



OpenShift Container Platform 4.5

Monitoring

Configuring and using the monitoring stack in OpenShift Container Platform

OpenShift Container Platform 4.5 Monitoring

Configuring and using the monitoring stack in OpenShift Container Platform

Legal Notice

Copyright © 2020 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

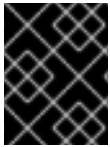
CHAPTER 1. CLUSTER MONITORING	4
1.1. ABOUT CLUSTER MONITORING	4
1.1.1. Stack components and monitored targets	4
1.1.2. Next steps	6
1.2. CONFIGURING THE MONITORING STACK	6
1.2.1. Prerequisites	6
1.2.2. Maintenance and support	6
1.2.3. Creating cluster monitoring ConfigMap	7
1.2.4. Configuring the cluster monitoring stack	7
1.2.5. Configurable monitoring components	8
1.2.6. Moving monitoring components to different nodes	9
1.2.7. Assigning tolerations to monitoring components	10
1.2.8. Configuring persistent storage	11
1.2.9. Prerequisites	11
1.2.9.1. Configuring a local persistent volume claim	12
1.2.9.2. Modifying retention time for Prometheus metrics data	13
1.2.10. Configuring Alertmanager	14
1.2.10.1. Alertmanager default configuration	14
1.2.10.2. Applying custom Alertmanager configuration	15
1.2.10.3. Alerting rules	16
1.2.10.4. Listing acting alerting rules	17
1.2.11. Attaching additional labels to your time series and alerts	18
1.2.12. Next steps	19
1.3. MANAGING CLUSTER ALERTS	19
1.3.1. Contents of the Alerting UI	19
1.3.2. Getting information about alerts and alerting rules	21
1.3.3. Silencing alerts	22
1.3.4. Getting information about silences	23
1.3.5. Editing silences	23
1.3.6. Expiring silences	24
1.3.7. Changing Alertmanager configuration	24
1.3.8. Next steps	24
1.4. EXAMINING CLUSTER METRICS	24
1.4.1. Contents of the Metrics UI	24
1.4.2. Running metrics queries	26
1.4.3. Exploring the visualized metrics	27
1.4.4. Non-administrator access to metrics	27
1.4.5. Next steps	28
1.5. ACCESSING PROMETHEUS, ALERTMANAGER, AND GRAFANA	28
1.5.1. Accessing Prometheus, Alerting UI, and Grafana using the web console	28
1.5.2. Accessing Prometheus, Alertmanager, and Grafana directly	28
CHAPTER 2. MONITORING YOUR OWN SERVICES	30
2.1. ENABLING MONITORING OF YOUR OWN SERVICES	30
2.2. DEPLOYING A SAMPLE SERVICE	31
2.3. GRANTING USER PERMISSIONS USING WEB CONSOLE	32
2.4. GRANTING USER PERMISSIONS USING CLI	33
2.5. SETTING UP METRICS COLLECTION	34
2.6. CREATING ALERTING RULES	35
2.7. REMOVING ALERTING RULES	36
2.8. ACCESSING ALERTING RULES FOR YOUR PROJECT	36

2.9. ACCESSING ALERTING RULES FOR ALL NAMESPACES	37
2.10. ACCESSING THE METRICS OF YOUR SERVICE AS A DEVELOPER	37
2.11. ACCESSING METRICS OF ALL SERVICES AS A CLUSTER ADMINISTRATOR	38
CHAPTER 3. EXPOSING CUSTOM APPLICATION METRICS FOR AUTOSCALING	40
3.1. EXPOSING CUSTOM APPLICATION METRICS FOR HORIZONTAL POD AUTOSCALING	40

CHAPTER 1. CLUSTER MONITORING

1.1. ABOUT CLUSTER MONITORING

OpenShift Container Platform includes a pre-configured, pre-installed, and self-updating monitoring stack that is based on the [Prometheus](#) open source project and its wider eco-system. It provides monitoring of cluster components and includes a set of alerts to immediately notify the cluster administrator about any occurring problems and a set of [Grafana](#) dashboards. The cluster monitoring stack is only supported for monitoring OpenShift Container Platform clusters.



IMPORTANT

To ensure compatibility with future OpenShift Container Platform updates, configuring only the specified monitoring stack options is supported.

1.1.1. Stack components and monitored targets

The monitoring stack includes these components:

Table 1.1. Monitoring stack components

Component	Description
Cluster Monitoring Operator	The OpenShift Container Platform Cluster Monitoring Operator (CMO) is the central component of the stack. It controls the deployed monitoring components and resources and ensures that they are always up to date.
Prometheus Operator	The Prometheus Operator (PO) creates, configures, and manages Prometheus and Alertmanager instances. It also automatically generates monitoring target configurations based on familiar Kubernetes label queries.
Prometheus	The Prometheus is the systems and service monitoring system, around which the monitoring stack is based.
Prometheus Adapter	The Prometheus Adapter exposes cluster resource metrics API for horizontal pod autoscaling. Resource metrics are CPU and memory utilization.
Alertmanager	The Alertmanager service handles alerts sent by Prometheus.
kube-state-metrics	The kube-state-metrics exporter agent converts Kubernetes objects to metrics that Prometheus can use.

Component	Description
openshift-state-metrics	The openshift-state-metrics exporter expands upon kube-state-metrics by adding metrics for OpenShift Container Platform-specific resources.
node-exporter	node-exporter is an agent deployed on every node to collect metrics about it.
Thanos Querier	The Thanos Querier enables aggregating and, optionally, deduplicating cluster and user workload metrics under a single, multi-tenant interface.
Grafana	The Grafana analytics platform provides dashboards for analyzing and visualizing the metrics. The Grafana instance that is provided with the monitoring stack, along with its dashboards, is read-only.

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

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

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

- Telemeter client



NOTE

Each OpenShift Container Platform component is responsible for its monitoring configuration. For problems with a component's monitoring, open a bug in Bugzilla 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.

