OpenShift Container Platform 4.11

Monitoring

Configuring and using the monitoring stack in OpenShift Container Platform
Configuring and using the monitoring stack in OpenShift Container Platform
Abstract

This document provides instructions for configuring and using the Prometheus monitoring stack in OpenShift Container Platform.
Table of Contents

CHAPTER 1. MONITORING OVERVIEW ................................................................. 7
  1.1. ABOUT OPENSHIFT CONTAINER PLATFORM MONITORING 7
  1.2. UNDERSTANDING THE MONITORING STACK 7
      1.2.1. Default monitoring components 8
      1.2.2. Default monitoring targets 10
      1.2.3. Components for monitoring user-defined projects 10
      1.2.4. Monitoring targets for user-defined projects 11
  1.3. GLOSSARY OF COMMON TERMS FOR OPENSHIFT CONTAINER PLATFORM MONITORING 11
  1.4. ADDITIONAL RESOURCES 13
  1.5. NEXT STEPS 14

CHAPTER 2. CONFIGURING THE MONITORING STACK ........................................ 15
  2.1. PREREQUISITES 15
  2.2. MAINTENANCE AND SUPPORT FOR MONITORING 15
      2.2.1. Support considerations for monitoring 15
      2.2.2. Support policy for monitoring Operators 16
  2.3. PREPARING TO CONFIGURE THE MONITORING STACK 16
      2.3.1. Creating a cluster monitoring config map 16
      2.3.2. Creating a user-defined workload monitoring config map 17
  2.4. CONFIGURING THE MONITORING STACK 18
  2.5. CONFIGURABLE MONITORING COMPONENTS 21
  2.6. MOVING MONITORING COMPONENTS TO DIFFERENT NODES 22
  2.7. ASSIGNING TOLERATIONS TO MONITORING COMPONENTS 26
  2.8. SETTING THE BODY SIZE LIMIT FOR METRICS SCRAPING 28
  2.9. CONFIGURING PERSISTENT STORAGE 30
      2.9.1. Persistent storage prerequisites 30
      2.9.2. Configuring a local persistent volume claim 30
      2.9.3. Resizing a persistent storage volume 34
      2.9.4. Modifying the retention time and size for Prometheus metrics data 38
      2.9.5. Modifying the retention time for Thanos Ruler metrics data 41
  2.10. CONFIGURING REMOTE WRITE STORAGE 42
      2.10.1. Supported remote write authentication settings 45
          2.10.1.1. Config map location for authentication settings 46
          2.10.1.2. Example remote write authentication settings 46
  2.11. ADDING CLUSTER ID LABELS TO METRICS 52
      2.11.1. Creating cluster ID labels for metrics 52
  2.12. CONTROLLING THE IMPACT OF UNBOUND METRICS ATTRIBUTES IN USER-DEFINED PROJECTS 54
      2.12.1. Setting scrape sample and label limits for user-defined projects 55
      2.12.2. Creating scrape sample alerts 57

CHAPTER 3. CONFIGURING EXTERNAL ALERTMANAGER INSTANCES .......................... 59
  3.1. ATTACHING ADDITIONAL LABELS TO YOUR TIME SERIES AND ALERTS 62
  3.2. SETTING LOG LEVELS FOR MONITORING COMPONENTS 65
  3.3. ENABLING THE QUERY LOG FILE FOR PROMETHEUS 67
  3.4. ENABLING QUERY LOGGING FOR THANOS QUERIER 70

CHAPTER 4. SETTING AUDIT LOG LEVELS FOR THE PROMETHEUS ADAPTER ............... 72
  4.1. DISABLING THE LOCAL ALERTMANAGER 74
  4.2. NEXT STEPS 74

CHAPTER 5. ENABLING MONITORING FOR USER-DEFINED PROJECTS ........................ 76
  5.1. ENABLING MONITORING FOR USER-DEFINED PROJECTS 76
5.2. GRANTING USERS PERMISSION TO MONITOR USER-DEFINED PROJECTS
5.2.1. Granting user permissions by using the web console
5.2.2. Granting user permissions by using the CLI
5.3. GRANTING USERS PERMISSION TO CONFIGURE MONITORING FOR USER-DEFINED PROJECTS
5.4. ACCESSING METRICS FROM OUTSIDE THE CLUSTER FOR CUSTOM APPLICATIONS
5.5. EXCLUDING A USER-DEFINED PROJECT FROM MONITORING
5.6. DISABLING MONITORING FOR USER-DEFINED PROJECTS
5.7. NEXT STEPS

CHAPTER 6. ENABLING ALERT ROUTING FOR USER-DEFINED PROJECTS
6.1. UNDERSTANDING ALERT ROUTING FOR USER-DEFINED PROJECTS
6.2. ENABLING THE PLATFORM ALERTMANAGER INSTANCE FOR USER-DEFINED ALERT ROUTING
6.3. ENABLING A SEPARATE ALERTMANAGER INSTANCE FOR USER-DEFINED ALERT ROUTING
6.4. GRANTING USERS PERMISSION TO CONFIGURE ALERT ROUTING FOR USER-DEFINED PROJECTS
6.5. NEXT STEPS

CHAPTER 7. MANAGING METRICS
7.1. UNDERSTANDING METRICS
7.2. SETTING UP METRICS COLLECTION FOR USER-DEFINED PROJECTS
7.2.1. Deploying a sample service
7.2.2. Specifying how a service is monitored
7.3. NEXT STEPS

CHAPTER 8. QUERYING METRICS
8.1. ABOUT QUERYING METRICS
8.1.1. Querying metrics for all projects as a cluster administrator
8.1.2. Querying metrics for user-defined projects as a developer
8.1.3. Exploring the visualized metrics
8.2. NEXT STEPS

CHAPTER 9. MANAGING METRICS TARGETS
9.1. ACCESSING THE METRICS TARGETS PAGE IN THE ADMINISTRATOR PERSPECTIVE
9.2. SEARCHING AND FILTERING METRICS TARGETS
9.3. GETTING DETAILED INFORMATION ABOUT A TARGET
9.4. NEXT STEPS

CHAPTER 10. MANAGING ALERTS
10.1. ACCESSING THE ALERTING UI IN THE ADMINISTRATOR AND DEVELOPER PERSPECTIVES
10.2. SEARCHING AND FILTERING ALERTS, SILENCES, AND ALERTING RULES
10.3. GETTING INFORMATION ABOUT ALERTS, SILENCES, AND ALERTING RULES
10.4. MANAGING SILENCES
10.4.1. Silencing alerts
10.4.2. Editing silences
10.4.3. Expiring silences
10.5. MANAGING ALERTING RULES FOR USER-DEFINED PROJECTS
10.5.1. Optimizing alerting for user-defined projects
10.5.2. Creating alerting rules for user-defined projects
10.5.3. Reducing latency for alerting rules that do not query platform metrics
10.5.4. Accessing alerting rules for user-defined projects
CHAPTER 14. ACCESSING THIRD-PARTY MONITORING APIs ........................................ 141
14.1. ACCESSING THIRD-PARTY MONITORING WEB SERVICE APIs .......................... 141
14.2. QUERYING METRICS BY USING THE FEDERATION ENDPOINT FOR PROMETHEUS 141
14.3. ADDITIONAL RESOURCES .................................................................................. 143

CHAPTER 15. TROUBLESHOOTING MONITORING ISSUES ........................................ 144
15.1. INVESTIGATING WHY USER-DEFINED METRICS ARE UNAVAILABLE .................. 144
15.2. DETERMINING WHY PROMETHEUS IS CONSUMING A LOT OF DISK SPACE .......... 147

CHAPTER 16. CONFIG MAP REFERENCE FOR THE CLUSTER MONITORING OPERATOR ......... 149
16.1. CLUSTER MONITORING OPERATOR CONFIGURATION REFERENCE ..................... 149
16.2. ADDITIONAL ALERTMANAGERCONFIG .............................................................. 149
  16.2.1. Description ....................................................................................................... 149
  16.2.2. Required ......................................................................................................... 149
16.3. ALERTMANAGERMAINCONFIG ......................................................................... 150
  16.3.1. Description ..................................................................................................... 150
16.4. ALERTMANAGERUSERWORKLOADCONFIG ...................................................... 151
  16.4.1. Description ..................................................................................................... 151
16.5. CLUSTERMONITORINGCONFIGURATION .......................................................... 152
  16.5.1. Description ..................................................................................................... 152
16.6. DEDICATEDSERVICEMONITORS ...................................................................... 153
  16.6.1. Description ..................................................................................................... 153
16.7. K8SPROMETHEUSADAPTER .............................................................................. 154
  16.7.1. Description ..................................................................................................... 154
16.8. KUBESTATEMETRICSCONFIG ......................................................................... 154
  16.8.1. Description ..................................................................................................... 155
16.9. OPENSIFTSTATEMETRICSCONFIG ................................................................. 155
  16.9.1. Description ..................................................................................................... 155
16.10. PROMETHEUSK8SCONFIG .............................................................................. 155
    16.10.1. Description ................................................................................................. 155
16.11. PROMETHEUSOPERATORCONFIG ................................................................. 157
    16.11.1. Description ................................................................................................. 157
16.12. PROMETHEUSRESTRICTEDCONFIG .............................................................. 158
    16.12.1. Description ................................................................................................. 158
16.13. REMOTEWritesPEC ......................................................................................... 161
    16.13.1. Description ................................................................................................. 161
    16.13.2. Required .................................................................................................... 161
16.14. TELEMETERCLIENTCONFIG ........................................................................... 162
    16.14.1. Description ................................................................................................. 162
    16.14.2. Required .................................................................................................... 162
16.15. THANOSQUERIERCONFIG .............................................................................. 163
    16.15.1. Description ................................................................................................. 163
16.16. THANOSRULERCONFIG ................................................................................. 163
    16.16.1. Description ................................................................................................. 163
16.17. TLSCONFIG ....................................................................................................... 164
    16.17.1. Description ................................................................................................. 164
    16.17.2. Required .................................................................................................... 164
16.18. USERWORKLOADCONFIGURATION .............................................................. 165
16.18.1. Description 165
CHAPTER 1. MONITORING OVERVIEW

1.1. ABOUT OPENSHIFT CONTAINER PLATFORM MONITORING

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

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

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

In the **Observe** section of OpenShift Container Platform web console, you can access and manage monitoring features such as metrics, alerts, monitoring dashboards, and metrics targets.

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

1.2. UNDERSTANDING THE MONITORING STACK

The OpenShift Container Platform 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 an OpenShift Container Platform installation. This provides monitoring for core OpenShift Container Platform components including Kubernetes services. The default monitoring stack also enables remote health monitoring for clusters. These components are illustrated in the **Installed by default** section in the following diagram.

- **Components for monitoring user-defined projects** After optionally enabling monitoring for user-defined projects, additional monitoring components are installed in the `openshift-user-workload-monitoring` project. This provides monitoring for user-defined projects. These components are illustrated in the **User** section in the following diagram.
1.2.1. Default monitoring components

By default, the OpenShift Container Platform 4.11 monitoring stack includes these components:

Table 1.1. Default monitoring stack components

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster Monitoring Operator</td>
<td>The Cluster Monitoring Operator (CMO) is a central component of the monitoring stack. It deploys and manages Prometheus instances, the Thanos Querier, the Telemeter Client, and metrics targets and ensures that they are up to date. The CMO is deployed by the Cluster Version Operator (CVO).</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Prometheus Operator</td>
<td>The Prometheus Operator (PO) in the openshift-monitoring project creates, configures, and manages platform Prometheus instances and Alertmanager instances. It also automatically generates monitoring target configurations based on Kubernetes label queries.</td>
</tr>
<tr>
<td>Prometheus</td>
<td>Prometheus is the monitoring system on which the OpenShift Container Platform monitoring stack is based. Prometheus is a time-series database and a rule evaluation engine for metrics. Prometheus sends alerts to Alertmanager for processing.</td>
</tr>
<tr>
<td>Prometheus Adapter</td>
<td>The Prometheus Adapter (PA in the preceding diagram) translates Kubernetes node and pod queries for use in Prometheus. The resource metrics that are translated include CPU and memory utilization metrics. The Prometheus Adapter exposes the cluster resource metrics API for horizontal pod autoscaling. The Prometheus Adapter is also used by the <code>oc adm top nodes</code> and <code>oc adm top pods</code> commands.</td>
</tr>
<tr>
<td>Alertmanager</td>
<td>The Alertmanager service handles alerts received from Prometheus. Alertmanager is also responsible for sending the alerts to external notification systems.</td>
</tr>
<tr>
<td>kube-state-metrics agent</td>
<td>The kube-state-metrics exporter agent (KSM in the preceding diagram) converts Kubernetes objects to metrics that Prometheus can use.</td>
</tr>
<tr>
<td>openshift-state-metrics agent</td>
<td>The openshift-state-metrics exporter (OSM in the preceding diagram) expands upon kube-state-metrics by adding metrics for OpenShift Container Platform-specific resources.</td>
</tr>
<tr>
<td>node-exporter agent</td>
<td>The node-exporter agent (NE in the preceding diagram) collects metrics about every node in a cluster. The node-exporter agent is deployed on every node.</td>
</tr>
<tr>
<td>Thanos Querier</td>
<td>The Thanos Querier aggregates and optionally deduplicates core OpenShift Container Platform metrics and metrics for user-defined projects under a single, multi-tenant interface.</td>
</tr>
<tr>
<td>Telemeter Client</td>
<td>The Telemeter Client sends a subsection of the data from platform Prometheus instances to Red Hat to facilitate Remote Health Monitoring for clusters.</td>
</tr>
</tbody>
</table>
All of the components in the monitoring stack are monitored by the stack and are automatically updated when OpenShift Container Platform is updated.

NOTE

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

1.2.2. Default monitoring targets

In addition to the components of the stack itself, the default monitoring stack monitors:

- CoreDNS
- Elasticsearch (if Logging is installed)
- etcd
- Fluentd (if Logging is installed)
- HAProxy
- Image registry
- Kubelets
- Kubernetes API server
- Kubernetes controller manager
- Kubernetes scheduler
- Metering (if Metering is installed)
- OpenShift API server
- OpenShift Controller Manager
- Operator Lifecycle Manager (OLM)

NOTE

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

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

1.2.3. Components for monitoring user-defined projects
OpenShift Container Platform 4.11 includes an optional enhancement to the monitoring stack that enables you to monitor services and pods in user-defined projects. This feature includes the following components:

Table 1.2. Components for monitoring user-defined projects

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prometheus Operator</td>
<td>The Prometheus Operator (PO) in the <code>openshift-user-workload-monitoring</code> project creates, configures, and manages Prometheus and Thanos Ruler instances in the same project.</td>
</tr>
<tr>
<td>Prometheus</td>
<td>Prometheus is the monitoring system through which monitoring is provided for user-defined projects. Prometheus sends alerts to Alertmanager for processing.</td>
</tr>
<tr>
<td>Thanos Ruler</td>
<td>The Thanos Ruler is a rule evaluation engine for Prometheus that is deployed as a separate process. In OpenShift Container Platform 4.11, Thanos Ruler provides rule and alerting evaluation for the monitoring of user-defined projects.</td>
</tr>
<tr>
<td>Alertmanager</td>
<td>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.</td>
</tr>
</tbody>
</table>

NOTE

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

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

1.2.4. Monitoring targets for user-defined projects

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

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

1.3. GLOSSARY OF COMMON TERMS FOR OPENSHIFT CONTAINER PLATFORM MONITORING

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

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

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

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

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

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

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

**custom resource (CR)**
A CR is an extension of the Kubernetes API. You can create custom resources.

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

**Fluentd**
Fluentd gathers logs from nodes and feeds them to Elasticsearch.

**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.

**Metering**
Metering is a general purpose data analysis tool that enables you to write reports to process data from different data sources.

**node**
A worker machine in the OpenShift Container Platform cluster. A node is either a virtual machine (VM) or a physical machine.

**Operator**

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

**Operator Lifecycle Manager (OLM)**

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

**Persistent storage**

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

**Persistent volume claim (PVC)**

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

**pod**

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

**Prometheus**

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

**Prometheus 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**

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

**Thanos Ruler**

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

**web console**

A user interface (UI) to manage OpenShift Container Platform.

### 1.4. ADDITIONAL RESOURCES
● About remote health monitoring
● Granting users permission to monitor user-defined projects
● Configuring TLS security profiles

1.5. NEXT STEPS
● Configuring the monitoring stack
CHAPTER 2. CONFIGURING THE MONITORING STACK

The OpenShift Container Platform 4 installation program provides only a low number of configuration options before installation. Configuring most OpenShift Container Platform framework components, including the cluster monitoring stack, happens post-installation.

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

2.1. PREREQUISITES

- The monitoring stack imposes additional resource requirements. Consult the computing resources recommendations in Scaling the Cluster Monitoring Operator and verify that you have sufficient resources.

2.2. MAINTENANCE AND SUPPORT FOR MONITORING

The supported way of configuring OpenShift Container Platform Monitoring is by configuring it using the options described in this document. **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 this section, your changes will disappear because the cluster-monitoring-operator reconciles any differences. The Operator resets everything to the defined state by default and by design.

2.2.1. Support considerations for monitoring

The following modifications are explicitly not supported:

- **Creating additional ServiceMonitor, PodMonitor, and PrometheusRule objects in the openshift-* and kube-* projects.**

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

**NOTE**

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

- **Modifying resources of the stack.** The OpenShift Container Platform monitoring stack ensures its resources are always in the state it expects them to be. If they are modified, the stack will reset them.

- **Deploying user-defined workloads to openshift-* and kube-* projects.** These projects are reserved for Red Hat provided components and they should not be used for user-defined workloads.
• Installing custom Prometheus instances on OpenShift Container Platform. A custom instance is a Prometheus custom resource (CR) managed by the Prometheus Operator.

• Enabling symptom based monitoring by using the `Probe` custom resource definition (CRD) in Prometheus Operator.

**NOTE**

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

2.2.2. Support policy for monitoring Operators

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

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

Overriding the Cluster Version Operator

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

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

**WARNING**

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

2.3. PREPARING TO CONFIGURE THE MONITORING STACK

You can configure the monitoring stack by creating and updating monitoring config maps.

2.3.1. Creating a cluster monitoring config map

To configure core OpenShift Container Platform monitoring components, you must create the `cluster-monitoring-config ConfigMap` object in the `openshift-monitoring` project.
When you save your changes to the `cluster-monitoring-config` ConfigMap object, some or all of the pods in the `openshift-monitoring` project might be redeployed. It can sometimes take a while for these components to redeploy.

