



# OpenShift Container Platform 4.18

## Cluster Observability Operator

Configuring and using the Cluster Observability Operator in OpenShift Container Platform



# OpenShift Container Platform 4.18 Cluster Observability Operator

---

Configuring and using the Cluster Observability Operator in OpenShift Container Platform

## Legal Notice

Copyright © 2025 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

Use the Cluster Observability Operator to deploy and configure observability components in OpenShift Container Platform.

## Table of Contents

<b>CHAPTER 1. CLUSTER OBSERVABILITY OPERATOR RELEASE NOTES</b>	<b>4</b>
1.1. CLUSTER OBSERVABILITY OPERATOR 1.1.1	4
1.1.1. Bug fixes	4
1.2. CLUSTER OBSERVABILITY OPERATOR 1.1	4
1.2.1. New features and enhancements	4
1.3. KNOWN ISSUES	4
1.3.1. Bug fixes	5
1.4. CLUSTER OBSERVABILITY OPERATOR 1.0	5
1.4.1. New features and enhancements	5
1.4.2. CVEs	6
1.4.3. Bug fixes	6
1.5. FEATURES AVAILABLE ON OLDER, TECHNOLOGY PREVIEW RELEASES	6
1.6. CLUSTER OBSERVABILITY OPERATOR 0.4.1	6
1.6.1. New features and enhancements	7
1.6.2. CVEs	7
1.6.3. Bug fixes	7
1.7. CLUSTER OBSERVABILITY OPERATOR 0.4.0	7
1.7.1. New features and enhancements	7
1.7.1.1. Troubleshooting UI plugin	7
1.7.1.2. Distributed tracing UI plugin	7
1.7.2. Bug fixes	8
1.8. CLUSTER OBSERVABILITY OPERATOR 0.3.2	8
1.8.1. New features and enhancements	8
1.8.2. Bug fixes	8
1.9. CLUSTER OBSERVABILITY OPERATOR 0.3.0	8
1.9.1. New features and enhancements	8
1.10. CLUSTER OBSERVABILITY OPERATOR 0.2.0	8
1.10.1. New features and enhancements	9
1.11. CLUSTER OBSERVABILITY OPERATOR 0.1.3	9
1.11.1. Bug fixes	9
1.12. CLUSTER OBSERVABILITY OPERATOR 0.1.2	9
1.12.1. CVEs	9
1.12.2. Bug fixes	9
1.13. CLUSTER OBSERVABILITY OPERATOR 0.1.1	9
1.13.1. New features and enhancements	10
1.14. CLUSTER OBSERVABILITY OPERATOR 0.1	10
<b>CHAPTER 2. CLUSTER OBSERVABILITY OPERATOR OVERVIEW</b>	<b>11</b>
2.1. COO COMPARED TO DEFAULT MONITORING STACK	11
2.2. KEY ADVANTAGES OF USING COO	12
2.2.1. Extensibility	12
2.2.2. Multi-tenancy support	12
2.2.3. Scalability	12
2.2.4. Flexibility	12
2.3. TARGET USERS FOR COO	13
2.3.1. Enterprise-level users and administrators	13
2.3.2. Operations teams in multi-tenant environments	13
2.3.3. Development and operations teams	13
2.4. USING SERVER-SIDE APPLY TO CUSTOMIZE PROMETHEUS RESOURCES	13
<b>CHAPTER 3. INSTALLING THE CLUSTER OBSERVABILITY OPERATOR</b>	<b>19</b>

3.1. INSTALLING THE CLUSTER OBSERVABILITY OPERATOR IN THE WEB CONSOLE	19
3.2. UNINSTALLING THE CLUSTER OBSERVABILITY OPERATOR USING THE WEB CONSOLE	20
<b>CHAPTER 4. CONFIGURING THE CLUSTER OBSERVABILITY OPERATOR TO MONITOR A SERVICE ....</b>	<b>21</b>
4.1. DEPLOYING A SAMPLE SERVICE FOR CLUSTER OBSERVABILITY OPERATOR	21
4.2. SPECIFYING HOW A SERVICE IS MONITORED BY CLUSTER OBSERVABILITY OPERATOR	22
4.3. CREATING A MONITORINGSTACK OBJECT FOR THE CLUSTER OBSERVABILITY OPERATOR	24
4.4. VALIDATING THE MONITORING STACK	26
<b>CHAPTER 5. OBSERVABILITY UI PLUGINS .....</b>	<b>28</b>
5.1. OBSERVABILITY UI PLUGINS OVERVIEW	28
5.1.1. Monitoring	28
5.1.2. Cluster logging	28
5.1.3. Troubleshooting	28
5.1.4. Distributed tracing	29
5.2. MONITORING UI PLUGIN	29
5.2.1. Installing the Cluster Observability Operator monitoring UI plugin	30
5.2.2. Cluster Observability Operator incident detection overview	31
5.2.3. Using Cluster Observability Operator incident detection	31
5.3. LOGGING UI PLUGIN	33
5.3.1. Installing the Cluster Observability Operator logging UI plugin	34
5.4. DISTRIBUTED TRACING UI PLUGIN	35
5.4.1. Installing the Cluster Observability Operator distributed tracing UI plugin	35
5.4.2. Using the Cluster Observability Operator distributed tracing UI plugin	36
5.5. TROUBLESHOOTING UI PLUGIN	36
5.5.1. Installing the Cluster Observability Operator Troubleshooting UI plugin	37
5.5.2. Using the Cluster Observability Operator troubleshooting UI plugin	37
5.5.3. Creating the example alert	40



# CHAPTER 1. CLUSTER OBSERVABILITY OPERATOR RELEASE NOTES

The Cluster Observability Operator (COO) is an optional OpenShift Container Platform Operator that enables administrators to create standalone monitoring stacks that are independently configurable for use by different services and users.

The COO complements the built-in monitoring capabilities of OpenShift Container Platform. You can deploy it in parallel with the default platform and user workload monitoring stacks managed by the Cluster Monitoring Operator (CMO).

These release notes track the development of the Cluster Observability Operator in OpenShift Container Platform.

The following table provides information about which features are available depending on the version of Cluster Observability Operator and OpenShift Container Platform:

COO Version	OCP Versions	Distributed tracing	Logging	Troubleshooting panel	ACM alerts	Incident detection
1.1+	4.12 - 4.14	✓	✓	✗	✗	✗
1.1+	4.15	✓	✓	✗	✓	✗
1.1+	4.16 - 4.17	✓	✓	✓	✓	✗
1.1+	4.18+	✓	✓	✓	✓	✓

## 1.1. CLUSTER OBSERVABILITY OPERATOR 1.1.1

### 1.1.1. Bug fixes

- Previously, **observability-operator** and **perses-operator** pods on many clusters entered a **CrashLoopBackOff** state due to **OutOfMemory** errors, after upgrading from Cluster Observability Operator 1.0. This release resolves the issue. ([COO-784](#))

## 1.2. CLUSTER OBSERVABILITY OPERATOR 1.1

### 1.2.1. New features and enhancements

- You can now install the monitoring UI plugin using COO. ([COO-262](#))
- You can enable incident detection in the monitoring UI plugin. ([COO-690](#))
- TLS support for the Thanos web endpoint has been added. ([COO-222](#))

## 1.3. KNOWN ISSUES

These are the known issues in Cluster Observability Operator 1.1.0:



- **observability-operator** and **perses-operator** pods enter a **CrashLoopBackOff** state due to **OutOfMemory** errors, after upgrading from Cluster Observability Operator 1.0. A workaround is provided in the knowledge base article [ClusterObservability and perses operator pod in CrashLoopBackOff due to OOMKilled in RHOC4](#).

This issue is being tracked in [COO-784](#).

### 1.3.1. Bug fixes

- Previously, the logging UI plugin did not support setting a custom LokiStack name or namespace. This release resolves the issue. ([COO-332](#))

## 1.4. CLUSTER OBSERVABILITY OPERATOR 1.0

### 1.4.1. New features and enhancements

- COO is now enabled for OpenShift Container Platform platform monitoring. ([COO-476](#))
  - Implements HTTPS support for COO web server. ([COO-480](#))
  - Implements authn/authz for COO web server. ([COO-481](#))
  - Configures ServiceMonitor resource to collect metrics from COO. ([COO-482](#))
  - Adds **operatorframework.io/cluster-monitoring=true** annotation to the OLM bundle. ([COO-483](#))
  - Defines the alerting strategy for COO. ([COO-484](#))
  - Configures PrometheusRule for alerting. ([COO-485](#))
- Support level annotations have been added to the **UIPlugin** CR when created. The support level is based on the plugin type, with values of **DevPreview**, **TechPreview**, or **GeneralAvailability**. ([COO-318](#))
- You can now configure the Alertmanager **scheme** and **tlsConfig** fields in the Prometheus CR. ([COO-219](#))
- The extended Technical Preview for the troubleshooting panel adds support for correlating traces with Kubernetes resources and directly with other observable signals including logs, alerts, metrics, and network events. ([COO-450](#))
  - You can select a Tempo instance and tenant when you navigate to the tracing page by clicking **Observe → Tracing** in the web console. The preview troubleshooting panel only works with the **openshift-tracing / platform** instance and the **platform** tenant.
  - The troubleshooting panel works best in the **Administrator** perspective. It has limited functionality in the Developer perspective due to authorization issues with some back ends, most notably Prometheus for metrics and alerts. This will be addressed in a future release.

