Red Hat OpenShift Service on AWS 4

Web console

Getting started with web console in Red Hat OpenShift Service on AWS

Last Updated: 2024-04-22

Red Hat OpenShift Service on AWS 4 Web console

Getting started with web console in Red Hat OpenShift Service on AWS

Legal Notice

Abstract

This document provides instructions for accessing and customizing the OpenShift Service on AWS web console.

Table of Contents

CHAPTER 1. WEB CONSOLE OVERVIEW

The Red Hat Red Hat OpenShift Service on AWS web console provides a graphical user interface to visualize your project data and perform administrative, management, and troubleshooting tasks. The web console runs as pods on the control plane nodes in the openshift-console project. It is managed by a **console-operator** pod. Both **Administrator** and **Developer** perspectives are supported.

Both **Administrator** and **Developer** perspectives enable you to create quick start tutorials for Red Hat OpenShift Service on AWS. A quick start is a guided tutorial with user tasks and is useful for getting oriented with an application, Operator, or other product offering.

1.1. ABOUT THE ADMINISTRATOR PERSPECTIVE IN THE WEB CONSOLE

The **Administrator** perspective enables you to view the cluster inventory, capacity, general and specific utilization information, and the stream of important events, all of which help you to simplify planning and troubleshooting tasks. Both project administrators and cluster administrators can view the **Administrator** perspective.

Cluster administrators can also open an embedded command line terminal instance with the web terminal Operator in Red Hat OpenShift Service on AWS 4.7 and later.

NOTE

The default web console perspective that is shown depends on the role of the user. The **Administrator** perspective is displayed by default if the user is recognized as an administrator.

The **Administrator** perspective provides workflows specific to administrator use cases, such as the ability to:

- Manage workload, storage, networking, and cluster settings.
- Install and manage Operators using the Operator Hub.
- Add identity providers that allow users to log in and manage user access through roles and role bindings.
- View and manage a variety of advanced settings such as cluster updates, partial cluster updates, cluster Operators, custom resource definitions (CRDs), role bindings, and resource quotas.
- Access and manage monitoring features such as metrics, alerts, and monitoring dashboards.
- View and manage logging, metrics, and high-status information about the cluster.
- Visually interact with applications, components, and services associated with the **Administrator** perspective in Red Hat OpenShift Service on AWS.

1.2. ABOUT THE DEVELOPER PERSPECTIVE IN THE WEB CONSOLE

The **Developer** perspective offers several built-in ways to deploy applications, services, and databases. In the **Developer** perspective, you can:

- View real-time visualization of rolling and recreating rollouts on the component.

- View the application status, resource utilization, project event streaming, and quota consumption.
- Share your project with others.
- Troubleshoot problems with your applications by running Prometheus Query Language (PromQL) queries on your project and examining the metrics visualized on a plot. The metrics provide information about the state of a cluster and any user-defined workloads that you are monitoring.

Cluster administrators can also open an embedded command line terminal instance in the web console in Red Hat OpenShift Service on AWS 4.7 and later.

NOTE

The default web console perspective that is shown depends on the role of the user. The **Developer** perspective is displayed by default if the user is recognised as a developer.

The **Developer** perspective provides workflows specific to developer use cases, such as the ability to:

- Create and deploy applications on Red Hat OpenShift Service on AWS by importing existing codebases, images, and container files.
- Visually interact with applications, components, and services associated with them within a project and monitor their deployment and build status.
- Group components within an application and connect the components within and across applications.
- Integrate serverless capabilities (Technology Preview).
- Create workspaces to edit your application code using Eclipse Che.

You can use the **Topology** view to display applications, components, and workloads of your project. If you have no workloads in the project, the **Topology** view will show some links to create or import them. You can also use the **Quick Search** to import components directly.

Additional Resources

See Viewing application composition using the Topology view for more information on using the **Topology** view in **Developer** perspective.

1.3. ACCESSING THE PERSPECTIVES

You can access the **Administrator** and **Developer** perspective from the web console as follows:

Prerequisites

To access a perspective, ensure that you have logged in to the web console. Your default perspective is automatically determined by the permission of the users. The **Administrator** perspective is selected for users with access to all projects, while the **Developer** perspective is selected for users with limited access to their own projects

Additional Resources

See Adding User Preferences for more information on changing perspectives.

Procedure

1. Use the perspective switcher to switch to the **Administrator** or **Developer** perspective.
2. Select an existing project from the **Project** drop-down list. You can also create a new project from this dropdown.

NOTE

You can use the perspective switcher only as **cluster-admin**.

Additional resources

- Viewing cluster information
- Using the web terminal
- Creating quick start tutorials
- Disabling the web console

CHAPTER 2. ACCESSING THE WEB CONSOLE

The Red Hat OpenShift Service on AWS web console is a user interface accessible from a web browser. Developers can use the web console to visualize, browse, and manage the contents of projects.

2.1. PREREQUISITES

- JavaScript must be enabled to use the web console. For the best experience, use a web browser that supports WebSockets.
- Review the OpenShift Container Platform 4.x Tested Integrations page before you create the supporting infrastructure for your cluster.

2.2. UNDERSTANDING AND ACCESSING THE WEB CONSOLE

The web console runs as a pod on the control plane node. The static assets required to run the web console are served by the pod.

Procedure

1. Log in to OpenShift Cluster Manager and click the name of your cluster.
2. On the cluster **Overview** tab, click **Open console**, and log in with your credentials.

Alternatively, use the **oc whoami --show-console** command to get the web console URL.

CHAPTER 3. USING THE RED HAT OPENSHIFT SERVICE ON AWS DASHBOARD TO GET CLUSTER INFORMATION

The Red Hat OpenShift Service on AWS web console captures high-level information about the cluster.

3.1. ABOUT THE RED HAT OPENSHIFT SERVICE ON AWS DASHBOARDS PAGE

Access the Red Hat OpenShift Service on AWS dashboard, which captures high-level information about the cluster, by navigating to **Home → Overview** from the Red Hat OpenShift Service on AWS web console.

The Red Hat OpenShift Service on AWS dashboard provides various cluster information, captured in individual dashboard cards.

The Red Hat OpenShift Service on AWS dashboard consists of the following cards:

- **Details** provides a brief overview of informational cluster details.
  Status include **ok**, **error**, **warning**, **in progress**, and **unknown**. Resources can add custom status names.
  - Cluster ID
  - Provider
  - Version
- **Cluster Inventory** details number of resources and associated statuses. It is helpful when intervention is required to resolve problems, including information about:
  - Number of nodes
  - Number of pods
  - Persistent storage volume claims
  - Bare metal hosts in the cluster, listed according to their state (only available in **metal3** environment)
- **Status** helps administrators understand how cluster resources are consumed. Click on a resource to jump to a detailed page listing pods and nodes that consume the largest amount of the specified cluster resource (CPU, memory, or storage).
- **Cluster Utilization** shows the capacity of various resources over a specified period of time, to help administrators understand the scale and frequency of high resource consumption, including information about:
  - CPU time
  - Memory allocation
  - Storage consumed
  - Network resources consumed
  - Pod count

- **Activity** lists messages related to recent activity in the cluster, such as pod creation or virtual machine migration to another host.

3.2. RECOGNIZING RESOURCE AND PROJECT LIMITS AND QUOTAS

You can view a graphical representation of available resources in the **Topology** view of the web console **Developer** perspective.

If a resource has a message about resource limitations or quotas being reached, a yellow border appears around the resource name. Click the resource to open a side panel to see the message. If the **Topology** view has been zoomed out, a yellow dot indicates that a message is available.

If you are using **List View** from the **View Shortcuts** menu, resources appear as a list. The **Alerts** column indicates if a message is available.

CHAPTER 4. DYNAMIC PLUGINS

4.1. OVERVIEW OF DYNAMIC PLUGINS

4.1.1. About dynamic plugins

Dynamic plugins are deployed as workloads on the cluster. They allow you to add custom pages and other extensions to your console user interface at runtime. The **ConsolePlugin** custom resource registers plugins with the console, and a cluster administrator enables plugins in the **console-operator** configuration.

4.1.2. Key features

A dynamic plugin allows you to make the following customizations to the Red Hat OpenShift Service on AWS experience:

- Add custom pages.
- Add perspectives beyond administrator and developer.
- Add navigation items.
- Add tabs and actions to resource pages.

4.1.3. General guidelines

When creating your plugin, follow these general guidelines:

- **Node.js** and **yarn** are required to build and run your plugin.
- Prefix your CSS class names with your plugin name to avoid collisions. For example, **my-plugin__heading** and **my-plugin__icon**.
- Maintain a consistent look, feel, and behavior with other console pages.
- Follow react-i18next localization guidelines when creating your plugin. You can use the **useTranslation** hook like the one in the following example:

  ```
  conster Header: React.FC = () => {
    const { t } = useTranslation('plugin__console-demo-plugin');
    return <h1>{t('Hello, World!')}</h1>;
  };
  ```

- Avoid selectors that could affect markup outside of your plugins components, such as element selectors. These are not APIs and are subject to change. Using them might break your plugin. Avoid selectors like element selectors that could affect markup outside of your plugins components.
- Provide valid JavaScript Multipurpose Internet Mail Extension (MIME) type using the **Content-Type** response header for all assets served by your plugin web server. Each plugin deployment should include a web server that hosts the generated assets of the given plugin.

PatternFly guidelines

When creating your plugin, follow these guidelines for using PatternFly:

- Use PatternFly components and PatternFly CSS variables. Core PatternFly components are available through the SDK. Using PatternFly components and variables help your plugin look consistent in future console versions.
  - Use Patternfly 4.x if you are using Red Hat OpenShift Service on AWS versions 4.14 and earlier.
  - Use Patternfly 5.x if you are using Red Hat OpenShift Service on AWS 4.15 or later.
- Make your plugin accessible by following PatternFly's accessibility fundamentals.
- Avoid using other CSS libraries such as Bootstrap or Tailwind. They can conflict with PatternFly and will not match the console look and feel. Plugins should only include styles that are specific to their user interfaces to be evaluated on top of base PatternFly styles. Avoid importing styles such as **@patternfly/react-styles/*/.css** or any styles from **@patternfly/patternfly** package in your plugin.
- The console application is responsible for loading base styles for all supported PatternFly version(s).

4.2. GETTING STARTED WITH DYNAMIC PLUGINS

To get started using the dynamic plugin, you must set up your environment to write a new Red Hat OpenShift Service on AWS dynamic plugin. For an example of how to write a new plugin, see Adding a tab to the pods page.

4.2.1. Dynamic plugin development

You can run the plugin using a local development environment. The Red Hat OpenShift Service on AWS web console runs in a container connected to the cluster you have logged into.

Prerequisites

- You must have an OpenShift cluster running.
- You must have the OpenShift CLI (**oc**) installed.
- You must have **yarn** installed.
- You must have Docker v3.2.0 or newer or Podman installed and running.

Procedure

1. In your terminal, run the following command to install the dependencies for your plugin using yarn.

```
$ yarn install
```

2. After installing, run the following command to start yarn.

```
$ yarn run start
```

3. In another terminal window, login to the Red Hat OpenShift Service on AWS through the CLI.

```
$ oc login
```

4. Run the Red Hat OpenShift Service on AWS web console in a container connected to the cluster you have logged into by running the following command:

```
$ yarn run start-console
```

Verification

- Visit localhost:9000 to view the running plugin. Inspect the value of **window.SERVER_FLAGS.consolePlugins** to see the list of plugins which load at runtime.

4.3. DEPLOY YOUR PLUGIN ON A CLUSTER

You can deploy the plugin to a Red Hat OpenShift Service on AWS cluster.

4.3.1. Build an image with Docker

To deploy your plugin on a cluster, you need to build an image and push it to an image registry.

Procedure

1. Build the image with the following command:

```
$ docker build -t quay.io/my-repositroy/my-plugin:latest .
```

2. Optional: If you want to test your image, run the following command:

```
$ docker run -it --rm -d -p 9001:80 quay.io/my-repository/my-plugin:latest
```

3. Push the image by running the following command:

```
$ docker push quay.io/my-repository/my-plugin:latest
```

4.3.2. Deploy your plugin on a cluster

After pushing an image with your changes to a registry, you can deploy the plugin to a cluster.

Procedure

1. To deploy your plugin to a cluster, install a Helm chart with the name of the plugin as the Helm release name into a new namespace or an existing namespace as specified by the **-n** command-line option. Provide the location of the image within the **plugin.image** parameter by using the following command:

```
$ helm upgrade -i  my-plugin charts/openshift-console-plugin -n my-plugin-namespace --
create-namespace --set plugin.image=my-plugin-image-location
```

Where:

n <my-plugin-namespace>

Specifies an existing namespace to deploy your plugin into.

--create-namespace

Optional: If deploying to a new namespace, use this parameter.

--set plugin.image=my-plugin-image-location

Specifies the location of the image within the **plugin.image** parameter.

