



# OpenShift Container Platform 4.18

## Monitoring

Configuring and using the monitoring stack in OpenShift Container Platform



# OpenShift Container Platform 4.18 Monitoring

---

Configuring and using the monitoring stack in OpenShift Container Platform

## Legal Notice

Copyright © 2025 Red Hat, Inc.

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

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

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

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

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

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

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

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

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

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

All other trademarks are the property of their respective owners.

## Abstract

Use metrics and customized alerts provided by the monitoring stack to track the health and performance of your applications running on OpenShift Container Platform clusters.

## Table of Contents

<b>CHAPTER 1. ABOUT OPENSIFT CONTAINER PLATFORM MONITORING .....</b>	<b>8</b>
1.1. ABOUT OPENSIFT CONTAINER PLATFORM MONITORING .....	8
1.2. MONITORING STACK ARCHITECTURE .....	8
1.2.1. Understanding the monitoring stack .....	8
1.2.2. Default monitoring components .....	9
1.2.2.1. Default monitoring targets .....	11
1.2.3. Components for monitoring user-defined projects .....	12
1.2.3.1. Monitoring targets for user-defined projects .....	13
1.2.4. The monitoring stack in high-availability clusters .....	13
1.2.5. Glossary of common terms for OpenShift Container Platform monitoring .....	14
1.2.6. Additional resources .....	16
1.3. UNDERSTANDING THE MONITORING STACK - KEY CONCEPTS .....	16
1.3.1. About performance and scalability .....	16
1.3.1.1. Using node selectors to move monitoring components .....	16
How node selectors work with other constraints .....	17
1.3.1.2. About pod topology spread constraints for monitoring .....	17
1.3.1.3. About specifying limits and requests for monitoring components .....	17
1.3.1.4. About metrics collection profiles .....	18
1.3.2. About storing and recording data .....	19
1.3.2.1. Retention time and size for Prometheus metrics .....	19
1.3.3. Understanding metrics .....	20
1.3.3.1. Controlling the impact of unbound metrics attributes in user-defined projects .....	21
1.3.3.2. Adding cluster ID labels to metrics .....	21
1.3.4. About monitoring dashboards .....	22
1.3.4.1. Monitoring dashboards in the Administrator perspective .....	22
1.3.4.2. Monitoring dashboards in the Developer perspective .....	23
1.3.5. Managing alerts .....	24
1.3.5.1. Managing silences .....	24
1.3.5.2. Managing alerting rules for core platform monitoring .....	24
1.3.5.3. Tips for optimizing alerting rules for core platform monitoring .....	25
1.3.5.4. Creating alerting rules for user-defined projects .....	26
1.3.5.5. Managing alerting rules for user-defined projects .....	26
1.3.5.6. Optimizing alerting for user-defined projects .....	26
1.3.5.7. Searching and filtering alerts, silences, and alerting rules .....	27
1.3.5.7.1. Understanding alert filters .....	27
1.3.5.7.2. Understanding silence filters .....	28
1.3.5.7.3. Understanding alerting rule filters .....	28
1.3.5.7.4. Searching and filtering alerts, silences, and alerting rules in the Developer perspective .....	29
1.3.6. Understanding alert routing for user-defined projects .....	29
1.3.7. Sending notifications to external systems .....	30
<b>CHAPTER 2. GETTING STARTED .....</b>	<b>31</b>
2.1. MAINTENANCE AND SUPPORT FOR MONITORING .....	31
2.1.1. Support considerations for monitoring .....	31
2.1.2. Support policy for monitoring Operators .....	32
2.1.3. Support version matrix for monitoring components .....	32
2.2. CORE PLATFORM MONITORING FIRST STEPS .....	33
2.2.1. Configuring core platform monitoring: Postinstallation steps .....	33
2.3. USER WORKLOAD MONITORING FIRST STEPS .....	34
2.4. DEVELOPER AND NON-ADMINISTRATOR STEPS .....	35

<b>CHAPTER 3. CONFIGURING CORE PLATFORM MONITORING .....</b>	<b>36</b>
3.1. PREPARING TO CONFIGURE CORE PLATFORM MONITORING STACK .....	36
3.1.1. Configurable monitoring components .....	36
3.1.2. Creating a cluster monitoring config map .....	37
3.1.3. Granting users permissions for core platform monitoring .....	38
3.1.3.1. Granting user permissions by using the web console .....	39
3.1.3.2. Granting user permissions by using the CLI .....	39
3.2. CONFIGURING PERFORMANCE AND SCALABILITY FOR CORE PLATFORM MONITORING .....	40
3.2.1. Controlling the placement and distribution of monitoring components .....	40
3.2.1.1. Moving monitoring components to different nodes .....	40
3.2.1.2. Assigning tolerations to monitoring components .....	42
3.2.2. Setting the body size limit for metrics scraping .....	43
3.2.3. Managing CPU and memory resources for monitoring components .....	44
3.2.3.1. Specifying limits and requests .....	44
3.2.4. Choosing a metrics collection profile .....	47
3.2.5. Configuring pod topology spread constraints .....	48
3.3. STORING AND RECORDING DATA FOR CORE PLATFORM MONITORING .....	50
3.3.1. Configuring persistent storage .....	50
3.3.1.1. Persistent storage prerequisites .....	50
3.3.1.2. Configuring a persistent volume claim .....	50
3.3.1.3. Resizing a persistent volume .....	52
3.3.2. Modifying retention time and size for Prometheus metrics data .....	53
3.3.3. Configuring audit logs for Metrics Server .....	55
3.3.4. Setting log levels for monitoring components .....	56
3.3.5. Enabling the query log file for Prometheus .....	57
3.3.6. Enabling query logging for Thanos Querier .....	58
3.4. CONFIGURING METRICS FOR CORE PLATFORM MONITORING .....	60
3.4.1. Configuring remote write storage .....	60
3.4.1.1. Supported remote write authentication settings .....	62
3.4.1.2. Example remote write authentication settings .....	63
3.4.1.2.1. Sample YAML for AWS Signature Version 4 authentication .....	63
3.4.1.2.2. Sample YAML for Basic authentication .....	65
3.4.1.2.3. Sample YAML for authentication with a bearer token using a Secret Object .....	65
3.4.1.2.4. Sample YAML for OAuth 2.0 authentication .....	66
3.4.1.2.5. Sample YAML for TLS client authentication .....	67
3.4.1.3. Example remote write queue configuration .....	68
3.4.2. Creating cluster ID labels for metrics .....	70
3.5. CONFIGURING ALERTS AND NOTIFICATIONS FOR CORE PLATFORM MONITORING .....	71
3.5.1. Configuring external Alertmanager instances .....	71
3.5.1.1. Disabling the local Alertmanager .....	73
3.5.2. Configuring secrets for Alertmanager .....	73
3.5.2.1. Adding a secret to the Alertmanager configuration .....	74
3.5.3. Attaching additional labels to your time series and alerts .....	75
3.5.4. Configuring alert notifications .....	76
3.5.4.1. Configuring alert routing for default platform alerts .....	77
3.5.4.2. Configuring alert routing with the OpenShift Container Platform web console .....	81
3.5.4.3. Configuring different alert receivers for default platform alerts and user-defined alerts .....	83
<b>CHAPTER 4. CONFIGURING USER WORKLOAD MONITORING .....</b>	<b>84</b>
4.1. PREPARING TO CONFIGURE THE USER WORKLOAD MONITORING STACK .....	84
4.1.1. Configurable monitoring components .....	84
4.1.2. Enabling monitoring for user-defined projects .....	85
4.1.2.1. Enabling monitoring for user-defined projects .....	85

4.1.2.2. Granting users permission to configure monitoring for user-defined projects	87
4.1.3. Enabling alert routing for user-defined projects	88
4.1.3.1. Enabling the platform Alertmanager instance for user-defined alert routing	88
4.1.3.2. Enabling a separate Alertmanager instance for user-defined alert routing	89
4.1.3.3. Granting users permission to configure alert routing for user-defined projects	90
4.1.4. Granting users permissions for monitoring for user-defined projects	90
4.1.4.1. Granting user permissions by using the web console	92
4.1.4.2. Granting user permissions by using the CLI	93
4.1.5. Excluding a user-defined project from monitoring	94
4.1.6. Disabling monitoring for user-defined projects	94
4.2. CONFIGURING PERFORMANCE AND SCALABILITY FOR USER WORKLOAD MONITORING	95
4.2.1. Controlling the placement and distribution of monitoring components	95
4.2.1.1. Moving monitoring components to different nodes	95
4.2.1.2. Assigning tolerations to monitoring components	97
4.2.2. Managing CPU and memory resources for monitoring components	98
4.2.2.1. Specifying limits and requests	98
4.2.3. Controlling the impact of unbound metrics attributes in user-defined projects	100
4.2.3.1. Setting scrape intervals, evaluation intervals, and enforced limits for user-defined projects	100
4.2.3.2. Creating scrape sample alerts	102
4.2.4. Configuring pod topology spread constraints	104
4.3. STORING AND RECORDING DATA FOR USER WORKLOAD MONITORING	105
4.3.1. Configuring persistent storage	105
4.3.1.1. Persistent storage prerequisites	106
4.3.1.2. Configuring a persistent volume claim	106
4.3.1.3. Resizing a persistent volume	108
4.3.2. Modifying retention time and size for Prometheus metrics data	110
4.3.2.1. Modifying the retention time for Thanos Ruler metrics data	111
4.3.3. Setting log levels for monitoring components	112
4.3.4. Enabling the query log file for Prometheus	113
4.4. CONFIGURING METRICS FOR USER WORKLOAD MONITORING	115
4.4.1. Configuring remote write storage	115
4.4.1.1. Supported remote write authentication settings	118
4.4.1.2. Example remote write authentication settings	119
4.4.1.2.1. Sample YAML for AWS Signature Version 4 authentication	119
4.4.1.2.2. Sample YAML for Basic authentication	120
4.4.1.2.3. Sample YAML for authentication with a bearer token using a Secret Object	121
4.4.1.2.4. Sample YAML for OAuth 2.0 authentication	122
4.4.1.2.5. Sample YAML for TLS client authentication	123
4.4.1.3. Example remote write queue configuration	124
4.4.2. Creating cluster ID labels for metrics	125
4.4.3. Setting up metrics collection for user-defined projects	127
4.4.3.1. Deploying a sample service	127
4.4.3.2. Specifying how a service is monitored	128
4.4.3.3. Example service endpoint authentication settings	130
4.4.3.3.1. Sample YAML authentication with a bearer token	130
4.4.3.3.2. Sample YAML for Basic authentication	131
4.4.3.3.3. Sample YAML authentication with OAuth 2.0	132
4.5. CONFIGURING ALERTS AND NOTIFICATIONS FOR USER WORKLOAD MONITORING	133
4.5.1. Configuring external Alertmanager instances	133
4.5.2. Configuring secrets for Alertmanager	135
4.5.2.1. Adding a secret to the Alertmanager configuration	135
4.5.3. Attaching additional labels to your time series and alerts	136
4.5.4. Configuring alert notifications	138

4.5.4.1. Configuring alert routing for user-defined projects	139
4.5.4.2. Configuring alert routing for user-defined projects with the Alertmanager secret	139
4.5.4.3. Configuring different alert receivers for default platform alerts and user-defined alerts	141
<b>CHAPTER 5. ACCESSING METRICS</b>	<b>142</b>
5.1. ACCESSING METRICS AS AN ADMINISTRATOR	142
5.1.1. Viewing a list of available metrics	142
5.1.2. Querying metrics for all projects with the OpenShift Container Platform web console	142
5.1.3. Getting detailed information about a metrics target	144
5.1.4. Reviewing monitoring dashboards as a cluster administrator	146
5.2. ACCESSING METRICS AS A DEVELOPER	147
5.2.1. Viewing a list of available metrics	147
5.2.2. Querying metrics for user-defined projects with the OpenShift Container Platform web console	148
5.2.3. Reviewing monitoring dashboards as a developer	150
5.3. ACCESSING MONITORING APIS BY USING THE CLI	151
5.3.1. About accessing monitoring web service APIs	151
5.3.2. Accessing a monitoring web service API	152
5.3.3. Querying metrics by using the federation endpoint for Prometheus	153
5.3.4. Accessing metrics from outside the cluster for custom applications	154
5.3.5. Resources reference for the Cluster Monitoring Operator	156
5.3.5.1. CMO routes resources	156
5.3.5.1.1. openshift-monitoring/alertmanager-main	156
5.3.5.1.2. openshift-monitoring/prometheus-k8s	156
5.3.5.1.3. openshift-monitoring/prometheus-k8s-federate	156
5.3.5.1.4. openshift-user-workload-monitoring/federate	156
5.3.5.1.5. openshift-monitoring/thanos-querier	157
5.3.5.1.6. openshift-user-workload-monitoring/thanos-ruler	157
5.3.5.2. CMO services resources	157
5.3.5.2.1. openshift-monitoring/prometheus-operator-admission-webhook	157
5.3.5.2.2. openshift-user-workload-monitoring/alertmanager-user-workload	157
5.3.5.2.3. openshift-monitoring/alertmanager-main	157
5.3.5.2.4. openshift-monitoring/kube-state-metrics	157
5.3.5.2.5. openshift-monitoring/metrics-server	158
5.3.5.2.6. openshift-monitoring/monitoring-plugin	158
5.3.5.2.7. openshift-monitoring/node-exporter	158
5.3.5.2.8. openshift-monitoring/openshift-state-metrics	158
5.3.5.2.9. openshift-monitoring/prometheus-k8s	158
5.3.5.2.10. openshift-user-workload-monitoring/prometheus-operator	158
5.3.5.2.11. openshift-monitoring/prometheus-operator	158
5.3.5.2.12. openshift-user-workload-monitoring/prometheus-user-workload	158
5.3.5.2.13. openshift-monitoring/telemeter-client	159
5.3.5.2.14. openshift-monitoring/thanos-querier	159
5.3.5.2.15. openshift-user-workload-monitoring/thanos-ruler	159
5.3.5.2.16. openshift-monitoring/cluster-monitoring-operator	159
5.3.6. Additional resources	159
<b>CHAPTER 6. MANAGING ALERTS</b>	<b>161</b>
6.1. MANAGING ALERTS AS AN ADMINISTRATOR	161
6.1.1. Accessing the Alerting UI from the Administrator perspective	161
6.1.2. Getting information about alerts, silences, and alerting rules from the Administrator perspective	161
6.1.3. Managing silences	163
6.1.3.1. Silencing alerts from the Administrator perspective	163
6.1.3.2. Editing silences from the Administrator perspective	164



6.1.3.3. Expiring silences from the Administrator perspective	164
6.1.4. Managing alerting rules for core platform monitoring	165
6.1.4.1. Creating new alerting rules	165
6.1.4.2. Modifying core platform alerting rules	166
6.1.5. Managing alerting rules for user-defined projects	168
6.1.5.1. Creating alerting rules for user-defined projects	168
6.1.5.2. Creating cross-project alerting rules for user-defined projects	169
6.1.5.3. Listing alerting rules for all projects in a single view	171
6.1.5.4. Removing alerting rules for user-defined projects	172
6.1.5.5. Disabling cross-project alerting rules for user-defined projects	172
6.2. MANAGING ALERTS AS A DEVELOPER	173
6.2.1. Accessing the Alerting UI from the Developer perspective	173
6.2.2. Getting information about alerts, silences, and alerting rules from the Developer perspective	173
6.2.3. Managing silences	174
6.2.3.1. Silencing alerts from the Developer perspective	175
6.2.3.2. Editing silences from the Developer perspective	176
6.2.3.3. Expiring silences from the Developer perspective	176
6.2.4. Managing alerting rules for user-defined projects	177
6.2.4.1. Creating alerting rules for user-defined projects	177
6.2.4.2. Creating cross-project alerting rules for user-defined projects	178
6.2.4.3. Accessing alerting rules for user-defined projects	180
6.2.4.4. Removing alerting rules for user-defined projects	181
<b>CHAPTER 7. TROUBLESHOOTING MONITORING ISSUES</b>	<b>182</b>
7.1. INVESTIGATING WHY USER-DEFINED PROJECT METRICS ARE UNAVAILABLE	182
7.2. DETERMINING WHY PROMETHEUS IS CONSUMING A LOT OF DISK SPACE	185
7.3. RESOLVING THE KUBEPERSISTENTVOLUMEFILLINGUP ALERT FIRING FOR PROMETHEUS	187
7.4. RESOLVING THE ALERTMANAGERRECEIVERSNOTCONFIGURED ALERT	189
<b>CHAPTER 8. CONFIG MAP REFERENCE FOR THE CLUSTER MONITORING OPERATOR</b>	<b>190</b>
8.1. CLUSTER MONITORING OPERATOR CONFIGURATION REFERENCE	190
8.2. ADDITIONALALERTMANAGERCONFIG	190
8.2.1. Description	190
8.2.2. Required	190
8.3. ALERTMANAGERMAINCONFIG	191
8.3.1. Description	191
8.4. ALERTMANAGERUSERWORKLOADCONFIG	192
8.4.1. Description	193
8.5. CLUSTERMONITORINGCONFIGURATION	194
8.5.1. Description	194
8.6. KUBESTATEMETRICSCONFIG	195
8.6.1. Description	195
8.7. METRICSSERVERCONFIG	196
8.7.1. Description	196
8.8. MONITORINGPLUGINCONFIG	196
8.8.1. Description	196
8.9. NODEEXPORTERCOLLECTORBUDDYINFOCONFIG	197
8.9.1. Description	197
8.10. NODEEXPORTERCOLLECTORCONFIG	197
8.10.1. Description	197
8.11. NODEEXPORTERCOLLECTORCPUFREQCONFIG	199
8.11.1. Description	199
8.12. NODEEXPORTERCOLLECTORKSMDCONFIG	199

8.12.1. Description	199
8.13. NODEEXPORTERCOLLECTORMOUNTSTATSCONFIG	199
8.13.1. Description	199
8.14. NODEEXPORTERCOLLECTORNETCLASSCONFIG	200
8.14.1. Description	200
8.15. NODEEXPORTERCOLLECTORNETDEVCONFIG	200
8.15.1. Description	200
8.16. NODEEXPORTERCOLLECTORPROCESSESCONFIG	201
8.16.1. Description	201
8.17. NODEEXPORTERCOLLECTORSYSTEMDCONFIG	201
8.17.1. Description	201
8.18. NODEEXPORTERCOLLECTORTCPSTATCONFIG	202
8.18.1. Description	202
8.19. NODEEXPORTERCONFIG	202
8.19.1. Description	202
8.20. OPENSIFTSTATEMETRICSCONFIG	203
8.20.1. Description	203
8.21. PROMETHEUSK8SCONFIG	204
8.21.1. Description	204
8.22. PROMETHEUSOPERATORCONFIG	206
8.22.1. Description	206
8.23. PROMETHEUSOPERATORADMISSIONWEBHOOKCONFIG	207
8.23.1. Description	207
8.24. PROMETHEUSRESTRICTEDCONFIG	207
8.24.1. Description	207
8.25. REMOTEWITESPEC	211
8.25.1. Description	211
8.25.2. Required	211
8.26. TLSCONFIG	213
8.26.1. Description	213
8.26.2. Required	213
8.27. TELEMETERCLIENTCONFIG	214
8.27.1. Description	214
8.27.2. Required	214
8.28. THANOSQUERIERCONFIG	214
8.28.1. Description	214
8.29. THANOSRULERCONFIG	215
8.29.1. Description	215
8.30. USERWORKLOADCONFIG	217
8.30.1. Description	217
8.31. USERWORKLOADCONFIGURATION	217
8.31.1. Description	217
<b>CHAPTER 9. MONITORING CLUSTERS THAT RUN ON RHOSO .....</b>	<b>219</b>
9.1. REMOTE WRITING TO AN EXTERNAL PROMETHEUS INSTANCE	219
9.2. COLLECTING CLUSTER METRICS FROM THE FEDERATION ENDPOINT	222
9.3. AVAILABLE METRICS FOR CLUSTERS THAT RUN ON RHOSO	223



# CHAPTER 1. ABOUT OPENSIFT CONTAINER PLATFORM MONITORING

## 1.1. ABOUT OPENSIFT CONTAINER PLATFORM MONITORING

OpenShift Container Platform includes a preconfigured, preinstalled, and self-updating monitoring stack that provides monitoring for core platform components. You also have the option to [enable monitoring for user-defined projects](#).

A cluster administrator can [configure the monitoring stack](#) with the supported configurations. OpenShift Container Platform delivers monitoring best practices out of the box.

A set of alerts are included by default that immediately notify administrators about issues with a cluster. Default dashboards in the OpenShift Container Platform web console include visual representations of cluster metrics to help you to quickly understand the state of your cluster. With the OpenShift Container Platform web console, you can [access metrics](#) and [manage alerts](#).

After installing OpenShift Container Platform, cluster administrators can optionally enable monitoring for user-defined projects. By using this feature, cluster administrators, developers, and other users can specify how services and pods are monitored in their own projects. As a cluster administrator, you can find answers to common problems such as user metrics unavailability and high consumption of disk space by Prometheus in [Troubleshooting monitoring issues](#).