1.1.2. Next steps

[Configure the monitoring stack.](#)

1.2. CONFIGURING THE MONITORING STACK

Prior to OpenShift Container Platform 4, the Prometheus Cluster Monitoring stack was configured through the Ansible inventory file. For that purpose, the stack exposed a subset of its available configuration options as Ansible variables. You configured the stack before you installed OpenShift Container Platform.

In OpenShift Container Platform 4, Ansible is not the primary technology to install OpenShift Container Platform anymore. The installation program provides only a very low number of configuration options before installation. Configuring most OpenShift framework components, including the Prometheus 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.

1.2.1. 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.

1.2.2. Maintenance and support

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 reverses everything to the defined state by default and by design.

Explicitly unsupported cases include:

- **Creating additional `ServiceMonitor` objects in the `openshift-*` namespaces.** This extends the targets the cluster monitoring Prometheus instance scrapes, which can cause collisions and load differences that cannot be accounted for. These factors might make the Prometheus setup unstable.

- **Creating unexpected ConfigMap objects or PrometheusRule objects.** This causes the cluster monitoring Prometheus instance to include additional alerting and recording rules.
- **Modifying resources of the stack.** The Prometheus Monitoring Stack ensures its resources are always in the state it expects them to be. If they are modified, the stack will reset them.
- **Using resources of the stack for your purposes.** The resources created by the Prometheus Cluster Monitoring stack are not meant to be used by any other resources, as there are no guarantees about their backward compatibility.
- **Stopping the Cluster Monitoring Operator from reconciling the monitoring stack.**
- **Adding new alerting rules.**
- **Modifying the monitoring stack Grafana instance.**

1.2.3. Creating cluster monitoring ConfigMap

To configure the Prometheus Cluster Monitoring stack, you must create the cluster monitoring ConfigMap.

Prerequisites

- An installed **oc** CLI tool
- Administrative privileges for the cluster

Procedure

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

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

2. If it does not exist, create it:

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

3. Start editing the **cluster-monitoring-config** ConfigMap:

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

4. Create the **data** section if it does not exist yet:

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

1.2.4. Configuring the cluster monitoring stack

You can configure the Prometheus Cluster Monitoring stack using ConfigMaps. ConfigMaps configure the Cluster Monitoring Operator, which in turn configures components of the stack.

Prerequisites

- Make sure you have the **cluster-monitoring-config** ConfigMap object with the **data/config.yaml** section.

Procedure

1. Start editing the **cluster-monitoring-config** ConfigMap:

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

2. Put your configuration under **data/config.yaml** as key-value pair **<component_name>: <component_configuration>**:

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

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

For example, create this ConfigMap to configure a Persistent Volume Claim (PVC) for Prometheus:

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

Here, **prometheusK8s** defines the Prometheus component and the following lines define its configuration.

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

1.2.5. Configurable monitoring components

This table shows the monitoring components you can configure and the keys used to specify the components in the ConfigMap:

Table 1.2. Configurable monitoring components

Component	Key
Prometheus Operator	prometheusOperator
Prometheus	prometheusK8s
Alertmanager	alertmanagerMain
kube-state-metrics	kubeStateMetrics
openshift-state-metrics	openshiftStateMetrics
Grafana	grafana
Telemeter Client	telemeterClient
Prometheus Adapter	k8sPrometheusAdapter
Thanos Querier	thanosQuerier

From this list, only Prometheus and Alertmanager have extensive configuration options. All other components usually provide only the **nodeSelector** field for being deployed on a specified node.

1.2.6. Moving monitoring components to different nodes

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

Prerequisites

- Make sure you have the **cluster-monitoring-config** ConfigMap object with the **data/config.yaml** section.

Procedure

1. Start editing the **cluster-monitoring-config** ConfigMap:

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

2. Specify the **nodeSelector** constraint for the component under **data/config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    <component>:
      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 node. Often, only a single key-value pair is used.

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

For example, to move components to the node that is labeled **foo: bar**, use:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusOperator: nodeSelector: foo: bar prometheusK8s: nodeSelector: foo:
    bar alertmanagerMain: nodeSelector: foo: bar kubeStateMetrics: nodeSelector: foo:
    bar grafana: nodeSelector: foo: bar telemeterClient: nodeSelector: foo: bar
    k8sPrometheusAdapter: nodeSelector: foo: bar
    openshiftStateMetrics:
      nodeSelector:
        foo: bar
    thanosQuerier:
      nodeSelector:
        foo: bar
```

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

Additional resources

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

1.2.7. Assigning tolerations to monitoring components

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

Prerequisites

- Make sure you have the **cluster-monitoring-config** ConfigMap object with the **data/config.yaml** section.

Procedure

1. Start editing the **cluster-monitoring-config** ConfigMap:

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

- Specify **tolerations** for the component:

```

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

```

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

For example, a **oc adm taint nodes node1 key1=value1:NoSchedule** taint prevents the scheduler from placing pods in the **foo: bar** node. To make the **alertmanagerMain** component ignore that taint and to place **alertmanagerMain** in **foo: bar** normally, use this toleration:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      nodeSelector:
        foo: bar
      tolerations: - key: "key1" operator: "Equal" value: "value1" effect: "NoSchedule"

```

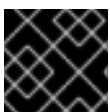
- Save the file to apply the changes. The new component placement configuration is applied automatically.

Additional resources

- See the [OpenShift Container Platform documentation](#) on taints and tolerations.
- See the [Kubernetes documentation](#) on taints and tolerations.

1.2.8. 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.



IMPORTANT

See [Recommended configurable storage technology](#).

1.2.9. 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. For information on system requirements for persistent storage, see [Prometheus database storage requirements](#).
- Make sure you have a persistent volume (PV) ready to be claimed by the persistent volume claim (PVC), one PV for each replica. Because Prometheus has two replicas and Alertmanager has three replicas, you need five PVs to support the entire monitoring stack. The PVs should be available from the Local Storage Operator. This does not apply if you enable dynamically provisioned storage.
- Use the block type of storage.
- [Configure local persistent storage](#).