2. Optional: You can specify any additional parameters by using the set of supported parameters in the **charts/openshift-console-plugin/values.yaml** file.

```
plugin:
  name: ""
  description: ""
  image: ""
  imagePullPolicy: IfNotPresent
  replicas: 2
  port: 9443
  securityContext:
    enabled: true
  podSecurityContext:
    enabled: true
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containerSecurityContext:
    enabled: true
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
  resources:
    requests:
      cpu: 10m
      memory: 50Mi
  basePath: /
  certificateSecretName: ""
  serviceAccount:
    create: true
    annotations: {}
    name: ""
  patcherServiceAccount:
    create: true
    annotations: {}
    name: ""
  jobs:
    patchConsoles:
      enabled: true
      image: "registry.redhat.io/openshift4/ose-tools-
rhel8@sha256:e44074f21e0cca6464e50cb6ff934747e0bd11162ea01d522433a1a1ae116103"

      podSecurityContext:
        enabled: true
        runAsNonRoot: true
        seccompProfile:
          type: RuntimeDefault
      containerSecurityContext:
        enabled: true
        allowPrivilegeEscalation: false
        capabilities:
          drop:
            - ALL
      resources:
        requests:
          cpu: 10m
          memory: 50Mi
```

Verification

- View the list of enabled plugins by navigating from **Administration** → **Cluster Settings** → **Configuration** → **Console operator.openshift.io** → **Console plugins** or by visiting the **Overview** page.

NOTE

It can take a few minutes for the new plugin configuration to appear. If you do not see your plugin, you might need to refresh your browser if the plugin was recently enabled. If you receive any errors at runtime, check the JS console in browser developer tools to look for any errors in your plugin code.

4.3.3. Disabling your plugin in the browser

Console users can use the **disable-plugins** query parameter to disable specific or all dynamic plugins that would normally get loaded at run-time.

Procedure

- To disable a specific plugin(s), remove the plugin you want to disable from the comma-separated list of plugin names.
- To disable all plugins, leave an empty string in the **disable-plugins** query parameter.

NOTE

Cluster administrators can disable plugins in the **Cluster Settings** page of the web console

4.4. DYNAMIC PLUGIN EXAMPLE

Before working through the example, verify that the plugin is working by following the steps in Dynamic plugin development

4.4.1. Adding a tab to the pods page

There are different customizations you can make to the Red Hat OpenShift Service on AWS web console. The following procedure adds a tab to the **Pod details** page as an example extension to your plugin.

NOTE

The Red Hat OpenShift Service on AWS web console runs in a container connected to the cluster you have logged into. See "Dynamic plugin development" for information to test the plugin before creating your own.

Procedure

1. Visit the **console-plugin-template** repository containing a template for creating plugins in a new tab.

   IMPORTANT

   Custom plugin code is not supported by Red Hat. Only Cooperative community support is available for your plugin.

2. Create a GitHub repository for the template by clicking **Use this template** → ***Create new repository***.

3. Rename the new repository with the name of your plugin.

4. Clone the new repository to your local machine so you can edit the code.

5. Edit the **package.json** file, adding your plugin's metadata to the **consolePlugin** declaration. For example:

```
"consolePlugin": {
  "name": "my-plugin", (1)
  "version": "0.0.1", (2)
  "displayName": "My Plugin", (3)
  "description": "Enjoy this shiny, new console plugin!", (4)
  "exposedModules": {
    "ExamplePage": "./components/ExamplePage"
  },
  "dependencies": {
    "@console/pluginAPI": "/*"
  }
}
```

(1) Update the name of your plugin.

(2) Update the version.

(3) Update the display name for your plugin.

(4) Update the description with a synopsis about your plugin.

6. Add the following to the **console-extensions.json** file:

```
{
  "type": "console.tab/horizontalNav",
  "properties": {
    "page": {
      "name": "Example Tab",
      "href": "example"
    },
    "model": {
      "group": "core",
      "version": "v1",
      "kind": "Pod"
    },
    "component": { "$codeRef": "ExampleTab" }
  }
}
```

7. Edit the **package.json** file to include the following changes:

```
        "exposedModules": {
            "ExamplePage": "./components/ExamplePage",
            "ExampleTab": "./components/ExampleTab"
        }
```

8. Write a message to display on a new custom tab on the **Pods** page by creating a new file **src/components/ExampleTab.tsx** and adding the following script:

```
import * as React from 'react';

export default function ExampleTab() {
    return (
        <p>This is a custom tab added to a resource using a dynamic plugin.</p>
    );
}
```

9. Install a Helm chart with the name of the plugin as the Helm release name into a new namespace or an existing namespace as specified by the **-n** command-line option to deploy your plugin on a cluster. Provide the location of the image within the **plugin.image** parameter by using the following command:

```
$ helm upgrade -i  my-plugin charts/openshift-console-plugin -n my-plugin-namespace --
create-namespace --set plugin.image=my-plugin-image-location
```

NOTE

For more information on deploying your plugin on a cluster, see "Deploy your plugin on a cluster".

Verification

- Visit a **Pod** page to view the added tab.

4.5. DYNAMIC PLUGIN REFERENCE

You can add extensions that allow you to customize your plugin. Those extensions are then loaded to the console at run-time.

4.5.1. Dynamic plugin extension types

console.action/filter
ActionFilter can be used to filter an action.

Name	Value Type	Optional	Description
contextId	**string**	no	The context ID helps to narrow the scope of contributed actions to a particular area of the application. Examples include **topology** and **helm**.
filter	**CodeRef<(scope: any, action: Action) ⇒ boolean>**	no	A function that will filter actions based on some conditions. **scope**: The scope in which actions should be provided for. A hook might be required if you want to remove the **ModifyCount** action from a deployment with a horizontal pod autoscaler (HPA).

console.action/group
ActionGroup contributes an action group that can also be a submenu.

Name	Value Type	Optional	Description
id	**string**	no	ID used to identify the action section.
label	**string**	yes	The label to display in the UI. Required for submenus.
submenu	**boolean**	yes	Whether this group should be displayed as submenu.
insertBefore	**string | string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.

Name	Value Type	Optional	Description
insertAfter	**string** | **string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. The **insertBefore** value takes precedence.

console.action/provider
ActionProvider contributes a hook that returns list of actions for specific context.

Name	Value Type	Optional	Description
contextId	**string**	no	The context ID helps to narrow the scope of contributed actions to a particular area of the application. Examples include **topology** and **helm**.
provider	**CodeRef<ExtensionHook<Action[], any>>**	no	A React hook that returns actions for the given scope. If **contextId** = **resource**, then the scope will always be a Kubernetes resource object.

console.action/resource-provider
ResourceActionProvider contributes a hook that returns list of actions for specific resource model.

Name	Value Type	Optional	Description
model	**ExtensionK8sKindVersionModel**	no	The model for which this provider provides actions for.
provider	**CodeRef<ExtensionHook<Action[], any>>**	no	A react hook which returns actions for the given resource model

console.alert-action
This extension can be used to trigger a specific action when a specific Prometheus alert is observed by the Console based on its **rule.name** value.

Name	Value Type	Optional	Description
alert	**string**	no	Alert name as defined by **alert.rule.name** property
text	**string**	no	
action	**CodeRef<(alert: any) ⇒ void>**	no	Function to perform side effect

console.catalog/item-filter
This extension can be used for plugins to contribute a handler that can filter specific catalog items. For example, the plugin can contribute a filter that filters helm charts from specific provider.

Name	Value Type	Optional	Description
catalogId	**string** | **string[]**	no	The unique identifier for the catalog this provider contributes to.
type	**string**	no	Type ID for the catalog item type.
filter	**CodeRef<(item: CatalogItem) ⇒ boolean>**	no	Filters items of a specific type. Value is a function that takes **CatalogItem[]** and returns a subset based on the filter criteria.

console.catalog/item-metadata
This extension can be used to contribute a provider that adds extra metadata to specific catalog items.

Name	Value Type	Optional	Description
catalogId	**string** | **string[]**	no	The unique identifier for the catalog this provider contributes to.
type	**string**	no	Type ID for the catalog item type.
provider	**CodeRef<ExtensionHook<CatalogItemMetadataProviderFunction, CatalogExtensionHookOptions>>**	no	A hook which returns a function that will be used to provide metadata to catalog items of a specific type.

Name	Value Type	Optional	Description

console.catalog/item-provider
This extension allows plugins to contribute a provider for a catalog item type. For example, a Helm Plugin can add a provider that fetches all the Helm Charts. This extension can also be used by other plugins to add more items to a specific catalog item type.

Name	Value Type	Optional	Description
catalogId	**string** | **string[]**	no	The unique identifier for the catalog this provider contributes to.
type	**string**	no	Type ID for the catalog item type.
title	**string**	no	Title for the catalog item provider
provider	**CodeRef<ExtensionHook<CatalogItem<any>[], CatalogExtensionHookOptions>>**	no	Fetch items and normalize it for the catalog. Value is a react effect hook.
priority	**number**	yes	Priority for this provider. Defaults to **0**. Higher priority providers may override catalog items provided by other providers.

console.catalog/item-type
This extension allows plugins to contribute a new type of catalog item. For example, a Helm plugin can define a new catalog item type as HelmCharts that it wants to contribute to the Developer Catalog.

Name	Value Type	Optional	Description
type	**string**	no	Type for the catalog item.

Name	Value Type	Optional	Description
title	**string**	no	Title for the catalog item.
catalogDescription	**string | CodeRef<React.ReactNode>**	yes	Description for the type specific catalog.
typeDescription	**string**	yes	Description for the catalog item type.
filters	**CatalogItemAttribute[]**	yes	Custom filters specific to the catalog item.
groupings	**CatalogItemAttribute[]**	yes	Custom groupings specific to the catalog item.

console.catalog/item-type-metadata
This extension allows plugins to contribute extra metadata like custom filters or groupings for any catalog item type. For example, a plugin can attach a custom filter for HelmCharts that can filter based on chart provider.

Name	Value Type	Optional	Description
type	**string**	no	Type for the catalog item.
filters	**CatalogItemAttribute[]**	yes	Custom filters specific to the catalog item.
groupings	**CatalogItemAttribute[]**	yes	Custom groupings specific to the catalog item.

console.cluster-overview/inventory-item
Adds a new inventory item into cluster overview page.

Name	Value Type	Optional	Description
component	**CodeRef<React.ComponentType<{}>>**	no	The component to be rendered.

console.cluster-overview/multiline-utilization-item
Adds a new cluster overview multi-line utilization item.

Name	Value Type	Optional	Description
title	**string**	no	The title of the utilization item.
getUtilizationQueries	**CodeRef<GetMultilineQueries>**	no	Prometheus utilization query.
humanize	**CodeRef<Humanize>**	no	Convert Prometheus data to human-readable form.
TopConsumerPopovers	**CodeRef<React.ComponentType<TopConsumerPopoverProps>[]>**	yes	Shows Top consumer popover instead of plain value.

console.cluster-overview/utilization-item
Adds a new cluster overview utilization item.

Name	Value Type	Optional	Description
title	**string**	no	The title of the utilization item.
getUtilizationQuery	**CodeRef<GetQuery>**	no	Prometheus utilization query.
humanize	**CodeRef<Humanize>**	no	Convert Prometheus data to human-readable form.
getTotalQuery	**CodeRef<GetQuery>**	yes	Prometheus total query.
getRequestQuery	**CodeRef<GetQuery>**	yes	Prometheus request query.
getLimitQuery	**CodeRef<GetQuery>**	yes	Prometheus limit query.
TopConsumerPopover	**CodeRef<React.ComponentType<TopConsumerPopoverProps>>**	yes	Shows Top consumer popover instead of plain value.

console.context-provider
Adds a new React context provider to the web console application root.

Name	Value Type	Optional	Description
provider	**CodeRef<Provider<T>>**	no	Context Provider component.
useValueHook	**CodeRef<() ⇒ T>**	no	Hook for the Context value.

console.dashboards/card
Adds a new dashboard card.

Name	Value Type	Optional	Description
tab	**string**	no	The ID of the dashboard tab to which the card will be added.
position	**'LEFT' | 'RIGHT' | 'MAIN'**	no	The grid position of the card on the dashboard.
component	**CodeRef<React.ComponentType<{}>>**	no	Dashboard card component.
span	**OverviewCardSpan**	yes	Card’s vertical span in the column. Ignored for small screens; defaults to **12**.

console.dashboards/custom/overview/detail/item
Adds an item to the Details card of Overview Dashboard.

Name	Value Type	Optional	Description
title	**string**	no	Details card title
component	**CodeRef<React.ComponentType<{}>>**	no	The value, rendered by the OverviewDetailItem component
valueClassName	**string**	yes	Value for a className
isLoading	**CodeRef<() ⇒ boolean>**	yes	Function returning the loading state of the component
error	**CodeRef<() ⇒ string>**	yes	Function returning errors to be displayed by the component

console.dashboards/overview/activity/resource
Adds an activity to the Activity Card of Overview Dashboard where the triggering of activity is based on watching a Kubernetes resource.