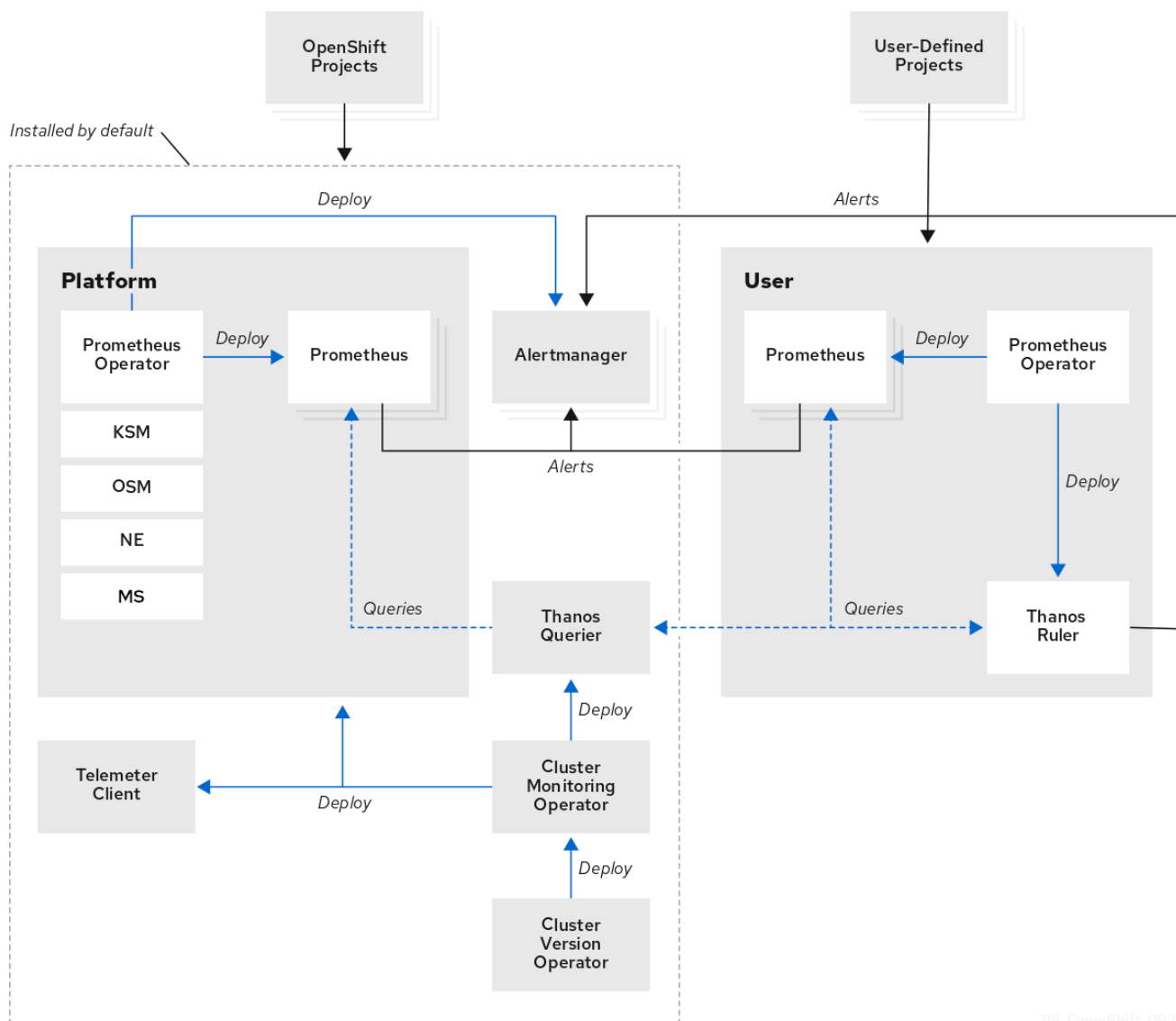
## 1.2. MONITORING STACK ARCHITECTURE

The OpenShift Container Platform monitoring stack is based on the [Prometheus](#) open source project and its wider ecosystem. The monitoring stack 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 an OpenShift Container Platform installation. This provides monitoring for core cluster components including Kubernetes services. The default monitoring stack also enables remote health monitoring for clusters. These components are illustrated in the **Installed by default** section in the following diagram.
- **Components for monitoring user-defined projects** After optionally enabling monitoring for user-defined projects, additional monitoring components are installed in the **openshift-user-workload-monitoring** project. This provides monitoring for user-defined projects. These components are illustrated in the **User** section in the following diagram.



118\_OpenShift\_092C

### 1.2.2. Default monitoring components

By default, the OpenShift Container Platform 4.18 monitoring stack includes these components:

**Table 1.1. Default monitoring stack components**

Component	Description
Cluster Monitoring Operator	The Cluster Monitoring Operator (CMO) is a central component of the monitoring stack. It deploys, manages, and automatically updates Prometheus and Alertmanager instances, Thanos Querier, Telemeter Client, and metrics targets. The CMO is deployed by the Cluster Version Operator (CVO).

Component	Description
Prometheus Operator	The Prometheus Operator (PO) in the <b>openshift-monitoring</b> project creates, configures, and manages platform Prometheus instances and Alertmanager instances. It also automatically generates monitoring target configurations based on Kubernetes label queries.
Prometheus	Prometheus is the monitoring system on which the OpenShift Container Platform monitoring stack is based. Prometheus is a time-series database and a rule evaluation engine for metrics. Prometheus sends alerts to Alertmanager for processing.
Metrics Server	The Metrics Server component (MS in the preceding diagram) collects resource metrics and exposes them in the <b>metrics.k8s.io</b> Metrics API service for use by other tools and APIs, which frees the core platform Prometheus stack from handling this functionality. Note that with the OpenShift Container Platform 4.16 release, Metrics Server replaces Prometheus Adapter.
Alertmanager	The Alertmanager service handles alerts received from Prometheus. Alertmanager is also responsible for sending the alerts to external notification systems.
kube-state-metrics agent	The kube-state-metrics exporter agent (KSM in the preceding diagram) converts Kubernetes objects to metrics that Prometheus can use.
monitoring-plugin	The monitoring-plugin dynamic plugin component deploys the monitoring pages in the <b>Observe</b> section of the OpenShift Container Platform web console. You can use Cluster Monitoring Operator config map settings to manage monitoring-plugin resources for the web console pages.
openshift-state-metrics agent	The openshift-state-metrics exporter (OSM in the preceding diagram) expands upon kube-state-metrics by adding metrics for OpenShift Container Platform-specific resources.
node-exporter agent	The node-exporter agent (NE in the preceding diagram) collects metrics about every node in a cluster. The node-exporter agent is deployed on every node.

Component	Description
Thanos Querier	Thanos Querier aggregates and optionally deduplicates core OpenShift Container Platform metrics and metrics for user-defined projects under a single, multi-tenant interface.
Telemeter Client	Telemeter Client sends a subsection of the data from platform Prometheus instances to Red Hat to facilitate Remote Health Monitoring for clusters.

All of the components in the monitoring stack are monitored by the stack and are automatically updated when OpenShift Container Platform is updated.



## NOTE

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

### 1.2.2.1. Default monitoring targets

In addition to the components of the stack itself, the default monitoring stack monitors additional platform components.

The following are examples of monitoring targets:

- 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.
- Each OpenShift Container Platform component is responsible for its monitoring configuration. For problems with the monitoring of an OpenShift Container Platform component, open a [Jira issue](#) against that component, not against the general monitoring component.

Other OpenShift Container Platform framework components might be exposing metrics as well. For details, see their respective documentation.

**Additional resources**

- [Getting detailed information about a metrics target](#)

**1.2.3. Components for monitoring user-defined projects**

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

**Table 1.2. Components for monitoring user-defined projects**

Component	Description
Prometheus Operator	The Prometheus Operator (PO) in the <b>openshift-user-workload-monitoring</b> project creates, configures, and manages Prometheus and Thanos Ruler instances in the same project.
Prometheus	Prometheus is the monitoring system through which monitoring is provided 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 Container Platform, 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.

**NOTE**

The components in the preceding table are deployed after monitoring is enabled for user-defined projects.



All of these components are monitored by the stack and are automatically updated when OpenShift Container Platform is updated.

### 1.2.3.1. Monitoring targets for user-defined projects

When monitoring is enabled for user-defined projects, you can monitor:

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

### 1.2.4. 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
- Thanos Querier
- Metrics Server
- Monitoring plugin

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

- [High-availability or single-node cluster detection and support](#)

- [Configuring persistent storage](#)
- [Configuring performance and scalability](#)

### 1.2.5. Glossary of common terms for OpenShift Container Platform monitoring

This glossary defines common terms that are used in OpenShift Container Platform 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 Container Platform 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)

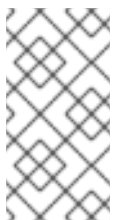
A CR is an extension of the Kubernetes API. You can create custom resources.

#### etcd

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

#### Fluentd

Fluentd is a log collector that resides on each OpenShift Container Platform node. It gathers application, infrastructure, and audit logs and forwards them to different outputs.



#### NOTE

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

#### 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 worker machine in the OpenShift Container Platform 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 an OpenShift Container Platform 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 Container Platform 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 (PO) 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 Container Platform supports many types of storage, both for on-premise and cloud providers. You can manage container storage for persistent and non-persistent data in an OpenShift Container Platform cluster.

### Thanos Ruler

The Thanos Ruler is a rule evaluation engine for Prometheus that is deployed as a separate process. In OpenShift Container Platform, 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 Container Platform 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 Container Platform.

## 1.2.6. Additional resources

- [About remote health monitoring](#)
- [Granting users permissions for monitoring for user-defined projects](#)
- [Configuring TLS security profiles](#)

## 1.3. UNDERSTANDING THE MONITORING STACK - KEY CONCEPTS

Get familiar with the OpenShift Container Platform 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 default 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.
- Set the body size limit for metrics scraping.
- Manage CPU and memory resources.
- Use metrics collection profiles.

### Additional resources

- [Configuring performance and scalability for core platform monitoring](#)
- [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 Container Platform 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 core platform monitoring components:

- Alertmanager
- kube-state-metrics
- monitoring-plugin
- node-exporter
- openshift-state-metrics
- Prometheus

- Metrics Server
- Prometheus Operator and its admission webhook service
- Telemeter Client
- Thanos Querier

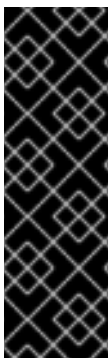
You can configure resource limits and requests for the following components that monitor user-defined projects:

- 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.

#### 1.3.1.4. About metrics collection profiles



##### IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

By default, Prometheus collects metrics exposed by all default metrics targets in OpenShift Container Platform components. However, you might want Prometheus to collect fewer metrics from a cluster in certain scenarios:

- If cluster administrators require only alert, telemetry, and console metrics and do not require other metrics to be available.
- If a cluster increases in size, and the increased size of the default metrics data collected now requires a significant increase in CPU and memory resources.

You can use a metrics collection profile to collect either the default amount of metrics data or a minimal amount of metrics data. When you collect minimal metrics data, basic monitoring features such as alerting continue to work. At the same time, the CPU and memory resources required by Prometheus decrease.

You can enable one of two metrics collection profiles:

- **full:** Prometheus collects metrics data exposed by all platform components. This setting is the default.

- **minimal:** Prometheus collects only the metrics data required for platform alerts, recording rules, telemetry, and console dashboards.

### 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 default 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:
  - Configure audit logs for Metrics Server.
  - Set log levels for monitoring.
  - Enable the query logging for Prometheus and Thanos Querier.

#### Additional resources

- [Storing and recording data for core platform monitoring](#)
- [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 Container Platform 4.18, 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 Container Platform, 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 core platform monitoring](#)
- [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.

Cluster administrators 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 Container Platform 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 Container Platform provides a set of monitoring dashboards that help you understand the state of cluster components and user-defined workloads.

#### Additional resources

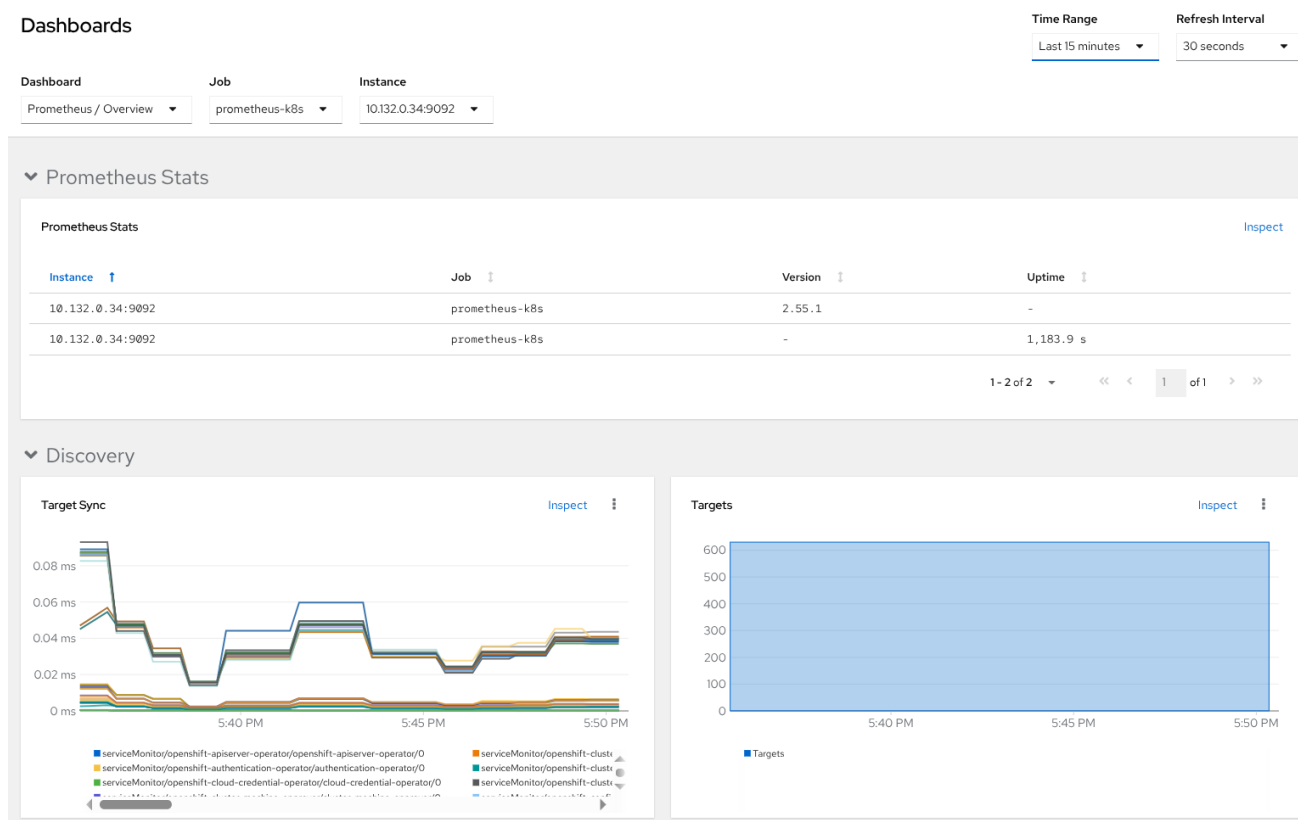
- [Reviewing monitoring dashboards as a cluster administrator](#)
- [Reviewing monitoring dashboards as a developer](#)

#### 1.3.4.1. Monitoring dashboards in the Administrator perspective

Use the **Administrator** perspective to access dashboards for the core OpenShift Container Platform 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

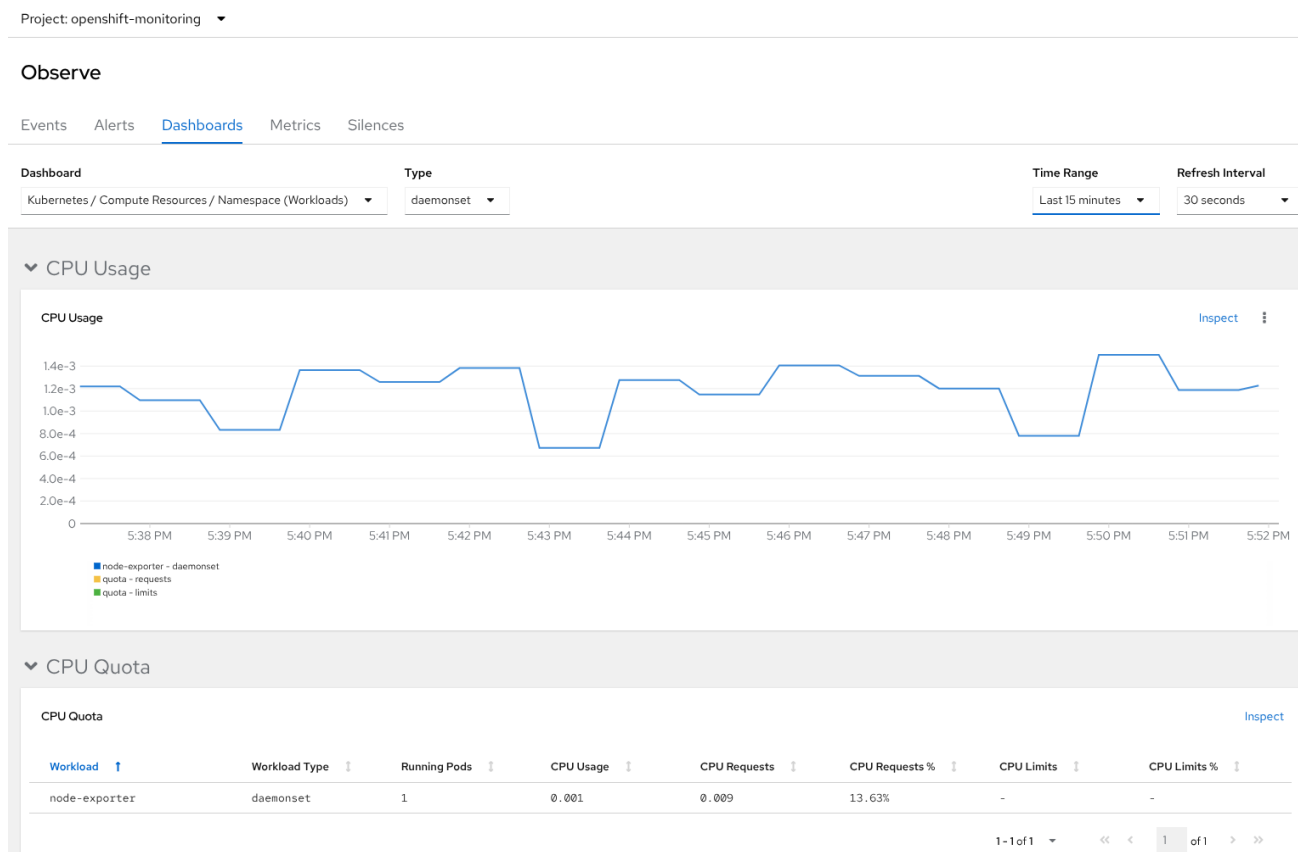
Figure 1.1. Example dashboard in the Administrator perspective



### 1.3.4.2. Monitoring dashboards in the Developer perspective

In the **Developer** perspective, you can access only the Kubernetes compute resources dashboards:

Figure 1.2. Example dashboard in the Developer perspective



### 1.3.5. Managing alerts

In the OpenShift Container Platform, 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 an OpenShift Container Platform 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 core platform monitoring](#)
- [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 Container Platform web console in both the **Administrator** and **Developer** perspectives. 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. Managing alerting rules for core platform monitoring

The OpenShift Container Platform monitoring includes a large set of default alerting rules for platform metrics. As a cluster administrator, you can customize this set of rules in two ways:

- Modify the settings for existing platform alerting rules by adjusting thresholds or by adding and modifying labels. For example, you can change the **severity** label for an alert from **warning** to **critical** to help you route and triage issues flagged by an alert.
- Define and add new custom alerting rules by constructing a query expression based on core platform metrics in the **openshift-monitoring** namespace.

### Core platform alerting rule considerations

- New alerting rules must be based on the default OpenShift Container Platform monitoring metrics.
- You must create the **AlertingRule** and **AlertRelabelConfig** objects in the **openshift-monitoring** namespace.
- You can only add and modify alerting rules. You cannot create new recording rules or modify existing recording rules.
- If you modify existing platform alerting rules by using an **AlertRelabelConfig** object, your modifications are not reflected in the Prometheus alerts API. Therefore, any dropped alerts still appear in the OpenShift Container Platform web console even though they are no longer forwarded to Alertmanager. Additionally, any modifications to alerts, such as a changed **severity** label, do not appear in the web console.

### 1.3.5.3. Tips for optimizing alerting rules for core platform monitoring

If you customize core platform alerting rules to meet your organization's specific needs, follow these guidelines to help ensure that the customized rules are efficient and effective.

- **Minimize the number of new rules** Create only rules that are essential to your specific requirements. By minimizing the number of rules, you create a more manageable and focused alerting system in your monitoring environment.
- **Focus on symptoms rather than causes** Create rules that notify users of symptoms instead of underlying causes. This approach ensures that users are promptly notified of a relevant symptom so that they can investigate the root cause after an alert has triggered. This tactic also significantly reduces the overall number of rules you need to create.
- **Plan and assess your needs before implementing changes** First, decide what symptoms are important and what actions you want users to take if these symptoms occur. Then, assess existing rules and decide if you can modify any of them to meet your needs instead of creating entirely new rules for each symptom. By modifying existing rules and creating new ones judiciously, you help to streamline your alerting system.
- **Provide clear alert messaging** When you create alert messages, describe the symptom, possible causes, and recommended actions. Include unambiguous, concise explanations along with troubleshooting steps or links to more information. Doing so helps users quickly assess the situation and respond appropriately.
- **Include severity levels** Assign severity levels to your rules to indicate how a user needs to react when a symptom occurs and triggers an alert. For example, classifying an alert as **Critical** signals that an individual or a critical response team needs to respond immediately. By defining severity

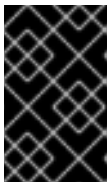
levels, you help users know how to respond to an alert and help ensure that the most urgent issues receive prompt attention.

#### 1.3.5.4. Creating alerting rules for user-defined projects

In OpenShift Container Platform, 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.

#### 1.3.5.5. Managing alerting rules for user-defined projects

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

##### Alerting rule considerations

- The default alerting rules are used specifically for the OpenShift Container Platform 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.

#### 1.3.5.6. 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.

### 1.3.5.7. 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.

#### 1.3.5.7.1. Understanding alert filters

In the **Administrator** perspective, the **Alerts** page in the Alerting UI provides details about alerts relating to default OpenShift Container Platform 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 Container Platform projects. These projects provide core OpenShift Container Platform 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.

#### 1.3.5.7.2. Understanding silence filters

In the **Administrator** perspective, the **Silences** page in the Alerting UI provides details about silences applied to alerts in default OpenShift Container Platform 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.

#### 1.3.5.7.3. Understanding alerting rule filters

In the **Administrator** perspective, the **Alerting rules** page in the Alerting UI provides details about alerting rules relating to default OpenShift Container Platform 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



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 Container Platform projects. These projects provide core OpenShift Container Platform 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.

#### 1.3.5.7.4. Searching and filtering alerts, silences, and alerting rules in the Developer perspective

In the **Developer** perspective, the **Alerts** page in the Alerting UI provides a combined view of alerts and silences relating to the selected project. A link to the governing alerting rule is provided for each displayed alert.

In this view, you can filter by alert state and severity. By default, all alerts in the selected project are displayed if you have permission to access the project. These filters are the same as those described for the **Administrator** perspective.