1.2.9.1. Configuring a local persistent volume claim

For the Prometheus or Alertmanager to use a persistent volume (PV), you first must configure a persistent volume claim (PVC).

Prerequisites

- Make sure you have the **cluster-monitoring-config** ConfigMap object with the **data/config.yaml** section.

Procedure

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

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

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

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

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

For example, to configure a PVC that claims local persistent storage for Prometheus, use:

```
apiVersion: v1
```



```

kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      volumeClaimTemplate:
        metadata:
          name: localpvc
        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**.

To configure a PVC that claims local persistent storage for Alertmanager, use:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    alertmanagerMain:
      volumeClaimTemplate:
        metadata:
          name: localpvc
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 40Gi

```

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

1.2.9.2. Modifying retention time for Prometheus metrics data

By default, the Prometheus Cluster Monitoring stack configures the retention time for Prometheus data to be 15 days. You can modify the retention time to change how soon the data is deleted.

Prerequisites

- Make sure you have the **cluster-monitoring-config** ConfigMap object with the **data/config.yaml** section.

Procedure

1. Start editing the **cluster-monitoring-config** ConfigMap:

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

- Put your retention time configuration under **data/config.yaml**:

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

Substitute **<time_specification>** with a number directly followed by **ms** (milliseconds), **s** (seconds), **m** (minutes), **h** (hours), **d** (days), **w** (weeks), or **y** (years).

For example, to configure retention time to be 24 hours, use:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    prometheusK8s:
      retention: 24h
```

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

1.2.10. Configuring Alertmanager

The Prometheus Alertmanager is a component that manages incoming alerts, including:

- Alert silencing
- Alert inhibition
- Alert aggregation
- Reliable deduplication of alerts
- Grouping alerts
- Sending grouped alerts as notifications through receivers such as email, PagerDuty, and HipChat

1.2.10.1. Alertmanager default configuration

The default configuration of the OpenShift Container Platform Monitoring Alertmanager cluster is this:

```
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
  receivers:
  - name: default
  - name: watchdog

```

OpenShift Container Platform monitoring ships with the Watchdog alert, which fires continuously. Alertmanager repeatedly sends notifications for the Watchdog alert to the notification provider, for example, to PagerDuty. The provider is usually configured to notify the administrator when it stops receiving the Watchdog alert. This mechanism helps ensure continuous operation of Prometheus as well as continuous communication between Alertmanager and the notification provider.

1.2.10.2. Applying custom Alertmanager configuration

You can overwrite the default Alertmanager configuration by editing the **alertmanager-main** secret inside the **openshift-monitoring** namespace.

Prerequisites

- An installed **jq** tool for processing JSON data

Procedure

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

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

2. Change the configuration in file **alertmanager.yaml** to your new configuration:

```

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: <your_service> 1
  routes:
  - match:

```

```

    <your_matching_rules> 2
    receiver: <receiver> 3
receivers:
- name: default
- name: watchdog
- name: <receiver>
  <receiver_configuration>

```

- 1 **service** specifies the service that fires the alerts.
- 2 **<your_matching_rules>** specify the target alerts.
- 3 **receiver** specifies the receiver to use for the alert.

For example, this listing configures PagerDuty for notifications:

```

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 fired by the **example-app** service are sent using the **team-frontend-page** receiver, which means that these alerts are paged to a chosen person.

3. Apply the new configuration in the file:

```

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

1.2.10.3. Alerting rules

OpenShift Container Platform Cluster Monitoring by default ships with a set of pre-defined alerting rules.

Note that:

- The default alerting rules are used specifically for the OpenShift Container Platform cluster and nothing else. For example, you get alerts for a persistent volume in the cluster, but you do not get them for persistent volume in your custom namespace.
- Currently you cannot add custom alerting rules.
- Some alerting rules have identical names. This is intentional. They are sending alerts about the same event with different thresholds, with different severity, or both.
- With the inhibition rules, the lower severity is inhibited when the higher severity is firing.

1.2.10.4. Listing acting alerting rules

You can list the alerting rules that currently apply to the cluster.

Procedure

1. Configure the necessary port forwarding:

```
$ oc -n openshift-monitoring port-forward svc/prometheus-operated 9090
```

2. Fetch the JSON object containing acting alerting rules and their properties:

```
$ curl -s http://localhost:9090/api/v1/rules | jq '[.data.groups[].rules[] | select(.type=="alerting")]'
```

Example output

```
[
  {
    "name": "ClusterOperatorDown",
    "query": "cluster_operator_up{job=\"cluster-version-operator\"} == 0",
    "duration": 600,
    "labels": {
      "severity": "critical"
    },
    "annotations": {
      "message": "Cluster operator {{ $labels.name }} has not been available for 10 mins. Operator may be down or disabled, cluster will not be kept up to date and upgrades will not be possible."
    },
    "alerts": [],
    "health": "ok",
    "type": "alerting"
  },
  {
    "name": "ClusterOperatorDegraded",
    ...
```

Additional resources

- See also [the Alertmanager documentation](#).

1.2.11. Attaching additional labels to your time series and alerts

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

Prerequisites

- Make sure you have the **cluster-monitoring-config** ConfigMap object with the **data/config.yaml** section.

Procedure

1. Start editing the **cluster-monitoring-config** ConfigMap:

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

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

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

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



WARNING

Do not use **prometheus** or **prometheus_replica** as key names, because they are reserved and would be overwritten.

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

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