### Prerequisites

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

### Procedure

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

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

2. If the ConfigMap object does not exist:

   a. Create the following YAML manifest. In this example the file is called `cluster-monitoring-config.yaml`:

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

   b. Apply the configuration to create the ConfigMap object:

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

### 2.3.2. Creating a user-defined workload monitoring config map

To configure the components that monitor user-defined projects, you must create the `user-workload-monitoring-config` ConfigMap object in the `openshift-user-workload-monitoring` project.

When you save your changes to the `user-workload-monitoring-config` ConfigMap object, some or all of the pods in the `openshift-user-workload-monitoring` project might be redeployed. It can sometimes take a while for these components to redeploy. You can create and configure the config map before you first enable monitoring for user-defined projects, to prevent having to redeploy the pods often.

### Prerequisites

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

1. Check whether the `user-workload-monitoring-config ConfigMap` object exists:
   
   ```bash
   $ oc -n openshift-user-workload-monitoring get configmap user-workload-monitoring-config
   ```

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

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

   b. Apply the configuration to create the `ConfigMap` object:

   ```bash
   $ oc apply -f user-workload-monitoring-config.yaml
   ```

   **NOTE**
   
   Configurations applied to the `user-workload-monitoring-config ConfigMap` object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.

Additional resources

- Enabling monitoring for user-defined projects

### 2.4. CONFIGURING THE MONITORING STACK

In OpenShift Container Platform 4.11, you can configure the monitoring stack using the `cluster-monitoring-config` or `user-workload-monitoring-config-config Map` objects. Config maps configure the Cluster Monitoring Operator (CMO), which in turn configures the components of the stack.

**Prerequisites**

- **If you are configuring core OpenShift Container Platform monitoring components**
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config ConfigMap` object.

- **If you are configuring components that monitor user-defined projects**
  - You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
  - You have created the `user-workload-monitoring-config ConfigMap` object.
• You have installed the OpenShift CLI (oc).

Procedure

1. Edit the ConfigMap object.

   • To configure core OpenShift Container Platform monitoring components
     a. Edit the cluster-monitoring-config ConfigMap object in the openshift-monitoring
        project:

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

     b. Add your configuration under data/config.yaml as a key-value pair

        `<component_name>: <component_configuration>`

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

        The following example ConfigMap object configures a persistent volume claim (PVC)
        for Prometheus. This relates to the Prometheus instance that monitors core OpenShift
        Container Platform components only:

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

        1 Defines the Prometheus component and the subsequent lines define its
        configuration.

   • To configure components that monitor user-defined projects
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>`:

```yaml
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:

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

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.

NOTE

Configurations applied to the user-workload-monitoring-config ConfigMap object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.

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 Preparing to configure the monitoring stack for steps to create monitoring config maps
- Enabling monitoring for user-defined projects

2.5. CONFIGURABLE MONITORING COMPONENTS

This table shows the monitoring components you can configure and the keys used to specify the components in the cluster-monitoring-config and user-workload-monitoring-config ConfigMap objects:

Table 2.1. Configurable monitoring components

<table>
<thead>
<tr>
<th>Component</th>
<th>cluster-monitoring-config config map key</th>
<th>user-workload-monitoring-config config map key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prometheus Operator</td>
<td>prometheusOperator</td>
<td>prometheusOperator</td>
</tr>
<tr>
<td>Prometheus</td>
<td>prometheusK8s</td>
<td>prometheus</td>
</tr>
<tr>
<td>Alertmanager</td>
<td>alertmanagerMain</td>
<td>alertmanager</td>
</tr>
<tr>
<td>kube-state-metrics</td>
<td>kubeStateMetrics</td>
<td></td>
</tr>
<tr>
<td>openshift-state-metrics</td>
<td>openshiftStateMetrics</td>
<td></td>
</tr>
</tbody>
</table>
### Component

<table>
<thead>
<tr>
<th>Component</th>
<th>cluster-monitoring-config config map key</th>
<th>user-workload-monitoring-config config map key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telemeter Client</td>
<td>telemeterClient</td>
<td></td>
</tr>
<tr>
<td>Prometheus Adapter</td>
<td>k8sPrometheusAdapter</td>
<td></td>
</tr>
<tr>
<td>Thanos Querier</td>
<td>thanosQuerier</td>
<td></td>
</tr>
<tr>
<td>Thanos Ruler</td>
<td>thanosRuler</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE**

The Prometheus key is called `prometheusK8s` in the `cluster-monitoring-config` ConfigMap object and `prometheus` in the `user-workload-monitoring-config` ConfigMap object.

### 2.6. MOVING MONITORING COMPONENTS TO DIFFERENT NODES

You can move any of the monitoring stack components to specific nodes.

**Prerequisites**

- **If you are configuring core OpenShift Container Platform monitoring components**
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config` ConfigMap object.

- **If you are configuring components that monitor user-defined projects**
  - You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
  - You have created the `user-workload-monitoring-config` ConfigMap object.
  - You have installed the OpenShift CLI (oc).

**Procedure**

1. Edit the ConfigMap object:
   - To move a component that monitors core OpenShift Container Platform projects
     a. Edit the `cluster-monitoring-config` ConfigMap object in the `openshift-monitoring` project:

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

     b. Specify the `nodeSelector` constraint for the component under `data/config.yaml`:
Substitute `<component>` accordingly and substitute `<node_key>: <node_value>` with the map of key-value pairs that specifies a group of destination nodes. Often, only a single key-value pair is used.

The component can only run on nodes that have each of the specified key-value pairs as labels. The nodes can have additional labels as well.

**IMPORTANT**

Many of the monitoring components are deployed by using multiple pods across different nodes in the cluster to maintain high availability. When moving monitoring components to labeled nodes, ensure that enough matching nodes are available to maintain resilience for the component. If only one label is specified, ensure that enough nodes contain that label to distribute all of the pods for the component across separate nodes. Alternatively, you can specify multiple labels each relating to individual nodes.

**NOTE**

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

For example, to move monitoring components for core OpenShift Container Platform projects to specific nodes that are labeled `nodename: controlplane1`, `nodename: worker1`, `nodename: worker2`, and `nodename: worker2`, use:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    <component>:
      nodeSelector:
        <node_key>: <node_value>
        <node_key>: <node_value>
        ...
```

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusOperator:
      nodeSelector:
        nodename: controlplane1
    prometheusK8s:
      nodeSelector:
        nodename: worker1
```
To move a component that monitors user-defined projects:

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 `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>
  nodeSelector:
    <node_key>: <node_value>
    <node_key>: <node_value>
  ...
```

Substitute `<component>` accordingly and substitute `<node_key>: <node_value>` with the map of key-value pairs that specifies the destination nodes. Often, only a single key-value pair is used.

The component can only run on nodes that have each of the specified key-value pairs as labels. The nodes can have additional labels as well.
IMPORTANT

Many of the monitoring components are deployed by using multiple pods across different nodes in the cluster to maintain high availability. When moving monitoring components to labeled nodes, ensure that enough matching nodes are available to maintain resilience for the component. If only one label is specified, ensure that enough nodes contain that label to distribute all of the pods for the component across separate nodes. Alternatively, you can specify multiple labels each relating to individual nodes.

NOTE

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

For example, to move monitoring components for user-defined projects to specific worker nodes labeled nodename: worker1, nodename: worker2, and nodename: worker2, use:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheusOperator: 
      nodeSelector: 
        nodename: worker1
    prometheus: 
      nodeSelector: 
        nodename: worker1
        nodename: worker2
    thanosRuler: 
      nodeSelector: 
        nodename: worker1
        nodename: worker2

2. Save the file to apply the changes. The components affected by the new configuration are moved to the new nodes automatically.

NOTE

Configurations applied to the user-workload-monitoring-config ConfigMap object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.
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 Preparing to configure the monitoring stack for steps to create monitoring config maps
- Enabling monitoring for user-defined projects
- Understanding how to update labels on nodes
- Placing pods on specific nodes using node selectors
- See the Kubernetes documentation for details on the nodeSelector constraint

2.7. ASSIGNING TOLERATIONS TO MONITORING COMPONENTS

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

Prerequisites

- If you are configuring core OpenShift Container Platform monitoring components
  - You have access to the cluster as a user with the cluster-admin role.
  - You have created the cluster-monitoring-config ConfigMap object.
- If you are configuring components that monitor user-defined projects
  - You have access to the cluster as a user with the cluster-admin role, or as a user with the user-workload-monitoring-config-edit role in the openshift-user-workload-monitoring project.
  - You have created the user-workload-monitoring-config ConfigMap object.
- You have installed the OpenShift CLI (oc).

Procedure

1. Edit the ConfigMap object:
   - To assign tolerations to a component that monitors core OpenShift Container Platform projects:
     a. Edit the cluster-monitoring-config ConfigMap object in the openshift-monitoring project:

```bash
$ oc -n openshift-monitoring edit configmap cluster-monitoring-config
```
b. Specify **tolerations** for the component:

```yaml
apiVersion: v1
type: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml:
    <component>:
      tolerations:
        <toleration_specification>
```

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

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

```yaml
apiVersion: v1
type: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml:
    alertmanagerMain:
      tolerations:
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoSchedule"
```

- **To assign tolerations to a component that monitors user-defined projects**

  a. Edit the `user-workload-monitoring-config` ConfigMap object in the `openshift-user-workload-monitoring` project:

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

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

```yaml
apiVersion: v1
type: 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:

```yaml
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.

**NOTE**

Configurations applied to the `user-workload-monitoring-config ConfigMap` object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.

**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 [Preparing to configure the monitoring stack](#) for steps to create monitoring config maps
- [Enabling monitoring for user-defined projects](#)
- See the [OpenShift Container Platform documentation](#) on taints and tolerations
- See the [Kubernetes documentation](#) on taints and tolerations

### 2.8. SETTING THE BODY SIZE LIMIT FOR METRICS SCRAPING

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

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

**NOTE**

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

**Prerequisites**

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

**Procedure**

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

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

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

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

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

3. Save the file to apply the changes automatically.
WARNING

When you save changes to a `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

- Prometheus scrape configuration documentation

2.9. 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.

2.9.1. Persistent storage prerequisites

- Dedicate sufficient local persistent storage to ensure that the disk does not become full. How much storage you need depends on the number of pods.

- Verify that you have a persistent volume (PV) ready to be claimed by the persistent volume claim (PVC), one PV for each replica. Because Prometheus and Alertmanager both have two replicas, you need four PVs to support the entire monitoring stack. The PVs are available from the Local Storage Operator, but not if you have enabled dynamically provisioned storage.

- Use the block type of storage.

NOTE

If you use a local volume for persistent storage, do not use a raw block volume, which is described with `volumeMode: block` in the `LocalVolume` object. Prometheus cannot use raw block volumes.

2.9.2. Configuring a local persistent volume claim

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

Prerequisites

- If you are configuring core OpenShift Container Platform monitoring components
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config ConfigMap` object.

- If you are configuring components that monitor user-defined projects
You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.

- You have created the `user-workload-monitoring-config` ConfigMap object.

- You have installed the OpenShift CLI (`oc`).

**Procedure**

1. Edit the ConfigMap object:
   - To configure a PVC for a component that monitors core OpenShift Container Platform projects:
     a. Edit the `cluster-monitoring-config` ConfigMap object in the `openshift-monitoring` project:

     ```bash
     $ oc -n openshift-monitoring edit configmap cluster-monitoring-config
     ```
     
     b. Add your PVC configuration for the component under `data/config.yaml`:

     ```yaml
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: cluster-monitoring-config
       namespace: openshift-monitoring
     data:
       config.yaml:
         <component>:
           volumeClaimTemplate:
             spec:
               storageClassName: <storage_class>
               resources:
                 requests:
                   storage: <amount_of_storage>
     ```

     See the [Kubernetes documentation on PersistentVolumeClaims](https://kubernetes.io/docs/concepts/storage/persistent-volumes/) for information on how to specify `volumeClaimTemplate`.

     The following example configures a PVC that claims local persistent storage for the Prometheus instance that monitors core OpenShift Container Platform components:

     ```yaml
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: cluster-monitoring-config
       namespace: openshift-monitoring
     data:
       config.yaml:
         prometheusK8s:
           volumeClaimTemplate:
             spec:
               storageClassName: local-storage
     ```
In the above example, the storage class created by the Local Storage Operator is called `local-storage`.

The following example configures a PVC that claims local persistent storage for Alertmanager:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
config.yaml: |
alertmanagerMain:
  volumeClaimTemplate:
    spec:
      storageClassName: local-storage
      resources:
        requests:
          storage: 10Gi
```

- To configure a PVC for a component that monitors user-defined projects

  a. Edit the `user-workload-monitoring-config` ConfigMap object in the `openshift-user-workload-monitoring` project:

  ```bash
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  
  $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
  
  b. Add your PVC configuration for the component under `data/config.yaml`:

  ```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 local persistent storage for the Prometheus instance that monitors user-defined projects:

```yaml
resources:
  requests:
    storage: 40Gi
```

In the above example, the storage class created by the Local Storage Operator is called `local-storage`.

The following example configures a PVC that claims local persistent storage for Alertmanager:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
config.yaml: |
alertmanagerMain:
  volumeClaimTemplate:
    spec:
      storageClassName: local-storage
      resources:
        requests:
          storage: 10Gi
```
In the above example, the storage class created by the Local Storage Operator is called `local-storage`.

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

```yaml
apiVersion: v1
classKind: ConfigMap
data:
  prometheus:
    volumeClaimTemplate:
      spec:
        storageClassName: local-storage
        resources:
          requests:
            storage: 40Gi
```

In the above example, the storage class created by the Local Storage Operator is called `local-storage`.

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

```yaml
apiVersion: v1
classKind: ConfigMap
data:
  thanosRuler:
    volumeClaimTemplate:
      spec:
        storageClassName: local-storage
        resources:
          requests:
            storage: 10Gi
```

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.

NOTE

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

NOTE

Configurations applied to the `user-workload-monitoring-config` ConfigMap object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.
2.9.3. Resizing a persistent storage volume

OpenShift Container Platform does not support resizing an existing persistent storage volume used by **StatefulSet** resources, even if the underlying **StorageClass** resource used supports persistent volume sizing. Therefore, even if you update the **storage** field for an existing persistent volume claim (PVC) with a larger size, this setting will not be propagated to the associated persistent volume (PV).

However, resizing a PV is still possible by using a manual process. If you want to resize a PV for a monitoring component such as Prometheus, Thanos Ruler, or Alertmanager, you can update the appropriate config map in which the component is configured. Then, patch the PVC, and delete and orphan the pods. Orphaning the pods recreates the **StatefulSet** resource immediately and automatically updates the size of the volumes mounted in the pods with the new PVC settings. No service disruption occurs during this process.

**Prerequisites**

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

- **If you are configuring core OpenShift Container Platform monitoring components**
  - You have access to the cluster as a user with the **cluster-admin** role.
  - You have created the **cluster-monitoring-config ConfigMap** object.
  - You have configured at least one PVC for core OpenShift Container Platform monitoring components.

- **If you are configuring components that monitor user-defined projects**
  - You have access to the cluster as a user with the **cluster-admin** role, or as a user with the **user-workload-monitoring-config-edit** role in the **openshift-user-workload-monitoring** project.
  - You have created the **user-workload-monitoring-config ConfigMap** object.
  - You have configured at least one PVC for components that monitor user-defined projects.

**Procedure**

1. Edit the **ConfigMap** object:

   - **To resize a PVC for a component that monitors core OpenShift Container Platform projects:**
     a. Edit the **cluster-monitoring-config ConfigMap** object in the **openshift-monitoring** project:
b. Add a new storage size for the PVC configuration for the component under `data/config.yaml`:

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

1. Specify the core monitoring component.
2. Specify the storage class.
3. Specify the new size for the storage volume.

The following example configures a PVC that sets the local persistent storage to 100 gigabytes for the Prometheus instance that monitors core OpenShift Container Platform components:

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

The following example configures a PVC that sets the local persistent storage for Alertmanager to 40 gigabytes:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
```
To resize a PVC for a component that monitors user-defined projects:

**NOTE**

You can resize the volumes for the Thanos Ruler and Prometheus instances that monitor user-defined projects.

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

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

b. Update the PVC configuration for the monitoring component under `data/config.yaml`:

```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>
```

1. Specify the core monitoring component.
2. Specify the storage class.
3. Specify the new size for the storage volume.

The following example configures the PVC size to 100 gigabytes for the Prometheus instance that monitors user-defined projects:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
```
The following example sets the PVC size to 20 gigabytes for Thanos Ruler:

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

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

```yaml
apiVersion: v1
class: ConfigMap
metadata:
  name: user-workload-monitoring-config
namespace: openshift-user-workload-monitoring
data:
  config.yaml:
    prometheus:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 100Gi
```

**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 restart automatically.

**WARNING**

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

3. Manually patch every PVC with the updated storage request. The following example resizes the storage size for the Prometheus component in the `openshift-monitoring` namespace to 100Gi:

```bash
$ for p in $(oc -n openshift-monitoring get pvc -l app.kubernetes.io/name=prometheus -o jsonpath='{range .items[*]}{.metadata.name} {end}'); do \
oc -n openshift-monitoring patch pvc/$p --patch '{"spec": {"resources": {"requests":
```
4. Delete the underlying StatefulSet with the `--cascade=orphan` parameter:

```
$ oc delete statefulset -l app.kubernetes.io/name=prometheus --cascade=orphan
```

### 2.9.4. Modifying the retention time and size for Prometheus metrics data

By default, Prometheus automatically retains metrics data for 15 days. You can modify the retention time to change how soon data is deleted by specifying a time value in the `retention` field. You can also configure the maximum amount of disk space the retained metrics data uses by specifying a size value in the `retentionSize` field. 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, and 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.

### Prerequisites

- If you are configuring core OpenShift Container Platform monitoring components
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config ConfigMap` object.

- If you are configuring components that monitor user-defined projects
  - A cluster administrator has enabled monitoring for user-defined projects.
You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.

You have created the `user-workload-monitoring-config` ConfigMap object.

You have installed the OpenShift CLI (`oc`).

**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.

### Procedure

1. Edit the **ConfigMap** object:

   - To modify the retention time and size for the Prometheus instance that monitors core OpenShift Container Platform projects:

     a. Edit the `cluster-monitoring-config` ConfigMap object in the `openshift-monitoring` project:

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

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

        ```yaml
        apiVersion: v1
        kind: ConfigMap
        metadata:
          name: cluster-monitoring-config
          namespace: openshift-monitoring
        data:
          config.yaml:
            prometheusK8s:
              retention: <time_specification> ¹
              retentionSize: <size_specification> ²
        ```

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

   ² The retention size: a number directly followed by **B** (bytes), **KB** (kilobytes), **MB** (megabytes), **GB** (gigabytes), **TB** (terabytes), **PB** (petabytes), and **EB** (exabytes).

