# OpenShift Container Platform 4.14

## Cluster Observability Operator

Configuring and using the Cluster Observability Operator in OpenShift Container Platform

Last Updated: 2024-12-06

# OpenShift Container Platform 4.14 Cluster Observability Operator

Configuring and using the Cluster Observability Operator in OpenShift Container Platform

## Legal Notice



## Abstract

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

# Table of Contents

# CHAPTER 1. CLUSTER OBSERVABILITY OPERATOR RELEASE NOTES

**IMPORTANT**

The Cluster Observability Operator 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 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 | Dashboards | Distributed Tracing | Logging | Troubleshooting Panel |
|---|---|---|---|---|---|
| 0.2.0 | 4.11 | ✔ | ✘ | ✘ | ✘ |
| 0.3.0+ | 4.11 - 4.15 | ✔ | ✔ | ✔ | ✘ |
| 0.3.0+ | 4.16+ | ✔ | ✔ | ✔ | ✔ |

## 1.1. 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

### 1.1.1. New features and enhancements

- You can now configure WebTLS for Prometheus and Alertmanager.

### 1.1.2. CVEs

- CVE-2024-6104

- CVE-2024-24786

### 1.1.3. 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)

## 1.2. 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

### 1.2.1. New features and enhancements

#### 1.2.1.1. 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.

#### 1.2.1.2. 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.

### 1.2.2. 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.

## 1.3. 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

### 1.3.1. New features and enhancements

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

### 1.3.2. 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)

## 1.4. 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

### 1.4.1. 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.

## 1.5. 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

### 1.5.1. 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)

## 1.6. 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

### 1.6.1. 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)

## 1.7. 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

### 1.7.1. CVEs

- CVE-2023-45142

### 1.7.2. 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)

## 1.8. 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

### 1.8.1. New features and enhancements

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

## 1.9. 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

**IMPORTANT**

The Cluster Observability Operator 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 Cluster Observability Operator (COO) is an optional component of the OpenShift Container Platform. You can deploy it to create standalone monitoring stacks that are independently configurable for use by different services and users.

The COO deploys the following monitoring components:

- Prometheus
- Thanos Querier (optional)
- Alertmanager (optional)

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.

## 2.1. UNDERSTANDING THE CLUSTER OBSERVABILITY OPERATOR

A default monitoring stack created by the Cluster Observability Operator (COO) includes a highly available Prometheus instance capable of sending metrics to an external endpoint by using remote write.

Each COO stack also includes an optional Thanos Querier component, which you can use to query a highly available Prometheus instance from a central location, and an optional Alertmanager component, which you can use to set up alert configurations for different services.

### 2.1.1. Advantages of using the Cluster Observability Operator

The **MonitoringStack** CRD used by the COO offers an opinionated default monitoring configuration for COO-deployed monitoring components, but you can customize it to suit more complex requirements.

Deploying a COO-managed monitoring stack can help meet monitoring needs that are difficult or impossible to address by using the core platform monitoring stack deployed by the Cluster Monitoring Operator (CMO). A monitoring stack deployed using COO has the following advantages over core platform and user workload monitoring:

**Extendability**

Users can add more metrics to a COO-deployed monitoring stack, which is not possible with core platform monitoring without losing support. In addition, COO-managed stacks can receive certain cluster-specific metrics from core platform monitoring by using federation.

**Multi-tenancy support**

The COO can create a monitoring stack per user namespace. You can also deploy multiple stacks per namespace or a single stack for multiple namespaces. For example, cluster administrators, SRE teams, and development teams can all deploy their own monitoring stacks on a single cluster, rather than having to use a single shared stack of monitoring components. Users on different teams can then independently configure features such as separate alerts, alert routing, and alert receivers for their applications and services.

**Scalability**

You can create COO-managed monitoring stacks as needed. Multiple monitoring stacks can run on a single cluster, which can facilitate the monitoring of very large clusters by using manual sharding. This ability addresses cases where the number of metrics exceeds the monitoring capabilities of a single Prometheus instance.