```

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

```

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

1.2.12. Next steps

- [Manage cluster alerts](#).
- Learn about [remote health reporting](#) and, if necessary, opt out of it.

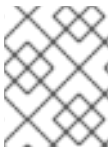
1.3. MANAGING CLUSTER ALERTS

OpenShift Container Platform 4.5 provides a web interface to the Alertmanager, which enables you to manage alerts. This section demonstrates how to use the Alerting UI.



IMPORTANT

Currently, the Alerting UI only displays alerts that come by default with the OpenShift Container Platform. It does not list user-defined alerts. That will change in a future release.



NOTE

The Alerting UI is only accessible through the Administrator Perspective and not through the Developer Perspective.

1.3.1. Contents of the Alerting UI

This section shows and explains the contents of the Alerting UI, a web interface to the Alertmanager.

The Alerting UI is accessible by clicking **Monitoring** → **Alerting** in the OpenShift Container Platform web console.

The three main pages of the Alerting UI are the **Alerts**, the **Silences**, and the **Alerting rules** pages.

The **Alerts** page:

Alerting [Alertmanager UI](#)

Alerts Silences Alerting Rules

Filter Alerts by name...

2 Firing 0 Silenced 0 Pending Select all filters 2 Items

Name ↑	Severity ↓	State ↓
<p>AlertmanagerReceiversNotConfigured</p> <p>Alerts are not configured to be sent to a notification system, meaning that you may not be notified in a timely fashion when important failures occur. Check</p>	Warning	Firing Since Jun 18, 10:52 am
<p>Watchdog</p> <p>This is an alert meant to ensure that the entire alerting pipeline is functional. This alert is always firing, therefore it should always be firing in Alertmanager</p>	None	Firing Since Jun 18, 10:49 am

1. Filtering alerts by their names.
2. Filtering the alerts by their states. To fire, some alerts need a certain condition to be true for the duration of a timeout. If a condition of an alert is currently true, but the timeout has not been reached, such an alert is in the **Pending** state.
3. Alert name.
4. Description of an alert.
5. Value of the Severity label of the alert.
6. Current state of the alert and when the alert went into this state.
7. Actions you can do with the alert.

The **Silences** page:

Alerting [Alertmanager UI](#)

Alerts Silences Alerting Rules

Create Silence

Filter Silences by name...

1 Active 0 Pending 0 Expired Select all filters 1 Item

Name ↑	Firing Alerts ↓	State ↓	Creator ↓
<p>watchdog=watchdog</p> <p>watchdog=watchdog</p>	-	Active Ends Jun 18, 3:49 pm	kube:admin

1. Creating a silence for an alert.
2. Filtering silences by their name.
3. Filtering silences by their states. If a silence is pending, it is currently not active because it is scheduled to start at a later time. If a silence expired, it is no longer active because it has reached its end time.
4. Description of a silence. It includes the silence name and the specification of alerts that it matches.
5. Number of alerts that are being silenced by the silence.
6. Current state of the silence. For active silences, it shows when the silence ends. For pending silences, it shows when the silence starts.
7. User that created the silence.
8. Actions you can do with a silence.

The **Alerting Rules** page:

The screenshot shows the 'Alerting Rules' page in the Alertmanager UI. The page has a header with 'Alerting' and a link to the 'Alertmanager UI'. Below the header are tabs for 'Alerts', 'Silences', and 'Alerting Rules'. A search bar is located at the top right, and filter buttons for 'Active' (2) and 'Inactive' (147) are at the top left. A table lists the alerting rules with columns for Name, Severity, and Alert State. Red circles 1-5 highlight specific UI elements: 1. Search bar, 2. Filter buttons, 3. Rule name, 4. Severity label, 5. Alert State.

Name ↑	Severity ↓	Alert State ↓
AR AggregatedAPIDown	Warning	Inactive
AR AggregatedAPIErrors	Warning	Inactive
AR AlertmanagerConfigInconsistent	Critical	Inactive
AR AlertmanagerFailedReload	Warning	Inactive

1. Filtering Alerting Rules by their name.
2. Filtering Alerting Rules by their states.
3. Alerting Rule name.
4. Value of the Severity label of the Alerting Rule.
5. State of the Alert defined by the Alerting Rule.

Also, next to the title of each of these pages is a link to the Alertmanager interface.

1.3.2. Getting information about alerts and alerting rules

You can find an alert and see information about it or its governing alerting rule.

Procedure

1. Open the OpenShift Container Platform web console and navigate to the **Monitoring → Alerting → Alerts** page.
2. Optional: Filter the alerts by name using the **Filter Alerts by name** field.
3. Optional: Filter the alerts by state using one or more of the state buttons **Firing**, **Silenced**, **Pending**.
4. Optional: Sort the alerts by clicking one or more of the **Name**, **Severity**, and **State** column headers.
5. After you see the alert you need, you can see either details of the alert or details of its governing alerting rule.

To see alert details, click on the name of the alert. The page that opens has the graph with timeseries of the alert. It also has information about the alert, including:

- A link to its governing alerting rule
- Description of the alert
- Labels attached to the alert

To see alerting rule details, click the button in the last column and select **View Alerting Rule**. The page that opens has information about the alerting rule, including:

- 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
- Graph for each alert governed by the alerting rule, showing the value with which the alert is firing
- Table of all alerts governed by the alerting rule

1.3.3. Silencing alerts

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

Procedure

To silence a set of alerts by creating an alert specification:

1. Navigate to the **Monitoring → Alerting → Silences** page of the OpenShift Container Platform web console.
2. Click **Create Silence**.
3. Populate the **Create Silence** form.
4. To create the silence, click **Silence**.

To silence a specific alert:

1. Navigate to the **Monitoring → Alerting → Alerts** page of the OpenShift Container Platform web console.
2. For the alert that you want to silence, click the button in the last column and click **Silence Alert**. The **Create Silence** form will appear with prepopulated specification of the chosen alert.
3. Optional: Modify the silence.
4. To create the silence, click **Silence**.

1.3.4. Getting information about silences

You can find a silence and view its details.

Procedure

1. Open the OpenShift Container Platform web console and navigate to the **Monitoring → Alerting → Silences** page.
2. Optional: Filter the silences by name using the **Filter Silences by name** field.
3. Optional: Filter the silences by state using one or more of the state buttons **Active**, **Pending**, **Expired**.
4. Optional: Sort the silences by clicking one or more of the **Name**, **Firing Alerts**, **State**, and **Creator** column headers.
5. After you see the silence, you can click its name to see the details, including:
 - Alert specification
 - State
 - Start time
 - End time
 - Number and list of firing alerts

1.3.5. Editing silences

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

Procedure

1. Navigate to the **Monitoring → Alerting → Silences** page.