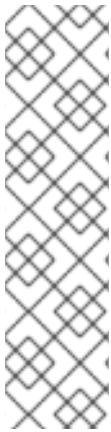
### 1.3.6. Understanding alert routing for user-defined projects

As a cluster administrator, 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 the default Alertmanager instance or, if enabled, an optional 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 as follows:

- To the **alertmanager-main** pods in the **openshift-monitoring** namespace if using the default platform Alertmanager instance.
- To the **alertmanager-user-workload** pods in the **openshift-user-workload-monitoring** namespace if you have enabled a separate instance of Alertmanager for user-defined projects.



## 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 Container Platform 4.18, 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 Container Platform 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 Container Platform 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 core platform monitoring](#)
- [Configuring alert notifications for user workload monitoring](#)

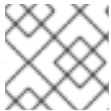
## CHAPTER 2. GETTING STARTED

### 2.1. MAINTENANCE AND SUPPORT FOR MONITORING

Not all configuration options for the monitoring stack are exposed. The only supported way of configuring OpenShift Container Platform 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.

#### 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:

- **Creating additional `ServiceMonitor`, `PodMonitor`, and `PrometheusRule` objects in the `openshift-*` and `kube-*` projects.**
- **Modifying any resources or objects deployed in the `openshift-monitoring` or `openshift-user-workload-monitoring` projects.** The resources created by the OpenShift Container Platform monitoring stack are not meant to be used by any other resources, as there are no guarantees about their backward compatibility.



##### NOTE

The Alertmanager configuration is deployed as the **`alertmanager-main`** secret resource in the **`openshift-monitoring`** namespace. If you have enabled a separate Alertmanager instance for user-defined alert routing, an Alertmanager configuration is also deployed as the **`alertmanager-user-workload`** secret resource in the **`openshift-user-workload-monitoring`** namespace. To configure additional routes for any instance of Alertmanager, you need to decode, modify, and then encode that secret. This procedure is a supported exception to the preceding statement.

- **Modifying resources of the stack.** The OpenShift Container Platform monitoring stack ensures its resources are always in the state it expects them to be. If they are modified, the stack will reset them.
- **Deploying user-defined workloads to `openshift-*`, and `kube-*` projects.** These projects are reserved for Red Hat provided components and they should not be used for user-defined workloads.
- **Enabling symptom based monitoring by using the `Probe` custom resource definition (CRD) in Prometheus Operator.**

- **Manually deploying monitoring resources into namespaces that have the `openshift.io/cluster-monitoring: "true"` label.**
- **Adding the `openshift.io/cluster-monitoring: "true"` label to namespaces.** This label is reserved only for the namespaces with core OpenShift Container Platform components and Red Hat certified components.
- **Installing custom Prometheus instances on OpenShift Container Platform.** A custom instance is a Prometheus custom resource (CR) managed by the Prometheus Operator.

### 2.1.2. Support policy for monitoring Operators

Monitoring Operators ensure that OpenShift Container Platform monitoring resources function as designed and tested. If Cluster Version Operator (CVO) control of an Operator is overridden, the Operator does not respond to configuration changes, reconcile the intended state of cluster objects, or receive updates.

While overriding CVO control for an Operator can be helpful during debugging, this is unsupported and the cluster administrator assumes full control of the individual component configurations and upgrades.

#### Overriding the Cluster Version Operator

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

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



#### WARNING

Setting a CVO override puts the entire cluster in an unsupported state and prevents the monitoring stack from being reconciled to its intended state. This impacts the reliability features built into Operators and prevents updates from being received. Reported issues must be reproduced after removing any overrides for support to proceed.

### 2.1.3. Support version matrix for monitoring components

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

Table 2.1. OpenShift Container Platform and component versions

OpenShift Container Platform	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 Container Platform.

## 2.2. CORE PLATFORM MONITORING FIRST STEPS

After OpenShift Container Platform is installed, core platform monitoring components immediately begin collecting metrics, which you can query and view. The default in-cluster monitoring stack includes the core platform Prometheus instance that collects metrics from your cluster and the core Alertmanager instance that routes alerts, among other components. Depending on who will use the monitoring stack and for what purposes, as a cluster administrator, you can further configure these monitoring components to suit the needs of different users in various scenarios.

### 2.2.1. Configuring core platform monitoring: Postinstallation steps

After OpenShift Container Platform is installed, cluster administrators typically configure core platform monitoring to suit their needs. These activities include setting up storage and configuring options for Prometheus, Alertmanager, and other monitoring components.

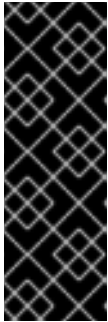


## NOTE

By default, in a newly installed OpenShift Container Platform system, users can query and view collected metrics. You need only configure an alert receiver if you want users to receive alert notifications. Any other configuration options listed here are optional.

- Create the **cluster-monitoring-config ConfigMap** object if it does not exist.

- [Configure notifications for default platform alerts](#) so that Alertmanager can send alerts to an external notification system such as email, Slack, or PagerDuty.
- For shorter term data retention, [configure persistent storage](#) for Prometheus and Alertmanager to store metrics and alert data. Specify the metrics data retention parameters for Prometheus and Thanos Ruler.



#### IMPORTANT

- In multi-node clusters, you must configure persistent storage for Prometheus, Alertmanager, and Thanos Ruler to ensure high availability.
- By default, in a newly installed OpenShift Container Platform system, the monitoring **ClusterOperator** resource reports a **PrometheusDataPersistenceNotConfigured** status message to remind you that storage is not configured.

- For longer term data retention, [configure the remote write feature](#) to enable Prometheus to send ingested metrics to remote systems for storage.



#### IMPORTANT

Be sure to [add cluster ID labels to metrics](#) for use with your remote write storage configuration.

- [Grant monitoring cluster roles](#) to any non-administrator users that need to access certain monitoring features.
- [Assign tolerations](#) to monitoring stack components so that administrators can move them to tainted nodes.
- [Set the body size limit](#) for metrics collection to help avoid situations in which Prometheus consumes excessive amounts of memory when scraped targets return a response that contains a large amount of data.
- [Modify or create alerting rules](#) for your cluster. These rules specify the conditions that trigger alerts, such as high CPU or memory usage, network latency, and so forth.
- [Specify resource limits and requests for monitoring components](#) to ensure that the containers that run monitoring components have enough CPU and memory resources.

With the monitoring stack configured to suit your needs, Prometheus collects metrics from the specified services and stores these metrics according to your settings. You can go to the **Observe** pages in the OpenShift Container Platform web console to view and query collected metrics, manage alerts, identify performance bottlenecks, and scale resources as needed:

- [View dashboards](#) to visualize collected metrics, troubleshoot alerts, and monitor other information about your cluster.
- [Query collected metrics](#) by creating PromQL queries or using predefined queries.

## 2.3. USER WORKLOAD MONITORING FIRST STEPS

As a cluster administrator, you can optionally enable monitoring for user-defined projects in addition to core platform monitoring. Non-administrator users such as developers can then monitor their own projects outside of core platform monitoring.

Cluster administrators typically complete the following activities to configure user-defined projects so that users can view collected metrics, query these metrics, and receive alerts for their own projects:

- [Enable user workload monitoring](#).
- [Grant non-administrator users permissions to monitor user-defined projects](#) by assigning the **monitoring-rules-view**, **monitoring-rules-edit**, or **monitoring-edit** cluster roles.
- [Assign the \*\*user-workload-monitoring-config-edit\*\* role](#) to grant non-administrator users permission to configure user-defined projects.
- [Enable alert routing for user-defined projects](#) so that developers and other users can configure custom alerts and alert routing for their projects.
- If needed, configure alert routing for user-defined projects to [use an optional Alertmanager instance dedicated for use only by user-defined projects](#).
- [Configure notifications for user-defined alerts](#).
- If you use the platform Alertmanager instance for user-defined alert routing, [configure different alert receivers](#) for default platform alerts and user-defined alerts.

## 2.4. DEVELOPER AND NON-ADMINISTRATOR STEPS

After monitoring for user-defined projects is enabled and configured, developers and other non-administrator users can then perform the following activities to set up and use monitoring for their own projects:

- [Deploy and monitor services](#).
- [Create and manage alerting rules](#).
- [Receive and manage alerts](#) for your projects.
- If granted the **alert-routing-edit** cluster role, [configure alert routing](#).
- [View dashboards](#) by using the OpenShift Container Platform web console.
- [Query the collected metrics](#) by creating PromQL queries or using predefined queries.

## CHAPTER 3. CONFIGURING CORE PLATFORM MONITORING

### 3.1. PREPARING TO CONFIGURE CORE PLATFORM MONITORING STACK

The OpenShift Container Platform installation program provides only a low number of configuration options before installation. Configuring most OpenShift Container Platform framework components, including the cluster monitoring stack, happens after the installation.

This section explains which monitoring components can be configured and how to prepare for configuring the 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.
- The monitoring stack imposes additional resource requirements. Consult the computing resources recommendations in [Scaling the Cluster Monitoring Operator](#) and verify that you have sufficient resources.

#### 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 **cluster-monitoring-config** config map.

**Table 3.1. Configurable core platform monitoring components**

Component	cluster-monitoring-config config map key
Prometheus Operator	<b>prometheusOperator</b>
Prometheus	<b>prometheusK8s</b>
Alertmanager	<b>alertmanagerMain</b>
Thanos Querier	<b>thanosQuerier</b>
kube-state-metrics	<b>kubeStateMetrics</b>
monitoring-plugin	<b>monitoringPlugin</b>
openshift-state-metrics	<b>openshiftStateMetrics</b>
Telemeter Client	<b>telemeterClient</b>
Metrics Server	<b>metricsServer</b>



**WARNING**

Different configuration changes to the **ConfigMap** object result in different outcomes:

- The pods are not redeployed. Therefore, there is no service outage.
- The affected pods are redeployed:
  - For single-node clusters, this results in temporary service outage.
  - For multi-node clusters, because of high-availability, the affected pods are gradually rolled out and the monitoring stack remains available.
  - Configuring and resizing a persistent volume always results in a service outage, regardless of high availability.

Each procedure that requires a change in the config map includes its expected outcome.

### 3.1.2. Creating a cluster monitoring config map

You can configure the core OpenShift Container Platform monitoring components by creating and updating the **cluster-monitoring-config** config map in the **openshift-monitoring** project. The Cluster Monitoring Operator (CMO) then configures the core components of the monitoring stack.

#### Prerequisites

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

#### Procedure

1. Check whether the **cluster-monitoring-config ConfigMap** object exists:

```
$ oc -n openshift-monitoring get configmap cluster-monitoring-config
```

2. If the **ConfigMap** object does not exist:
  - a. Create the following YAML manifest. In this example the file is called **cluster-monitoring-config.yaml**:

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

- b. Apply the configuration to create the **ConfigMap** object:

```
$ oc apply -f cluster-monitoring-config.yaml
```

### 3.1.3. Granting users permissions for core platform monitoring

As a cluster administrator, you can monitor all core OpenShift Container Platform and user-defined projects.

You can also grant developers and other users different permissions for core platform monitoring. You can grant the permissions by assigning one of the following monitoring roles or cluster roles:

Name	Description	Project
<b>cluster-monitoring-metrics-api</b>	Users with this role have the ability to access Thanos Querier API endpoints. Additionally, it grants access to the core platform Prometheus API and user-defined Thanos Ruler API endpoints.	<b>openshift-monitoring</b>
<b>cluster-monitoring-operator-alert-customization</b>	Users with this role can manage <b>AlertingRule</b> and <b>AlertRelabelConfig</b> resources for core platform monitoring. These permissions are required for the alert customization feature.	<b>openshift-monitoring</b>
<b>monitoring-alertmanager-edit</b>	Users with this role can manage the Alertmanager API for core platform monitoring. They can also manage alert silences in the <b>Administrator</b> perspective of the OpenShift Container Platform web console.	<b>openshift-monitoring</b>
<b>monitoring-alertmanager-view</b>	Users with this role can monitor the Alertmanager API for core platform monitoring. They can also view alert silences in the <b>Administrator</b> perspective of the OpenShift Container Platform web console.	<b>openshift-monitoring</b>
<b>cluster-monitoring-view</b>	Users with this cluster role have the same access rights as <b>cluster-monitoring-metrics-api</b> role, with additional permissions, providing access to the <b>/federate</b> endpoint for the user-defined Prometheus.	Must be bound with <b>ClusterRoleBinding</b> to gain access to the <b>/federate</b> endpoint for the user-defined Prometheus.

## Additional resources

- [Resources reference for the Cluster Monitoring Operator](#)
- [CMO services resources](#)

### 3.1.3.1. Granting user permissions by using the web console

You can grant users permissions for the **openshift-monitoring** project or their own projects, by using the OpenShift Container Platform web console.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- The user account that you are assigning the role to already exists.

#### Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform web console, go to **User Management** → **RoleBindings** → **Create binding**.
2. In the **Binding Type** section, select the **Namespace Role Binding** type.
3. In the **Name** field, enter a name for the role binding.
4. In the **Namespace** field, select the project where you want to grant the access.



#### IMPORTANT

The monitoring role or cluster role permissions that you grant to a user by using this procedure apply only to the project that you select in the **Namespace** field.

5. Select a monitoring role or cluster role from the **Role Name** list.
6. In the **Subject** section, select **User**.
7. In the **Subject Name** field, enter the name of the user.
8. Select **Create** to apply the role binding.

### 3.1.3.2. Granting user permissions by using the CLI

You can grant users permissions for the **openshift-monitoring** project or their own projects, by using the OpenShift CLI (**oc**).



#### IMPORTANT

Whichever role or cluster role you choose, you must bind it against a specific project as a cluster administrator.

#### Prerequisites

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

- The user account that you are assigning the role to already exists.
- You have installed the OpenShift CLI (**oc**).

### Procedure

- To assign a monitoring role to a user for a project, enter the following command:

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

- 1 Substitute **<role>** with the wanted monitoring role, **<user>** with the user to whom you want to assign the role, and **<namespace>** with the project where you want to grant the access.

- To assign a monitoring cluster role to a user for a project, enter the following command:

```
$ oc adm policy add-cluster-role-to-user <cluster-role> <user> -n <namespace> 1
```

- 1 Substitute **<cluster-role>** with the wanted monitoring cluster role, **<user>** with the user to whom you want to assign the cluster role, and **<namespace>** with the project where you want to grant the access.

## 3.2. CONFIGURING PERFORMANCE AND SCALABILITY FOR CORE PLATFORM 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.

- [About performance and scalability](#)

### 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

To specify the nodes in your cluster on which monitoring stack components will run, configure the **nodeSelector** constraint for the components in the **cluster-monitoring-config** config map to match labels assigned to the nodes.



## NOTE

You cannot add a node selector constraint directly to an existing scheduled pod.

## Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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 **cluster-monitoring-config ConfigMap** object in the **openshift-monitoring** project:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-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**

- [Preparing to configure core platform monitoring stack](#)
- [Understanding how to update labels on nodes](#)
- [Placing pods on specific nodes using node selectors](#)
- [nodeSelector](#) (Kubernetes documentation)

**3.2.1.2. Assigning tolerations to monitoring components**

You can assign tolerations to any of the monitoring stack components to enable moving them to tainted nodes.

**Prerequisites**

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Specify **tolerations** for the component:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-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 **alertmanagerMain** component to tolerate the example taint:

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

- [Preparing to configure core platform monitoring stack](#)
- [Controlling pod placement using node taints](#)
- [Taints and Tolerations](#) (Kubernetes documentation)

### 3.2.2. Setting the body size limit for metrics scraping

By default, no limit exists for the uncompressed body size for data returned from scraped metrics targets. You can set a body size limit to help avoid situations in which Prometheus consumes excessive amounts of memory when scraped targets return a response that contains a large amount of data. In addition, by setting a body size limit, you can reduce the impact that a malicious target might have on Prometheus and on the cluster as a whole.

After you set a value for **enforcedBodySizeLimit**, the alert **PrometheusScrapeBodySizeLimitHit** fires when at least one Prometheus scrape target replies with a response body larger than the configured value.



#### NOTE

If metrics data scraped from a target has an uncompressed body size exceeding the configured size limit, the scrape fails. Prometheus then considers this target to be down and sets its **up** metric value to **0**, which can trigger the **TargetDown** alert.

#### Prerequisites

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

#### Procedure

1. Edit the **cluster-monitoring-config ConfigMap** object in the **openshift-monitoring** namespace:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. Add a value for **enforcedBodySizeLimit** to **data/config.yaml/prometheusK8s** to limit the body size that can be accepted per target scrape:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |-
    prometheusK8s:
      enforcedBodySizeLimit: 40MB 1
```

- 1 Specify the maximum body size for scraped metrics targets. This **enforcedBodySizeLimit** example limits the uncompressed size per target scrape to 40 megabytes. Valid numeric values use the Prometheus data size format: B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes), TB (terabytes), PB (petabytes), and EB (exabytes). The default value is **0**, which specifies no limit. You can also set the value to **automatic** to calculate the limit automatically based on cluster capacity.

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

### Additional resources

- [scrape\\_config configuration](#) (Prometheus documentation)

## 3.2.3. 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 core platform monitoring components in the **openshift-monitoring** namespace.

### 3.2.3.1. Specifying limits and requests

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

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **ConfigMap** object named **cluster-monitoring-config**.
- 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. 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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    prometheusK8s:
      resources:
        limits:
          cpu: 500m
          memory: 3Gi
        requests:
          cpu: 200m
          memory: 500Mi
    thanosQuerier:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    prometheusOperator:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    metricsServer:
      resources:
        requests:
```

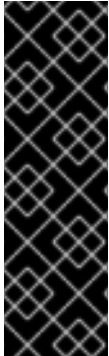
```
    cpu: 10m
    memory: 50Mi
  limits:
    cpu: 50m
    memory: 500Mi
  kubeStateMetrics:
  resources:
  limits:
    cpu: 500m
    memory: 1Gi
  requests:
    cpu: 200m
    memory: 500Mi
  telemetryClient:
  resources:
  limits:
    cpu: 500m
    memory: 1Gi
  requests:
    cpu: 200m
    memory: 500Mi
  openshiftStateMetrics:
  resources:
  limits:
    cpu: 500m
    memory: 1Gi
  requests:
    cpu: 200m
    memory: 500Mi
  nodeExporter:
  resources:
  limits:
    cpu: 50m
    memory: 150Mi
  requests:
    cpu: 20m
    memory: 50Mi
  monitoringPlugin:
  resources:
  limits:
    cpu: 500m
    memory: 1Gi
  requests:
    cpu: 200m
    memory: 500Mi
  prometheusOperatorAdmissionWebhook:
  resources:
  limits:
    cpu: 50m
    memory: 100Mi
  requests:
    cpu: 20m
    memory: 50Mi
```

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](#)
- [Kubernetes requests and limits documentation](#) (Kubernetes documentation)

### 3.2.4. Choosing a metrics collection profile



#### IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

To choose a metrics collection profile for core OpenShift Container Platform monitoring components, edit the **cluster-monitoring-config ConfigMap** object.

#### Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have enabled Technology Preview features by using the **FeatureGate** custom resource (CR).
- You have created the **cluster-monitoring-config ConfigMap** object.
- You have access to the cluster as a user with the **cluster-admin** cluster role.

#### Procedure

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

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. Add the metrics collection profile setting under **data/config.yaml/prometheusK8s**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      collectionProfile: <metrics_collection_profile_name> 1
```

- 1 The name of the metrics collection profile. The available values are **full** or **minimal**. If you do not specify a value or if the **collectionProfile** key name does not exist in the config map, the default setting of **full** is used.

The following example sets the metrics collection profile to **minimal** for the core platform instance of Prometheus:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      collectionProfile: minimal
```

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

### Additional resources

- [About metrics collection profiles](#)
- [Viewing a list of available metrics](#)
- [Enabling features using feature gates](#)

### 3.2.5. Configuring pod topology spread constraints

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.

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

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Add the following settings under the **data/config.yaml** field to configure pod topology spread constraints:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
```

```

namespace: openshift-monitoring
data:
  config.yaml: |
    <component>: ❶
    topologySpreadConstraints:
      - maxSkew: <n> ❷
        topologyKey: <key> ❸
        whenUnsatisfiable: <value> ❹
        labelSelector: ❺
          <match_option>

```

- ❶ Specify a name of the component for which you want to set up pod topology spread constraints.
- ❷ Specify a numeric value for **maxSkew**, which defines the degree to which pods are allowed to be unevenly distributed.
- ❸ 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.
- ❹ 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.
- ❺ 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 Prometheus

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: monitoring
          whenUnsatisfiable: DoNotSchedule
          labelSelector:
            matchLabels:
              app.kubernetes.io/name: prometheus

```

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](#)

- [Controlling pod placement by using pod topology spread constraints](#)
- [Pod Topology Spread Constraints](#) (Kubernetes documentation)

### 3.3. STORING AND RECORDING DATA FOR CORE PLATFORM 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.

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

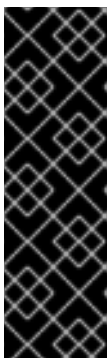


#### IMPORTANT

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

##### 3.3.1.1. Persistent storage prerequisites

- Dedicate sufficient persistent storage to ensure that the disk does not become full.
- Use **Filesystem** as the storage type value for the **volumeMode** parameter when you configure the persistent volume.



#### IMPORTANT

- Do not use a raw block volume, which is described with **volumeMode: Block** in the **PersistentVolume** resource. Prometheus cannot use raw block volumes.
- Prometheus does not support file systems that are not POSIX compliant. For example, some NFS file system implementations are not POSIX compliant. If you want to use an NFS file system for storage, verify with the vendor that their NFS implementation is fully POSIX compliant.

