OpenShift Container Platform 4.13

Web console

Getting started with the web console in OpenShift Container Platform
OpenShift Container Platform 4.13 Web console

Getting started with the web console in OpenShift Container Platform
Abstract

This document provides instructions for accessing and customizing the OpenShift Container Platform web console.
## Table of Contents

**CHAPTER 1. WEB CONSOLE OVERVIEW** ................................................................. 7  
   1.1. ABOUT THE ADMINISTRATOR PERSPECTIVE IN THE WEB CONSOLE 7  
   1.2. ABOUT THE DEVELOPER PERSPECTIVE IN THE WEB CONSOLE 7  
   1.3. ACCESSING THE PERSPECTIVES 8  

**CHAPTER 2. ACCESSING THE WEB CONSOLE** ..................................................... 10  
   2.1. PREREQUISITES 10  
   2.2. UNDERSTANDING AND ACCESSING THE WEB CONSOLE 10  

**CHAPTER 3. USING THE OPENSOURF CONTAINER PLATFORM DASHBOARD TO GET CLUSTER INFORMATION** ................................................................. 11  
   3.1. ABOUT THE OPENSOURF CONTAINER PLATFORM DASHBOARDS PAGE 11  
   3.2. RECOGNIZING RESOURCE AND PROJECT LIMITS AND QUOTAS 12  

**CHAPTER 4. ADDING USER PREFERENCES** ......................................................... 13  
   4.1. SETTING USER PREFERENCES 13  

**CHAPTER 5. CONFIGURING THE WEB CONSOLE IN OPENSOURF CONTAINER PLATFORM** ................................................................. 14  
   5.1. PREREQUISITES 14  
   5.2. CONFIGURING THE WEB CONSOLE 14  
   5.3. DISABLING QUICK STARTS IN THE WEB CONSOLE 14  

**CHAPTER 6. CUSTOMIZING THE WEB CONSOLE IN OPENSOURF CONTAINER PLATFORM** ................................................................. 16  
   6.1. ADDING A CUSTOM LOGO AND PRODUCT NAME 16  
   6.2. CREATING CUSTOM LINKS IN THE WEB CONSOLE 17  
   6.3. CUSTOMIZING CONSOLE ROUTES 18  
      6.3.1. Customizing the console route 18  
      6.3.2. Customizing the download route 19  
   6.4. CUSTOMIZING THE LOGIN PAGE 20  
   6.5. DEFINING A TEMPLATE FOR AN EXTERNAL LOG LINK 21  
   6.6. CREATING CUSTOM NOTIFICATION BANNERS 22  
   6.7. CUSTOMIZING CLI DOWNLOADS 22  
   6.8. ADDING YAML EXAMPLES TO KUBERNETES RESOURCES 23  
   6.9. CUSTOMIZING USER PERSPECTIVES 24  
      6.9.1. Customizing a perspective using YAML view 24  
      6.9.2. Customizing a perspective using form view 26  
   6.10. DEVELOPER CATALOG AND SUB-CATALOG CUSTOMIZATION 26  
      6.10.1. Customizing a developer catalog or its sub-catalogs using the YAML view 27  
      6.10.2. Customizing a developer catalog or its sub-catalogs using the form view 28  
      6.10.2.1. Example YAML file changes 29  

**CHAPTER 7. DYNAMIC PLUGINS** ................................................................. 31  
   7.1. OVERVIEW OF DYNAMIC PLUGINS 31  
      7.1.1. About dynamic plugins 31  
      7.1.2. Key features 31  
      7.1.3. General guidelines 31  
         PatternFly guidelines 31  
   7.2. GETTING STARTED WITH DYNAMIC PLUGINS 32  
   7.2.1. Dynamic plugin development 32  
   7.3. DEPLOY YOUR PLUGIN ON A CLUSTER 32  
      7.3.1. Build an image with Docker 33  
      7.3.2. Deploy your plugin on a cluster 33  
      7.3.3. Disabling your plugin in the browser 35
7.3.4. Additional resources