**Flexibility**

Deploying the COO with Operator Lifecycle Manager (OLM) decouples COO releases from OpenShift Container Platform release cycles. This method of deployment enables faster release iterations and the ability to respond rapidly to changing requirements and issues. Additionally, by deploying a COO-managed monitoring stack, users can manage alerting rules independently of OpenShift Container Platform release cycles.

**Highly customizable**

The COO can delegate ownership of single configurable fields in custom resources to users by using Server-Side Apply (SSA), which enhances customization.

**Additional resources**

- Kubernetes documentation for Server-Side Apply (SSA)

# CHAPTER 3. INSTALLING THE CLUSTER OBSERVABILITY OPERATOR

**IMPORTANT**

The Cluster Observability Operator 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 .

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 review the following default installation settings:
   - **Update channel → development**
   - **Version →** <most_recent_version>
   - **Installation mode → All namespaces on the cluster (default)**
   - **Installed Namespace → openshift-operators**
   - **Update approval → Automatic**

5. Optional: Change default 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

**IMPORTANT**

The Cluster Observability Operator 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 .

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

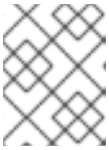
To use the metrics exposed by the sample service you created in the "Deploying a sample service for Cluster Observability Operator" section, you must configure monitoring components to scrape metrics from the /**metrics** endpoint.

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
```

# CHAPTER 5. OBSERVABILITY UI PLUGINS

## 5.1. OBSERVABILITY UI PLUGINS OVERVIEW

**IMPORTANT**

The Cluster Observability Operator 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 .

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 monitoring, troubleshooting, distributed tracing, and cluster logging.

### 5.1.1. Dashboards

The dashboard UI plugin supports enhanced dashboards in the OpenShift Container Platform web console at **Observe → Dashboards**. You can add other Prometheus data sources from the cluster to the default dashboards, in addition to the in-cluster data source. This results in a unified observability experience across different data sources.

For more information, see the dashboard UI plugin page.

### 5.1.2. Troubleshooting

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.

### 5.1.3. Distributed tracing

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.

### 5.1.4. 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.

## 5.2. DASHBOARD UI PLUGIN

**IMPORTANT**

The Cluster Observability Operator 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 dashboard UI plugin supports enhanced dashboards in the OpenShift web console at **Observe → Dashboards** . You can add other Prometheus datasources from the cluster to the default dashboards, in addition to the in-cluster datasource. This results in a unified observability experience across different data sources.

The plugin searches for datasources from **ConfigMap** resources in the **openshift-config-managed** namespace, that have the label **console.openshift.io/dashboard-datasource: 'true'**.

### 5.2.1. Installing the Cluster Observability Operator dashboard 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: dashboards
spec:
  type: Dashboards
```

### 5.2.2. Configuring a dashboard

The dashboard UI plugin searches for datasources from **ConfigMap** resources in the **openshift-config-managed** namespace, that have the label **console.openshift.io/dashboard-datasource: 'true'**. The **ConfigMap** resource must define a datasource type and an in-cluster service where the data can be fetched.

The examples in the following section are taken from https://github.com/openshift/console-dashboards-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 dashboard UI plugin.

#### Procedure

1. Create a **ConfigMap** resource in the **openshift-config-managed** namespace, with the label **console.openshift.io/dashboard-datasource: 'true'**. The example below is from prometheus-datasource-example.yaml