2. For the silence you want to modify, click the button in the last column and click **Edit silence**. Alternatively, you can click **Actions → Edit Silence** in the **Silence Overview** screen for a particular silence.
3. In the **Edit Silence** screen, enter your changes and click the **Save** button. This will expire the existing silence and create one with the chosen configuration.

1.3.6. Expiring silences

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

Procedure

1. Navigate to the **Monitoring** → **Alerting** → **Silences** page.
2. For the silence you want to expire, click the button in the last column and click **Expire Silence**. Alternatively, you can click the **Actions** → **Expire Silence** button in the **Silence Overview** page for a particular silence.
3. Confirm by clicking **Expire Silence**. This expires the silence.

1.3.7. Changing Alertmanager configuration

You can change the Alertmanager configuration by editing the YAML configuration file using the OpenShift Container Platform web console.

Procedure

To change the Alertmanager configuration:

1. Navigate to the **Administration** → **Cluster Settings** → **Global Configuration** → **Alertmanager** → **YAML** page of the web console.
2. Modify the YAML configuration file.
3. Click **Save**.

Additional resources

- See [Configuring Alertmanager](#) for more information on changing Alertmanager configuration.

1.3.8. Next steps

[Examine cluster metrics.](#)

1.4. EXAMINING CLUSTER METRICS

OpenShift Container Platform 4.5 provides a web interface to Prometheus, which enables you to run Prometheus Query Language (PromQL) queries and examine the metrics visualized on a plot. This functionality provides an extensive overview of the cluster state and enables you to troubleshoot problems.



NOTE

The Metrics UI queries not only cluster metrics, but metrics of all namespaces.

1.4.1. Contents of the Metrics UI

This section shows and explains the contents of the Metrics UI, a web interface to Prometheus.

The **Metrics** page is accessible by clicking **Monitoring** → **Metrics** in the OpenShift Container Platform web console.

The screenshot displays the Prometheus UI 'Metrics' page. The left sidebar contains navigation links. The main area features a line graph showing alert counts over time. Below the graph, a query editor contains the query: `sum(sort_desc(sum_over_time(ALERTS(alertstate="firing")[24h]))) by (alertname)`. The results are shown in a table:

Alert Name	Value
ClusterOperatorDegraded	6
ClusterOperatorDown	12
KubePodCrashLooping	32
Watchdog	71

At the bottom, there are two query form inputs, each with a placeholder text: 'Expression (press Shift+Enter for newlines)'.

1. Actions.
 - Add query.
 - Expand or collapse all query tables.
 - Delete all queries.
2. Hide the plot.
3. The interactive plot.
4. The catalog of available metrics.
5. Add query.
6. Run queries.
7. Query forms.
8. Expand or collapse the form.


9. The query.
10. Clear query.
11. Enable or disable query.
12. Actions for a specific query.
 - Enable or disable query.
 - Show or hide all series of the query from the plot.
 - Delete query.
13. The metrics table for a query.
14. Color assigned to the graph of the metric. Clicking the square shows or hides the metric's graph.

Additionally, there is a link to the old Prometheus interface next to the title of the page.

1.4.2. Running metrics queries

You begin working with metrics by entering one or several Prometheus Query Language (PromQL) queries.

Procedure

1. Open the OpenShift Container Platform web console and navigate to the **Monitoring** → **Metrics** page.
2. In the query field, enter your PromQL query.
 - To show all available metrics and PromQL functions, click **Insert Metric at Cursor**.
3. For multiple queries, click **Add Query**.
4. For deleting queries, click  for the query, then select **Delete query**.
5. For keeping but not running a query, click the **Disable query** button.
6. Once you finish creating queries, click the **Run Queries** button. 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 timeout or overload the browser when drawing timeseries graphs. To avoid this, hide the graph and calibrate your query using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.

7. 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


See the [Prometheus Query Language documentation](#).

1.4.3. Exploring the visualized metrics

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

Procedure

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

- 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 shown 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.

To reset the time range, click **Reset Zoom**.

3. To display outputs of 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.
4. For more detailed information about metrics of a specific query, expand the table of that query using the drop-down button. Every metric is shown with its current value.
5. To hide the plot, click **Hide Graph**.

1.4.4. Non-administrator access to metrics

As a developer, one can enable user workload monitoring for an application or service in a project. As an administrator, you use the same feature to enable monitoring for infrastructure workloads. In that case, a developer or administrator of that project can examine the exposed metrics using the Developer Perspective in the web console.



IMPORTANT

Examining metrics using the Developer Perspective 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/>.

Additional resources

See the [documentation on monitoring your own services](#). It includes details on accessing non-cluster metrics as a developer or a privileged user.

1.4.5. Next steps

[Access the Prometheus, Alertmanager, and Grafana.](#)

1.5. ACCESSING PROMETHEUS, ALERTMANAGER, AND GRAFANA

To work with data gathered by the monitoring stack, you might want to use the Prometheus, Alertmanager, and Grafana interfaces. They are available by default.

1.5.1. Accessing Prometheus, Alerting UI, and Grafana using the web console

You can access Prometheus, Alerting, and Grafana web UIs using a web browser through the OpenShift Container Platform web console.



NOTE

The Alerting UI accessed in this procedure is the new interface for Alertmanager.

Prerequisites

- Authentication is performed against the OpenShift Container Platform identity and uses the same credentials or means of authentication as is used elsewhere in OpenShift Container Platform. You must use a role that has read access to all namespaces, such as the **cluster-monitoring-view** cluster role.

Procedure

1. Navigate to the OpenShift Container Platform web console and authenticate.
2. To access Prometheus, navigate to the "Monitoring" → "Metrics" page.
To access the Alerting UI, navigate to the "Monitoring" → "Alerting" page.

To access Grafana, navigate to the "Monitoring" → "Dashboards" page.

1.5.2. Accessing Prometheus, Alertmanager, and Grafana directly

You can access Prometheus, Alertmanager, and Grafana web UIs using the **oc** tool and a web browser.



NOTE

The Alertmanager UI accessed in this procedure is the old interface for Alertmanager.

Prerequisites

- Authentication is performed against the OpenShift Container Platform identity and uses the same credentials or means of authentication as is used elsewhere in OpenShift Container Platform. You must use a role that has read access to all namespaces, such as the **cluster-monitoring-view** cluster role.

Procedure

1. Run:

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

Example output

