a new cross-project alerting rule called **example-security**. The alerting rule fires when a user project does not enforce the restricted pod security policy:

Example cross-project alerting rule

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-security
  namespace: ns1 1
spec:
  groups:
    - name: pod-security-policy
      rules:
        - alert: "ProjectNotEnforcingRestrictedPolicy" 2
          for: 5m 3
          expr: kube_namespace_labels{namespace!~"
(openshift|kube).*|default",label_pod_security_kubernetes_io_enforce!="restricted"} 4
          annotations:
```

```
        message: "Restricted policy not enforced. Project {{ $labels.namespace }} does not
enforce the restricted pod security policy." 5
      labels:
        severity: warning 6
```

1. Ensure that you specify the project that you defined in the **namespacesWithoutLabelEnforcement** field.
2. The name of the alerting rule you want to create.
3. The duration for which the condition should be true before an alert is fired.
4. The PromQL query expression that defines the new rule. You can use label matchers on the **namespace** label to filter which projects are included or excluded from the alerting rule.
5. The message associated with the alert.
6. The severity that alerting rule assigns to the alert.



IMPORTANT

Ensure that you create a specific cross-project alerting rule in only one of the projects that you specified in the **namespacesWithoutLabelEnforcement** field. If you create the same cross-project alerting rule in multiple projects, it results in repeated alerts.

5. Apply the configuration file to the cluster:

```
$ oc apply -f example-cross-project-alerting-rule.yaml
```

Additional resources

- Monitoring stack architecture
- Alerting (Prometheus documentation)

5.2.4.3. Accessing alerting rules for user-defined projects

To list alerting rules for a user-defined project, you must have been assigned the **monitoring-rules-view** cluster role for the project.

Prerequisites

- You have enabled monitoring for user-defined projects.
- You are logged in as a user that has the **monitoring-rules-view** cluster role for your project.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. To list alerting rules in **<project>**:

```
$ oc -n <project> get prometheusrule
```

2. To list the configuration of an alerting rule, run the following:

```
$ oc -n <project> get prometheusrule <rule> -o yaml
```

5.2.4.4. Removing alerting rules for user-defined projects

You can remove alerting rules for user-defined projects.

Prerequisites

- You have enabled monitoring for user-defined projects.
- You are logged in as a cluster administrator or as a user that has the **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule.
- You have installed the OpenShift CLI (**oc**).

Procedure

- To remove rule **<alerting_rule>** in **<namespace>**, run the following:

```
$ oc -n <namespace> delete prometheusrule <alerting_rule>
```

Additional resources

- Alertmanager (Prometheus documentation)

CHAPTER 6. TROUBLESHOOTING MONITORING ISSUES

Find troubleshooting steps for common issues with user-defined project monitoring.

6.1. DETERMINING WHY USER-DEFINED PROJECT METRICS ARE UNAVAILABLE

If metrics are not displaying when monitoring user-defined projects, follow these steps to troubleshoot the issue.

Procedure

1. Query the metric name and verify that the project is correct:
   a. In the **Developer** perspective of the web console, click **Observe** and go to the **Metrics** tab.
   b. Select the project that you want to view metrics for in the **Project:** list.
   c. Select an existing query from the **Select query** list, or run a custom query by adding a PromQL query to the **Expression** field.
   The metrics are displayed in a chart.

   Queries must be done on a per-project basis. The metrics that are shown relate to the project that you have selected.

2. Verify that the pod that you want metrics from is actively serving metrics. Run the following **oc exec** command into a pod to target the **podIP**, **port**, and /**metrics**.

```
$ oc exec <sample_pod> -n <sample_namespace> -- curl <target_pod_IP>:<port>/metrics
```

NOTE

You must run the command on a pod that has **curl** installed.

The following example output shows a result with a valid version metric.

Example output

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
# HELP version Version information about this binary-- --:--:-- --:--:--     0
# TYPE version gauge
version{version="v0.1.0"} 1
100   102  100   102    0     0  51000      0 --:--:-- --:--:-- --:--:-- 51000
```

An invalid output indicates that there is a problem with the corresponding application.

3. If you are using a **PodMonitor** CRD, verify that the **PodMonitor** CRD is configured to point to the correct pods using label matching. For more information, see the Prometheus Operator documentation.

4. If you are using a **ServiceMonitor** CRD, and if the /**metrics** endpoint of the pod is showing metric data, follow these steps to verify the configuration:

a. Verify that the service is pointed to the correct /**metrics** endpoint. The service **labels** in this output must match the services monitor **labels** and the /**metrics** endpoint defined by the service in the subsequent steps.

```
$ oc get service
```

Example output

```
apiVersion: v1
kind: Service (1)
metadata:
  labels: (2)
    app: prometheus-example-app
  name: prometheus-example-app
  namespace: ns1
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
    name: web
  selector:
    app: prometheus-example-app
  type: ClusterIP
```

(1) Specifies that this is a service API.

(2) Specifies the labels that are being used for this service.

b. Query the **serviceIP**, **port**, and /**metrics** endpoints to see if the same metrics from the **curl** command you ran on the pod previously:

i. Run the following command to find the service IP:

```
$ oc get service -n <target_namespace>
```

ii. Query the /**metrics** endpoint:

```
$ oc exec <sample_pod> -n <sample_namespace> -- curl <service_IP>:
<port>/metrics
```

Valid metrics are returned in the following example.