##### 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 **cluster-admin** cluster role.

- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Add your PVC configuration for the component under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-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 Prometheus:

## Example PVC configuration

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      volumeClaimTemplate:
        spec:
          storageClassName: my-storage-class
          resources:
            requests:
              storage: 40Gi
```

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**

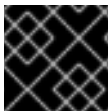
When you update the config map with a PVC configuration, the affected **StatefulSet** object is recreated, resulting in a temporary service outage.

**Additional resources**

- [Understanding persistent storage](#)
- [PersistentVolumeClaims](#) (Kubernetes documentation)

**3.3.1.3. Resizing a persistent volume**

You can resize a persistent volume (PV) for monitoring components, such as Prometheus or Alertmanager. You need to manually expand a persistent volume claim (PVC), and then update the config map in which the component is configured.

**IMPORTANT**

You can only expand the size of the PVC. Shrinking the storage size is not possible.

**Prerequisites**

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- You have configured at least one PVC for core OpenShift Container Platform monitoring components.
- You have installed the OpenShift CLI (**oc**).

**Procedure**

1. Manually expand a PVC with the updated storage request. For more information, see "Expanding persistent volume claims (PVCs) with a file system" in *Expanding persistent volumes*.
2. Edit the **cluster-monitoring-config** config map in the **openshift-monitoring** project:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

3. Add a new storage size for the PVC configuration for the component under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
```



```

namespace: openshift-monitoring
data:
  config.yaml: |
    <component>: ❶
    volumeClaimTemplate:
      spec:
        resources:
          requests:
            storage: <amount_of_storage> ❷

```

- ❶ The component for which you want to change the storage size.
- ❷ Specify the new size for the storage volume. It must be greater than the previous value.

The following example sets the new PVC request to 100 gigabytes for the Prometheus instance:

### Example storage configuration for prometheusK8s

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      volumeClaimTemplate:
        spec:
          resources:
            requests:
              storage: 100Gi

```

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



#### WARNING

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

#### Additional resources

- [Prometheus database storage requirements](#)
- [Expanding persistent volume claims \(PVCs\) with a file system](#)

### 3.3.2. Modifying retention time and size for Prometheus metrics data

By default, Prometheus retains metrics data for 15 days for core platform monitoring. 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 **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Add the retention time and size configuration under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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: cluster-monitoring-config
```

```

namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      retention: 24h
      retentionSize: 10GB

```

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](#)
- [Preparing to configure core platform monitoring stack](#)
- [Prometheus database storage requirements](#)
- [Recommended configurable storage technology](#)
- [Understanding persistent storage](#)
- [Optimizing storage](#)

### 3.3.3. Configuring audit logs for Metrics Server

You can configure audit logs for Metrics Server to help you troubleshoot issues with the server. Audit logs record the sequence of actions in a cluster. It can record user, application, or control plane activities.

You can set audit log rules, which determine what events are recorded and what data they should include. This can be achieved with the following audit profiles:

- **Metadata (default):** This profile enables the logging of event metadata including user, timestamps, resource, and verb. It does not record request and response bodies.
- **Request:** This enables the logging of event metadata and request body, but it does not record response body. This configuration does not apply for non-resource requests.
- **RequestResponse:** This enables the logging of event metadata, and request and response bodies. This configuration does not apply for non-resource requests.
- **None:** None of the previously described events are recorded.

You can configure the audit profiles by modifying the **cluster-monitoring-config** config map. The following example sets the profile to **Request**, allowing the logging of event metadata and request body for Metrics Server:

```

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

```

```
metricsServer:
  audit:
    profile: Request
```

### 3.3.4. Setting log levels for monitoring components

You can configure the log level for Alertmanager, Prometheus Operator, Prometheus, and Thanos Querier.

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

- **debug**. Log debug, informational, warning, and error messages.
- **info**. Log informational, warning, and error messages.
- **warn**. Log warning and error messages only.
- **error**. Log error messages only.

The default log level is **info**.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Add **logLevel: <log\_level>** for a component under **data/config.yaml**:

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

- 1 The monitoring stack component for which you are setting a log level. Available component values are **prometheusK8s**, **alertmanagerMain**, **prometheusOperator**, and **thanosQuerier**.
- 2 The log level to set 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. Confirm that the log level has been 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-monitoring get deploy prometheus-operator -o yaml | grep "log-level"
```

#### Example output

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

5. Check that the pods for the component are running. The following example lists the status of pods:

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



#### NOTE

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

### 3.3.5. 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 **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Add the **queryLogFile** parameter for Prometheus under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
```

```
namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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:

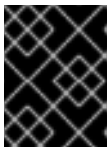
```
$ oc -n openshift-monitoring get pods
```

### Example output

```
...
prometheus-operator-567c9bc75c-96wkj 2/2 Running 0 62m
prometheus-k8s-0 6/6 Running 1 57m
prometheus-k8s-1 6/6 Running 1 57m
thanos-querier-56c76d7df4-2xkpc 6/6 Running 0 57m
thanos-querier-56c76d7df4-j5p29 6/6 Running 0 57m
...
```

5. Read the query log:

```
$ oc -n openshift-monitoring exec prometheus-k8s-0 -- cat <path>
```



### IMPORTANT

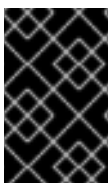
Revert the setting in the config map after you have examined the logged query information.

### Additional resources

- [Preparing to configure core platform monitoring stack](#)

## 3.3.6. Enabling query logging for Thanos Querier

For default platform monitoring in the **openshift-monitoring** project, you can enable the Cluster Monitoring Operator (CMO) to log all queries run by Thanos Querier.



### 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 installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.

## Procedure

You can enable query logging for Thanos Querier in the **openshift-monitoring** project:

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

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. Add a **thanosQuerier** section under **data/config.yaml** and add values as shown in the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    thanosQuerier:
      enableRequestLogging: <value> 1
      logLevel: <value> 2
```

- 1 Set the value to **true** to enable logging and **false** to disable logging. The default value is **false**.

- 2 Set the value to **debug**, **info**, **warn**, or **error**. If no value exists for **logLevel**, the log level defaults to **error**.

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

## Verification

1. Verify that the Thanos Querier pods are running. The following sample command lists the status of pods in the **openshift-monitoring** project:

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

2. Run a test query using the following sample commands as a model:

```
$ token=`oc create token prometheus-k8s -n openshift-monitoring`
$ oc -n openshift-monitoring exec -c prometheus prometheus-k8s-0 -- curl -k -H
"Authorization: Bearer $token" 'https://thanos-querier.openshift-
monitoring.svc:9091/api/v1/query?query=cluster_version'
```

3. Run the following command to read the query log:

```
$ oc -n openshift-monitoring logs <thanos_querier_pod_name> -c thanos-query
```

**NOTE**

Because the **thanos-querier** pods are highly available (HA) pods, you might be able to see logs in only one pod.

4. After you examine the logged query information, disable query logging by changing the **enableRequestLogging** value to **false** in the config map.

## 3.4. CONFIGURING METRICS FOR CORE PLATFORM 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 **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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-monitoring** namespace.



**WARNING**

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

**Procedure**

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

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. Add a **remoteWrite** section under **data/config.yaml/prometheusK8s**, as shown in the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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

- [writeRelabelConfigs](#)
- [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	<b>sigv4</b>	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	<b>basicAuth</b>	Basic authentication sets the authorization header on every remote write request with the configured username and password.
authorization	<b>authorization</b>	Authorization sets the <b>Authorization</b> header on every remote write request using the configured token.
OAuth 2.0	<b>oauth2</b>	An OAuth 2.0 configuration uses the client credentials grant type. Prometheus fetches an access token from <b>tokenUrl</b> 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.
TLS client	<b>tlsConfig</b>	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 default platform monitoring in the **openshift-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-monitoring** namespace.

```
apiVersion: v1
kind: Secret
metadata:
  name: sigv4-credentials
  namespace: openshift-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-monitoring** namespace:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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-monitoring** namespace:

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

❶ The username.

❷ The password.

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://basicauth.example.com/api/write"
          basicAuth:
            username:
              name: rw-basic-auth ❶
              key: user ❷
            password:
              name: rw-basic-auth ❸
              key: password ❹
```

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

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

❹ 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-monitoring** namespace:

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: rw-bearer-auth
  namespace: openshift-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-monitoring** namespace:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
    prometheusK8s:
      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-monitoring** namespace:

```
apiVersion: v1
kind: Secret
metadata:
  name: oauth2-credentials
  namespace: openshift-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-monitoring** namespace:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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-monitoring** namespace.

```
apiVersion: v1
kind: Secret
metadata:
  name: mtls-bundle
  namespace: openshift-monitoring
```

```
data:
  ca.crt: <ca_cert> ❶
  client.crt: <client_cert> ❷
  client.key: <client_key> ❸
type: tls
```

- ❶ The CA certificate in the Prometheus container with which to validate the server certificate.
- ❷ The client certificate for authentication with the server.
- ❸ 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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://remote-write-endpoint.example.com"
          tlsConfig:
            ca:
              secret:
                name: mtls-bundle ❶
                key: ca.crt ❷
            cert:
              secret:
                name: mtls-bundle ❸
                key: client.crt ❹
            keySecret:
              name: mtls-bundle ❺
              key: client.key ❻
```

- ❶ ❸ ❺ 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.
- ❷ The key in the specified **Secret** object that contains the CA certificate for the endpoint.
- ❹ The key in the specified **Secret** object that contains the client certificate for the endpoint.
- ❻ 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 default platform monitoring in the **openshift-monitoring** namespace.



## Example configuration of remote write parameters with default values

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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

- [Prometheus REST API reference for remote write](#)
- [Setting up remote write compatible endpoints](#) (Prometheus documentation)
- [Tuning remote write settings](#) (Prometheus documentation)
- [Understanding secrets](#)

### 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 **cluster-monitoring-config** config map in the **openshift-monitoring** namespace.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).
- You have configured remote write storage.

#### 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 **writeRelabelConfigs:** section under **data/config.yaml/prometheusK8s/remoteWrite**, add cluster ID relabel configuration values:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
```

```
- url: "https://remote-write-endpoint.example.com"
  writeRelabelConfigs:
    - sourceLabels:
        - __tmp_openshift_cluster_id__ ❶
      targetLabel: cluster_id ❷
      action: replace ❸
```

- ❶ 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.
- ❷ 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.
- ❸ 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](#)

## 3.5. CONFIGURING ALERTS AND NOTIFICATIONS FOR CORE PLATFORM 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 Container Platform monitoring stack includes a local Alertmanager instance that routes alerts from Prometheus.

You can add external Alertmanager instances to route alerts for core OpenShift Container Platform 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 **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Add an **additionalAlertmanagerConfigs** section with configuration details under **data/config.yaml/prometheusK8s:**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      additionalAlertmanagerConfigs:
        - <alertmanager_specification> 1
```

- 1 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**).

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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.

### 3.5.1.1. Disabling the local Alertmanager

A local Alertmanager that routes alerts from Prometheus instances is enabled by default in the **openshift-monitoring** project of the OpenShift Container Platform monitoring stack.

If you do not need the local Alertmanager, you can disable it by configuring the **cluster-monitoring-config** config map in the **openshift-monitoring** project.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config** config map.
- 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. Add **enabled: false** for the **alertmanagerMain** component under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      enabled: false
```

3. Save the file to apply the changes. The Alertmanager instance is disabled automatically when you apply the change.

#### Additional resources

- [Alertmanager](#) (Prometheus documentation)
- [Managing alerts as an Administrator](#)

### 3.5.2. Configuring secrets for Alertmanager

The OpenShift Container Platform 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 **cluster-monitoring-config** config map in the **openshift-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 **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config** config map.
- You have created the secret to be configured in Alertmanager in the **openshift-monitoring** project.
- 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. Add a **secrets:** section under **data/config.yaml/alertmanagerMain** with the following configuration:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      secrets: ❶
      - <secret_name_1> ❷
      - <secret_name_2>
```

❶ This section contains the secrets to be mounted into Alertmanager. The secrets must be located within the same namespace as the Alertmanager object.

❷ 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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      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 **cluster-admin** cluster role.
- You have created the **cluster-monitoring-config ConfigMap** object.
- 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. Define labels you want to add for every metric under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      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** or **managed\_cluster** as key names. Using them can cause issues where you are unable to see data in the developer dashboards.

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: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      externalLabels:
        region: eu
        environment: prod
```

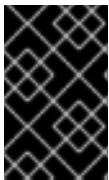
3. Save the file to apply the changes. The pods affected by the new configuration are automatically redeployed.

**Additional resources**

- [Preparing to configure core platform monitoring stack](#)

**3.5.4. Configuring alert notifications**

In OpenShift Container Platform 4.18, you can view firing alerts in the Alerting UI. You can configure Alertmanager to send notifications about default platform alerts by configuring alert receivers.

**IMPORTANT**

Alertmanager does not send notifications by default. It is strongly recommended to configure Alertmanager to receive notifications by configuring alert receivers through the web console or through the **alertmanager-main** secret.

**Additional resources**

- [Sending notifications to external systems](#)
- [PagerDuty](#) (PagerDuty official site)
- [Prometheus Integration Guide](#) (PagerDuty official site)



- [Support version matrix for monitoring components](#)
- [Enabling alert routing for user-defined projects](#)

### 3.5.4.1. Configuring alert routing for default platform alerts

You can configure Alertmanager to send notifications to receive important alerts coming from your cluster. Customize where and how Alertmanager sends notifications about default platform alerts by editing the default configuration in the **alertmanager-main** secret in the **openshift-monitoring** namespace.



#### NOTE

All features of a supported version of upstream Alertmanager are also supported in an OpenShift Container Platform 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 **cluster-admin** cluster role.
- You have installed the OpenShift CLI (**oc**).

#### Procedure

1. Extract the currently active Alertmanager configuration from the **alertmanager-main** secret and save it as a local **alertmanager.yaml** file:

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

2. Open the **alertmanager.yaml** file.
3. Edit the Alertmanager configuration:
  - a. Optional: Change the default Alertmanager configuration:

#### Example of the default Alertmanager secret YAML

```
global:
  resolve_timeout: 5m
  http_config:
    proxy_from_environment: true 1
route:
  group_wait: 30s 2
  group_interval: 5m 3
  repeat_interval: 12h 4
  receiver: default
routes:
- matchers:
  - "alertname=Watchdog"
  repeat_interval: 2m
  receiver: watchdog
```

```

receivers:
- name: default
- name: watchdog

```

- 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 how long Alertmanager waits while collecting initial alerts for a group of alerts before sending a notification.
- 3 Specify how much time must elapse before Alertmanager sends a notification about new alerts added to a group of alerts for which an initial notification was already sent.
- 4 Specify the minimum amount of time that must pass before an alert notification is repeated. If you want a notification to repeat at each group interval, set the **repeat\_interval** value to less than the **group\_interval** value. The repeated notification can still be delayed, for example, when certain Alertmanager pods are restarted or rescheduled.

b. Add your alert receiver configuration:

```

# ...
receivers:
- name: default
- name: watchdog
- name: <receiver> 1
  <receiver_configuration> 2
# ...

```

- 1 The name of the receiver.
- 2 The receiver configuration. The supported receivers are PagerDuty, webhook, email, Slack, and Microsoft Teams.

### Example of configuring PagerDuty as an alert receiver

```

# ...
receivers:
- name: default
- name: watchdog
- name: team-frontend-page
  pagerduty_configs:
  - routing_key: xxxxxxxxxx 1
    http_config: 2
      proxy_from_environment: true
      authorization:
        credentials: xxxxxxxxxx
# ...

```

- 1 Defines the PagerDuty integration key.
- 2 Optional: Add the custom HTTP configuration for a specific receiver. That receiver does not inherit the global HTTP configuration settings.

## Example of configuring email as an alert receiver

```
# ...
receivers:
- name: default
- name: watchdog
- name: team-frontend-page
email_configs:
- to: myemail@example.com ❶
  from: alertmanager@example.com ❷
  smarthost: 'smtp.example.com:587' ❸
  auth_username: alertmanager@example.com ❹
  auth_password: password
  hello: alertmanager ❺
# ...
```

- ❶ Specify an email address to send notifications to.
- ❷ Specify an email address to send notifications from.
- ❸ Specify the SMTP server address used for sending emails, including the port number.
- ❹ Specify the authentication credentials that Alertmanager uses to connect to the SMTP server. This example uses username and password.
- ❺ Specify the hostname to identify to the SMTP server. If you do not include this parameter, the hostname defaults to **localhost**.



### IMPORTANT

Alertmanager requires an external SMTP server to send email alerts. To configure email alert receivers, ensure you have the necessary connection details for an external SMTP server.

- c. Add the routing configuration:

```
# ...
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
  routes:
  - matchers:
    - "alertname=Watchdog"
    repeat_interval: 2m
    receiver: watchdog
  - matchers: ❶
    - "<your_matching_rules>" ❷
    receiver: <receiver> ❸
# ...
```

- 1 Use the **matchers** key name to specify the matching rules that an alert has to fulfill to match the node. If you define inhibition rules, use **target\_matchers** key name for
- 2 Specify labels to match your alerts.
- 3 Specify the name of the receiver to use for the alerts.

**WARNING**

Do not use the **match**, **match\_re**, **target\_match**, **target\_match\_re**, **source\_match**, and **source\_match\_re** key names, which are deprecated and planned for removal in a future release.

**Example of alert routing**

```
# ...
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
  routes:
    - matchers:
      - "alertname=Watchdog"
      repeat_interval: 2m
      receiver: watchdog
    - matchers: 1
      - "service=example-app"
      routes: 2
        - matchers:
          - "severity=critical"
          receiver: team-frontend-page
# ...
```

- 1 This example matches alerts from the **example-app** service.
- 2 You can create routes within other routes for more complex alert routing.

The previous example routes alerts of **critical** severity that are fired by the **example-app** service to the **team-frontend-page** receiver. Typically, these types of alerts are paged to an individual or a critical response team.

4. Apply the new configuration in the file:

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

5. Verify your routing configuration by visualizing the routing tree:

```
$ oc exec alertmanager-main-0 -n openshift-monitoring -- amtool config routes show --
alertmanager.url http://localhost:9093
```

### Example output

Routing tree:

```
.
├── default-route receiver: default
│   ├── {alertname="Watchdog"} receiver: Watchdog
│   └── {service="example-app"} receiver: default
│       └── {severity="critical"} receiver: team-frontend-page
```

### Additional resources

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

#### 3.5.4.2. Configuring alert routing with the OpenShift Container Platform web console

You can configure alert routing through the OpenShift Container Platform web console to ensure that you learn about important issues with your cluster.



### NOTE

The OpenShift Container Platform web console provides fewer settings to configure alert routing than the **alertmanager-main** secret. To configure alert routing with the access to more configuration settings, see "Configuring alert routing for default platform alerts".

### Prerequisites

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

### Procedure


1. In the **Administrator** perspective, go to **Administration** → **Cluster Settings** → **Configuration** → **Alertmanager**.



### NOTE

Alternatively, you can go to the same page through the notification drawer. Select the bell icon at the top right of the OpenShift Container Platform web console and choose **Configure** in the **AlertmanagerReceiverNotConfigured** alert.

2. Click **Create Receiver** in the **Receivers** section of the page.
3. In the **Create Receiver** form, add a **Receiver name** and choose a **Receiver type** from the list.
4. Edit the receiver configuration:
  - For PagerDuty receivers:

- a. Choose an integration type and add a PagerDuty integration key.
  - b. Add the URL of your PagerDuty installation.
  - c. Click **Show advanced configuration** if you want to edit the client and incident details or the severity specification.
- For webhook receivers:
    - a. Add the endpoint to send HTTP POST requests to.
    - b. Click **Show advanced configuration** if you want to edit the default option to send resolved alerts to the receiver.
  - For email receivers:
    - a. Add the email address to send notifications to.
    - b. Add SMTP configuration details, including the address to send notifications from, the smarthost and port number used for sending emails, the hostname of the SMTP server, and authentication details.
- 
- IMPORTANT**
- Alertmanager requires an external SMTP server to send email alerts. To configure email alert receivers, ensure you have the necessary connection details for an external SMTP server.
- c. Select whether TLS is required.
  - d. Click **Show advanced configuration** if you want to edit the default option not to send resolved alerts to the receiver or edit the body of email notifications configuration.
- For Slack receivers:
    - a. Add the URL of the Slack webhook.
    - b. Add the Slack channel or user name to send notifications to.
    - c. Select **Show advanced configuration** if you want to edit the default option not to send resolved alerts to the receiver or edit the icon and username configuration. You can also choose whether to find and link channel names and usernames.
5. By default, firing alerts with labels that match all of the selectors are sent to the receiver. If you want label values for firing alerts to be matched exactly before they are sent to the receiver, perform the following steps:
    - a. Add routing label names and values in the **Routing labels** section of the form.
    - b. Click **Add label** to add further routing labels.
  6. Click **Create** to create the receiver.

### 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. CONFIGURING USER WORKLOAD MONITORING

### 4.1. PREPARING TO CONFIGURE THE USER WORKLOAD MONITORING STACK

This section explains which user-defined monitoring components can be configured, how to enable user workload monitoring, 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.
- The monitoring stack imposes additional resource requirements. Consult the computing resources recommendations in [Scaling the Cluster Monitoring Operator](#) and verify that you have sufficient resources.

#### 4.1.1. 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.

Table 4.1. Configurable monitoring components for user-defined projects

Component	user-workload-monitoring-config config map key
Prometheus Operator	<b>prometheusOperator</b>
Prometheus	<b>prometheus</b>
Alertmanager	<b>alertmanager</b>
Thanos Ruler	<b>thanosRuler</b>



**WARNING**

Different configuration changes to the **ConfigMap** object result in different outcomes:

- The pods are not redeployed. Therefore, there is no service outage.
- The affected pods are redeployed:
  - For single-node clusters, this results in temporary service outage.
  - For multi-node clusters, because of high-availability, the affected pods are gradually rolled out and the monitoring stack remains available.
  - Configuring and resizing a persistent volume always results in a service outage, regardless of high availability.

Each procedure that requires a change in the config map includes its expected outcome.

## 4.1.2. Enabling monitoring for user-defined projects

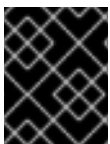
In OpenShift Container Platform, you can enable monitoring for user-defined projects in addition to the default platform monitoring. You can monitor your own projects in OpenShift Container Platform without the need for an additional monitoring solution. Using this feature centralizes monitoring for core platform components and user-defined projects.

**NOTE**

Versions of Prometheus Operator installed using Operator Lifecycle Manager (OLM) are not compatible with user-defined monitoring. Therefore, custom Prometheus instances installed as a Prometheus custom resource (CR) managed by the OLM Prometheus Operator are not supported in OpenShift Container Platform.

### 4.1.2.1. Enabling monitoring for user-defined projects

Cluster administrators can enable monitoring for user-defined projects by setting the **enableUserWorkload: true** field in the cluster monitoring **ConfigMap** object.

**IMPORTANT**

You must remove any custom Prometheus instances before enabling monitoring for user-defined projects.

**NOTE**

You must have access to the cluster as a user with the **cluster-admin** cluster role to enable monitoring for user-defined projects in OpenShift Container Platform. Cluster administrators can then optionally grant users permission to configure the components that are responsible for monitoring user-defined projects.

## Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have installed the OpenShift CLI (**oc**).
- You have created the **cluster-monitoring-config ConfigMap** object.
- You have optionally created and configured the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project. You can add configuration options to this **ConfigMap** object for the components that monitor user-defined projects.



### NOTE

Every time you save configuration changes to the **user-workload-monitoring-config ConfigMap** object, the pods in the **openshift-user-workload-monitoring** project are redeployed. It might sometimes take a while for these components to redeploy.

## Procedure

1. Edit the **cluster-monitoring-config ConfigMap** object:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. Add **enableUserWorkload: true** under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true 1
```

- 1 When set to **true**, the **enableUserWorkload** parameter enables monitoring for user-defined projects in a cluster.