   The following example sets the retention time to 24 hours and the retention size to 10 gigabytes for the Prometheus instance that monitors core OpenShift Container Platform components:
To modify the retention time and size for the Prometheus instance that monitors user-defined projects:

a. Edit the `user-workload-monitoring-config` ConfigMap object in the `openshift-user-workload-monitoring` project:

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

b. Add the retention time and size configuration under `data/config.yaml`:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml:
    prometheusK8s:
      retention: 24h
      retentionSize: 10GB
```

- 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`.

- 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:

```yaml
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.

2.9.5. 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 installed the OpenShift CLI (oc).
- A cluster administrator has enabled monitoring for user-defined projects.
- You have access to the cluster as a user with the `cluster-admin` role or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
- You have created the `user-workload-monitoring-config` ConfigMap object.

**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.

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`:

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

   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`. 

---

41
The following example sets the retention time to 10 days for Thanos Ruler data:

```yaml
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.

Additional resources

- Creating a cluster monitoring config map
- Prometheus database storage requirements
- Recommended configurable storage technology
- Understanding persistent storage
- Optimizing storage
- Configure local persistent storage
- Enabling monitoring for user-defined projects

2.10. 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

- **If you are configuring core OpenShift Container Platform monitoring components:**
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config` ConfigMap object.

- **If you are configuring components that monitor user-defined projects:**
  - You have access to the cluster as a user with the `cluster-admin` role or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
  - You have created the `user-workload-monitoring-config` ConfigMap object.

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

You have set up authentication credentials in a Secret object for the remote write endpoint. You must create the secret in the same namespace as the Prometheus object for which you configure remote write: the openshift-monitoring namespace for default platform monitoring or the openshift-user-workload-monitoring namespace for user workload monitoring.

**CAUTION**

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

**Procedure**

Follow these steps to configure remote write for default platform monitoring in the cluster-monitoring-config config map in the openshift-monitoring namespace.

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

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

2. Add a remoteWrite: section under `data/config.yaml/prometheusK8s`.

3. Add an endpoint URL and authentication credentials in this section:

   ```yaml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: cluster-monitoring-config
     namespace: openshift-monitoring
   data:
     config.yaml: |
     prometheusK8s:
       remoteWrite:
         - url: "https://remote-write-endpoint.example.com"
           <endpoint_authentication_credentials>
   ```

   **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.
4. Add write relabel configuration values after the authentication credentials:

```yaml
apiVersion: v1
type: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://remote-write-endpoint.example.com"
          endpointAuthenticationCredentials:
          writeRelabelConfigs:
            - sourceLabels: [__name__]
              regex: 'my_metric'
              action: keep
```

The write relabel configuration settings.

For `<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`:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://remote-write-endpoint.example.com"
          endpointAuthenticationCredentials:
          writeRelabelConfigs:
            - sourceLabels: [__name__]
              regex: 'my_metric'
              action: keep
```

See the [Prometheus relabel_config documentation](https://prometheus.io/docs/prometheus/latest/configuration/configuration/#write_relabel_configs) for information about write relabel configuration options.

5. Save the file to apply the changes to the ConfigMap object. The pods affected by the new configuration restart automatically.

**NOTE**

Configurations applied to the `user-workload-monitoring-config` ConfigMap object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.
2.10.1. 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.

<table>
<thead>
<tr>
<th>Authentication method</th>
<th>Config map field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Signature Version 4</td>
<td>sigv4</td>
<td>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.</td>
</tr>
<tr>
<td>basic authentication</td>
<td>basicAuth</td>
<td>Basic authentication sets the authorization header on every remote write request with the configured username and password.</td>
</tr>
<tr>
<td>authorization</td>
<td>authorization</td>
<td>Authorization sets the Authorization header on every remote write request using the configured token.</td>
</tr>
<tr>
<td>OAuth 2.0</td>
<td>oauth2</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

### 2.10.1.1. Config map location for authentication settings

The following shows the location of the authentication configuration in the ConfigMap object for default platform monitoring.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://remote-write-endpoint.example.com" 1
          <endpoint_authentication_details> 2
```

1. The URL of the remote write endpoint.
2. The required configuration details for the authentication method for the endpoint. Currently supported authentication methods are Amazon Web Services (AWS) Signature Version 4, authorization, basic authentication, OAuth 2.0, and TLS client.

**NOTE**

If you configure remote write for the Prometheus instance that monitors user-defined projects, edit the `user-workload-monitoring-config` config map in the `openshift-user-workload-monitoring` namespace. Note that the Prometheus config map component is called `prometheus` in the `user-workload-monitoring-config ConfigMap` object and not `prometheusK8s`, as it is in the `cluster-monitoring-config ConfigMap` object.

### 2.10.1.2. Example remote write authentication settings

The following samples show different authentication settings you can use to connect to a remote write endpoint. Each sample also shows how to configure a corresponding Secret object that contains authentication credentials and other relevant settings. Each sample configures authentication for use with default platform monitoring in the `openshift-monitoring` namespace.
Sample YAML for AWS Signature Version 4 authentication

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

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

① The AWS API access key.

② The AWS API secret key.

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

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://authorization.example.com/api/write"
          sigv4:
            region: <AWS_region> ①
            accessKey:
              name: sigv4-credentials ②
              key: accessKey ③
            secretKey:
              name: sigv4-credentials ④
              key: secretKey ⑤
            profile: <AWS_profile_name> ⑥
            roleArn: <AWS_role_arn> ⑦
```

① The AWS region.

② 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.

④ The name of the **Secret** object containing the AWS API secret credentials.

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.
Sample YAML for basic authentication

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

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: rw-basic-auth
  namespace: openshift-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-monitoring` namespace. It assumes that you have already set up authentication credentials for the endpoint.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://basicauth.example.com/api/write"
          basicAuth:
            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.

Sample YAML for authentication with a bearer token using a `Secret` Object

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

```yaml
apiVersion: v1
```
The following shows sample bearer token config map settings that use a `Secret` object named `rw-bearer-auth` in the `openshift-monitoring` namespace:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml:
    enableUserWorkload: true
    prometheusK8s:
      remoteWrite:
        - url: "https://authorization.example.com/api/write"
          authorization:
            type: Bearer
            credentials:
              name: rw-bearer-auth
              key: token
```

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.

Sample YAML for OAuth 2.0 authentication

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

```
apiVersion: v1
kind: Secret
metadata:
  name: oauth2-credentials
  namespace: openshift-monitoring
stringData:
  id: <oauth2_id>
  secret: <oauth2_secret>
  token: <oauth2_authentication_token>
type: Opaque
```

1. The Oauth 2.0 ID.
The OAuth 2.0 secret.

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-monitoring** namespace:

```yaml
apiVersion: v1
class: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
  prometheusK8s:
    remoteWrite:
      - url: "https://test.example.com/api/write"
    oauth2:
      clientId:
        secret:
          name: oauth2-credentials
          key: id
      clientSecret:
        name: oauth2-credentials
        key: secret
      tokenUrl: https://example.com/oauth2/token
      scopes: |
        - <scope_1>
        - <scope_2>
      endpointParams:
        param1: <parameter_1>
        param2: <parameter_2>
```

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

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

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

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

**Sample YAML for TLS client authentication**

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

```yaml
apiVersion: v1
class: Secret
metadata:
  name: mtls-bundle
```

---

50
The following sample shows a `tlsConfig` remote write authentication configuration that uses a TLS `Secret` object named `mtls-bundle`.

```yaml
apiVersion: v1
category: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      remoteWrite:
        - url: "https://remote-write-endpoint.example.com"
          tlsConfig:
            ca:
              secret:
                name: mts-bundle
                key: ca.crt
            cert:
                secret:
                    name: mts-bundle
                    key: client.crt
            keySecret:
                name: mts-bundle
                key: client.key
```

1. 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.
3. The key in the specified `Secret` object that contains the client certificate for the endpoint.
4. 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.

• See Understanding secrets for steps to create and configure Secret objects in OpenShift Container Platform.

• See the Prometheus REST API reference for remote write for information about additional optional fields.

2.11. ADDING CLUSTER ID LABELS TO METRICS

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

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

Creating and using cluster ID labels involves three general steps:

• Configuring the write relabel settings for remote write storage.

• Adding cluster ID labels to the metrics.

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

2.11.1. Creating cluster ID labels for metrics

You can create cluster ID labels for metrics for default platform monitoring and for user workload monitoring.

For default platform monitoring, you add cluster ID labels for metrics in the `write_relabel` settings for remote write storage in the `cluster-monitoring-config` config map in the `openshift-monitoring` namespace.

For user workload monitoring, you edit the settings in the `user-workload-monitoring-config` config map in the `openshift-user-workload-monitoring` namespace.

Prerequisites

• You have installed the OpenShift CLI (`oc`).

• You have configured remote write storage.

• If you are configuring default platform monitoring components:
  • You have access to the cluster as a user with the `cluster-admin` role.
  • You have created the `cluster-monitoring-config ConfigMap` object.

• If you are configuring components that monitor user-defined projects:
You have access to the cluster as a user with the `cluster-admin` role or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.

You have created the `user-workload-monitoring-config` ConfigMap object.

**Procedure**

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

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

   **NOTE**

   If you configure cluster ID labels for metrics for the Prometheus instance that monitors user-defined projects, edit the `user-workload-monitoring-config` config map in the `openshift-user-workload-monitoring` namespace. Note that the Prometheus component is called `prometheus` in this config map and not `prometheusK8s`, which is the name used in the `cluster-monitoring-config` config map.

2. In the `writeRelabelConfigs` section under `data/config.yaml/prometheusK8s/remoteWrite`, add cluster ID relabel configuration values:

   ```yaml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: cluster-monitoring-config
     namespace: openshift-monitoring
   data:
     config.yaml: |
     prometheusK8s:
       remoteWrite:
       - url: "https://remote-write-endpoint.example.com"
         <endpoint_authentication_credentials>
         writeRelabelConfigs:
           - <relabel_config>  

   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 default platform monitoring:

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

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.

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

3. Save the file to apply the changes 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.

### 2.12. 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. Using many unbound attributes in labels can create exponentially more time series, which can impact Prometheus performance and available disk space.

Cluster administrators can use the following measures to control the impact of unbound metrics attributes in user-defined projects:
2.12.1. 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 `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
- You have enabled monitoring for user-defined projects.
- You have installed the OpenShift CLI (`oc`).

**Procedure**

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

   ```bash
   $ 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:

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

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

   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.

   **NOTE**

   Configurations applied to the `user-workload-monitoring-config ConfigMap` object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.

   **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.
2.12.2. Creating scrape sample alerts

You can create alerts that notify you when:

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

**Prerequisites**

- You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
- You have enabled monitoring for user-defined projects.
- You have created the `user-workload-monitoring-config` ConfigMap object.
- You have limited the number of samples that can be accepted per target scrape in user-defined projects, by using `enforcedSampleLimit`.
- You have installed the OpenShift CLI (`oc`).

**Procedure**

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

   ```yaml
   apiVersion: monitoring.coreos.com/v1
   kind: PrometheusRule
   metadata:
     labels:
       prometheus: k8s
       role: alert-rules
     name: monitoring-stack-alerts
     namespace: ns1
   spec:
     groups:
       - name: general.rules
         rules:
           - alert: TargetDown
             annotations:
               message: '{{ printf %.4g $value }}% of the {{ $labels.job }}/{{ $labels.service }} targets in {{ $labels.namespace }} namespace are down.'
             expr: 100 * (count(up == 0) BY (job, namespace, service)) / count(up) BY (job, namespace, service)) > 10
             for: 10m
             labels:
               severity: warning
           - alert: ApproachingEnforcedSamplesLimit
             annotations:
               message: '{{ $labels.container }} container of the {{ $labels.pod }} pod in the {{ $labels.namespace }} namespace consumes {{ $value | humanizePercentage }} of the samples limit budget.'
             expr: scrape_samples_scraped/50000 > 0.8
   ```

CHAPTER 2. CONFIGURING THE MONITORING STACK
for: 10m
labels:
  severity: warning

1. Defines the name of the alerting rule.
2. Specifies the user-defined project where the alerting rule will be deployed.
3. The TargetDown alert will fire if the target cannot be scraped or is not available for the for duration.
4. The message that will be output when the TargetDown alert fires.
5. The conditions for the TargetDown alert must be true for this duration before the alert is fired.
6. Defines the severity for the TargetDown alert.
7. The ApproachingEnforcedSamplesLimit alert will fire when the defined scrape sample threshold is reached or exceeded for the specified for duration.
8. The message that will be output when the ApproachingEnforcedSamplesLimit alert fires.
9. The threshold for the ApproachingEnforcedSamplesLimit alert. In this example the alert will fire when the number of samples per target scrape has exceeded 80% of the enforced sample limit of 50000. The for duration must also have passed before the alert will fire. The <number> in the expression scrape_samples_scraped/<number> > <threshold> must match the enforcedSampleLimit value defined in the user-workload-monitoring-config ConfigMap object.
10. The conditions for the ApproachingEnforcedSamplesLimit alert must be true for this duration before the alert is fired.
11. Defines the severity for the ApproachingEnforcedSamplesLimit alert.

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

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

Additional resources

- Creating a user-defined workload monitoring config map
- Enabling monitoring for user-defined projects
- See Determining why Prometheus is consuming a lot of disk space for steps to query which metrics have the highest number of scrape samples.
CHAPTER 3. CONFIGURING EXTERNAL ALERTMANAGER INSTANCES

The OpenShift Container Platform monitoring stack includes a local Alertmanager instance that routes alerts from Prometheus. You can add external Alertmanager instances by configuring the `cluster-monitoring-config` config map in either the `openshift-monitoring` project or the `user-workload-monitoring-config` project.

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 installed the OpenShift CLI (`oc`).

  - If you are configuring core OpenShift Container Platform monitoring components in the `openshift-monitoring` project:
    - You have access to the cluster as a user with the `cluster-admin` role.
    - You have created the `cluster-monitoring-config` config map.

  - If you are configuring components that monitor user-defined projects:
    - You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
    - You have created the `user-workload-monitoring-config` config map.

Procedure

1. Edit the `ConfigMap` object.

   - To configure additional Alertmanagers for routing alerts from core OpenShift Container Platform projects:
     a. Edit the `cluster-monitoring-config` config map in the `openshift-monitoring` project:

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

     b. Add an `additionalAlertmanagerConfigs` section under `data/config.yaml/prometheusK8s`.

     c. Add the configuration details for additional Alertmanagers in this section:

        ```yaml
        apiVersion: v1
        kind: ConfigMap
        metadata:
          name: cluster-monitoring-config
          namespace: openshift-monitoring
        data:
          config.yaml: |
        ```
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 a bearer token with client TLS authentication:

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

To configure additional Alertmanager instances for routing alerts from user-defined projects:

a. Edit the `user-workload-monitoring-config` config map in the `openshift-user-workload-monitoring` project:

```bash
$ 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:

```yaml
apiVersion: v1
```
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:

```yaml
apiVersion: v1
class: 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
```

**NOTE**

Configurations applied to the `user-workload-monitoring-config ConfigMap` object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.
2. Save the file to apply the changes to the ConfigMap object. The new component placement configuration is applied automatically.

### 3.1. ATTACHING ADDITIONAL LABELS TO YOUR TIME SERIES AND ALERTS

Using the external labels feature of Prometheus, you can attach custom labels to all time series and alerts leaving Prometheus.

**Prerequisites**
- **If you are configuring core OpenShift Container Platform monitoring components**
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config` ConfigMap object.
- **If you are configuring components that monitor user-defined projects**
  - You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
  - You have created the `user-workload-monitoring-config` ConfigMap object.
- You have installed the OpenShift CLI (`oc`).

**Procedure**

1. Edit the ConfigMap object:

   **To attach custom labels to all time series and alerts leaving the Prometheus instance that monitors core OpenShift Container Platform projects:**
   - Edit the `cluster-monitoring-config` ConfigMap object in the `openshift-monitoring` project:
     ```bash
     $ oc -n openshift-monitoring edit configmap cluster-monitoring-config
     ```
   - Define a map of labels you want to add for every metric under `data/config.yaml`:
     ```yaml
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: cluster-monitoring-config
     namespace: openshift-monitoring
     data:
       config.yaml:
         prometheusK8s: |
         externalLabels:
           <key>: <value>  
     ```

     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.

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

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      externalLabels:
        region: eu
        environment: prod
```

- To attach custom labels to all time series and alerts leaving the Prometheus instance that monitors user-defined projects:
  
  a. Edit the user-workload-monitoring-config ConfigMap object in the openshift-user-workload-monitoring project:

    ```bash
    $ 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:

    ```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.

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:

```
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.

NOTE
Configurations applied to the user-workload-monitoring-config ConfigMap object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.

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 Preparing to configure the monitoring stack for steps to create monitoring config maps.
Enabling monitoring for user-defined projects

3.2. SETTING LOG LEVELS FOR MONITORING COMPONENTS

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

The following log levels can be applied to the relevant component in the `cluster-monitoring-config` and `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

- **If you are setting a log level for Alertmanager, Prometheus Operator, Prometheus, or Thanos Querier in the openshift-monitoring project:**
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config` ConfigMap object.

- **If you are setting a log level for Prometheus Operator, Prometheus, or Thanos Ruler in the openshift-user-workload-monitoring project:**
  - You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.
  - You have created the `user-workload-monitoring-config` ConfigMap object.

- You have installed the OpenShift CLI (`oc`).

Procedure

1. Edit the **ConfigMap** object:
   - **To set a log level for a component in the openshift-monitoring project:**
     a. Edit the `cluster-monitoring-config` ConfigMap object in the openshift-monitoring project:

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

     b. Add `logLevel: <log_level>` for a component under `data/config.yaml`:

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

1. The monitoring stack component for which you are setting a log level. For default platform monitoring, available component values are `prometheusK8s`, `alertmanagerMain`, `prometheusOperator`, and `thanosQuerier`.

2. The log level to set for the component. The available values are `error`, `warn`, `info`, and `debug`. The default value is `info`.

- To set a log level for a component in the `openshift-user-workload-monitoring` project:
  a. Edit the `user-workload-monitoring-config ConfigMap` object in the `openshift-user-workload-monitoring` project:

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

  b. Add `logLevel: <log_level>` for a component under `data/config.yaml`:

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

    1. The monitoring stack component for which you are setting a log level. For user workload monitoring, available component values are `prometheus`, `prometheusOperator`, and `thanosRuler`.