Example output

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   102  100   102    0     0  51000      0 --:--:-- --:--:-- --:--:--   99k
# HELP version Version information about this binary
# TYPE version gauge
version{version="v0.1.0"} 1
```

c. Use label matching to verify that the **ServiceMonitor** object is configured to point to the

desired service. To do this, compare the **Service** object from the **oc get service** output to the **ServiceMonitor** object from the **oc get servicemonitor** output. The labels must match for the metrics to be displayed.
For example, from the previous steps, notice how the **Service** object has the **app: prometheus-example-app** label and the **ServiceMonitor** object has the same **app: prometheus-example-app** match label.

5. If everything looks valid and the metrics are still unavailable, please contact the support team for further help.

6.2. DETERMINING WHY PROMETHEUS IS CONSUMING A LOT OF DISK SPACE

Developers can create labels to define attributes for metrics in the form of key-value pairs. The number of potential key-value pairs corresponds to the number of possible values for an attribute. An attribute that has an unlimited number of potential values is called an unbound attribute. For example, a **customer_id** attribute is unbound because it has an infinite number of possible values.

Every assigned key-value pair has a unique time series. The use of many unbound attributes in labels can result in an exponential increase in the number of time series created. This can impact Prometheus performance and can consume a lot of disk space.

You can use the following measures when Prometheus consumes a lot of disk:

- **Check the time series database (TSDB) status using the Prometheus HTTP API** for more information about which labels are creating the most time series data. Doing so requires cluster administrator privileges.
- **Check the number of scrape samples** that are being collected.
- **Reduce the number of unique time series that are created** by reducing the number of unbound attributes that are assigned to user-defined metrics.

NOTE

Using attributes that are bound to a limited set of possible values reduces the number of potential key-value pair combinations.

- **Enforce limits on the number of samples that can be scraped** across user-defined projects. This requires cluster administrator privileges.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In the OpenShift Dedicated web console, go to **Observe → Metrics**.
2. Enter a Prometheus Query Language (PromQL) query in the **Expression** field. The following example queries help to identify high cardinality metrics that might result in high disk space consumption:

- By running the following query, you can identify the ten jobs that have the highest number of scrape samples:

```
topk(10, max by(namespace, job) (topk by(namespace, job) (1,
scrape_samples_post_metric_relabeling)))
```

- By running the following query, you can pinpoint time series churn by identifying the ten jobs that have created the most time series data in the last hour:

```
topk(10, sum by(namespace, job) (sum_over_time(scrape_series_added[1h])))
```

3. Investigate the number of unbound label values assigned to metrics with higher than expected scrape sample counts:

- **If the metrics relate to a user-defined project**, review the metrics key-value pairs assigned to your workload. These are implemented through Prometheus client libraries at the application level. Try to limit the number of unbound attributes referenced in your labels.

- **If the metrics relate to a core OpenShift Dedicated project**, create a Red Hat support case on the Red Hat Customer Portal .

4. Review the TSDB status using the Prometheus HTTP API by following these steps when logged in as a **dedicated-admin**:

a. Get the Prometheus API route URL by running the following command:

```
$ HOST=$(oc -n openshift-monitoring get route prometheus-k8s -
ojsonpath='{.status.ingress[].host}')
```

b. Extract an authentication token by running the following command:

```
$ TOKEN=$(oc whoami -t)
```

c. Query the TSDB status for Prometheus by running the following command:

```
$ curl -H "Authorization: Bearer $TOKEN" -k "https://$HOST/api/v1/status/tsdb"
```

Example output

```
"status": "success","data":{"headStats":{"numSeries":507473,
"numLabelPairs":19832,"chunkCount":946298,"minTime":1712253600010,
"maxTime":1712257935346},"seriesCountByMetricName":
[{"name":"etcd_request_duration_seconds_bucket","value":51840},
{"name":"apiserver_request_sli_duration_seconds_bucket","value":47718},
...
```

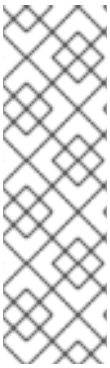
Additional resources

- Accessing monitoring APIs by using the CLI
- Setting scrape intervals, evaluation intervals, and enforced limits for user-defined projects
- Submitting a support case

6.3. RESOLVING THE KUBEPERSISTENTVOLUMEFILLINGUP ALERT FIRING FOR PROMETHEUS

As a cluster administrator, you can resolve the **KubePersistentVolumeFillingUp** alert being triggered for Prometheus.

The critical alert fires when a persistent volume (PV) claimed by a **prometheus-k8s-*** pod in the **openshift-monitoring** project has less than 3% total space remaining. This can cause Prometheus to function abnormally.



NOTE

There are two **KubePersistentVolumeFillingUp** alerts:

- **Critical alert**: The alert with the **severity="critical"** label is triggered when the mounted PV has less than 3% total space remaining.
- **Warning alert**: The alert with the **severity="warning"** label is triggered when the mounted PV has less than 15% total space remaining and is expected to fill up within four days.

To address this issue, you can remove Prometheus time-series database (TSDB) blocks to create more space for the PV.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. List the size of all TSDB blocks, sorted from oldest to newest, by running the following command:

```
$ oc debug <prometheus_k8s_pod_name> -n openshift-monitoring \
-c prometheus --image=$(oc get po -n openshift-monitoring <prometheus_k8s_pod_name> \
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') \
-- sh -c 'cd /prometheus/;du -hs $(ls -dtr */ | grep -Eo "[0-9|A-Z]{26}")'
```

Replace **<prometheus_k8s_pod_name>** with the pod mentioned in the **KubePersistentVolumeFillingUp** alert description.

Example output