```
NAME          HOST/PORT          ...
alertmanager-main alertmanager-main-openshift-monitoring.apps._url_.openshift.com ...
grafana        grafana-openshift-monitoring.apps._url_.openshift.com      ...
prometheus-k8s prometheus-k8s-openshift-monitoring.apps._url_.openshift.com ...
```

2. Prepend **https://** to the address, you cannot access web UIs using unencrypted connection. For example, this is the resulting URL for Alertmanager:

```
https://alertmanager-main-openshift-monitoring.apps._url_.openshift.com
```

3. Navigate to the address using a web browser and authenticate.

Additional resources

- For documentation on the new interface for Alertmanager, see [Managing cluster alerts](#).



IMPORTANT

The monitoring routes are managed by the Cluster Monitoring Operator and cannot be modified by the user.

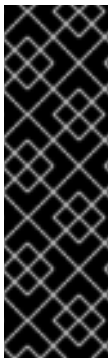
CHAPTER 2. MONITORING YOUR OWN SERVICES

You can use OpenShift Monitoring for your own services in addition to monitoring the cluster. This way, you do not need to use an additional monitoring solution. This helps keeping monitoring centralized. Additionally, you can extend the access to the metrics of your services beyond cluster administrators. This enables developers and arbitrary users to access these metrics.



NOTE

Opting into monitoring your own services is mutually exclusive with either a custom installation of Prometheus Operator or installing Prometheus Operator using Operator Lifecycle Manager (OLM).



IMPORTANT

Monitoring your own services 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/>.

2.1. ENABLING MONITORING OF YOUR OWN SERVICES

You can enable monitoring your own services by setting the **techPreviewUserWorkload/enabled** flag in the cluster monitoring ConfigMap.

Prerequisites

- Make sure you have the **cluster-monitoring-config** ConfigMap object with the **data/config.yaml** section.

Procedure

1. Start editing the **cluster-monitoring-config** ConfigMap:

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

2. Set the **techPreviewUserWorkload** setting to **true** under **data/config.yaml**:

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

3. Save the file to apply the changes. Monitoring your own services is enabled automatically.

4. Optional: You can check that the **prometheus-user-workload** Pods were created:

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

Example output

```
NAME                                READY STATUS    RESTARTS  AGE
prometheus-operator-6f7b748d5b-t7nbg 2/2   Running    0         3h
prometheus-user-workload-0           5/5   Running    1         3h
prometheus-user-workload-1           5/5   Running    1         3h
thanos-ruler-user-workload-0         3/3   Running    0         3h
thanos-ruler-user-workload-1         3/3   Running    0         3h
```

Additional resources

- See [Configuring the monitoring stack](#) to learn how to create **cluster-monitoring-config**.

2.2. DEPLOYING A SAMPLE SERVICE

To test monitoring your own services, 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. Fill the file with the configuration for deploying the service:

```
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: quay.io/brancz/prometheus-example-app:v0.2.0
          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 **ns1** project. This service exposes the custom **version** metric.

3. Apply the configuration file to the cluster:

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

It will take some time to deploy the service.

4. You can check that the service is running:

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

Example output

```

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

```

2.3. GRANTING USER PERMISSIONS USING WEB CONSOLE

This procedure shows how to grant users permissions for monitoring their own services using the web console.

Prerequisites

- Have a user created.
- Log in to the web console as a cluster administrator.

Procedure

1. In the web console, navigate to **User Management** → **Role Bindings** → **Create Binding**.
2. In **Binding Type**, select the "Namespace Role Binding" type.
3. In **Name**, enter a name for the binding.

4. In **Namespace**, select the namespace where you want to grant the access. For example, select **ns1**.
5. In **Role Name**, enter **monitoring-rules-view**, **monitoring-rules-edit**, or **monitoring-edit**.
 - **monitoring-rules-view** allows reading PrometheusRule custom resources within the namespace.
 - **monitoring-rules-edit** allows creating, modifying, and deleting PrometheusRule custom resources matching the permitted namespace.
 - **monitoring-edit** gives the same permissions as **monitoring-rules-edit**. Additionally, it allows creating new scraping targets for services or Pods. It also allows creating, modifying, and deleting ServiceMonitors and PodMonitors.



IMPORTANT

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

For example, enter **monitoring-edit**.

6. In **Subject**, select **User**.
7. In **Subject Name**, enter the name of the user. For example, enter **johnsmith**.
8. Confirm the role binding. If you followed the example, then user **johnsmith** has been assigned the permissions for setting up metrics collection and creating alerting rules in the **ns1** namespace.

2.4. GRANTING USER PERMISSIONS USING CLI

This procedure shows how to grant users permissions for monitoring their own services using the CLI.

Prerequisites

- Have a user created.
- Log in using the **oc** command.

Procedure

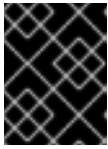
- Run this command to assign `<role>` to `<user>` in `<namespace>`:

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

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

- **monitoring-rules-view** allows reading PrometheusRule custom resources within the namespace.
- **monitoring-rules-edit** allows creating, modifying, and deleting PrometheusRule custom resources matching the permitted namespace.

- **monitoring-edit** gives the same permissions as **monitoring-rules-edit**. Additionally, it allows creating new scraping targets for services or Pods. It also allows creating, modifying, and deleting ServiceMonitors and PodMonitors.