The following table provides information about which features are available depending on the version of Cluster Observability Operator and OpenShift Container Platform:

COO Version	OCP Versions	Distributed Tracing	Logging	Troubleshooting Panel
1.0	4.12 - 4.15	✓	✓	✗
1.0	4.16+	✓	✓	✓

### 1.4.2. CVEs

- [CVE-2023-26159](#)
- [CVE-2024-28849](#)
- [CVE-2024-45338](#)

### 1.4.3. Bug fixes

- Previously, the default namespace for the COO installation was **openshift-operators**. With this release, the default namespace changes to **openshift-cluster-observability-operator**. ([COO-32](#))
- Previously, **korrel8r** was only able to parse time series selector expressions. With this release, **korrel8r** can parse any valid PromQL expression to extract the time series selectors that it uses for correlation. ([COO-558](#))
- Previously, when viewing a Tempo instance from the Distributed Tracing UI plugin, the scatter plot graph showing the traces duration was not rendered correctly. The bubble size was too large and overlapped the x and y axis. With this release, the graph is rendered correctly. ([COO-319](#))

## 1.5. FEATURES AVAILABLE ON OLDER, TECHNOLOGY PREVIEW RELEASES

The following table provides information about which features are available depending on older version of Cluster Observability Operator and OpenShift Container Platform:

COO Version	OCP Versions	Dashboards	Distributed Tracing	Logging	Troubleshooting Panel
0.2.0	4.11	✓	✗	✗	✗
0.3.0+, 0.4.0+	4.11 - 4.15	✓	✓	✓	✗
0.3.0+, 0.4.0+	4.16+	✓	✓	✓	✓

## 1.6. CLUSTER OBSERVABILITY OPERATOR 0.4.1

The following advisory is available for Cluster Observability Operator 0.4.1:

- [RHEA-2024:8040 Cluster Observability Operator 0.4.1](#)

### 1.6.1. New features and enhancements

- You can now configure WebTLS for Prometheus and Alertmanager.

### 1.6.2. CVEs

- [CVE-2024-6104](#)
- [CVE-2024-24786](#)

### 1.6.3. Bug fixes

- Previously, when you deleted the dashboard UI plugin, the **consoles.operator.openshift.io** resource still contained **console-dashboards-plugin**. This release resolves the issue. ([COO-152](#))
- Previously, the web console did not display the correct icon for Red Hat COO . This release resolves the issue. ([COO-353](#))
- Previously, when you installed the COO from the web console, the support section contained an invalid link. This release resolves the issue. ([COO-354](#))
- Previously, the cluster service version (CSV) for COO linked to an unofficial version of the documentation. This release resolves the issue. ([COO-356](#))


## 1.7. CLUSTER OBSERVABILITY OPERATOR 0.4.0

The following advisory is available for Cluster Observability Operator 0.4.0:

- [RHEA-2024:6699 Cluster Observability Operator 0.4.0](#)

### 1.7.1. New features and enhancements

#### 1.7.1.1. Troubleshooting UI plugin

- The troubleshooting UI panel has been improved so you can now select and focus on a specific starting signal.
- There is more visibility into Korrel8r queries, with the option of selecting the depth.
- Users of OpenShift Container Platform version 4.17+ can access the troubleshooting UI panel from the Application Launcher  . Alternatively, on versions 4.16+, you can access it in the web console by clicking on **Observe** → **Alerting**.

For more information, see [troubleshooting UI plugin](#).

#### 1.7.1.2. Distributed tracing UI plugin

- The distributed tracing UI plugin has been enhanced, with a Gantt chart now available for exploring traces.

For more information, see [distributed tracing UI plugin](#).

### 1.7.2. Bug fixes

- Previously, metrics were not available to normal users when accessed in the Developer perspective of the web console, by clicking on **Observe** → **Logs**. This release resolves the issue. ([COO-288](#))
- Previously, the troubleshooting UI plugin used the wrong filter for network observability. This release resolves the issue. ([COO-299](#))
- Previously, the troubleshooting UI plugin generated an incorrect URL for pod label searches. This release resolves the issue. ([COO-298](#))
- Previously, there was an authorization vulnerability in the Distributed tracing UI plugin. This release resolves the issue and the Distributed tracing UI plugin has been hardened by using only multi-tenant **TempoStack** and **TempoMonolithic** instances going forward.

## 1.8. CLUSTER OBSERVABILITY OPERATOR 0.3.2

The following advisory is available for Cluster Observability Operator 0.3.2:

- [RHEA-2024:5985 Cluster Observability Operator 0.3.2](#)

### 1.8.1. New features and enhancements

- With this release, you can now use tolerations and node selectors with **MonitoringStack** components.

### 1.8.2. Bug fixes

- Previously, the logging UIPlugin was not in the **Available** state and the logging pod was not created, when installed on a specific version of OpenShift Container Platform. This release resolves the issue. ([COO-260](#))

## 1.9. CLUSTER OBSERVABILITY OPERATOR 0.3.0

The following advisory is available for Cluster Observability Operator 0.3.0:

- [RHEA-2024:4399 Cluster Observability Operator 0.3.0](#)

### 1.9.1. New features and enhancements

- With this release, the Cluster Observability Operator adds backend support for future OpenShift Container Platform observability web console UI plugins and observability components.

## 1.10. CLUSTER OBSERVABILITY OPERATOR 0.2.0

The following advisory is available for Cluster Observability Operator 0.2.0:

- [RHEA-2024:2662 Cluster Observability Operator 0.2.0](#)

### 1.10.1. New features and enhancements

- With this release, the Cluster Observability Operator supports installing and managing observability-related plugins for the OpenShift Container Platform web console user interface (UI). ([COO-58](#))

## 1.11. CLUSTER OBSERVABILITY OPERATOR 0.1.3

The following advisory is available for Cluster Observability Operator 0.1.3:

- [RHEA-2024:1744 Cluster Observability Operator 0.1.3](#)

### 1.11.1. Bug fixes

- Previously, if you tried to access the Prometheus web user interface (UI) at `http://<prometheus_url>:9090/graph`, the following error message would display: **Error opening React index.html: open web/ui/static/react/index.html: no such file or directory.** This release resolves the issue, and the Prometheus web UI now displays correctly. ([COO-34](#))

## 1.12. CLUSTER OBSERVABILITY OPERATOR 0.1.2

The following advisory is available for Cluster Observability Operator 0.1.2:

- [RHEA-2024:1534 Cluster Observability Operator 0.1.2](#)

### 1.12.1. CVEs

- [CVE-2023-45142](#)

### 1.12.2. Bug fixes

- Previously, certain cluster service version (CSV) annotations were not included in the metadata for COO. Because of these missing annotations, certain COO features and capabilities did not appear in the package manifest or in the OperatorHub user interface. This release adds the missing annotations, thereby resolving this issue. ([COO-11](#))
- Previously, automatic updates of the COO did not work, and a newer version of the Operator did not automatically replace the older version, even though the newer version was available in OperatorHub. This release resolves the issue. ([COO-12](#))
- Previously, Thanos Querier only listened for network traffic on port 9090 of 127.0.0.1 (**localhost**), which resulted in a **502 Bad Gateway** error if you tried to reach the Thanos Querier service. With this release, the Thanos Querier configuration has been updated so that the component now listens on the default port (10902), thereby resolving the issue. As a result of this change, you can also now modify the port via server side apply (SSA) and add a proxy chain, if required. ([COO-14](#))

## 1.13. CLUSTER OBSERVABILITY OPERATOR 0.1.1

The following advisory is available for Cluster Observability Operator 0.1.1:

- [2024:0550 Cluster Observability Operator 0.1.1](#)

### **1.13.1. New features and enhancements**

This release updates the Cluster Observability Operator to support installing the Operator in restricted networks or disconnected environments.

## **1.14. CLUSTER OBSERVABILITY OPERATOR 0.1**

This release makes a Technology Preview version of the Cluster Observability Operator available on OperatorHub.

## CHAPTER 2. CLUSTER OBSERVABILITY OPERATOR OVERVIEW

The Cluster Observability Operator (COO) is an optional component of the OpenShift Container Platform designed for creating and managing highly customizable monitoring stacks. It enables cluster administrators to automate configuration and management of monitoring needs extensively, offering a more tailored and detailed view of each namespace compared to the default OpenShift Container Platform monitoring system.

