



Red Hat OpenShift Service on AWS 4

Monitoring

Monitoring projects on Red Hat OpenShift Service on AWS

Red Hat OpenShift Service on AWS 4 Monitoring

Monitoring projects on Red Hat OpenShift Service on AWS

Legal Notice

Copyright © 2024 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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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

This document provides information about monitoring projects on Red Hat OpenShift Service on AWS (ROSA).

Table of Contents

CHAPTER 1. MONITORING OVERVIEW	6
1.1. ABOUT RED HAT OPENSIFT SERVICE ON AWS MONITORING	6
1.2. UNDERSTANDING THE MONITORING STACK	6
1.2.1. Default monitoring targets	7
1.2.2. Components for monitoring user-defined projects	8
1.2.3. Monitoring targets for user-defined projects	8
1.3. GLOSSARY OF COMMON TERMS FOR RED HAT OPENSIFT SERVICE ON AWS MONITORING	9
1.4. NEXT STEPS	11
CHAPTER 2. ACCESSING MONITORING FOR USER-DEFINED PROJECTS	12
2.1. NEXT STEPS	12
CHAPTER 3. CONFIGURING THE MONITORING STACK	13
3.1. MAINTENANCE AND SUPPORT FOR MONITORING	13
3.1.1. Support considerations for monitoring	13
3.2. CONFIGURING THE MONITORING STACK	13
3.3. CONFIGURABLE MONITORING COMPONENTS	15
3.4. USING NODE SELECTORS TO MOVE MONITORING COMPONENTS	16
3.4.1. How node selectors work with other constraints	16
3.4.2. Moving monitoring components to different nodes	16
3.5. ASSIGNING TOLERATIONS TO MONITORING COMPONENTS	18
3.6. MANAGING CPU AND MEMORY RESOURCES FOR MONITORING COMPONENTS	19
3.6.1. About specifying limits and requests for monitoring components	19
3.6.2. Specifying limits and requests for monitoring components	20
3.7. CONFIGURING PERSISTENT STORAGE	22
3.7.1. Persistent storage prerequisites	23
3.7.2. Configuring a persistent volume claim	23
3.7.3. Modifying the retention time and size for Prometheus metrics data	25
3.7.4. Modifying the retention time for Thanos Ruler metrics data	27
3.8. CONFIGURING REMOTE WRITE STORAGE	28
3.8.1. Supported remote write authentication settings	30
3.8.2. Example remote write authentication settings	31
3.9. ADDING CLUSTER ID LABELS TO METRICS	37
3.9.1. Creating cluster ID labels for metrics	37
3.10. CONTROLLING THE IMPACT OF UNBOUND METRICS ATTRIBUTES IN USER-DEFINED PROJECTS	39
3.10.1. Setting scrape sample and label limits for user-defined projects	40
CHAPTER 4. CONFIGURING EXTERNAL ALERTMANAGER INSTANCES	42
CHAPTER 5. CONFIGURING SECRETS FOR ALERTMANAGER	44
5.1. ADDING A SECRET TO THE ALERTMANAGER CONFIGURATION	44
5.2. ATTACHING ADDITIONAL LABELS TO YOUR TIME SERIES AND ALERTS	45
CHAPTER 6. CONFIGURING POD TOPOLOGY SPREAD CONSTRAINTS FOR MONITORING	48
6.1. SETTING UP POD TOPOLOGY SPREAD CONSTRAINTS FOR THANOS RULER	48
6.2. SETTING LOG LEVELS FOR MONITORING COMPONENTS	49
6.3. ENABLING THE QUERY LOG FILE FOR PROMETHEUS	51
CHAPTER 7. DISABLING MONITORING FOR USER-DEFINED PROJECTS	53
7.1. DISABLING MONITORING FOR USER-DEFINED PROJECTS	53
7.2. EXCLUDING A USER-DEFINED PROJECT FROM MONITORING	53
CHAPTER 8. ENABLING ALERT ROUTING FOR USER-DEFINED PROJECTS	54

8.1. UNDERSTANDING ALERT ROUTING FOR USER-DEFINED PROJECTS	54
8.2. ENABLING A SEPARATE ALERTMANAGER INSTANCE FOR USER-DEFINED ALERT ROUTING	54
8.3. GRANTING USERS PERMISSION TO CONFIGURE ALERT ROUTING FOR USER-DEFINED PROJECTS	55
8.4. NEXT STEPS	56
CHAPTER 9. MANAGING METRICS	57
9.1. UNDERSTANDING METRICS	57
9.2. SETTING UP METRICS COLLECTION FOR USER-DEFINED PROJECTS	57
9.2.1. Deploying a sample service	57
9.2.2. Specifying how a service is monitored	59
9.3. QUERYING METRICS	60
9.3.1. Querying metrics for all projects as a cluster administrator	60
9.3.2. Querying metrics for user-defined projects as a developer	62
9.4. GETTING DETAILED INFORMATION ABOUT A METRICS TARGET	63
CHAPTER 10. MANAGING ALERTS	66
10.1. ACCESSING THE ALERTING UI IN THE ADMINISTRATOR AND DEVELOPER PERSPECTIVES	66
10.2. SEARCHING AND FILTERING ALERTS, SILENCES, AND ALERTING RULES	67
Understanding alert filters	67
Understanding silence filters	67
Understanding alerting rule filters	68
Searching and filtering alerts, silences, and alerting rules in the Developer perspective	69
10.3. GETTING INFORMATION ABOUT ALERTS, SILENCES, AND ALERTING RULES	69
10.4. MANAGING SILENCES	71
10.4.1. Silencing alerts	71
10.4.2. Editing silences	73
10.4.3. Expiring silences	74
10.5. MANAGING ALERTING RULES FOR USER-DEFINED PROJECTS	74
10.5.1. Optimizing alerting for user-defined projects	75
10.5.2. About creating alerting rules for user-defined projects	75
10.5.3. Creating alerting rules for user-defined projects	76
10.5.4. Accessing alerting rules for user-defined projects	77
10.5.5. Listing alerting rules for all projects in a single view	77
10.5.6. Removing alerting rules for user-defined projects	78
10.6. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS	78
10.6.1. Creating alert routing for user-defined projects	79
10.7. APPLYING A CUSTOM CONFIGURATION TO ALERTMANAGER FOR USER-DEFINED ALERT ROUTING	80
CHAPTER 11. REVIEWING MONITORING DASHBOARDS	82
11.1. REVIEWING MONITORING DASHBOARDS AS A CLUSTER ADMINISTRATOR	83
11.2. REVIEWING MONITORING DASHBOARDS AS A DEVELOPER	84
11.3. NEXT STEPS	84
CHAPTER 12. ACCESSING MONITORING APIS BY USING THE CLI	85
12.1. ABOUT ACCESSING MONITORING WEB SERVICE APIS	85
12.2. ACCESSING A MONITORING WEB SERVICE API	86
12.3. QUERYING METRICS BY USING THE FEDERATION ENDPOINT FOR PROMETHEUS	86
12.4. ACCESSING METRICS FROM OUTSIDE THE CLUSTER FOR CUSTOM APPLICATIONS	88
12.5. ADDITIONAL RESOURCES	89
CHAPTER 13. TROUBLESHOOTING MONITORING ISSUES	90
13.1. DETERMINING WHY USER-DEFINED PROJECT METRICS ARE UNAVAILABLE	90
13.2. DETERMINING WHY PROMETHEUS IS CONSUMING A LOT OF DISK SPACE	92

CHAPTER 14. CONFIG MAP REFERENCE FOR THE CLUSTER MONITORING OPERATOR	94
14.1. CLUSTER MONITORING OPERATOR CONFIGURATION REFERENCE	94
14.2. ADDITIONALALERTMANAGERCONFIG	94
14.2.1. Description	94
14.2.2. Required	94
14.3. ALERTMANAGERMAINCONFIG	95
14.3.1. Description	95
14.4. ALERTMANAGERUSERWORKLOADCONFIG	96
14.4.1. Description	96
14.5. CLUSTERMONITORINGCONFIGURATION	98
14.5.1. Description	98
14.6. DEDICATEDSERVICEMONITORS	99
14.6.1. Description	99
14.7. K8SPROMETHEUSADAPTER	100
14.7.1. Description	100
14.8. KUBESTATEMETRICSCONFIG	101
14.8.1. Description	101
14.9. METRICSSERVERCONFIG	101
14.9.1. Description	101
14.10. PROMETHEUSOPERATORADMISSIONWEBHOOKCONFIG	102
14.10.1. Description	102
14.11. MONITORINGPLUGINCONFIG	102
14.11.1. Description	102
14.12. NODEEXPORTERCOLLECTORBUDDYINFOCONFIG	103
14.12.1. Description	103
14.13. NODEEXPORTERCOLLECTORCONFIG	103
14.13.1. Description	103
14.14. NODEEXPORTERCOLLECTORCPUFREQCONFIG	104
14.14.1. Description	104
14.15. NODEEXPORTERCOLLECTORKSMDCONFIG	105
14.15.1. Description	105
14.16. NODEEXPORTERCOLLECTORMOUNTSTATSCONFIG	105
14.16.1. Description	105
14.17. NODEEXPORTERCOLLECTORNETCLASSCONFIG	105
14.17.1. Description	105
14.18. NODEEXPORTERCOLLECTORNETDEVCONFIG	106
14.18.1. Description	106
14.19. NODEEXPORTERCOLLECTORPROCESSESCONFIG	106
14.19.1. Description	107
14.20. NODEEXPORTERCOLLECTORSYSTEMDCONFIG	107
14.20.1. Description	107
14.21. NODEEXPORTERCOLLECTORTCPSTATCONFIG	108
14.21.1. Description	108
14.22. NODEEXPORTERCONFIG	108
14.22.1. Description	108
14.23. OPENSIFTSTATEMETRICSCONFIG	109
14.23.1. Description	109
14.24. PROMETHEUSK8SCONFIG	109
14.24.1. Description	109
14.25. PROMETHEUSOPERATORCONFIG	112
14.25.1. Description	112
14.26. PROMETHEUSRRESTRICTEDCONFIG	113
14.26.1. Description	113

14.27. REMOTEWITESPEC	116
14.27.1. Description	116
14.27.2. Required	116
14.28. TLSCONFIG	117
14.28.1. Description	117
14.28.2. Required	117
14.29. TELEMETRCLIENTCONFIG	118
14.29.1. Description	118
14.29.2. Required	118
14.30. THANOSQUERIERCONFIG	119
14.30.1. Description	119
14.31. THANOSRULERCONFIG	119
14.31.1. Description	119
14.32. USERWORKLOADCONFIGURATION	120
14.32.1. Description	120