Name	Value Type	Optional	Description
k8sResource	**CodeRef<FirehoseResource & { isList: true; }>**	no	The utilization item to be replaced.
component	**CodeRef<React.ComponentType<K8sActivityProps<T>>>**	no	The action component.
isActivity	**CodeRef<(resource: T) ⇒ boolean>**	yes	Function which determines if the given resource represents the action. If not defined, every resource represents activity.
getTimestamp	**CodeRef<(resource: T) ⇒ Date>**	yes	Time stamp for the given action, which will be used for ordering.

console.dashboards/overview/health/operator
Adds a health subsystem to the status card of the **Overview** dashboard, where the source of status is a Kubernetes REST API.

Name	Value Type	Optional	Description
title	**string**	no	Title of Operators section in the pop-up menu.
resources	**CodeRef<FirehoseResource[]>**	no	Kubernetes resources which will be fetched and passed to **healthHandler**.
getOperatorsWithStatuses	**CodeRef<GetOperatorsWithStatuses<T>>**	yes	Resolves status for the Operators.
operatorRowLoader	**CodeRef<React.ComponentType<OperatorRowProps<T>>>**	yes	Loader for pop-up row component.

Name	Value Type	Optional	Description
viewAllLink	**string**	yes	Links to all resources page. If not provided, then a list page of the first resource from resources prop is used.

console.dashboards/overview/health/prometheus
Adds a health subsystem to the status card of Overview dashboard where the source of status is Prometheus.

Name	Value Type	Optional	Description
title	**string**	no	The display name of the subsystem.
queries	**string[]**	no	The Prometheus queries.
healthHandler	**CodeRef<PrometheusHealthHandler>**	no	Resolve the subsystem's health.
additionalResource	**CodeRef<FirehoseResource>**	yes	Additional resource which will be fetched and passed to **healthHandler**.
popupComponent	**CodeRef<React.ComponentType<PrometheusHealthPopupProps>>**	yes	Loader for pop-up menu content. If defined, a health item is represented as a link, which opens a pop-up menu with the given content.
popupTitle	**string**	yes	The title of the popover.
disallowedControlPlaneTopology	**string[]**	yes	Control plane topology for which the subsystem should be hidden.

console.dashboards/overview/health/resource
Adds a health subsystem to the status card of Overview dashboard where the source of status is a Kubernetes Resource.

Name	Value Type	Optional	Description
title	**string**	no	The display name of the subsystem.
resources	**CodeRef<WatchK8sResources<T>>**	no	Kubernetes resources that will be fetched and passed to **healthHandler**.
healthHandler	**CodeRef<ResourceHealthHandler<T>>**	no	Resolve the subsystem's health.
popupComponent	**CodeRef<WatchK8sResults<T>>**	yes	Loader for pop-up menu content. If defined, a health item is represented as a link, which opens a pop-up menu with the given content.
popupTitle	**string**	yes	The title of the popover.

console.dashboards/overview/health/url
Adds a health subsystem to the status card of Overview dashboard where the source of status is a Kubernetes REST API.

Name	Value Type	Optional	Description
title	**string**	no	The display name of the subsystem.
url	**string**	no	The URL to fetch data from. It will be prefixed with base Kubernetes URL.
healthHandler	**CodeRef<URLHealthHandler<T, K8sResourceCommon | K8sResourceCommon[]>>**	no	Resolve the subsystem's health.
additionalResource	**CodeRef<FirehoseResource>**	yes	Additional resource which will be fetched and passed to **healthHandler**.

Name	Value Type	Optional	Description
popupComponent	**CodeRef<React.ComponentType<{ healthResult?: T; healthResultError?: any; k8sResult?: FirehoseResult<R>; }>>**	yes	Loader for popup content. If defined, a health item will be represented as a link which opens popup with given content.
popupTitle	**string**	yes	The title of the popover.

console.dashboards/overview/inventory/item
Adds a resource tile to the overview inventory card.

Name	Value Type	Optional	Description
model	**CodeRef<T>**	no	The model for **resource** which will be fetched. Used to get the model's **label** or **abbr**.
mapper	**CodeRef<StatusGroupMapper<T, R>>**	yes	Function which maps various statuses to groups.
additionalResources	**CodeRef<WatchK8sResources<R>>**	yes	Additional resources which will be fetched and passed to the **mapper** function.

console.dashboards/overview/inventory/item/group
Adds an inventory status group.

Name	Value Type	Optional	Description
id	**string**	no	The ID of the status group.
icon	**CodeRef<React.ReactElement<any, string | React.JSXElementConstructor<any>>>**	no	React component representing the status group icon.

console.dashboards/overview/inventory/item/replacement
Replaces an overview inventory card.

Name	Value Type	Optional	Description
model	**CodeRef<T>**	no	The model for **resource** which will be fetched. Used to get the model's **label** or **abbr**.
mapper	**CodeRef<StatusGroupMapper<T, R>>**	yes	Function which maps various statuses to groups.
additionalResources	**CodeRef<WatchK8sResources<R>>**	yes	Additional resources which will be fetched and passed to the **mapper** function.

console.dashboards/overview/prometheus/activity/resource
Adds an activity to the Activity Card of Prometheus Overview Dashboard where the triggering of activity is based on watching a Kubernetes resource.

Name	Value Type	Optional	Description
queries	**string[]**	no	Queries to watch.
component	**CodeRef<React.ComponentType<PrometheusActivityProps>>**	no	The action component.
isActivity	**CodeRef<(results: PrometheusResponse[]) ⇒ boolean>**	yes	Function which determines if the given resource represents the action. If not defined, every resource represents activity.

console.dashboards/project/overview/item
Adds a resource tile to the project overview inventory card.

Name	Value Type	Optional	Description
model	**CodeRef<T>**	no	The model for **resource** which will be fetched. Used to get the model's **label** or **abbr**.
mapper	**CodeRef<StatusGroupMapper<T, R>>**	yes	Function which maps various statuses to groups.

Name	Value Type	Optional	Description
additionalResources	**CodeRef<WatchK8s Resources<R>>**	yes	Additional resources which will be fetched and passed to the **mapper** function.

console.dashboards/tab
Adds a new dashboard tab, placed after the **Overview** tab.

Name	Value Type	Optional	Description
id	**string**	no	A unique tab identifier, used as tab link **href** and when adding cards to this tab.
navSection	**'home' | 'storage'**	no	Navigation section to which the tab belongs to.
title	**string**	no	The title of the tab.

console.file-upload
This extension can be used to provide a handler for the file drop action on specific file extensions.

Name	Value Type	Optional	Description
fileExtensions	**string[]**	no	Supported file extensions.
handler	**CodeRef<FileUpload Handler>**	no	Function which handles the file drop action.

console.flag
Gives full control over the web console feature flags.

Name	Value Type	Optional	Description
handler	**CodeRef<FeatureFla gHandler>**	no	Used to set or unset arbitrary feature flags.

console.flag/hookProvider
Gives full control over the web console feature flags with hook handlers.

Name	Value Type	Optional	Description
handler	**CodeRef<FeatureFlagHandler>**	no	Used to set or unset arbitrary feature flags.

console.flag/model
Adds a new web console feature flag driven by the presence of a **CustomResourceDefinition** (CRD) object on the cluster.

Name	Value Type	Optional	Description
flag	**string**	no	The name of the flag to set after the CRD is detected.
model	**ExtensionK8sModel**	no	The model which refers to a CRD.

console.global-config
This extension identifies a resource used to manage the configuration of the cluster. A link to the resource will be added to the **Administration → Cluster Settings → Configuration** page.

Name	Value Type	Optional	Description
id	**string**	no	Unique identifier for the cluster config resource instance.
name	**string**	no	The name of the cluster config resource instance.
model	**ExtensionK8sModel**	no	The model which refers to a cluster config resource.
namespace	**string**	no	The namespace of the cluster config resource instance.

console.model-metadata
Customize the display of models by overriding values retrieved and generated through API discovery.

Name	Value Type	Optional	Description

Name	Value Type	Optional	Description
model	**ExtensionK8sGroupModel**	no	The model to customize. May specify only a group, or optional version and kind.
badge	**ModelBadge**	yes	Whether to consider this model reference as Technology Preview or Developer Preview.
color	**string**	yes	The color to associate to this model.
label	**string**	yes	Override the label. Requires **kind** be provided.
labelPlural	**string**	yes	Override the plural label. Requires **kind** be provided.
abbr	**string**	yes	Customize the abbreviation. Defaults to all uppercase characters in **kind**, up to 4 characters long. Requires that **kind** is provided.

console.navigation/href
This extension can be used to contribute a navigation item that points to a specific link in the UI.

Name	Value Type	Optional	Description
id	**string**	no	A unique identifier for this item.
name	**string**	no	The name of this item.
href	**string**	no	The link **href** value.
perspective	**string**	yes	The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.

Name	Value Type	Optional	Description
section	**string**	yes	Navigation section to which this item belongs to. If not specified, render this item as a top level link.
dataAttributes	**{ [key: string]: string; }**	yes	Adds data attributes to the DOM.
startsWith	**string[]**	yes	Mark this item as active when the URL starts with one of these paths.
insertBefore	**string | string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.
insertAfter	**string | string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. **insertBefore** takes precedence.
namespaced	**boolean**	yes	If **true**, adds **/ns/active-namespace** to the end.
prefixNamespaced	**boolean**	yes	If **true**, adds **/k8s/ns/active-namespace** to the beginning.

console.navigation/resource-cluster
This extension can be used to contribute a navigation item that points to a cluster resource details page. The K8s model of that resource can be used to define the navigation item.

Name	Value Type	Optional	Description
id	**string**	no	A unique identifier for this item.
model	**ExtensionK8sModel**	no	The model for which this navigation item links to.

Name	Value Type	Optional	Description
perspective	**string**	yes	The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.
section	**string**	yes	Navigation section to which this item belongs to. If not specified, render this item as a top-level link.
dataAttributes	**{ [key: string]: string; }**	yes	Adds data attributes to the DOM.
startsWith	**string[]**	yes	Mark this item as active when the URL starts with one of these paths.
insertBefore	**string | string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.
insertAfter	**string | string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. **insertBefore** takes precedence.
name	**string**	yes	Overrides the default name. If not supplied the name of the link will equal the plural value of the model.

console.navigation/resource-ns
This extension can be used to contribute a navigation item that points to a namespaced resource details page. The K8s model of that resource can be used to define the navigation item.

Name	Value Type	Optional	Description
id	**string**	no	A unique identifier for this item.

Name	Value Type	Optional	Description
model	**ExtensionK8sModel**	no	The model for which this navigation item links to.
perspective	**string**	yes	The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.
section	**string**	yes	Navigation section to which this item belongs to. If not specified, render this item as a top-level link.
dataAttributes	**{ [key: string]: string; }**	yes	Adds data attributes to the DOM.
startsWith	**string[]**	yes	Mark this item as active when the URL starts with one of these paths.
insertBefore	**string | string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.
insertAfter	**string | string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. **insertBefore** takes precedence.
name	**string**	yes	Overrides the default name. If not supplied the name of the link will equal the plural value of the model.

console.navigation/section
This extension can be used to define a new section of navigation items in the navigation tab.

Name	Value Type	Optional	Description

Name	Value Type	Optional	Description
id	**string**	no	A unique identifier for this item.
perspective	**string**	yes	The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.
dataAttributes	**{ [key: string]: string; }**	yes	Adds data attributes to the DOM.
insertBefore	**string | string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.
insertAfter	**string | string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. **insertBefore** takes precedence.
name	**string**	yes	Name of this section. If not supplied, only a separator will be shown above the section.

console.navigation/separator
This extension can be used to add a separator between navigation items in the navigation.

Name	Value Type	Optional	Description
id	**string**	no	A unique identifier for this item.
perspective	**string**	yes	The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.

Name	Value Type	Optional	Description
section	**string**	yes	Navigation section to which this item belongs to. If not specified, render this item as a top level link.
dataAttributes	**{ [key: string]: string; }**	yes	Adds data attributes to the DOM.
insertBefore	**string** | **string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.
insertAfter	**string** | **string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. **insertBefore** takes precedence.

console.page/resource/details

Name	Value Type	Optional	Description
model	**ExtensionK8sGroupKindModel**	no	The model for which this resource page links to.
component	**CodeRef<React.ComponentType<{ match: match<{}>; namespace: string; model: ExtensionK8sModel; }>>**	no	The component to be rendered when the route matches.

console.page/resource/list
Adds new resource list page to Console router.

Name	Value Type	Optional	Description
model	**ExtensionK8sGroupKindModel**	no	The model for which this resource page links to.