7.4. DYNAMIC PLUGIN EXAMPLE
7.4.1. Adding a tab to the pods page

7.5. DYNAMIC PLUGIN REFERENCE
7.5.1. Dynamic plugin extension types
  console.action/filter
  console.action/group
  console.action/provider
  console.action/resource-provider
  console.alert-action
  console.catalog/item-filter
  console.catalog/item-metadata
  console.catalog/item-provider
  console.catalog/item-type
  console.catalog/item-type-metadata
  console.cluster-overview/inventory-item
  console.cluster-overview/multiline-utilization-item
  console.cluster-overview/utilization-item
  console.context-provider
  console.dashboards/card
  console.dashboards/custom/overview/detail/item
  console.dashboards/overview/activity/resource
  console.dashboards/overview/health/operator
  console.dashboards/overview/health/prometheus
  console.dashboards/overview/health/resource
  console.dashboards/overview/health/url
  console.dashboards/overview/inventory/item
  console.dashboards/overview/inventory/item/group
  console.dashboards/overview/inventory/item/replacement
  console.dashboards/overview/prometheus/activity/resource
  console.dashboards/project/overview/item
  console.dashboards/tab
  console.file-upload
  console.flag
  console.flag/hookProvider
  console.flag/model
  console.global-config
  console.model-metadata
  console.navigation/href
  console.navigation/resource-cluster
  console.navigation/resource-ns
  console.navigation/section
  console.navigation/separators
  console.page/resource/details
  console.page/resource/list
  console.page/route
  console.page/route/standalone
  console.perspective
  console.project-overview/inventory-item
  console.project-overview/utilization-item
  console.pvc/alert
  console.pvc/create-prop
  console.pvc/delete
7.5.2. OpenShift Container Platform console API
CHAPTER 8. WEB TERMINAL

8.1. INSTALLING THE WEB TERMINAL

Prerequisites
Procedure

8.2. CONFIGURING THE WEB TERMINAL

8.2.1. Configuring the web terminal timeout for a session
8.2.2. Configuring the web terminal image for a session

8.3. USING THE WEB TERMINAL

8.3.1. Accessing the web terminal

7.5.3. Troubleshooting your dynamic plugin
8.4. TROUBLESHOOTING THE WEB TERMINAL
8.4.1. Web terminal and network policies
8.5. UNINSTALLING THE WEB TERMINAL
8.5.1. Removing the Web Terminal Operator
8.5.2. Removing the DevWorkspace Operator

CHAPTER 9. DISABLING THE WEB CONSOLE IN OPENShift CONTAINER PLATFORM
9.1. PREREQUISITES
9.2. DISABLING THE WEB CONSOLE

CHAPTER 10. CREATING QUICK START TUTORIALS IN THE WEB CONSOLE
10.1. UNDERSTANDING QUICK STARTS
10.2. QUICK START USER WORKFLOW
10.3. QUICK START COMPONENTS
10.4. CONTRIBUTING QUICK STARTS
10.4.1. Viewing the quick start API documentation
10.4.2. Mapping the elements in the quick start to the quick start CR
10.4.2.1. conclusion element
10.4.2.2. description element
10.4.2.3. displayName element
10.4.2.4. durationMinutes element
10.4.2.5. icon element
10.4.2.6. introduction element
10.4.3. Adding a custom icon to a quick start
10.4.4. Limiting access to a quick start
10.4.5. Linking to other quick starts
10.4.6. Supported tags for quick starts
10.4.7. Quick start highlighting markdown reference
10.4.7.1. Perspective switcher
10.4.7.2. Administrator perspective navigation links
10.4.7.3. Developer perspective navigation links
10.4.7.4. Common navigation links
10.4.7.5. Masthead links
10.4.8. Code snippet markdown reference
10.4.8.1. Syntax for inline code snippets
10.4.8.2. Syntax for multi-line code snippets
10.5. QUICK START CONTENT GUIDELINES
10.5.1. Card copy
10.5.2. Introduction
10.5.3. Task steps
10.5.4. Check your work module
10.5.5. Formatting UI elements
10.6. ADDITIONAL RESOURCES
CHAPTER 1. WEB CONSOLE OVERVIEW

The Red Hat OpenShift Container Platform 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 OpenShift Container Platform. 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 OpenShift Container Platform 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 OpenShift Container Platform.