CHAPTER 1. MONITORING OVERVIEW

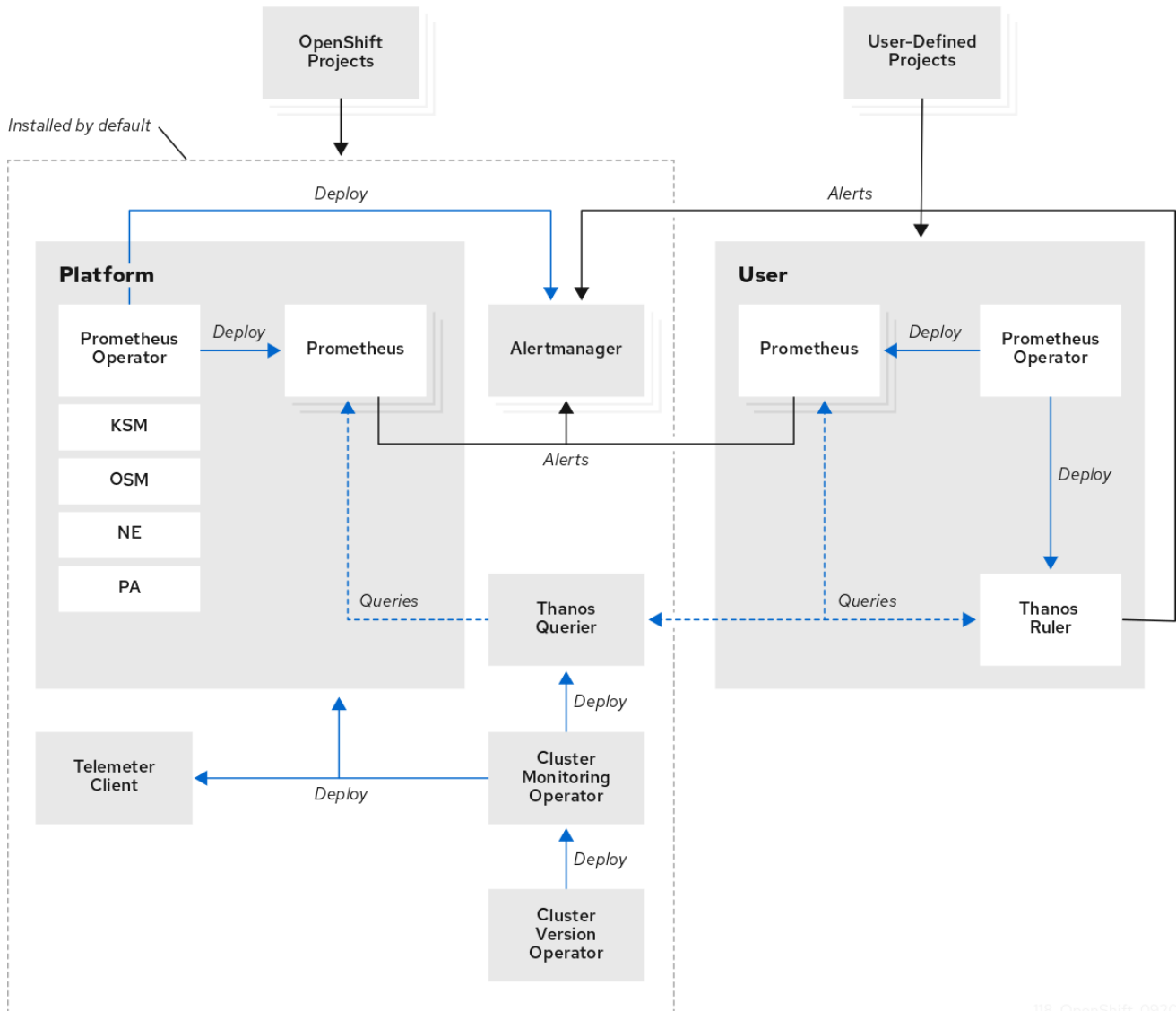
1.1. ABOUT RED HAT OPENSIFT SERVICE ON AWS MONITORING

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

1.2. UNDERSTANDING THE MONITORING STACK

The Red Hat OpenShift Service on AWS (ROSA) monitoring stack is based on the [Prometheus](#) open source project and its wider ecosystem. The monitoring stack includes the following:

- **Default platform monitoring components.** A set of platform monitoring components are installed in the **openshift-monitoring** project by default during a Red Hat OpenShift Service on AWS installation. Red Hat Site Reliability Engineers (SRE) use these components to monitor core cluster components including Kubernetes services. This includes critical metrics, such as CPU and memory, collected from all of the workloads in every namespace. These components are illustrated in the **Installed by default** section in the following diagram.
- **Components for monitoring user-defined projects** A set of user-defined project monitoring components are installed in the **openshift-user-workload-monitoring** project by default during a Red Hat OpenShift Service on AWS installation. You can use these components to monitor services and pods in user-defined projects. These components are illustrated in the **User** section in the following diagram.



118_OpenShift_092C

1.2.1. Default monitoring targets

Red Hat Site Reliability Engineers (SRE) monitor the following platform targets in your Red Hat OpenShift Service on AWS cluster:

- CoreDNS
- Elasticsearch (if Logging is installed)
- etcd
- Fluentd (if Logging is installed)
- HAProxy
- Image registry
- Kubelets
- Kubernetes API server
- Kubernetes controller manager

- Kubernetes scheduler
- OpenShift API server
- OpenShift Controller Manager
- Operator Lifecycle Manager (OLM)
- Vector (if Logging is installed)

1.2.2. Components for monitoring user-defined projects

Red Hat OpenShift Service on AWS 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 1.1. Components for monitoring user-defined projects

Component	Description
Prometheus Operator	The Prometheus Operator (PO) in the openshift-user-workload-monitoring project creates, configures, and manages Prometheus and Thanos Ruler instances in the same project.
Prometheus	Prometheus is the monitoring system 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 Red Hat OpenShift Service on AWS, 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.

All of these components are monitored by the stack and are automatically updated when Red Hat OpenShift Service on AWS is updated.

1.2.3. Monitoring targets for user-defined projects

Monitoring is enabled by default for Red Hat OpenShift Service on AWS user-defined projects. You can monitor:

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

1.3. GLOSSARY OF COMMON TERMS FOR RED HAT OPENSIFT SERVICE ON AWS MONITORING

This glossary defines common terms that are used in Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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)

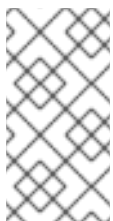
A CR is an extension of the Kubernetes API. You can create custom resources.

etcd

etcd is the key-value store for Red Hat OpenShift Service on AWS, which stores the state of all resource objects.

Fluentd

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



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.

node

A worker machine in the Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS cluster. An Operator takes human operational knowledge and encodes it into software that is packaged and shared with customers.

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 Red Hat OpenShift Service on AWS 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 adapter

The Prometheus Adapter translates Kubernetes node and pod queries for use in Prometheus. The resource metrics that are translated include CPU and memory utilization. The Prometheus Adapter exposes the cluster resource metrics API for horizontal pod autoscaling.

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

Red Hat OpenShift Service on AWS supports many types of storage on AWS. You can manage container storage for persistent and non-persistent data in an Red Hat OpenShift Service on AWS cluster.

Thanos Ruler

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

Vector

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

web console

A user interface (UI) to manage Red Hat OpenShift Service on AWS.

1.4. NEXT STEPS

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

CHAPTER 2. ACCESSING MONITORING FOR USER-DEFINED PROJECTS

When you install a Red Hat OpenShift Service on AWS (ROSA) cluster, monitoring for user-defined projects is enabled by default. With monitoring for user-defined projects enabled, you can monitor your own ROSA projects without the need for an additional monitoring solution.

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



NOTE

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

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

2.1. NEXT STEPS

- [Configuring the monitoring stack](#)

CHAPTER 3. CONFIGURING THE MONITORING STACK

This section explains what configuration is supported, shows how to configure the monitoring stack for user-defined projects, and demonstrates several common configuration scenarios.



IMPORTANT

Not all configuration parameters for the monitoring stack are exposed. Only the parameters and fields listed in the [Config map reference for the Cluster Monitoring Operator](#) are supported for configuration.

3.1. MAINTENANCE AND SUPPORT FOR MONITORING

Not all configuration options for the monitoring stack are exposed. The only supported way of configuring Red Hat OpenShift Service on AWS monitoring is by configuring the Cluster Monitoring Operator 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 Cluster Monitoring Operator automatically reconciles any differences and resets any unsupported changes back to the originally defined state by default and by design.



IMPORTANT

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

3.1.1. Support considerations for monitoring



NOTE

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

The following modifications are explicitly not supported:

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

3.2. CONFIGURING THE MONITORING STACK

In Red Hat OpenShift Service on AWS, you can configure the stack that monitors workloads for user-defined projects by using the **user-workload-monitoring-config ConfigMap** object. Config maps configure the Cluster Monitoring Operator (CMO), which in turn configures the components of the stack.

Prerequisites

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

Procedure

1. Edit the **ConfigMap** object.
 - 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 your configuration under **data/config.yaml** as a key-value pair **<component_name>: <component_configuration>**:

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

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

The following example **ConfigMap** object configures a data retention period and minimum container resource requests for Prometheus. This relates to the Prometheus instance that monitors user-defined projects only:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus: 1
      retention: 24h 2
      resources:
        requests:
          cpu: 200m 3
          memory: 2Gi 4
```

- 1 Defines the Prometheus component and the subsequent lines define its configuration.
- 2 Configures a twenty-four hour data retention period for the Prometheus instance that monitors user-defined projects.

- 3 Defines a minimum resource request of 200 millicores for the Prometheus container.
 - 4 Defines a minimum pod resource request of 2 GiB of memory for the Prometheus container.
2. Save the file to apply the changes to the **ConfigMap** object. The pods affected by the new configuration are restarted automatically.

**WARNING**

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

Additional resources

- Configuration reference for the [user-workload-monitoring-config](#) config map

3.3. 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 ConfigMap** objects.

**WARNING**

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

Table 3.1. Configurable monitoring components

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

3.4. 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 segregate workloads based on specific requirements or policies.

3.4.1. 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, Thanos Querier, 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.

Additional resources

- [Configuring pod topology spread constraints for monitoring](#)
- [Kubernetes documentation about node selectors](#)

3.4.2. Moving monitoring components to different nodes

You can move any of the components that monitor workloads for user-defined projects to specific worker nodes. It is not permitted to move components to control plane or infrastructure nodes.

Prerequisites

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

Procedure

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

```
$ oc label nodes <node-name> <node-label>
```

2. Edit the **ConfigMap** object:

- 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. 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.

3. Save the file to apply the changes. The components specified in the new configuration are moved to the new nodes automatically.



WARNING

When you save changes to a monitoring config map, the pods and other resources in the project might be redeployed. The running monitoring processes in that project might also restart.

Additional resources

- See the [Kubernetes documentation](#) for details on the **nodeSelector** constraint

3.5. ASSIGNING TOLERATIONS TO MONITORING COMPONENTS

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

Prerequisites

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

Procedure

1. Edit the **ConfigMap** object:
 - 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. 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"

```

2. Save the file to apply the changes. The new component placement configuration is applied automatically.



WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

Additional resources

- See the [Kubernetes documentation](#) on taints and tolerations

3.6. 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 and for the components that monitor user-defined projects in the **openshift-user-workload-monitoring** namespace.

3.6.1. About specifying limits and requests for monitoring components

You can configure resource limits and request settings for core platform monitoring components and for the components that monitor user-defined projects, including the following components:

- Alertmanager (for core platform monitoring and for user-defined projects)
- kube-state-metrics
- monitoring-plugin
- node-exporter
- openshift-state-metrics
- Prometheus (for core platform monitoring and for user-defined projects)
- Prometheus Adapter
- Prometheus Operator and its admission webhook service

- Telemeter Client
- Thanos Querier
- Thanos Ruler

By defining 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 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.

3.6.2. Specifying limits and requests for monitoring components

To configure CPU and memory resources, specify values for resource limits and requests in the appropriate **ConfigMap** object for the namespace in which the monitoring component is located:

- The **cluster-monitoring-config** config map in the **openshift-monitoring** namespace for core platform monitoring
- The **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** namespace for components that monitor user-defined projects

Prerequisites

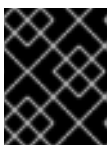
- **If you are configuring core platform monitoring components**
 - You have access to the cluster as a user with the **cluster-admin** cluster role.
 - You have created a **ConfigMap** object named **cluster-monitoring-config**.
- **If you are configuring components that monitor user-defined projects**
 - 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. To configure core platform monitoring components, edit the **cluster-monitoring-config** config map object in the **openshift-monitoring** namespace:

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

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



IMPORTANT

Make sure 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


```
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
    prometheusOperator:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    k8sPrometheusAdapter:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    kubeStateMetrics:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    telemeterClient:
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 200m
          memory: 500Mi
    openshiftStateMetrics:
```

```

resources:
  limits:
    cpu: 500m
    memory: 1Gi
  requests:
    cpu: 200m
    memory: 500Mi
thanosQuerier:
  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 automatically.



IMPORTANT

When you save changes to the **cluster-monitoring-config** config map, the pods and other resources in the **openshift-monitoring** project might be redeployed. The running monitoring processes in that project might also restart.

Additional resources

- [Kubernetes requests and limits documentation](#)

3.7. CONFIGURING PERSISTENT STORAGE

Running cluster monitoring with persistent storage means that your metrics are stored to a persistent

volume (PV) and can survive a pod being restarted or recreated. This is ideal if you require your metrics or alerting data to be guarded from data loss. For production environments, it is highly recommended to configure persistent storage. Because of the high IO demands, it is advantageous to use local storage.

3.7.1. Persistent storage prerequisites

- Use the block type of storage.

3.7.2. Configuring a persistent volume claim

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

Prerequisites

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

Procedure

1. Edit the **ConfigMap** object:
 - 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 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>:
      volumeClaimTemplate:
        spec:
          storageClassName: <storage_class>
          resources:
            requests:
              storage: <amount_of_storage>
```

See the [Kubernetes documentation on PersistentVolumeClaims](#) for information on how to specify **volumeClaimTemplate**.