```
308M    01HVKMPKQWZYWS8WVDAYQHNMW6
52M     01HVK64DTDA81799TBR9QDECEZ
102M    01HVK64DS7TRZRWF2756KHST5X
140M    01HVJS59K11FBVAPVY57K88Z11
90M     01HVH2A5Z58SKT810EM6B9AT50
152M    01HV8ZDVQMX41MKCN84S32RRZ1
354M    01HV6Q2N26BK63G4RYTST71FBF
156M    01HV664H9J9Z1FTZD73RD1563E
216M    01HTHXB60A7F239HN7S2TENPNS
```

```
104M    01HTHMGRXGS0WXA3WATRXHR36B
```

2. Identify which and how many blocks could be removed, then remove the blocks. The following example command removes the three oldest Prometheus TSDB blocks from the **prometheus-k8s-0** pod:

```
$ oc debug prometheus-k8s-0 -n openshift-monitoring \
-c prometheus --image=$(oc get po -n openshift-monitoring prometheus-k8s-0 \
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') \
-- sh -c 'ls -latr /prometheus/ | egrep -o "[0-9|A-Z]{26}" | head -3 | \
while read BLOCK; do rm -r /prometheus/$BLOCK; done'
```

3. Verify the usage of the mounted PV and ensure there is enough space available by running the following command:

```
$ oc debug <prometheus_k8s_pod_name> -n openshift-monitoring \
--image=$(oc get po -n openshift-monitoring <prometheus_k8s_pod_name> \
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') -- df -h /prometheus/
```

Replace **<prometheus_k8s_pod_name>** with the pod mentioned in the **KubePersistentVolumeFillingUp** alert description.

The following example output shows the mounted PV claimed by the **prometheus-k8s-0** pod that has 63% of space remaining:

Example output

```
Starting pod/prometheus-k8s-0-debug-j82w4 ...
Filesystem      Size  Used Avail Use% Mounted on
/dev/nvme0n1p4  40G   15G  40G  37% /prometheus