The COO deploys the following monitoring components:

- **Prometheus** - A highly available Prometheus instance capable of sending metrics to an external endpoint by using remote write.
- **Thanos Querier** (optional) - Enables querying of Prometheus instances from a central location.
- **Alertmanager** (optional) - Provides alert configuration capabilities for different services.
- **UI plugins** (optional) - Enhances the observability capabilities with plugins for monitoring, logging, distributed tracing and troubleshooting.
- **Korrel8r** (optional) - Provides observability signal correlation, powered by the open source Korrel8r project.
- **Incident detection** (optional) - Groups related alerts into incidents, to help you identify the root causes of alert bursts.

### 2.1. COO COMPARED TO DEFAULT MONITORING STACK

The COO components function independently of the default in-cluster monitoring stack, which is deployed and managed by the Cluster Monitoring Operator (CMO). Monitoring stacks deployed by the two Operators do not conflict. You can use a COO monitoring stack in addition to the default platform monitoring components deployed by the CMO.

The key differences between COO and the default in-cluster monitoring stack are shown in the following table:

Feature	COO	Default monitoring stack
Scope and integration	Offers comprehensive monitoring and analytics for enterprise-level needs, covering cluster and workload performance.	Limited to core components within the cluster, for example, API server and etcd, and to OpenShift-specific namespaces.
	However, it lacks direct integration with OpenShift Container Platform and typically requires an external Grafana instance for dashboards.	There is deep integration into OpenShift Container Platform including console dashboards and alert management in the console.

Feature	COO	Default monitoring stack
<b>Configuration and customization</b>	<p>Broader configuration options including data retention periods, storage methods, and collected data types.</p> <p>The COO can delegate ownership of single configurable fields in custom resources to users by using Server-Side Apply (SSA), which enhances customization.</p>	Built-in configurations with limited customization options.
<b>Data retention and storage</b>	Long-term data retention, supporting historical analysis and capacity planning	Shorter data retention times, focusing on short-term monitoring and real-time detection.

## 2.2. KEY ADVANTAGES OF USING COO

Deploying COO helps you address monitoring requirements that are hard to achieve using the default monitoring stack.

### 2.2.1. Extensibility

- You can add more metrics to a COO-deployed monitoring stack, which is not possible with core platform monitoring without losing support.
- You can receive cluster-specific metrics from core platform monitoring through federation.
- COO supports advanced monitoring scenarios like trend forecasting and anomaly detection.

### 2.2.2. Multi-tenancy support

- You can create monitoring stacks per user namespace.
- You can deploy multiple stacks per namespace or a single stack for multiple namespaces.
- COO enables independent configuration of alerts and receivers for different teams.

### 2.2.3. Scalability

- Supports multiple monitoring stacks on a single cluster.
- Enables monitoring of large clusters through manual sharding.
- Addresses cases where metrics exceed the capabilities of a single Prometheus instance.

### 2.2.4. Flexibility

- Decoupled from OpenShift Container Platform release cycles.
- Faster release iterations and rapid response to changing requirements.
- Independent management of alerting rules.



## 2.3. TARGET USERS FOR COO

COO is ideal for users who need high customizability, scalability, and long-term data retention, especially in complex, multi-tenant enterprise environments.

### 2.3.1. Enterprise-level users and administrators

Enterprise users require in-depth monitoring capabilities for OpenShift Container Platform clusters, including advanced performance analysis, long-term data retention, trend forecasting, and historical analysis. These features help enterprises better understand resource usage, prevent performance issues, and optimize resource allocation.

### 2.3.2. Operations teams in multi-tenant environments

With multi-tenancy support, COO allows different teams to configure monitoring views for their projects and applications, making it suitable for teams with flexible monitoring needs.

### 2.3.3. Development and operations teams

COO provides fine-grained monitoring and customizable observability views for in-depth troubleshooting, anomaly detection, and performance tuning during development and operations.

## 2.4. USING SERVER-SIDE APPLY TO CUSTOMIZE PROMETHEUS RESOURCES

Server-Side Apply is a feature that enables collaborative management of Kubernetes resources. The control plane tracks how different users and controllers manage fields within a Kubernetes object. It introduces the concept of field managers and tracks ownership of fields. This centralized control provides conflict detection and resolution, and reduces the risk of unintended overwrites.

Compared to Client-Side Apply, it is more declarative, and tracks field management instead of last applied state.

### Server-Side Apply

Declarative configuration management by updating a resource's state without needing to delete and recreate it.

### Field management

Users can specify which fields of a resource they want to update, without affecting the other fields.

### Managed fields

Kubernetes stores metadata about who manages each field of an object in the **managedFields** field within metadata.

### Conflicts

If multiple managers try to modify the same field, a conflict occurs. The applier can choose to overwrite, relinquish control, or share management.

### Merge strategy

Server-Side Apply merges fields based on the actor who manages them.

### Procedure

1. Add a **MonitoringStack** resource using the following configuration:

## Example MonitoringStack object

```

apiVersion: monitoring.rhobs/v1alpha1
kind: MonitoringStack
metadata:
  labels:
    coo: example
  name: sample-monitoring-stack
  namespace: coo-demo
spec:
  logLevel: debug
  retention: 1d
  resourceSelector:
    matchLabels:
      app: demo

```

2. A Prometheus resource named **sample-monitoring-stack** is generated in the **coo-demo** namespace. Retrieve the managed fields of the generated Prometheus resource by running the following command:

```
$ oc -n coo-demo get Prometheus.monitoring.rhobs -oyaml --show-managed-fields
```

## Example output

```

managedFields:
- apiVersion: monitoring.rhobs/v1
  fieldsType: FieldsV1
  fieldsV1:
    f:metadata:
      f:labels:
        f:app.kubernetes.io/managed-by: {}
        f:app.kubernetes.io/name: {}
        f:app.kubernetes.io/part-of: {}
      f:ownerReferences:
        k:{"uid":"81da0d9a-61aa-4df3-affc-71015bcbde5a"}: {}
    f:spec:
      f:additionalScrapeConfigs: {}
      f:affinity:
        f:podAntiAffinity:
          f:requiredDuringSchedulingIgnoredDuringExecution: {}
      f:alerting:
        f:alertmanagers: {}
      f:arbitraryFSAccessThroughSMs: {}
      f:logLevel: {}
      f:podMetadata:
        f:labels:
          f:app.kubernetes.io/component: {}
          f:app.kubernetes.io/part-of: {}
      f:podMonitorSelector: {}
      f:replicas: {}
      f:resources:
        f:limits:
          f:cpu: {}
          f:memory: {}
        f:requests:

```

```

    f:cpu: {}
    f:memory: {}
  f:retention: {}
  f:ruleSelector: {}
  f:rules:
    f:alert: {}
  f:securityContext:
    f:fsGroup: {}
    f:runAsNonRoot: {}
    f:runAsUser: {}
  f:serviceName: {}
  f:serviceMonitorSelector: {}
  f:thanos:
    f:baseImage: {}
    f:resources: {}
    f:version: {}
  f:tsdb: {}
manager: observability-operator
operation: Apply
- apiVersion: monitoring.rhobs/v1
fieldsType: FieldsV1
fieldsV1:
  f:status:
    .: {}
  f:availableReplicas: {}
  f:conditions:
    .: {}
    k:{"type":"Available"}:
      .: {}
      f:lastTransitionTime: {}
      f:observedGeneration: {}
      f:status: {}
      f:type: {}
    k:{"type":"Reconciled"}:
      .: {}
      f:lastTransitionTime: {}
      f:observedGeneration: {}
      f:status: {}
      f:type: {}
  f:paused: {}
  f:replicas: {}
  f:shardStatuses:
    .: {}
    k:{"shardID":"0"}:
      .: {}
      f:availableReplicas: {}
      f:replicas: {}
      f:shardID: {}
      f:unavailableReplicas: {}
      f:updatedReplicas: {}
  f:unavailableReplicas: {}
  f:updatedReplicas: {}
manager: PrometheusOperator
operation: Update
subresource: status

```

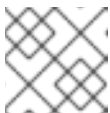
3. Check the **metadata.managedFields** values, and observe that some fields in **metadata** and **spec** are managed by the **MonitoringStack** resource.
4. Modify a field that is not controlled by the **MonitoringStack** resource:
  - a. Change **spec.enforcedSampleLimit**, which is a field not set by the **MonitoringStack** resource. Create the file **prom-spec-edited.yaml**:

#### **prom-spec-edited.yaml**

```
apiVersion: monitoring.rhobs/v1
kind: Prometheus
metadata:
  name: sample-monitoring-stack
  namespace: coo-demo
spec:
  enforcedSampleLimit: 1000
```

- b. Apply the YAML by running the following command:

```
$ oc apply -f ./prom-spec-edited.yaml --server-side
```



#### **NOTE**

You must use the **--server-side** flag.