The following example configures a PVC that claims persistent storage for the Prometheus instance that monitors user-defined projects:

■

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      volumeClaimTemplate:
        spec:
          storageClassName: gp3
          resources:
            requests:
              storage: 40Gi

```

The above example uses the **gp3** storage class.

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

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      volumeClaimTemplate:
        spec:
          storageClassName: gp3
          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.

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



WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

3.7.3. Modifying the retention time and size for Prometheus metrics data

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 that monitors user-defined projects, 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.

Prerequisites

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

- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **ConfigMap** object:

- 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 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), or **EB** (exabytes).

The following example sets the retention time to 24 hours and the retention size to 10 gigabytes for the Prometheus instance that monitors user-defined projects:

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

2. Save the file to apply the changes. The pods affected by the new configuration restart automatically.

**WARNING**

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

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

Procedure

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

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

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

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

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

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

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

```

metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      retention: 10d

```

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



WARNING

Saving changes to a monitoring config map might restart monitoring processes and redeploy the pods and other resources in the related project. The running monitoring processes in that project might also restart.

Additional resources

- [Understanding persistent storage](#)

3.8. CONFIGURING REMOTE WRITE STORAGE

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

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).
- You have set up a remote write compatible endpoint (such as Thanos) and know the endpoint URL. See the [Prometheus remote endpoints and storage documentation](#) for information about endpoints that are compatible with the remote write feature.



IMPORTANT

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

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

**WARNING**

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

Procedure

1. Edit the **ConfigMap** object:
 - 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 a **remoteWrite:** section under **data/config.yaml/prometheus**.
 - c. Add an endpoint URL and authentication credentials in this section:

```
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 an **Authorization** request header, basic authentication, OAuth 2.0, and TLS client. See *Supported remote write authentication settings* below for sample configurations of supported authentication methods.

- d. 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>
  <your_write_relabel_configs> 1
```

- 1** The write relabel configuration settings.

For **<your_write_relabel_configs>** substitute a list of write relabel configurations for metrics that you want to send to the remote endpoint.

The following sample shows how to forward 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
```

See the [Prometheus relabel_config documentation](#) for information about write relabel configuration options.

- Save the file to apply the changes. The pods affected by the new configuration restart automatically.



WARNING

Saving changes to a monitoring **ConfigMap** object might redeploy the pods and other resources in the related project. Saving changes might also restart the running monitoring processes in that project.

3.8.1. 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
-----------------------	------------------	-------------

Authentication method	Config map field	Description
AWS Signature Version 4	sigv4	This method uses AWS Signature Version 4 authentication to sign requests. You cannot use this method simultaneously with authorization, OAuth 2.0, or Basic authentication.
Basic authentication	basicAuth	Basic authentication sets the authorization header on every remote write request with the configured username and password.
authorization	authorization	Authorization sets the Authorization header on every remote write request using the configured token.
OAuth 2.0	oauth2	An OAuth 2.0 configuration uses the client credentials grant type. Prometheus fetches an access token from tokenUrl with the specified client ID and client secret to access the remote write endpoint. You cannot use this method simultaneously with authorization, AWS Signature Version 4, or Basic authentication.
TLS client	tlsConfig	A TLS client configuration specifies the CA certificate, the client certificate, and the client key file information used to authenticate with the remote write endpoint server using TLS. The sample configuration assumes that you have already created a CA certificate file, a client certificate file, and a client key file.

3.8.2. 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 user-defined projects in the **openshift-user-workload-monitoring** namespace.

Example 3.1. Sample YAML for AWS Signature Version 4 authentication

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

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

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

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
        - url: "https://authorization.example.com/api/write"
          sigv4:
            region: <AWS_region> 1
            accessKey:
              name: sigv4-credentials 2
              key: accessKey 3
            secretKey:
              name: sigv4-credentials 4
              key: secretKey 5
            profile: <AWS_profile_name> 6
            roleArn: <AWS_role_arn> 7
```

- 1 The AWS region.
- 2 4 The name of the **Secret** object containing the AWS API access credentials.
- 3 The key that contains the AWS API access key in the specified **Secret** object.
- 5 The key that contains the AWS API secret key in the specified **Secret** object.
- 6 The name of the AWS profile that is being used to authenticate.

- 7 The unique identifier for the Amazon Resource Name (ARN) assigned to your role.

Example 3.2. Sample YAML for basic authentication

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

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

- 1 The username.
- 2 The password.

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
        - url: "https://basicauth.example.com/api/write"
          basicAuth:
            username:
              name: rw-basic-auth 1
              key: user 2
            password:
              name: rw-basic-auth 3
              key: password 4
```

- 1 3 The name of the **Secret** object that contains the authentication credentials.
- 2 The key that contains the username in the specified **Secret** object.
- 4 The key that contains the password in the specified **Secret** object.

Example 3.3. Sample YAML for authentication with a bearer token using a **Secret** Object

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

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

- 1 The authentication token.

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
  prometheus:
    remoteWrite:
      - url: "https://authorization.example.com/api/write"
      authorization:
        type: Bearer 1
        credentials:
          name: rw-bearer-auth 2
          key: token 3
```

- 1 The authentication type of the request. The default value is **Bearer**.
- 2 The name of the **Secret** object that contains the authentication credentials.
- 3 The key that contains the authentication token in the specified **Secret** object.

Example 3.4. Sample YAML for OAuth 2.0 authentication

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

```
apiVersion: v1
kind: Secret
metadata:
  name: oauth2-credentials
```

```

namespace: openshift-user-workload-monitoring
stringData:
  id: <oauth2_id> 1
  secret: <oauth2_secret> 2
  token: <oauth2_authentication_token> 3
type: Opaque

```

- 1 The OAuth 2.0 ID.
- 2 The OAuth 2.0 secret.
- 3 The OAuth 2.0 token.

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

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
        - url: "https://test.example.com/api/write"
          oauth2:
            clientId:
              secret:
                name: oauth2-credentials 1
                key: id 2
            clientSecret:
              name: oauth2-credentials 3
              key: secret 4
            tokenUrl: https://example.com/oauth2/token 5
            scopes: 6
              - <scope_1>
              - <scope_2>
            endpointParams: 7
              param1: <parameter_1>
              param2: <parameter_2>

```

- 1 3 The name of the corresponding **Secret** object. Note that **ClientId** can alternatively refer to a **ConfigMap** object, although **clientSecret** must refer to a **Secret** object.
- 2 4 The key that contains the OAuth 2.0 credentials in the specified **Secret** object.
- 5 The URL used to fetch a token with the specified **clientId** and **clientSecret**.
- 6 The OAuth 2.0 scopes for the authorization request. These scopes limit what data the tokens can access.
- 7 The OAuth 2.0 authorization request parameters required for the authorization server.

Example 3.5. Sample YAML for TLS client authentication

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

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

- 1** The CA certificate in the Prometheus container with which to validate the server certificate.
- 2** The client certificate for authentication with the server.
- 3** The client key.

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      remoteWrite:
        - url: "https://remote-write-endpoint.example.com"
          tlsConfig:
            ca:
              secret:
                name: mtls-bundle 1
                key: ca.crt 2
            cert:
              secret:
                name: mtls-bundle 3
                key: client.crt 4
            keySecret:
              name: mtls-bundle 5
              key: client.key 6
```

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

- 2 The key in the specified **Secret** object that contains the CA certificate for the endpoint.
- 4 The key in the specified **Secret** object that contains the client certificate for the endpoint.
- 6 The key in the specified **Secret** object that contains the client key secret.

Additional resources

- See [Setting up remote write compatible endpoints](#) for steps to create a remote write compatible endpoint (such as Thanos).
- See [Tuning remote write settings](#) for information about how to optimize remote write settings for different use cases.

3.9. ADDING CLUSTER ID LABELS TO METRICS

If you manage multiple Red Hat OpenShift Service on AWS 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.

3.9.1. Creating cluster ID labels for metrics

You can create cluster ID labels for metrics by editing the settings in the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** namespace.

Prerequisites

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

Procedure

1. Edit the **ConfigMap** object:

- 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. 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** in user-workload monitoring:

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

- 1 The system initially applies a temporary cluster ID source label named **__tmp_openshift_cluster_id__**. This temporary label gets replaced by the cluster ID label name that you specify.
- 2 Specify the name of the cluster ID label for metrics sent to remote write storage. If you use a label name that already exists for a metric, that value is overwritten with the name of this cluster ID label. For the label name, do not use

__**tmp_openshift_cluster_id**__. The final relabeling step removes labels that use this name.

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

2. Save the file to apply the changes to the **ConfigMap** object. The pods affected by the updated configuration automatically restart.



WARNING

Saving changes to a monitoring **ConfigMap** object might redeploy the pods and other resources in the related project. Saving changes might also restart the running monitoring processes in that project.

Additional resources

- For details about write relabel configuration, see [Configuring remote write storage](#).

3.10. CONTROLLING THE IMPACT OF UNBOUND METRICS ATTRIBUTES IN USER-DEFINED PROJECTS

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

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

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

- Limit the number of samples that can be accepted per target scrape in user-defined projects
- Limit the number of scraped labels, the length of label names, and the length of label values
- 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.

3.10.1. Setting scrape sample and label limits for user-defined projects

You can limit the number of samples that can be accepted per target scrape in user-defined projects. You can also limit the number of scraped labels, the length of label names, and the length of label values.



WARNING

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

Prerequisites

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

Procedure

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

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

2. Add the **enforcedSampleLimit** configuration to **data/config.yaml** to limit the number of samples that can be accepted per target scrape in user-defined projects:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      enforcedSampleLimit: 50000 1
```

- 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.

3. Add the **enforcedLabelLimit**, **enforcedLabelNameLengthLimit**, and **enforcedLabelValueLengthLimit** configurations to **data/config.yaml** to limit the number of scraped labels, the length of label names, and the length of label values in user-defined projects:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      enforcedLabelLimit: 500 1
      enforcedLabelNameLengthLimit: 50 2
      enforcedLabelValueLengthLimit: 600 3
```

- 1** Specifies the maximum number of labels per scrape. The default value is **0**, which specifies no limit.
- 2** Specifies the maximum length in characters of a label name. The default value is **0**, which specifies no limit.
- 3** Specifies the maximum length in characters of a label value. The default value is **0**, which specifies no limit.

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



WARNING

When changes are saved to the **user-workload-monitoring-config ConfigMap** object, the pods and other resources in the **openshift-user-workload-monitoring** project might be redeployed. The running monitoring processes in that project might also be restarted.

CHAPTER 4. CONFIGURING EXTERNAL ALERTMANAGER INSTANCES

The Red Hat OpenShift Service on AWS monitoring stack includes a local Alertmanager instance that routes alerts from Prometheus. You can add external Alertmanager instances to route alerts for user-defined projects.

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

Prerequisites

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

Procedure

1. Edit the **ConfigMap** object.
 - a. 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
```

- b. Add a **<component>/additionalAlertmanagerConfigs:** section under **data/config.yaml/**.
- c. Add the configuration details for additional Alertmanagers in this section:

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

For **<component>**, substitute one of two supported external Alertmanager components: **prometheus** or **thanosRuler**.