Removing debug pod ...
```

CHAPTER 7. CONFIG MAP REFERENCE FOR THE CLUSTER MONITORING OPERATOR

7.1. CLUSTER MONITORING OPERATOR CONFIGURATION REFERENCE

Parts of OpenShift Dedicated cluster monitoring are configurable. The API is accessible by setting parameters defined in various config maps.

- To configure monitoring components that monitor user-defined projects, edit the **ConfigMap** object named **user-workload-monitoring-config** in the **openshift-user-workload-monitoring** namespace. These configurations are defined by UserWorkloadConfiguration.

The configuration file is always defined under the **config.yaml** key in the config map data.

IMPORTANT

- Not all configuration parameters for the monitoring stack are exposed. Only the parameters and fields listed in this reference are supported for configuration. For more information about supported configurations, see Maintenance and support for monitoring.
- Configuring cluster monitoring is optional.
- If a configuration does not exist or is empty, default values are used.
- If the configuration has invalid YAML data, or if it contains unsupported or duplicated fields that bypassed early validation, the Cluster Monitoring Operator stops reconciling the resources and reports the **Degraded=True** status in the status conditions of the Operator.

7.2. ADDITIONALALERTMANAGERCONFIG

7.2.1. Description

The **AdditionalAlertmanagerConfig** resource defines settings for how a component communicates with additional Alertmanager instances.

7.2.2. Required

- **apiVersion**

Appears in: PrometheusK8sConfig, PrometheusRestrictedConfig, ThanosRulerConfig

Property	Type	Description
apiVersion	string	Defines the API version of Alertmanager. **v1** is no longer supported, **v2** is set as the default value.

Property	Type	Description
bearerToken	*v1.SecretKeySelector	Defines the secret key reference containing the bearer token to use when authenticating to Alertmanager.
pathPrefix	string	Defines the path prefix to add in front of the push endpoint path.
scheme	string	Defines the URL scheme to use when communicating with Alertmanager instances. Possible values are **http** or **https**. The default value is **http**.
staticConfigs	[]string	A list of statically configured Alertmanager endpoints in the form of **<hosts>:<port>**.
timeout	*string	Defines the timeout value used when sending alerts.
tlsConfig	TLSConfig	Defines the TLS settings to use for Alertmanager connections.

7.3. ALERTMANAGERMAINCONFIG

7.3.1. Description

The **AlertmanagerMainConfig** resource defines settings for the Alertmanager component in the **openshift-monitoring** namespace.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
enabled	*bool	A Boolean flag that enables or disables the main Alertmanager instance in the **openshift-monitoring** namespace. The default value is **true**.

Property	Type	Description
enableUserAlertmanagerConfig	bool	A Boolean flag that enables or disables user-defined namespaces to be selected for **AlertmanagerConfig** lookups. This setting only applies if the user workload monitoring instance of Alertmanager is not enabled. The default value is **false**.
logLevel	string	Defines the log level setting for Alertmanager. The possible values are: **error**, **warn**, **info**, **debug**. The default value is **info**.
nodeSelector	map[string]string	Defines the nodes on which the Pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Alertmanager container.
secrets	[]string	Defines a list of secrets to be mounted into Alertmanager. The secrets must reside within the same namespace as the Alertmanager object. They are added as volumes named **secret-<secret-name>** and mounted at **/etc/alertmanager/secrets/<secret-name>** in the **alertmanager** container of the Alertmanager pods.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Alertmanager. Use this setting to configure the persistent volume claim, including storage class, volume size, and name.

7.4. ALERTMANAGERUSERWORKLOADCONFIG

7.4.1. Description

The **AlertmanagerUserWorkloadConfig** resource defines the settings for the Alertmanager instance used for user-defined projects.

Appears in: UserWorkloadConfiguration

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables a dedicated instance of Alertmanager for user-defined alerts in the **openshift-user-workload-monitoring** namespace. The default value is **false**.
enableAlertmanagerConfig	bool	A Boolean flag to enable or disable user-defined namespaces to be selected for **AlertmanagerConfig** lookup. The default value is **false**.
logLevel	string	Defines the log level setting for Alertmanager for user workload monitoring. The possible values are **error**, **warn**, **info**, and **debug**. The default value is **info**.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Alertmanager container.
secrets	[]string	Defines a list of secrets to be mounted into Alertmanager. The secrets must be located within the same namespace as the Alertmanager object. They are added as volumes named **secret-<secret-name>** and mounted at **/etc/alertmanager/secrets/<secret-name>** in the **alertmanager** container of the Alertmanager pods.
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
tolerations	[]v1.Toleration	Defines tolerations for the pods.

Property	Type	Description
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Alertmanager. Use this setting to configure the persistent volume claim, including storage class, volume size and name.

7.5. CLUSTERMONITORINGCONFIGURATION

7.5.1. Description

The **ClusterMonitoringConfiguration** resource defines settings that customize the default platform monitoring stack through the **cluster-monitoring-config** config map in the **openshift-monitoring** namespace.

Property	Type	Description
alertmanagerMain	*AlertmanagerMainConfig	**AlertmanagerMainConfig** defines settings for the Alertmanager component in the **openshift-monitoring** namespace.
enableUserWorkload	*bool	**UserWorkloadEnabled** is a Boolean flag that enables monitoring for user-defined projects.
userWorkload	*UserWorkloadConfig	**UserWorkload** defines settings for the monitoring of user-defined projects.
kubeStateMetrics	*KubeStateMetricsConfig	**KubeStateMetricsConfig** defines settings for the **kube-state-metrics** agent.
metricsServer	*MetricsServerConfig	**MetricsServer** defines settings for the Metrics Server component.
prometheusK8s	*PrometheusK8sConfig	**PrometheusK8sConfig** defines settings for the Prometheus component.

Property	Type	Description
prometheusOperator	*PrometheusOperatorConfig	**PrometheusOperatorConfig** defines settings for the Prometheus Operator component.
prometheusOperatorAdmissionWebhook	*PrometheusOperatorAdmissionWebhookConfig	**PrometheusOperatorAdmissionWebhookConfig** defines settings for the admission webhook component of Prometheus Operator.
openshiftStateMetrics	*OpenShiftStateMetricsConfig	**OpenShiftMetricsConfig** defines settings for the **openshift-state-metrics** agent.
telemeterClient	*TelemeterClientConfig	**TelemeterClientConfig** defines settings for the Telemeter Client component.
thanosQuerier	*ThanosQuerierConfig	**ThanosQuerierConfig** defines settings for the Thanos Querier component.
nodeExporter	NodeExporterConfig	**NodeExporterConfig** defines settings for the **node-exporter** agent.
monitoringPlugin	*MonitoringPluginConfig	**MonitoringPluginConfig** defines settings for the monitoring **console-plugin** component.