Name	Value Type	Optional	Description
component	**CodeRef<React.ComponentType<{ match: match<{}>; namespace: string; model: ExtensionK8sModel; }>>**	no	The component to be rendered when the route matches.

console.page/route
Adds a new page to the web console router. See React Router.

Name	Value Type	Optional	Description
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	no	The component to be rendered when the route matches.
path	**string | string[]**	no	Valid URL path or array of paths that **path-to-regexp@^1.7.0** understands.
perspective	**string**	yes	The perspective to which this page belongs to. If not specified, contributes to all perspectives.
exact	**boolean**	yes	When true, will only match if the path matches the **location.pathname** exactly.

console.page/route/standalone
Adds a new standalone page, rendered outside the common page layout, to the web console router. See React Router.

Name	Value Type	Optional	Description
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	no	The component to be rendered when the route matches.

Name	Value Type	Optional	Description
path	**string | string[]**	no	Valid URL path or array of paths that **path-to-regexp@^1.7.0** understands.
exact	**boolean**	yes	When true, will only match if the path matches the **location.pathname** exactly.

console.perspective
This extension contributes a new perspective to the console, which enables customization of the navigation menu.

Name	Value Type	Optional	Description
id	**string**	no	The perspective identifier.
name	**string**	no	The perspective display name.
icon	**CodeRef<LazyComponent>**	no	The perspective display icon.
landingPageURL	**CodeRef<(flags: { [key: string]: boolean; }, isFirstVisit: boolean) ⇒ string>**	no	The function to get perspective landing page URL.
importRedirectURL	**CodeRef<(namespace: string) ⇒ string>**	no	The function to get redirect URL for import flow.
default	**boolean**	yes	Whether the perspective is the default. There can only be one default.
defaultPins	**ExtensionK8sModel[]**	yes	Default pinned resources on the nav
usePerspectiveDetection	**CodeRef<() ⇒ [boolean, boolean]>**	yes	The hook to detect default perspective

console.project-overview/inventory-item

Adds a new inventory item into the **Project Overview** page.

Name	Value Type	Optional	Description
component	**CodeRef<React.ComponentType<{ projectName: string; }>>**	no	The component to be rendered.

console.project-overview/utilization-item
Adds a new project overview utilization item.

Name	Value Type	Optional	Description
title	**string**	no	The title of the utilization item.
getUtilizationQuery	**CodeRef<GetProjectQuery>**	no	Prometheus utilization query.
humanize	**CodeRef<Humanize>**	no	Convert Prometheus data to human-readable form.
getTotalQuery	**CodeRef<GetProjectQuery>**	yes	Prometheus total query.
getRequestQuery	**CodeRef<GetProjectQuery>**	yes	Prometheus request query.
getLimitQuery	**CodeRef<GetProjectQuery>**	yes	Prometheus limit query.
TopConsumerPopover	**CodeRef<React.ComponentType<TopConsumerPopoverProps>>**	yes	Shows the top consumer popover instead of plain value.

console.pvc/alert
This extension can be used to contribute custom alerts on the PVC details page.

Name	Value Type	Optional	Description
alert	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	no	The alert component.

console.pvc/create-prop
This extension can be used to specify additional properties that will be used when creating PVC resources on the PVC list page.

Name	Value Type	Optional	Description
label	**string**	no	Label for the create prop action.
path	**string**	no	Path for the create prop action.

console.pvc/delete
This extension allows hooking into deleting PVC resources. It can provide an alert with additional information and custom PVC delete logic.

Name	Value Type	Optional	Description
predicate	**CodeRef<(pvc: K8sResourceCommon) ⇒ boolean>**	no	Predicate that tells whether to use the extension or not.
onPVCKill	**CodeRef<(pvc: K8sResourceCommon) ⇒ Promise<void>>**	no	Method for the PVC delete operation.
alert	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	no	Alert component to show additional information.

console.pvc/status

Name	Value Type	Optional	Description
priority	**number**	no	Priority for the status component. A larger value means higher priority.
status	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	no	The status component.
predicate	**CodeRef<(pvc: K8sResourceCommon) ⇒ boolean>**	no	Predicate that tells whether to render the status component or not.

console.redux-reducer
Adds new reducer to Console Redux store which operates on **plugins.<scope>** substate.

Name	Value Type	Optional	Description
scope	**string**	no	The key to represent the reducer-managed substate within the Redux state object.
reducer	**CodeRef<Reducer<any, AnyAction>>**	no	The reducer function, operating on the reducer-managed substate.

console.resource/create
This extension allows plugins to provide a custom component (i.e., wizard or form) for specific resources, which will be rendered, when users try to create a new resource instance.

Name	Value Type	Optional	Description
model	**ExtensionK8sModel**	no	The model for which this create resource page will be rendered
component	**CodeRef<React.ComponentType<CreateResourceComponentProps>>**	no	The component to be rendered when the model matches

console.resource/details-item
Adds a new details item to the default resource summary on the details page.

Name	Value Type	Optional	Description
model	**ExtensionK8sModel**	no	The subject resource’s API group, version, and kind.
id	**string**	no	A unique identifier.
column	**DetailsItemColumn**	no	Determines if the item will appear in the 'left' or 'right' column of the resource summary on the details page. Default: 'right'
title	**string**	no	The details item title.

Name	Value Type	Optional	Description
path	**string**	yes	An optional, fully-qualified path to a resource property to used as the details item value. Only primitive type values can be rendered directly. Use the component property to handle other data types.
component	**CodeRef<React.ComponentType<DetailsItemComponentProps<K8sResourceCommon, any>>>**	yes	An optional React component that will render the details item value.
sortWeight	**number**	yes	An optional sort weight, relative to all other details items in the same column. Represented by any valid JavaScriptNumber. Items in each column are sorted independently, lowest to highest. Items without sort weightsare sorted after items with sort weights.

console.storage-class/provisioner
Adds a new storage class provisioner as an option during storage class creation.

Name	Value Type	Optional	Description
CSI	**ProvisionerDetails**	yes	Container Storage Interface provisioner type
OTHERS	**ProvisionerDetails**	yes	Other provisioner type

console.storage-provider
This extension can be used to contribute a new storage provider to select, when attaching storage and a provider specific component.

Name	Value Type	Optional	Description
name	**string**	no	Displayed name of the provider.

Name	Value Type	Optional	Description
Component	**CodeRef<React.ComponentType<Partial<RouteComponentProps<{}, StaticContext, any>>>>**	no	Provider specific component to render.

console.tab
Adds a tab to a horizontal nav matching the **contextId**.

Name	Value Type	Optional	Description
contextId	**string**	no	Context ID assigned to the horizontal nav in which the tab will be injected. Possible values: **dev-console-observe**
name	**string**	no	The display label of the tab
href	**string**	no	The **href** appended to the existing URL
component	**CodeRef<React.ComponentType<PageComponentProps<K8sResourceCommon>>>**	no	Tab content component.

console.tab/horizontalNav
This extension can be used to add a tab on the resource details page.

Name	Value Type	Optional	Description
model	**ExtensionK8sKindVersionModel**	no	The model for which this provider show tab.
page	**{ name: string; href: string; }**	no	The page to be show in horizontal tab. It takes tab name as name and href of the tab

Name	Value Type	Optional	Description
component	**CodeRef<React.ComponentType<PageComponentProps<K8sResourceCommon>>>**	no	The component to be rendered when the route matches.

console.telemetry/listener
This component can be used to register a listener function receiving telemetry events. These events include user identification, page navigation, and other application specific events. The listener may use this data for reporting and analytics purposes.

Name	Value Type	Optional	Description
listener	**CodeRef<TelemetryEventListener>**	no	Listen for telemetry events

console.topology/adapter/build
BuildAdapter contributes an adapter to adapt element to data that can be used by the Build component.

Name	Value Type	Optional	Description
adapt	**CodeRef<(element: GraphElement) ⇒ AdapterDataType<BuildConfigData> | undefined>**	no	Adapter to adapt element to data that can be used by Build component.

console.topology/adapter/network
NetworkAdapater contributes an adapter to adapt element to data that can be used by the **Networking** component.

Name	Value Type	Optional	Description
adapt	**CodeRef<(element: GraphElement) ⇒ NetworkAdapterType | undefined>**	no	Adapter to adapt element to data that can be used by Networking component.

console.topology/adapter/pod
PodAdapter contributes an adapter to adapt element to data that can be used by the **Pod** component.

Name	Value Type	Optional	Description

Name	Value Type	Optional	Description
adapt	**CodeRef<(element: GraphElement) ⇒ AdapterDataType<PodsAdapterDataType> | undefined>**	no	Adapter to adapt element to data that can be used by Pod component.

console.topology/component/factory
Getter for a **ViewComponentFactory**.

Name	Value Type	Optional	Description
getFactory	**CodeRef<ViewComponentFactory>**	no	Getter for a **ViewComponentFactory**.

console.topology/create/connector
Getter for the create connector function.

Name	Value Type	Optional	Description
getCreateConnector	**CodeRef<CreateConnectionGetter>**	no	Getter for the create connector function.

console.topology/data/factory
Topology Data Model Factory Extension

Name	Value Type	Optional	Description
id	**string**	no	Unique ID for the factory.
priority	**number**	no	Priority for the factory
resources	**WatchK8sResourcesGeneric**	yes	Resources to be fetched from **useK8sWatchResources** hook.
workloadKeys	**string[]**	yes	Keys in resources containing workloads.
getDataModel	**CodeRef<TopologyDataModelGetter>**	yes	Getter for the data model factory.

Name	Value Type	Optional	Description
isResourceDepicted	**CodeRef<TopologyDataModelDepicted>**	yes	Getter for function to determine if a resource is depicted by this model factory.
getDataModelReconciler	**CodeRef<TopologyDataModelReconciler>**	yes	Getter for function to reconcile data model after all extensions' models have loaded.

console.topology/decorator/provider
Topology Decorator Provider Extension

Name	Value Type	Optional	Description
id	**string**	no	ID for topology decorator specific to the extension
priority	**number**	no	Priority for topology decorator specific to the extension
quadrant	**TopologyQuadrant**	no	Quadrant for topology decorator specific to the extension
decorator	**CodeRef<TopologyDecoratorGetter>**	no	Decorator specific to the extension

console.topology/details/resource-alert
DetailsResourceAlert contributes an alert for specific topology context or graph element.

Name	Value Type	Optional	Description
id	**string**	no	The ID of this alert. Used to save state if the alert should not be shown after dismissed.
contentProvider	**CodeRef<(element: GraphElement) ⇒ DetailsResourceAlertContent | null>**	no	Hook to return the contents of the alert.

console.topology/details/resource-link

DetailsResourceLink contributes a link for specific topology context or graph element.

Name	Value Type	Optional	Description
link	**CodeRef<(element: GraphElement) ⇒ React.Component | undefined>**	no	Return the resource link if provided, otherwise undefined. Use the **ResourceIcon** and **ResourceLink** properties for styles.
priority	**number**	yes	A higher priority factory will get the first chance to create the link.

console.topology/details/tab
DetailsTab contributes a tab for the topology details panel.

Name	Value Type	Optional	Description
id	**string**	no	A unique identifier for this details tab.
label	**string**	no	The tab label to display in the UI.
insertBefore	**string** | **string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.
insertAfter	**string** | **string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. The **insertBefore** value takes precedence.

console.topology/details/tab-section
DetailsTabSection contributes a section for a specific tab in the topology details panel.

Name	Value Type	Optional	Description
id	**string**	no	A unique identifier for this details tab section.

Name	Value Type	Optional	Description
tab	**string**	no	The parent tab ID that this section should contribute to.
provider	**CodeRef<DetailsTabSectionExtensionHook>**	no	A hook that returns a component, or if null or undefined, renders in the topology sidebar. SDK component: **<Section title=\{}>…** padded area
section	**CodeRef<(element: GraphElement, renderNull?: () ⇒ null) ⇒ React.Component | undefined>**	no	Deprecated: Fallback if no provider is defined. renderNull is a no-op already.
insertBefore	**string** | **string[]**	yes	Insert this item before the item referenced here. For arrays, the first one found in order is used.
insertAfter	**string** | **string[]**	yes	Insert this item after the item referenced here. For arrays, the first one found in order is used. The **insertBefore** value takes precedence.

console.topology/display/filters
Topology Display Filters Extension

Name	Value Type	Optional	Description
getTopologyFilters	**CodeRef<() ⇒ TopologyDisplayOption[]>**	no	Getter for topology filters specific to the extension
applyDisplayOptions	**CodeRef<TopologyApplyDisplayOptions>**	no	Function to apply filters to the model

console.topology/relationship/provider
Topology relationship provider connector extension

Name	Value Type	Optional	Description
provides	**CodeRef<RelationshipProviderProvides>**	no	Use to determine if a connection can be created between the source and target node
tooltip	**string**	no	Tooltip to show when connector operation is hovering over the drop target, for example, "Create a Visual Connector"
create	**CodeRef<RelationshipProviderCreate>**	no	Callback to execute when connector is drop over target node to create a connection
priority	**number**	no	Priority for relationship, higher will be preferred in case of multiple

console.user-preference/group
This extension can be used to add a group on the console user-preferences page. It will appear as a vertical tab option on the console user-preferences page.