    2. The log level to set for 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 restarts automatically when you apply the log-level change.

**NOTE**

Configurations applied to the `user-workload-monitoring-config ConfigMap` object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.
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
```

3.3. 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. You can do so for default platform monitoring and for user-defined workload monitoring.

**IMPORTANT**

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

**Prerequisites**

- You have installed the OpenShift CLI (`oc`).
- If you are enabling the query log file feature for Prometheus in the `openshift-monitoring` project:
  - You have access to the cluster as a user with the `cluster-admin` role.
  - You have created the `cluster-monitoring-config ConfigMap` object.
• If you are enabling the query log file feature for Prometheus in the `openshift-user-workload-monitoring` project:

  o You have access to the cluster as a user with the `cluster-admin` role, or as a user with the `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project.

  o You have created the `user-workload-monitoring-config` ConfigMap object.

Procedure

• To set the query log file for Prometheus in the `openshift-monitoring` project:

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

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

  2. Add `queryLogFile: <path>` for `prometheusK8s` under `data/config.yaml`:

     ```yaml
     apiVersion: v1
     kind: ConfigMap
     metadata:
       name: cluster-monitoring-config
       namespace: openshift-monitoring
     data:
       config.yaml: |
         prometheusK8s:
           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 sample command lists the status of pods in the `openshift-monitoring` project:

     $ oc -n openshift-monitoring get pods

  5. Read the query log:

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

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

To set the query log file for Prometheus in the 
openshift-user-workload-monitoring project:

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:

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

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

3. Save the file to apply the changes.

NOTE

Configurations applied to the user-workload-monitoring-config ConfigMap object are not activated unless a cluster administrator has enabled monitoring for user-defined projects.

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.

Additional resources
- See `Preparing to configure the monitoring stack` for steps to create monitoring config maps
- See `Enabling monitoring for user-defined projects` for steps to enable user-defined monitoring.

3.4. ENABLING QUERY LOGGING FOR THANOS QUERIER

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

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

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

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

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

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

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

   ```yaml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: cluster-monitoring-config
     namespace: openshift-monitoring
   data:
     config.yaml: |
     thanosQuerier:
       enableRequestLogging: <value>  
       logLevel: <value>  
   ```
Set the value to **true** to enable logging and **false** to disable logging. The default value is **false**.

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

3. Save the file to apply the changes.

### 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.

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

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

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

   ```bash
   $ token=`oc sa get-token prometheus-k8s -n openshift-monitoring`
   ```

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

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

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

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

#### Additional resources
- See [Preparing to configure the monitoring stack](#) for steps to create monitoring config maps.
CHAPTER 4. SETTING AUDIT LOG LEVELS FOR THE PROMETHEUS ADAPTER

In default platform monitoring, you can configure the audit log level for the Prometheus Adapter.

Prerequisites

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

Procedure

You can set an audit log level for the Prometheus Adapter in the default openshift-monitoring project:

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

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

2. Add profile: in the k8sPrometheusAdapter/audit section under data/config.yaml:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: cluster-monitoring-config
     namespace: openshift-monitoring
   data:
     config.yaml: |
     k8sPrometheusAdapter:
       audit:
         profile: <audit_log_level> 1
   ```
   
   1 The audit log level to apply to the Prometheus Adapter.

3. Set the audit log level by using one of the following values for the profile: parameter:

   - None: Do not log events.
   - Metadata: Log only the metadata for the request, such as user, timestamp, and so forth. Do not log the request text and the response text. Metadata is the default audit log level.
   - Request: Log only the metadata and the request text but not the response text. This option does not apply for non-resource requests.
   - RequestResponse: Log event metadata, request text, and response text. This option does not apply for non-resource requests.

4. Save the file to apply the changes. The pods for the Prometheus Adapter restart automatically when you apply the 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.

Verification

1. In the config map, under `k8sPrometheusAdapter/audit/profile`, set the log level to **Request** and save the file.

2. Confirm that the pods for the Prometheus Adapter are running. The following example lists the status of pods in the `openshift-monitoring` project:

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

3. Confirm that the audit log level and audit log file path are correctly configured:

   ```
   $ oc -n openshift-monitoring get deploy prometheus-adapter -o yaml
   ```

   **Example output**

   ```
   ...  
   - --audit-policy-file=/etc/audit/request-profile.yaml  
   - --audit-log-path=/var/log/adapter/audit.log
   ```

4. Confirm that the correct log level has been applied in the `prometheus-adapter` deployment in the `openshift-monitoring` project:

   ```
   $ oc -n openshift-monitoring exec deploy/prometheus-adapter -c prometheus-adapter -- cat /etc/audit/request-profile.yaml
   ```

   **Example output**

   ```
   "apiVersion": "audit.k8s.io/v1"  
   "kind": "Policy"  
   "metadata":  
   "name": "Request"  
   "omitStages":  
   - "RequestReceived"  
   "rules":  
   - "level": "Request"
   ```

**NOTE**

If you enter an unrecognized **profile** value for the Prometheus Adapter in the **ConfigMap** object, no changes are made to the Prometheus Adapter, and an error is logged by the Cluster Monitoring Operator.
5. Review the audit log for the Prometheus Adapter:

```bash
$ oc -n openshift-monitoring exec -c <prometheus_adapter_pod_name> -- cat /var/log/adapter/audit.log
```

Additional resources

- See [Preparing to configure the monitoring stack](#) for steps to create monitoring config maps.

### 4.1. DISABLING THE LOCAL ALERTMANAGER

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

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

**Prerequisites**

- You have access to the cluster as a user with the `cluster-admin` role.
- You have created the `cluster-monitoring-config` config map.
- You have installed the OpenShift CLI (`oc`).

**Procedure**

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

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

2. Add `enabled: false` for the `alertmanagerMain` component under `data/config.yaml`:

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

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

**Additional resources**

- [Prometheus Alertmanager documentation](#)
- [Managing alerts](#)

### 4.2. NEXT STEPS
• Enabling monitoring for user-defined projects

• Learn about remote health reporting and, if necessary, opt out of it.
CHAPTER 5. ENABLING MONITORING FOR USER-DEFINED PROJECTS

In OpenShift Container Platform 4.11, you can enable monitoring for user-defined projects in addition to the default platform monitoring. You can monitor your own projects in OpenShift Container Platform without the need for an additional monitoring solution. Using this feature centralizes monitoring for core platform components and user-defined projects.

**NOTE**

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

5.1. ENABLING MONITORING FOR USER-DEFINED PROJECTS

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

**IMPORTANT**

In OpenShift Container Platform 4.11 you must remove any custom Prometheus instances before enabling monitoring for user-defined projects.

**NOTE**

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

Prerequisites

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

**NOTE**

Every time you save configuration changes to the `user-workload-monitoring-config` ConfigMap object, the pods in the `openshift-user-workload-monitoring` project are redeployed. It can sometimes take a while for these components to redeploy. You can create and configure the ConfigMap object before you first enable monitoring for user-defined projects, to prevent having to redeploy the pods often.
Procedure

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

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

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

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

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

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

   **WARNING**

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

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

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

   **Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>prometheus-operator-6f7b748d5b-t7nbg</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>3h</td>
</tr>
<tr>
<td>prometheus-user-workload-0</td>
<td>4/4</td>
<td>Running</td>
<td>1</td>
<td>3h</td>
</tr>
<tr>
<td>prometheus-user-workload-1</td>
<td>4/4</td>
<td>Running</td>
<td>1</td>
<td>3h</td>
</tr>
<tr>
<td>thanos-ruler-user-workload-0</td>
<td>3/3</td>
<td>Running</td>
<td>0</td>
<td>3h</td>
</tr>
<tr>
<td>thanos-ruler-user-workload-1</td>
<td>3/3</td>
<td>Running</td>
<td>0</td>
<td>3h</td>
</tr>
</tbody>
</table>

   **Additional resources**

   - Creating a cluster monitoring config map
5.2. GRANTING USERS PERMISSION TO MONITOR USER-DEFINED PROJECTS

Cluster administrators can monitor all core OpenShift Container Platform and user-defined projects.

Cluster administrators can grant developers and other users permission to monitor their own projects. Privileges are granted by assigning one of the following monitoring roles:

- The `monitoring-rules-view` role provides read access to `PrometheusRule` custom resources for a project.
- The `monitoring-rules-edit` role grants a user permission to create, modify, and deleting `PrometheusRule` custom resources for a project.
- The `monitoring-edit` role grants the same privileges as the `monitoring-rules-edit` role. Additionally, it enables a user to create new scrape targets for services or pods. With this role, you can also create, modify, and delete `ServiceMonitor` and `PodMonitor` resources.

You can also grant users permission to configure the components that are responsible for monitoring user-defined projects:

- The `user-workload-monitoring-config-edit` role in the `openshift-user-workload-monitoring` project enables you to edit the `user-workload-monitoring-config ConfigMap` object. With this role, you can edit the `ConfigMap` object to configure Prometheus, Prometheus Operator, and Thanos Ruler for user-defined workload monitoring.

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

- The `alert-routing-edit` role grants a user permission to create, update, and delete `AlertmanagerConfig` custom resources for a project.

This section provides details on how to assign these roles by using the OpenShift Container Platform web console or the CLI.

5.2.1. Granting user permissions by using the web console

You can grant users permissions to monitor their own projects, by using the OpenShift Container Platform web console.

**Prerequisites**

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

**Procedure**

1. In the Administrator perspective within the OpenShift Container Platform web console, navigate to User Management → Role Bindings → Create Binding.

2. In the Binding Type section, select the "Namespace Role Binding" type.
3. In the Name field, enter a name for the role binding.

4. In the Namespace field, select the user-defined project where you want to grant the access.

   **IMPORTANT**

   The monitoring role will be bound to the project that you apply in the Namespace field. The permissions that you grant to a user by using this procedure will apply only to the selected project.

5. Select monitoring-rules-view, monitoring-rules-edit, or monitoring-edit in the Role Name list.

6. In the Subject section, select User.

7. In the Subject Name field, enter the name of the user.

8. Select Create to apply the role binding.

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

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

**Prerequisites**

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

**Procedure**

- Assign a monitoring role to a user for a project:

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

  Substituted `<role>` with monitoring-rules-view, monitoring-rules-edit, or monitoring-edit.

  **IMPORTANT**

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

  As an example, substitute `<role>` with monitoring-edit, `<user>` with johnsmith, and `<namespace>` with ns1. This assigns the user johnsmith permission to set up metrics collection and to create alerting rules in the ns1 namespace.
You can grant users permission to configure monitoring for user-defined projects.

Prerequisites

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

Procedure

- Assign the `user-workload-monitoring-config-edit` role to a user in the `openshift-user-workload-monitoring` project:

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

5.4. ACCESSING METRICS FROM OUTSIDE THE CLUSTER FOR CUSTOM APPLICATIONS

Learn how to query Prometheus statistics from the command line when monitoring your own services. You can access monitoring data from outside the cluster with the `thanos-querier` route.

Prerequisites

- You deployed your own service, following the Enabling monitoring for user-defined projects procedure.

Procedure

1. Extract a token to connect to Prometheus:

```bash
$ SECRET=`oc get secret -n openshift-user-workload-monitoring | grep prometheus-user-workload-token | head -n 1 | awk '{print $1 }`

$ TOKEN=`echo $(oc get secret $SECRET -n openshift-user-workload-monitoring -o json | jq -r '.data.token') | base64 -d`

$ THANOS_QUERIER_HOST=`oc get route thanos-querier -n openshift-monitoring -o json | jq -r '.spec.host'`

2. Extract your route host:

```bash
$ NAMESPACE=ns1

```
The output will show you the duration that your application pods have been up.

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-jzt2","service":"prometheus-example-app"},"value":[1591881154.748,1]}}}
```

### 5.5. Excluding a User-Defined Project from Monitoring

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

**Procedure**

1. Add the label to the project namespace:

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

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

   ```bash
   $ 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.

### 5.6. Disabling Monitoring for User-Defined Projects

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

**NOTE**

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

**Procedure**

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

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

   a. Set `enableUserWorkload: false` under `data/config.yaml`:

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

   ```yaml
   enableUserWorkload: false
   ```
2. Save the file to apply the changes. Monitoring for user-defined projects is then disabled automatically.

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

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

Example output

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

**NOTE**

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

### 5.7. NEXT STEPS

- Managing metrics
CHAPTER 6. ENABLING ALERT ROUTING FOR USER-DEFINED PROJECTS

In OpenShift Container Platform 4.11, a cluster administrator can enable alert routing for user-defined projects. This process consists of two general steps:

- Enable alert routing for user-defined projects to use the default platform Alertmanager instance or, optionally, a separate Alertmanager instance only for user-defined projects.
- Grant users permission to configure alert routing for user-defined projects.

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

6.1. UNDERSTANDING ALERT ROUTING FOR USER-DEFINED PROJECTS

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

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

After a user has defined alert routing for a user-defined project, user-defined alert notifications are routed as follows:

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

**NOTE**

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.

6.2. ENABLING THE PLATFORM ALERTMANAGER INSTANCE FOR USER-DEFINED ALERT ROUTING

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

**Prerequisites**
Prerequisites

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

Procedure

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

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

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

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

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

3. Save the file to apply the changes.

### 6.3. Enabling a Separate Alertmanager Instance for User-Defined Alert Routing

In some clusters, you might want to deploy a dedicated Alertmanager instance for user-defined projects, which can help reduce the load on the default platform Alertmanager instance and can better separate user-defined alerts from default platform alerts. In these cases, you can optionally enable a separate instance of Alertmanager to send alerts for user-defined projects only.

Prerequisites

- You have access to the cluster as a user with the `cluster-admin` role.
- You have enabled monitoring for user-defined projects in the `cluster-monitoring-config` config map for the `openshift-monitoring` namespace.
- You have installed the OpenShift CLI (`oc`).

Procedure

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

   ```bash
   $ 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 `user-workload` Alertmanager instance has started:

  ```bash
  # oc -n openshift-user-workload-monitoring get alertmanager
  ```

  **Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>VERSION</th>
<th>REPLICAS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>user-workload</td>
<td>0.24.0</td>
<td>2</td>
<td>100s</td>
</tr>
</tbody>
</table>

**6.4. GRANTING USERS PERMISSION TO CONFIGURE ALERT ROUTING FOR USER-DEFINED PROJECTS**

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

**Prerequisites**

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

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

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

  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**

- Enabling monitoring for user defined projects
- Creating alert routing for user-defined projects

**6.5. NEXT STEPS**

- Managing alerts
CHAPTER 7. MANAGING METRICS

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

7.1. UNDERSTANDING METRICS

In OpenShift Container Platform 4.11, cluster components are monitored by scraping metrics exposed through service endpoints. You can also configure metrics collection for user-defined projects.

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

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

```bash
$ 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

7.2. 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.

7.2.1. 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:

```yaml
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:0.4.1
    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:

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

It takes some time to deploy the service.

4. You can check that the pod is running:
7.2.2. Specifying how a service is monitored

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

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

**Prerequisites**

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

**NOTE**

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

**Procedure**

1. Create a 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:

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

```bash
$ oc -n ns1 get pod

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>prometheus-example-app-7857545cb7-sbgwq</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>81m</td>
</tr>
</tbody>
</table>
```
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

<table>
<thead>
<tr>
<th>NAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>prometheus-example-monitor</td>
<td>81m</td>
</tr>
</tbody>
</table>

Additional resources

- Enabling monitoring for user-defined projects
- How to scrape metrics using TLS in a ServiceMonitor configuration in a user-defined project
- PodMonitor API
- ServiceMonitor API

7.3. NEXT STEPS

- Querying metrics
CHAPTER 8. QUERYING METRICS

You can query metrics to view data about how cluster components and your own workloads are performing.

8.1. ABOUT QUERYING METRICS

The OpenShift Container Platform 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 **cluster administrator**, you can query metrics for all core OpenShift Container Platform and 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.

8.1.1. Querying metrics for all projects as a cluster administrator

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

**Prerequisites**

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

**Procedure**

1. Select the **Administrator** perspective in the OpenShift Container Platform web console.

2. Select **Observe** → **Metrics**.

3. Select **Insert Metric at Cursor** to view a list of predefined queries.

4. To create a custom query, add your Prometheus Query Language (PromQL) query to the **Expression** field.

**NOTE**

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.

5. To add multiple queries, select **Add Query**.

6. To duplicate an existing query, select **next to the query, then choose Duplicate query.**
7. To delete a query, select next to the query, then choose **Delete query**.

8. To disable a query from being run, select next to the query and choose **Disable query**.

9. 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.

10. Optional: The page URL now contains the queries you ran. To use this set of queries again in the future, save this URL.

**Additional resources**

- For more information about creating PromQL queries, see the Prometheus query documentation.

### 8.1.2. 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 in the **Observe → Metrics** page in the web console for your user-defined project.

**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. Select the **Developer** perspective in the OpenShift Container Platform web console.

2. Select **Observe → Metrics**.

3. Select the project that you want to view metrics for in the **Project** list.

4. Select a query from the **Select query** list, or create a custom PromQL query based on the selected query by selecting **Show PromQL**.

5. Optional: Select **Custom query** from the **Select query** list to enter a new query. As you type, autocomplete suggestions appear in a drop-down list. These suggestions include functions and metrics. Click a suggested item to select it.

**NOTE**

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

**Additional resources**

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

8.1.3. Exploring the visualized metrics

After running the queries, the metrics are displayed on an interactive plot. The X-axis in the plot represents time and the Y-axis represents metrics values. Each metric is shown as a colored line on the graph. You can manipulate the plot interactively and explore the metrics.

**Procedure**

**In the Administrator perspective:**

1. Initially, all metrics from all enabled queries are shown on the plot. You can select which metrics are shown.

   **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.

   - To hide all metrics from a query, click **▼** for the query and click **Hide all series**.
   - To hide a specific metric, go to the query table and click the colored square near the metric name.

2. To zoom into the plot and change the time range, do one of the following:
   - 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.

3. To reset the time range, select **Reset Zoom**.
4. To 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.

5. To hide the plot, select **Hide Graph**.

In the **Developer** perspective:

1. To zoom into the plot and change the time range, do one of the following:
   - 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.

2. To reset the time range, select **Reset Zoom**.

3. To 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.

**Additional resources**

- See [Querying metrics](#) for details on using the PromQL interface.
- See [Querying metrics for all projects as an administrator](#) for details on accessing metrics for all projects as an administrator.
- See [Querying metrics for user-defined projects as a developer](#) for details on accessing non-cluster metrics as a developer or a privileged user.