- c. Get the changed Prometheus object and note that there is one more section in **managedFields** which has **spec.enforcedSampleLimit**:

```
$ oc get prometheus -n coo-demo
```

#### **Example output**

```
managedFields: ❶
- apiVersion: monitoring.rhobs/v1
  fieldsType: FieldsV1
  fieldsV1:
    f:metadata:
      f:labels:
        f:app.kubernetes.io/managed-by: {}
        f:app.kubernetes.io/name: {}
        f:app.kubernetes.io/part-of: {}
    f:spec:
      f:enforcedSampleLimit: {} ❷
  manager: kubectl
  operation: Apply
```

❶ **managedFields**

❷ **spec.enforcedSampleLimit**

5. Modify a field that is managed by the **MonitoringStack** resource:

- a. Change **spec.LogLevel**, which is a field managed by the **MonitoringStack** resource, using the following YAML configuration:

```
# changing the logLevel from debug to info
apiVersion: monitoring.rhobs/v1
kind: Prometheus
metadata:
  name: sample-monitoring-stack
  namespace: coo-demo
spec:
  logLevel: info 1
```

1 **spec.logLevel** has been added

- b. Apply the YAML by running the following command:

```
$ oc apply -f ./prom-spec-edited.yaml --server-side
```

### Example output

```
error: Apply failed with 1 conflict: conflict with "observability-operator": .spec.logLevel
Please review the fields above--they currently have other managers. Here
are the ways you can resolve this warning:
* If you intend to manage all of these fields, please re-run the apply
  command with the `--force-conflicts` flag.
* If you do not intend to manage all of the fields, please edit your
  manifest to remove references to the fields that should keep their
  current managers.
* You may co-own fields by updating your manifest to match the existing
  value; in this case, you'll become the manager if the other manager(s)
  stop managing the field (remove it from their configuration).
See https://kubernetes.io/docs/reference/using-api/server-side-apply/#conflicts
```

- c. Notice that the field **spec.logLevel** cannot be changed using Server-Side Apply, because it is already managed by **observability-operator**.
- d. Use the **--force-conflicts** flag to force the change.

```
$ oc apply -f ./prom-spec-edited.yaml --server-side --force-conflicts
```

### Example output

```
prometheus.monitoring.rhobs/sample-monitoring-stack serverside-applied
```

With **--force-conflicts** flag, the field can be forced to change, but since the same field is also managed by the **MonitoringStack** resource, the Observability Operator detects the change, and reverts it back to the value set by the **MonitoringStack** resource.

**NOTE**

Some Prometheus fields generated by the **MonitoringStack** resource are influenced by the fields in the **MonitoringStack spec** stanza, for example, **logLevel**. These can be changed by changing the **MonitoringStack spec**.

- e. To change the **logLevel** in the Prometheus object, apply the following YAML to change the **MonitoringStack** resource:

```
apiVersion: monitoring.rhobs/v1alpha1
kind: MonitoringStack
metadata:
  name: sample-monitoring-stack
  labels:
    coo: example
spec:
  logLevel: info
```

- f. To confirm that the change has taken place, query for the log level by running the following command:

```
$ oc -n coo-demo get Prometheus.monitoring.rhobs -
o=jsonpath='{.items[0].spec.logLevel}'
```

**Example output**

```
info
```

**NOTE**

1. If a new version of an Operator generates a field that was previously generated and controlled by an actor, the value set by the actor will be overridden. For example, you are managing a field **enforcedSampleLimit** which is not generated by the **MonitoringStack** resource. If the Observability Operator is upgraded, and the new version of the Operator generates a value for **enforcedSampleLimit**, this will override the value you have previously set.
2. The **Prometheus** object generated by the **MonitoringStack** resource may contain some fields which are not explicitly set by the monitoring stack. These fields appear because they have default values.

**Additional resources**

- [Kubernetes documentation for Server-Side Apply \(SSA\)](#)

## CHAPTER 3. INSTALLING THE CLUSTER OBSERVABILITY OPERATOR

As a cluster administrator, you can install or remove the Cluster Observability Operator (COO) from OperatorHub by using the OpenShift Container Platform web console. OperatorHub is a user interface that works in conjunction with Operator Lifecycle Manager (OLM), which installs and manages Operators on a cluster.

### 3.1. INSTALLING THE CLUSTER OBSERVABILITY OPERATOR IN THE WEB CONSOLE

Install the Cluster Observability Operator (COO) from OperatorHub by using the OpenShift Container Platform web console.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have logged in to the OpenShift Container Platform web console.

#### Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Type **cluster observability operator** in the **Filter by keyword** box.
3. Click **Cluster Observability Operator** in the list of results.
4. Read the information about the Operator, and configure the following installation settings:
  - **Update channel** → **stable**
  - **Version** → **1.0.0** or later
  - **Installation mode** → **All namespaces on the cluster (default)**
  - **Installed Namespace** → **Operator recommended Namespace: openshift-cluster-observability-operator**
  - Select **Enable Operator recommended cluster monitoring on this Namespace**
  - **Update approval** → **Automatic**
5. Optional: You can change the installation settings to suit your requirements. For example, you can select to subscribe to a different update channel, to install an older released version of the Operator, or to require manual approval for updates to new versions of the Operator.
6. Click **Install**.

#### Verification

- Go to **Operators → Installed Operators**, and verify that the **Cluster Observability Operator** entry appears in the list.

#### Additional resources

[Adding Operators to a cluster](#)

## 3.2. UNINSTALLING THE CLUSTER OBSERVABILITY OPERATOR USING THE WEB CONSOLE


If you have installed the Cluster Observability Operator (COO) by using OperatorHub, you can uninstall it in the OpenShift Container Platform web console.

### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have logged in to the OpenShift Container Platform web console.

### Procedure

1. Go to **Operators** → **Installed Operators**.
2. Locate the **Cluster Observability Operator** entry in the list.

3. Click  for this entry and select **Uninstall Operator**.

### Verification

- Go to **Operators** → **Installed Operators**, and verify that the **Cluster Observability Operator** entry no longer appears in the list.



## CHAPTER 4. CONFIGURING THE CLUSTER OBSERVABILITY OPERATOR TO MONITOR A SERVICE

You can monitor metrics for a service by configuring monitoring stacks managed by the Cluster Observability Operator (COO).

To test monitoring a service, follow these steps:

- Deploy a sample service that defines a service endpoint.
- Create a **ServiceMonitor** object that specifies how the service is to be monitored by the COO.
- Create a **MonitoringStack** object to discover the **ServiceMonitor** object.

### 4.1. DEPLOYING A SAMPLE SERVICE FOR CLUSTER OBSERVABILITY OPERATOR

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

#### Prerequisites

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

#### Procedure

1. Create a YAML file named **prometheus-coo-example-app.yaml** that contains the following configuration details for a namespace, deployment, and service:

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

```

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

```

2. Save the file.
3. Apply the configuration to the cluster by running the following command:

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

4. Verify that the pod is running by running the following command and observing the output:

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

### Example output

NAME	READY	STATUS	RESTARTS	AGE
prometheus-coo-example-app-0927545cb7-anskj	1/1	Running	0	81m

## 4.2. SPECIFYING HOW A SERVICE IS MONITORED BY CLUSTER OBSERVABILITY OPERATOR

To use the metrics exposed by the sample service you created in the "Deploying a sample service for Cluster Observability Operator" section, you must configure monitoring components to scrape metrics from the **/metrics** endpoint.

You can create this configuration by using a **ServiceMonitor** object that specifies how the service is to be monitored, or a **PodMonitor** object that specifies how a pod is to be monitored. The **ServiceMonitor** object requires a **Service** object. The **PodMonitor** object does not, which enables the **MonitoringStack** object to scrape metrics directly from the metrics endpoint exposed by a pod.

This procedure shows how to create a **ServiceMonitor** object for a sample service named **prometheus-coo-example-app** in the **ns1-coo** namespace.

### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role or as a user with administrative permissions for the namespace.
- You have installed the Cluster Observability Operator.
- You have deployed the **prometheus-coo-example-app** sample service in the **ns1-coo** namespace.



#### NOTE

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

### Procedure

1. Create a YAML file named **example-coo-app-service-monitor.yaml** that contains the following **ServiceMonitor** object configuration details:

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

This configuration defines a **ServiceMonitor** object that the **MonitoringStack** object will reference to scrape the metrics data exposed by the **prometheus-coo-example-app** sample service.