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 OpenShift Container Platform 4.7 and later.

\[\text{NOTE}\]

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 OpenShift Container Platform 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.

\section*{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

- Learn more about Cluster Administrator
- **Overview of the Administrator** perspective
- Creating and deploying applications on OpenShift Container Platform using the **Developer** perspective
- Viewing the applications in your project, verifying their deployment status, and interacting with them in the **Topology** view
- Viewing cluster information
- Configuring the web console
- Customizing the web console
- Using the web terminal
- Creating quick start tutorials
- Disabling the web console
CHAPTER 2. ACCESSING THE WEB CONSOLE

The OpenShift Container Platform 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 master. The static assets required to run the web console are served by the pod. After OpenShift Container Platform is successfully installed using openshift-install create cluster, find the URL for the web console and login credentials for your installed cluster in the CLI output of the installation program. For example:

Example output

INFO Install complete!
INFO Run 'export KUBECONFIG=<your working directory>/auth/kubeconfig' to manage the cluster with 'oc', the OpenShift CLI.
INFO The cluster is ready when 'oc login -u kubeadmin -p <provided>' succeeds (wait a few minutes).
INFO Access the OpenShift web-console here: https://console-openshift-console.apps.demo1.openshift4-beta-abcorp.com
INFO Login to the console with user: kubeadmin, password: <provided>

Use those details to log in and access the web console.

For existing clusters that you did not install, you can use oc whoami --show-console to see the web console URL.

IMPORTANT

The dir parameter specifies the assets directory, which stores the manifest files, the ISO image, and the auth directory. The auth directory stores the kubeadmin-password and kubeconfig files. As a kubeadmin user, you can use the kubeconfig file to access the cluster with the following setting: export KUBECONFIG=<install_directory>/auth/kubeconfig. The kubeconfig is specific to the generated ISO image, so if the kubeconfig is set and the oc command fails, it is possible that the system did not boot with the generated ISO image. To perform debugging, during the bootstrap process, you can log in to the console as the core user by using the contents of the kubeadmin-password file.

Additional resources

- Enabling feature sets using the web console
CHAPTER 3. USING THE OPENSHIFT CONTAINER PLATFORM DASHBOARD TO GET CLUSTER INFORMATION

The OpenShift Container Platform web console captures high-level information about the cluster.

3.1. ABOUT THE OPENSHIFT CONTAINER PLATFORM DASHBOARDS PAGE

Access the OpenShift Container Platform dashboard, which captures high-level information about the cluster, by navigating to Home → Overview from the OpenShift Container Platform web console.

The OpenShift Container Platform dashboard provides various cluster information, captured in individual dashboard cards.

The OpenShift Container Platform 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. ADDING USER PREFERENCES

You can change the default preferences for your profile to meet your requirements. You can set your default project, topology view (graph or list), editing medium (form or YAML), language preferences, and resource type.

The changes made to the user preferences are automatically saved.

4.1. SETTING USER PREFERENCES

You can set the default user preferences for your cluster.