**8.2. NEXT STEPS**

- Managing metrics targets
CHAPTER 9. MANAGING METRICS TARGETS

OpenShift Container Platform Monitoring collects metrics from targeted cluster components by scraping data from exposed service endpoints.

In the Administrator perspective in the OpenShift Container Platform 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 OpenShift Container Platform Monitoring is not able to scrape metrics from a targeted component.

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

9.1. ACCESSING THE METRICS TARGETS PAGE IN THE ADMINISTRATOR PERSPECTIVE

You can view the Metrics Targets page in the Administrator perspective in the OpenShift Container Platform web console.

Prerequisites

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

Procedure

- 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.

9.2. SEARCHING AND FILTERING METRICS TARGETS

The list of metrics targets can be long. You can filter and search these targets based on various criteria.

In the Administrator perspective, the Metrics Targets page provides details about targets for default OpenShift Container Platform and user-defined projects. This page lists the following information for each target:

- the service endpoint URL being scraped
- the ServiceMonitor component being monitored
- the up or down status of the target
- the namespace
- the last scrape time
- the duration of the last scrape

You can filter the list of targets by status and source. The following filtering options are available:

- **Status** filters:
  - **Up**. The target is currently up and being actively scraped for metrics.
Down. The target is currently down and not being scraped for metrics.

- **Source** filters:
  - **Platform**: Platform-level targets relate only to default OpenShift Container Platform projects. These projects provide core OpenShift Container Platform functionality.
  - **User**: User targets relate to user-defined projects. These projects are user-created and can be customized.

You can also use the search box to find a target by target name or label. Select **Text** or **Label** from the search box menu to limit your search.

### 9.3. GETTING DETAILED INFORMATION ABOUT A TARGET

On the **Target details** page, you can view detailed information about a metric target.

**Prerequisites**

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

**Procedure**

**To view detailed information about a target in the Administrator perspective**

1. Open the OpenShift Container Platform web console and navigate to **Observe → Targets**.
2. Optional: Filter the targets by status and source by selecting filters in the **Filter** list.
3. Optional: Search for a target by name or label by using the **Text** or **Label** field next to the search box.
4. Optional: Sort the targets by clicking one or more of the **Endpoint**, **Status**, **Namespace**, **Last Scrape**, and **Scrape Duration** column headers.
5. 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 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

### 9.4. NEXT STEPS

- Managing alerts
CHAPTER 10. MANAGING ALERTS

In OpenShift Container Platform 4.11, the Alerting UI enables you to manage alerts, silences, and alerting rules.

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

- **Alerts.** An alert is fired when the conditions defined in an alerting rule are true. Alerts provide a notification that a set of circumstances are apparent within an OpenShift Container Platform cluster.

- **Silences.** A silence can be applied to an alert to prevent notifications from being sent when the conditions for an alert are true. You can mute an alert after the initial notification, while you work on resolving the 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 with `cluster-admin` privileges, 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` 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` 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 in the OpenShift Container Platform web console.

- In the **Administrator** perspective, select **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, select **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 OpenShift Container Platform and user-defined projects that you have access to in the **Project** list. However, alerts, silences, and alerting rules relating to core OpenShift Container Platform projects are not displayed if you do not have `cluster-admin` privileges.
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 OpenShift Container Platform and user-defined projects. The page includes a summary of severity, state, and source for each alert. The time at which an alert went into its current state is also shown.

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

- **Alert State** filters:
  - **Firing**: The alert is firing because the alert condition is true and the optional for duration has passed. The alert will continue to fire as long as 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 will not be sent for alerts that match all the listed values or regular expressions.

- **Severity** filters:
  - **Critical**: The condition that triggered the alert could have a critical impact. The alert requires immediate attention when fired and is typically paged to an individual or to a critical response team.
  - **Warning**: The alert provides a warning notification about something that might require attention to prevent a problem from occurring. Warnings are typically routed to a ticketing system for non-immediate review.
  - **Info**: The alert is provided for informational purposes only.
  - **None**: The alert has no defined severity.
  - You can also create custom severity definitions for alerts relating to user-defined projects.

- **Source** filters:
  - **Platform**: Platform-level alerts relate only to default OpenShift Container Platform projects. These projects provide core OpenShift Container Platform functionality.
  - **User**: User alerts relate to user-defined projects. These alerts are user-created and are customizable. User-defined workload monitoring can be enabled post-installation 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 OpenShift Container Platform and user-defined projects. The page includes a summary of the state of each silence and the time at which a silence ends.
You can filter by silence state. By default, only Active and Pending silences are displayed. The following describes each silence state filter option:

- **Silence 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 OpenShift Container Platform and user-defined projects. The page includes a summary of the state, severity, and source for each alerting rule.

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

- **Alert State filters:**
  - **Firing.** The alert is firing because the alert condition is true and the optional for duration has passed. The alert will continue to fire as long as 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 will not be sent for alerts that match all the listed values or regular expressions.
  - **Not Firing.** The alert is not firing.

- **Severity filters:**
  - **Critical.** The conditions defined in the alerting rule could have a critical impact. When true, these conditions require immediate attention. Alerts relating to the rule are typically paged to an individual or to a critical response team.
  - **Warning.** The conditions defined in the alerting rule might require attention to prevent a problem from occurring. Alerts relating to the rule are typically routed to a ticketing system for non-immediate review.
  - **Info.** The alerting rule provides informational alerts only.
  - **None.** The alerting rule has no defined severity.
  - You can also create custom severity definitions for alerting rules relating to user-defined projects.

- **Source filters:**
  - **Platform.** Platform-level alerting rules relate only to default OpenShift Container Platform projects. These projects provide core OpenShift Container Platform functionality.
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 post-installation 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 metrics for.

Procedure
To obtain information about alerts in the Administrator perspective
1. Open the OpenShift Container Platform web console and navigate to the Observe → Alerting → Alerts page.
2. Optional: Search for alerts by name 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. Select the name of an alert to navigate to its Alert Details page. The page includes a graph that illustrates alert time series data. It also provides information about the alert, including:
   - A description of the alert
   - Messages associated with the alerts
   - 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. Navigate 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, and State column headers.

5. Select the name of a silence to navigate to 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. Navigate 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 navigate to 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. Navigate to the Observe → <project_name> → Alerts page.

2. View details for an alert, silence, or an alerting rule:
   - Alert Details can be viewed by selecting > to the left of an alert name and then selecting the alert in the list.

   - Silence Details can be viewed by selecting 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 selecting View Alerting Rule in the menu on the right of an alert in the Alerts page.

**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 OpenShift Container Platform monitoring alerts.

### 10.4. MANAGING SILENCES

You can create a silence to stop receiving notifications about an alert when it is firing. It might be useful to silence an alert after being first notified, while you resolve the underlying issue.

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.

You can view, edit, and expire existing silences.

#### 10.4.1. Silencing alerts

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

**Prerequisites**

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

- You are a non-administrator user and 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** 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. Navigate to the **Observe → Alerting → Alerts** page of the OpenShift Container Platform web console.

2. For the alert that you want to silence, select the **Silence Alert** in the right-hand column and select **Silence Alert**. The **Silence Alert** form will appear with a pre-populated specification for the chosen alert.

3. Optional: Modify the silence.

4. You must add a comment before creating the silence.

5. To create the silence, select **Silence**.

   - In the **Developer** perspective:

     1. Navigate to the **Observe → <project_name> → Alerts** page in the OpenShift Container Platform web console.

     2. Expand the details for an alert by selecting > to the left of the alert name. Select the name of the alert in the expanded view to open the **Alert Details** page for the alert.

     3. Select **Silence Alert**. The **Silence Alert** form will appear with a pre-populated specification for the chosen alert.

     4. Optional: Modify the silence.

     5. You must add a comment before creating the silence.

     6. To create the silence, select **Silence**.

To silence a set of alerts by creating an alert specification in the **Administrator** perspective:

1. Navigate to the **Observe → Alerting → Silences** page in the OpenShift Container Platform web console.

2. Select **Create Silence**.

3. Set the schedule, duration, and label details for an alert in the **Create Silence** form. You must also add a comment for the silence.

4. To create silences for alerts that match the label sectors that you entered in the previous step, select **Silence**.

### 10.4.2. Editing silences

You can edit a silence, which will expire the existing silence and create a new one with the changed configuration.

**Procedure**

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

1. Navigate to the **Observe → Alerting → Silences** page.
2. For the silence you want to modify, select the in the last column and choose Edit silence. Alternatively, you can select Actions → Edit Silence in the Silence Details page for a silence.

3. In the Edit Silence page, enter your changes and select Silence. This will expire the existing silence and create one with the chosen configuration.

To edit a silence in the Developer perspective:

1. Navigate to the Observe → <project_name> → Alerts page.

2. Expand the details for an alert by selecting to the left of the alert name. Select the name of the alert in the expanded view to open the Alert Details page for the alert.

3. Select the name of a silence in the Silenced By section in that page to navigate to the Silence Details page for the silence.

4. Select the name of a silence to navigate to its Silence Details page.

5. Select Actions → Edit Silence in the Silence Details page for a silence.

6. In the Edit Silence page, enter your changes and select Silence. This will expire the existing silence and create one with the chosen configuration.

10.4.3. Expiring silences

You can expire a silence. Expiring a silence deactivates it forever.

Procedure

To expire a silence in the Administrator perspective:

1. Navigate to the Observe → Alerting → Silences page.

2. For the silence you want to modify, select the in the last column and choose Expire silence. Alternatively, you can select Actions → Expire Silence in the Silence Details page for a silence.

To expire a silence in the Developer perspective:

1. Navigate to the Observe → <project_name> → Alerts page.

2. Expand the details for an alert by selecting to the left of the alert name. Select the name of the alert in the expanded view to open the Alert Details page for the alert.

3. Select the name of a silence in the Silenced By section in that page to navigate to the Silence Details page for the silence.

4. Select the name of a silence to navigate to its Silence Details page.

5. Select Actions → Expire Silence in the Silence Details page for a silence.

10.5. MANAGING ALERTING RULES FOR USER-DEFINED PROJECTS
OpenShift Container Platform monitoring ships with a set of default alerting rules. As a cluster administrator, you can view the default alerting rules.

In OpenShift Container Platform 4.11, you can create, view, edit, and remove alerting rules in user-defined projects.

Alerting rule considerations

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

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.

- **Optimize alert routing** Deploy an alerting rule directly on the Prometheus instance in the `openshift-user-workload-monitoring` project if the rule does not query default OpenShift Container Platform metrics. This reduces latency for alerting rules and minimizes the load on monitoring components.
WARNING

Default OpenShift Container Platform metrics for user-defined projects provide information about CPU and memory usage, bandwidth statistics, and packet rate information. Those metrics cannot be included in an alerting rule if you route the rule directly to the Prometheus instance in the openshift-user-workload-monitoring project. Alerting rule optimization should be used only if you have read the documentation and have a comprehensive understanding of the monitoring architecture.

Additional resources

- See the Prometheus alerting documentation for further guidelines on optimizing alerts
- See Monitoring overview for details about OpenShift Container Platform 4.11 monitoring architecture

10.5.2. Creating alerting rules for user-defined projects

You can create alerting rules for user-defined projects. Those alerting rules will fire alerts based on the values of chosen metrics.

Prerequisites

- You have enabled monitoring for user-defined projects.
- You are logged in as a user that has the monitoring-rules-edit 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. For example:

```yaml
NOTE
When you create an alerting rule, a project label is enforced on it if a rule with the same name exists in another project.

apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-alert
namespace: ns1
spec:
groups:
```
This configuration creates an alerting rule named example-alert. The alerting rule fires an alert when the version metric exposed by the sample service becomes 0.

**IMPORTANT**

A user-defined alerting rule can include metrics for its own project and cluster metrics. You cannot include metrics for another user-defined project.

For example, an alerting rule for the user-defined project ns1 can have metrics from ns1 and cluster metrics, such as the CPU and memory metrics. However, the rule cannot include metrics from ns2.

Additionally, you cannot create alerting rules for the openshift-* core OpenShift Container Platform projects. OpenShift Container Platform monitoring by default provides a set of alerting rules for these projects.

3. Apply the configuration file to the cluster:

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

   It takes some time to create the alerting rule.

### 10.5.3. Reducing latency for alerting rules that do not query platform metrics

If an alerting rule for a user-defined project does not query default cluster metrics, you can deploy the rule directly on the Prometheus instance in the openshift-user-workload-monitoring project. This reduces latency for alerting rules by bypassing Thanos Ruler when it is not required. This also helps to minimize the overall load on monitoring components.

**WARNING**

Default OpenShift Container Platform metrics for user-defined projects provide information about CPU and memory usage, bandwidth statistics, and packet rate information. Those metrics cannot be included in an alerting rule if you deploy the rule directly to the Prometheus instance in the openshift-user-workload-monitoring project. The procedure outlined in this section should only be used if you have read the documentation and have a comprehensive understanding of the monitoring architecture.

**Prerequisites**

- You have enabled monitoring for user-defined projects.
- You are logged in as a user that has the monitoring-rules-edit 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 that includes a label with the key `openshift.io/prometheus-rule-evaluation-scope` and value `leaf-prometheus`. For example:

   ```yaml
   apiVersion: monitoring.coreos.com/v1
   kind: PrometheusRule
   metadata:
     name: example-alert
     namespace: ns1
     labels:
       openshift.io/prometheus-rule-evaluation-scope: leaf-prometheus
   spec:
     groups:
       - name: example
         rules:
           - alert: VersionAlert
             expr: version{job="prometheus-example-app"} == 0
   ```

   If that label is present, the alerting rule is deployed on the Prometheus instance in the `openshift-user-workload-monitoring` project. If the label is not present, the alerting rule is deployed to Thanos Ruler.

1. Apply the configuration file to the cluster:

   ```bash
   $ oc apply -f example-app-alerting-rule.yaml
   $ oc -n <project> get prometheusrule
   ```

   It takes some time to create the alerting rule.

   • See Monitoring overview for details about OpenShift Container Platform 4.11 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` 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` role for your project.

• You have installed the OpenShift CLI (oc).

Procedure

1. You can list alerting rules in `<project>`:

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

**Prerequisites**

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

**Procedure**

1. 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` 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](https://monitoring.Openshift.redhat.com/docs/latest/prometheus/alertmanager/)

### 10.6. MANAGING ALERTING RULES FOR CORE PLATFORM MONITORING
IMPORTANT

Creating and modifying alerting rules for core platform monitoring 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 https://access.redhat.com/support/offerings/techpreview/.

OpenShift Container Platform 4.11 monitoring ships with a large set of default alerting rules for platform metrics. As a cluster administrator, you can customize this set of rules in two ways:

- Modify the settings for existing platform alerting rules by adjusting thresholds or by adding and modifying labels. For example, you can change the severity label for an alert from warning to critical to help you route and triage issues flagged by an alert.

- Define and add new custom alerting rules by constructing a query expression based on core platform metrics in the openshift-monitoring namespace.

Core platform alerting rule considerations

- New alerting rules must be based on the default OpenShift Container Platform monitoring metrics.

- You can only add and modify alerting rules. You cannot create new recording rules or modify existing recording rules.

- If you modify existing platform alerting rules by using an AlertRelabelConfig object, your modifications are not reflected in the Prometheus alerts API. Therefore, any dropped alerts still appear in the OpenShift Container Platform web console even though they are no longer forwarded to Alertmanager. Additionally, any modifications to alerts, such as a changed severity label, do not appear in the web console.

10.6.1. Modifying core platform alerting rules

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

Prerequisites

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

- You have installed the OpenShift CLI (oc).

- You have enabled Technology Preview features, and all nodes in the cluster are ready.

Procedure

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

```yaml
apiVersion: monitoring.openshift.io/v1alpha1
document kind: AlertRelabelConfig
metadata:
  name: watchdog
  namespace: openshift-monitoring
document spec:
  configs:
    - sourceLabels: [alertname,severity]
      regex: "Watchdog:none"
      targetLabel: severity
      replacement: critical
      action: Replace
```

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

3. Apply the configuration file to the cluster:

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

10.6.2. Creating new alerting rules

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

**NOTE**

If you create a customized AlertingRule resource based on an existing platform alerting rule, silence the original alert to avoid receiving conflicting alerts.

**Prerequisites**

- You are logged in as a user that has the cluster-admin role.
- You have installed the OpenShift CLI (`oc`).
- You have enabled Technology Preview features, and all nodes in the cluster are ready.

**Procedure**
1. Create a new YAML configuration file named `example-alerting-rule.yaml` in the `openshift-monitoring` namespace.