2. Apply the configuration to the cluster by running the following command:

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

3. Verify that the **ServiceMonitor** resource is created by running the following command and observing the output:

```
$ oc -n ns1-coo get servicemonitors.monitoring.rhobs
```

### Example output

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

## 4.3. CREATING A MONITORINGSTACK OBJECT FOR THE CLUSTER OBSERVABILITY OPERATOR

To scrape the metrics data exposed by the target **prometheus-coo-example-app** service, create a **MonitoringStack** object that references the **ServiceMonitor** object you created in the "Specifying how a service is monitored for Cluster Observability Operator" section. This **MonitoringStack** object can then discover the service and scrape the exposed metrics data from it.

### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role or as a user with administrative permissions for the namespace.
- You have installed the Cluster Observability Operator.
- You have deployed the **prometheus-coo-example-app** sample service in the **ns1-coo** namespace.
- You have created a **ServiceMonitor** object named **prometheus-coo-example-monitor** in the **ns1-coo** namespace.

### Procedure

1. Create a YAML file for the **MonitoringStack** object configuration. For this example, name the file **example-coo-monitoring-stack.yaml**.
2. Add the following **MonitoringStack** object configuration details:

#### Example MonitoringStack object

```
apiVersion: monitoring.rhobs/v1alpha1
kind: MonitoringStack
metadata:
  name: example-coo-monitoring-stack
  namespace: ns1-coo
spec:
  logLevel: debug
  retention: 1d
  resourceSelector:
    matchLabels:
      k8s-app: prometheus-coo-example-monitor
```

3. Apply the **MonitoringStack** object by running the following command:

```
$ oc apply -f example-coo-monitoring-stack.yaml
```

4. Verify that the **MonitoringStack** object is available by running the following command and inspecting the output:

```
$ oc -n ns1-coo get monitoringstack
```

#### Example output

NAME	AGE
example-coo-monitoring-stack	81m

- Run the following command to retrieve information about the active targets from Prometheus and filter the output to list only targets labeled with **app=prometheus-coo-example-app**. This verifies which targets are discovered and actively monitored by Prometheus with this specific label.

```
$ oc -n ns1-coo exec -c prometheus prometheus-example-coo-monitoring-stack-0 -- curl -s
'http://localhost:9090/api/v1/targets' | jq '.data.activeTargets[].discoveredLabels |
select(.__meta_kubernetes_endpoints_label_app=="prometheus-coo-example-app")'
```

### Example output

```
{
  "__address__": "10.129.2.25:8080",
  "__meta_kubernetes_endpoint_address_target_kind": "Pod",
  "__meta_kubernetes_endpoint_address_target_name": "prometheus-coo-example-app-
5d8cd498c7-9j2gj",
  "__meta_kubernetes_endpoint_node_name": "ci-ln-8tt8vxb-72292-6cxjr-worker-a-wdfnz",
  "__meta_kubernetes_endpoint_port_name": "web",
  "__meta_kubernetes_endpoint_port_protocol": "TCP",
  "__meta_kubernetes_endpoint_ready": "true",

  "__meta_kubernetes_endpoints_annotation_endpoints_kubernetes_io_last_change_trigger_time": "2024-11-05T11:24:09Z",

  "__meta_kubernetes_endpoints_annotationpresent_endpoints_kubernetes_io_last_change_trigger_time": "true",
  "__meta_kubernetes_endpoints_label_app": "prometheus-coo-example-app",
  "__meta_kubernetes_endpoints_labelpresent_app": "true",
  "__meta_kubernetes_endpoints_name": "prometheus-coo-example-app",
  "__meta_kubernetes_namespace": "ns1-coo",
  "__meta_kubernetes_pod_annotation_k8s_ovn_org_pod_networks": "{\"default\": {\"ip_addresses\": [\"10.129.2.25/23\"], \"mac_address\": \"0a:58:0a:81:02:19\", \"gateway_ips\": [\"10.129.2.1\"], \"routes\": [{\"dest\": \"10.128.0.0/14\", \"nextHop\": \"10.129.2.1\"}, {\"dest\": \"172.30.0.0/16\", \"nextHop\": \"10.129.2.1\"}, {\"dest\": \"100.64.0.0/16\", \"nextHop\": \"10.129.2.1\"}], \"ip_address\": \"10.129.2.25/23\", \"gateway_ip\": \"10.129.2.1\", \"role\": \"primary\"}}",
  "__meta_kubernetes_pod_annotation_k8s_v1_cni_cncf_io_network_status": "[{\\n \\\"name\\\": \\\"ovn-kubernetes\\\",\\n \\\"interface\\\": \\\"eth0\\\",\\n \\\"ips\\\": [\\n \\\"10.129.2.25\\\"\\n ],\\n \\\"mac\\\": \\\"0a:58:0a:81:02:19\\\",\\n \\\"default\\\": true,\\n \\\"dns\\\": {}\\n}]",
  "__meta_kubernetes_pod_annotation_openshift_io_scc": "restricted-v2",
  "__meta_kubernetes_pod_annotation_seccomp_security_alpha_kubernetes_io_pod": "runtime/default",
  "__meta_kubernetes_pod_annotationpresent_k8s_ovn_org_pod_networks": "true",
  "__meta_kubernetes_pod_annotationpresent_k8s_v1_cni_cncf_io_network_status": "true",
  "__meta_kubernetes_pod_annotationpresent_openshift_io_scc": "true",

  "__meta_kubernetes_pod_annotationpresent_seccomp_security_alpha_kubernetes_io_pod": "true",
  "__meta_kubernetes_pod_controller_kind": "ReplicaSet",
  "__meta_kubernetes_pod_controller_name": "prometheus-coo-example-app-5d8cd498c7",
  "__meta_kubernetes_pod_host_ip": "10.0.128.2",
  "__meta_kubernetes_pod_ip": "10.129.2.25",
```

```

__meta_kubernetes_pod_label_app": "prometheus-coo-example-app",
__meta_kubernetes_pod_label_pod_template_hash": "5d8cd498c7",
__meta_kubernetes_pod_labelpresent_app": "true",
__meta_kubernetes_pod_labelpresent_pod_template_hash": "true",
__meta_kubernetes_pod_name": "prometheus-coo-example-app-5d8cd498c7-9j2gj",
__meta_kubernetes_pod_node_name": "ci-ln-8tt8vxb-72292-6cxjr-worker-a-wdfnz",
__meta_kubernetes_pod_phase": "Running",
__meta_kubernetes_pod_ready": "true",
__meta_kubernetes_pod_uid": "054c11b6-9a76-4827-a860-47f3a4596871",
__meta_kubernetes_service_label_app": "prometheus-coo-example-app",
__meta_kubernetes_service_labelpresent_app": "true",
__meta_kubernetes_service_name": "prometheus-coo-example-app",
__metrics_path__: "/metrics",
__scheme__: "http",
__scrape_interval__: "30s",
__scrape_timeout__: "10s",
"job": "serviceMonitor/ns1-coo/prometheus-coo-example-monitor/0"
}

```



#### NOTE

The above example uses [jq command-line JSON processor](#) to format the output for convenience.

## 4.4. VALIDATING THE MONITORING STACK

To validate that the monitoring stack is working correctly, access the example service and then view the gathered metrics.

### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role or as a user with administrative permissions for the namespace.
- You have installed the Cluster Observability Operator.
- You have deployed the **prometheus-coo-example-app** sample service in the **ns1-coo** namespace.
- You have created a **ServiceMonitor** object named **prometheus-coo-example-monitor** in the **ns1-coo** namespace.
- You have created a **MonitoringStack** object named **example-coo-monitoring-stack** in the **ns1-coo** namespace.

### Procedure

1. Create a route to expose the example **prometheus-coo-example-app** service. From your terminal, run the command:

```
$ oc expose svc prometheus-coo-example-app -n ns1-coo
```

2. Access the route from your browser, or command line, to generate metrics.

3. Execute a query on the Prometheus pod to return the total HTTP requests metric:

```
$ oc -n ns1-coo exec -c prometheus prometheus-example-coo-monitoring-stack-0 -- curl -s
'http://localhost:9090/api/v1/query?query=http_requests_total'
```

#### Example output (formatted using jq for convenience)

```
{
  "status": "success",
  "data": {
    "resultType": "vector",
    "result": [
      {
        "metric": {
          "__name__": "http_requests_total",
          "code": "200",
          "endpoint": "web",
          "instance": "10.129.2.25:8080",
          "job": "prometheus-coo-example-app",
          "method": "get",
          "namespace": "ns1-coo",
          "pod": "prometheus-coo-example-app-5d8cd498c7-9j2gj",
          "service": "prometheus-coo-example-app"
        },
        "value": [
          1730807483.632,
          "3"
        ]
      },
      {
        "metric": {
          "__name__": "http_requests_total",
          "code": "404",
          "endpoint": "web",
          "instance": "10.129.2.25:8080",
          "job": "prometheus-coo-example-app",
          "method": "get",
          "namespace": "ns1-coo",
          "pod": "prometheus-coo-example-app-5d8cd498c7-9j2gj",
          "service": "prometheus-coo-example-app"
        },
        "value": [
          1730807483.632,
          "0"
        ]
      }
    ]
  }
}
```

## CHAPTER 5. OBSERVABILITY UI PLUGINS

### 5.1. OBSERVABILITY UI PLUGINS OVERVIEW

You can use the Cluster Observability Operator (COO) to install and manage UI plugins to enhance the observability capabilities of the OpenShift Container Platform web console. The plugins extend the default functionality, providing new UI features for troubleshooting, distributed tracing, and cluster logging.