For **<alertmanager_specification>**, substitute 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 using Thanos Ruler with a bearer token and 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
```

2. Save the file to apply the changes to the **ConfigMap** object. The new component placement configuration is applied automatically.
3. Save the file to apply the changes to the **ConfigMap** object. The new component placement configuration is applied automatically.

CHAPTER 5. CONFIGURING SECRETS FOR ALERTMANAGER

The Red Hat OpenShift Service on AWS 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.

5.1. ADDING A SECRET TO THE ALERTMANAGER CONFIGURATION

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

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

Prerequisites

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

Procedure

1. Edit the **ConfigMap** object.
 - a. 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
```

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

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



```
secrets: 1
- <secret_name_1> 2
- <secret_name_2>
```

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

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

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

2. Save the file to apply the changes to the **ConfigMap** object. The new configuration is applied automatically.

5.2. ATTACHING ADDITIONAL LABELS TO YOUR TIME SERIES AND ALERTS

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

Prerequisites

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

Procedure

1. Edit the **ConfigMap** object:
 - 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. Define a map of 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 a map of 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 related to user-defined projects, 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
```

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

**WARNING**

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

CHAPTER 6. CONFIGURING POD TOPOLOGY SPREAD CONSTRAINTS FOR MONITORING

You can use pod topology spread constraints to control how Thanos Ruler pods are spread across a network topology when Red Hat OpenShift Service on AWS 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.

Additional resources

- [Kubernetes Pod Topology Spread Constraints documentation](#)

6.1. SETTING UP POD TOPOLOGY SPREAD CONSTRAINTS FOR THANOS RULER

For user-defined monitoring, you can set up pod topology spread constraints for Thanos Ruler to fine tune how pod replicas are scheduled to nodes across zones. Doing so helps ensure that Thanos Ruler pods are highly available and run more efficiently, because workloads are spread across nodes in different data centers or hierarchical infrastructure zones.

You configure pod topology spread constraints for Thanos Ruler in the **user-workload-monitoring-config** config map.

Prerequisites

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

Procedure

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

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

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

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

```

thanosRuler:
  topologySpreadConstraints:
    - maxSkew: 1 1
      topologyKey: monitoring 2
      whenUnsatisfiable: ScheduleAnyway 3
  labelSelector:
    matchLabels: 4
      app.kubernetes.io/name: thanos-ruler

```

- 1** Specify a numeric value for **maxSkew**, which defines the degree to which pods are allowed to be unevenly distributed. This field is required, and the value must be greater than zero. The value specified has a different effect depending on what value you specify for **whenUnsatisfiable**.
- 2** Specify a key of node labels for **topologyKey**. This field is required. Nodes that have a label with this key and identical values are considered to be in the same topology. The scheduler will try to put a balanced number of pods into each domain.
- 3** Specify a value for **whenUnsatisfiable**. This field is required. 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.
- 4** Specify a value for **matchLabels**. This value is used to identify the set of matching pods to which to apply the constraints.

3. Save the file to apply the changes automatically.



WARNING

When you save changes to the **user-workload-monitoring-config** config map, the pods and other resources in the **openshift-user-workload-monitoring** project might be redeployed. The running monitoring processes in that project might also restart.

6.2. 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** objects:

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

Procedure

1. Edit the **ConfigMap** object:

- 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: <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>: 1
    logLevel: <log_level> 2
```

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

2. Save the file to apply the changes. The pods for the component restart automatically when you apply the log-level change.



WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

3. 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 in 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
```

4. Check that the pods for the component are running. The following example lists the status of pods in the **openshift-user-workload-monitoring** project:

```
$ 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.

6.3. ENABLING THE QUERY LOG FILE FOR PROMETHEUS

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



IMPORTANT

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

Prerequisites

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

Procedure

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

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

2. Add **queryLogFile: <path>** 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 The full path to the file in which queries will be logged.

3. Save the file to apply the changes.



WARNING

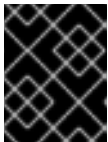
When you save changes to a monitoring config map, pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

4. Verify that the pods for the component are running. The following example command lists the status of pods in the **openshift-user-workload-monitoring** project:

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

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.

CHAPTER 7. DISABLING MONITORING FOR USER-DEFINED PROJECTS

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

7.1. DISABLING MONITORING FOR USER-DEFINED PROJECTS

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

Prerequisites

- You logged in to [OpenShift Cluster Manager](#).

Procedure

- From the OpenShift Cluster Manager Hybrid Cloud Console, select a cluster.
- Click the **Settings** tab.
- Click the **Enable user workload monitoring** check box to unselect the option, and then click **Save**.

User workload monitoring is disabled. The Prometheus, Prometheus Operator, and Thanos Ruler components are stopped in the **openshift-user-workload-monitoring** project.

7.2. EXCLUDING A USER-DEFINED PROJECT FROM MONITORING

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

Procedure

- Add the label to the project namespace:

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

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

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



NOTE

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

CHAPTER 8. ENABLING ALERT ROUTING FOR USER-DEFINED PROJECTS

In Red Hat OpenShift Service on AWS, a **dedicated-admin** can enable alert routing for user-defined projects. This process consists of two general steps:

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

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

8.1. UNDERSTANDING ALERT ROUTING FOR USER-DEFINED PROJECTS

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

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

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



NOTE

The following are 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.

8.2. ENABLING A SEPARATE ALERTMANAGER INSTANCE FOR USER-DEFINED ALERT ROUTING

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

Prerequisites

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

- The **user-workload-monitoring-config ConfigMap** object exists. This object is created by default when the cluster is created.
- You have installed the OpenShift CLI (**oc**).

Procedure

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

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

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

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

- 1 Set the **enabled** value to **true** to enable a dedicated instance of the Alertmanager for user-defined projects in a cluster. Set the value to **false** or omit the key entirely to disable the Alertmanager for user-defined projects. If you set this value to **false** or if the key is omitted, user-defined alerts are routed to the default platform Alertmanager instance.
- 2 Set the **enableAlertmanagerConfig** value to **true** to enable users to define their own alert routing configurations with **AlertmanagerConfig** objects.

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

Verification

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

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

Example output

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

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

Procedure

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

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

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

Additional resources

- [Creating alert routing for user-defined projects](#)

8.4. NEXT STEPS

- [Managing alerts](#)

CHAPTER 9. MANAGING METRICS

You can collect metrics to monitor how cluster components and your own workloads are performing.

9.1. UNDERSTANDING METRICS

In Red Hat OpenShift Service on AWS, 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 Red Hat OpenShift Service on AWS, 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

- [Prometheus client library documentation](#)

9.2. 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.

9.2.1. Deploying a sample service

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

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
```

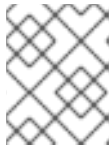
9.2.2. Specifying how a service is monitored

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

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

Prerequisites

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



NOTE

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

Procedure

1. Create a YAML file for the **ServiceMonitor** resource configuration. In this example, the file is called **example-app-service-monitor.yaml**.
2. Add the following **ServiceMonitor** resource configuration details:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: prometheus-example-monitor
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
    - interval: 30s
      port: web
      scheme: http
  selector:
    matchLabels:
      app: prometheus-example-app
```

-

This defines a **ServiceMonitor** resource that scrapes the metrics exposed by the **prometheus-example-app** sample service, which includes the **version** metric.



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. You can check that the **ServiceMonitor** resource is running:

```
$ oc -n ns1 get servicemonitor
```

Example output

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

Additional resources

- [How to scrape metrics using TLS in a ServiceMonitor configuration in a user-defined project](#)

9.3. QUERYING METRICS

The Red Hat OpenShift Service on AWS monitoring dashboard enables you 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 **dedicated-admin**, you can query one or more namespaces at a time for metrics about user-defined projects.

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.

9.3.1. Querying metrics for all projects as a cluster administrator

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



NOTE



Only dedicated administrators have access to the third-party UIs provided with Red Hat OpenShift Service on AWS monitoring.

Prerequisites

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

Procedure

1. From the **Administrator** perspective in the Red Hat OpenShift Service on AWS web console, select **Observe** → **Metrics**.
2. To add one or more queries, do any of the following:

Option	Description
Create a custom query.	<p>Add your Prometheus Query Language (PromQL) query to the Expression 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. You can use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. You can also move your mouse pointer over a suggested item to view a brief description of that item.</p>
Add multiple queries.	Select Add query .
Duplicate an existing query.	<p>Select the Options menu  next to the query, then choose Duplicate query.</p>
Disable a query from being run.	<p>Select the Options menu  next to the query and choose Disable query.</p>

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




NOTE


Queries that operate on large amounts of data might time out or overload the browser when drawing time series graphs. To avoid this, select **Hide graph** and calibrate your query using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.



NOTE

By default, the query table shows an expanded view that lists every metric and its current value. You can select  to minimize the expanded view for a query.

4. Optional: The page URL now contains the queries you ran. To use this set of queries again in the future, save this URL.
5. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot. You can select which metrics are shown by doing any of the following:

Option	Description
Hide all metrics from a query.	 Click the Options menu for the query and click Hide all series .
Hide a specific metric.	Go to the query table and click the colored square near the metric name.
Zoom into the plot and change the time range.	Either: <ul style="list-style-type: none"> ● Visually select the time range by clicking and dragging on the plot horizontally. ● Use the menu in the left upper corner to select the time range.
Reset the time range.	Select Reset zoom .
Display outputs for all queries at a specific point in time.	Hold the mouse cursor on the plot at that point. The query outputs will appear in a pop-up box.
Hide the plot.	Select Hide graph .

Additional resources

- For more information about creating PromQL queries, see the [Prometheus query documentation](#).

9.3.2. Querying metrics for user-defined projects as a developer

You can access metrics for a user-defined project as a developer or as a user with view permissions for the project.

In the **Developer** perspective, the Metrics UI includes some predefined CPU, memory, bandwidth, and network packet queries for the selected project. You can also run custom Prometheus Query Language (PromQL) queries for CPU, memory, bandwidth, network packet and application metrics 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. Developers cannot access the third-party UIs provided with Red Hat OpenShift Service on AWS monitoring.

Prerequisites

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

Procedure

1. From the **Developer** perspective in the Red Hat OpenShift Service on AWS web console, select **Observe → Metrics**.
2. Select the project that you want to view metrics for in the **Project:** list.
3. Select a query from the **Select query** list, or create a custom PromQL query based on the selected query by selecting **Show PromQL**. The metrics from the queries are visualized on the plot.



NOTE

In the Developer perspective, you can only run one query at a time.

4. Explore the visualized metrics by doing any of the following:

Option	Description
Zoom into the plot and change the time range.	Either: <ul style="list-style-type: none"> • Visually select the time range by clicking and dragging on the plot horizontally. • Use the menu in the left upper corner to select the time range.
Reset the time range.	Select Reset zoom .
Display outputs for all queries at a specific point in time.	Hold the mouse cursor on the plot at that point. The query outputs appear in a pop-up box.

Additional resources

- For more information about creating PromQL queries, see the [Prometheus query documentation](#).

9.4. GETTING DETAILED INFORMATION ABOUT A METRICS TARGET

In the **Administrator** perspective in the Red Hat OpenShift Service on AWS web console, you can use the **Metrics targets** page 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 Red Hat OpenShift Service on AWS Monitoring is not able to scrape metrics from a targeted component.