2. Add an `AlertingRule` resource to the YAML file. The following example creates a new alerting rule named `example`, similar to the default `watchdog` alert:

   ```yaml
   apiVersion: monitoring.openshift.io/v1alpha1
   kind: AlertingRule
   metadata:
     name: example
     namespace: openshift-monitoring
   spec:
     groups:
       - name: example-rules
         rules:
           - alert: ExampleAlert
             expr: vector(1)
   
   1. The name of the alerting rule you want to create.
   2. The PromQL query expression that defines the new rule.

3. Apply the configuration file to the cluster:

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

Additional resources

- See [Monitoring overview](#) for details about OpenShift Container Platform 4.11 monitoring architecture.
- See the [Alertmanager documentation](#) for information about alerting rules.
- See the [Prometheus relabeling documentation](#) for information about how relabeling works.
- See the [Prometheus alerting documentation](#) for further guidelines on optimizing alerts.

10.7. SENDING NOTIFICATIONS TO EXTERNAL SYSTEMS

In OpenShift Container Platform 4.11, firing alerts can be viewed in the Alerting UI. Alerts are not configured by default to be sent to any notification systems. You can configure OpenShift Container Platform to send alerts to the following receiver types:

- PagerDuty
- Webhook
- Email
- Slack

Routing alerts to receivers enables you to send timely notifications to the appropriate teams when failures occur. For example, critical alerts require immediate attention and are typically paged to an individual or a critical response team. Alerts that provide non-critical warning notifications might instead
be routed to a ticketing system for non-immediate review.

**Checking that alerting is operational by using the watchdog alert**

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

**10.7.1. Configuring alert receivers**

You can configure alert receivers to ensure that you learn about important issues with your cluster.

**Prerequisites**

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

**Procedure**

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

   **NOTE**

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

2. Select Create Receiver in the Receivers section of the page.

3. In the Create Receiver form, add a Receiver Name and choose a Receiver Type from the list.

4. Edit the receiver configuration:
   - For PagerDuty receivers:
     a. Choose an integration type and add a PagerDuty integration key.
     b. Add the URL of your PagerDuty installation.
     c. Select Show advanced configuration if you want to edit the client and incident details or the severity specification.
   - For webhook receivers:
     a. Add the endpoint to send HTTP POST requests to.
     b. Select Show advanced configuration if you want to edit the default option to send resolved alerts to the receiver.
   - For email receivers:
     a. Add the email address to send notifications to.
b. Add SMTP configuration details, including the address to send notifications from, the smarthost and port number used for sending emails, the hostname of the SMTP server, and authentication details.

c. Choose whether TLS is required.

d. Select Show advanced configuration if you want to edit the default option not to send resolved alerts to the receiver or edit the body of email notifications configuration.

- For Slack receivers:
  a. Add the URL of the Slack webhook.
  b. Add the Slack channel or user name to send notifications to.
  c. Select Show advanced configuration if you want to edit the default option not to send resolved alerts to the receiver or edit the icon and username configuration. You can also choose whether to find and link channel names and usernames.

5. By default, firing alerts with labels that match all of the selectors will be sent to the receiver. If you want label values for firing alerts to be matched exactly before they are sent to the receiver:
   a. Add routing label names and values in the Routing Labels section of the form.
   b. Select Regular Expression if want to use a regular expression.
   c. Select Add Label to add further routing labels.

6. Select Create to create the receiver.

10.7.2. Creating alert routing for user-defined projects

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

Prerequisites

- A cluster administrator has enabled monitoring for user-defined projects.
- A cluster administrator has enabled alert routing for user-defined projects.
- You are logged in as a user that has the alert-routing-edit 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:

```yaml
apiVersion: monitoring.coreos.com/v1beta1
kind: AlertmanagerConfig
metadata:
  name: example-routing
```
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:

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

   The configuration is automatically applied to the Alertmanager pods.

---

### 10.8. APPLYING A CUSTOM ALERTMANAGER CONFIGURATION

You can overwrite the default Alertmanager configuration by editing the `alertmanager-main` secret in the `openshift-monitoring` namespace for the platform instance of Alertmanager.

**Prerequisites**

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

**Procedure**

To change the Alertmanager configuration from the CLI:

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

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

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

   ```yaml
   namespace: ns1
   spec:
   route:
     receiver: default
    groupBy: [job]
   receivers:
     - name: default
   webhookConfigs:
     - url: https://example.org/post
   ```

   ```yaml
   global:
     resolve_timeout: 5m
   route:
     group_wait: 30s
     group_interval: 5m
     repeat_interval: 12h
   receiver: default
   routes:
     - match:
   ```
service specifies the service that fires the alerts.

<your_matching_rules> specifies the target alerts.

receiver specifies the receiver to use for the alert.

The following Alertmanager configuration example configures PagerDuty as an alert receiver:

```
global:
  resolve_timeout: 5m
route:
group_wait: 30s
group_interval: 5m
repeat_interval: 12h
receiver: default
routes:
  - match:
      alertname: Watchdog
      repeat_interval: 5m
      receiver: watchdog
  - match: service: example-app
    routes: - match: severity: critical
      receiver: team-frontend-page
receivers:
  - name: default
  - name: watchdog
  - name: team-frontend-page pagerduty_configs: - service_key: "your-key"
```

With this configuration, alerts of critical severity that are fired by the example-app service are sent using the team-frontend-page receiver. Typically these types of alerts would be paged to an individual or a critical response team.

3. Apply the new configuration in the file:

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

To change the Alertmanager configuration from the OpenShift Container Platform web console:
1. Navigate to the Administration → Cluster Settings → Configuration → Alertmanager → YAML page of the web console.

2. Modify the YAML configuration file.

3. Select Save.

10.9. 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 cluster-admin role.

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

   ```bash
   $ 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:

   ```yaml
   route:
     receiver: Default
     group_by:
       - name: Default
         routes:
           - matchers:
             - "service = prometheus-example-monitor"
               receiver: <receiver>
     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:

   ```bash
   $ 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.

• See Enabling alert routing for user-defined projects to learn how to enable a dedicated instance of Alertmanager for user-defined alert routing.

10.10. NEXT STEPS

• Reviewing monitoring dashboards
CHAPTER 11. REVIEWING MONITORING DASHBOARDS

OpenShift Container Platform 4.11 provides a comprehensive set of monitoring dashboards that help you understand the state of cluster components and user-defined workloads.

Use the **Administrator** perspective to access dashboards for the core OpenShift Container Platform components, including the following items:

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

**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
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 OpenShift Container Platform cluster components.

Prerequisites

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

Procedure

1. In the Administrator perspective in the OpenShift Container Platform 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.
11.2. REVIEWING MONITORING DASHBOARDS AS A DEVELOPER

Use the Developer perspective to view Kubernetes compute resources dashboards of a selected project.

**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 OpenShift Container Platform 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.

**Additional resources**

- Monitoring project and application metrics using the Developer perspective

**11.3. NEXT STEPS**
- Accessing third-party monitoring APIs
12.1. INTRODUCTION

The OpenShift Console NVIDIA GPU plug-in is a dedicated administration dashboard for NVIDIA GPU usage visualization in the OpenShift Container Platform (OCP) Console. The visualizations in the administration dashboard provide guidance on how to best optimize GPU resources in clusters, such as when a GPU is under- or over-utilized.

The OpenShift Console NVIDIA GPU plug-in works as a remote bundle for the OCP console. To run the plug-in, the OCP console must be running.

12.2. INSTALLING THE NVIDIA GPU ADMINISTRATION DASHBOARD

Install the NVIDIA GPU plug-in by using Helm on the OpenShift Container Platform (OCP) Console to add GPU capabilities.

The OpenShift Console NVIDIA GPU plug-in works as a remote bundle for the OCP console. To run the OpenShift Console NVIDIA GPU plug-in, an instance of the OCP console must be running.

Prerequisites

- Red Hat OpenShift 4.11+
- NVIDIA GPU operator
- Helm

Procedure

Use the following procedure to install the OpenShift Console NVIDIA GPU plug-in.

1. Add the Helm repository:


   $ helm repo update

2. Install the Helm chart in the default NVIDIA GPU operator namespace:

   $ helm install -n nvidia-gpu-operator console-plugin-nvidia-gpu rh-ecosystem-edge/console-plugin-nvidia-gpu

Example output

NAME: console-plugin-nvidia-gpu
LAST DEPLOYED: Tue Aug 23 15:37:35 2022
NAMESPACE: nvidia-gpu-operator
STATUS: deployed
REVISION: 1
NOTES:
View the Console Plugin NVIDIA GPU deployed resources by running the following command:

```
$ oc -n {{ .Release.Namespace }} get all -l app.kubernetes.io/name=console-plugin-nvidia-gpu
```

Enable the plugin by running the following command:

```
# Check if a plugins field is specified
$ oc get consoles.operator.openshift.io cluster --output=jsonpath="{.spec.plugins}"

# if not, then run the following command to enable the plugin
$ oc patch consoles.operator.openshift.io cluster --patch '{ "spec": { "plugins": ["console-plugin-nvidia-gpu"] } }' --type=merge

# if yes, then run the following command to enable the plugin
$ oc patch consoles.operator.openshift.io cluster --patch '["op": "add", "path": "/spec/plugins/-", "value": "console-plugin-nvidia-gpu"]' --type=json

# add the required DCGM Exporter metrics ConfigMap to the existing NVIDIA operator ClusterPolicy CR:
oc patch clusterpolicies.nvidia.com gpu-cluster-policy --patch '{ "spec": { "dcgmExporter": { "config": { "name": "console-plugin-nvidia-gpu" } } } }' --type=merge
```

The dashboard relies mostly on Prometheus metrics exposed by the NVIDIA DCGM Exporter, but the default exposed metrics are not enough for the dashboard to render the required gauges. Therefore, the DGCN exporter is configured to expose a custom set of metrics, as shown here.

```yaml
apiVersion: v1
data:
dcgm-metrics.csv:
  DCGM_FI_PROF_GR_ENGINE_ACTIVE, gauge, gpu utilization.
  DCGM_FI_DEV_MEM_COPY_UTIL, gauge, mem utilization.
  DCGM_FI_DEV_ENC_UTIL, gauge, enc utilization.
  DCGM_FI_DEV_DEC_UTIL, gauge, dec utilization.
  DCGM_FI_DEV_POWER_USAGE, gauge, power usage.
  DCGM_FI_DEV_POWER_MGMT_LIMIT_MAX, gauge, power mgmt limit.
  DCGM_FI_DEV_GPU_TEMP, gauge, gpu temp.
  DCGM_FI_DEV_SM_CLOCK, gauge, sm clock.
  DCGM_FI_DEV_MAX_SM_CLOCK, gauge, max sm clock.
  DCGM_FI_DEV_MEM_CLOCK, gauge, mem clock.
  DCGM_FI_DEV_MAX_MEM_CLOCK, gauge, max mem clock.
kind: ConfigMap
metadata:
  annotations:
    meta.helm.sh/release-name: console-plugin-nvidia-gpu
    meta.helm.sh/release-namespace: nvidia-gpu-operator
  creationTimestamp: "2022-10-26T19:46:41Z"
lables:
  app.kubernetes.io/component: console-plugin-nvidia-gpu
  app.kubernetes.io/instance: console-plugin-nvidia-gpu
  app.kubernetes.io/managed-by: Helm
  app.kubernetes.io/name: console-plugin-nvidia-gpu
  app.kubernetes.io/part-of: console-plugin-nvidia-gpu
  app.kubernetes.io/version: latest
```
Install the ConfigMap and edit the NVIDIA Operator ClusterPolicy CR to add that ConfigMap in the DCGM exporter configuration. The installation of the ConfigMap is done by the new version of the Console Plugin NVIDIA GPU Helm Chart, but the ClusterPolicy CR editing is done by the user.

3. View the deployed resources:

```
$ oc -n nvidia-gpu-operator get all -l app.kubernetes.io/name=console-plugin-nvidia-gpu
```

**Example output**

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod/console-plugin-nvidia-gpu-7dc9c9sf5df-ztkxs</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>2m6s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>service/console-plugin-nvidia-gpu</td>
<td>ClusterIP</td>
<td>172.30.240.138</td>
<td>&lt;none&gt;</td>
<td>9443/TCP</td>
<td>2m6s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>UP-TO-DATE</th>
<th>AVAILABLE</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment.apps/console-plugin-nvidia-gpu</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>2m6s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NAME</th>
<th>DESIRED</th>
<th>CURRENT</th>
<th>READY</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>replicaset.apps/console-plugin-nvidia-gpu-7dc9c9sf5df</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2m6s</td>
</tr>
</tbody>
</table>

### 12.3. USING THE NVIDIA GPU ADMINISTRATION DASHBOARD

After deploying the OpenShift Console NVIDIA GPU plug-in, log in to the OpenShift Container Platform web console using your login credentials to access the **Administrator** perspective.

To view the changes, you need to refresh the console to see the **GPUs** tab under **Compute**.

#### 12.3.1. Viewing the cluster GPU overview

You can view the status of your cluster GPUs in the Overview page by selecting Overview in the Home section.

The Overview page provides information about the cluster GPUs, including:

- Details about the GPU providers
- Status of the GPUs
- Cluster utilization of the GPUs

#### 12.3.2. Viewing the GPUs dashboard

You can view the NVIDIA GPU administration dashboard by selecting GPUs in the Compute section of the OpenShift Console.
Charts on the GPUs dashboard include:

- **GPU utilization**: Shows the ratio of time the graphics engine is active and is based on the `DCGM_FI_PROF_GR_ENGINE_ACTIVE` metric.

- **Memory utilization**: Shows the memory being used by the GPU and is based on the `DCGM_FI_DEV_MEM_COPY_UTIL` metric.

- **Encoder utilization**: Shows the video encoder rate of utilization and is based on the `DCGM_FI_DEV_ENC_UTIL` metric.

- **Decoder utilization**: Shows the video decoder rate of utilization and is based on the `DCGM_FI_DEV_DEC_UTIL` metric.

- **Power consumption**: Shows the average power usage of the GPU in Watts and is based on the `DCGM_FI_DEV_POWER_USAGE` metric.

- **GPU temperature**: Shows the current GPU temperature and is based on the `DCGM_FI_DEV_GPU_TEMP` metric. The maximum is set to **110**, which is an empirical number, as the actual number is not exposed via a metric.

- **GPU clock speed**: Shows the average clock speed utilized by the GPU and is based on the `DCGM_FI_DEV_SM_CLOCK` metric.

- **Memory clock speed**: Shows the average clock speed utilized by memory and is based on the `DCGM_FI_DEV_MEM_CLOCK` metric.

### 12.3.3. Viewing the GPU Metrics

You can view the metrics for the GPUs by selecting the metric at the bottom of each GPU to view the Metrics page.

On the Metrics page, you can:

- Specify a refresh rate for the metrics
- Add, run, disable, and delete queries
- Insert Metrics
- Reset the zoom view
CHAPTER 13. MONITORING BARE-METAL EVENTS WITH THE BARE METAL EVENT RELAY

IMPORTANT

Bare Metal Event Relay 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 https://access.redhat.com/support/offerings/techpreview/.

13.1. ABOUT BARE-METAL EVENTS

Use the Bare Metal Event Relay to subscribe applications that run in your OpenShift Container Platform cluster to events that are generated on the underlying bare-metal host. The Redfish service publishes events on a node and transmits them on an advanced message queue to subscribed applications.

Bare-metal events are based on the open Redfish standard that is developed under the guidance of the Distributed Management Task Force (DMTF). Redfish provides a secure industry-standard protocol with a REST API. The protocol is used for the management of distributed, converged or software-defined resources and infrastructure.

Hardware-related events published through Redfish includes:

- Breaches of temperature limits
- Server status
- Fan status

Begin using bare-metal events by deploying the Bare Metal Event Relay Operator and subscribing your application to the service. The Bare Metal Event Relay Operator installs and manages the lifecycle of the Redfish bare-metal event service.

NOTE

The Bare Metal Event Relay works only with Redfish-capable devices on single-node clusters provisioned on bare-metal infrastructure.

13.2. HOW BARE-METAL EVENTS WORK

The Bare Metal Event Relay enables applications running on bare-metal clusters to respond quickly to Redfish hardware changes and failures such as breaches of temperature thresholds, fan failure, disk loss, power outages, and memory failure. These hardware events are delivered over a reliable low-latency transport channel based on Advanced Message Queuing Protocol (AMQP). The latency of the messaging service is between 10 to 20 milliseconds.

The Bare Metal Event Relay provides a publish-subscribe service for the hardware events, where multiple applications can use REST APIs to subscribe and consume the events. The Bare Metal Event Relay supports hardware that complies with Redfish OpenAPI v1.8 or higher.
13.2.1. Bare Metal Event Relay data flow

The following figure illustrates an example of bare-metal events data flow:

Figure 13.1. Bare Metal Event Relay data flow

13.2.1.1. Operator-managed pod

The Operator uses custom resources to manage the pod containing the Bare Metal Event Relay and its components using the `HardwareEvent` CR.

13.2.1.2. Bare Metal Event Relay

At startup, the Bare Metal Event Relay queries the Redfish API and downloads all the message registries, including custom registries. The Bare Metal Event Relay then begins to receive subscribed events from the Redfish hardware.

The Bare Metal Event Relay enables applications running on bare-metal clusters to respond quickly to Redfish hardware changes and failures such as breaches of temperature thresholds, fan failure, disk loss, power outages, and memory failure. The events are reported using the `HardwareEvent` CR.

13.2.1.3. Cloud native event

Cloud native events (CNE) is a REST API specification for defining the format of event data.

13.2.1.4. CNCF CloudEvents

`CloudEvents` is a vendor-neutral specification developed by the Cloud Native Computing Foundation (CNCF) for defining the format of event data.

13.2.1.5. AMQP dispatch router

The dispatch router is responsible for the message delivery service between publisher and subscriber. AMQP 1.0 qpid is an open standard that supports reliable, high-performance, fully-symmetrical messaging over the internet.

13.2.1.6. Cloud event proxy sidecar
The cloud event proxy sidecar container image is based on the ORAN API specification and provides a publish-subscribe event framework for hardware events.

13.2.2. Redfish message parsing service

In addition to handling Redfish events, the Bare Metal Event Relay provides message parsing for events without a Message property. The proxy downloads all the Redfish message registries including vendor specific registries from the hardware when it starts. If an event does not contain a Message property, the proxy uses the Redfish message registries to construct the Message and Resolution properties and add them to the event before passing the event to the cloud events framework. This service allows Redfish events to have smaller message size and lower transmission latency.

13.2.3. Installing the Bare Metal Event Relay using the CLI

As a cluster administrator, you can install the Bare Metal Event Relay Operator by using the CLI.

**Prerequisites**

- A cluster that is installed on bare-metal hardware with nodes that have a RedFish-enabled Baseboard Management Controller (BMC).
- Install the OpenShift CLI (oc).
- Log in as a user with cluster-admin privileges.