Name	Value Type	Optional	Description
id	**string**	no	ID used to identify the user preference group.
label	**string**	no	The label of the user preference group
insertBefore	**string**	yes	ID of user preference group before which this group should be placed
insertAfter	**string**	yes	ID of user preference group after which this group should be placed

console.user-preference/item
This extension can be used to add an item to the user preferences group on the console user preferences page.

Name	Value Type	Optional	Description
id	**string**	no	ID used to identify the user preference item and referenced in insertAfter and insertBefore to define the item order
label	**string**	no	The label of the user preference
description	**string**	no	The description of the user preference
field	**UserPreferenceField**	no	The input field options used to render the values to set the user preference
groupId	**string**	yes	IDs used to identify the user preference groups the item would belong to
insertBefore	**string**	yes	ID of user preference item before which this item should be placed
insertAfter	**string**	yes	ID of user preference item after which this item should be placed

console.yaml-template
YAML templates for editing resources via the yaml editor.

Name	Value Type	Optional	Description
model	**ExtensionK8sModel**	no	Model associated with the template.
template	**CodeRef<string>**	no	The YAML template.
name	**string**	no	The name of the template. Use the name **default** to mark this as the default template.

dev-console.add/action

This extension allows plugins to contribute an add action item to the add page of developer perspective. For example, a Serverless plugin can add a new action item for adding serverless functions to the add page of developer console.

Name	Value Type	Optional	Description
id	**string**	no	ID used to identify the action.
label	**string**	no	The label of the action.
description	**string**	no	The description of the action.
href	**string**	no	The **href** to navigate to.
groupId	**string**	yes	IDs used to identify the action groups the action would belong to.
icon	**CodeRef<React.ReactNode>**	yes	The perspective display icon.
accessReview	**AccessReviewResourceAttributes[]**	yes	Optional access review to control the visibility or enablement of the action.

dev-console.add/action-group
This extension allows plugins to contibute a group in the add page of developer console. Groups can be referenced by actions, which will be grouped together in the add action page based on their extension definition. For example, a Serverless plugin can contribute a Serverless group and together with multiple add actions.

Name	Value Type	Optional	Description
id	**string**	no	ID used to identify the action group
name	**string**	no	The title of the action group
insertBefore	**string**	yes	ID of action group before which this group should be placed
insertAfter	**string**	yes	ID of action group after which this group should be placed

dev-console.import/environment
This extension can be used to specify extra build environment variable fields under the builder image selector in the developer console git import form. When set, the fields will override environment variables of the same name in the build section.

Name	Value Type	Optional	Description
imageStreamName	**string**	no	Name of the image stream to provide custom environment variables for
imageStreamTags	**string[]**	no	List of supported image stream tags
environments	**ImageEnvironment[]**	no	List of environment variables

console.dashboards/overview/detail/item
Deprecated. use **CustomOverviewDetailItem** type instead

Name	Value Type	Optional	Description
component	**CodeRef<React.ComponentType<{}>>**	no	The value, based on the **DetailItem** component

console.page/resource/tab
Deprecated. Use **console.tab/horizontalNav** instead. Adds a new resource tab page to Console router.

Name	Value Type	Optional	Description
model	**ExtensionK8sGroupKindModel**	no	The model for which this resource page links to.
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	no	The component to be rendered when the route matches.
name	**string**	no	The name of the tab.
href	**string**	yes	The optional **href** for the tab link. If not provided, the first **path** is used.

Name	Value Type	Optional	Description
exact	**boolean**	yes	When true, will only match if the path matches the **location.pathname** exactly.

4.5.2. Red Hat OpenShift Service on AWS console API

useActivePerspective
Hook that provides the currently active perspective and a callback for setting the active perspective. It returns a tuple containing the current active perspective and setter callback.

Example

```
const Component: React.FC = (props) => {
  const [activePerspective, setActivePerspective] = useActivePerspective();
  return <select
    value={activePerspective}
    onChange={(e) => setActivePerspective(e.target.value)}
  >
    {
      // ...perspective options
    }
  </select>
}
```

GreenCheckCircleIcon
Component for displaying a green check mark circle icon.

Example

```
<GreenCheckCircleIcon title="Healthy" />
```

Parameter Name	Description
className	(optional) additional class name for the component
title	(optional) icon title
size	(optional) icon size: (**sm**, **md**, **lg**, **xl**)

RedExclamationCircleIcon
Component for displaying a red exclamation mark circle icon.

Example

```
<RedExclamationCircleIcon title="Failed" />
```

Parameter Name	Description
className	(optional) additional class name for the component
title	(optional) icon title
size	(optional) icon size: (**sm**, **md**, **lg**, **xl**)

YellowExclamationTriangleIcon
Component for displaying a yellow triangle exclamation icon.

Example

```
<YellowExclamationTriangleIcon title="Warning" />
```

Parameter Name	Description
className	(optional) additional class name for the component
title	(optional) icon title
size	(optional) icon size: (**sm**, **md**, **lg**, **xl**)

BlueInfoCircleIcon
Component for displaying a blue info circle icon.

Example

```
<BlueInfoCircleIcon title="Info" />
```

Parameter Name	Description
className	(optional) additional class name for the component
title	(optional) icon title
size	(optional) icon size: ('sm', 'md', 'lg', 'xl')

ErrorStatus
Component for displaying an error status popover.

Example

```
<ErrorStatus title={errorMsg} />
```

Parameter Name	Description
title	(optional) status text
iconOnly	(optional) if true, only displays icon
noTooltip	(optional) if true, tooltip won't be displayed
className	(optional) additional class name for the component
popoverTitle	(optional) title for popover

InfoStatus
Component for displaying an information status popover.

Example

```
<InfoStatus title={infoMsg} />
```

Parameter Name	Description
title	(optional) status text
iconOnly	(optional) if true, only displays icon
noTooltip	(optional) if true, tooltip won't be displayed
className	(optional) additional class name for the component
popoverTitle	(optional) title for popover

ProgressStatus
Component for displaying a progressing status popover.

Example

```
<ProgressStatus title={progressMsg} />
```

Parameter Name	Description
title	(optional) status text
iconOnly	(optional) if true, only displays icon
noTooltip	(optional) if true, tooltip won't be displayed

Parameter Name	Description
className	(optional) additional class name for the component
popoverTitle	(optional) title for popover

SuccessStatus
Component for displaying a success status popover.

Example

```
<SuccessStatus title={successMsg} />
```

Parameter Name	Description
title	(optional) status text
iconOnly	(optional) if true, only displays icon
noTooltip	(optional) if true, tooltip won’t be displayed
className	(optional) additional class name for the component
popoverTitle	(optional) title for popover

checkAccess
Provides information about user access to a given resource. It returns an object with resource access information.

Parameter Name	Description
resourceAttributes	resource attributes for access review
impersonate	impersonation details

useAccessReview
Hook that provides information about user access to a given resource. It returns an array with **isAllowed** and **loading** values.

Parameter Name	Description
resourceAttributes	resource attributes for access review
impersonate	impersonation details

useResolvedExtensions
React hook for consuming Console extensions with resolved **CodeRef** properties. This hook accepts the same argument(s) as **useExtensions** hook and returns an adapted list of extension instances, resolving all code references within each extension's properties.

Initially, the hook returns an empty array. After the resolution is complete, the React component is re-rendered with the hook returning an adapted list of extensions. When the list of matching extensions changes, the resolution is restarted. The hook will continue to return the previous result until the resolution completes.

The hook's result elements are guaranteed to be referentially stable across re-renders. It returns a tuple containing a list of adapted extension instances with resolved code references, a boolean flag indicating whether the resolution is complete, and a list of errors detected during the resolution.

Example

```
const [navItemExtensions, navItemsResolved] = useResolvedExtensions<NavItem>(isNavItem);
// process adapted extensions and render your component
```

Parameter Name	Description
typeGuards	A list of callbacks that each accept a dynamic plugin extension as an argument and return a boolean flag indicating whether or not the extension meets desired type constraints

HorizontalNav
A component that creates a Navigation bar for a page. Routing is handled as part of the component. **console.tab/horizontalNav** can be used to add additional content to any horizontal navigation.

Example

```
const HomePage: React.FC = (props) => {
    const page = {
      href: '/home',
      name: 'Home',
      component: () => <>Home</>
    }
    return <HorizontalNav match={props.match} pages={[page]} />
}
```

Parameter Name	Description
resource	The resource associated with this Navigation, an object of K8sResourceCommon type
pages	An array of page objects
match	match object provided by React Router

VirtualizedTable
A component for making virtualized tables.

Example

```
const MachineList: React.FC<MachineListProps> = (props) => {
  return (
    <VirtualizedTable<MachineKind>
     {...props}
     aria-label='Machines'
     columns={getMachineColumns}
     Row={getMachineTableRow}
    />
  );
}
```

Parameter Name	Description
data	data for table
loaded	flag indicating data is loaded
loadError	error object if issue loading data
columns	column setup
Row	row setup
unfilteredData	original data without filter
NoDataEmptyMsg	(optional) no data empty message component
EmptyMsg	(optional) empty message component
scrollNode	(optional) function to handle scroll
label	(optional) label for table
ariaLabel	(optional) aria label
gridBreakPoint	sizing of how to break up grid for responsiveness
onSelect	(optional) function for handling select of table
rowData	(optional) data specific to row

TableData
Component for displaying table data within a table row.

Example

```
const PodRow: React.FC<RowProps<K8sResourceCommon>> = ({ obj, activeColumnIDs }) => {
  return (
    <>
      <TableData id={columns[0].id} activeColumnIDs={activeColumnIDs}>
        <ResourceLink kind="Pod" name={obj.metadata.name} namespace={obj.metadata.namespace}
/>
      </TableData>
      <TableData id={columns[1].id} activeColumnIDs={activeColumnIDs}>
        <ResourceLink kind="Namespace" name={obj.metadata.namespace} />
      </TableData>
    </>
  );
};
```

Parameter Name	Description
id	unique ID for table
activeColumnIDs	active columns
className	(optional) option class name for styling

useActiveColumns
A hook that provides a list of user-selected active TableColumns.

Example

```
// See implementation for more details on TableColumn type
  const [activeColumns, userSettingsLoaded] = useActiveColumns({
    columns,
    showNamespaceOverride: false,
    columnManagementID,
  });
  return userSettingsAreLoaded ? <VirtualizedTable columns={activeColumns} {...otherProps} /> : null
```

Parameter Name	Description
options	Which are passed as a key-value map
\{TableColumn[]} options.columns	An array of all available TableColumns
{boolean} [options.showNamespaceOverride]	(optional) If true, a namespace column will be included, regardless of column management selections

Parameter Name	Description
{string} [options.columnManagementID]	(optional) A unique ID used to persist and retrieve column management selections to and from user settings. Usually a group/version/kind (GVK) string for a resource.

A tuple containing the current user selected active columns (a subset of options.columns), and a boolean flag indicating whether user settings have been loaded.

ListPageHeader
Component for generating a page header.

Example

```
const exampleList: React.FC = () => {
  return (
    <>
      <ListPageHeader title="Example List Page"/>
    </>
  );
};
```

Parameter Name	Description
title	heading title
helpText	(optional) help section as react node
badge	(optional) badge icon as react node

ListPageCreate
Component for adding a create button for a specific resource kind that automatically generates a link to the create YAML for this resource.

Example

```
const exampleList: React.FC<MyProps> = () => {
  return (
    <>
      <ListPageHeader title="Example Pod List Page"/>
        <ListPageCreate groupVersionKind="Pod">Create Pod</ListPageCreate>
      </ListPageHeader>
    </>
  );
};
```

Parameter Name	Description
groupVersionKind	the resource group/version/kind to represent

ListPageCreateLink
Component for creating a stylized link.

Example

```
const exampleList: React.FC<MyProps> = () => {
 return (
  <>
   <ListPageHeader title="Example Pod List Page"/>
      <ListPageCreateLink to={'/link/to/my/page'}>Create Item</ListPageCreateLink>
   </ListPageHeader>
  </>
 );
};
```

Parameter Name	Description
to	string location where link should direct
createAccessReview	(optional) object with namespace and kind used to determine access
children	(optional) children for the component

ListPageCreateButton
Component for creating button.

Example

```
const exampleList: React.FC<MyProps> = () => {
  return (
    <>
      <ListPageHeader title="Example Pod List Page"/>
        <ListPageCreateButton createAccessReview={access}>Create Pod</ListPageCreateButton>
      </ListPageHeader>
    </>
  );
};
```

Parameter Name	Description
createAccessReview	(optional) object with namespace and kind used to determine access

Parameter Name	Description
pfButtonProps	(optional) Patternfly Button props

ListPageCreateDropdown
Component for creating a dropdown wrapped with permissions check.