7.6. KUBESTATEMETRICSCONFIG

7.6.1. Description

The **KubeStateMetricsConfig** resource defines settings for the **kube-state-metrics** agent.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.

Property	Type	Description
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **KubeStateMetrics** container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

7.7. METRICSSERVERCONFIG

7.7.1. Description

The **MetricsServerConfig** resource defines settings for the Metrics Server component.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
audit	*Audit	Defines the audit configuration used by the Metrics Server instance. Possible profile values are **Metadata**, **Request**, **RequestResponse**, and **None**. The default value is **Metadata**.
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
verbosity	uint8	Defines the verbosity of log messages for Metrics Server. Valid values are positive integers, values over 10 are usually unnecessary. The default value is **0**.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Metrics Server container.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

7.8. MONITORINGPLUGINCONFIG

7.8.1. Description

The **MonitoringPluginConfig** resource defines settings for the web console plugin component in the **openshift-monitoring** namespace.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **console-plugin** container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

7.9. NODEEXPORTERCOLLECTORBUDDYINFOCONFIG

7.9.1. Description

The **NodeExporterCollectorBuddyInfoConfig** resource works as an on/off switch for the **buddyinfo** collector of the **node-exporter** agent. By default, the **buddyinfo** collector is disabled.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **buddyinfo** collector.

7.10. NODEEXPORTERCOLLECTORCONFIG

7.10.1. Description

The **NodeExporterCollectorConfig** resource defines settings for individual collectors of the **node-exporter** agent.

Appears in: NodeExporterConfig

Property	Type	Description
cpufreq	NodeExporterCollectorCpufreqConfig	Defines the configuration of the **cpufreq** collector, which collects CPU frequency statistics. Disabled by default.
tcpstat	NodeExporterCollectorTcpStatConfig	Defines the configuration of the **tcpstat** collector, which collects TCP connection statistics. Disabled by default.
netdev	NodeExporterCollectorNetDevConfig	Defines the configuration of the **netdev** collector, which collects network devices statistics. Enabled by default.
netclass	NodeExporterCollectorNetClassConfig	Defines the configuration of the **netclass** collector, which collects information about network devices. Enabled by default.
buddyinfo	NodeExporterCollectorBuddyInfoConfig	Defines the configuration of the **buddyinfo** collector, which collects statistics about memory fragmentation from the **node_buddyinfo_blocks** metric. This metric collects data from /**proc**/**buddyinfo**. Disabled by default.
mountstats	NodeExporterCollectorMountStatsConfig	Defines the configuration of the **mountstats** collector, which collects statistics about NFS volume I/O activities. Disabled by default.
ksmd	NodeExporterCollectorKSMDConfig	Defines the configuration of the **ksmd** collector, which collects statistics from the kernel same-page merger daemon. Disabled by default.
processes	NodeExporterCollectorProcessesConfig	Defines the configuration of the **processes** collector, which collects statistics from processes and threads running in the system. Disabled by default.

Property	Type	Description
systemd	NodeExporterCollectorSystemdConfig	Defines the configuration of the **systemd** collector, which collects statistics on the systemd daemon and its managed services. Disabled by default.

7.11. NODEEXPORTERCOLLECTORCPUFREQCONFIG

7.11.1. Description

Use the **NodeExporterCollectorCpufreqConfig** resource to enable or disable the **cpufreq** collector of the **node-exporter** agent. By default, the **cpufreq** collector is disabled. Under certain circumstances, enabling the **cpufreq** collector increases CPU usage on machines with many cores. If you enable this collector and have machines with many cores, monitor your systems closely for excessive CPU usage.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **cpufreq** collector.

7.12. NODEEXPORTERCOLLECTORKSMDCONFIG

7.12.1. Description

Use the **NodeExporterCollectorKSMDConfig** resource to enable or disable the **ksmd** collector of the **node-exporter** agent. By default, the **ksmd** collector is disabled.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **ksmd** collector.

7.13. NODEEXPORTERCOLLECTORMOUNTSTATSCONFIG

7.13.1. Description

Use the **NodeExporterCollectorMountStatsConfig** resource to enable or disable the **mountstats** collector of the **node-exporter** agent. By default, the **mountstats** collector is disabled. If you enable the collector, the following metrics become available: **node_mountstats_nfs_read_bytes_total**,

node_mountstats_nfs_write_bytes_total, and **node_mountstats_nfs_operations_requests_total**. Be aware that these metrics can have a high cardinality. If you enable this collector, closely monitor any increases in memory usage for the **prometheus-k8s** pods.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **mountstats** collector.

7.14. NODEEXPORTERCOLLECTORNETCLASSCONFIG

7.14.1. Description

Use the **NodeExporterCollectorNetClassConfig** resource to enable or disable the **netclass** collector of the **node-exporter** agent. By default, the **netclass** collector is enabled. If you disable this collector, these metrics become unavailable: **node_network_info**, **node_network_address_assign_type**, **node_network_carrier**, **node_network_carrier_changes_total**, **node_network_carrier_up_changes_total**, **node_network_carrier_down_changes_total**, **node_network_device_id**, **node_network_dormant**, **node_network_flags**, **node_network_iface_id**, **node_network_iface_link**, **node_network_iface_link_mode**, **node_network_mtu_bytes**, **node_network_name_assign_type**, **node_network_net_dev_group**, **node_network_speed_bytes**, **node_network_transmit_queue_length**, and **node_network_protocol_type**.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **netclass** collector.
useNetlink	bool	A Boolean flag that activates the **netlink** implementation of the **netclass** collector. The default value is **true**, which activates the **netlink** mode. This implementation improves the performance of the **netclass** collector.

7.15. NODEEXPORTERCOLLECTORNETDEVCONFIG

7.15.1. Description

Use the **NodeExporterCollectorNetDevConfig** resource to enable or disable the **netdev** collector of the **node-exporter** agent. By default, the **netdev** collector is enabled. If disabled, these metrics become unavailable: **node_network_receive_bytes_total**, **node_network_receive_compressed_total**,

node_network_receive_drop_total, **node_network_receive_errs_total**, **node_network_receive_fifo_total**, **node_network_receive_frame_total**, **node_network_receive_multicast_total**, **node_network_receive_nohandler_total**, **node_network_receive_packets_total**, **node_network_transmit_bytes_total**, **node_network_transmit_carrier_total**, **node_network_transmit_colls_total**, **node_network_transmit_compressed_total**, **node_network_transmit_drop_total**, **node_network_transmit_errs_total**, **node_network_transmit_fifo_total**, and **node_network_transmit_packets_total**.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **netdev** collector.