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

Prerequisites

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

Procedure

1. In the **Administrator** perspective, select **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 Red Hat OpenShift Service on AWS and user-defined projects. This page lists the following information for each target:
 - Service endpoint URL being scraped
 - ServiceMonitor component being monitored
 - The **up** or **down** status of the target
 - Namespace
 - Last scrape time
 - Duration of the last scrape
2. Optional: The list of metrics targets can be long. To find a specific target, do any of the following:

Option	Description
Filter the targets by status and source.	Select filters in the Filter list. The following filtering options are available: <ul style="list-style-type: none"> • Status filters: <ul style="list-style-type: none"> ○ Up. The target is currently up and being actively scraped for metrics. ○ Down. The target is currently down and not being scraped for metrics. • Source filters: <ul style="list-style-type: none"> ○ Platform. Platform-level targets relate only to default Red Hat OpenShift Service on AWS projects. These projects provide core Red Hat OpenShift Service on AWS functionality. ○ User. User targets relate to user-defined projects. These projects are user-created and can be customized.

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

3. Click the URL in the **Endpoint** column for a target to navigate to its **Target details** page. This page provides information about the target, including the following:
 - 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 details
 - Labels attached to the target
 - The most recent time that the target was scraped for metrics

CHAPTER 10. MANAGING ALERTS

In Red Hat OpenShift Service on AWS 4, 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 Red Hat OpenShift Service on AWS 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 underlying 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.

If you are a non-administrator user, you can create and silence alerts if you are assigned 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

10.1. ACCESSING THE ALERTING UI IN THE ADMINISTRATOR AND DEVELOPER PERSPECTIVES

The Alerting UI is accessible through the **Administrator** perspective and the **Developer** perspective of the Red Hat OpenShift Service on AWS web console.

- In the **Administrator** perspective, go to **Observe** → **Alerting**. The three main pages in the Alerting UI in this perspective are the **Alerts**, **Silences**, and **Alerting rules** pages.
- In the **Developer** perspective, go to **Observe** → **<project_name>** → **Alerts**. In this perspective, alerts, silences, and alerting rules are all managed from the **Alerts** page. The results shown in the **Alerts** page are specific to the selected project.

NOTE

In the **Developer** perspective, you can select from core Red Hat OpenShift Service on AWS and user-defined projects that you have access to in the **Project: <project_name>** list. However, alerts, silences, and alerting rules relating to core Red Hat OpenShift Service on AWS projects are not displayed if you are not logged in as a cluster administrator.

10.2. 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.

Understanding alert filters

In the **Administrator** perspective, the **Alerts** page in the Alerting UI provides details about alerts relating to default Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS projects. These projects provide core Red Hat OpenShift Service on AWS 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.

Understanding silence filters

In the **Administrator** perspective, the **Silences** page in the Alerting UI provides details about silences applied to alerts in default Red Hat OpenShift Service on AWS 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.

Understanding alerting rule filters

In the **Administrator** perspective, the **Alerting rules** page in the Alerting UI provides details about alerting rules relating to default Red Hat OpenShift Service on AWS and user-defined projects. The page includes a summary of the state, severity, and source for each alerting rule.

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

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

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.

10.3. GETTING INFORMATION ABOUT ALERTS, SILENCES, AND ALERTING RULES

The Alerting UI provides detailed information about alerts and their governing alerting rules and silences.

Prerequisites

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing alerts for.

Procedure

To obtain information about alerts in the Administrator perspective

1. Open the Red Hat OpenShift Service on AWS web console and 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
 - Labels attached to the alert
 - A link to its governing alerting rule
 - Silences for the alert, if any exist

To obtain information about silences in the Administrator perspective

1. 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

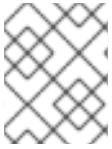
To obtain information about alerting rules in the Administrator perspective

1. 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.

To obtain information about alerts, silences, and alerting rules in the Developer perspective

1. 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

- See the [Cluster Monitoring Operator runbooks](#) to help diagnose and resolve issues that trigger specific Red Hat OpenShift Service on AWS monitoring alerts.

10.4. MANAGING SILENCES

You can create a silence for an alert in the Red Hat OpenShift Service on AWS 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.

10.4.1. Silencing alerts

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


Prerequisites

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

1. Go to **Observe** → **Alerting** → **Alerts** in the Red Hat OpenShift Service on AWS web console.

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.

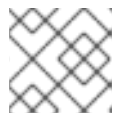
**NOTE**

You must add a comment before saving a silence.

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

To silence a specific alert in the **Developer** perspective:

1. Go to **Observe** → **<project_name>** → **Alerts** in the Red Hat OpenShift Service on AWS web console.
2. If necessary, expand the details for the alert by selecting a greater than symbol (>) next to the alert name.
3. Click the alert message in the expanded view to open the **Alert details** page for the alert.
4. Click **Silence alert** to open the **Silence alert** page with a default configuration for the alert.
5. Optional: Change the default configuration details for the silence.

**NOTE**

You must add a comment before saving a silence.

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

To silence a set of alerts by creating a silence configuration in the **Administrator** perspective:

1. Go to **Observe** → **Alerting** → **Silences** in the Red Hat OpenShift Service on AWS web console.
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**.

To silence a set of alerts by creating a silence configuration in the **Developer** perspective:

1. Go to **Observe** → **<project_name>** → **Silences** in the Red Hat OpenShift Service on AWS web console.

2. Click **Create silence**.
3. On the **Create silence** page, set the 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**.

10.4.2. Editing silences


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

Prerequisites


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

1. 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.

To edit a silence in the **Developer** perspective:

1. Go to **Observe** → **<project_name>** → **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.

10.4.3. Expiring silences

You can expire a single silence or multiple silences. Expiring a silence deactivates it permanently.



NOTE

You cannot delete expired, silenced alerts. Expired silences older than 120 hours are garbage collected.

Prerequisites

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

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.

To expire a silence in the **Developer** perspective:

1. Go to **Observe** → <project_name> → **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.

10.5. MANAGING ALERTING RULES FOR USER-DEFINED PROJECTS

Red Hat OpenShift Service on AWS monitoring ships with a set of default alerting rules. As a cluster administrator, you can view the default alerting rules.

In Red Hat OpenShift Service on AWS 4, you can create, view, edit, and remove alerting rules in user-defined projects.



IMPORTANT

Managing alerting rules for user-defined projects is only available in Red Hat OpenShift Service on AWS version 4.11 and later.

Alerting rule considerations

- The default alerting rules are used specifically for the Red Hat OpenShift Service on AWS 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.

10.5.1. 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.

Additional resources

- See the [Prometheus alerting documentation](#) for further guidelines on optimizing alerts

10.5.2. About creating alerting rules for user-defined projects

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.

- 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.

10.5.3. 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

- When you create an alerting rule, a project label is enforced on it even if a rule with the same name exists in another project.
- 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 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
  expr: version{job="prometheus-example-app"} == 0 2
  labels:
    severity: warning 3
  annotations:
    message: This is an example alert. 4
```

- 1** The name of the alerting rule you want to create.
- 2** The PromQL query expression that defines the new rule.
- 3** The severity assigned to the alert.
- 4** The message associated with the alert.

3. Apply the configuration file to the cluster:

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

Additional resources

- See [Monitoring overview](#) for details about Red Hat OpenShift Service on AWS 4 monitoring architecture.

10.5.4. 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
```

10.5.5. Listing alerting rules for all projects in a single view

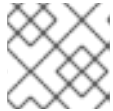
As a **dedicated-admin**, you can list alerting rules for core Red Hat OpenShift Service on AWS and user-defined projects together in a single view.

Prerequisites

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

Procedure

1. In the **Administrator** perspective, navigate 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.

10.5.6. 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 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 **<foo>** in **<namespace>**, run the following:

```
$ oc -n <namespace> delete prometheusrule <foo>
```

Additional resources

- See the [Alertmanager documentation](#)

10.6. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS

In Red Hat OpenShift Service on AWS 4, 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 Red Hat OpenShift Service on AWS to send alerts to the following receiver types:

- PagerDuty
- Webhook
- Email
- Slack

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

Red Hat OpenShift Service on AWS 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.

10.6.1. Creating alert routing for user-defined projects

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

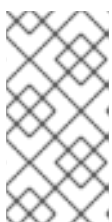
Prerequisites

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

Procedure

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

```
apiVersion: monitoring.coreos.com/v1beta1
kind: AlertmanagerConfig
metadata:
  name: example-routing
  namespace: ns1
spec:
  route:
    receiver: default
    groupBy: [job]
  receivers:
  - name: default
    webhookConfigs:
    - url: https://example.org/post
```



NOTE

For user-defined alerting rules, user-defined routing is scoped to the namespace in which the resource is defined. For example, a routing configuration defined in the **AlertmanagerConfig** object for namespace **ns1** only applies to **PrometheusRules** resources in the same namespace.

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.

10.7. APPLYING A CUSTOM CONFIGURATION TO ALERTMANAGER FOR USER-DEFINED ALERT ROUTING

If you have enabled a separate instance of Alertmanager dedicated to user-defined alert routing, you can overwrite the configuration for this instance of Alertmanager by editing the **alertmanager-user-workload** secret in the **openshift-user-workload-monitoring** namespace.

Prerequisites

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

Procedure

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

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

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

```
route:
  receiver: Default
  group_by:
  - name: Default
  routes:
  - matchers:
    - "service = prometheus-example-monitor" 1
    receiver: <receiver> 2
  receivers:
  - name: Default
  - name: <receiver>
  # <receiver_configuration>
```

1 Specifies which alerts match the route. This example shows all alerts that have the **service="prometheus-example-monitor"** label.

2 Specifies the receiver to use for the alerts group.

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

- See [the PagerDuty official site](#) for more information on PagerDuty.
- See [the PagerDuty Prometheus Integration Guide](#) to learn how to retrieve the **service_key**.
- See [Alertmanager configuration](#) for configuring alerting through different alert receivers.

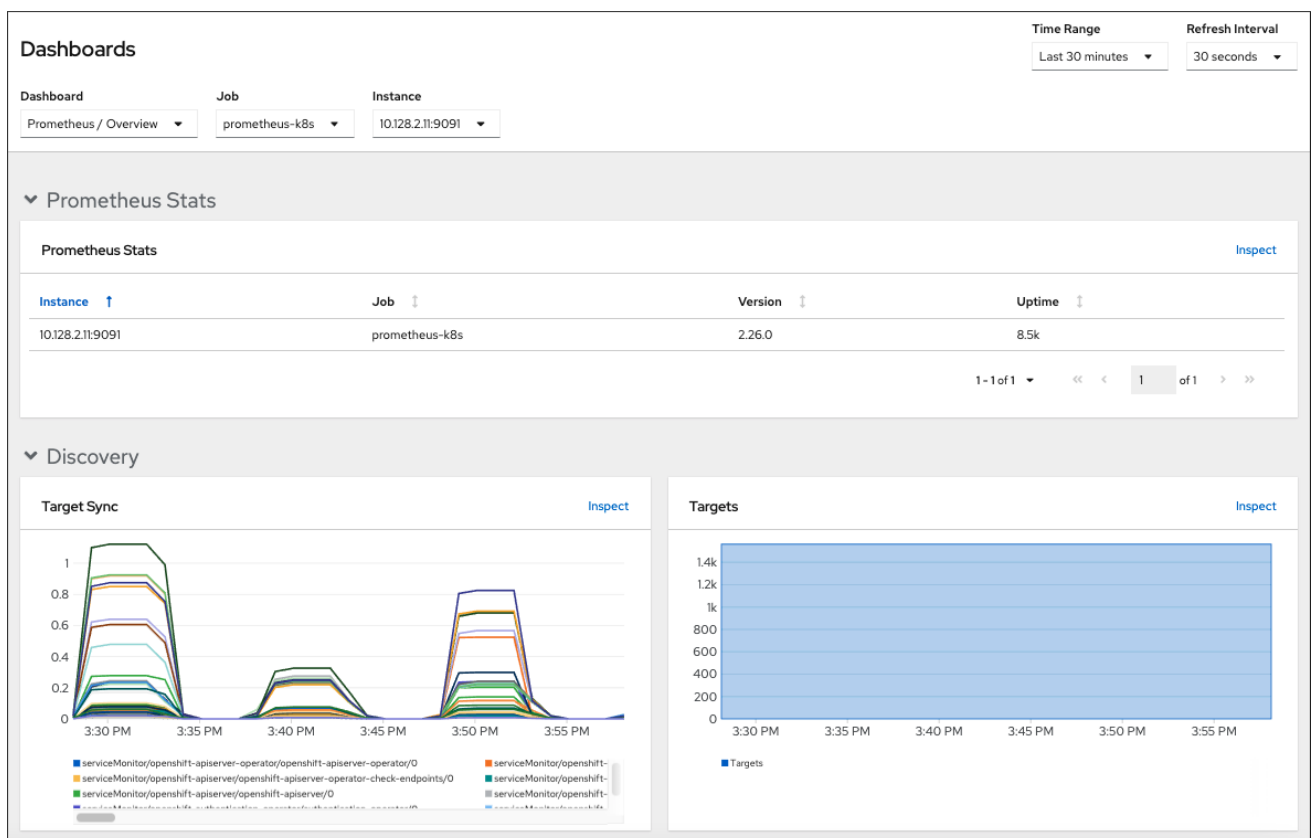
CHAPTER 11. REVIEWING MONITORING DASHBOARDS

Red Hat OpenShift Service on AWS provides monitoring dashboards that help you understand the state of user-defined projects.