Example

```
const exampleList: React.FC<MyProps> = () => {
  const items = {
    SAVE: 'Save',
    DELETE: 'Delete',
  }
  return (
    <>
     <ListPageHeader title="Example Pod List Page"/>
       <ListPageCreateDropdown createAccessReview={access}
items={items}>Actions</ListPageCreateDropdown>
     </ListPageHeader>
    </>
  );
};
```

Parameter Name	Description
items	key:ReactNode pairs of items to display in dropdown component
onClick	callback function for click on dropdown items
createAccessReview	(optional) object with namespace and kind used to determine access
children	(optional) children for the dropdown toggle

ListPageFilter
Component that generates filter for list page.

Example

```
  // See implementation for more details on RowFilter and FilterValue types
  const [staticData, filteredData, onFilterChange] = useListPageFilter(
    data,
    rowFilters,
    staticFilters,
  );
  // ListPageFilter updates filter state based on user interaction and resulting filtered data can be
rendered in an independent component.
  return (
    <>
      <ListPageHeader .../>
      <ListPagBody>
        <ListPageFilter data={staticData} onFilterChange={onFilterChange} />
        <List data={filteredData} />
      </ListPageBody>
    </>
  )
```

Parameter Name	Description
data	An array of data points
loaded	indicates that data has loaded
onFilterChange	callback function for when filter is updated
rowFilters	(optional) An array of RowFilter elements that define the available filter options
nameFilterPlaceholder	(optional) placeholder for name filter
labelFilterPlaceholder	(optional) placeholder for label filter
hideLabelFilter	(optional) only shows the name filter instead of both name and label filter
hideNameLabelFilter	(optional) hides both name and label filter
columnLayout	(optional) column layout object
hideColumnManagement	(optional) flag to hide the column management

useListPageFilter
A hook that manages filter state for the ListPageFilter component. It returns a tuple containing the data filtered by all static filters, the data filtered by all static and row filters, and a callback that updates rowFilters.

Example

```
  // See implementation for more details on RowFilter and FilterValue types
  const [staticData, filteredData, onFilterChange] = useListPageFilter(
    data,
    rowFilters,
    staticFilters,
  );
  // ListPageFilter updates filter state based on user interaction and resulting filtered data can be
rendered in an independent component.
  return (
    <>
```

```
      <ListPageHeader .../>
      <ListPagBody>
        <ListPageFilter data={staticData} onFilterChange={onFilterChange} />
        <List data={filteredData} />
      </ListPageBody>
    </>
  )
```

Parameter Name	Description
data	An array of data points
rowFilters	(optional) An array of RowFilter elements that define the available filter options
staticFilters	(optional) An array of FilterValue elements that are statically applied to the data

ResourceLink
Component that creates a link to a specific resource type with an icon badge.

Example

```
  <ResourceLink
      kind="Pod"
      name="testPod"
      title={metadata.uid}
  />
```

Parameter Name	Description
kind	(optional) the kind of resource i.e. Pod, Deployment, Namespace
groupVersionKind	(optional) object with group, version, and kind
className	(optional) class style for component
displayName	(optional) display name for component, overwrites the resource name if set
inline	(optional) flag to create icon badge and name inline with children
linkTo	(optional) flag to create a Link object - defaults to true
name	(optional) name of resource

Parameter Name	Description
namesapce	(optional) specific namespace for the kind resource to link to
hideIcon	(optional) flag to hide the icon badge
title	(optional) title for the link object (not displayed)
dataTest	(optional) identifier for testing
onClick	(optional) callback function for when component is clicked
truncate	(optional) flag to truncate the link if too long

ResourceIcon
Component that creates an icon badge for a specific resource type.

Example

```
<ResourceIcon kind="Pod"/>
```

Parameter Name	Description
kind	(optional) the kind of resource i.e. Pod, Deployment, Namespace
groupVersionKind	(optional) object with group, version, and kind
className	(optional) class style for component

useK8sModel
Hook that retrieves the k8s model for provided K8sGroupVersionKind from redux. It returns an array with the first item as k8s model and second item as **inFlight** status.

Example

```
const Component: React.FC = () => {
  const [model, inFlight] = useK8sModel({ group: 'app'; version: 'v1'; kind: 'Deployment' });
  return ...
}
```

Parameter Name	Description

Parameter Name	Description
groupVersionKind	group, version, kind of k8s resource K8sGroupVersionKind is preferred alternatively can pass reference for group, version, kind which is deprecated, i.e, group/version/kind (GVK) K8sResourceKindReference.

useK8sModels
Hook that retrieves all current k8s models from redux. It returns an array with the first item as the list of k8s model and second item as **inFlight** status.

Example

```
const Component: React.FC = () => {
  const [models, inFlight] = UseK8sModels();
  return ...
}
```

useK8sWatchResource
Hook that retrieves the k8s resource along with status for loaded and error. It returns an array with first item as resource(s), second item as loaded status and third item as error state if any.

Example

```
const Component: React.FC = () => {
  const watchRes = {
        ...
      }
  const [data, loaded, error] = useK8sWatchResource(watchRes)
  return ...
}
```

Parameter Name	Description
initResource	options needed to watch for resource.

useK8sWatchResources
Hook that retrieves the k8s resources along with their respective status for loaded and error. It returns a map where keys are as provided in initResouces and value has three properties data, loaded and error.

Example

```
const Component: React.FC = () => {
  const watchResources = {
        'deployment': {...},
        'pod': {...}
        ...
```

```
    }
  const {deployment, pod} = useK8sWatchResources(watchResources)
  return ...
}
```

Parameter Name	Description
initResources	Resources must be watched as key-value pair, wherein key will be unique to resource and value will be options needed to watch for the respective resource.

consoleFetch
A custom wrapper around **fetch** that adds console specific headers and allows for retries and timeouts.It also validates the response status code and throws appropriate error or logs out the user if required. It returns a promise that resolves to the response.

Parameter Name	Description
url	The URL to fetch
options	The options to pass to fetch
timeout	The timeout in milliseconds

consoleFetchJSON
A custom wrapper around **fetch** that adds console specific headers and allows for retries and timeouts. It also validates the response status code and throws appropriate error or logs out the user if required. It returns the response as a JSON object. Uses **consoleFetch** internally. It returns a promise that resolves to the response as JSON object.

Parameter Name	Description
url	The URL to fetch
method	The HTTP method to use. Defaults to GET
options	The options to pass to fetch
timeout	The timeout in milliseconds
cluster	The name of the cluster to make the request to. Defaults to the active cluster the user has selected

consoleFetchText
A custom wrapper around **fetch** that adds console specific headers and allows for retries and timeouts. It also validates the response status code and throws appropriate error or logs out the user if required. It returns the response as a text. Uses **consoleFetch** internally. It returns a promise that resolves to the

response as text.

Parameter Name	Description
url	The URL to fetch
options	The options to pass to fetch
timeout	The timeout in milliseconds
cluster	The name of the cluster to make the request to. Defaults to the active cluster the user has selected

getConsoleRequestHeaders
A function that creates impersonation and multicluster related headers for API requests using current redux state. It returns an object containing the appropriate impersonation and clustr requst headers, based on redux state.

Parameter Name	Description
targetCluster	Override the current active cluster with the provided targetCluster

k8sGetResource
It fetches a resource from the cluster, based on the provided options. If the name is provided it returns one resource else it returns all the resources matching the model. It returns a promise that resolves to the response as JSON object with a resource if the name is providedelse it returns all the resources matching the model. In case of failure, the promise gets rejected with HTTP error response.

Parameter Name	Description
options	Which are passed as key-value pairs in the map
options.model	k8s model
options.name	The name of the resource, if not provided then it will look for all the resources matching the model.
options.ns	The namespace to look into, should not be specified for cluster-scoped resources.
options.path	Appends as subpath if provided
options.queryParams	The query parameters to be included in the URL.

Parameter Name	Description
options.requestInit	The fetch init object to use. This can have request headers, method, redirect, etc. See Interface RequestInit for more.

k8sCreateResource
It creates a resource in the cluster, based on the provided options. It returns a promise that resolves to the response of the resource created. In case of failure promise gets rejected with HTTP error response.

Parameter Name	Description
options	Which are passed as key-value pairs in the map
options.model	k8s model
options.data	Payload for the resource to be created
options.path	Appends as subpath if provided
options.queryParams	The query parameters to be included in the URL.

k8sUpdateResource
It updates the entire resource in the cluster, based on providedoptions. When a client needs to replace an existing resource entirely, they can use k8sUpdate. Alternatively can use k8sPatch to perform the partial update. It returns a promise that resolves to the response of the resource updated. In case of failure promise gets rejected with HTTP error response.

Parameter Name	Description
options	Which are passed as key-value pair in the map
options.model	k8s model
options.data	Payload for the k8s resource to be updated
options.ns	Namespace to look into, it should not be specified for cluster-scoped resources.
options.name	Resource name to be updated.
options.path	Appends as subpath if provided
options.queryParams	The query parameters to be included in the URL.

k8sPatchResource
It patches any resource in the cluster, based on provided options. When a client needs to perform the

partial update, they can use k8sPatch. Alternatively can use k8sUpdate to replace an existing resource entirely. See Data Tracker for more. It returns a promise that resolves to the response of the resource patched. In case of failure promise gets rejected with HTTP error response.

Parameter Name	Description
options	Which are passed as key-value pairs in the map.
options.model	k8s model
options.resource	The resource to be patched.
options.data	Only the data to be patched on existing resource with the operation, path, and value.
options.path	Appends as subpath if provided.
options.queryParams	The query parameters to be included in the URL.

k8sDeleteResource
It deletes resources from the cluster, based on the provided model, resource. The garbage collection works based on **Foreground**|**Background** can be configured with propagationPolicy property in provided model or passed in json. It returns a promise that resolves to the response of kind Status. In case of failure promise gets rejected with HTTP error response.

Example

kind: 'DeleteOptions', apiVersion: 'v1', propagationPolicy

Parameter Name	Description
options	Which are passed as key-value pair in the map.
options.model	k8s model
options.resource	The resource to be deleted.
options.path	Appends as subpath if provided
options.queryParams	The query parameters to be included in the URL.
options.requestInit	The fetch init object to use. This can have request headers, method, redirect, etc. See Interface RequestInit for more.
options.json	Can control garbage collection of resources explicitly if provided else will default to model's "propagationPolicy".

k8sListResource
Lists the resources as an array in the cluster, based on provided options. It returns a promise that resolves to the response.

Parameter Name	Description
options	Which are passed as key-value pairs in the map
options.model	k8s model
options.queryParams	The query parameters to be included in the URL and can pass label selector's as well with key "labelSelector".
options.requestInit	The fetch init object to use. This can have request headers, method, redirect, etc. See Interface RequestInit for more.

k8sListResourceItems
Same interface as k8sListResource but returns the sub items. It returns the apiVersion for the model, i.e., **group/version**.

getAPIVersionForModel
Provides apiVersion for a k8s model.

Parameter Name	Description
model	k8s model

getGroupVersionKindForResource
Provides a group, version, and kind for a resource. It returns the group, version, kind for the provided resource. If the resource does not have an API group, group "core" will be returned. If the resource has an invalid apiVersion, then it will throw an Error.

Parameter Name	Description
resource	k8s resource

getGroupVersionKindForModel
Provides a group, version, and kind for a k8s model. This returns the group, version, kind for the provided model. If the model does not have an apiGroup, group "core" will be returned.

Parameter Name	Description
model	k8s model

StatusPopupSection

Component that shows the status in a popup window. Helpful component for building **console.dashboards/overview/health/resource** extensions.

Example

```
<StatusPopupSection
  firstColumn={
    <>
      <span>{title}</span>
      <span className="text-secondary">
        My Example Item
      </span>
    </>
  }
  secondColumn='Status'
>
```

Parameter Name	Description
firstColumn	values for first column of popup
secondColumn	(optional) values for second column of popup
children	(optional) children for the popup

StatusPopupItem
Status element used in status popup; used in **StatusPopupSection**.

Example

```
<StatusPopupSection
   firstColumn='Example'
   secondColumn='Status'
>
   <StatusPopupItem icon={healthStateMapping[MCGMetrics.state]?.icon}>
      Complete
   </StatusPopupItem>
   <StatusPopupItem icon={healthStateMapping[RGWMetrics.state]?.icon}>
       Pending
   </StatusPopupItem>
</StatusPopupSection>
```

Parameter Name	Description
value	(optional) text value to display
icon	(optional) icon to display
children	child elements

Overview
Creates a wrapper component for a dashboard.