#### 5.1.1. Monitoring

The monitoring UI plugin adds monitoring related UI features to the OpenShift web console, for the Advance Cluster Management (ACM) perspective and for incident detection.

- **ACM:** The monitoring plugin in Cluster Observability Operator (COO) allows it to function in Red Hat Advanced Cluster Management (RHACM) environments, providing ACM with the same monitoring capabilities as OpenShift Container Platform.
- **Incident Detection:** The incident detection feature groups alerts into incidents to help you identify the root causes of alert bursts instead of being overwhelmed by individual alerts. It presents a timeline of incidents, color-coded by severity, and you can drill down into the individual alerts within an incident. The system also categorizes alerts by affected component to help you focus on the most critical areas first.

For more information, see the [monitoring UI plugin](#) page.

#### 5.1.2. Cluster logging

The logging UI plugin surfaces logging data in the web console on the **Observe → Logs** page. You can specify filters, queries, time ranges and refresh rates. The results displayed a list of collapsed logs, which can then be expanded to show more detailed information for each log.

For more information, see the [logging UI plugin](#) page.

#### 5.1.3. Troubleshooting




##### IMPORTANT

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

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

The troubleshooting panel UI plugin for OpenShift Container Platform version 4.16+ provides observability signal correlation, powered by the open source Korrel8r project. You can use the troubleshooting panel available from the **Observe → Alerting** page to easily correlate metrics, logs,



alerts, netflows, and additional observability signals and resources, across different data stores. Users of OpenShift Container Platform version 4.17+ can also access the troubleshooting UI panel from the Application Launcher .

The output of Korrel8r is displayed as an interactive node graph. When you click on a node, you are automatically redirected to the corresponding web console page with the specific information for that node, for example, metric, log, or pod.

For more information, see the [troubleshooting UI plugin](#) page.

#### 5.1.4. Distributed tracing



##### IMPORTANT

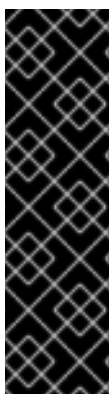
The Cluster Observability Operator distributed tracing UI plugin is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These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

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

The distributed tracing UI plugin adds tracing-related features to the web console on the **Observe → Traces** page. You can follow requests through the front end and into the backend of microservices, helping you identify code errors and performance bottlenecks in distributed systems. You can select a supported **TempoStack** or **TempoMonolithic** multi-tenant instance running in the cluster and set a time range and query to view the trace data.

For more information, see the [distributed tracing UI plugin](#) page.

## 5.2. MONITORING UI PLUGIN



##### IMPORTANT

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

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

The monitoring UI plugin adds monitoring features to the Administrator perspective of the OpenShift web console.

- **RHACM:** The monitoring plugin in Cluster Observability Operator (COO) allows it to function in Red Hat Advanced Cluster Management (RHACM) environments, providing RHACM with the same alerting capabilities as OpenShift Container Platform. You can configure the plugin to

fetch alerts from the RHACM Alertmanager backend. This enables seamless integration and user experience by aligning RHACM and OpenShift Container Platform monitoring workflows.

- **Incident detection:** The incident detection feature groups related alerts into incidents, to help you identify the root causes of alert bursts, instead of being overwhelmed by individual alerts. It presents a timeline of incidents, color-coded by severity, and you can drill down into the individual alerts within an incident. The system also categorizes alerts by affected component, grouped by severity. This helps you focus on the most critical areas first. The incident detection feature is available in the Administrator perspective of the OpenShift web console at **Observe → Incidents**.

### 5.2.1. Installing the Cluster Observability Operator monitoring UI plugin

The monitoring UI plugin adds monitoring related UI features to the OpenShift web console, for the Advance Cluster Management (ACM) perspective and for incident detection.

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have logged in to the OpenShift Container Platform web console.
- You have installed the Cluster Observability Operator

#### Procedure

1. In the OpenShift Container Platform web console, click **Operators → Installed Operators** and select Cluster Observability Operator
2. Choose the **UI Plugin** tab (at the far right of the tab list) and press **Create UIPlugin**
3. Select **YAML view**, enter the following content, and then press **Create**:

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: monitoring
spec:
  type: Monitoring
  monitoring:
    acm: 1
    enabled: true
    alertmanager:
      url: 'https://alertmanager.open-cluster-management-observability.svc:9095'
    thanosQuerier:
      url: 'https://rbac-query-proxy.open-cluster-management-observability.svc:8443'
  incidents: 2
    enabled: true
```

- 1 Enable RHACM features. You must configure the Alertmanager and ThanosQuerier Service endpoints.
- 2 Enable incident detection features.

### 5.2.2. Cluster Observability Operator incident detection overview

Clusters can generate significant volumes of monitoring data, making it hard for you to distinguish critical signals from noise. Single incidents can trigger a cascade of alerts, and this results in extended time to detect and resolve issues.

The Cluster Observability Operator incident detection feature groups related alerts into **incidents**. These incidents are then visualized as timelines that are color-coded by severity. Alerts are mapped to specific components, grouped by severity, helping you to identify root causes by focusing on high impact components first. You can then drill down from the incident timelines to individual alerts to determine how to fix the underlying issue.

Cluster Observability Operator incident detection transforms the alert storm into clear steps for faster understanding and resolution of the incidents that occur on your clusters.

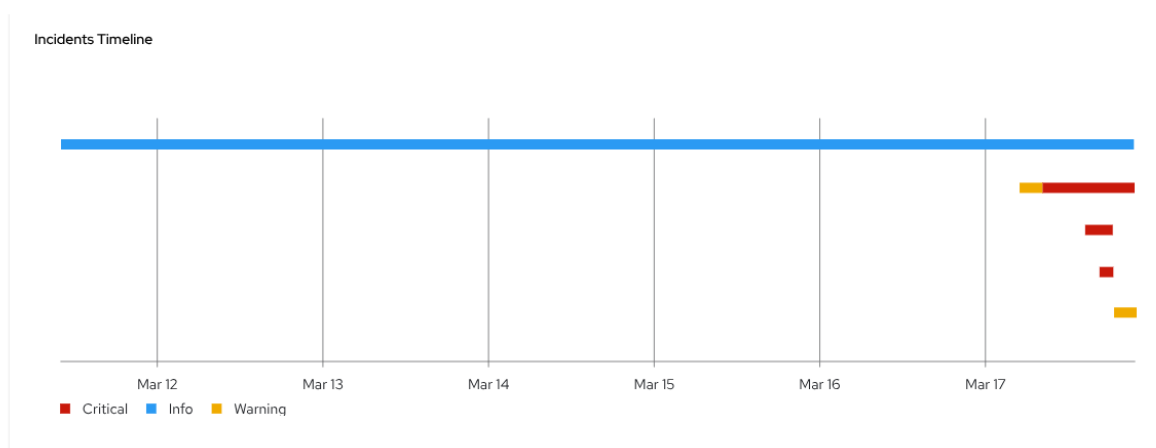
### 5.2.3. Using Cluster Observability Operator incident detection

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have logged in to the OpenShift Container Platform web console.
- You have installed the Cluster Observability Operator.
- You have installed the Cluster Observability Operator monitoring UI plugin with incident detection enabled.

#### Procedure

1. In the Administrator perspective of the web console, click on **Observe → Incidents**.
2. The Incidents Timeline UI shows the grouping of alerts into **incidents**. The color coding of the lines in the graph corresponds to the severity of the incident. By default, a seven day timeline is presented.



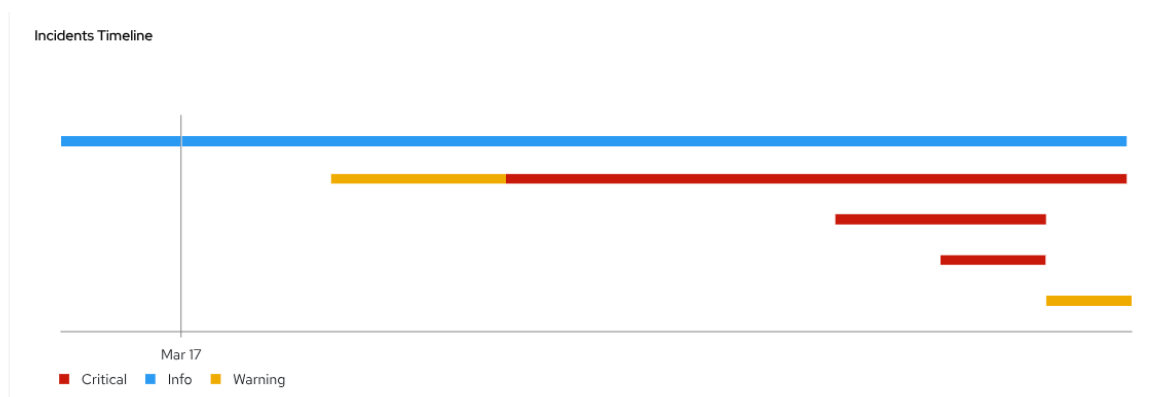


## NOTE

It will take at least 10 minutes to process the correlations and to see the timeline, after you enable incident detection.

The analysis and grouping into incidents is performed only for alerts that are firing after you have enabled this feature. Alerts that have been resolved before feature enablement are not included.

- Zoom in to a 1-day view by clicking on the drop-down to specify the duration.



- By clicking on an incident, you can see the timeline of alerts that are part of that incident, in the Alerts Timeline UI.



- In the list of alerts that follows, alerts are mapped to specific components, which are grouped by severity.

Component	Severity	State
> etcd	1 2	
> kube-apiserver	1	
> kube-controller-manager	1	
> compute	2	

6. Click to expand a compute component in the list. The underlying alerts related to that component are displayed.

Component	Severity	State
> etcd	1 2	
> kube-apiserver	1	
> kube-controller-manager	1	
▼ compute	2	

Alert Name	Namespace	Severity	State	Start	End
KubeNodeNotReady	openshift-monitoring	Warning	Firing	17 Mar 2025, 05:06	---
KubeNodeUnreachable	openshift-monitoring	Warning	Firing	17 Mar 2025, 05:06	---

7. Click the link for a firing alert, to see detailed information about that alert.



## NOTE

### Known issues

- Depending on the order of the timeline bars, the tooltip might overlap and hide the underlying bar. You can still click the bar and select the incident or alert.
- The Silence Alert button in the **Incidents** → **Component** section does not pre-populate the fields and is not usable. As a workaround, you can use the same menu and the Silence Alert button in the **Alerting** section instead.

## 5.3. LOGGING UI PLUGIN

The logging UI plugin surfaces logging data in the OpenShift Container Platform web console on the **Observe** → **Logs** page. You can specify filters, queries, time ranges and refresh rates, with the results displayed as a list of collapsed logs, which can then be expanded to show more detailed information for each log.

When you have also deployed the Troubleshooting UI plugin on OpenShift Container Platform version 4.16+, it connects to the Korrel8r service and adds direct links from the Administration perspective, from

the **Observe → Logs** page, to the **Observe → Metrics** page with a correlated PromQL query. It also adds a **See Related Logs** link from the Administration perspective alerting detail page, at **Observe → Alerting**, to the **Observe → Logs** page with a correlated filter set selected.

The features of the plugin are categorized as:

#### dev-console

Adds the logging view to the Developer perspective.

#### alerts

Merges the web console alerts with log-based alerts defined in the Loki ruler. Adds a log-based metrics chart in the alert detail view.

#### dev-alerts

Merges the web console alerts with log-based alerts defined in the Loki ruler. Adds a log-based metrics chart in the alert detail view for the Developer perspective.

For Cluster Observability Operator (COO) versions, the support for these features in OpenShift Container Platform versions is shown in the following table:

COO version	OCP versions	Features
0.3.0+	4.12	<b>dev-console</b>
0.3.0+	4.13	<b>dev-console, alerts</b>
0.3.0+	4.14+	<b>dev-console, alerts, dev-alerts</b>

### 5.3.1. Installing the Cluster Observability Operator logging UI plugin

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have logged in to the OpenShift Container Platform web console.
- You have installed the Cluster Observability Operator.
- You have a **LokiStack** instance in your cluster.

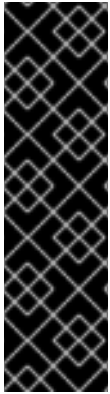
#### Procedure

1. In the OpenShift Container Platform web console, click **Operators → Installed Operators** and select Cluster Observability Operator.
2. Choose the **UI Plugin** tab (at the far right of the tab list) and click **Create UIPlugin**.
3. Select **YAML view**, enter the following content, and then click **Create**:

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: logging
```

```
spec:
  type: Logging
  logging:
    lokiStack:
      name: logging-loki
    logsLimit: 50
    timeout: 30s
```

## 5.4. DISTRIBUTED TRACING UI PLUGIN



### IMPORTANT

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

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

The distributed tracing UI plugin adds tracing-related features to the Administrator perspective of the OpenShift web console at **Observe → Traces**. You can follow requests through the front end and into the backend of microservices, helping you identify code errors and performance bottlenecks in distributed systems.

### 5.4.1. Installing the Cluster Observability Operator distributed tracing UI plugin

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have logged in to the OpenShift Container Platform web console.
- You have installed the Cluster Observability Operator

#### Procedure

1. In the OpenShift Container Platform web console, click **Operators → Installed Operators** and select Cluster Observability Operator
2. Choose the **UI Plugin** tab (at the far right of the tab list) and press **Create UIPlugin**
3. Select **YAML view**, enter the following content, and then press **Create**:

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: distributed-tracing
spec:
  type: DistributedTracing
```

## 5.4.2. Using the Cluster Observability Operator distributed tracing UI plugin

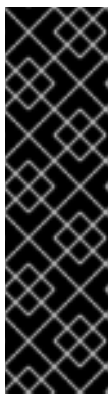
### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have logged in to the OpenShift Container Platform web console.
- You have installed the Cluster Observability Operator.
- You have installed the Cluster Observability Operator distributed tracing UI plugin.
- You have a **TempoStack** or **TempoMonolithic** multi-tenant instance in the cluster.

### Procedure

1. In the Administrator perspective of the OpenShift Container Platform web console, click **Observe → Traces**.
2. Select a **TempoStack** or **TempoMonolithic** multi-tenant instance and set a time range and query for the traces to be loaded.  
The traces are displayed on a scatter-plot showing the trace start time, duration, and number of spans. Underneath the scatter plot, there is a list of traces showing information such as the **Trace Name**, number of **Spans**, and **Duration**.
3. Click on a trace name link.  
The trace detail page for the selected trace contains a Gantt Chart of all of the spans within the trace. Select a span to show a breakdown of the configured attributes.


## 5.5. TROUBLESHOOTING UI PLUGIN



### IMPORTANT

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

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

The troubleshooting UI plugin for OpenShift Container Platform version 4.16+ provides observability signal correlation, powered by the open source Korrel8r project. With the troubleshooting panel that is available under **Observe → Alerting**, you can easily correlate metrics, logs, alerts, netflows, and additional observability signals and resources, across different data stores. Users of OpenShift Container Platform version 4.17+ can also access the troubleshooting UI panel from the Application Launcher .

When you install the troubleshooting UI plugin, a [Korrel8r](#) service named **korrel8r** is deployed in the same namespace, and it is able to locate related observability signals and Kubernetes resources from its correlation engine.



The output of Korrel8r is displayed in the form of an interactive node graph in the OpenShift Container Platform web console. Nodes in the graph represent a type of resource or signal, while edges represent relationships. When you click on a node, you are automatically redirected to the corresponding web console page with the specific information for that node, for example, metric, log, pod.

### 5.5.1. Installing the Cluster Observability Operator Troubleshooting UI plugin

#### Prerequisites

- You have access to the cluster as a user with the **cluster-admin** cluster role.
- You have logged in to the OpenShift Container Platform web console.
- You have installed the Cluster Observability Operator


#### Procedure

1. In the OpenShift Container Platform web console, click **Operators** → **Installed Operators** and select Cluster Observability Operator
2. Choose the **UI Plugin** tab (at the far right of the tab list) and press **Create UIPlugin**
3. Select **YAML view**, enter the following content, and then press **Create**:

```
apiVersion: observability.openshift.io/v1alpha1
kind: UIPlugin
metadata:
  name: troubleshooting-panel
spec:
  type: TroubleshootingPanel
```

### 5.5.2. Using the Cluster Observability Operator troubleshooting UI plugin

#### Prerequisites

- You have access to the OpenShift Container Platform cluster as a user with the **cluster-admin** cluster role. If your cluster version is 4.17+, you can access the troubleshooting UI panel from the Application Launcher .
- You have logged in to the OpenShift Container Platform web console.
- You have installed OpenShift Container Platform Logging, if you want to visualize correlated logs.
- You have installed OpenShift Container Platform Network Observability, if you want to visualize correlated netflows.
- You have installed the Cluster Observability Operator.
- You have installed the Cluster Observability Operator troubleshooting UI plugin.



NOTE

- The troubleshooting panel relies on the observability signal stores installed in your cluster. Kubernetes resources, alerts and metrics are always available by default in an OpenShift Container Platform cluster. Other signal types require optional components to be installed:
- **Logs:** Red Hat Openshift Logging (collection) and Loki Operator provided by Red Hat (store)
  - **Network events:** Network observability provided by Red Hat (collection) and Loki Operator provided by Red Hat (store)

Procedure

1. In the admin perspective of the web console, navigate to **Observe → Alerting** and then select an alert. If the alert has correlated items, a **Troubleshooting Panel** link will appear above the chart on the alert detail page.



Click on the **Troubleshooting Panel** link to display the panel.

2. The panel consists of query details and a topology graph of the query results. The selected alert is converted into a Korrel8r query string and sent to the **korrel8r** service. The results are displayed as a graph network connecting the returned signals and resources. This is a *neighbourhood* graph, starting at the current resource and including related objects up to 3 steps away from the starting point. Clicking on nodes in the graph takes you to the corresponding web console pages for those resources.
3. You can use the troubleshooting panel to find resources relating to the chosen alert.

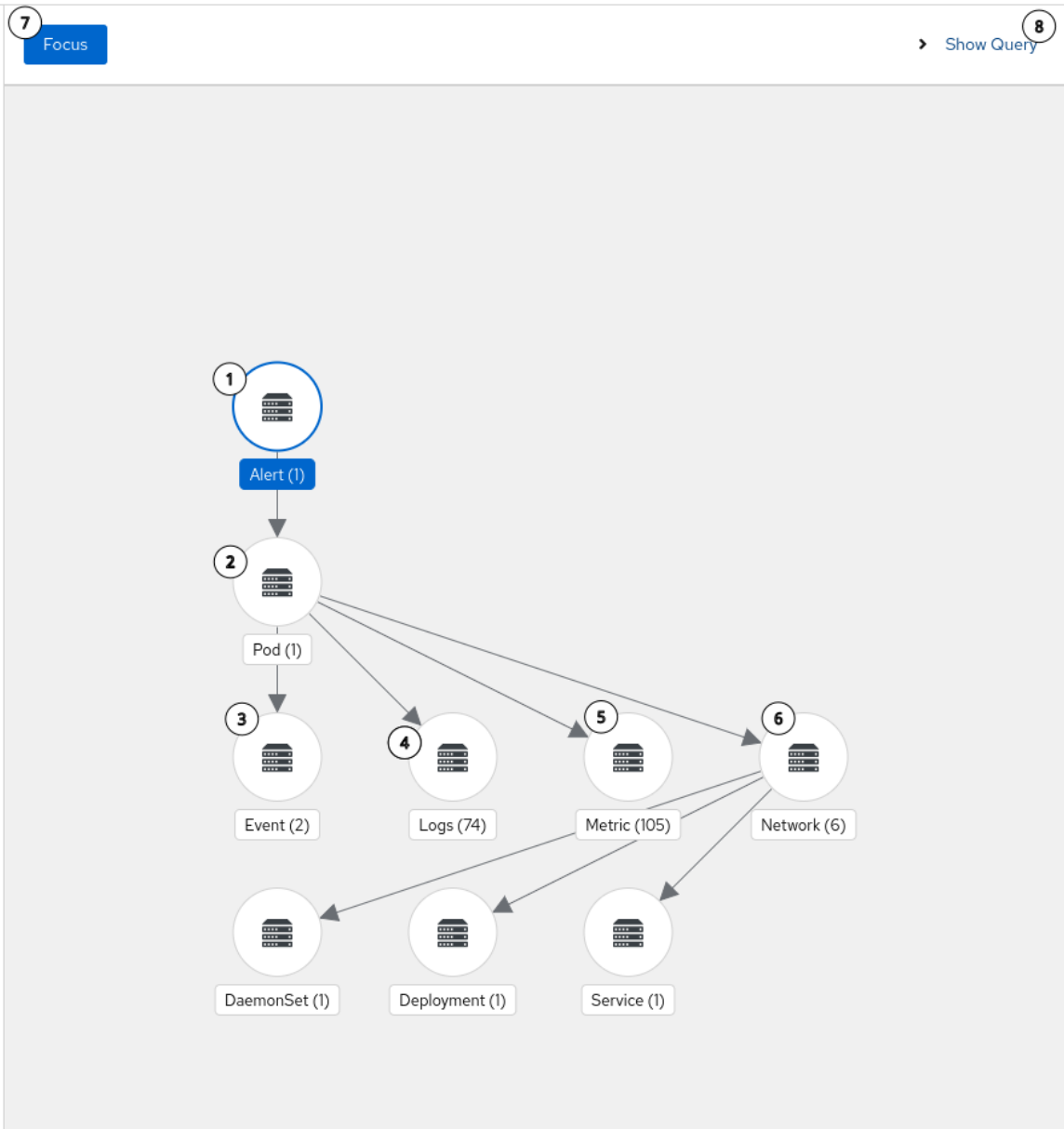


NOTE

Clicking on a node may sometimes show fewer results than indicated on the graph. This is a known issue that will be addressed in a future release.

## Troubleshooting

Find related resources.



1. **Alert (1):** This node is the starting point in the graph and represents the **KubeContainerWaiting** alert displayed in the web console.
2. **Pod (1):** This node indicates that there is a single **Pod** resource associated with this alert. Clicking on this node will open a console search showing the related pod directly.
3. **Event (2):** There are two Kubernetes events associated with the pod. Click this node to see the events.
4. **Logs (74):** This pod has 74 lines of logs, which you can access by clicking on this node.
5. **Metrics (105):** There are many metrics associated with the pod.
6. **Network (6):** There are network events, meaning the pod has communicated over the network. The remaining nodes in the graph represent the **Service**, **Deployment** and **DaemonSet** resources that the pod has communicated with.
7. **Focus:** Clicking this button updates the graph. By default, the graph itself does not change when you click on nodes in the graph. Instead, the main web console page

changes, and you can then navigate to other resources using links on the page, while the troubleshooting panel itself stays open and unchanged. To force an update to the graph in the troubleshooting panel, click **Focus**. This draws a new graph, using the current resource in the web console as the starting point.

8. **Show Query:** Clicking this button enables some experimental features:

1. **Hide Query** hides the experimental features.
2. The query that identifies the starting point for the graph. The query language, part of the [Korrel8r](#) correlation engine used to create the graphs, is experimental and may change in future. The query is updated by the **Focus** button to correspond to the resources in the main web console window.
3. **Neighbourhood depth** is used to display a smaller or larger neighbourhood.



#### NOTE

Setting a large value in a large cluster might cause the query to fail, if the number of results is too big.

4. **Goal class** results in a goal directed search instead of a neighbourhood search. A goal directed search shows all paths from the starting point to the goal class, which indicates a type of resource or signal. The format of the goal class is experimental and may change. Currently, the following goals are valid:
  - **k8s:RESOURCE[VERSION.[GROUP]]** identifying a kind of kubernetes resource. For example **k8s:Pod** or **k8s:Deployment.apps.v1**.
  - **alert:alert** representing any alert.
  - **metric:metric** representing any metric.
  - **netflow:network** representing any network observability network event.
  - **log:LOG\_TYPE** representing stored logs, where **LOG\_TYPE** must be one of **application**, **infrastructure** or **audit**.

### 5.5.3. Creating the example alert

To trigger an alert as a starting point to use in the troubleshooting UI panel, you can deploy a container that is deliberately misconfigured.

#### Procedure

1. Use the following YAML, either from the command line or in the web console, to create a broken deployment in a system namespace:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bad-deployment
  namespace: default 1
spec:
  selector:
    matchLabels:
      app: bad-deployment
  template:
    metadata:
      labels:
        app: bad-deployment
    spec:
      containers: 2
      - name: bad-deployment
        image: quay.io/openshift-logging/vector:5.8
```

- 1** The deployment must be in a system namespace (such as **default**) to cause the desired alerts.
- 2** This container deliberately tries to start a **vector** server with no configuration file. The server logs a few messages, and then exits with an error. Alternatively, you can deploy any container you like that is badly configured, causing it to trigger an alert.

2. View the alerts:

- a. Go to **Observe → Alerting** and click **clear all filters**. View the **Pending** alerts.



### IMPORTANT

Alerts first appear in the **Pending** state. They do not start **Firing** until the container has been crashing for some time. By viewing **Pending** alerts, you do not have to wait as long to see them occur.

- b. Choose one of the **KubeContainerWaiting**, **KubePodCrashLooping**, or **KubePodNotReady** alerts and open the troubleshooting panel by clicking on the link. Alternatively, if the panel is already open, click the "Focus" button to update the graph.