3. Save the file to apply the changes. Monitoring for user-defined projects is then enabled automatically.



### NOTE

If you enable monitoring for user-defined projects, the **user-workload-monitoring-config ConfigMap** object is created by default.

4. Verify that the **prometheus-operator**, **prometheus-user-workload**, and **thanos-ruler-user-workload** pods are running in the **openshift-user-workload-monitoring** project. It might take a short while for the pods to start:

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

### Example output

NAME	READY	STATUS	RESTARTS	AGE
prometheus-operator-6f7b748d5b-t7nbg	2/2	Running	0	3h
prometheus-user-workload-0	4/4	Running	1	3h
prometheus-user-workload-1	4/4	Running	1	3h
thanos-ruler-user-workload-0	3/3	Running	0	3h
thanos-ruler-user-workload-1	3/3	Running	0	3h

### Additional resources

- [User workload monitoring first steps](#)

#### 4.1.2.2. Granting users permission to configure monitoring for user-defined projects

As a cluster administrator, you can assign the **user-workload-monitoring-config-edit** role to a user. This grants permission to configure and manage monitoring for user-defined projects without giving them permission to configure and manage core OpenShift Container Platform monitoring components.

### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- The user account that you are assigning the role to already exists.
- You have installed the OpenShift CLI (**oc**).

### Procedure

1. Assign the **user-workload-monitoring-config-edit** role to a user in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring adm policy add-role-to-user \
  user-workload-monitoring-config-edit <user> \
  --role-namespace openshift-user-workload-monitoring
```

2. Verify that the user is correctly assigned to the **user-workload-monitoring-config-edit** role by displaying the related role binding:

```
$ oc describe rolebinding <role_binding_name> -n openshift-user-workload-monitoring
```

### Example command

```
$ oc describe rolebinding user-workload-monitoring-config-edit -n openshift-user-workload-monitoring
```

### Example output

```
Name:      user-workload-monitoring-config-edit
Labels:    <none>
Annotations: <none>
Role:
  Kind: Role
```

```

Name: user-workload-monitoring-config-edit
Subjects:
  Kind Name Namespace
  ---- ----
  User user1

```

- 1 In this example, **user1** is assigned to the **user-workload-monitoring-config-edit** role.

### 4.1.3. Enabling alert routing for user-defined projects

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

- Enable alert routing for user-defined projects:
  - Use the default platform Alertmanager instance.
  - Use a separate Alertmanager instance only for user-defined projects.
- 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](#)

#### 4.1.3.1. Enabling the platform Alertmanager instance for user-defined alert routing

You can allow users to create user-defined alert routing configurations that use the main platform instance of Alertmanager.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- A cluster administrator has enabled monitoring for user-defined projects.
- You have installed the OpenShift CLI (**oc**).

#### Procedure

1. Edit the **cluster-monitoring-config ConfigMap** object:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

2. Add **enableUserAlertmanagerConfig: true** in the **alertmanagerMain** section under **data/config.yaml**:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config

```

```

namespace: openshift-monitoring
data:
  config.yaml: |
    # ...
    alertmanagerMain:
      enableUserAlertmanagerConfig: true ❶
    # ...

```

- ❶ Set the **enableUserAlertmanagerConfig** value to **true** to allow users to create user-defined alert routing configurations that use the main platform instance of Alertmanager.

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

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

In some clusters, you might want to deploy a dedicated Alertmanager instance for user-defined projects, which can help reduce the load on the default platform Alertmanager instance and can better separate user-defined alerts 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 **cluster-admin** cluster role.
- You have enabled monitoring for user-defined projects.
- 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 ❶
      enableAlertmanagerConfig: true ❷

```

- ❶ 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 **user-workload** Alertmanager instance has started:

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

#### Example output

```
NAME          VERSION  REPLICAS  AGE
user-workload  0.24.0   2         100s
```

### 4.1.3.3. 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 **cluster-admin** cluster role.
- You have enabled monitoring for user-defined projects.
- 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](#)

### 4.1.4. Granting users permissions for monitoring for user-defined projects

As a cluster administrator, you can monitor all core OpenShift Container Platform and user-defined projects.

You can also grant developers and other users different permissions:

- Monitoring user-defined projects

- Configuring the components that monitor user-defined projects
- Configuring alert routing for user-defined projects
- Managing alerts and silences for user-defined projects

You can grant the permissions by assigning one of the following monitoring roles or cluster roles:

Table 4.2. Monitoring roles

Role name	Description	Project
<b>user-workload-monitoring-config-edit</b>	Users with this role can edit the <b>user-workload-monitoring-config ConfigMap</b> object to configure Prometheus, Prometheus Operator, Alertmanager, and Thanos Ruler for user-defined workload monitoring.	<b>openshift-user-workload-monitoring</b>
<b>monitoring-alertmanager-api-reader</b>	Users with this role have read access to the user-defined Alertmanager API for all projects, if the user-defined Alertmanager is enabled.	<b>openshift-user-workload-monitoring</b>
<b>monitoring-alertmanager-api-writer</b>	Users with this role have read and write access to the user-defined Alertmanager API for all projects, if the user-defined Alertmanager is enabled.	<b>openshift-user-workload-monitoring</b>

Table 4.3. Monitoring cluster roles

Cluster role name	Description	Project
<b>monitoring-rules-view</b>	Users with this cluster role have read access to <b>PrometheusRule</b> custom resources (CRs) for user-defined projects. They can also view the alerts and silences in the <b>Developer</b> perspective of the OpenShift Container Platform web console.	Can be bound with <b>RoleBinding</b> to any user project.

Cluster role name	Description	Project
<b>monitoring-rules-edit</b>	Users with this cluster role can create, modify, and delete <b>PrometheusRule</b> CRs for user-defined projects. They can also manage alerts and silences in the <b>Developer</b> perspective of the OpenShift Container Platform web console.	Can be bound with <b>RoleBinding</b> to any user project.
<b>monitoring-edit</b>	Users with this cluster role have the same privileges as users with the <b>monitoring-rules-edit</b> cluster role. Additionally, users can create, read, modify, and delete <b>ServiceMonitor</b> and <b>PodMonitor</b> resources to scrape metrics from services and pods.	Can be bound with <b>RoleBinding</b> to any user project.
<b>alert-routing-edit</b>	Users with this cluster role can create, update, and delete <b>AlertmanagerConfig</b> CRs for user-defined projects.	Can be bound with <b>RoleBinding</b> to any user project.
<b>pod-metrics-reader</b>	Users with this cluster role can access Thanos API endpoints for user-defined projects.	Can be bound with <b>RoleBinding</b> to any user project.

#### Additional resources

- [CMO services resources](#)
- [Granting users permission to configure monitoring for user-defined projects](#)
- [Granting users permission to configure alert routing for user-defined projects](#)

#### 4.1.4.1. Granting user permissions by using the web console

You can grant users permissions for the **openshift-monitoring** project or their own projects, by using the OpenShift Container Platform web console.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- The user account that you are assigning the role to already exists.

#### Procedure



1. In the **Administrator** perspective of the OpenShift Container Platform web console, go to **User Management → RoleBindings → Create binding**.
2. In the **Binding Type** section, select the **Namespace Role Binding** type.
3. In the **Name** field, enter a name for the role binding.
4. In the **Namespace** field, select the project where you want to grant the access.



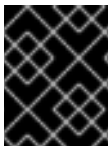
### IMPORTANT

The monitoring role or cluster role permissions that you grant to a user by using this procedure apply only to the project that you select in the **Namespace** field.

5. Select a monitoring role or cluster role from the **Role Name** list.
6. In the **Subject** section, select **User**.
7. In the **Subject Name** field, enter the name of the user.
8. Select **Create** to apply the role binding.

#### 4.1.4.2. Granting user permissions by using the CLI

You can grant users permissions to monitor their own projects, by using the OpenShift CLI (**oc**).



### IMPORTANT

Whichever role or cluster role you choose, you must bind it against a specific project as a cluster administrator.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- The user account that you are assigning the role to already exists.
- You have installed the OpenShift CLI (**oc**).

#### Procedure

- To assign a monitoring role to a user for a project, enter the following command:

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

- 1 Substitute **<role>** with the wanted monitoring role, **<user>** with the user to whom you want to assign the role, and **<namespace>** with the project where you want to grant the access.

- To assign a monitoring cluster role to a user for a project, enter the following command:

```
$ oc adm policy add-cluster-role-to-user <cluster-role> <user> -n <namespace> 1
```

- 1 Substitute **<cluster-role>** with the wanted monitoring cluster role, **<user>** with the user to whom you want to assign the cluster role, and **<namespace>** with the project where you

### 4.1.5. 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-'
```

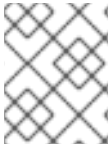


#### NOTE

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.

### 4.1.6. Disabling monitoring for user-defined projects

After enabling monitoring for user-defined projects, you can disable it again by setting **enableUserWorkload: false** in the cluster monitoring **ConfigMap** object.



#### NOTE

Alternatively, you can remove **enableUserWorkload: true** to disable monitoring for user-defined projects.

#### Procedure

1. Edit the **cluster-monitoring-config ConfigMap** object:

```
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```

- a. Set **enableUserWorkload:** to **false** under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: false
```

2. Save the file to apply the changes. Monitoring for user-defined projects is then disabled automatically.

3. Check that the **prometheus-operator**, **prometheus-user-workload** and **thanos-ruler-user-workload** pods are terminated in the **openshift-user-workload-monitoring** project. This might take a short while:

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

#### Example output

```
No resources found in openshift-user-workload-monitoring project.
```



#### NOTE

The **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project is not automatically deleted when monitoring for user-defined projects is disabled. This is to preserve any custom configurations that you may have created in the **ConfigMap** object.

## 4.2. 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.

### 4.2.1. 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](#)

#### 4.2.1.1. 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 **cluster-admin** cluster role or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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

- [Enabling monitoring for user-defined projects](#)
- [Understanding how to update labels on nodes](#)
- [Placing pods on specific nodes using node selectors](#)
- [nodeSelector](#) (Kubernetes documentation)

#### 4.2.1.2. 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 **cluster-admin** cluster role, or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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

- [Enabling monitoring for user-defined projects](#)
- [Controlling pod placement using node taints](#)
- [Taints and Tolerations](#) (Kubernetes documentation)

## 4.2.2. 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.

### 4.2.2.1. 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](#)
- [Kubernetes requests and limits documentation](#) (Kubernetes documentation)

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

Cluster administrators 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.

#### Additional resources

- [Controlling the impact of unbound metrics attributes in user-defined projects](#)
- [Enabling monitoring for user-defined projects](#)
- [Determining why Prometheus is consuming a lot of disk space](#)

#### 4.2.3.1. 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 **cluster-admin** cluster role, or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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.

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.

### 4.2.3.2. Creating scrape sample alerts

You can create alerts that notify you when:

- The target cannot be scraped or is not available for the specified **for** duration
- A scrape sample threshold is reached or is exceeded for the specified **for** duration

#### 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.
- A cluster administrator has enabled monitoring for user-defined projects.
- You have limited the number of samples that can be accepted per target scrape in user-defined projects, by using **enforcedSampleLimit**.
- You have installed the OpenShift CLI (**oc**).

#### Procedure

1. Create a YAML file with alerts that inform you when the targets are down and when the enforced sample limit is approaching. The file in this example is called **monitoring-stack-alerts.yaml**:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  labels:
    prometheus: k8s
    role: alert-rules
    name: monitoring-stack-alerts 1
    namespace: ns1 2
spec:
  groups:
    - name: general.rules
```

```

rules:
- alert: TargetDown 3
  annotations:
    message: '{{ printf "%.4g" $value }}% of the {{ $labels.job }}/{{ $labels.service
    }} targets in {{ $labels.namespace }} namespace are down.' 4
    expr: 100 * (count(up == 0) BY (job, namespace, service) / count(up) BY (job,
    namespace, service)) > 10
    for: 10m 5
    labels:
      severity: warning 6
- alert: ApproachingEnforcedSamplesLimit 7
  annotations:
    message: '{{ $labels.container }} container of the {{ $labels.pod }} pod in the {{
    $labels.namespace }} namespace consumes {{ $value | humanizePercentage }} of the
    samples limit budget.' 8
    expr: (scrape_samples_post_metric_relabeling / (scrape_sample_limit > 0)) > 0.9 9
    for: 10m 10
    labels:
      severity: warning 11

```

- 1** Defines the name of the alerting rule.
- 2** Specifies the user-defined project where the alerting rule is deployed.
- 3** The **TargetDown** alert fires if the target cannot be scraped and is not available for the **for** duration.
- 4** The message that is displayed when the **TargetDown** alert fires.
- 5** The conditions for the **TargetDown** alert must be true for this duration before the alert is fired.
- 6** Defines the severity for the **TargetDown** alert.
- 7** The **ApproachingEnforcedSamplesLimit** alert fires when the defined scrape sample threshold is exceeded and lasts for the specified **for** duration.
- 8** The message that is displayed when the **ApproachingEnforcedSamplesLimit** alert fires.
- 9** The threshold for the **ApproachingEnforcedSamplesLimit** alert. In this example, the alert fires when the number of ingested samples exceeds 90% of the configured limit.
- 10** The conditions for the **ApproachingEnforcedSamplesLimit** alert must be true for this duration before the alert is fired.
- 11** Defines the severity for the **ApproachingEnforcedSamplesLimit** alert.

2. Apply the configuration to the user-defined project:

```
$ oc apply -f monitoring-stack-alerts.yaml
```

3. Additionally, you can check if a target has hit the configured limit:

- a. In the **Administrator** perspective of the web console, go to **Observe → Targets** and select

an endpoint with a **Down** status that you want to check.

The **Scrape failed: sample limit exceeded** message is displayed if the endpoint failed because of an exceeded sample limit.

#### 4.2.4. 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 **cluster-admin** cluster role or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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
- 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](#)
- [Controlling pod placement by using pod topology spread constraints](#)
- [Pod Topology Spread Constraints](#) (Kubernetes documentation)

## 4.3. 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.

### 4.3.1. 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.

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

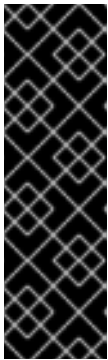


### IMPORTANT

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

#### 4.3.1.1. Persistent storage prerequisites

- Dedicate sufficient persistent storage to ensure that the disk does not become full.
- Use **Filesystem** as the storage type value for the **volumeMode** parameter when you configure the persistent volume.



### IMPORTANT

- Do not use a raw block volume, which is described with **volumeMode: Block** in the **PersistentVolume** resource. Prometheus cannot use raw block volumes.
- Prometheus does not support file systems that are not POSIX compliant. For example, some NFS file system implementations are not POSIX compliant. If you want to use an NFS file system for storage, verify with the vendor that their NFS implementation is fully POSIX compliant.

#### 4.3.1.2. 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 **cluster-admin** cluster role, or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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>: ❶
    volumeClaimTemplate:
      spec:
        storageClassName: <storage_class> ❷
        resources:
          requests:
            storage: <amount_of_storage> ❸

```

- ❶ Specify the monitoring component for which you want to configure the PVC.
- ❷ Specify an existing storage class. If a storage class is not specified, the default storage class is used.
- ❸ 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**

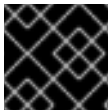
When you update the config map with a PVC configuration, the affected **StatefulSet** object is recreated, resulting in a temporary service outage.

**Additional resources**

- [Understanding persistent storage](#)
- [PersistentVolumeClaims](#) (Kubernetes documentation)

**4.3.1.3. Resizing a persistent volume**

You can resize a persistent volume (PV) for the instances of Prometheus, Thanos Ruler, and Alertmanager. You need to manually expand a persistent volume claim (PVC), and then update the config map in which the component is configured.

**IMPORTANT**

You can only expand the size of the PVC. Shrinking the storage size is not possible.

**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.
- A cluster administrator has enabled monitoring for user-defined projects.
- You have configured at least one PVC for components that monitor user-defined projects.
- You have installed the OpenShift CLI (**oc**).

**Procedure**

1. Manually expand a PVC with the updated storage request. For more information, see "Expanding persistent volume claims (PVCs) with a file system" in *Expanding persistent volumes*.
2. 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
```

3. Add a new storage size for the 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>: ❶
    volumeClaimTemplate:
      spec:
        resources:
          requests:
            storage: <amount_of_storage> ❷

```

- ❶ The component for which you want to change the storage size.
- ❷ Specify the new size for the storage volume. It must be greater than the previous value.

The following example sets the new PVC request to 20 gigabytes for Thanos Ruler:

### Example storage configuration for `thanosRuler`

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      volumeClaimTemplate:
        spec:
          resources:
            requests:
              storage: 20Gi

```



#### NOTE

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

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



#### WARNING

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

### Additional resources

- [Prometheus database storage requirements](#)
- [Expanding persistent volume claims \(PVCs\) with a file system](#)

### 4.3.2. 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 **cluster-admin** cluster role, or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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.

#### 4.3.2.1. 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.

##### 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.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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](#)
- [Enabling monitoring for user-defined projects](#)
- [Prometheus database storage requirements](#)
- [Recommended configurable storage technology](#)
- [Understanding persistent storage](#)
- [Optimizing storage](#)

### 4.3.3. Setting log levels for monitoring components

You can configure the log level for Alertmanager, Prometheus Operator, Prometheus, and Thanos Ruler.

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.** Log informational, warning, and error messages.
- **warn.** Log warning and error messages only.
- **error.** Log error messages only.

The default log level is **info**.

### 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.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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 **logLevel: <log\_level>** 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>: ❶
    logLevel: <log_level> ❷
```

- ❶ The monitoring stack component for which you are setting a log level. Available component values are **prometheus**, **alertmanager**, **prometheusOperator**, and **thanosRuler**.
- ❷ The log level to set 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. Confirm that the log level has been applied by reviewing the deployment or pod configuration in the related project. The following example checks the log level for the **prometheus-operator** deployment:

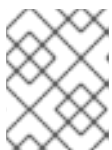
```
$ oc -n openshift-user-workload-monitoring get deploy prometheus-operator -o yaml | grep "log-level"
```

### Example output

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

5. Check that the pods for the component are running. The following example lists the status of pods:

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

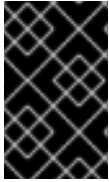


### NOTE

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

## 4.3.4. 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 **cluster-admin** cluster role or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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-776fcbdd56-2nbfm 2/2 Running 0 132m
prometheus-user-workload-0 5/5 Running 1 132m
prometheus-user-workload-1 5/5 Running 1 132m
```

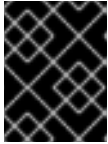
```

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.

#### Additional resources

- [Enabling monitoring for user-defined projects](#)

## 4.4. 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](#)

### 4.4.1. 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 **cluster-admin** cluster role or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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> ❶

```

- ❶ 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

- [writeRelabelConfigs](#)
- [relabel\\_config](#) (Prometheus documentation)

#### 4.4.1.1. 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	<b>sigv4</b>	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	<b>basicAuth</b>	Basic authentication sets the authorization header on every remote write request with the configured username and password.
authorization	<b>authorization</b>	Authorization sets the <b>Authorization</b> header on every remote write request using the configured token.
OAuth 2.0	<b>oauth2</b>	An OAuth 2.0 configuration uses the client credentials grant type. Prometheus fetches an access token from <b>tokenUrl</b> 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	<b>tlsConfig</b>	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.

#### 4.4.1.2. 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.

##### 4.4.1.2.1. 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> ❶
  secretKey: <AWS_secret_key> ❷
type: Opaque

```

❶ The AWS API access key.

❷ 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> ❶
      accessKey:
        name: sigv4-credentials ❷
        key: accessKey ❸
      secretKey:
        name: sigv4-credentials ❹
        key: secretKey ❺
      profile: <AWS_profile_name> ❻
      roleArn: <AWS_role_arn> ❼

```

- ❶ The AWS region.
- ❷ ❹ The name of the **Secret** object containing the AWS API access credentials.
- ❸ The key that contains the AWS API access key in the specified **Secret** object.
- ❺ The key that contains the AWS API secret key in the specified **Secret** object.
- ❻ The name of the AWS profile that is being used to authenticate.
- ❼ The unique identifier for the Amazon Resource Name (ARN) assigned to your role.

#### 4.4.1.2.2. 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> ❶
  password: <basic_password> ❷
type: Opaque

```

- ❶ The username.
- ❷ 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 ❶
          key: user ❷
        password:
          name: rw-basic-auth ❸
          key: password ❹

```

- ❶ ❸ The name of the **Secret** object that contains the authentication credentials.
- ❷ The key that contains the username in the specified **Secret** object.
- ❹ The key that contains the password in the specified **Secret** object.

#### 4.4.1.2.3. 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> ❶
type: Opaque

```

- ❶ 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: |
    enableUserWorkload: true
    prometheus:
      remoteWrite:
        - url: "https://authorization.example.com/api/write"
      authorization:
        type: Bearer ❶

```

```
credentials:
  name: rw-bearer-auth ❷
  key: token ❸
```

- ❶ The authentication type of the request. The default value is **Bearer**.
- ❷ The name of the **Secret** object that contains the authentication credentials.
- ❸ The key that contains the authentication token in the specified **Secret** object.

#### 4.4.1.2.4. 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> ❶
  secret: <oauth2_secret> ❷
type: Opaque
```

- ❶ The OAuth 2.0 ID.
- ❷ 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 ❶
                key: id ❷
            clientSecret:
              name: oauth2-credentials ❸
              key: secret ❹
            tokenUrl: https://example.com/oauth2/token ❺
            scopes: ❻
```

```

- <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.