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-prometheus-proxy
  namespace: openshift-config-managed
  labels:
    console.openshift.io/dashboard-datasource: "true"
data:
  "dashboard-datasource.yaml": |-
    kind: "Datasource"
    metadata:
      name: "cluster-prometheus-proxy"
      project: "openshift-config-managed"
    spec:
      plugin:
        kind: "prometheus"
        spec:
          direct_url: "https://prometheus-k8s.openshift-monitoring.svc.cluster.local:9091"
```

2. Configure a custom dashboard that connects to the datasource. The YAML for a sample dashboard is available at prometheus-dashboard-example.yaml. An excerpt from that file is shown below for demonstration purposes:

**Example 5.1. Extract from example dashboard, taken from prometheus-dashboard-example.yaml**

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dashboard-example
  namespace: openshift-config-managed
  labels:
    console.openshift.io/dashboard: "true"
data:
  k8s-resources-workloads-namespace.json: |-
    {
       "annotations": {
          "list": [

          ]
       },
       "editable": true,
       "gnetId": null,
       "graphTooltip": 0,
       "hideControls": false,
       "links": [

       ],
       "refresh": "10s",
       "rows": [
          {
             "collapse": false,
             "height": "250px",
             "panels": [
                {
                   "aliasColors": {

                   },
                   "bars": false,
                   "dashLength": 10,
                   "dashes": false,
                   "datasource": {
                      "name": "cluster-prometheus-proxy",
                      "type": "prometheus"
                   },
                   "fill": 10,
                   "id": 1,
                   "interval": "1m",
                   "legend": {
                      "alignAsTable": true,
                      "avg": false,
                      "current": false,
                      "max": false,
                      "min": false,
                      "rightSide": true,
                      "show": true,
                      "total": false,
                      "values": false
                   },
                   "lines": true,
                   "linewidth": 0,
                   "links": [
```