Use the **Administrator** perspective to access dashboards for the core Red Hat OpenShift Service on AWS 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 11.1. Example dashboard in the Administrator perspective

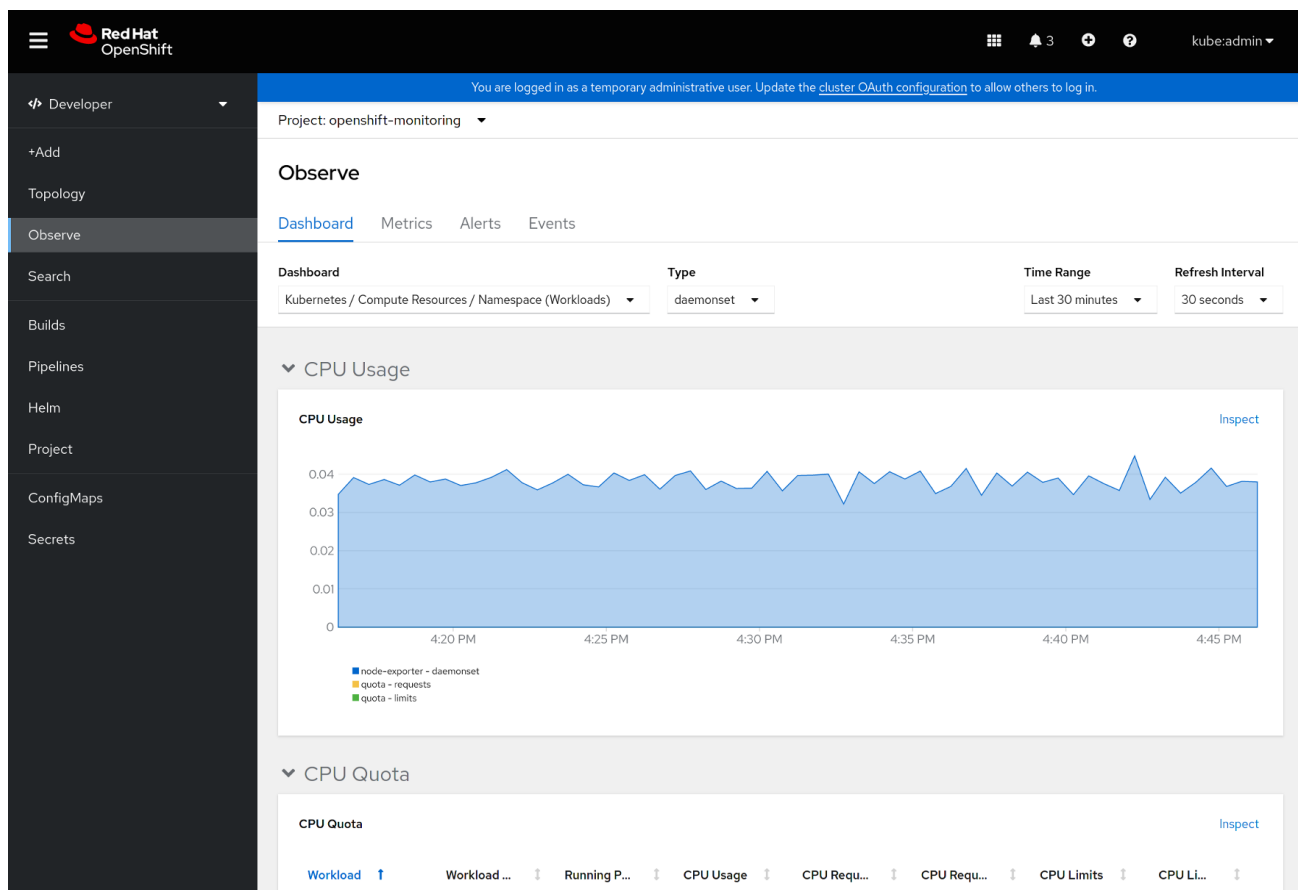


Use the **Developer** perspective to access Kubernetes compute resources dashboards that provide the following application metrics for a selected project:

- CPU usage
- Memory usage
- Bandwidth information

- Packet rate information

Figure 11.2. Example dashboard in the Developer perspective



NOTE

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

11.1. REVIEWING MONITORING DASHBOARDS AS A CLUSTER ADMINISTRATOR

In the **Administrator** perspective, you can view dashboards relating to core Red Hat OpenShift Service on AWS cluster components.

Prerequisites

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

Procedure

1. In the **Administrator** perspective in the Red Hat OpenShift Service on AWS web console, navigate 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 selecting **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.

11.2. REVIEWING MONITORING DASHBOARDS AS A DEVELOPER

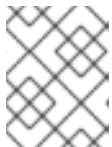
In the **Developer** perspective, you can view dashboards relating to a selected project. You must have access to monitor a project to view dashboard information for it.

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

1. In the Developer perspective in the Red Hat OpenShift Service on AWS web console, navigate to **Observe** → **Dashboard**.
2. Select a project from the **Project:** drop-down list.
3. Select a dashboard from the **Dashboard** drop-down list to see the filtered metrics.



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 selecting **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.

11.3. NEXT STEPS

- [Accessing monitoring APIs by using the CLI](#)

CHAPTER 12. ACCESSING MONITORING APIS BY USING THE CLI

In Red Hat OpenShift Service on AWS 4, 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, 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 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.

12.1. 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



NOTE

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 Red Hat OpenShift Service on AWS web console to review monitoring dashboards.

Additional resources

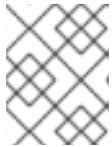
- [Reviewing monitoring dashboards](#)

12.2. 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={.spec.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"
```

12.3. 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 Red Hat OpenShift Service on AWS 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={.spec.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
...
```

12.4. 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={.spec.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:

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"]}]}}
```

12.5. ADDITIONAL RESOURCES

- [Configuring remote write storage](#)
- [Managing metrics](#)
- [Managing alerts](#)

CHAPTER 13. TROUBLESHOOTING MONITORING ISSUES

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

13.1. DETERMINING WHY USER-DEFINED PROJECT METRICS ARE UNAVAILABLE

If metrics are not displaying when monitoring user-defined projects, follow these steps to troubleshoot the issue.

Procedure

1. Query the metric name and verify that the project is correct:
 - a. From the **Developer** perspective in the web console, select **Observe** → **Metrics**.
 - b. Select the project that you want to view metrics for in the **Project:** list.
 - c. Choose a query from the **Select query** list, or run a custom PromQL query by selecting **Show PromQL**.
The metrics are displayed in a chart.

Queries must be done on a per-project basis. The metrics that are shown relate to the project that you have selected.
2. Verify that the pod that you want metrics from is actively serving metrics. Run the following **oc exec** command into a pod to target the **podIP**, **port**, and **/metrics**.

```
$ oc exec <sample_pod> -n <sample_namespace> -- curl <target_pod_IP>:<port>/metrics
```



NOTE

You must run the command on a pod that has **curl** installed.

The following example output shows a result with a valid version metric.

Example output

```
% Total % Received % Xferd Average Speed Time Time Time Current
          Dload Upload Total Spent Left Speed
# HELP version Version information about this binary-- --:--:-- --:--:-- 0
# TYPE version gauge
version{version="v0.1.0"} 1
100 102 100 102 0 0 51000 0 --:--:-- --:--:-- --:--:-- 51000
```

An invalid output indicates that there is a problem with the corresponding application.

3. If you are using a **PodMonitor** CRD, verify that the **PodMonitor** CRD is configured to point to the correct pods using label matching. For more information, see the Prometheus Operator documentation.
4. If you are using a **ServiceMonitor** CRD, and if the **/metrics** endpoint of the pod is showing metric data, follow these steps to verify the configuration:

- a. Verify that the service is pointed to the correct **/metrics** endpoint. The service **labels** in this output must match the services monitor **labels** and the **/metrics** endpoint defined by the service in the subsequent steps.

```
$ oc get service
```

Example output

```
apiVersion: v1
kind: Service 1
metadata:
  labels: 2
    app: prometheus-example-app
    name: prometheus-example-app
    namespace: ns1
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
    name: web
  selector:
    app: prometheus-example-app
  type: ClusterIP
```

- 1** Specifies that this is a service API.
- 2** Specifies the labels that are being used for this service.

- b. Query the **serviceIP**, **port**, and **/metrics** endpoints to see if the same metrics from the **curl** command you ran on the pod previously:

- i. Run the following command to find the service IP:

```
$ oc get service -n <target_namespace>
```

- ii. Query the **/metrics** endpoint:

```
$ oc exec <sample_pod> -n <sample_namespace> -- curl <service_IP>:
<port>/metrics
```

Valid metrics are returned in the following example.

Example output

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload  Total   Spent    Left  Speed
100 102 100 102  0  0 51000  0  ---:--:--  ---:--:--  ---:--:--  99k
# HELP version Version information about this binary
# TYPE version gauge
version{version="v0.1.0"} 1
```

- c. Use label matching to verify that the **ServiceMonitor** object is configured to point to the

desired service. To do this, compare the **Service** object from the **oc get service** output to the **ServiceMonitor** object from the **oc get servicemonitor** output. The labels must match for the metrics to be displayed.

For example, from the previous steps, notice how the **Service** object has the **app: prometheus-example-app** label and the **ServiceMonitor** object has the same **app: prometheus-example-app** match label.

5. If everything looks valid and the metrics are still unavailable, please contact the support team for further help.

13.2. DETERMINING WHY PROMETHEUS IS CONSUMING A LOT OF DISK SPACE

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

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

You can use the following measures when Prometheus consumes a lot of disk:

- **Check the time series database (TSDB) status using the Prometheus HTTP API** for more information about which labels are creating the most time series data. Doing so requires cluster administrator privileges.
- **Check the number of scrape samples** that are being collected.
- **Reduce the number of unique time series that are created** by reducing the number of unbound attributes that are assigned to user-defined metrics.



NOTE

Using attributes that are bound to a limited set of possible values reduces the number of potential key-value pair combinations.

- **Enforce limits on the number of samples that can be scraped** across user-defined projects. This requires cluster administrator privileges.