#### 4.4.1.2.5. 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 ❶
      key: ca.crt ❷
  cert:
    secret:
      name: mtls-bundle ❸
      key: client.crt ❹
  keySecret:
    name: mtls-bundle ❺
    key: client.key ❻

```

❶ ❸ ❺ 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.

❷ The key in the specified **Secret** object that contains the CA certificate for the endpoint.

❹ The key in the specified **Secret** object that contains the client certificate for the endpoint.

❻ The key in the specified **Secret** object that contains the client key secret.

#### 4.4.1.3. 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 ❶
        minShards: 1 ❷
        maxShards: 50 ❸
        maxSamplesPerSend: 2000 ❹
        batchSendDeadline: 5s ❺
        minBackoff: 30ms ❻
        maxBackoff: 5s ❼
        retryOnRateLimit: false ❽
        sampleAgeLimit: 0s ❾

```



- 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

- [Prometheus REST API reference for remote write](#)
- [Setting up remote write compatible endpoints](#) (Prometheus documentation)
- [Tuning remote write settings](#) (Prometheus documentation)
- [Understanding secrets](#)

#### 4.4.2. 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.



#### NOTE

When Prometheus scrapes user workload targets that expose a **namespace** label, the system stores this label as **exported\_namespace**. This behavior ensures that the final namespace label value is equal to the namespace of the target pod. You cannot override this default configuration by setting the value of the **honorLabels** field to **true** for **PodMonitor** or **ServiceMonitor** objects.

#### 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.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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
- 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](#)

### 4.4.3. 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.

#### 4.4.3.1. 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

```

#### 4.4.3.2. Specifying how a service is monitored

To use the metrics exposed by your service, you must configure OpenShift Container Platform 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 **cluster-admin** cluster role or the **monitoring-edit** cluster role.
- You have enabled monitoring for user-defined projects.
- 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 ❶
spec:
  endpoints:
    - interval: 30s
      port: web ❷
      scheme: http
  selector: ❸
    matchLabels:
      app: prometheus-example-app
```

- ❶ Specify a user-defined namespace where your service runs.
- ❷ Specify endpoint ports to be scraped by Prometheus.
- ❸ 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
```

#### 4.4.3.3. 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.

##### 4.4.3.3.1. 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 ❶
          name: example-bearer-auth ❷
      port: web
  selector:
    matchLabels:
      app: prometheus-example-app

```

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

❷ 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.

#### 4.4.3.3.2. 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> ❶
  password: <basic_password> ❷

```

❶ Specify a username for authentication.

❷ 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 ❶
          name: example-basic-auth ❷
        password:
          key: password ❸
          name: example-basic-auth ❹
    port: web
  selector:
    matchLabels:
      app: prometheus-example-app

```

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

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

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

#### 4.4.3.3.3. 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> ❶
  secret: <oauth2_secret> ❷

```

❶ Specify an OAuth 2.0 ID.

❷ 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 ❶
          name: example-oauth2 ❷
      clientSecret:
        key: secret ❸
        name: example-oauth2 ❹
      tokenUrl: https://example.com/oauth2/token ❺
  port: web
  selector:
    matchLabels:
      app: prometheus-example-app
```

- ❶ The key that contains the OAuth 2.0 ID in the specified **Secret** object.
- ❷❹ The name of the **Secret** object that contains the OAuth 2.0 credentials.
- ❸ The key that contains the OAuth 2.0 secret in the specified **Secret** object.
- ❺ The URL used to fetch a token with the specified **clientId** and **clientSecret**.

#### Additional resources

- [Enabling monitoring for user-defined projects](#)
- [Scrape Prometheus metrics using TLS in ServiceMonitor configuration](#) (Red Hat Customer Portal article)
- [PodMonitor API](#)
- [ServiceMonitor API](#)

## 4.5. 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.

### 4.5.1. Configuring external Alertmanager instances

The OpenShift Container Platform 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 **cluster-admin** cluster role or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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.

## 4.5.2. Configuring secrets for Alertmanager

The OpenShift Container Platform 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.

### 4.5.2.1. 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 **cluster-admin** cluster role or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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: ❶
      - <secret_name_1> ❷
      - <secret_name_2>
```

- ❶ This section contains the secrets to be mounted into Alertmanager. The secrets must be located within the same namespace as the Alertmanager object.
- ❷ 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.

### 4.5.3. 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 **cluster-admin** cluster role or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.

- A cluster administrator has enabled monitoring for user-defined projects.
- 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** or **managed\_cluster** as key names. Using them 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.

### Additional resources

- [Enabling monitoring for user-defined projects](#)

#### 4.5.4. Configuring alert notifications

In OpenShift Container Platform, an administrator can enable alert routing for user-defined projects with one of the following methods:

- Use the default platform Alertmanager instance.
- Use a separate Alertmanager instance only 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](#) (PagerDuty official site)
- [Prometheus Integration Guide](#) (PagerDuty official site)
- [Support version matrix for monitoring components](#)
- [Enabling alert routing for user-defined projects](#)

#### 4.5.4.1. 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

- A cluster administrator has enabled monitoring for user-defined projects.
- A cluster administrator has enabled alert routing 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)

#### 4.5.4.2. 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 Container Platform 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 **cluster-admin** cluster role.
- You have enabled a separate instance of Alertmanager for user-defined alert routing.
- 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 ❶
route:
  receiver: Default
  group_by:
  - name: Default
  routes:
  - matchers:
    - "service = prometheus-example-monitor" ❷
    receiver: <receiver> ❸
  receivers:
  - name: Default
  - name: <receiver>
    <receiver_configuration> ❹
```

- ❶ If you configured an HTTP cluster-wide proxy, set the **proxy\_from\_environment** parameter to **true** to enable proxying for all alert receivers.
- ❷ Specify labels to match your alerts. This example targets all alerts that have the **service="prometheus-example-monitor"** label.
- ❸ Specify the name of the receiver to use for the alerts group.
- ❹ 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)

#### 4.5.4.3. 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 5. ACCESSING METRICS

### 5.1. ACCESSING METRICS AS AN ADMINISTRATOR

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

#### Additional resources

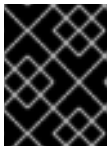
- [Understanding metrics](#)

#### 5.1.1. Viewing a list of available metrics

As a cluster administrator or as a user with view permissions for all projects, you can view a list of metrics available in a cluster and output the list in JSON format.

#### Prerequisites

- You are a cluster administrator, or you have access to the cluster as a user with the **cluster-monitoring-view** cluster role.
- You have installed the OpenShift Container Platform CLI (**oc**).
- You have obtained the OpenShift Container Platform API route for Thanos Querier.
- You are able to get a bearer token by using the **oc whoami -t** command.



#### IMPORTANT

You can only use bearer token authentication to access the Thanos Querier API route.

#### Procedure

1. If you have not obtained the OpenShift Container Platform API route for Thanos Querier, run the following command:

```
$ oc get routes -n openshift-monitoring thanos-querier -o jsonpath='{.status.ingress[0].host}'
```

2. Retrieve a list of metrics in JSON format from the Thanos Querier API route by running the following command. This command uses **oc** to authenticate with a bearer token.

```
$ curl -k -H "Authorization: Bearer $(oc whoami -t)"  
https://<thanos_querier_route>/api/v1/metadata 1
```

- 1** Replace **<thanos\_querier\_route>** with the OpenShift Container Platform API route for Thanos Querier.

#### 5.1.2. Querying metrics for all projects with the OpenShift Container Platform web console

You can use the OpenShift Container Platform metrics query browser to run Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot. This functionality provides information about the state of a cluster and any user-defined workloads that you are monitoring.

As a cluster administrator or as a user with view permissions for all projects, you can access metrics for all default OpenShift Container Platform and user-defined projects in the Metrics UI.



The Metrics UI includes predefined queries, for example, CPU, memory, bandwidth, or network packet for all projects. You can also run custom Prometheus Query Language (PromQL) queries.

## Prerequisites

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

## Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform web console, click **Observe** and go to the **Metrics** tab.
2. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the <b>Select query</b> drop-down list, select an existing query.
Create a custom query.	<p>Add your Prometheus Query Language (PromQL) query to the <b>Expression</b> field.</p> <p>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.</p>
Add multiple queries.	Click <b>Add query</b> .
Duplicate an existing query.	<p>Click the options menu  next to the query, then choose <b>Duplicate query</b>.</p>
Disable a query from being run.	<p>Click the options menu  next to the query and choose <b>Disable query</b>.</p>


- 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.

- Optional: Save the page URL to use this set of queries again in the future.
- 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 <b>Hide all series</b> .
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: <ul style="list-style-type: none"> <li>Visually select the time range by clicking and dragging on the plot horizontally.</li> <li>Use the menu to select the time range.</li> </ul>
Reset the time range.	Click <b>Reset zoom</b> .
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 <b>Hide graph</b> .

## Additional resources

- [Querying Prometheus](#) (Prometheus documentation)

## 5.1.3. Getting detailed information about a metrics target

You can use the OpenShift Container Platform 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 Container Platform monitoring is not able to scrape metrics from a targeted component.

The **Metrics targets** page shows targets for default OpenShift Container Platform projects and for user-defined projects.

### Prerequisites

- You have access to the cluster as an administrator for the project for which you want to view metrics targets.

### Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform 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 Container Platform 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
--------	-------------

Option	Description
Filter the targets by status and source.	<p>Choose filters in the <b>Filter</b> list.</p> <p>The following filtering options are available:</p> <ul style="list-style-type: none"> <li>● <b>Status</b> filters: <ul style="list-style-type: none"> <li>○ <b>Up</b>. The target is currently up and being actively scraped for metrics.</li> <li>○ <b>Down</b>. The target is currently down and not being scraped for metrics.</li> </ul> </li> <li>● <b>Source</b> filters: <ul style="list-style-type: none"> <li>○ <b>Platform</b>. 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.</li> <li>○ <b>User</b>. User targets relate to user-defined projects. These projects are user-created and can be customized.</li> </ul> </li> </ul>
Search for a target by name or label.	Enter a search term in the <b>Text</b> or <b>Label</b> field next to the search box.
Sort the targets.	Click one or more of the <b>Endpoint Status</b> , <b>Namespace</b> , <b>Last Scrape</b> , and <b>Scrape Duration</b> column headers.

- 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

#### 5.1.4. Reviewing monitoring dashboards as a cluster administrator

In the **Administrator** perspective, you can view dashboards relating to core OpenShift Container Platform cluster components.

#### Prerequisites

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

### Procedure

1. In the **Administrator** perspective of the OpenShift Container Platform 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 pre-defined 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](#)

## 5.2. ACCESSING METRICS AS A DEVELOPER

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

### Additional resources

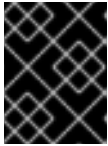
- [Understanding metrics](#)

### 5.2.1. Viewing a list of available metrics

As a cluster administrator or as a user with view permissions for all projects, you can view a list of metrics available in a cluster and output the list in JSON format.

### Prerequisites

- You are a cluster administrator, or you have access to the cluster as a user with the **cluster-monitoring-view** cluster role.
- You have installed the OpenShift Container Platform CLI (**oc**).
- You have obtained the OpenShift Container Platform API route for Thanos Querier.
- You are able to get a bearer token by using the **oc whoami -t** command.



## IMPORTANT

You can only use bearer token authentication to access the Thanos Querier API route.

### Procedure

1. If you have not obtained the OpenShift Container Platform API route for Thanos Querier, run the following command:

```
$ oc get routes -n openshift-monitoring thanos-querier -o jsonpath='{.status.ingress[0].host}'
```

2. Retrieve a list of metrics in JSON format from the Thanos Querier API route by running the following command. This command uses **oc** to authenticate with a bearer token.

```
$ curl -k -H "Authorization: Bearer $(oc whoami -t)"  
https://<thanos_querier_route>/api/v1/metadata 1
```

- 1 Replace **<thanos\_querier\_route>** with the OpenShift Container Platform API route for Thanos Querier.

### 5.2.2. Querying metrics for user-defined projects with the OpenShift Container Platform web console

You can use the OpenShift Container Platform metrics query browser to run Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot. This functionality provides information about any user-defined workloads that you are monitoring.

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.

The Metrics UI includes predefined queries, for example, CPU, memory, bandwidth, or network packet. These queries are restricted to the selected project. You can also run custom Prometheus Query Language (PromQL) queries for the project.



## NOTE

Developers can only use the **Developer** perspective and not the **Administrator** perspective. As a developer, you can only query metrics for one project at a time.



### 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 **Developer** perspective of the OpenShift Container Platform web console, click **Observe** and go to the **Metrics** tab.
2. Select the project that you want to view metrics for from the **Project:** list.
3. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the <b>Select query</b> drop-down list, select an existing query.
Create a custom query.	<p>Add your Prometheus Query Language (PromQL) query to the <b>Expression</b> field.</p> <p>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.</p>
Add multiple queries.	Click <b>Add query</b> .
Duplicate an existing query.	 <p>Click the options menu next to the query, then choose <b>Duplicate query</b>.</p>
Disable a query from being run.	 <p>Click the options menu next to the query and choose <b>Disable query</b>.</p>

4. 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.

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

6. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot.

- b. 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 <b>Hide all series</b> .
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: <ul style="list-style-type: none"> <li>Visually select the time range by clicking and dragging on the plot horizontally.</li> <li>Use the menu to select the time range.</li> </ul>
Reset the time range.	Click <b>Reset zoom</b> .
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 <b>Hide graph</b> .

### Additional resources

- [Querying Prometheus](#) (Prometheus documentation)

### 5.2.3. Reviewing monitoring dashboards as a developer

In the **Developer** perspective, you can view dashboards relating to a selected project.



#### NOTE

In the **Developer** perspective, you can view dashboards for only one project at a time.

### 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.

### Procedure

- In the Developer perspective in the OpenShift Container Platform web console, click **Observe** and go to the **Dashboards** tab.
- Select a project from the **Project:** drop-down list.

3. Select a dashboard from the **Dashboard** drop-down list to see the filtered metrics.



#### NOTE

All dashboards produce additional sub-menus when selected, except **Kubernetes / Compute Resources / Namespace (Pods)**

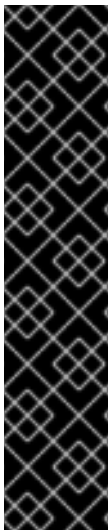
4. Optional: Select a time range for the graphs in the **Time Range** list.
  - Select a pre-defined time period.
  - Set a custom time range by clicking **Custom time range** in the **Time Range** list.
    - a. Input or select the **From** and **To** dates and times.
    - b. Click **Save** to save 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](#)

## 5.3. ACCESSING MONITORING APIS BY USING THE CLI

In OpenShift Container Platform, 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:

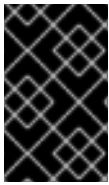
- Avoid querying endpoints frequently. Limit queries to a maximum of one every 30 seconds.
- Do not retrieve all metrics data through the **/federate** endpoint for Prometheus. Query the endpoint only when you want to retrieve a limited, aggregated data set. For example, retrieving fewer than 1,000 samples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

### 5.3.1. About accessing monitoring web service APIs

You can directly access web service API endpoints from the command line for the following monitoring stack components:

- Prometheus

- Alertmanager
- Thanos Ruler
- Thanos Querier



### 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 Container Platform web console to review monitoring dashboards.

### Additional resources

- [Reviewing monitoring dashboards as a cluster administrator](#)
- [Reviewing monitoring dashboards as a developer](#)

## 5.3.2. 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"
```

### 5.3.3. 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 Container Platform 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
...
```

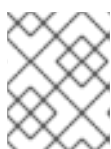
### 5.3.4. Accessing metrics from outside the cluster for custom applications

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.

### 5.3.5. 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.

#### 5.3.5.1. CMO routes resources

##### 5.3.5.1.1. openshift-monitoring/alertmanager-main

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

##### 5.3.5.1.2. openshift-monitoring/prometheus-k8s

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

##### 5.3.5.1.3. openshift-monitoring/prometheus-k8s-federate

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

##### 5.3.5.1.4. openshift-user-workload-monitoring/federate

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



#### 5.3.5.1.5. openshift-monitoring/thanos-querier

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

#### 5.3.5.1.6. openshift-user-workload-monitoring/thanos-ruler

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

### 5.3.5.2. CMO services resources

#### 5.3.5.2.1. openshift-monitoring/prometheus-operator-admission-webhook

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

#### 5.3.5.2.2. 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 **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.

#### 5.3.5.2.3. 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** (for read-only operations) or **monitoring-alertmanager-edit** role in the **openshift-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.

#### 5.3.5.2.4. 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.

#### 5.3.5.2.5. 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.

#### 5.3.5.2.6. openshift-monitoring/monitoring-plugin

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

#### 5.3.5.2.7. openshift-monitoring/node-exporter

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

#### 5.3.5.2.8. 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.

#### 5.3.5.2.9. 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.
- Port 9092 provides access to the **/metrics** and **/federate** endpoints only. This port is for internal use, and no other usage is guaranteed.

#### 5.3.5.2.10. 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.

#### 5.3.5.2.11. openshift-monitoring/prometheus-operator

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

#### 5.3.5.2.12. 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.

#### 5.3.5.2.13. openshift-monitoring/telemeter-client

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

#### 5.3.5.2.14. 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.
- 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.
- 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.
- Port 9094 provides access to the **/metrics** endpoint only. This port is for internal use, and no other usage is guaranteed.

#### 5.3.5.2.15. 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.

#### 5.3.5.2.16. 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.

### 5.3.6. Additional resources

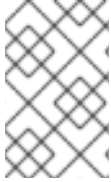
- [Enabling monitoring for user-defined projects](#)
- [Configuring remote write storage for core platform monitoring](#)
- [Configuring remote write storage for monitoring of user-defined projects](#)
- [Accessing metrics as an administrator](#)
- [Accessing metrics as a developer](#)

- [Managing alerts as an Administrator](#)
- [Managing alerts as a Developer](#)

## CHAPTER 6. MANAGING ALERTS

### 6.1. MANAGING ALERTS AS AN ADMINISTRATOR

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



#### 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.

#### 6.1.1. Accessing the Alerting UI from the Administrator perspective

The Alerting UI is accessible through the **Administrator** perspective of the OpenShift Container Platform web console.

- From the **Administrator** perspective, go to **Observe → Alerting**. The three main pages in the Alerting UI in this perspective are the **Alerts**, **Silences**, and **Alerting rules** pages.

#### Additional resources

- [Searching and filtering alerts, silences, and alerting rules](#)

#### 6.1.2. Getting information about alerts, silences, and alerting rules from the Administrator perspective

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. From the **Administrator** perspective of the OpenShift Container Platform 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. From the **Administrator** perspective of the OpenShift Container Platform 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. From the **Administrator** perspective of the OpenShift Container Platform 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

- [Cluster Monitoring Operator runbooks](#) (Cluster Monitoring Operator GitHub repository)

### 6.1.3. Managing silences

You can create a silence for an alert in the OpenShift Container Platform web console in the **Administrator** perspective. 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](#)

#### 6.1.3.1. Silencing alerts from the Administrator perspective


You can silence a specific alert or silence alerts that match a specification that you define.

#### Prerequisites

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

#### Procedure

To silence a specific alert:

1. From the **Administrator** perspective of the OpenShift Container Platform 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. From the **Administrator** perspective of the OpenShift Container Platform 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**.


### 6.1.3.2. Editing silences from the Administrator perspective

You can edit a silence, which expires the existing silence and creates a new one with the changed configuration.

#### Prerequisites

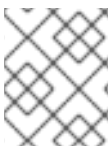
- If you are a cluster administrator, you have access to the cluster as a user with the **cluster-admin** role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **cluster-monitoring-view** cluster role, which allows you to access Alertmanager.
  - The **monitoring-alertmanager-edit** role, which permits you to create and silence alerts in the **Administrator** perspective in the web console.

#### Procedure

1. From the **Administrator** perspective of the OpenShift Container Platform 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.

### 6.1.3.3. Expiring silences from the Administrator perspective

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 **cluster-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 in the **Administrator** perspective in the web console.

## 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.

### 6.1.4. Managing alerting rules for core platform monitoring

The OpenShift Container Platform monitoring includes a large set of default alerting rules for platform metrics. As a cluster administrator, you can customize this set of rules in two ways:

- Modify the settings for existing platform alerting rules by adjusting thresholds or by adding and modifying labels. For example, you can change the **severity** label for an alert from **warning** to **critical** to help you route and triage issues flagged by an alert.
- Define and add new custom alerting rules by constructing a query expression based on core platform metrics in the **openshift-monitoring** project.

## Additional resources

- [Managing alerting rules for core platform monitoring](#)
- [Tips for optimizing alerting rules for core platform monitoring](#)

#### 6.1.4.1. Creating new alerting rules

As a cluster administrator, you can create new alerting rules based on platform metrics. These alerting rules trigger alerts based on the values of chosen metrics.



## NOTE

- If you create a customized **AlertingRule** resource based on an existing platform alerting rule, silence the original alert to avoid receiving conflicting alerts.
- To help users understand the impact and cause of the alert, ensure that your alerting rule contains an alert message and severity value.

## Prerequisites

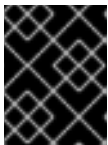
- You have access to the cluster as a user that has the **cluster-admin** cluster role.
- You have installed the OpenShift CLI (**oc**).