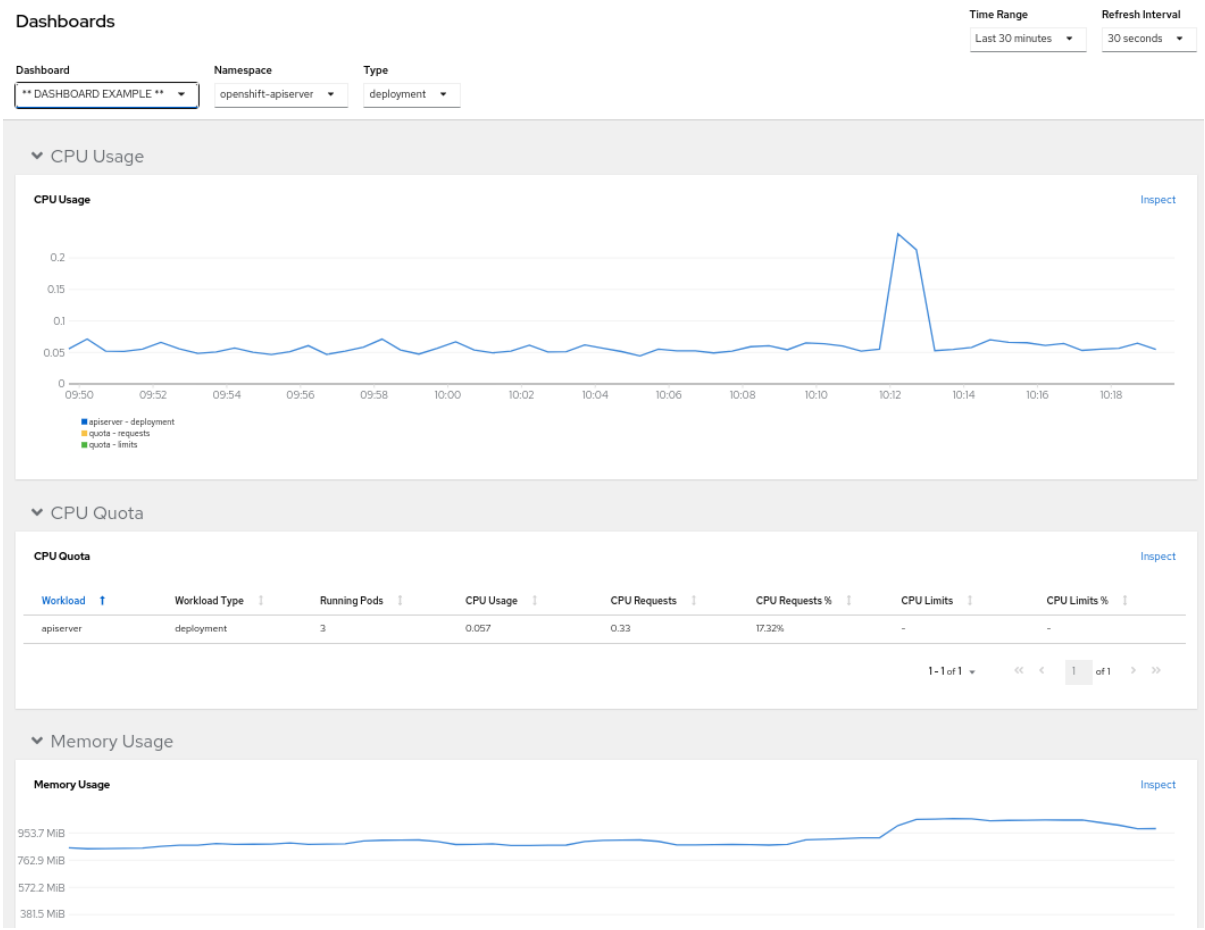
```
                ],
                "nullPointMode": "null as zero",
                "percentage": false,
                "pointradius": 5,
                "points": false,
                "renderer": "flot",
                "seriesOverrides": [
                  {
                    "alias": "quota - requests",
                    "color": "#F2495C",
                    "dashes": true,
                    "fill": 0,
                    "hiddenSeries": true,
                    "hideTooltip": true,
                    "legend": true,
                    "linewidth": 2,
                    "stack": false
                  },
                  {
                    "alias": "quota - limits",
                    "color": "#FF9830",
                    "dashes": true,
                    "fill": 0,
                    "hiddenSeries": true,
                    "hideTooltip": true,
                    "legend": true,
                    "linewidth": 2,
                    "stack": false
                  }
                ],
                "spaceLength": 10,
                "span": 12,
                "stack": false,
                "steppedLine": false,
                "targets": [
                  {
                    "expr": "sum(
node_namespace_pod_container:container_cpu_usage_seconds_total:sum_irate{cluster=\"
$cluster\", namespace=\"$namespace\"}* on(namespace,pod)  group_left(workload,
workload_type) namespace_workload_pod:kube_pod_owner:relabel{cluster=\"$cluster\",
namespace=\"$namespace\", workload_type=\"$type\"}) by (workload, workload_type)",
                    "format": "time_series",
                    "intervalFactor": 2,
                    "legendFormat": "{{workload}} - {{workload_type}}",
                    "legendLink": null,
                    "step": 10
                  },
                  {
                    "expr": "scalar(kube_resourcequota{cluster=\"$cluster\",
namespace=\"$namespace\", type=\"hard\",resource=\"requests.cpu\"})",
                    "format": "time_series",
                    "intervalFactor": 2,
                    "legendFormat": "quota - requests",
                    "legendLink": null,
                    "step": 10
```

```
                    },
                    {
                        "expr": "scalar(kube_resourcequota{cluster=\"$cluster\",
namespace=\"$namespace\", type=\"hard\",resource=\"limits.cpu\"})",
                        "format": "time_series",
                        "intervalFactor": 2,
                        "legendFormat": "quota - limits",
                        "legendLink": null,
                        "step": 10
                    }
                ],
                "thresholds": [

                ],
                "timeFrom": null,
                "timeShift": null,
                "title": "CPU Usage",
                "tooltip": {
                    "shared": false,
                    "sort": 2,
                    "value_type": "individual"
                },
                "type": "graph",
                "xaxis": {
                    "buckets": null,
                    "mode": "time",
                    "name": null,
                    "show": true,
                    "values": [

                    ]
                },
...
```

3. Click **Observe → Dashboards** and the custom dashboard is available with the title ** DASHBOARD EXAMPLE **, based on the configuration in **prometheus-dashboard-example.yaml**.