**Procedure**

1. Create a namespace for the Bare Metal Event Relay.
   
   a. Save the following YAML in the bare-metal-events-namespace.yaml file:

   ```yaml
   apiVersion: v1
   kind: Namespace
   metadata:
     name: openshift-bare-metal-events
     labels:
       name: openshift-bare-metal-events
       openshift.io/cluster-monitoring: "true"
   ```

   b. Create the Namespace CR:

   ```bash
   $ oc create -f bare-metal-events-namespace.yaml
   ```

2. Create an Operator group for the Bare Metal Event Relay Operator.
   
   a. Save the following YAML in the bare-metal-events-operatorgroup.yaml file:

   ```yaml
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: bare-metal-event-relay-group
     namespace: openshift-bare-metal-events
   spec:
     targetNamespaces:
       - openshift-bare-metal-events
   ```
b. Create the **OperatorGroup** CR:

```
$ oc create -f bare-metal-events-operatorgroup.yaml
```

3. Subscribe to the Bare Metal Event Relay.

a. Save the following YAML in the `bare-metal-events-sub.yaml` file:

```yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: bare-metal-event-relay-subscription
  namespace: openshift-bare-metal-events
spec:
  channel: "stable"
  name: bare-metal-event-relay
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. Create the **Subscription** CR:

```
$ oc create -f bare-metal-events-sub.yaml
```

**Verification**

To verify that the Bare Metal Event Relay Operator is installed, run the following command:

```
$ oc get csv -n openshift-bare-metal-events -o custom-columns=Name:.metadata.name,Phase:.status.phase
```

**Example output**

<table>
<thead>
<tr>
<th>Name</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>bare-metal-event-relay.4.11.0-xxxxxxxxxx</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>

**13.2.4. Installing the Bare Metal Event Relay using the web console**

As a cluster administrator, you can install the Bare Metal Event Relay Operator using the web console.

**Prerequisites**

- A cluster that is installed on bare-metal hardware with nodes that have a RedFish-enabled Baseboard Management Controller (BMC).
- Log in as a user with **cluster-admin** privileges.

**Procedure**

1. Install the Bare Metal Event Relay using the OpenShift Container Platform web console:
   
a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.

   b. Choose **Bare Metal Event Relay** from the list of available Operators, and then click **Install**.
c. On the Install Operator page, select or create a Namespace, select openshift-bare-metal-events, and then click Install.

**Verification**

Optional: You can verify that the Operator installed successfully by performing the following check:

1. Switch to the Operators → Installed Operators page.

2. Ensure that Bare Metal Event Relay is listed in the project with a Status of InstallSucceeded.

   **NOTE**

   During installation an Operator might display a Failed status. If the installation later succeeds with an InstallSucceeded message, you can ignore the Failed message.

If the operator does not appear as installed, to troubleshoot further:

- Go to the Operators → Installed Operators page and inspect the Operator Subscriptions and Install Plans tabs for any failure or errors under Status.

- Go to the Workloads → Pods page and check the logs for pods in the project namespace.

### 13.3. INSTALLING THE AMQ MESSAGING BUS

To pass Redfish bare-metal event notifications between publisher and subscriber on a node, you must install and configure an AMQ messaging bus to run locally on the node. You do this by installing the AMQ Interconnect Operator for use in the cluster.

**Prerequisites**

- Install the OpenShift Container Platform CLI (oc).

- Log in as a user with cluster-admin privileges.

**Procedure**

- Install the AMQ Interconnect Operator to its own amq-interconnect namespace. See Installing the AMQ Interconnect Operator.

**Verification**

1. Verify that the AMQ Interconnect Operator is available and the required pods are running:

   ```
   $ oc get pods -n amq-interconnect
   NAME                                    READY   STATUS    RESTARTS   AGE
   amq-interconnect-645db76c76-k8ghs       1/1     Running   0          23h
   interconnect-operator-5cb5fc7cc-4v7qm   1/1     Running   0          23h
   ```
2. Verify that the required `bare-metal-event-relay` bare-metal event producer pod is running in the `openshift-bare-metal-events` namespace:

```bash
$ oc get pods -n openshift-bare-metal-events
```

**Example output**

```
NAME                                                            READY   STATUS    RESTARTS   AGE
hw-event-proxy-operator-controller-manager-74d5649b7c-dzgtl     2/2     Running   0 25s
```

### 13.4. SUBSCRIBING TO REDFISH BMC BARE-METAL EVENTS FOR A CLUSTER NODE

As a cluster administrator, you can subscribe to Redfish BMC events generated on a node in your cluster by creating a `BMCEventSubscription` custom resource (CR) for the node, creating a `HardwareEvent` CR for the event, and a `Secret` CR for the BMC.

#### 13.4.1. Subscribing to bare-metal events

You can configure the baseboard management controller (BMC) to send bare-metal events to subscribed applications running in an OpenShift Container Platform cluster. Example Redfish bare-metal events include an increase in device temperature, or removal of a device. You subscribe applications to bare-metal events using a REST API.

**IMPORTANT**

You can only create a `BMCEventSubscription` custom resource (CR) for physical hardware that supports Redfish and has a vendor interface set to `redfish` or `idrac-redfish`.

**NOTE**

Use the `BMCEventSubscription` CR to subscribe to predefined Redfish events. The Redfish standard does not provide an option to create specific alerts and thresholds. For example, to receive an alert event when an enclosure’s temperature exceeds 40° Celsius, you must manually configure the event according to the vendor’s recommendations.

Perform the following procedure to subscribe to bare-metal events for the node using a `BMCEventSubscription` CR.

**Prerequisites**

- Install the OpenShift CLI (`oc`).
- Log in as a user with `cluster-admin` privileges.
- Get the user name and password for the BMC.
- Deploy a bare-metal node with a Redfish-enabled Baseboard Management Controller (BMC) in your cluster, and enable Redfish events on the BMC.
NOTE

Enabling Redfish events on specific hardware is outside the scope of this information. For more information about enabling Redfish events for your specific hardware, consult the BMC manufacturer documentation.

Procedure

1. Confirm that the node hardware has the Redfish EventService enabled by running the following curl command:

   ```
   curl https://<bmc_ip_address>/redfish/v1/EventService --insecure -H 'Content-Type: application/json' -u "<bmc_username>:<password>"
   ```

   where:

   **bmc_ip_address**

   is the IP address of the BMC where the Redfish events are generated.

Example output

```json
{
   "@odata.context": "/redfish/v1/$metadata#EventService" EventService",
   "@odata.id": "/redfish/v1/EventService",
   "@odata.type": ">#EventService.v1_0_2.EventService",
   "Actions": {
      "#EventService.SubmitTestEvent": { 
      "EventType@Redfish.AllowableValues": ["StatusChange", "ResourceUpdated",
      "ResourceAdded", "ResourceRemoved", "Alert"],
      "target": "/redfish/v1/EventService/Actions/EventService.SubmitTestEvent"
   },
   "DeliveryRetryAttempts": 3,
   "DeliveryRetryIntervalSeconds": 30,
   "Description": "Event Service represents the properties for the service",
   "EventTypesForSubscription": ["StatusChange", "ResourceUpdated", "ResourceAdded",
   "ResourceRemoved", "Alert"],
   "EventTypesForSubscription@odata.count": 5,
   "Id": "EventService",
   "Name": "Event Service",
   "ServiceEnabled": true,
   "Status": {
      "Health": "OK",
      "HealthRollup": "OK",
      "State": "Enabled"
   },
   "Subscriptions": {
      "@odata.id": "/redfish/v1/EventService/Subscriptions"
   }
}
```

2. Get the Bare Metal Event Relay service route for the cluster by running the following command:

```
$ oc get route -n openshift-bare-metal-events
```
Example output

```
NAME             HOST/PORT                                                                                           PATH
SERVICES                 PORT   TERMINATION   WILDCARD
hw-event-proxy   hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com
hw-event-proxy-service   9087   edge          None
```

3. Create a **BMCEventSubscription** resource to subscribe to the Redfish events:
   a. Save the following YAML in the `bmc_sub.yaml` file:

   ```yaml
   apiVersion: metal3.io/v1alpha1
   kind: BMCEventSubscription
   metadata:
     name: sub-01
     namespace: openshift-machine-api
   spec:
     hostName: <hostname> ①
     destination: <proxy_service_url> ②
     context: ""
   ```

   ① Specifies the name or UUID of the worker node where the Redfish events are generated.

   ② Specifies the bare-metal event proxy service, for example, [https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook](https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook).

   b. Create the **BMCEventSubscription** CR:

   ```
   $ oc create -f bmc_sub.yaml
   ```

4. Optional: To delete the BMC event subscription, run the following command:

   ```
   $ oc delete -f bmc_sub.yaml
   ```

5. Optional: To manually create a Redfish event subscription without creating a **BMCEventSubscription** CR, run the following `curl` command, specifying the BMC username and password.

   ```
   ```

   where:

   - **proxy_service_url**
     is the bare-metal event proxy service, for example, [https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook](https://hw-event-proxy-openshift-bare-metal-events.apps.compute-1.example.com/webhook).

   - **bmc_ip_address**
     is the IP address of the BMC where the Redfish events are generated.
13.4.2. Querying Redfish bare-metal event subscriptions with curl

Some hardware vendors limit the amount of Redfish hardware event subscriptions. You can query the number of Redfish event subscriptions by using curl.

Prerequisites

- Get the user name and password for the BMC.
- Deploy a bare-metal node with a Redfish-enabled Baseboard Management Controller (BMC) in your cluster, and enable Redfish hardware events on the BMC.

Procedure

1. Check the current subscriptions for the BMC by running the following curl command:

   ```
   $ curl --globoff -H "Content-Type: application/json" -k -X GET --user <bmc_username>:<password> https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions
   ```

   where:

   - **bmc_ip_address** is the IP address of the BMC where the Redfish events are generated.

Example output

```bash
% Total % Received % Xferd Average Speed Time Time Time Current Dload Upload Total Spent Left Speed
100 435 100 435 0 0 399 0 0:00:01 0:00:01 --:--:-- 399
{
  "@odata.context": "/redfish/v1/$metadata#EventDestinationCollection.EventDestinationCollection",
  "@odata.etag": ""1651137375"
}
```
In this example, a single subscription is configured: /redfish/v1/EventService/Subscriptions/1.

2. Optional: To remove the /redfish/v1/EventService/Subscriptions/1 subscription with curl, run the following command, specifying the BMC username and password:

   ```bash
   $ curl --globoff -L -w "%{http_code} %{url_effective}\n" -k -u <bmc_username>:<password >-
   H "Content-Type: application/json" -d '{}' -X DELETE
   https://<bmc_ip_address>/redfish/v1/EventService/Subscriptions/1
   ```

   where:

   `bmc_ip_address`

   is the IP address of the BMC where the Redfish events are generated.

13.4.3. Creating the bare-metal event and Secret CRs

To start using bare-metal events, create the **HardwareEvent** custom resource (CR) for the host where the Redfish hardware is present. Hardware events and faults are reported in the **hw-event-proxy** logs.

**Prerequisites**

- Install the OpenShift CLI (`oc`).
- Log in as a user with `cluster-admin` privileges.
- Install the Bare Metal Event Relay.
- Create a **BMCEventSubscription** CR for the BMC Redfish hardware.

**NOTE**

Multiple **HardwareEvent** resources are not permitted.

**Procedure**

1. Create the **HardwareEvent** custom resource (CR):
   a. Save the following YAML in the **hw-event.yaml** file:

   ```yaml
   apiVersion: "event.redhat-cne.org/v1alpha1"
   kind: "HardwareEvent"
   metadata:
       name: "hardware-event"
   spec:
   ```
nodeSelector:
  node-role.kubernetes.io/hw-event: ""
transportHost: "amqp://amq-router-service-name.amq-namespace.svc.cluster.local"
logLevel: "debug"
magParserTimeout: "10"

1. Required. Use the nodeSelector field to target nodes with the specified label, for example, node-role.kubernetes.io/hw-event: "".

2. Required. AMQP host that delivers the events at the transport layer using the AMQP protocol.

3. Optional. The default value is debug. Sets the log level in hw-event-proxy logs. The following log levels are available: fatal, error, warning, info, debug, trace.

4. Optional. Sets the timeout value in milliseconds for the Message Parser. If a message parsing request is not responded to within the timeout duration, the original hardware event message is passed to the cloud native event framework. The default value is 10.

b. Create the HardwareEvent CR:

   $ oc create -f hardware-event.yaml

2. Create a BMC username and password Secret CR that enables the hardware events proxy to access the Redfish message registry for the bare-metal host.

   a. Save the following YAML in the hw-event-bmc-secret.yaml file:

```yaml
apiVersion: v1
class: Secret
metadata:
  name: redfish-basic-auth
type: Opaque
stringData:
  username: <bmc_username>
  password: <bmc_password>
  hostaddr: <bmc_host_ip_address>
```

   1. Enter plain text values for the various items under stringData.

   b. Create the Secret CR:

      $ oc create -f hw-event-bmc-secret.yaml

### 13.5. SUBSCRIBING APPLICATIONS TO BARE-METAL EVENTS REST API REFERENCE

Use the bare-metal events REST API to subscribe an application to the bare-metal events that are generated on the parent node.
Subscribe applications to Redfish events by using the resource address
/cluster/node/<node_name>/redfish/event, where <node_name> is the cluster node running the
application.

Deploy your cloud-event-consumer application container and cloud-event-proxy sidecar container in
a separate application pod. The cloud-event-consumer application subscribes to the cloud-event-
proxy container in the application pod.

Use the following API endpoints to subscribe the cloud-event-consumer application to Redfish events
posted by the cloud-event-proxy container at http://localhost:8089/api/cloudNotifications/v1/ in the
application pod:

- /api/cloudNotifications/v1/subscriptions
  - POST: Creates a new subscription
  - GET: Retrieves a list of subscriptions
- /api/cloudNotifications/v1/subscriptions/<subscription_id>
  - GET: Returns details for the specified subscription ID
- api/cloudNotifications/v1/subscriptions/status/<subscription_id>
  - PUT: Creates a new status ping request for the specified subscription ID
- /api/cloudNotifications/v1/health
  - GET: Returns the health status of cloudNotifications API

**NOTE**

9089 is the default port for the cloud-event-consumer container deployed in the
application pod. You can configure a different port for your application as required.

api/cloudNotifications/v1/subscriptions
HTTP method
GET api/cloudNotifications/v1/subscriptions

Description
Returns a list of subscriptions. If subscriptions exist, a 200 OK status code is returned along with the list
of subscriptions.

Example API response

```
[
  {
    "id": "ca11ab76-86f9-428c-8d3a-666c24e34d32",
    "endpointUri": "http://localhost:9089/api/cloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:8089/api/cloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-8d3a-666c24e34d32",
    "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
  }
]
```

HTTP method
POST api/cloudNotifications/v1/subscriptions

Description
Creates a new subscription. If a subscription is successfully created, or if it already exists, a **201 Created** status code is returned.

Table 13.1. Query parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscription</td>
<td>data</td>
</tr>
</tbody>
</table>

Example payload

```json
{
    "uriLocation": "http://localhost:8089/api/cloudNotifications/v1/subscriptions",
    "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/cloudNotifications/v1/subscriptions/<subscription_id>

HTTP method
GET

Description
Returns details for the subscription with ID `<subscription_id>`

Table 13.2. Query parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;subscription_id&gt;</td>
<td>string</td>
</tr>
</tbody>
</table>

Example API response

```json
{
    "id": "ca11ab76-86f9-428c-8d3a-666c24e34d32",
    "endpointUri": "http://localhost:9089/api/cloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:8089/api/cloudNotifications/v1/subscriptions/ca11ab76-86f9-428c-8d3a-666c24e34d32",
    "resource": "/cluster/node/openshift-worker-0.openshift.example.com/redfish/event"
}
```

api/cloudNotifications/v1/subscriptions/status/<subscription_id>

HTTP method
PUT

Description
Creates a new status ping request for subscription with ID `<subscription_id>`. If a subscription is present, the status request is successful and a **202 Accepted** status code is returned.

Table 13.3. Query parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;subscription_id&gt;</code></td>
<td>string</td>
</tr>
</tbody>
</table>

Example API response

```json
{"status":"ping sent"}
```

/api/cloudNotifications/v1/health/

HTTP method
GET /api/cloudNotifications/v1/health/

Description
Returns the health status for the cloudNotifications REST API.

Example API response

OK
CHAPTER 14. ACCESSING THIRD-PARTY MONITORING APIS

In OpenShift Container Platform 4.11, you can access web service APIs for some third-party monitoring components from the command line interface (CLI).

14.1. ACCESSING THIRD-PARTY MONITORING WEB SERVICE APIS

You can directly access third-party web service APIs from the command line for the following monitoring stack components: Prometheus, Alertmanager, Thanos Ruler, and Thanos Querier.

The following example commands show how to query the service API receivers for Alertmanager. This example requires that the associated user account be bound against the `monitoring-alertmanager-edit` role in the `openshift-monitoring` namespace and that the account has the privilege to view the route. This access only supports using a Bearer Token for authentication.

```
$ oc login -u <username> -p <password>
$ host=$(oc -n openshift-monitoring get route alertmanager-main -o jsonpath={.spec.host})
$ token=$(oc whoami -t)
$ curl -H "Authorization: Bearer $token" -k "https://$host/api/v2/receivers"
```

**NOTE**

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

14.2. QUERYING METRICS BY USING THE FEDERATION ENDPOINT FOR PROMETHEUS

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

```bash
$ oc login -u <username> -p <password>
$ host=$(oc -n openshift-monitoring get route alertmanager-main -o jsonpath={.spec.host})
$ token=$(oc whoami -t)
$ curl -H "Authorization: Bearer $token" -k "https://$host/api/v2/receivers"
```
**WARNING**

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. 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 querying the federation endpoint frequently. 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 obtained the host URL for the OpenShift Container Platform route.

- You have access to the cluster as a user with the *cluster-monitoring-view* 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 federation endpoint.