7.16. NODEEXPORTERCOLLECTORPROCESSESCONFIG

7.16.1. Description

Use the **NodeExporterCollectorProcessesConfig** resource to enable or disable the **processes** collector of the **node-exporter** agent. If the collector is enabled, the following metrics become available: **node_processes_max_processes**, **node_processes_pids**, **node_processes_state**, **node_processes_threads**, **node_processes_threads_state**. The metric **node_processes_state** and **node_processes_threads_state** can have up to five series each, depending on the state of the processes and threads. The possible states of a process or a thread are: **D** (UNINTERRUPTABLE_SLEEP), **R** (RUNNING & RUNNABLE), **S** (INTERRUPTABLE_SLEEP), **T** (STOPPED), or **Z** (ZOMBIE). By default, the **processes** collector is disabled.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **processes** collector.

7.17. NODEEXPORTERCOLLECTORSYSTEMDCONFIG

7.17.1. Description

Use the **NodeExporterCollectorSystemdConfig** resource to enable or disable the **systemd** collector of the **node-exporter** agent. By default, the **systemd** collector is disabled. If enabled, the following metrics become available: **node_systemd_system_running**, **node_systemd_units**, **node_systemd_version**. If the unit uses a socket, it also generates the following metrics: **node_systemd_socket_accepted_connections_total**, **node_systemd_socket_current_connections**, **node_systemd_socket_refused_connections_total**. You can use the **units** parameter to select the **systemd** units to be included by the **systemd** collector. The selected units are used to generate the **node_systemd_unit_state** metric, which shows the state of

each **systemd** unit. However, this metric's cardinality might be high (at least five series per unit per node). If you enable this collector with a long list of selected units, closely monitor the **prometheus-k8s** deployment for excessive memory usage. Note that the **node_systemd_timer_last_trigger_seconds** metric is only shown if you have configured the value of the **units** parameter as **logrotate.timer**.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **systemd** collector.
units	[]string	A list of regular expression (regex) patterns that match systemd units to be included by the **systemd** collector. By default, the list is empty, so the collector exposes no metrics for systemd units.

7.18. NODEEXPORTERCOLLECTORTCPSTATCONFIG

7.18.1. Description

The **NodeExporterCollectorTcpStatConfig** resource works as an on/off switch for the **tcpstat** collector of the **node-exporter** agent. By default, the **tcpstat** collector is disabled.

Appears in: NodeExporterCollectorConfig

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the **tcpstat** collector.

7.19. NODEEXPORTERCONFIG

7.19.1. Description

The **NodeExporterConfig** resource defines settings for the **node-exporter** agent.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
collectors	NodeExporterCollectorConfig	Defines which collectors are enabled and their additional configuration parameters.

Property	Type	Description
maxProcs	uint32	The target number of CPUs on which the node-exporter's process will run. The default value is **0**, which means that node-exporter runs on all CPUs. If a kernel deadlock occurs or if performance degrades when reading from **sysfs** concurrently, you can change this value to **1**, which limits node-exporter to running on one CPU. For nodes with a high CPU count, you can set the limit to a low number, which saves resources by preventing Go routines from being scheduled to run on all CPUs. However, I/O performance degrades if the **maxProcs** value is set too low and there are many metrics to collect.
ignoredNetworkDevices	*[]string	A list of network devices, defined as regular expressions, that you want to exclude from the relevant collector configuration such as **netdev** and **netclass**. If no list is specified, the Cluster Monitoring Operator uses a predefined list of devices to be excluded to minimize the impact on memory usage. If the list is empty, no devices are excluded. If you modify this setting, monitor the **prometheus-k8s** deployment closely for excessive memory usage.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **NodeExporter** container.

7.20. OPENSHIFTSTATEMETRICSCONFIG

7.20.1. Description

The **OpenShiftStateMetricsConfig** resource defines settings for the **openshift-state-metrics** agent.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **OpenShiftStateMetrics** container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

7.21. PROMETHEUSK8SCONFIG

7.21.1. Description

The **PrometheusK8sConfig** resource defines settings for the Prometheus component.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Configures additional Alertmanager instances that receive alerts from the Prometheus component. By default, no additional Alertmanager instances are configured.
enforcedBodySizeLimit	string	Enforces a body size limit for Prometheus scraped metrics. If a scraped target's body response is larger than the limit, the scrape will fail. The following values are valid: an empty value to specify no limit, a numeric value in Prometheus size format (such as **64MB**), or the string**automatic**, which indicates that the limit will be automatically calculated based on cluster capacity. The default value is empty, which indicates no limit.

Property	Type	Description
externalLabels	ExternalLabels	Defines labels to be added to any time series or alerts when communicating with external systems such as federation, remote storage, and Alertmanager. The type is map[string]string. By default, no labels are added.
logLevel	string	Defines the log level setting for Prometheus. The possible values are: **error**, **warn**, **info**, and **debug**. The default value is **info**.
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
queryLogFile	string	Specifies the file to which PromQL queries are logged. This setting can be either a filename, in which case the queries are saved to an **emptyDir** volume at **/var/log/prometheus**, or a full path to a location where an **emptyDir** volume will be mounted and the queries saved. Writing to **/dev/stderr**, **/dev/stdout** or **/dev/null** is supported, but writing to any other **/dev/** path is not supported. Relative paths are also not supported. By default, PromQL queries are not logged.
remoteWrite	[]RemoteWriteSpec	Defines the remote write configuration, including URL, authentication, and relabeling settings.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **Prometheus** container.

Property	Type	Description
retention	string	Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: **[0-9]+(ms|s|m|h|d|w|y)** (ms = milliseconds, s= seconds,m = minutes, h = hours, d = days, w = weeks, y = years). The default value is **15d**.
retentionSize	string	Defines the maximum amount of disk space used by data blocks plus the write-ahead log (WAL). Supported values are **B**, **KB**, **KiB**, **MB**, **MiB**, **GB**, **GiB**, **TB**, **TiB**, **PB**, **PiB**, **EB**, and **EiB**. By default, no limit is defined.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.