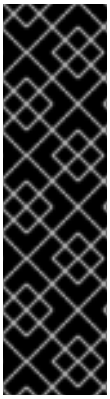



You can set the namespace, time range and refresh interval for the dashboard in the UI.

### 5.2.3. Additional resources

- See how to add a new datasource in the console-dashboards-plugin GitHub repository.

## 5.3. DISTRIBUTED TRACING UI PLUGIN



**IMPORTANT**

The Cluster Observability Operator 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.

### 5.3.1. 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
```

### 5.3.2. 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.

## 5.4. TROUBLESHOOTING UI PLUGIN

**IMPORTANT**

The Cluster Observability Operator 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.

### 5.4.1. 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
```

## 5.4.2. 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.

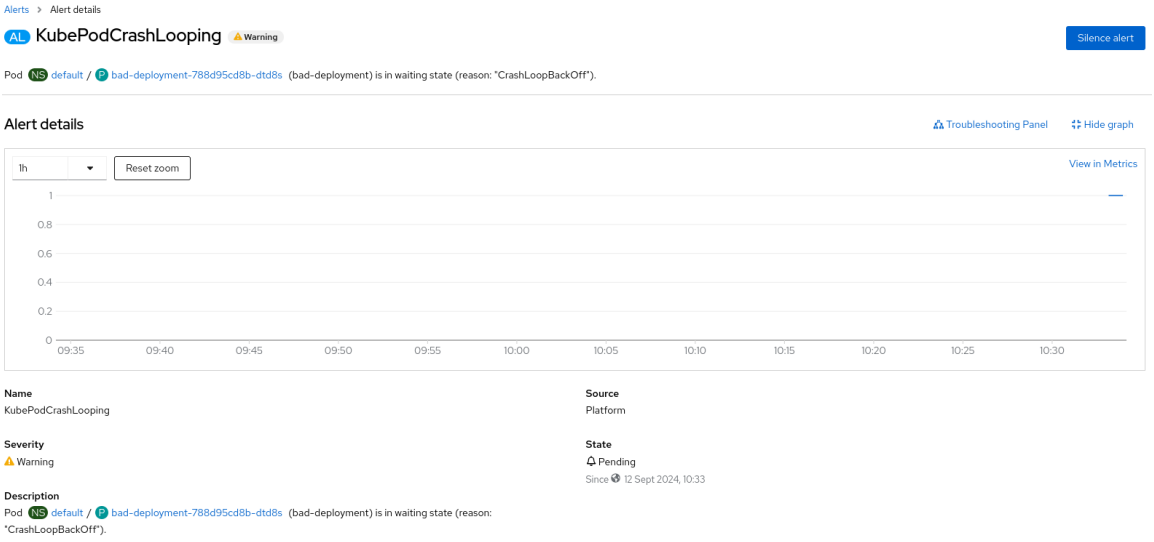
**NOTE**

The troubleshooting panel relies on the observability signal stores installed in your cluster. Kuberenetes 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.