**Procedure**

1. Retrieve the bearer token:

   ```bash
   $ token=`oc whoami -t`
   
2. Query metrics from the */federate* route. The following example queries *up* metrics:

   ```bash
   ```

   For `<federation_host>`, substitute the host URL for the federation route.

**Example output**
14.3. ADDITIONAL RESOURCES

- Configuring remote write storage
- Managing metrics
- Managing alerts
CHAPTER 15. TROUBLESHOOTING MONITORING ISSUES

15.1. INVESTIGATING WHY USER-DEFINED METRICS ARE UNAVAILABLE

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

Prerequisites

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

Procedure

1. Check that the corresponding labels match in the service and ServiceMonitor resource configurations.
   a. Obtain the label defined in the service. The following example queries the prometheus-example-app service in the ns1 project:

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

   Example output

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

   b. Check that the matchLabels app label in the ServiceMonitor resource configuration matches the label output in the preceding step:

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

   Example output

   ```yaml
   spec:
   endpoints:
   - interval: 30s
     port: web
     scheme: http
     selector:
     matchLabels:
     app: prometheus-example-app
   ```
NOTE

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

2. Inspect the logs for the Prometheus Operator in the openshift-user-workload-monitoring project.
   a. List the pods in the openshift-user-workload-monitoring project:

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

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>prometheus-operator-776fcbbd56-2nbfm</td>
<td>2/2</td>
<td>Running</td>
<td>0</td>
<td>132m</td>
</tr>
<tr>
<td>prometheus-user-workload-0</td>
<td>5/5</td>
<td>Running</td>
<td>1</td>
<td>132m</td>
</tr>
<tr>
<td>prometheus-user-workload-1</td>
<td>5/5</td>
<td>Running</td>
<td>1</td>
<td>132m</td>
</tr>
<tr>
<td>thanos-ruler-user-workload-0</td>
<td>3/3</td>
<td>Running</td>
<td>0</td>
<td>132m</td>
</tr>
<tr>
<td>thanos-ruler-user-workload-1</td>
<td>3/3</td>
<td>Running</td>
<td>0</td>
<td>132m</td>
</tr>
</tbody>
</table>

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

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

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

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

3. Review the target status for your endpoint on the Metrics targets page in the OpenShift Container Platform web console UI.
   a. Log in to the OpenShift Container Platform web console and navigate to Observe → Targets in the Administrator perspective.
   b. Locate the metrics endpoint in the list, and review the status of the target in the Status column.
   c. If the Status is Down, click the URL for the endpoint to view more information on the Target Details page for that metrics target.

4. Configure debug level logging for the Prometheus Operator in the openshift-user-workload-monitoring project.
   a. Edit the user-workload-monitoring-config ConfigMap object in the openshift-user-workload-monitoring project:
b. Add `logLevel: debug` for `prometheusOperator` under `data/config.yaml` to set the log level to `debug`:

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

c. Save the file to apply the changes.

**NOTE**

The `prometheus-operator` in the `openshift-user-workload-monitoring` project restarts automatically when you apply the log-level change.

d. Confirm that the `debug` log-level has been applied to the `prometheus-operator` deployment in the `openshift-user-workload-monitoring` project:

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

**Example output**

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

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

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

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

**NOTE**

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

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

**Additional resources**

- Creating a user-defined workload monitoring config map
• See Specifying how a service is monitored for details on how to create a ServiceMonitor or PodMonitor resource

• See Accessing metrics targets in the Administrator perspective

15.2. 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 number of scrape samples that are being collected.

• Check the time series database (TSDB) status using the Prometheus HTTP API for more information about which labels are creating the most time series. Doing so requires cluster administrator privileges.

• Reduce the number of unique time series that are created by reducing the number of unbound attributes that are assigned to user-defined metrics.

NOTE

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

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

Prerequisites

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

• You have installed the OpenShift CLI (oc).

Procedure

1. In the Administrator perspective, navigate to Observe → Metrics.

2. Run the following Prometheus Query Language (PromQL) query in the Expression field. This returns the ten metrics that have the highest number of scrape samples:

   \[\text{topk(10, count by (job)({\_name\_=~".\+"}))}\]

3. Investigate the number of unbound label values assigned to metrics with higher than expected scrape sample counts.
- If the metrics relate to a user-defined project, review the metrics key-value pairs assigned to your workload. These are implemented through Prometheus client libraries at the application level. Try to limit the number of unbound attributes referenced in your labels.

- If the metrics relate to a core OpenShift Container Platform project, create a Red Hat support case on the Red Hat Customer Portal.

4. Review the TSDB status using the Prometheus HTTP API by running the following commands as a cluster administrator:

```bash
$ oc login -u <username> -p <password>
$ host=$(oc -n openshift-monitoring get route prometheus-k8s -o jsonpath={.spec.host})
$ token=$(oc whoami -t)
$ curl -H "Authorization: Bearer $token" -k https://$host/api/v1/status/tsdb"
```

**Example output**

```
"status": "success",
```

**Additional resources**

- See [Setting a scrape sample limit for user-defined projects](setting-a-scrape-sample-limit-for-user-defined-projects) for details on how to set a scrape sample limit and create related alerting rules

- [Submitting a support case](submitting-a-support-case)
CHAPTER 16. CONFIG MAP REFERENCE FOR THE CLUSTER MONITORING OPERATOR

16.1. CLUSTER MONITORING OPERATOR CONFIGURATION REFERENCE

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

- To configure monitoring components, edit the ConfigMap object named `cluster-monitoring-config` in the `openshift-monitoring` namespace. These configurations are defined by `ClusterMonitoringConfiguration`.

- To configure monitoring components that monitor user-defined projects, edit the ConfigMap object named `user-workload-monitoring-config` in the `openshift-user-workload-monitoring` namespace. These configurations are defined by `UserWorkloadConfiguration`.

The configuration file is always defined under the `config.yaml` key in the config map data.

**NOTE**

- Not all configuration parameters are exposed.
- 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.

16.2. ADDITIONALALERTMANAGERCONFIG

16.2.1. Description

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

16.2.2. Required

- `apiVersion`

Appears in: `PrometheusK8sConfig, PrometheusRestrictedConfig, ThanosRulerConfig`

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

### 16.3. ALERTMANAGERMAINCONFIG

#### 16.3.1. Description

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

Appears in: `ClusterMonitoringConfiguration`
### Property

**enabled**

_type_: bool

_description_: 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`. 

### logLevel

_type_: string

_description_: Defines the log level setting for Alertmanager. The possible values are: `error`, `warn`, `info`, `debug`. The default value is `info`.

### nodeSelector

_type_: map[string]string

_description_: Defines the nodes on which the Pods are scheduled.

### resources

_type_: *v1.ResourceRequirements

_description_: Defines resource requests and limits for the Alertmanager container.

### tolerations

_type_: []v1.Toleration

_description_: Defines tolerations for the pods.

### volumeClaimTemplate

_type_: *monv1.EmbeddedPersistentVolumeClaim

_description_: Defines persistent storage for Alertmanager. Use this setting to configure the persistent volume claim, including storage class, volume size, and name.

## 16.4. ALERTMANAGERUSERWORKLOADCONFIG

### 16.4.1. Description

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

Appears in: **UserWorkloadConfiguration**
<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableAlertmanagerConfig</td>
<td>bool</td>
<td>A Boolean flag to enable or disable user-defined namespaces to be selected for AlertmanagerConfig lookup. The default value is false.</td>
</tr>
<tr>
<td>logLevel</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>resources</td>
<td>*v1.ResourceRequirements</td>
<td>Defines resource requests and limits for the Alertmanager container.</td>
</tr>
<tr>
<td>nodeSelector</td>
<td>map[string]string</td>
<td>Defines the nodes on which the pods are scheduled.</td>
</tr>
<tr>
<td>tolerations</td>
<td>[]v1.Toleration</td>
<td>Defines tolerations for the pods.</td>
</tr>
<tr>
<td>volumeClaimTemplate</td>
<td>*monv1.EmbeddedPersistentVolumeClaim</td>
<td>Defines persistent storage for Alertmanager. Use this setting to configure the persistent volume claim, including storage class, volume size and name.</td>
</tr>
</tbody>
</table>

### 16.5. CLUSTERMONITORINGCONFIGURATION

#### 16.5.1. 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alertmanagerMain</td>
<td>*AlertmanagerMainConfig</td>
<td>AlertmanagerMainConfig defines settings for the Alertmanager component in the openshift-monitoring namespace.</td>
</tr>
<tr>
<td>enableUserWorkload</td>
<td>*bool</td>
<td>UserWorkloadEnabled is a Boolean flag that enables monitoring for user-defined projects.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>k8sPrometheusAdapter</td>
<td>*K8sPrometheusAdapter</td>
<td>K8sPrometheusAdapter defines settings for the Prometheus Adapter component.</td>
</tr>
<tr>
<td>kubeStateMetrics</td>
<td>*KubeStateMetricsConfig</td>
<td>KubeStateMetricsConfig defines settings for the kube-state-metrics agent.</td>
</tr>
<tr>
<td>prometheusK8s</td>
<td>*PrometheusK8sConfig</td>
<td>PrometheusK8sConfig defines settings for the Prometheus component.</td>
</tr>
<tr>
<td>openshiftStateMetrics</td>
<td>*OpenShiftStateMetricsConfig</td>
<td>OpenShiftMetricsConfig defines settings for the openshift-state-metrics agent.</td>
</tr>
<tr>
<td>telemeterClient</td>
<td>*TelemeterClientConfig</td>
<td>TelemeterClientConfig defines settings for the Telemeter Client component.</td>
</tr>
<tr>
<td>thanosQuerier</td>
<td>*ThanosQuerierConfig</td>
<td>ThanosQuerierConfig defines settings for the Thanos Querier component.</td>
</tr>
</tbody>
</table>

### 16.6. DEDICATEDSERVICEMONITORS

#### 16.6.1. Description

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

**Appears in:** K8sPrometheusAdapter
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.

### 16.7. K8SPROMETHEUSADAPTER

#### 16.7.1. Description

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

Appears in: **ClusterMonitoringConfiguration**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>audit</code></td>
<td>*Audit</td>
<td>Defines the audit configuration used by the Prometheus Adapter instance. Possible profile values are: <code>metadata</code>, <code>request</code>, <code>requestresponse</code>, and <code>none</code>. The default value is <code>metadata</code>.</td>
</tr>
<tr>
<td><code>nodeSelector</code></td>
<td>map[string]string</td>
<td>Defines the nodes on which the pods are scheduled.</td>
</tr>
<tr>
<td><code>tolerations</code></td>
<td>[]v1.Toleration</td>
<td>Defines tolerations for the pods.</td>
</tr>
<tr>
<td><code>dedicatedServiceMonitors</code></td>
<td>*DedicatedServiceMonitors</td>
<td>Defines dedicated service monitors.</td>
</tr>
</tbody>
</table>

### 16.8. KUBESTATEMETRICSCONFIG
16.8.1. Description

The KubeStateMetricsConfig resource defines settings for the kube-state-metrics agent.

Appears in: ClusterMonitoringConfiguration

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeSelector</td>
<td>map[string]string</td>
<td>Defines the nodes on which the pods are scheduled.</td>
</tr>
<tr>
<td>tolerations</td>
<td>[]v1.Toleration</td>
<td>Defines tolerations for the pods.</td>
</tr>
</tbody>
</table>

16.9. OPENSHIFTSTATEMETRICSCONFIG

16.9.1. Description

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

Appears in: ClusterMonitoringConfiguration

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodeSelector</td>
<td>map[string]string</td>
<td>Defines the nodes on which the pods are scheduled.</td>
</tr>
<tr>
<td>tolerations</td>
<td>[]v1.Toleration</td>
<td>Defines tolerations for the pods.</td>
</tr>
</tbody>
</table>

16.10. PROMETHEUSK8SCONFIG

16.10.1. Description

The PrometheusK8sConfig resource defines settings for the Prometheus component.

Appears in: ClusterMonitoringConfiguration

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalAlertmanagerConfigfs</td>
<td>[]AdditionalAlertmanagerConfig</td>
<td>Configures additional Alertmanager instances that receive alerts from the Prometheus component. By default, no additional Alertmanager instances are configured.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enforcedBodySizeLimit</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>externalLabels</td>
<td>map[string]string</td>
<td>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.</td>
</tr>
<tr>
<td>logLevel</td>
<td>string</td>
<td>Defines the log level setting for Prometheus. The possible values are: error, warn, info, and debug. The default value is info.</td>
</tr>
<tr>
<td>nodeSelector</td>
<td>map[string]string</td>
<td>Defines the nodes on which the pods are scheduled.</td>
</tr>
<tr>
<td>queryLogFile</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
### remoteWrite

- **Property**: remoteWrite
- **Type**: []RemoteWriteSpec
- **Description**: Defines the remote write configuration, including URL, authentication, and relabeling settings.

### resources

- **Property**: resources
- **Type**: *v1.ResourceRequirements
- **Description**: Defines resource requests and limits for the Prometheus container.

### retention

- **Property**: retention
- **Type**: string
- **Description**: 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

- **Property**: retentionSize
- **Type**: string
- **Description**: 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

- **Property**: tolerations
- **Type**: []v1.Toleration
- **Description**: Defines tolerations for the pods.

### volumeClaimTemplate

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

### 16.11. PROMETHEUSOPERATORCONFIG

#### 16.11.1. Description

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

**Appears in**: ClusterMonitoringConfiguration, UserWorkloadConfiguration
logLevel

Defines the log level settings for Prometheus Operator. The possible values are `error`, `warn`, `info`, and `debug`. The default value is `info`.

nodeSelector

Defines the nodes on which the pods are scheduled.

tolerations

Defines tolerations for the pods.

16.12. PROMETHEUSRESTRICTEDCONFIG

16.12.1. Description

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

Appears in: `UserWorkloadConfiguration`

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalAlertmanagerConfigs</td>
<td><code>[]AdditionalAlertmanagerConfig</code></td>
<td>Configures additional Alertmanager instances that receive alerts from the Prometheus component. By default, no additional Alertmanager instances are configured.</td>
</tr>
<tr>
<td>enforcedLabelLimit</td>
<td><code>*uint64</code></td>
<td>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.</td>
</tr>
<tr>
<td>enforcedLabelNameLengthLimit</td>
<td><code>*uint64</code></td>
<td>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.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enforcedLabelValueLengthLimit</td>
<td>*uint64</td>
<td>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.</td>
</tr>
<tr>
<td>enforcedSampleLimit</td>
<td>*uint64</td>
<td>Specifies a global limit on the number of scraped samples that will be accepted. This setting overrides the <code>SampleLimit</code> value set in any user-defined <code>ServiceMonitor</code> or <code>PodMonitor</code> object if the value is greater than <code>enforcedTargetLimit</code>. 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.</td>
</tr>
<tr>
<td>enforcedTargetLimit</td>
<td>*uint64</td>
<td>Specifies a global limit on the number of scraped targets. This setting overrides the <code>TargetLimit</code> value set in any user-defined <code>ServiceMonitor</code> or <code>PodMonitor</code> object if the value is greater than <code>enforcedSampleLimit</code>. Administrators can use this setting to keep the overall number of targets under control. The default value is 0.</td>
</tr>
<tr>
<td>externalLabels</td>
<td>map[string]string</td>
<td>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.</td>
</tr>
<tr>
<td>logLevel</td>
<td>string</td>
<td>Defines the log level setting for Prometheus. The possible values are <code>error</code>, <code>warn</code>, <code>info</code>, and <code>debug</code>. The default setting is <code>info</code>.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>nodeSelector</td>
<td>map[string]string</td>
<td>Defines the nodes on which the pods are scheduled.</td>
</tr>
<tr>
<td>queryLogFile</td>
<td>string</td>
<td>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 <code>/var/log/prometheus</code>, or a full path to a location where an emptyDir volume will be mounted and the queries saved. Writing to <code>/dev/stderr</code>, <code>/dev/stdout</code> or <code>/dev/null</code> is supported, but writing to any other <code>/dev/</code> path is not supported. Relative paths are also not supported. By default, PromQL queries are not logged.</td>
</tr>
<tr>
<td>remoteWrite</td>
<td>[]RemoteWriteSpec</td>
<td>Defines the remote write configuration, including URL, authentication, and relabeling settings.</td>
</tr>
<tr>
<td>resources</td>
<td>*v1.ResourceRequirements</td>
<td>Defines resource requests and limits for the Prometheus container.</td>
</tr>
<tr>
<td>retention</td>
<td>string</td>
<td>Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: [0-9]+(ms</td>
</tr>
<tr>
<td>retentionSize</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>tolerations</td>
<td>[]v1.Toleration</td>
<td>Defines tolerations for the pods.</td>
</tr>
</tbody>
</table>
volumeClaimTemplate | *monv1.EmbeddedPersistentVolumeClaim | Defines persistent storage for Prometheus. Use this setting to configure the storage class and size of a volume.

### 16.13. REMOTEWRITESPEC

#### 16.13.1. Description

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

#### 16.13.2. Required

- url

Appears in: PrometheusK8sConfig, PrometheusRestrictedConfig

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authorization</td>
<td>*monv1.SafeAuthorization</td>
<td>Defines the authorization settings for remote write storage.</td>
</tr>
<tr>
<td>basicAuth</td>
<td>*monv1.BasicAuth</td>
<td>Defines basic authentication settings for the remote write endpoint URL.</td>
</tr>
<tr>
<td>bearerTokenFile</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>headers</td>
<td>map[string]string</td>
<td>Specifies the custom HTTP headers to be sent along with each remote write request. Headers set by Prometheus cannot be overwritten.</td>
</tr>
<tr>
<td>metadataConfig</td>
<td>*monv1.MetadataConfig</td>
<td>Defines settings for sending series metadata to remote write storage.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.
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.

16.14. TELEMETERCLIENTCONFIG

16.14.1. Description

The TelemeterClientConfig resource defines settings for the telemeter-client component.

16.14.2. Required

- nodeSelector
- tolerations

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.
### 16.15. THANOSQUERIERCONFIG

#### 16.15.1. Description

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

Appears in: [ClusterMonitoringConfiguration](#)

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableRequestLogging</td>
<td>bool</td>
<td>A Boolean flag that enables or disables request logging. The default value is <strong>false</strong>.</td>
</tr>
<tr>
<td>logLevel</td>
<td>string</td>
<td>Defines the log level setting for Thanos Querier. The possible values are <strong>error</strong>, <strong>warn</strong>, <strong>info</strong>, and <strong>debug</strong>. The default value is <strong>info</strong>.</td>
</tr>
<tr>
<td>nodeSelector</td>
<td>map[string]string</td>
<td>Defines the nodes on which the pods are scheduled.</td>
</tr>
<tr>
<td>resources</td>
<td>*v1.ResourceRequirements</td>
<td>Defines resource requests and limits for the Thanos Querier container.</td>
</tr>
<tr>
<td>tolerations</td>
<td>[]v1.Toleration</td>
<td>Defines tolerations for the pods.</td>
</tr>
</tbody>
</table>

### 16.16. THANOSRULERCONFIG

#### 16.16.1. Description

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

Appears in: [UserWorkloadConfiguration](#)

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalAlertmanagerConfigs</td>
<td>[]AdditionalAlertmanagerConfig</td>
<td>Configures how the Thanos Ruler component communicates with additional Alertmanager instances. The default value is <strong>nil</strong>.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>logLevel</td>
<td>string</td>
<td>Defines the log level setting for Thanos Ruler. The possible values are <strong>error, warn, info, and debug</strong>. The default value is <strong>info</strong>.</td>
</tr>
<tr>
<td>nodeSelector</td>
<td>map[string]string</td>
<td>Defines the nodes on which the Pods are scheduled.</td>
</tr>
<tr>
<td>resources</td>
<td>*v1.ResourceRequirements</td>
<td>Defines resource requests and limits for the Alertmanager container.</td>
</tr>
<tr>
<td>retention</td>
<td>string</td>
<td>Defines the duration for which Prometheus retains data. This definition must be specified using the following regular expression pattern: [[0-9]+(ms</td>
</tr>
<tr>
<td>tolerations</td>
<td>[]v1.Toleration</td>
<td>Defines tolerations for the pods.</td>
</tr>
<tr>
<td>volumeClaimTemplate</td>
<td>*monv1.EmbeddedPersistentVolumeClaim</td>
<td>Defines persistent storage for Thanos Ruler. Use this setting to configure the storage class and size of a volume.</td>
</tr>
</tbody>
</table>

### 16.17. TLSCONFIG

#### 16.17.1. Description

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

#### 16.17.2. 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.

### 16.18. USERWORKLOADCONFIGURATION

#### 16.18.1. 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.
<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>thanosRuler</td>
<td><code>*ThanosRulerConfig</code></td>
<td>Defines the settings for the Thanos Ruler component in user workload monitoring.</td>
</tr>
</tbody>
</table>