Procedure

1. Log in to the OpenShift Container Platform web console using your login credentials.

2. Use the masthead to access the user preferences under the user profile.

3. In the General section:
   a. In the Theme field, you can set the theme that you want to work in. The console defaults to the selected theme each time you log in.
   b. In the Perspective field, you can set the default perspective you want to be logged in to. You can select the Administrator or the Developer perspective as required. If a perspective is not selected, you are logged into the perspective you last visited.
   c. In the Project field, select a project you want to work in. The console defaults to the project every time you log in.
   d. In the Topology field, you can set the topology view to default to the graph or list view. If not selected, the console defaults to the last view you used.
   e. In the Create/Edit resource method field, you can set a preference for creating or editing a resource. If both the form and YAML options are available, the console defaults to your selection.

4. In the Language section, select Default browser language to use the default browser language settings. Otherwise, select the language that you want to use for the console.

5. In the Notifications section, you can toggle display notifications created by users for specific projects on the Overview page or notification drawer.

6. In the Applications section:
   a. You can view the default Resource type. For example, if the OpenShift Serverless Operator is installed, the default resource type is Serverless Deployment. Otherwise, the default resource type is Deployment.
   b. You can select another resource type to be the default resource type from the Resource Type field.
CHAPTER 5. CONFIGURING THE WEB CONSOLE IN OPENSHIFT CONTAINER PLATFORM

You can modify the OpenShift Container Platform web console to set a logout redirect URL or disable the quick start tutorials.

5.1. PREREQUISITES

- Deploy an OpenShift Container Platform cluster.

5.2. CONFIGURING THE WEB CONSOLE

You can configure the web console settings by editing the `console.config.openshift.io` resource.

- Edit the `console.config.openshift.io` resource:

  ```shell
  $ oc edit console.config.openshift.io cluster
  ```

  The following example displays the sample resource definition for the console:

  ```yaml
  apiVersion: config.openshift.io/v1
  kind: Console
  metadata:
    name: cluster
  spec:
    authentication:
      logoutRedirect: ""  # Specify the URL of the page to load when a user logs out of the web console. If you do not specify a value, the user returns to the login page for the web console. Specifying a logoutRedirect URL allows your users to perform single logout (SLO) through the identity provider to destroy their single sign-on session.
    status:
      consoleURL: ""  # The web console URL. To update this to a custom value, see Customizing the web console URL.
  ```

5.3. DISABLING QUICK STARTS IN THE WEB CONSOLE

You can use the Administrator perspective of the web console to disable one or more quick starts.

**Prerequisites**

- You have cluster administrator permissions and are logged in to the web console.

**Procedure**

1. In the Administrator perspective, navigate to Administration → Cluster Settings.
2. On the Cluster Settings page, click the Configuration tab.
3. On the Configuration page, click the Console configuration resource with the description `operator.openshift.io`.

4. From the Action drop-down list, select Customize, which opens the Cluster configuration page.

5. On the General tab, in the Quick starts section, you can select items in either the Enabled or Disabled list, and move them from one list to the other by using the arrow buttons.
   - To enable or disable a single quick start, click the quick start, then use the single arrow buttons to move the quick start to the appropriate list.
   - To enable or disable multiple quick starts at once, press Ctrl and click the quick starts you want to move. Then, use the single arrow buttons to move the quick starts to the appropriate list.
   - To enable or disable all quick starts at once, click the double arrow buttons to move all of the quick starts to the appropriate list.
CHAPTER 6. CUSTOMIZING THE WEB CONSOLE IN OPENSHIFT CONTAINER PLATFORM

You can customize the OpenShift Container Platform web console to set a custom logo, product name, links, notifications, and command line downloads. This is especially helpful if you need to tailor the web console to meet specific corporate or government requirements.

6.1. ADDING A CUSTOM LOGO AND PRODUCT NAME

You can create custom branding by adding a custom logo or custom product name. You can set both or one without the other, as these settings are independent of each other.

Prerequisites

- You must have administrator privileges.
- Create a file of the logo that you want to use. The logo can be a file in any common image format, including GIF, JPG, PNG, or SVG, and is constrained to a max-width of 200px and a max-height of 68px. Image size must not exceed 1 MB due to constraints on the ConfigMap object size.