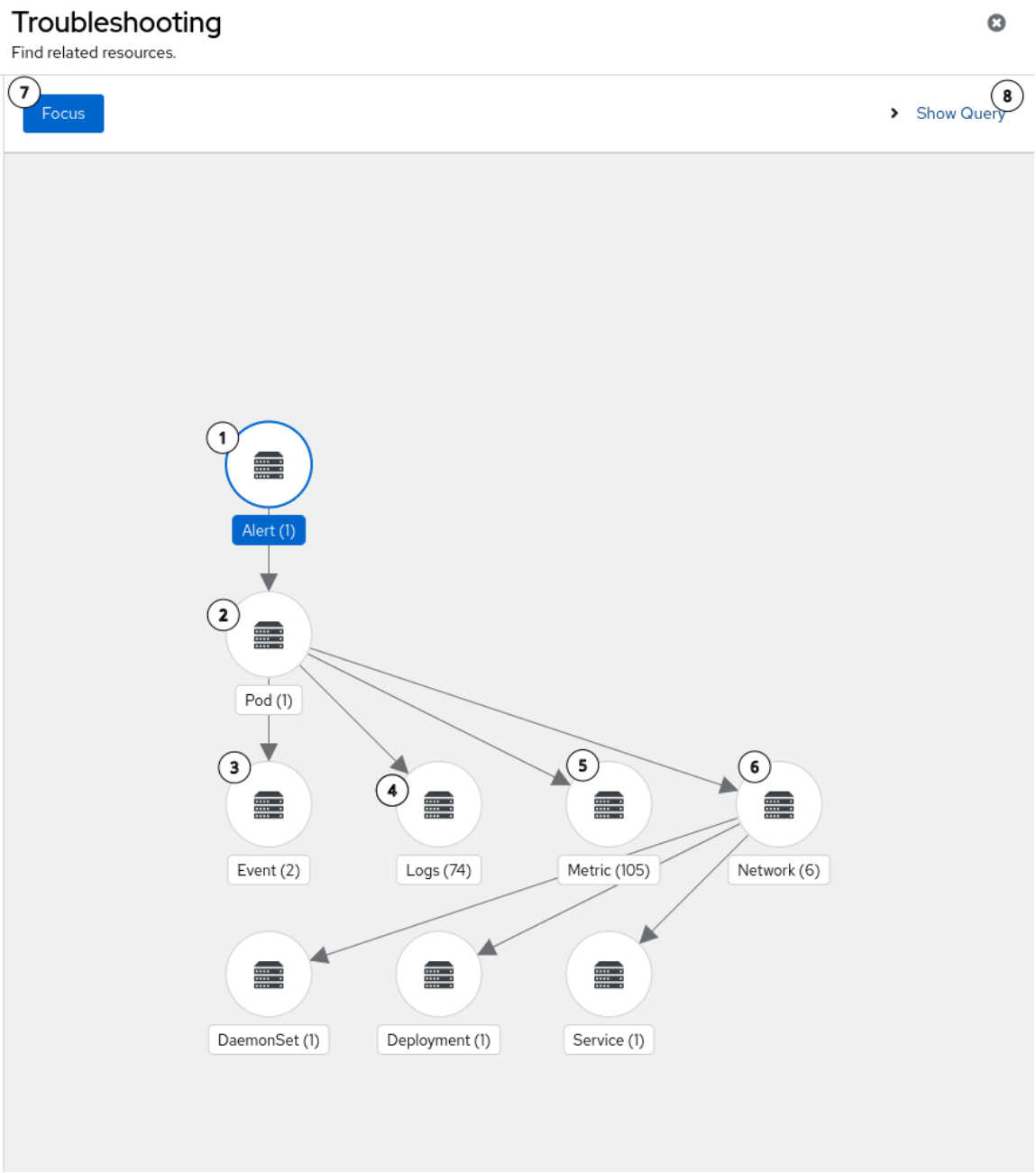

Click on the **Troubleshooting Panel** link to display the panel.

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 resouces.

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.




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 Kuberenetes 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.

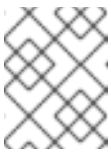
7. **Focus:** Clicking this button updates the graph. By default, the graph itself does not change when you click on nodes in the graph. Instead, the main web console page changes, and you can then navigate to other resources using links on the page, while the troubleshooting panel itself stays open 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 kuberenetes 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**.

### 5.4.3. 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.

## 5.5. LOGGING UI PLUGIN

### IMPORTANT

Until the approaching General Availability (GA) release of the Cluster Observability Operator (COO), which is currently in Technology Preview (TP), Red Hat provides support to customers who are using Logging 6.0 or later with the COO for the logging UI plugin on OpenShift Container Platform 4.14 or later. This support exception is temporary as the COO includes several independent features, some of which are still TP features, but the logging UI plugin is ready for GA.

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 | **dev-console** |
| 0.3.0+ | 4.13 | **dev-console**, **alerts** |
| 0.3.0+ | 4.14+ | **dev-console**, **alerts**, **dev-alerts** |

### 5.5.1. 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
```