IMPORTANT

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

As an example, substitute `<role>` with **monitoring-edit**, `<user>` with **johnsmith**, and `<namespace>` with **ns1**. This assigns to user **johnsmith** the permissions for setting up metrics collection and creating alerting rules in the **ns1** namespace.

2.5. SETTING UP METRICS COLLECTION

To use the metrics exposed by your service, you must configure OpenShift Monitoring to scrape metrics from the `/metrics` endpoint. You can do this using a ServiceMonitor, which is a custom resource definition (CRD) that specifies how a service should be monitored, or a PodMonitor, which is a 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 how to create a ServiceMonitor for the service.

Prerequisites

- Log in as a cluster administrator or a user with the **monitoring-edit** role.

Procedure

1. Create a YAML file for the ServiceMonitor configuration. In this example, the file is called **example-app-service-monitor.yaml**.
2. Fill the file with the configuration for creating the ServiceMonitor:

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

This configuration makes OpenShift Monitoring scrape the metrics exposed by the sample service deployed in "Deploying a sample service", which includes the single **version** metric.

3. Apply the configuration file to the cluster:
 -

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

It will take some time to deploy the ServiceMonitor.

4. You can check that the ServiceMonitor is running:

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

Example output

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

Additional resources

See the [Prometheus Operator API documentation](#) for more information on ServiceMonitors and PodMonitors.

2.6. CREATING ALERTING RULES

You can create alerting rules, which will fire alerts based on values of chosen metrics.



NOTE

Viewing and managing your rules and alerts is not yet integrated into the web console. A cluster administrator can instead use the Alertmanager UI or the Thanos Ruler. See the respective sections for instructions.

Prerequisites

- Log in as a user that has the **monitoring-rules-edit** role for the namespace where you want to create the alerting rule.

Procedure

1. Create a YAML file for alerting rules. In this example, it is called **example-app-alerting-rule.yaml**.
2. Fill the file with the configuration for the alerting rules:



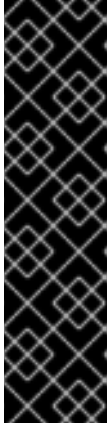
NOTE

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

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: example-alert
  namespace: ns1
spec:
  groups:
  - name: example
```

```
rules:
- alert: VersionAlert
  expr: version{job="prometheus-example-app"} == 0
```

This configuration creates an alerting rule named **example-alert**, which fires an alert when the **version** metric exposed by the sample service becomes **0**.



IMPORTANT

For every namespace, you can use metrics of that namespace and cluster metrics, but not metrics of another namespace.

For example, an alerting rule for **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 namespaces. OpenShift Container Platform Monitoring by default provides a set of alerting rules for these namespaces.

3. Apply the configuration file to the cluster:

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

It will take some time to create the alerting rules.

2.7. REMOVING ALERTING RULES

You can remove an alerting rule.

Prerequisites

- Log in as a user that has the **monitoring-rules-edit** role for the namespace where you want to remove an alerting rule.

Procedure

- To remove rule <foo> in <namespace>, run:

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

2.8. ACCESSING ALERTING RULES FOR YOUR PROJECT

You can list existing alerting rules for your project.

Prerequisites

- Log in as a user with the **monitoring-rules-view** role against your project.

Procedure

1. To list alerting rules in <project>, run:

■


```
$ oc -n <project> get prometheusrule
```

- To list the configuration of an alerting rule, run:

```
$ oc -n <project> get prometheusrule <rule> -oyaml
```

2.9. ACCESSING ALERTING RULES FOR ALL NAMESPACES

As a cluster administrator, you can access alerting rules from all namespaces together in a single view.



NOTE

In a future release, the route to the Thanos Ruler UI will be deprecated in favor of the web console.

Prerequisites

- Have the **oc** command installed.
- Log in as a cluster administrator.

Procedure

- List routes for the **openshift-user-workload-monitoring** namespace:

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

The output shows the URL for the Thanos Ruler UI:

```
NAME          HOST/PORT
...
thanos-ruler  thanos-ruler-openshift-user-workload-
monitoring.apps.example.devcluster.openshift.com
```

- Navigate to the listed URL. Here you can see user alerting rules from all namespaces.

2.10. ACCESSING THE METRICS OF YOUR SERVICE AS A DEVELOPER

After you have enabled monitoring your own services, deployed a service, and set up metrics collection for the service, you can access the metrics of the service as a developer or as a user with view permissions for the project.



NOTE

The Grafana instance shipped within OpenShift Container Platform Monitoring is read-only and displays only infrastructure-related dashboards.

Prerequisites

- Deploy the service that you want to monitor.
- Enable monitoring of your own services.

- Have metrics scraping set up for the service.
- Log in as a developer or as a user with view permissions for the project.

Procedure

1. Go to the OpenShift Container Platform web console, switch to the Developer Perspective, then click **Advanced** → **Metrics**. Select the project you want to see the metrics for.



NOTE

Developers can only use the Developer Perspective and not the Administrator Perspective. They can only query metrics from a single project. They cannot access the third-party UIs provided with OpenShift Container Platform Monitoring.

2. Use the PromQL interface to run queries for your services.

2.11. ACCESSING METRICS OF ALL SERVICES AS A CLUSTER ADMINISTRATOR

If you are a cluster administrator or a user with view permissions for all namespaces, you can access metrics of all services from all namespaces together in a single view.

Prerequisites

- Log in to the web console as a cluster administrator or a user with view permissions for all namespaces.
- Optionally, log in with the **oc** command as well.

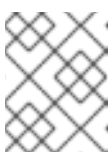
Procedure

- Using the Metrics web interface:
 1. Go to the OpenShift Container Platform web console, switch to the Administrator Perspective, and click **Monitoring** → **Metrics**.



NOTE

Cluster administrators, when using the Administrator Perspective, have access to all cluster metrics and to custom service metrics from all projects.



NOTE

Only cluster administrators have access to the third-party UIs provided with OpenShift Container Platform Monitoring.

2. Use the PromQL interface to run queries for your services.
- Using the Thanos Querier UI:

**NOTE**

In a future release, the route to the Thanos Querier UI will be deprecated in favor of the web console.

1. List routes for the **openshift-monitoring** namespace:

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

The output shows the URL for the Thanos Querier UI:

```
NAME          HOST/PORT
...
thanos-querier  thanos-querier-openshift-
monitoring.apps.example.devcluster.openshift.com
```

2. Navigate to the listed URL. Here you can see all metrics from all namespaces.

Additional resources

- See the [section on using the PromQL interface](#).

CHAPTER 3. EXPOSING CUSTOM APPLICATION METRICS FOR AUTOSCALING

You can export custom application metrics for the horizontal pod autoscaler.



IMPORTANT

Prometheus Adapter 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/>.

3.1. EXPOSING CUSTOM APPLICATION METRICS FOR HORIZONTAL POD AUTOSCALING

You can use the **prometheus-adapter** resource to expose custom application metrics for the horizontal pod autoscaler.

Prerequisites

- Make sure you have a custom Prometheus instance installed. In this example, it is presumed that Prometheus was installed in the **default** namespace.
- Make sure you configured monitoring for your application. In this example, it is presumed that the application and the service monitor for it were installed in the **default** namespace.

Procedure

1. Create a YAML file for your configuration. In this example, it is called **deploy.yaml**.
2. Add configuration for creating the service account, necessary roles, and role bindings for **prometheus-adapter**:

```
kind: ServiceAccount
apiVersion: v1
metadata:
  name: custom-metrics-apiserver
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: custom-metrics-server-resources
rules:
- apiGroups:
  - custom.metrics.k8s.io
  resources: ["*"]
  verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
```

```

kind: ClusterRole
metadata:
  name: custom-metrics-resource-reader
rules:
- apiGroups:
  - ""
  resources:
  - namespaces
  - pods
  - services
  verbs:
  - get
  - list
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: custom-metrics:system:auth-delegator
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:auth-delegator
subjects:
- kind: ServiceAccount
  name: custom-metrics-apiserver
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: custom-metrics-auth-reader
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: extension-apiserver-authentication-reader
subjects:
- kind: ServiceAccount
  name: custom-metrics-apiserver
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: custom-metrics-resource-reader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: custom-metrics-resource-reader
subjects:
- kind: ServiceAccount
  name: custom-metrics-apiserver
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding

```

```

metadata:
  name: hpa-controller-custom-metrics
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: custom-metrics-server-resources
subjects:
- kind: ServiceAccount
  name: horizontal-pod-autoscaler
  namespace: kube-system
---
```

3. Add configuration for the custom metrics for **prometheus-adapter**:

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: adapter-config
  namespace: default
data:
  config.yaml: |
    rules:
    - seriesQuery: 'http_requests_total{namespace!="",pod!=""}' 1
      resources:
        overrides:
          namespace: {resource: "namespace"}
          pod: {resource: "pod"}
          service: {resource: "service"}
      name:
        matches: "^(.*)_total"
        as: "${1}_per_second" 2
      metricsQuery: 'sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)'
---
```

1 Specifies the chosen metric to be the number of HTTP requests.

2 Specifies the frequency for the metric.

4. Add configuration for registering **prometheus-adapter** as an API service:

```

apiVersion: v1
kind: Service
metadata:
  annotations:
    service.alpha.openshift.io/serving-cert-secret-name: prometheus-adapter-tls
  labels:
    name: prometheus-adapter
    name: prometheus-adapter
  namespace: default
spec:
  ports:
  - name: https
    port: 443
    targetPort: 6443
```

```

selector:
  app: prometheus-adapter
  type: ClusterIP
---
apiVersion: apiregistration.k8s.io/v1beta1
kind: APIService
metadata:
  name: v1beta1.custom.metrics.k8s.io
spec:
  service:
    name: prometheus-adapter
    namespace: default
  group: custom.metrics.k8s.io
  version: v1beta1
  insecureSkipTLSVerify: true
  groupPriorityMinimum: 100
  versionPriority: 100
---

```

5. Show the Prometheus Adapter image to use:

```
$ oc get -n openshift-monitoring deploy/prometheus-adapter -o jsonpath="{..image}"
```

Example output

```
quay.io/openshift-release-dev/ocp-v4.3-art-dev@sha256:76db3c86554ad7f581ba33844d6a6ebc891236f7db64f2d290c3135ba81c264c
```

6. Add configuration for deploying **prometheus-adapter**:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus-adapter
    name: prometheus-adapter
    namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-adapter
  template:
    metadata:
      labels:
        app: prometheus-adapter
        name: prometheus-adapter
    spec:
      serviceAccountName: custom-metrics-apiserver
      containers:
      - name: prometheus-adapter
        image: openshift-release-dev/ocp-v4.3-art-dev 1
        args:
          - --secure-port=6443
          - --tls-cert-file=/var/run/serving-cert/tls.crt

```

```

- --tls-private-key-file=/var/run/serving-cert/tls.key
- --logtostderr=true
- --prometheus-url=http://prometheus-operated.default.svc:9090/
- --metrics-relist-interval=1m
- --v=4
- --config=/etc/adapter/config.yaml
ports:
- containerPort: 6443
volumeMounts:
- mountPath: /var/run/serving-cert
  name: volume-serving-cert
  readOnly: true
- mountPath: /etc/adapter/
  name: config
  readOnly: true
- mountPath: /tmp
  name: tmp-vol
volumes:
- name: volume-serving-cert
  secret:
    secretName: prometheus-adapter-tls
- name: config
  configMap:
    name: adapter-config
- name: tmp-vol
  emptyDir: {}

```

- 1** **image: openshift-release-dev/ocp-v4.3-art-dev** specifies the Prometheus Adapter image found in the previous step.

7. Apply the configuration file to the cluster:

```
$ oc apply -f deploy.yaml
```

8. Now the application's metrics are exposed and can be used to configure horizontal pod autoscaling.

Additional resources

- See the [horizontal pod autoscaling documentation](#).
- See the [Kubernetes documentation on horizontal pod autoscaler](#).