collectionProfile	CollectionProfile	Defines the metrics collection profile that Prometheus uses to collect metrics from the platform components. Supported values are **full** or **minimal**. In the **full** profile (default), Prometheus collects all metrics that are exposed by the platform components. In the **minimal** profile, Prometheus only collects metrics necessary for the default platform alerts, recording rules, telemetry, and console dashboards.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Prometheus. Use this setting to configure the persistent volume claim, including storage class, volume size and name.

7.22. PROMETHEUSOPERATORCONFIG

7.22.1. Description

The **PrometheusOperatorConfig** resource defines settings for the Prometheus Operator component.

Appears in: ClusterMonitoringConfiguration, UserWorkloadConfiguration

Property	Type	Description
logLevel	string	Defines the log level settings for Prometheus Operator. The possible values are **error**, **warn**, **info**, and **debug**. The default value is **info**.
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **PrometheusOperator** container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

7.23. PROMETHEUSOPERATORADMISSIONWEBHOOKCONFIG

7.23.1. Description

The **PrometheusOperatorAdmissionWebhookConfig** resource defines settings for the admission webhook workload for Prometheus Operator.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **prometheus-operator-admission-webhook** container.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

7.24. PROMETHEUSRESTRICTEDCONFIG

7.24.1. Description

The **PrometheusRestrictedConfig** resource defines the settings for the Prometheus component that monitors user-defined projects.

Appears in: UserWorkloadConfiguration

Property	Type	Description
scrapeInterval	string	Configures the default interval between consecutive scrapes in case the **ServiceMonitor** or **PodMonitor** resource does not specify any value. The interval must be set between 5 seconds and 5 minutes. The value can be expressed in: seconds (for example **30s**), minutes (for example **1m**) or a mix of minutes and seconds (for example **1m30s**). The default value is **30s**.
evaluationInterval	string	Configures the default interval between rule evaluations in case the **PrometheusRule** resource does not specify any value. The interval must be set between 5 seconds and 5 minutes. The value can be expressed in: seconds (for example **30s**), minutes (for example **1m**) or a mix of minutes and seconds (for example **1m30s**). It only applies to **PrometheusRule** resources with the **openshift.io/prometheus-rule-evaluation-scope=\"leaf-prometheus\"** label. The default value is **30s**.
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Configures additional Alertmanager instances that receive alerts from the Prometheus component. By default, no additional Alertmanager instances are configured.
enforcedLabelLimit	*uint64	Specifies a per-scrape limit on the number of labels accepted for a sample. If the number of labels exceeds this limit after metric relabeling, the entire scrape is treated as failed. The default value is **0**, which means that no limit is set.

Property	Type	Description
enforcedLabelNameLengthLimit	*uint64	Specifies a per-scrape limit on the length of a label name for a sample. If the length of a label name exceeds this limit after metric relabeling, the entire scrape is treated as failed. The default value is **0**, which means that no limit is set.
enforcedLabelValueLengthLimit	*uint64	Specifies a per-scrape limit on the length of a label value for a sample. If the length of a label value exceeds this limit after metric relabeling, the entire scrape is treated as failed. The default value is **0**, which means that no limit is set.
enforcedSampleLimit	*uint64	Specifies a global limit on the number of scraped samples that will be accepted. This setting overrides the **SampleLimit** value set in any user-defined **ServiceMonitor** or **PodMonitor** object if the value is greater than **enforcedTargetLimit**. Administrators can use this setting to keep the overall number of samples under control. The default value is **0**, which means that no limit is set.
enforcedTargetLimit	*uint64	Specifies a global limit on the number of scraped targets. This setting overrides the **TargetLimit** value set in any user-defined **ServiceMonitor** or **PodMonitor** object if the value is greater than **enforcedSampleLimit**. Administrators can use this setting to keep the overall number of targets under control. The default value is **0**.

Property	Type	Description
externalLabels	ExternalLabels	Defines labels to be added to any time series or alerts when communicating with external systems such as federation, remote storage, and Alertmanager. The type is map[string]string. By default, no labels are added.
logLevel	string	Defines the log level setting for Prometheus. The possible values are **error**, **warn**, **info**, and **debug**. The default setting is **info**.
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
queryLogFile	string	Specifies the file to which PromQL queries are logged. This setting can be either a filename, in which case the queries are saved to an **emptyDir** volume at **/var/log/prometheus**, or a full path to a location where an **emptyDir** volume will be mounted and the queries saved. Writing to **/dev/stderr**, **/dev/stdout** or **/dev/null** is supported, but writing to any other **/dev/** path is not supported. Relative paths are also not supported. By default, PromQL queries are not logged.
remoteWrite	[]RemoteWriteSpec	Defines the remote write configuration, including URL, authentication, and relabeling settings.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Prometheus container.

Property	Type	Description
retention	string	Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: **[0-9]+(ms|s|m|h|d|w|y)** (ms = milliseconds, s= seconds,m = minutes, h = hours, d = days, w = weeks, y = years). The default value is **24h**.
retentionSize	string	Defines the maximum amount of disk space used by data blocks plus the write-ahead log (WAL). Supported values are **B**, **KB**, **KiB**, **MB**, **MiB**, **GB**, **GiB**, **TB**, **TiB**, **PB**, **PiB**, **EB**, and **EiB**. The default value is **nil**.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Prometheus. Use this setting to configure the storage class and size of a volume.

7.25. REMOTEWRITESPEC

7.25.1. Description

The **RemoteWriteSpec** resource defines the settings for remote write storage.

7.25.2. Required

- **url**

Appears in: PrometheusK8sConfig, PrometheusRestrictedConfig

Property	Type	Description
authorization	*monv1.SafeAuthorization	Defines the authorization settings for remote write storage.

Property	Type	Description
basicAuth	*monv1.BasicAuth	Defines Basic authentication settings for the remote write endpoint URL.
bearerTokenFile	string	Defines the file that contains the bearer token for the remote write endpoint. However, because you cannot mount secrets in a pod, in practice you can only reference the token of the service account.
headers	map[string]string	Specifies the custom HTTP headers to be sent along with each remote write request. Headers set by Prometheus cannot be overwritten.