Prerequisites

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

Procedure

1. In the **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])))
```

- 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 Red Hat OpenShift Service on AWS project** create a Red Hat support case on the [Red Hat Customer Portal](#).
- Review the TSDB status using the Prometheus HTTP API by following these steps when logged in as a **dedicated-admin**:

- Get the Prometheus API route URL by running the following command:

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

- Extract an authentication token by running the following command:

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

- 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 a scrape sample limit for user-defined projects](#)
- [Submitting a support case](#)

CHAPTER 14. CONFIG MAP REFERENCE FOR THE CLUSTER MONITORING OPERATOR

14.1. CLUSTER MONITORING OPERATOR CONFIGURATION REFERENCE

Parts of Red Hat OpenShift Service on AWS 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 is invalid YAML data, the Cluster Monitoring Operator stops reconciling the resources and reports **Degraded=True** in the status conditions of the Operator.

14.2. ADDITIONALALERTMANAGERCONFIG

14.2.1. Description

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

14.2.2. Required

- **apiVersion**

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

Property	Type	Description
----------	------	-------------

Property	Type	Description
apiVersion	string	Defines the API version of Alertmanager. Possible values are v1 or v2 . The default is v2 .
bearerToken	*v1.SecretKeySelector	Defines the secret key reference containing the bearer token to use when authenticating to Alertmanager.
pathPrefix	string	Defines the path prefix to add in front of the push endpoint path.
scheme	string	Defines the URL scheme to use when communicating with Alertmanager instances. Possible values are http or https . The default value is http .
staticConfigs	[]string	A list of statically configured Alertmanager endpoints in the form of <hosts>:<port> .
timeout	*string	Defines the timeout value used when sending alerts.
tlsConfig	TLSCConfig	Defines the TLS settings to use for Alertmanager connections.

14.3. ALERTMANAGERMAINCONFIG

14.3.1. Description

The **AlertmanagerMainConfig** resource defines settings for the Alertmanager component in the **openshift-monitoring** namespace.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
enabled	*bool	A Boolean flag that enables or disables the main Alertmanager instance in the openshift-monitoring namespace. The default value is true .

Property	Type	Description
enableUserAlertmanagerConfig	bool	A Boolean flag that enables or disables user-defined namespaces to be selected for AlertmanagerConfig lookups. This setting only applies if the user workload monitoring instance of Alertmanager is not enabled. The default value is false .
logLevel	string	Defines the log level setting for Alertmanager. The possible values are: error , warn , info , debug . The default value is info .
nodeSelector	map[string]string	Defines the nodes on which the Pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Alertmanager container.
secrets	[]string	Defines a list of secrets to be mounted into Alertmanager. The secrets must reside within the same namespace as the Alertmanager object. They are added as volumes named secret- <secret-name> and mounted at /etc/alertmanager/secrets/<secret-name> in the alertmanager container of the Alertmanager pods.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Alertmanager. Use this setting to configure the persistent volume claim, including storage class, volume size, and name.

14.4. ALERTMANAGERUSERWORKLOADCONFIG

14.4.1. Description

The **AlertmanagerUserWorkloadConfig** resource defines the settings for the Alertmanager instance used for user-defined projects.

Appears in: [UserWorkloadConfiguration](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables a dedicated instance of Alertmanager for user-defined alerts in the openshift-user-workload-monitoring namespace. The default value is false .
enableAlertmanagerConfig	bool	A Boolean flag to enable or disable user-defined namespaces to be selected for AlertmanagerConfig lookup. The default value is false .
logLevel	string	Defines the log level setting for Alertmanager for user workload monitoring. The possible values are error , warn , info , and debug . The default value is info .
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Alertmanager container.
secrets	[]string	Defines a list of secrets to be mounted into Alertmanager. The secrets must be located within the same namespace as the Alertmanager object. They are added as volumes named secret-<i><secret-name></i> and mounted at /etc/alertmanager/secrets/<i><secret-name></i> in the alertmanager container of the Alertmanager pods.
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
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.

14.5. CLUSTERMONITORINGCONFIGURATION

14.5.1. Description

The **ClusterMonitoringConfiguration** resource defines settings that customize the default platform monitoring stack through the **cluster-monitoring-config** config map in the **openshift-monitoring** namespace.

Property	Type	Description
alertmanagerMain	* AlertmanagerMainConfig	AlertmanagerMainConfig defines settings for the Alertmanager component in the openshift-monitoring namespace.
enableUserWorkload	*bool	UserWorkloadEnabled is a Boolean flag that enables monitoring for user-defined projects.
k8sPrometheusAdapter	* K8sPrometheusAdapter	K8sPrometheusAdapter defines settings for the Prometheus Adapter component.
kubeStateMetrics	* KubeStateMetricsConfig	KubeStateMetricsConfig defines settings for the kube-state-metrics agent.
metricsServer	* MetricsServerConfig	MetricsServer defines settings for the Metrics Server component.
prometheusK8s	* PrometheusK8sConfig	PrometheusK8sConfig defines settings for the Prometheus component.

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

14.6. DEDICATEDSERVICEMONITORS

14.6.1. Description



IMPORTANT

This setting is deprecated and is planned to be removed in a future Red Hat OpenShift Service on AWS version. In the current version, this setting still exists but has no effect.

You can use the **DedicatedServiceMonitors** resource to configure dedicated Service Monitors for the Prometheus Adapter

Appears in: [K8sPrometheusAdapter](#)

Property	Type	Description
enabled	bool	When enabled is set to true , the Cluster Monitoring Operator (CMO) deploys a dedicated Service Monitor that exposes the kubelet /metrics/resource endpoint. This Service Monitor sets honorTimestamps: true and only keeps metrics that are relevant for the pod resource queries of Prometheus Adapter. Additionally, Prometheus Adapter is configured to use these dedicated metrics. Overall, this feature improves the consistency of Prometheus Adapter-based CPU usage measurements used by, for example, the oc adm top pod command or the Horizontal Pod Autoscaler.

14.7. K8SPROMETHEUSADAPTER

14.7.1. Description

The **K8sPrometheusAdapter** resource defines settings for the Prometheus Adapter component.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
audit	*Audit	Defines the audit configuration used by the Prometheus Adapter instance. Possible profile values are: metadata , request , requestresponse , and none . The default value is metadata .
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the PrometheusAdapter container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

Property	Type	Description
dedicatedServiceMonitors	* DedicatedServiceMonitors	Defines dedicated service monitors.

14.8. KUBESTATEMETRICSCONFIG

14.8.1. 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.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the KubeStateMetrics container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

14.9. METRICSSERVERCONFIG

14.9.1. Description



IMPORTANT

Metrics Server 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](#).

The **MetricsServerConfig** resource defines settings for the Metrics Server component. Note that this setting only applies when the **TechPreviewNoUpgrade** feature gate is enabled.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
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.

14.10. PROMETHEUSOPERATORADMISSIONWEBHOOKCONFIG

14.10.1. Description

The **PrometheusOperatorAdmissionWebhookConfig** resource defines settings for the admission webhook workload for Prometheus Operator.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
resources	*v1.ResourceRequirements	Defines resource requests and limits for the prometheus-operator-admission-webhook container.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

14.11. MONITORINGPLUGINCONFIG

14.11.1. 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.

Property	Type	Description
resources	*v1.ResourceRequirements	Defines resource requests and limits for the console-plugin container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines a pod's topology spread constraints.

14.12. NODEEXPORTERCOLLECTORBUDDYINFOCONFIG

14.12.1. Description

The **NodeExporterCollectorBuddyInfoConfig** resource works as an on/off switch for the **buddyinfo** collector of the **node-exporter** agent. By default, the **buddyinfo** collector is disabled.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the buddyinfo collector.

14.13. NODEEXPORTERCOLLECTORCONFIG

14.13.1. Description

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

Appears in: [NodeExporterConfig](#)

Property	Type	Description
cpufreq	NodeExporterCollectorCpufreqConfig	Defines the configuration of the cpufreq collector, which collects CPU frequency statistics. Disabled by default.
tcpstat	NodeExporterCollectorTcpStatConfig	Defines the configuration of the tcpstat collector, which collects TCP connection statistics. Disabled by default.

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

14.14. NODEEXPORTERCOLLECTORCPUFREQCONFIG

14.14.1. Description

Use the **NodeExporterCollectorCpufreqConfig** resource to enable or disable the **cpufreq** collector of the **node-exporter** agent. By default, the **cpufreq** collector is disabled. Under certain circumstances, enabling the **cpufreq** collector increases CPU usage on machines with many cores. If you enable this collector and have machines with many cores, monitor your systems closely for excessive CPU usage.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the cpufreq collector.

14.15. NODEEXPORTERCOLLECTORKSMDCONFIG

14.15.1. Description

Use the **NodeExporterCollectorKSMDConfig** resource to enable or disable the **ksmd** collector of the **node-exporter** agent. By default, the **ksmd** collector is disabled.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the ksmd collector.

14.16. NODEEXPORTERCOLLECTORMOUNTSTATSCONFIG

14.16.1. Description

Use the **NodeExporterCollectorMountStatsConfig** resource to enable or disable the **mountstats** collector of the **node-exporter** agent. By default, the **mountstats** collector is disabled. If you enable the collector, the following metrics become available: **node_mountstats_nfs_read_bytes_total**, **node_mountstats_nfs_write_bytes_total**, and **node_mountstats_nfs_operations_requests_total**. Be aware that these metrics can have a high cardinality. If you enable this collector, closely monitor any increases in memory usage for the **prometheus-k8s** pods.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the mountstats collector.

14.17. NODEEXPORTERCOLLECTORNETCLASSCONFIG

14.17.1. Description

Use the **NodeExporterCollectorNetClassConfig** resource to enable or disable the **netclass** collector of the **node-exporter** agent. By default, the **netclass** collector is enabled. If you disable this collector, these metrics become unavailable: **node_network_info**, **node_network_address_assign_type**, **node_network_carrier**, **node_network_carrier_changes_total**, **node_network_carrier_up_changes_total**, **node_network_carrier_down_changes_total**, **node_network_device_id**, **node_network_dormant**, **node_network_flags**, **node_network_iface_id**, **node_network_iface_link**, **node_network_iface_link_mode**, **node_network_mtu_bytes**, **node_network_name_assign_type**, **node_network_net_dev_group**, **node_network_speed_bytes**, **node_network_transmit_queue_length**, and **node_network_protocol_type**.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the netclass collector.
useNetlink	bool	A Boolean flag that activates the netlink implementation of the netclass collector. The default value is true , which activates the netlink mode. This implementation improves the performance of the netclass collector.

14.18. NODEEXPORTERCOLLECTORNETDEVCONFIG

14.18.1. Description

Use the **NodeExporterCollectorNetDevConfig** resource to enable or disable the **netdev** collector of the **node-exporter** agent. By default, the **netdev** collector is enabled. If disabled, these metrics become unavailable: **node_network_receive_bytes_total**, **node_network_receive_compressed_total**, **node_network_receive_drop_total**, **node_network_receive_errs_total**, **node_network_receive_fifo_total**, **node_network_receive_frame_total**, **node_network_receive_multicast_total**, **node_network_receive_nohandler_total**, **node_network_receive_packets_total**, **node_network_transmit_bytes_total**, **node_network_transmit_carrier_total**, **node_network_transmit_colls_total**, **node_network_transmit_compressed_total**, **node_network_transmit_drop_total**, **node_network_transmit_errs_total**, **node_network_transmit_fifo_total**, and **node_network_transmit_packets_total**.

Appears in: [NodeExporterCollectorConfig](#)

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

14.19. NODEEXPORTERCOLLECTORPROCESSESCONFIG

14.19.1. Description

Use the **NodeExporterCollectorProcessesConfig** resource to enable or disable the **processes** collector of the **node-exporter** agent. If the collector is enabled, the following metrics become available: **node_processes_max_processes**, **node_processes_pids**, **node_processes_state**, **node_processes_threads**, **node_processes_threads_state**. The metric **node_processes_state** and **node_processes_threads_state** can have up to five series each, depending on the state of the processes and threads. The possible states of a process or a thread are: **D** (UNINTERRUPTABLE_SLEEP), **R** (RUNNING & RUNNABLE), **S** (INTERRUPTABLE_SLEEP), **T** (STOPPED), or **Z** (ZOMBIE). By default, the **processes** collector is disabled.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the processes collector.

14.20. NODEEXPORTERCOLLECTORSYSTEMDCONFIG

14.20.1. Description