Example

```
<Overview>
  <OverviewGrid mainCards={mainCards} leftCards={leftCards} rightCards={rightCards} />
</Overview>
```

Parameter Name	Description
className	(optional) style class for div
children	(optional) elements of the dashboard

OverviewGrid
Creates a grid of card elements for a dashboard; used within **Overview**.

Example

```
<Overview>
  <OverviewGrid mainCards={mainCards} leftCards={leftCards} rightCards={rightCards} />
</Overview>
```

Parameter Name	Description
mainCards	cards for grid
leftCards	(optional) cards for left side of grid
rightCards	(optional) cards for right side of grid

InventoryItem
Creates an inventory card item.

Example

```
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
)
```

Parameter Name	Description
children	elements to render inside the item

InventoryItemTitle
Creates a title for an inventory card item; used within **InventoryItem**.

Example

```
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
)
```

Parameter Name	Description
children	elements to render inside the title

InventoryItemBody
Creates the body of an inventory card; used within **InventoryCard** and can be used with **InventoryTitle**.

Example

```
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
)
```

Parameter Name	Description
children	elements to render inside the Inventory Card or title
error	elements of the div

InventoryItemStatus
Creates a count and icon for an inventory card with optional link address; used within **InventoryItemBody**

Example

```
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
)
```

Parameter Name	Description
count	count for display
icon	icon for display
linkTo	(optional) link address

InventoryItemLoading
Creates a skeleton container for when an inventory card is loading; used with **InventoryItem** and related components

Example

```
if (loadError) {
   title = <Link to={workerNodesLink}>{t('Worker Nodes')}</Link>;
} else if (!loaded) {
  title = <><InventoryItemLoading /><Link to={workerNodesLink}>{t('Worker Nodes')}</Link></>;
}
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
  </InventoryItem>
)
```

useFlag
Hook that returns the given feature flag from FLAGS redux state. It returns the boolean value of the requested feature flag or undefined.

Parameter Name	Description
flag	The feature flag to return

CodeEditor
A basic lazy loaded Code editor with hover help and completion.

Example

```
<React.Suspense fallback={<LoadingBox />}>
  <CodeEditor
    value={code}
    language="yaml"
  />
</React.Suspense>
```

Parameter Name	Description
value	String representing the yaml code to render.
language	String representing the language of the editor.
options	Monaco editor options. For more details, please, visit Interface IStandAloneEditorConstructionOptions.
minHeight	Minimum editor height in valid CSS height values.
showShortcuts	Boolean to show shortcuts on top of the editor.
toolbarLinks	Array of ReactNode rendered on the toolbar links section on top of the editor.
onChange	Callback for on code change event.
onSave	Callback called when the command CTRL / CMD + S is triggered.
ref	React reference to **{ editor?: IStandaloneCodeEditor }**. Using the **editor** property, you are able to access to all methods to control the editor. For more information, visit Interface IStandaloneCodeEditor.

ResourceYAMLEditor
A lazy loaded YAML editor for Kubernetes resources with hover help and completion. The component use the YAMLEditor and add on top of it more functionality likeresource update handling, alerts, save, cancel and reload buttons, accessibility and more. Unless **onSave** callback is provided, the resource update is automatically handled.It should be wrapped in a **React.Suspense** component.

Example

```
<React.Suspense fallback={<LoadingBox />}>
  <ResourceYAMLEditor
    initialResource={resource}
    header="Create resource"
    onSave={(content) => updateResource(content)}
  />
</React.Suspense>
```

Parameter Name	Description
initialResource	YAML/Object representing a resource to be shown by the editor. This prop is used only during the initial render
header	Add a header on top of the YAML editor
onSave	Callback for the Save button. Passing it will override the default update performed on the resource by the editor

ResourceEventStream
A component to show events related to a particular resource.

Example

```
const [resource, loaded, loadError] = useK8sWatchResource(clusterResource);
return <ResourceEventStream resource={resource} />
```

Parameter Name	Description
resource	An object whose related events should be shown.

usePrometheusPoll
Sets up a poll to Prometheus for a single query. It returns a tuple containing the query response, a boolean flag indicating whether the response has completed, and any errors encountered during the request or post-processing of the request.

Parameter Name	Description
{PrometheusEndpoint} props.endpoint	one of the PrometheusEndpoint (label, query, range, rules, targets)
{string} [props.query]	(optional) Prometheus query string. If empty or undefined, polling is not started.
{number} [props.delay]	(optional) polling delay interval (ms)
{number} [props.endTime]	(optional) for QUERY_RANGE enpoint, end of the query range
{number} [props.samples]	(optional) for QUERY_RANGE enpoint
{number} [options.timespan]	(optional) for QUERY_RANGE enpoint

Parameter Name	Description
{string} [options.namespace]	(optional) a search param to append
{string} [options.timeout]	(optional) a search param to append

Timestamp
A component to render timestamp. The timestamps are synchronized between invidual instances of the Timestamp component. The provided timestamp is formatted according to user locale.

Parameter Name	Description
timestamp	the timestamp to render. Format is expected to be ISO 8601 (used by Kubernetes), epoch timestamp, or an instance of a Date.
simple	render simple version of the component omitting icon and tooltip.
omitSuffix	formats the date ommiting the suffix.
className	additional class name for the component.

useModal
A hook to launch Modals.

Example

```
const context: AppPage: React.FC = () => {<br/> const [launchModal] = useModal();<br/> const
onClick = () => launchModal(ModalComponent);<br/> return (<br/>   <Button onClick=
{onClick}>Launch a Modal</Button><br/> )<br/>}<br/>`
```

ActionServiceProvider
Component that allows to receive contributions from other plugins for the **console.action/provider** extension type.

Example

```
  const context: ActionContext = { 'a-context-id': { dataFromDynamicPlugin } };

  ...

  <ActionServiceProvider context={context}>
      {({ actions, options, loaded }) =>
        loaded && (
          <ActionMenu actions={actions} options={options} variant={ActionMenuVariant.DROPDOWN}
/>
        )
      }
  </ActionServiceProvider>
```

Parameter Name	Description
context	Object with contextId and optional plugin data

NamespaceBar
A component that renders a horizontal toolbar with a namespace dropdown menu in the leftmost position. Additional components can be passed in as children and will be rendered to the right of the namespace dropdown. This component is designed to be used at the top of the page. It should be used on pages where the user needs to be able to change the active namespace, such as on pages with k8s resources.

Example

```
const logNamespaceChange = (namespace) => console.log(`New namespace: ${namespace}`);

...

<NamespaceBar onNamespaceChange={logNamespaceChange}>
  <NamespaceBarApplicationSelector />
</NamespaceBar>
<Page>

  ...
```

Parameter Name	Description
onNamespaceChange	(optional) A function that is executed when a namespace option is selected. It accepts the new namespace in the form of a string as its only argument. The active namespace is updated automatically when an option is selected, but additional logic can be applied via this function. When the namespace is changed, the namespace parameter in the URL will be changed from the previous namespace to the newly selected namespace.
isDisabled	(optional) A boolean flag that disables the namespace dropdown if set to true. This option only applies to the namespace dropdown and has no effect on child components.
children	(optional) Additional elements to be rendered inside the toolbar to the right of the namespace dropdown.

ErrorBoundaryFallbackPage
Creates full page ErrorBoundaryFallbackPage component to display the "Oh no! Something went wrong." message along with the stack trace and other helpful debugging information. This is to be used inconjunction with an component.

Example

```
//in ErrorBoundary component
 return (
   if (this.state.hasError) {
     return <ErrorBoundaryFallbackPage errorMessage={errorString} componentStack=
{componentStackString}
      stack={stackTraceString} title={errorString}/>;
   }

   return this.props.children;
)
```

Parameter Name	Description
errorMessage	text description of the error message
componentStack	component trace of the exception
stack	stack trace of the exception
title	title to render as the header of the error boundary page

QueryBrowser
A component that renders a graph of the results from a Prometheus PromQL query along with controls for interacting with the graph.

Example

```
<QueryBrowser
  defaultTimespan={15 * 60 * 1000}
  namespace={namespace}
  pollInterval={30 * 1000}
  queries={[
    'process_resident_memory_bytes{job="console"}',
    'sum(irate(container_network_receive_bytes_total[6h:5m])) by (pod)',
  ]}
/>
```

Parameter Name	Description
customDataSource	(optional) Base URL of an API endpoint that handles PromQL queries. If provided, this is used instead of the default API for fetching data.
defaultSamples	(optional) The default number of data samples plotted for each data series. If there are many data series, QueryBrowser might automatically pick a lower number of data samples than specified here.

Parameter Name	Description
defaultTimespan	(optional) The default timespan for the graph in milliseconds - defaults to 1,800,000 (30 minutes).
disabledSeries	(optional) Disable (don’t display) data series with these exact label / value pairs.
disableZoom	(optional) Flag to disable the graph zoom controls.
filterLabels	(optional) Optionally filter the returned data series to only those that match these label / value pairs.
fixedEndTime	(optional) Set the end time for the displayed time range rather than showing data up to the current time.
formatSeriesTitle	(optional) Function that returns a string to use as the title for a single data series.
GraphLink	(optional) Component for rendering a link to another page (for example getting more information about this query).
hideControls	(optional) Flag to hide the graph controls for changing the graph timespan, and so on.
isStack	(optional) Flag to display a stacked graph instead of a line graph. If showStackedControl is set, it will still be possible for the user to switch to a line graph.
namespace	(optional) If provided, data is only returned for this namespace (only series that have this namespace label).
onZoom	(optional) Callback called when the graph is zoomed.
pollInterval	(optional) If set, determines how often the graph is updated to show the latest data (in milliseconds).
queries	Array of PromQL queries to run and display the results in the graph.
showLegend	(optional) Flag to enable displaying a legend below the graph.

Parameter Name	Description
showStackedControl	Flag to enable displaying a graph control for switching between stacked graph mode and line graph mode.
timespan	(optional) The timespan that should be covered by the graph in milliseconds.
units	(optional) Units to display on the Y-axis and in the tooltip.

useAnnotationsModal
A hook that provides a callback to launch a modal for editing Kubernetes resource annotations.

Example

```
const PodAnnotationsButton = ({ pod }) => {
  const { t } = useTranslation();
  const launchAnnotationsModal = useAnnotationsModal<PodKind>(pod);
  return <button onClick={launchAnnotationsModal}>{t('Edit Pod Annotations')}</button>
}
```

Parameter Name	Description
resource	The resource to edit annotations for an object of K8sResourceCommon type.

Returns

A function which will launch a modal for editing a resource's annotations.

useDeleteModal
A hook that provides a callback to launch a modal for deleting a resource.

Example

```
const DeletePodButton = ({ pod }) => {
  const { t } = useTranslation();
  const launchDeleteModal = useDeleteModal<PodKind>(pod);
  return <button onClick={launchDeleteModal}>{t('Delete Pod')}</button>
}
```

Parameter Name	Description
resource	The resource to delete.

Parameter Name	Description
redirectTo	(optional) A location to redirect to after deleting the resource.
message	(optional) A message to display in the modal.
btnText	(optional) The text to display on the delete button.
deleteAllResources	(optional) A function to delete all resources of the same kind.

Returns

A function which will launch a modal for deleting a resource.

useLabelsModel
A hook that provides a callback to launch a modal for editing Kubernetes resource labels.

Example

```
const PodLabelsButton = ({ pod }) => {
  const { t } = useTranslation();
  const launchLabelsModal = useLabelsModal<PodKind>(pod);
  return <button onClick={launchLabelsModal}>{t('Edit Pod Labels')}</button>
}
```

Parameter Name	Description
resource	The resource to edit labels for, an object of K8sResourceCommon type.

Returns

A function which will launch a modal for editing a resource’s labels.

useActiveNamespace
Hook that provides the currently active namespace and a callback for setting the active namespace.

Example

```
const Component: React.FC = (props) => {
   const [activeNamespace, setActiveNamespace] = useActiveNamespace();
   return <select
     value={activeNamespace}
     onChange={(e) => setActiveNamespace(e.target.value)}
   >
     {
       // ...namespace options
```

```
      }
    </select>
  }
```

Returns

A tuple containing the current active namespace and setter callback.

PerspectiveContext
Deprecated: Use the provided **usePerspectiveContext** instead. Creates the perspective context.

Parameter Name	Description
PerspectiveContextType	object with active perspective and setter

useAccessReviewAllowed
Deprecated: Use **useAccessReview** from **@console/dynamic-plugin-sdk** instead. Hook that provides allowed status about user access to a given resource. It returns the **isAllowed** boolean value.

Parameter Name	Description
resourceAttributes	resource attributes for access review
impersonate	impersonation details

useSafetyFirst
Deprecated: This hook is not related to console functionality. Hook that ensures a safe asynchronnous setting of React state in case a given component could be unmounted. It returns an array with a pair of state value and its set function.