metadataConfig	*monv1.MetadataConfig	Defines settings for sending series metadata to remote write storage.
name	string	Defines the name of the remote write queue. This name is used in metrics and logging to differentiate queues. If specified, this name must be unique.
oauth2	*monv1.OAuth2	Defines OAuth2 authentication settings for the remote write endpoint.
proxyUrl	string	Defines an optional proxy URL. If the cluster-wide proxy is enabled, it replaces the proxyUrl setting. The cluster-wide proxy supports both HTTP and HTTPS proxies, with HTTPS taking precedence.
queueConfig	*monv1.QueueConfig	Allows tuning configuration for remote write queue parameters.
remoteTimeout	string	Defines the timeout value for requests to the remote write endpoint.

Property	Type	Description
sendExemplars	*bool	Enables sending exemplars via remote write. When enabled, this setting configures Prometheus to store a maximum of 100,000 exemplars in memory. This setting only applies to user-defined monitoring and is not applicable to core platform monitoring.
sigv4	*monv1.Sigv4	Defines AWS Signature Version 4 authentication settings.
tlsConfig	*monv1.SafeTLSConfig	Defines TLS authentication settings for the remote write endpoint.
url	string	Defines the URL of the remote write endpoint to which samples will be sent.
writeRelabelConfigs	[]monv1.RelabelConfig	Defines the list of remote write relabel configurations.

7.26. TLSCONFIG

7.26.1. Description

The **TLSConfig** resource configures the settings for TLS connections.

7.26.2. Required

- **insecureSkipVerify**

Appears in: AdditionalAlertmanagerConfig

Property	Type	Description
ca	*v1.SecretKeySelector	Defines the secret key reference containing the Certificate Authority (CA) to use for the remote host.
cert	*v1.SecretKeySelector	Defines the secret key reference containing the public certificate to use for the remote host.

Property	Type	Description
key	*v1.SecretKeySelector	Defines the secret key reference containing the private key to use for the remote host.
serverName	string	Used to verify the hostname on the returned certificate.
insecureSkipVerify	bool	When set to **true**, disables the verification of the remote host's certificate and name.

7.27. TELEMETERCLIENTCONFIG

7.27.1. Description

TelemeterClientConfig defines settings for the Telemeter Client component.

7.27.2. Required

- **nodeSelector**
- **tolerations**

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the **TelemeterClient** container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

7.28. THANOSQUERIERCONFIG

7.28.1. Description

The **ThanosQuerierConfig** resource defines settings for the Thanos Querier component.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
enableRequestLogging	bool	A Boolean flag that enables or disables request logging. The default value is **false**.
logLevel	string	Defines the log level setting for Thanos Querier. The possible values are **error**, **warn**, **info**, and **debug**. The default value is **info**.
enableCORS	bool	A Boolean flag that enables setting CORS headers. The headers allow access from any origin. The default value is **false**.
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Thanos Querier container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

7.29. THANOSRULERCONFIG

7.29.1. Description

The **ThanosRulerConfig** resource defines configuration for the Thanos Ruler instance for user-defined projects.

Appears in: UserWorkloadConfiguration

Property	Type	Description

Property	Type	Description
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Configures how the Thanos Ruler component communicates with additional Alertmanager instances. The Cluster Monitoring Operator reads the cluster-wide proxy settings and configures the appropriate proxy URL for the Alertmanager endpoints. All Alertmanager endpoints in this group are expected to use the same proxy URL. Endpoints that bypass the cluster proxy should be placed in a separate group. The default value is **nil**.
evaluationInterval	string	Configures the default interval between Prometheus rule evaluations in case the **PrometheusRule** resource does not specify any value. The interval must be set between 5 seconds and 5 minutes. The value can be expressed in: seconds (for example **30s**), minutes (for example **1m**) or a mix of minutes and seconds (for example **1m30s**). It applies to **PrometheusRule** resources without the **openshift.io/prometheus-rule-evaluation-scope=\"leaf-prometheus\"** label. The default value is **15s**.
logLevel	string	Defines the log level setting for Thanos Ruler. The possible values are **error**, **warn**, **info**, and **debug**. The default value is **info**.
nodeSelector	map[string]string	Defines the nodes on which the Pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Alertmanager container.

Property	Type	Description
retention	string	Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: **[0-9]+(ms|s|m|h|d|w|y)** (ms = milliseconds, s= seconds,m = minutes, h = hours, d = days, w = weeks, y = years). The default value is **24h**.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Thanos Ruler. Use this setting to configure the storage class and size of a volume.

7.30. USERWORKLOADCONFIG

7.30.1. Description

The **UserWorkloadConfig** resource defines settings for the monitoring of user-defined projects.

Appears in: ClusterMonitoringConfiguration

Property	Type	Description
rulesWithoutLabelEnforcementAllowed	*bool	A Boolean flag that enables or disables the ability to deploy user-defined **PrometheusRules** objects for which the **namespace** label is not enforced to the namespace of the object. Such objects should be created in a namespace configured under the **namespacesWithoutLabelEnforcement** property of the **UserWorkloadConfiguration** resource. The default value is **true**.

7.31. USERWORKLOADCONFIGURATION

7.31.1. Description

The **UserWorkloadConfiguration** resource defines the settings responsible for user-defined projects in the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** namespace. You can only enable **UserWorkloadConfiguration** after you have set **enableUserWorkload** to **true** in the **cluster-monitoring-config** config map under the **openshift-monitoring** namespace.

Property	Type	Description
alertmanager	*AlertmanagerUserWorkloadConfig	Defines the settings for the Alertmanager component in user workload monitoring.
prometheus	*PrometheusRestrictedConfig	Defines the settings for the Prometheus component in user workload monitoring.
prometheusOperator	*PrometheusOperatorConfig	Defines the settings for the Prometheus Operator component in user workload monitoring.
thanosRuler	*ThanosRulerConfig	Defines the settings for the Thanos Ruler component in user workload monitoring.
namespacesWithoutLabelEnforcement	[]string	Defines the list of namespaces for which Prometheus and Thanos Ruler in user-defined monitoring do not enforce the **namespace** label value in **PrometheusRule** objects. The **namespacesWithoutLabelEnforcement** property allows users to define recording and alerting rules that can query across multiple projects (not limited to user-defined projects) instead of deploying identical **PrometheusRule** objects in each user project. To make the resulting alerts and metrics visible to project users, the query expressions should return a **namespace** label with a non-empty value.