Use the **NodeExporterCollectorSystemdConfig** resource to enable or disable the **systemd** collector of the **node-exporter** agent. By default, the **systemd** collector is disabled. If enabled, the following metrics become available: **node_systemd_system_running**, **node_systemd_units**, **node_systemd_version**. If the unit uses a socket, it also generates the following metrics: **node_systemd_socket_accepted_connections_total**, **node_systemd_socket_current_connections**, **node_systemd_socket_refused_connections_total**. You can use the **units** parameter to select the **systemd** units to be included by the **systemd** collector. The selected units are used to generate the **node_systemd_unit_state** metric, which shows the state of each **systemd** unit. However, this metric's cardinality might be high (at least five series per unit per node). If you enable this collector with a long list of selected units, closely monitor the **prometheus-k8s** deployment for excessive memory usage. Note that the **node_systemd_timer_last_trigger_seconds** metric is only shown if you have configured the value of the **units** parameter as **logrotate.timer**.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the systemd collector.
units	[]string	A list of regular expression (regex) patterns that match systemd units to be included by the systemd collector. By default, the list is empty, so the collector exposes no metrics for systemd units.

14.21. NODEEXPORTERCOLLECTORTCPSTATCONFIG

14.21.1. Description

The **NodeExporterCollectorTcpStatConfig** resource works as an on/off switch for the **tcpstat** collector of the **node-exporter** agent. By default, the **tcpstat** collector is disabled.

Appears in: [NodeExporterCollectorConfig](#)

Property	Type	Description
enabled	bool	A Boolean flag that enables or disables the tcpstat collector.

14.22. NODEEXPORTERCONFIG

14.22.1. Description

The **NodeExporterConfig** resource defines settings for the **node-exporter** agent.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
collectors	NodeExporterCollectorConfig	Defines which collectors are enabled and their additional configuration parameters.
maxProcs	uint32	The target number of CPUs on which the node-exporter's process will run. The default value is 0 , which means that node-exporter runs on all CPUs. If a kernel deadlock occurs or if performance degrades when reading from sysfs concurrently, you can change this value to 1 , which limits node-exporter to running on one CPU. For nodes with a high CPU count, you can set the limit to a low number, which saves resources by preventing Go routines from being scheduled to run on all CPUs. However, I/O performance degrades if the maxProcs value is set too low and there are many metrics to collect.

Property	Type	Description
ignoredNetworkDevices	*[]string	A list of network devices, defined as regular expressions, that you want to exclude from the relevant collector configuration such as netdev and netclass . If no list is specified, the Cluster Monitoring Operator uses a predefined list of devices to be excluded to minimize the impact on memory usage. If the list is empty, no devices are excluded. If you modify this setting, monitor the prometheus-k8s deployment closely for excessive memory usage.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the NodeExporter container.

14.23. OPENSIFTSTATEMETRICSCONFIG

14.23.1. Description

The **OpenShiftStateMetricsConfig** resource defines settings for the **openshift-state-metrics** agent.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the OpenShiftStateMetrics container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

14.24. PROMETHEUSK8SCONFIG

14.24.1. Description

The **PrometheusK8sConfig** resource defines settings for the Prometheus component.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Configures additional Alertmanager instances that receive alerts from the Prometheus component. By default, no additional Alertmanager instances are configured.
enforcedBodySizeLimit	string	Enforces a body size limit for Prometheus scraped metrics. If a scraped target's body response is larger than the limit, the scrape will fail. The following values are valid: an empty value to specify no limit, a numeric value in Prometheus size format (such as 64MB), or the string automatic , which indicates that the limit will be automatically calculated based on cluster capacity. The default value is empty, which indicates no limit.
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: error , warn , info , and debug . The default value is info .
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.

Property	Type	Description
queryLogFile	string	Specifies the file to which PromQL queries are logged. This setting can be either a filename, in which case the queries are saved to an emptyDir volume at /var/log/prometheus , or a full path to a location where an emptyDir volume will be mounted and the queries saved. Writing to /dev/stderr , /dev/stdout or /dev/null is supported, but writing to any other /dev/ path is not supported. Relative paths are also not supported. By default, PromQL queries are not logged.
remoteWrite	[]RemoteWriteSpec	Defines the remote write configuration, including URL, authentication, and relabeling settings.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Prometheus container.
retention	string	Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: [0-9]+(ms s m h d w y) (ms = milliseconds, s = seconds, m = minutes, h = hours, d = days, w = weeks, y = years). The default value is 15d .
retentionSize	string	Defines the maximum amount of disk space used by data blocks plus the write-ahead log (WAL). Supported values are B, KB, KiB, MB, MiB, GB, GiB, TB, TiB, PB, PiB, EB , and EiB . By default, no limit is defined.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

Property	Type	Description
collectionProfile	CollectionProfile	Defines the metrics collection profile that Prometheus uses to collect metrics from the platform components. Supported values are full or minimal . In the full profile (default), Prometheus collects all metrics that are exposed by the platform components. In the minimal profile, Prometheus only collects metrics necessary for the default platform alerts, recording rules, telemetry, and console dashboards.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Prometheus. Use this setting to configure the persistent volume claim, including storage class, volume size and name.

14.25. PROMETHEUSOPERATORCONFIG

14.25.1. Description

The **PrometheusOperatorConfig** resource defines settings for the Prometheus Operator component.

Appears in: [ClusterMonitoringConfiguration](#), [UserWorkloadConfiguration](#)

Property	Type	Description
logLevel	string	Defines the log level settings for Prometheus Operator. The possible values are error , warn , info , and debug . The default value is info .
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the PrometheusOperator container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.

Property	Type	Description
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

14.26. PROMETHEUSRESTRICTEDCONFIG

14.26.1. Description

The **PrometheusRestrictedConfig** resource defines the settings for the Prometheus component that monitors user-defined projects.

Appears in: [UserWorkloadConfiguration](#)

Property	Type	Description
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Configures additional Alertmanager instances that receive alerts from the Prometheus component. By default, no additional Alertmanager instances are configured.
enforcedLabelLimit	*uint64	Specifies a per-scrape limit on the number of labels accepted for a sample. If the number of labels exceeds this limit after metric relabeling, the entire scrape is treated as failed. The default value is 0 , which means that no limit is set.
enforcedLabelNameLengthLimit	*uint64	Specifies a per-scrape limit on the length of a label name for a sample. If the length of a label name exceeds this limit after metric relabeling, the entire scrape is treated as failed. The default value is 0 , which means that no limit is set.
enforcedLabelValueLengthLimit	*uint64	Specifies a per-scrape limit on the length of a label value for a sample. If the length of a label value exceeds this limit after metric relabeling, the entire scrape is treated as failed. The default value is 0 , which means that no limit is set.

Property	Type	Description
enforcedSampleLimit	*uint64	Specifies a global limit on the number of scraped samples that will be accepted. This setting overrides the SampleLimit value set in any user-defined ServiceMonitor or PodMonitor object if the value is greater than enforcedTargetLimit . Administrators can use this setting to keep the overall number of samples under control. The default value is 0 , which means that no limit is set.
enforcedTargetLimit	*uint64	Specifies a global limit on the number of scraped targets. This setting overrides the TargetLimit value set in any user-defined ServiceMonitor or PodMonitor object if the value is greater than enforcedSampleLimit . Administrators can use this setting to keep the overall number of targets under control. The default value is 0 .
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 error , warn , info , and debug . The default setting is info .
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.

Property	Type	Description
queryLogFile	string	Specifies the file to which PromQL queries are logged. This setting can be either a filename, in which case the queries are saved to an emptyDir volume at /var/log/prometheus , or a full path to a location where an emptyDir volume will be mounted and the queries saved. Writing to /dev/stderr , /dev/stdout or /dev/null is supported, but writing to any other /dev/ path is not supported. Relative paths are also not supported. By default, PromQL queries are not logged.
remoteWrite	[]RemoteWriteSpec	Defines the remote write configuration, including URL, authentication, and relabeling settings.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Prometheus container.
retention	string	Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: [0-9]+(ms s m h d w y) (ms = milliseconds, s = seconds, m = minutes, h = hours, d = days, w = weeks, y = years). The default value is 15d .
retentionSize	string	Defines the maximum amount of disk space used by data blocks plus the write-ahead log (WAL). Supported values are B, KB, KiB, MB, MiB, GB, GiB, TB, TiB, PB, PiB, EB , and EiB . The default value is nil .
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 Prometheus. Use this setting to configure the storage class and size of a volume.

14.27. REMOTEWITESPEC

14.27.1. Description

The **RemoteWriteSpec** resource defines the settings for remote write storage.

14.27.2. 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.
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.

Property	Type	Description
oauth2	*monv1.OAuth2	Defines OAuth2 authentication settings for the remote write endpoint.
proxyUrl	string	Defines an optional proxy URL.
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.
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.

14.28. TLSCONFIG

14.28.1. Description

The **TLSCONFIG** resource configures the settings for TLS connections.

14.28.2. 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 true , disables the verification of the remote host's certificate and name.

14.29. TELEMETRCLIENTCONFIG

14.29.1. Description

TelemeterClientConfig defines settings for the Telemeter Client component.

14.29.2. Required

- **nodeSelector**
- **tolerations**

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the TelemeterClient container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

14.30. THANOSQUERIERCONFIG

14.30.1. Description

The **ThanosQuerierConfig** resource defines settings for the Thanos Querier component.

Appears in: [ClusterMonitoringConfiguration](#)

Property	Type	Description
enableRequestLogging	bool	A Boolean flag that enables or disables request logging. The default value is false .
logLevel	string	Defines the log level setting for Thanos Querier. The possible values are error , warn , info , and debug . The default value is info .
enableCORS	bool	A Boolean flag that enables setting CORS headers. The headers allow access from any origin. The default value is false .
nodeSelector	map[string]string	Defines the nodes on which the pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Thanos Querier container.
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.

14.31. THANOSRULERCONFIG

14.31.1. Description

The **ThanosRulerConfig** resource defines configuration for the Thanos Ruler instance for user-defined projects.

Appears in: [UserWorkloadConfiguration](#)

Property	Type	Description
----------	------	-------------

Property	Type	Description
additionalAlertmanagerConfigs	[]AdditionalAlertmanagerConfig	Configures how the Thanos Ruler component communicates with additional Alertmanager instances. The default value is nil .
logLevel	string	Defines the log level setting for Thanos Ruler. The possible values are error , warn , info , and debug . The default value is info .
nodeSelector	map[string]string	Defines the nodes on which the Pods are scheduled.
resources	*v1.ResourceRequirements	Defines resource requests and limits for the Alertmanager container.
retention	string	Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: [0-9]+(ms s m h d w y) (ms = milliseconds, s = seconds, m = minutes, h = hours, d = days, w = weeks, y = years). The default value is 15d .
tolerations	[]v1.Toleration	Defines tolerations for the pods.
topologySpreadConstraints	[]v1.TopologySpreadConstraint	Defines the pod's topology spread constraints.
volumeClaimTemplate	*monv1.EmbeddedPersistentVolumeClaim	Defines persistent storage for Thanos Ruler. Use this setting to configure the storage class and size of a volume.

14.32. USERWORKLOADCONFIGURATION

14.32.1. Description

The **UserWorkloadConfiguration** resource defines the settings responsible for user-defined projects in the **user-workload-monitoring-config** config map in the **openshift-user-workload-monitoring** namespace. You can only enable **UserWorkloadConfiguration** after you have set **enableUserWorkload** to **true** in the **cluster-monitoring-config** config map under the **openshift-monitoring** namespace.

Property	Type	Description
alertmanager	* AlertmanagerUserWorkloadConfig	Defines the settings for the Alertmanager component in user workload monitoring.
prometheus	* PrometheusRestrictedConfig	Defines the settings for the Prometheus component in user workload monitoring.
prometheusOperator	* PrometheusOperatorConfig	Defines the settings for the Prometheus Operator component in user workload monitoring.
thanosRuler	* ThanosRulerConfig	Defines the settings for the Thanos Ruler component in user workload monitoring.