Parameter Name	Description
initialState	initial state value

YAMLEditor
Deprecated: A basic lazy loaded YAML editor with hover help and completion.

Example

```
<React.Suspense fallback={<LoadingBox />}>
  <YAMLEditor
    value={code}
  />
</React.Suspense>
```

Parameter Name	Description
value	String representing the yaml code to render.

Parameter Name	Description
options	Monaco editor options.
minHeight	Minimum editor height in valid CSS height values.
showShortcuts	Boolean to show shortcuts on top of the editor.
toolbarLinks	Array of ReactNode rendered on the toolbar links section on top of the editor.
onChange	Callback for on code change event.
onSave	Callback called when the command CTRL / CMD + S is triggered.
ref	React reference to **{ editor?: IStandaloneCodeEditor }**. Using the **editor** property, you are able to access to all methods to control the editor.

4.5.3. Troubleshooting your dynamic plugin

Refer to this list of troubleshooting tips if you run into issues loading your plugin.

- Verify that you have enabled your plugin in the console Operator configuration and your plugin name is the output by running the following command:

  ```
  $ oc get console.operator.openshift.io cluster -o jsonpath='{.spec.plugins}'
  ```

  - Verify the enabled plugins on the status card of the **Overview** page in the **Administrator** perspective. You must refresh your browser if the plugin was recently enabled.

- Verify your plugin service is healthy by:
  - Verifying your plugin pod status is running and your containers are ready.
  - Verifying the service label selector matches the pod and the target port is correct.
  - Curl the **plugin-manifest.json** from the service in a terminal on the console pod or another pod on the cluster.

- Verify your **ConsolePlugin** resource name (**consolePlugin.name**) matches the plugin name used in **package.json**.

- Verify your service name, namespace, port, and path are declared correctly in the **ConsolePlugin** resource.

- Verify your plugin service uses HTTPS and service serving certificates.

- Verify any certificates or connection errors in the console pod logs.

- Verify the feature flag your plugin relys on is not disabled.
- Verify your plugin does not have any **consolePlugin.dependencies** in **package.json** that are not met.
  - This can include console version dependencies or dependencies on other plugins. Filter the JS console in your browser for your plugin's name to see messages that are logged.
- Verify there are no typos in the nav extension perspective or section IDs.
  - Your plugin may be loaded, but nav items missing if IDs are incorrect. Try navigating to a plugin page directly by editing the URL.
- Verify there are no network policies that are blocking traffic from the console pod to your plugin service.
  - If necessary, adjust network policies to allow console pods in the openshift-console namespace to make requests to your service.
- Verify the list of dynamic plugins to be loaded in your browser in the **Console** tab of the developer tools browser.
  - Evaluate **window.SERVER_FLAGS.consolePlugins** to see the dynamic plugin on the Console frontend.

CHAPTER 5. WEB TERMINAL

5.1. INSTALLING THE WEB TERMINAL

You can install the web terminal by using the Web Terminal Operator listed in the Red Hat OpenShift Service on AWS OperatorHub. When you install the Web Terminal Operator, the custom resource definitions (CRDs) that are required for the command line configuration, such as the **DevWorkspace** CRD, are automatically installed. The web console creates the required resources when you open the web terminal.

Prerequisites

- You are logged into the Red Hat OpenShift Service on AWS web console.
- You have cluster administrator permissions.

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Operators → OperatorHub**.
2. Use the **Filter by keyword** box to search for the Web Terminal Operator in the catalog, and then click the **Web Terminal** tile.
3. Read the brief description about the Operator on the **Web Terminal** page, and then click **Install**.
4. On the **Install Operator** page, retain the default values for all fields.
   - The **fast** option in the **Update Channel** menu enables installation of the latest release of the Web Terminal Operator.
   - The **All namespaces on the cluster** option in the **Installation Mode** menu enables the Operator to watch and be available to all namespaces in the cluster.
   - The **openshift-operators** option in the **Installed Namespace** menu installs the Operator in the default **openshift-operators** namespace.
   - The **Automatic** option in the **Approval Strategy** menu ensures that the future upgrades to the Operator are handled automatically by the Operator Lifecycle Manager.
5. Click **Install**.
6. In the **Installed Operators** page, click the **View Operator** to verify that the Operator is listed on the **Installed Operators** page.

   NOTE

   The Web Terminal Operator installs the DevWorkspace Operator as a dependency.

7. After the Operator is installed, refresh your page to see the command line terminal icon (>_) in the masthead of the console.

5.2. USING THE WEB TERMINAL

You can launch an embedded command line terminal instance in the web console. This terminal instance is preinstalled with common CLI tools for interacting with the cluster, such as **oc**, **kubectl**,**odo**, **kn**, **tkn**, **helm**, and **subctl**. It also has the context of the project you are working on and automatically logs you in using your credentials.

5.2.1. Accessing the web terminal

After the Web Terminal Operator is installed, you can access the web terminal. After the web terminal is initialized, you can use the preinstalled CLI tools like **oc**, **kubectl**, **odo**, **kn**, **tkn**, **helm**, and **subctl** in the web terminal. You can re-run commands by selecting them from the list of commands you have run in the terminal. These commands persist across multiple terminal sessions. The web terminal remains open until you close it or until you close the browser window or tab.

Prerequisites

- You have access to an Red Hat OpenShift Service on AWS cluster and are logged into the web console.
- The Web Terminal Operator is installed on your cluster.

Procedure

1. To launch the web terminal, click the command line terminal icon ( ) in the masthead of the console. A web terminal instance is displayed in the **Command line terminal** pane. This instance is automatically logged in with your credentials.
2. If a project has not been selected in the current session, select the project where the **DevWorkspace** CR must be created from the **Project** drop-down list. By default, the current project is selected.

   NOTE

   - The **DevWorkspace** CR is created only if it does not already exist.

3. Click **Start** to initialize the web terminal using the selected project.
4. Click **+** to open multiple tabs within the web terminal in the console.

5.3. TROUBLESHOOTING THE WEB TERMINAL

5.3.1. Web terminal and network policies

The web terminal might fail to launch if the cluster has network policies configured. To initialize a web terminal instance, the Web Terminal Operator must communicate with the web terminal's pod to verify it is running, and the Red Hat OpenShift Service on AWS web console needs to send information to automatically log in to the cluster within the terminal. If either step fails, the web terminal fails to initialize and the terminal panel appears to be in a loading state.

To avoid this issue, ensure that the network policies for namespaces that are used for terminals allow ingress from the **openshift-console** and **openshift-operators** namespaces.

5.4. UNINSTALLING THE WEB TERMINAL

Uninstalling the Web Terminal Operator does not remove any of the custom resource definitions (CRDs) or managed resources that are created when the Operator is installed. For security purposes, you must manually uninstall these components. By removing these components, you save cluster resources because terminals do not idle when the Operator is uninstalled.

Uninstalling the web terminal is a two-step process:

1. Uninstall the Web Terminal Operator and related custom resources (CRs) that were added when you installed the Operator.

2. Uninstall the DevWorkspace Operator and its related custom resources that were added as a dependency of the Web Terminal Operator.

5.4.1. Removing the Web Terminal Operator

You can uninstall the web terminal by removing the Web Terminal Operator and custom resources used by the Operator.

Prerequisites

- You have access to an Red Hat OpenShift Service on AWS cluster with cluster administrator permissions.

- You have installed the **oc** CLI.

Procedure

1. In the **Administrator** perspective of the web console, navigate to **Operators → Installed Operators**.

2. Scroll the filter list or type a keyword into the **Filter by name** box to find the Web Terminal Operator.

3. Click the Options menu ⋮ for the Web Terminal Operator, and then select **Uninstall Operator**.

4. In the **Uninstall Operator** confirmation dialog box, click **Uninstall** to remove the Operator, Operator deployments, and pods from the cluster. The Operator stops running and no longer receives updates.

5.4.2. Removing the DevWorkspace Operator

To completely uninstall the web terminal, you must also remove the DevWorkspace Operator and custom resources used by the Operator.

IMPORTANT

The DevWorkspace Operator is a standalone Operator and may be required as a dependency for other Operators installed in the cluster. Follow the steps below only if you are sure that the DevWorkspace Operator is no longer needed.

Prerequisites

- You have access to an Red Hat OpenShift Service on AWS cluster with cluster administrator permissions.
- You have installed the **oc** CLI.

Procedure

1. Remove the **DevWorkspace** custom resources used by the Operator, along with any related Kubernetes objects:

```
$ oc delete devworkspaces.workspace.devfile.io --all-namespaces --all --wait
```

```
$ oc delete devworkspaceroutings.controller.devfile.io --all-namespaces --all --wait
```

> **WARNING**
>
> If this step is not complete, finalizers make it difficult to fully uninstall the Operator.

2. Remove any remaining services, secrets, and config maps. Depending on the installation, some resources included in the following commands may not exist in the cluster.

```
$ oc delete all --selector app.kubernetes.io/part-of=devworkspace-
operator,app.kubernetes.io/name=devworkspace-webhook-server -n openshift-operators
```

```
$ oc delete serviceaccounts devworkspace-webhook-server -n openshift-operators
```

```
$ oc delete clusterrole devworkspace-webhook-server
```

```
$ oc delete clusterrolebinding devworkspace-webhook-server
```

3. Uninstall the DevWorkspace Operator:
    a. In the **Administrator** perspective of the web console, navigate to **Operators → Installed Operators**.
    b. Scroll the filter list or type a keyword into the **Filter by name** box to find the DevWorkspace Operator.
    c. Click the Options menu ⋮ for the Operator, and then select **Uninstall Operator**.
    d. In the **Uninstall Operator** confirmation dialog box, click **Uninstall** to remove the Operator, Operator deployments, and pods from the cluster. The Operator stops running and no longer receives updates.

CHAPTER 6. DISABLING THE WEB CONSOLE IN RED HAT OPENSHIFT SERVICE ON AWS

You can disable the Red Hat OpenShift Service on AWS web console.

6.1. PREREQUISITES

- Deploy an Red Hat OpenShift Service on AWS cluster.

6.2. DISABLING THE WEB CONSOLE

You can disable the web console by editing the **consoles.operator.openshift.io** resource.

- Edit the **consoles.operator.openshift.io** resource:

  ```
  $ oc edit consoles.operator.openshift.io cluster
  ```

  The following example displays the parameters from this resource that you can modify:

  ```
  apiVersion: operator.openshift.io/v1
  kind: Console
  metadata:
    name: cluster
  spec:
    managementState: Removed 1
  ```

  1 Set the **managementState** parameter value to **Removed** to disable the web console. The other valid values for this parameter are **Managed**, which enables the console under the cluster’s control, and **Unmanaged**, which means that you are taking control of web console management.

CHAPTER 7. ABOUT QUICK START TUTORIALS

If you are creating quick start tutorials for the Red Hat OpenShift Service on AWS web console, follow these guidelines to maintain a consistent user experience across all quick starts.

7.1. UNDERSTANDING QUICK STARTS

A quick start is a guided tutorial with user tasks. In the web console, you can access quick starts under the **Help** menu. They are especially useful for getting oriented with an application, Operator, or other product offering.

A quick start primarily consists of tasks and steps. Each task has multiple steps, and each quick start has multiple tasks. For example:

- Task 1
  - Step 1
  - Step 2
  - Step 3
- Task 2
  - Step 1
  - Step 2
  - Step 3
- Task 3
  - Step 1
  - Step 2
  - Step 3

7.2. QUICK START USER WORKFLOW

When you interact with an existing quick start tutorial, this is the expected workflow experience:

1. In the **Administrator** or **Developer** perspective, click the **Help icon** and select **Quick Starts**.
2. Click a quick start card.
3. In the panel that appears, click **Start**.
4. Complete the on-screen instructions, then click **Next**.
5. In the **Check your work** module that appears, answer the question to confirm that you successfully completed the task.
   a. If you select **Yes**, click **Next** to continue to the next task.
   b. If you select **No**, repeat the task instructions and check your work again.

6. Repeat steps 1 through 6 above to complete the remaining tasks in the quick start.

7. After completing the final task, click **Close** to close the quick start.

7.3. QUICK START COMPONENTS

A quick start consists of the following sections:

- **Card**: The catalog tile that provides the basic information of the quick start, including title, description, time commitment, and completion status
- **Introduction**: A brief overview of the goal and tasks of the quick start
- **Task headings**: Hyper-linked titles for each task in the quick start
- **Check your work module**: A module for a user to confirm that they completed a task successfully before advancing to the next task in the quick start
- **Hints**: An animation to help users identify specific areas of the product
- **Buttons**
  - **Next and back buttons** Buttons for navigating the steps and modules within each task of a quick start
  - **Final screen buttons**: Buttons for closing the quick start, going back to previous tasks within the quick start, and viewing all quick starts

The main content area of a quick start includes the following sections:

- **Card copy**
- **Introduction**
- **Task steps**
- **Modals and in-app messaging**
- **Check your work module**