## Procedure

1. Create a new YAML configuration file named **example-alerting-rule.yaml**.
2. Add an **AlertingRule** resource to the YAML file. The following example creates a new alerting rule named **example**, similar to the default **Watchdog** alert:

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: example
  namespace: openshift-monitoring 1
spec:
  groups:
  - name: example-rules
    rules:
    - alert: ExampleAlert 2
      for: 1m 3
      expr: vector(1) 4
      labels:
        severity: warning 5
      annotations:
        message: This is an example alert. 6
```

- 1** Ensure that the namespace is **openshift-monitoring**.
- 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.
- 5** The severity that alerting rule assigns to the alert.
- 6** The message associated with the alert.



### IMPORTANT

You must create the **AlertingRule** object in the **openshift-monitoring** namespace. Otherwise, the alerting rule is not accepted.

3. Apply the configuration file to the cluster:

```
$ oc apply -f example-alerting-rule.yaml
```

## 6.1.4.2. Modifying core platform alerting rules

As a cluster administrator, you can modify core platform alerts before Alertmanager routes them to a receiver. For example, you can change the severity label of an alert, add a custom label, or exclude an alert from being sent to Alertmanager.

## Prerequisites

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

## Procedure

1. Create a new YAML configuration file named **example-modified-alerting-rule.yaml**.
2. Add an **AlertRelabelConfig** resource to the YAML file. The following example modifies the **severity** setting to **critical** for the default platform **watchdog** alerting rule:

```
apiVersion: monitoring.openshift.io/v1
kind: AlertRelabelConfig
metadata:
  name: watchdog
  namespace: openshift-monitoring 1
spec:
  configs:
  - sourceLabels: [alertname,severity] 2
    regex: "Watchdog;none" 3
    targetLabel: severity 4
    replacement: critical 5
    action: Replace 6
```

- 1** Ensure that the namespace is **openshift-monitoring**.
- 2** The source labels for the values you want to modify.
- 3** The regular expression against which the value of **sourceLabels** is matched.
- 4** The target label of the value you want to modify.
- 5** The new value to replace the target label.
- 6** The relabel action that replaces the old value based on regex matching. The default action is **Replace**. Other possible values are **Keep**, **Drop**, **HashMod**, **LabelMap**, **LabelDrop**, and **LabelKeep**.



### IMPORTANT

You must create the **AlertRelabelConfig** object in the **openshift-monitoring** namespace. Otherwise, the alert label will not change.

3. Apply the configuration file to the cluster:

```
$ oc apply -f example-modified-alerting-rule.yaml
```

## Additional resources

- [Monitoring stack architecture](#)
- [Alertmanager](#) (Prometheus documentation)
- [relabel\\_config configuration](#) (Prometheus documentation)
- [Alerting](#) (Prometheus documentation)

## 6.1.5. Managing alerting rules for user-defined projects

In OpenShift Container Platform, 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](#)

### 6.1.5.1. 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 ❶
      for: 1m ❷
      expr: version{job="prometheus-example-app"} == 0 ❸
      labels:
        severity: warning ❹
      annotations:
        message: This is an example alert. ❺

```

- ❶ The name of the alerting rule you want to create.
- ❷ The duration for which the condition should be true before an alert is fired.
- ❸ The PromQL query expression that defines the new rule.
- ❹ The severity that alerting rule assigns to the alert.
- ❺ The message associated with the alert.

3. Apply the configuration file to the cluster:

```
$ oc apply -f example-app-alerting-rule.yaml
```

### 6.1.5.2. 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

- If you are a cluster administrator, you have access to the cluster as a user with the **cluster-admin** cluster role.
- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project to edit the **user-workload-monitoring-config** config map.
  - The **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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
    # ...
```

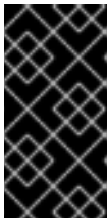
- 1 Specify one or more projects in which you want to create cross-project alerting rules. Prometheus and Thanos Ruler for user-defined monitoring do not enforce the **namespace** label in **PrometheusRule** objects created in these projects, making the **PrometheusRule** objects applicable to all projects.

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)

### 6.1.5.3. Listing alerting rules for all projects in a single view

As a cluster administrator, you can list alerting rules for core OpenShift Container Platform and user-defined projects together in a single view.

### Prerequisites

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

### Procedure

1. From the **Administrator** perspective of the OpenShift Container Platform 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.

#### 6.1.5.4. 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>
```

#### 6.1.5.5. 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 **cluster-admin** cluster 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)

## 6.2. MANAGING ALERTS AS A DEVELOPER

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



### NOTE

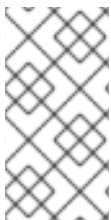
The alerts, silences, and alerting rules that are available in the Alerting UI relate to the projects that you have access to.

### 6.2.1. Accessing the Alerting UI from the Developer perspective

The Alerting UI is accessible through the **Developer** perspective of the OpenShift Container Platform web console.

- From the **Developer** perspective, go to **Observe** and go to the **Alerts** tab.
- Select the project that you want to manage alerts for from the **Project:** list.

In this perspective, alerts, silences, and alerting rules are all managed from the **Alerts** tab. The results shown in the **Alerts** tab are specific to the selected project.



### NOTE

In the **Developer** perspective, you can select from core OpenShift Container Platform and user-defined projects that you have access to in the **Project:** <project\_name> list. However, alerts, silences, and alerting rules relating to core OpenShift Container Platform projects are not displayed if you are not logged in as a cluster administrator.

### Additional resources

- [Searching and filtering alerts, silences, and alerting rules](#)

### 6.2.2. Getting information about alerts, silences, and alerting rules from the Developer perspective


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, silences, and alerting rules:

1. From the **Developer** perspective of the OpenShift Container Platform web console, go to the **Observe** → **<project\_name>** → **Alerts** page.
2. View details for an alert, silence, or an alerting rule:
  - **Alert details** can be viewed by clicking a greater than symbol ( >) next to an alert name and then selecting the alert from the list.
  - **Silence details** can be viewed by clicking a silence in the **Silenced by** section of the **Alert details** page. The **Silence details** page includes the following information:
    - Alert specification
    - Start time
    - End time
    - Silence state
    - Number and list of firing alerts
  - **Alerting rule details** can be viewed by clicking the  menu next to an alert in the **Alerts** page and then clicking **View Alerting Rule**.



### NOTE

Only alerts, silences, and alerting rules relating to the selected project are displayed in the **Developer** perspective.

## Additional resources

- [Cluster Monitoring Operator runbooks](#) (Cluster Monitoring Operator GitHub repository)

### 6.2.3. Managing silences

You can create a silence for an alert in the OpenShift Container Platform web console in the **Developer** perspective. 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](#)

### 6.2.3.1. Silencing alerts from the Developer perspective

You can silence a specific alert or silence alerts that match a specification that you define.

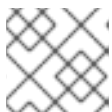
#### Prerequisites

- If you are a cluster administrator, you have access to the cluster as a user with the **cluster-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 in the **Administrator** perspective in the web console.
  - The **monitoring-rules-edit** cluster role, which permits you to create and silence alerts in the **Developer** perspective in the web console.

#### Procedure

To silence a specific alert:

1. From the **Developer** perspective of the OpenShift Container Platform web console, go to **Observe** and go to the **Alerts** tab.
2. Select the project that you want to silence an alert for from the **Project:** list.
3. If necessary, expand the details for the alert by clicking a greater than symbol (>) next to the alert name.
4. Click the alert message in the expanded view to open the **Alert details** page for the alert.
5. Click **Silence alert** to open the **Silence alert** page with a default configuration for the alert.
6. Optional: Change the default configuration details for the silence.



#### NOTE

You must add a comment before saving a silence.

7. To save the silence, click **Silence**.

To silence a set of alerts:

1. From the **Developer** perspective of the OpenShift Container Platform web console, go to **Observe** and go to the **Silences** tab.
2. Select the project that you want to silence alerts for from the **Project:** list.
3. Click **Create silence**.
4. On the **Create silence** page, set the duration and label details for an alert.

**NOTE**

You must add a comment before saving a silence.

- To create silences for alerts that match the labels that you entered, click **Silence**.


### 6.2.3.2. Editing silences from the Developer perspective

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 **cluster-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-rules-edit** cluster role, which permits you to create and silence alerts in the **Developer** perspective in the web console.

#### Procedure

- From the **Developer** perspective of the OpenShift Container Platform web console, go to **Observe** and go to the **Silences** tab.
- Select the project that you want to edit silences for from the **Project:** list.
- 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.
- On the **Edit silence** page, make changes and click **Silence**. Doing so expires the existing silence and creates one with the updated configuration.

### 6.2.3.3. Expiring silences from the Developer perspective

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 **cluster-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-rules-edit** cluster role, which permits you to create and silence alerts in the **Developer** perspective in the web console.

### Procedure

1. From the **Developer** perspective of the OpenShift Container Platform web console, go to **Observe** and go to the **Silences** tab.
2. Select the project that you want to expire a silence for from the **Project:** list.
3. For the silence or silences you want to expire, select the checkbox in the corresponding row.
4. 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.

## 6.2.4. Managing alerting rules for user-defined projects

In OpenShift Container Platform, 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](#)

### 6.2.4.1. 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
```

#### 6.2.4.2. 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

- If you are a cluster administrator, you have access to the cluster as a user with the **cluster-admin** cluster role.

- If you are a non-administrator user, you have access to the cluster as a user with the following user roles:
  - The **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project to edit the **user-workload-monitoring-config** config map.
  - The **monitoring-rules-edit** cluster role for the project where you want to create an alerting rule.
- A cluster administrator has enabled monitoring for user-defined projects.
- 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" ❷
    for: 5m ❸
    expr: kube_namespace_labels{namespace!~"
(openshift|kube).*"|default",label_pod_security_kubernetes_io_enforce!="restricted"} ❹
    annotations:
      message: "Restricted policy not enforced. Project {{ $labels.namespace }} does not
enforce the restricted pod security policy." ❺
    labels:
      severity: warning ❻

```

- ❶ Ensure that you specify the project that you defined in the **namespacesWithoutLabelEnforcement** field.
- ❷ The name of the alerting rule you want to create.
- ❸ The duration for which the condition should be true before an alert is fired.
- ❹ 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.
- ❺ The message associated with the alert.
- ❻ 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)

### 6.2.4.3. 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
```

#### 6.2.4.4. 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 7. TROUBLESHOOTING MONITORING ISSUES

Find troubleshooting steps for common issues with core platform and user-defined project monitoring.

### 7.1. INVESTIGATING WHY USER-DEFINED PROJECT METRICS ARE UNAVAILABLE

**ServiceMonitor** resources enable you to determine how to use the metrics exposed by a service in user-defined projects. Follow the steps outlined in this procedure if you have created a **ServiceMonitor** resource but cannot see any corresponding metrics in the Metrics UI.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).
- You have enabled and configured monitoring for user-defined projects.
- You have created a **ServiceMonitor** resource.

#### Procedure

1. Ensure that your project and resources are not excluded from user workload monitoring. The following examples use the **ns1** project.
  - a. Verify that the project *does not* have the **openshift.io/user-monitoring=false** label attached:

```
$ oc get namespace ns1 --show-labels | grep 'openshift.io/user-monitoring=false'
```



#### NOTE

The default label set for user workload projects is **openshift.io/user-monitoring=true**. However, the label is not visible unless you manually apply it.

- b. Verify that the **ServiceMonitor** and **PodMonitor** resources *do not* have the **openshift.io/user-monitoring=false** label attached. The following example checks the **prometheus-example-monitor** service monitor.

```
$ oc -n ns1 get servicemonitor prometheus-example-monitor --show-labels | grep 'openshift.io/user-monitoring=false'
```

- c. If the label is attached, remove the label:

#### Example of removing the label from the project

```
$ oc label namespace ns1 'openshift.io/user-monitoring='
```

#### Example of removing the label from the resource

```
$ oc -n ns1 label servicemonitor prometheus-example-monitor 'openshift.io/user-monitoring-'
```

### Example output

```
namespace/ns1 unlabeled
```

2. Check that the corresponding labels match in the service and **ServiceMonitor** resource configurations. The following examples use the **prometheus-example-app** service, the **prometheus-example-monitor** service monitor, and the **ns1** project.

- a. Obtain the label defined in the service.

```
$ oc -n ns1 get service prometheus-example-app -o yaml
```

### Example output

```
labels:
  app: prometheus-example-app
```

- b. Check that the **matchLabels** definition in the **ServiceMonitor** resource configuration matches the label output in the previous step.

```
$ oc -n ns1 get servicemonitor prometheus-example-monitor -o yaml
```

### Example output

```
apiVersion: v1
kind: ServiceMonitor
metadata:
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
    - interval: 30s
      port: web
      scheme: http
  selector:
    matchLabels:
      app: prometheus-example-app
```



### NOTE

You can check service and **ServiceMonitor** resource labels as a developer with view permissions for the project.

3. Inspect the logs for the Prometheus Operator in the **openshift-user-workload-monitoring** project.

- a. List the pods in the **openshift-user-workload-monitoring** project:

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

## Example output

NAME	READY	STATUS	RESTARTS	AGE
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

- b. Obtain the logs from the **prometheus-operator** container in the **prometheus-operator** pod. In the following example, the pod is called **prometheus-operator-776fcbbd56-2nbfm**:

```
$ oc -n openshift-user-workload-monitoring logs prometheus-operator-776fcbbd56-2nbfm -c prometheus-operator
```

If there is a issue with the service monitor, the logs might include an error similar to this example:

```
level=warn ts=2020-08-10T11:48:20.906739623Z caller=operator.go:1829
component=prometheusoperator msg="skipping servicemonitor" error="it accesses file
system via bearer token file which Prometheus specification prohibits"
servicemonitor=eagle/eagle namespace=openshift-user-workload-monitoring
prometheus=user-workload
```

4. Review the target status for your endpoint on the **Metrics targets** page in the OpenShift Container Platform web console UI.
  - a. Log in to the OpenShift Container Platform web console and navigate to **Observe** → **Targets** in the **Administrator** perspective.
  - b. Locate the metrics endpoint in the list, and review the status of the target in the **Status** column.
  - c. If the **Status** is **Down**, click the URL for the endpoint to view more information on the **Target Details** page for that metrics target.
5. Configure debug level logging for the Prometheus Operator in the **openshift-user-workload-monitoring** project.
  - a. 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
```

- b. Add **logLevel: debug** for **prometheusOperator** under **data/config.yaml** to set the log level to **debug**:

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

```
prometheusOperator:
  logLevel: debug
# ...
```

- c. Save the file to apply the changes. The affected **prometheus-operator** pod is automatically redeployed.
- d. Confirm that the **debug** log-level has been applied to the **prometheus-operator** deployment in the **openshift-user-workload-monitoring** project:

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

### Example output

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

Debug level logging will show all calls made by the Prometheus Operator.

- e. Check that the **prometheus-operator** pod is running:

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



### NOTE

If an unrecognized Prometheus Operator **loglevel** value is included in the config map, the **prometheus-operator** pod might not restart successfully.

- f. Review the debug logs to see if the Prometheus Operator is using the **ServiceMonitor** resource. Review the logs for other related errors.

### Additional resources

- [Enabling monitoring for user-defined projects](#)
- [Specifying how a service is monitored](#)
- [Getting detailed information about a metrics target](#)

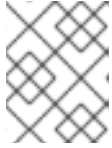
## 7.2. DETERMINING WHY PROMETHEUS IS CONSUMING A LOT OF DISK SPACE

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 **cluster-admin** cluster role.
- You have installed the OpenShift CLI (**oc**).

### Procedure

1. In the **Administrator** perspective, navigate 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 Container Platform 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 cluster administrator:

- 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](#)

## 7.3. 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 **cluster-admin** cluster 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 \ 1
-c prometheus --image=$(oc get po -n openshift-monitoring <prometheus_k8s_pod_name> \
2
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') \
-- sh -c 'cd /prometheus/;du -hs $(ls -dtr */ | grep -Eo "[0-9|A-Z]{26}")'
```

- 1 2** 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 01HVVH2A5Z58SKT810EM6B9AT50
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 \ 1
--image=$(oc get po -n openshift-monitoring <prometheus_k8s_pod_name> \ 2
-o jsonpath='{.spec.containers[?(@.name=="prometheus")].image}') -- df -h /prometheus/
```

- 1 2** 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 ...
```

## 7.4. RESOLVING THE ALERTMANAGERRECEIVERSNOTCONFIGURED ALERT

Every cluster that is deployed has the **AlertmanagerReceiversNotConfigured** alert firing by default. To resolve the issue, you must configure alert receivers.

- For default platform monitoring, follow the steps in "Configuring alert notifications" in *Configuring core platform monitoring*.
- For user workload monitoring, follow the steps in "Configuring alert notifications" in *Configuring user workload monitoring*.

### Additional resources

- [Configuring alert notifications for default platform monitoring](#)
- [Configuring alert notifications for user workload monitoring](#)

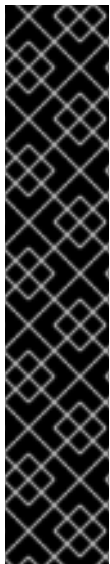
## CHAPTER 8. CONFIG MAP REFERENCE FOR THE CLUSTER MONITORING OPERATOR

### 8.1. CLUSTER MONITORING OPERATOR CONFIGURATION REFERENCE

Parts of OpenShift Container Platform cluster monitoring are configurable. The API is accessible by setting parameters defined in various config maps.

- To configure monitoring components, edit the **ConfigMap** object named **cluster-monitoring-config** in the **openshift-monitoring** namespace. These configurations are defined by [ClusterMonitoringConfiguration](#).
- 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.

### 8.2. ADDITIONALALERTMANAGERCONFIG

#### 8.2.1. Description

The **AdditionalAlertmanagerConfig** resource defines settings for how a component communicates with additional Alertmanager instances.

#### 8.2.2. Required

- **apiVersion**

Appears in: [PrometheusK8sConfig](#), [PrometheusRestrictedConfig](#), [ThanosRulerConfig](#)

Property	Type	Description
apiVersion	string	Defines the API version of Alertmanager. Possible values are <b>v1</b> or <b>v2</b> . The default is <b>v2</b> .
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 <b>http</b> or <b>https</b> . The default value is <b>http</b> .
staticConfigs	[]string	A list of statically configured Alertmanager endpoints in the form of <b>&lt;hosts&gt;:&lt;port&gt;</b> .
timeout	*string	Defines the timeout value used when sending alerts.
tlsConfig	<a href="#">TLSConfig</a>	Defines the TLS settings to use for Alertmanager connections.

## 8.3. ALERTMANAGERMAINCONFIG

### 8.3.1. 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 <b>openshift-monitoring</b> namespace. The default value is <b>true</b> .

Property	Type	Description
enableUserAlertmanagerConfig	bool	A Boolean flag that enables or disables user-defined namespaces to be selected for <b>AlertmanagerConfig</b> lookups. This setting only applies if the user workload monitoring instance of Alertmanager is not enabled. The default value is <b>false</b> .
logLevel	string	Defines the log level setting for Alertmanager. The possible values are: <b>error</b> , <b>warn</b> , <b>info</b> , <b>debug</b> . The default value is <b>info</b> .
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 <b>secret- &lt;secret-name&gt;</b> and mounted at <b>/etc/alertmanager/secrets/&lt;secret-name&gt;</b> in the <b>alertmanager</b> 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.

## 8.4. ALERTMANAGERUSERWORKLOADCONFIG

### 8.4.1. 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 <b>openshift-user-workload-monitoring</b> namespace. The default value is <b>false</b> .
enableAlertmanagerConfig	bool	A Boolean flag to enable or disable user-defined namespaces to be selected for <b>AlertmanagerConfig</b> lookup. The default value is <b>false</b> .
logLevel	string	Defines the log level setting for Alertmanager for user workload monitoring. The possible values are <b>error</b> , <b>warn</b> , <b>info</b> , and <b>debug</b> . The default value is <b>info</b> .
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 <b>secret-<i>&lt;secret-name&gt;</i></b> and mounted at <b>/etc/alertmanager/secrets/<i>&lt;secret-name&gt;</i></b> in the <b>alertmanager</b> 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.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

Property	Type	Description
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Alertmanager. Use this setting to configure the persistent volume claim, including storage class, volume size and name.

## 8.5. CLUSTERMONITORINGCONFIGURATION

### 8.5.1. 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	* <a href="#">AlertmanagerMainConfig</a>	<b>AlertmanagerMainConfig</b> defines settings for the Alertmanager component in the <b>openshift-monitoring</b> namespace.
enableUserWorkload	*bool	<b>UserWorkloadEnabled</b> is a Boolean flag that enables monitoring for user-defined projects.
userWorkload	* <a href="#">UserWorkloadConfig</a>	<b>UserWorkload</b> defines settings for the monitoring of user-defined projects.
kubeStateMetrics	* <a href="#">KubeStateMetricsConfig</a>	<b>KubeStateMetricsConfig</b> defines settings for the <b>kube-state-metrics</b> agent.
metricsServer	* <a href="#">MetricsServerConfig</a>	<b>MetricsServer</b> defines settings for the Metrics Server component.
prometheusK8s	* <a href="#">PrometheusK8sConfig</a>	<b>PrometheusK8sConfig</b> defines settings for the Prometheus component.

Property	Type	Description
prometheusOperator	<a href="#">*PrometheusOperatorConfig</a>	<b>PrometheusOperatorConfig</b> defines settings for the Prometheus Operator component.
prometheusOperatorAdmissionWebhook	<a href="#">*PrometheusOperatorAdmissionWebhookConfig</a>	<b>PrometheusOperatorAdmissionWebhookConfig</b> defines settings for the admission webhook component of Prometheus Operator.
openshiftStateMetrics	<a href="#">*OpenShiftStateMetricsConfig</a>	<b>OpenShiftMetricsConfig</b> defines settings for the <b>openshift-state-metrics</b> agent.
telemeterClient	<a href="#">*TelemeterClientConfig</a>	<b>TelemeterClientConfig</b> defines settings for the Telemeter Client component.
thanosQuerier	<a href="#">*ThanosQuerierConfig</a>	<b>ThanosQuerierConfig</b> defines settings for the Thanos Querier component.
nodeExporter	<a href="#">NodeExporterConfig</a>	<b>NodeExporterConfig</b> defines settings for the <b>node-exporter</b> agent.
monitoringPlugin	<a href="#">*MonitoringPluginConfig</a>	<b>MonitoringPluginConfig</b> defines settings for the monitoring <b>console-plugin</b> component.

## 8.6. KUBESTATEMETRICSCONFIG

### 8.6.1. 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 <b>KubeStateMetrics</b> container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

## 8.7. METRICSSERVERCONFIG

### 8.7.1. 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 <b>Metadata</b> , <b>Request</b> , <b>RequestResponse</b> , and <b>None</b> . The default value is <b>Metadata</b> .
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Metrics Server container.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

## 8.8. MONITORINGPLUGINCONFIG

### 8.8.1. 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 <b>console-plugin</b> container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

## 8.9. NODEEXPORTERCOLLECTORBUDDYINFOCONFIG

### 8.9.1. 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 <b>buddyinfo</b> collector.

## 8.10. NODEEXPORTERCOLLECTORCONFIG

### 8.10.1. Description

The **NodeExporterCollectorConfig** resource defines settings for individual collectors of the **node-exporter** agent.

Appears in: [NodeExporterConfig](#)

Property	Type	Description
cpufreq	<a href="#">NodeExporterCollectorCpufreqConfig</a>	Defines the configuration of the <b>cpufreq</b> collector, which collects CPU frequency statistics. Disabled by default.
tcpstat	<a href="#">NodeExporterCollectorTcpStatConfig</a>	Defines the configuration of the <b>tcpstat</b> collector, which collects TCP connection statistics. Disabled by default.

Property	Type	Description
netdev	<a href="#">NodeExporterCollectorNetDevConfig</a>	Defines the configuration of the <b>netdev</b> collector, which collects network devices statistics. Enabled by default.
netclass	<a href="#">NodeExporterCollectorNetClassConfig</a>	Defines the configuration of the <b>netclass</b> collector, which collects information about network devices. Enabled by default.
buddyinfo	<a href="#">NodeExporterCollectorBuddyInfoConfig</a>	Defines the configuration of the <b>buddyinfo</b> collector, which collects statistics about memory fragmentation from the <b>node_buddyinfo_blocks</b> metric. This metric collects data from <b>/proc/buddyinfo</b> . Disabled by default.
mountstats	<a href="#">NodeExporterCollectorMountStatsConfig</a>	Defines the configuration of the <b>mountstats</b> collector, which collects statistics about NFS volume I/O activities. Disabled by default.
ksmd	<a href="#">NodeExporterCollectorKSMDConfig</a>	Defines the configuration of the <b>ksmd</b> collector, which collects statistics from the kernel same-page merger daemon. Disabled by default.
processes	<a href="#">NodeExporterCollectorProcessesConfig</a>	Defines the configuration of the <b>processes</b> collector, which collects statistics from processes and threads running in the system. Disabled by default.
systemd	<a href="#">NodeExporterCollectorSystemdConfig</a>	Defines the configuration of the <b>systemd</b> collector, which collects statistics on the systemd daemon and its managed services. Disabled by default.

## 8.11. NODEEXPORTERCOLLECTORCPUFREQCONFIG

### 8.11.1. 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 <b>cpufreq</b> collector.

## 8.12. NODEEXPORTERCOLLECTORKSMDCONFIG

### 8.12.1. 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 <b>ksmd</b> collector.

## 8.13. NODEEXPORTERCOLLECTORMOUNTSTATSCONFIG

### 8.13.1. 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 <b>mountstats</b> collector.

## 8.14. NODEEXPORTERCOLLECTORNETCLASSCONFIG

### 8.14.1. 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 <b>netclass</b> collector.
useNetlink	bool	A Boolean flag that activates the <b>netlink</b> implementation of the <b>netclass</b> collector. The default value is <b>true</b> , which activates the <b>netlink</b> mode. This implementation improves the performance of the <b>netclass</b> collector.

## 8.15. NODEEXPORTERCOLLECTORNETDEVCONFIG

### 8.15.1. 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
----------	------	-------------

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the <b>netdev</b> collector.

## 8.16. NODEEXPORTERCOLLECTORPROCESSESCONFIG

### 8.16.1. 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 <b>processes</b> collector.

## 8.17. NODEEXPORTERCOLLECTORSYSTEMDCONFIG

### 8.17.1. 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 <b>systemd</b> collector.

Property	Type	Description
units	[]string	A list of regular expression (regex) patterns that match systemd units to be included by the <b>systemd</b> collector. By default, the list is empty, so the collector exposes no metrics for systemd units.

## 8.18. NODEEXPORTERCOLLECTORTCPSTATCONFIG

### 8.18.1. 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 <b>tcpstat</b> collector.

## 8.19. NODEEXPORTERCONFIG

### 8.19.1. Description

The **NodeExporterConfig** resource defines settings for the **node-exporter** agent.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
collectors	<a href="#">NodeExporterCollectorConfig</a>	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 <b>0</b> , which means that node-exporter runs on all CPUs. If a kernel deadlock occurs or if performance degrades when reading from <b>sysfs</b> concurrently, you can change this value to <b>1</b> , 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 <b>maxProcs</b> 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 <b>netdev</b> and <b>netclass</b> . 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 <b>prometheus-k8s</b> deployment closely for excessive memory usage.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the <b>NodeExporter</b> container.

## 8.20. OPENSIFTSTATEMETRICSCONFIG

### 8.20.1. 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 <b>OpenShiftStateMetrics</b> container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

## 8.21. PROMETHEUSK8SCONFIG

### 8.21.1. Description

The **PrometheusK8sConfig** resource defines settings for the Prometheus component.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
additionalAlertmanagerConfigs	[] <a href="#">AdditionalAlertmanagerConfig</a>	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 <b>64MB</b> ), or the string <b>automatic</b> , 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	map[string]string	Defines labels to be added to any time series or alerts when communicating with external systems such as federation, remote storage, and Alertmanager. By default, no labels are added.
logLevel	string	Defines the log level setting for Prometheus. The possible values are: <b>error</b> , <b>warn</b> , <b>info</b> , and <b>debug</b> . The default value is <b>info</b> .
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 <b>emptyDir</b> volume at <b>/var/log/prometheus</b> , or a full path to a location where an <b>emptyDir</b> volume will be mounted and the queries saved. Writing to <b>/dev/stderr</b> , <b>/dev/stdout</b> or <b>/dev/null</b> is supported, but writing to any other <b>/dev/</b> path is not supported. Relative paths are also not supported. By default, PromQL queries are not logged.
remoteWrite	<a href="#">[]RemoteWriteSpec</a>	Defines the remote write configuration, including URL, authentication, and relabeling settings.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the <b>Prometheus</b> 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: <b>[0-9]+(ms s m h d w y)</b> (ms = milliseconds, s= seconds, m = minutes, h = hours, d = days, w = weeks, y = years). The default value is <b>15d</b> .
retentionSize	string	Defines the maximum amount of disk space used by data blocks plus the write-ahead log (WAL). Supported values are <b>B, KB, KiB, MB, MiB, GB, GiB, TB, TiB, PB, PiB, EB, and EiB</b> . 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 <b>full</b> or <b>minimal</b> . In the <b>full</b> profile (default), Prometheus collects all metrics that are exposed by the platform components. In the <b>minimal</b> 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.

## 8.22. PROMETHEUSOPERATORCONFIG

### 8.22.1. 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 <b>error</b> , <b>warn</b> , <b>info</b> , and <b>debug</b> . The default value is <b>info</b> .
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the <b>PrometheusOperator</b> container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

## 8.23. PROMETHEUSOPERATORADMISSIONWEBHOOKCONFIG

### 8.23.1. 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 <b>prometheus-operator-admission-webhook</b> container.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

## 8.24. PROMETHEUSRESTRICTEDCONFIG

### 8.24.1. 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 <b>ServiceMonitor</b> or <b>PodMonitor</b> 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 <b>30s</b> ), minutes (for example <b>1m</b> ) or a mix of minutes and seconds (for example <b>1m30s</b> ). The default value is <b>30s</b> .
evaluationInterval	string	Configures the default interval between rule evaluations in case the <b>PrometheusRule</b> 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 <b>30s</b> ), minutes (for example <b>1m</b> ) or a mix of minutes and seconds (for example <b>1m30s</b> ). It only applies to <b>PrometheusRule</b> resources with the <b>openshift.io/prometheus-rule-evaluation-scope="leaf-prometheus"</b> label. The default value is <b>30s</b> .
additionalAlertmanagerConfigs	<a href="#">[]AdditionalAlertmanagerConfig</a>	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 <b>0</b> , 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 <b>0</b> , 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 <b>0</b> , 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 <b>SampleLimit</b> value set in any user-defined <b>ServiceMonitor</b> or <b>PodMonitor</b> object if the value is greater than <b>enforcedTargetLimit</b> . Administrators can use this setting to keep the overall number of samples under control. The default value is <b>0</b> , which means that no limit is set.
enforcedTargetLimit	*uint64	Specifies a global limit on the number of scraped targets. This setting overrides the <b>TargetLimit</b> value set in any user-defined <b>ServiceMonitor</b> or <b>PodMonitor</b> object if the value is greater than <b>enforcedSampleLimit</b> . Administrators can use this setting to keep the overall number of targets under control. The default value is <b>0</b> .

Property	Type	Description
externalLabels	map[string]string	Defines labels to be added to any time series or alerts when communicating with external systems such as federation, remote storage, and Alertmanager. By default, no labels are added.
logLevel	string	Defines the log level setting for Prometheus. The possible values are <b>error</b> , <b>warn</b> , <b>info</b> , and <b>debug</b> . The default setting is <b>info</b> .
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 <b>emptyDir</b> volume at <b>/var/log/prometheus</b> , or a full path to a location where an <b>emptyDir</b> volume will be mounted and the queries saved. Writing to <b>/dev/stderr</b> , <b>/dev/stdout</b> or <b>/dev/null</b> is supported, but writing to any other <b>/dev/</b> path is not supported. Relative paths are also not supported. By default, PromQL queries are not logged.
remoteWrite	<a href="#">[]RemoteWriteSpec</a>	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: <b>[0-9]+(ms s m h d w y)</b> (ms = milliseconds, s= seconds, m = minutes, h = hours, d = days, w = weeks, y = years). The default value is <b>24h</b> .
retentionSize	string	Defines the maximum amount of disk space used by data blocks plus the write-ahead log (WAL). Supported values are <b>B, KB, KiB, MB, MiB, GB, GiB, TB, TiB, PB, PiB, EB, and EiB</b> . The default value is <b>nil</b> .
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.

## 8.25. REMOTEWritespec

### 8.25.1. Description

The **RemoteWriteSpec** resource defines the settings for remote write storage.

### 8.25.2. Required

- **url**

Appears in: [PrometheusK8sConfig](#), [PrometheusRestrictedConfig](#)

Property	Type	Description
authorization	*monv1.SafeAuthorization	Defines the authorization settings for remote write storage.
basicAuth	*monv1.BasicAuth	Defines Basic authentication settings for the remote write endpoint URL.

Property	Type	Description
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.
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.



Property	Type	Description
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.

## 8.26. TLSCONFIG

### 8.26.1. Description

The **TLSConfig** resource configures the settings for TLS connections.

### 8.26.2. 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.
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 <b>true</b> , disables the verification of the remote host's certificate and name.

## 8.27. TELEMETRICCLIENTCONFIG

### 8.27.1. Description

**TelemeterClientConfig** defines settings for the Telemeter Client component.

### 8.27.2. 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 <b>TelemeterClient</b> container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

## 8.28. THANOSQUERIERCONFIG

### 8.28.1. 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 <b>false</b> .
logLevel	string	Defines the log level setting for Thanos Querier. The possible values are <b>error</b> , <b>warn</b> , <b>info</b> , and <b>debug</b> . The default value is <b>info</b> .

Property	Type	Description
enableCORS	bool	A Boolean flag that enables setting CORS headers. The headers allow access from any origin. The default value is <b>false</b> .
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.

## 8.29. THANOSRULERCONFIG

### 8.29.1. Description

The **ThanosRulerConfig** resource defines configuration for the Thanos Ruler instance for user-defined projects.

Appears in: [UserWorkloadConfiguration](#)

Property	Type	Description
additionalAlertmanagerConfigs	[] <a href="#">AdditionalAlertmanagerConfig</a>	Configures how the Thanos Ruler component communicates with additional Alertmanager instances. The default value is <b>nil</b> .

Property	Type	Description
evaluationInterval	string	Configures the default interval between Prometheus rule evaluations in case the <b>PrometheusRule</b> 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 <b>30s</b> ), minutes (for example <b>1m</b> ) or a mix of minutes and seconds (for example <b>1m30s</b> ). It applies to <b>PrometheusRule</b> resources without the <b>openshift.io/prometheus-rule-evaluation-scope="leaf-prometheus"</b> label. The default value is <b>15s</b> .
logLevel	string	Defines the log level setting for Thanos Ruler. The possible values are <b>error</b> , <b>warn</b> , <b>info</b> , and <b>debug</b> . The default value is <b>info</b> .
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.
retention	string	Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: <b>[0-9]+(ms s m h d w y)</b> (ms = milliseconds, s = seconds, m = minutes, h = hours, d = days, w = weeks, y = years). The default value is <b>24h</b> .
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

Property	Type	Description
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Thanos Ruler. Use this setting to configure the storage class and size of a volume.

## 8.30. USERWORKLOADCONFIG

### 8.30.1. 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 <b>PrometheusRules</b> objects for which the <b>namespace</b> label is not enforced to the namespace of the object. Such objects should be created in a namespace configured under the <b>namespacesWithoutLabelEnforcement</b> property of the <b>UserWorkloadConfiguration</b> resource. The default value is <b>true</b> .

## 8.31. USERWORKLOADCONFIGURATION

### 8.31.1. 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	* <a href="#">AlertmanagerUserWorkloadConfig</a>	Defines the settings for the Alertmanager component in user workload monitoring.

Property	Type	Description
prometheus	<a href="#">*PrometheusRestrictedConfig</a>	Defines the settings for the Prometheus component in user workload monitoring.
prometheusOperator	<a href="#">*PrometheusOperatorConfig</a>	Defines the settings for the Prometheus Operator component in user workload monitoring.
thanosRuler	<a href="#">*ThanosRulerConfig</a>	Defines the settings for the Thanos Ruler component in user workload monitoring.
namespacesWithoutLabelEnforcement	[]string	<p>Defines the list of namespaces for which Prometheus and Thanos Ruler in user-defined monitoring do not enforce the <b>namespace</b> label value in <b>PrometheusRule</b> objects.</p> <p>The <b>namespacesWithoutLabelEnforcement</b> 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 <b>PrometheusRule</b> objects in each user project.</p> <p>To make the resulting alerts and metrics visible to project users, the query expressions should return a <b>namespace</b> label with a non-empty value.</p>

## CHAPTER 9. MONITORING CLUSTERS THAT RUN ON RHOSO

You can correlate observability metrics for clusters that run on Red Hat OpenStack Services on OpenShift (RHOSO). By collecting metrics from both environments, you can monitor and troubleshoot issues across the infrastructure and application layers.

There are two supported methods for metric correlation for clusters that run on RHOSO:

- [Remote writing](#) to an external Prometheus instance.
- Collecting data from the OpenShift Container Platform federation endpoint to the RHOSO observability stack.

### 9.1. REMOTE WRITING TO AN EXTERNAL PROMETHEUS INSTANCE

Use remote write with both Red Hat OpenStack Services on OpenShift (RHOSO) and OpenShift Container Platform to push their metrics to an external Prometheus instance.

#### Prerequisites

- You have access to an external Prometheus instance.
- You have administrative access to RHOSO and your cluster.
- You have certificates for secure communication with mTLS.
- Your Prometheus instance is configured for client TLS certificates and has been set up as a remote write receiver.
- The Cluster Observability Operator is installed on your RHOSO cluster.
- The monitoring stack for your RHOSO cluster is configured to collect the metrics that you are interested in.
- Telemetry is enabled in the RHOSO environment.



#### NOTE

To verify that the telemetry service is operating normally, entering the following command:

```
$ oc -n openstack get monitoringstacks metric-storage -o yaml
```

The **monitoringstacks** CRD indicates whether telemetry is enabled correctly.

#### Procedure

1. Configure your RHOSO management cluster to send metrics to Prometheus:
  - a. Create a secret that is named **mtls-bundle** in the **openstack** namespace that contains HTTPS client certificates for authentication to Prometheus by entering the following command:

```
$ oc --namespace openstack \
  create secret generic mtls-bundle \
```

```
--from-file=./ca.crt \
--from-file=osp-client.crt \
--from-file=osp-client.key
```

- b. Open the **controlplane** configuration for editing by running the following command:

```
$ oc -n openstack edit openstackcontrolplane/controlplane
```

- c. With the configuration open, replace the **.spec.telemetry.template.metricStorage** section so that RHOSO sends metrics to Prometheus. As an example:

```
metricStorage:
  customMonitoringStack:
    alertmanagerConfig:
      disabled: false
    logLevel: info
    prometheusConfig:
      scrapeInterval: 30s
      remoteWrite:
        - url: https://external-prometheus.example.com/api/v1/write 1
      tlsConfig:
        ca:
          secret:
            name: mtls-bundle
            key: ca.crt
        cert:
          secret:
            name: mtls-bundle
            key: ocp-client.crt
        keySecret:
          name: mtls-bundle
          key: ocp-client.key
      replicas: 2
    resourceSelector:
      matchLabels:
        service: metricStorage
    resources:
      limits:
        cpu: 500m
        memory: 512Mi
      requests:
        cpu: 100m
        memory: 256Mi
    retention: 1d 2
  dashboardsEnabled: false
  dataplaneNetwork: ctlplane
  enabled: true
  prometheusTls: {}
```

**1** Replace this URL with the URL of your Prometheus instance.

**2** Set a retention period. Optionally, you can reduce retention for local metrics because of external collection.



2. Configure the tenant cluster on which your workloads run to send metrics to Prometheus:

- a. Create a cluster monitoring config map as a YAML file. The map must include a remote write configuration and cluster identifiers. As an example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      retention: 1d ❶
      remoteWrite:
        - url: "https://external-prometheus.example.com/api/v1/write"
          writeRelabelConfigs:
            - sourceLabels:
                - __tmp_openshift_cluster_id__
              targetLabel: cluster_id
              action: replace
      tlsConfig:
        ca:
          secret:
            name: mtls-bundle
            key: ca.crt
        cert:
          secret:
            name: mtls-bundle
            key: ocp-client.crt
        keySecret:
          name: mtls-bundle
          key: ocp-client.key
```

- ❶ Set a retention period. Optionally, you can reduce retention for local metrics because of external collection.

- b. Save the config map as a file called **cluster-monitoring-config.yaml**.
- c. Create a secret that is named **mtls-bundle** in the **openshift-monitoring** namespace that contains HTTPS client certificates for authentication to Prometheus by entering the following command:

```
$ oc --namespace openshift-monitoring \
  create secret generic mtls-bundle \
    --from-file=./ca.crt \
    --from-file=ocp-client.crt \
    --from-file=ocp-client.key
```

- d. Apply the cluster monitoring configuration by running the following command:

```
$ oc apply -f cluster-monitoring-config.yaml
```

After the changes propagate, you can see aggregated metrics in your external Prometheus instance.

### Additional resources

- [Configuring remote write storage](#)
- [Adding cluster ID labels to metrics](#)

## 9.2. COLLECTING CLUSTER METRICS FROM THE FEDERATION ENDPOINT

You can employ the federation endpoint of your OpenShift Container Platform cluster to make metrics available to a Red Hat OpenStack Services on OpenShift (RHOSO) cluster to practice pull-based monitoring.

### Prerequisites

- You have administrative access to RHOSO and the tenant cluster that is running on it.
- Telemetry is enabled in the RHOSO environment.
- The Cluster Observability Operator is installed on your cluster.
- The monitoring stack for your cluster is configured.
- Your cluster has its federation endpoint exposed.

### Procedure

1. Connect to your cluster by using a username and password; do not log in by using a **kubeconfig** file that was generated by the installation program.
2. To retrieve a token from the OpenShift Container Platform cluster, run the following command on it:

```
$ oc whoami -t
```

3. Make the token available as a secret in the **openstack** namespace in the RHOSO management cluster by running the following command:

```
$ oc -n openstack create secret generic ocp-federated --from-literal=token=  
<the_token_fetched_previously>
```

4. To get the Prometheus federation route URL from your OpenShift Container Platform cluster, run the following command:

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

5. Write a manifest for a scrape configuration and save it as a file called **cluster-scrape-config.yaml**. As an example:

```
apiVersion: monitoring.rhobs/v1alpha1  
kind: ScrapeConfig  
metadata:  
  labels:
```

```

    service: metricStorage
    name: sos1-federated
    namespace: openstack
spec:
  params:
    'match[]':
      - '{__name__=~"kube_node_info|kube_persistentvolume_info|cluster:master_nodes"}' ❶
    metricsPath: '/federate'
  authorization:
    type: Bearer
    credentials:
      name: ocp-federated ❷
      key: token
  scheme: HTTPS # or HTTP
  scrapeInterval: 30s ❸
  staticConfigs:
    - targets:
        - prometheus-k8s-federate-openshift-monitoring.apps.openshift.example ❹

```

- ❶ Add metrics here. In this example, only the metrics **kube\_node\_info**, **kube\_persistentvolume\_info**, and **cluster:master\_nodes** are requested.
- ❷ Insert the previously generated secret name here.
- ❸ Limit scraping to fewer than 1000 samples for each request with a maximum frequency of once every 30 seconds.
- ❹ Insert the URL you fetched previously here. If the endpoint is HTTPS and uses a custom certificate authority, add a **tlsConfig** section after it.

6. While connected to the RHOSO management cluster, apply the manifest by running the following command:

```
$ oc apply -f cluster-scrape-config.yaml
```

After the config propagates, the cluster metrics are accessible for querying in the OpenShift Container Platform UI in RHOSO.

#### Additional resources

- [Querying metrics by using the federation endpoint for Prometheus](#)

## 9.3. AVAILABLE METRICS FOR CLUSTERS THAT RUN ON RHOSO

To query metrics and identifying resources across the stack, there are helper metrics that establish a correlation between Red Hat OpenStack Services on OpenShift (RHOSO) infrastructure resources and their representations in the tenant OpenShift Container Platform cluster.

To map nodes with RHOSO compute instances, in the metric **kube\_node\_info**:

- **node** is the Kubernetes node name.
- **provider\_id** contains the identifier of the corresponding compute service instance.

To map persistent volumes with RHOSO block storage or shared filesystems shares, in the metric **kube\_persistentvolume\_info**:

- **persistentvolume** is the volume name.
- **csi\_volume\_handle** is the block storage volume or share identifier.

By default, the compute machines that back the cluster control plane nodes are created in a server group with a soft anti-affinity policy. As a result, the compute service creates them on separate hypervisors on a best-effort basis. However, if the state of the RHOSO cluster is not appropriate for this distribution, the machines are created anyway.

In combination with the default soft anti-affinity policy, you can configure an alert that activates when a hypervisor hosts more than one control plane node of a given cluster to highlight the degraded level of high availability.

As an example, this PromQL query returns the number of OpenShift Container Platform master nodes per RHOSP host:

```
sum by (vm_instance) (
  group by (vm_instance, resource) (ceilometer_cpu)
  / on (resource) group_right(vm_instance) (
    group by (node, resource) (
      label_replace(kube_node_info, "resource", "$1", "system_uuid", "(.+)")
    )
  / on (node) group_left group by (node) (
    cluster:master_nodes
  )
)
```