Procedure

1. Import your logo file into a config map in the openshift-config namespace:

   ```bash
   $ oc create configmap console-custom-logo --from-file /path/to/console-custom-logo.png -n openshift-config
   
   TIP
   
   You can alternatively apply the following YAML to create the config map:

   ```yaml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: console-custom-logo
     namespace: openshift-config
   binaryData:
     console-custom-logo.png: <base64-encoded_logo> ...
   
   1 Provide a valid base64-encoded logo.
   ```

2. Edit the web console’s Operator configuration to include customLogoFile and customProductName:

   ```bash
   $ oc edit consoles.operator.openshift.io cluster
   
   apiVersion: operator.openshift.io/v1
   kind: Console
   metadata:
     name: cluster
   spec:
   ```
Once the Operator configuration is updated, it will sync the custom logo config map into the console namespace, mount it to the console pod, and redeploy.

3. Check for success. If there are any issues, the console cluster Operator will report a **Degraded** status, and the console Operator configuration will also report a **CustomLogoDegraded** status, but with reasons like **KeyOrFilenameInvalid** or **NoImageProvided**.

   To check the **clusteroperator**, run:

   ```bash
   $ oc get clusteroperator console -o yaml
   ```

   To check the console Operator configuration, run:

   ```bash
   $ oc get consoles.operator.openshift.io -o yaml
   ```

# 6.2. CREATING CUSTOM LINKS IN THE WEB CONSOLE

**Prerequisites**

- You must have administrator privileges.

**Procedure**

1. From Administration → Custom Resource Definitions, click on **ConsoleLink**.

2. Select **Instances** tab

3. Click **Create Console Link** and edit the file:

   ```yaml
   apiVersion: console.openshift.io/v1
   kind: ConsoleLink
   metadata:
     name: example
   spec:
     href: 'https://www.example.com'
     location: HelpMenu
     text: Link
   ```

   Valid location settings are **HelpMenu**, **UserMenu**, **ApplicationMenu**, and **NamespaceDashboard**.

To make the custom link appear in all namespaces, follow this example:

```yaml
apiVersion: console.openshift.io/v1
kind: ConsoleLink
metadata:
  name: namespaced-dashboard-link-for-all-namespaces
```
To make the custom link appear in only some namespaces, follow this example:

```
apiVersion: console.openshift.io/v1
kind: ConsoleLink
metadata:
  name: namespaced-dashboard-for-some-namespaces
spec:
  href: 'https://www.example.com'
location: NamespaceDashboard
# This text will appear in a box called "Launcher" under "namespace" or "project" in the web console

text: Custom Link Text
namespaceDashboard:
  namespaces:
    # for these specific namespaces
    - my-namespace
    - your-namespace
    - other-namespace
```

To make the custom link appear in the application menu, follow this example:

```
apiVersion: console.openshift.io/v1
kind: ConsoleLink
metadata:
  name: application-menu-link-1
spec:
  href: 'https://www.example.com'
location: ApplicationMenu

text: Link
applicationMenu:
  section: My New Section
  # image that is 24x24 in size
  imageURL: https://via.placeholder.com/24
```

4. Click **Save** to apply your changes.

### 6.3. CUSTOMIZING CONSOLE ROUTES

For **console** and **downloads** routes, custom routes functionality uses the **ingress** config route configuration API. If the **console** custom route is set up in both the **ingress** config and **console-operator** config, then the new **ingress** config custom route configuration takes precedence. The route configuration with the **console-operator** config is deprecated.

#### 6.3.1. Customizing the console route

You can customize the console route by setting the custom hostname and TLS certificate in the `spec.componentRoutes` field of the cluster **Ingress** configuration.

**Prerequisites**
You have logged in to the cluster as a user with administrative privileges.

You have created a secret in the `openshift-config` namespace containing the TLS certificate and key. This is required if the domain for the custom hostname suffix does not match the cluster domain suffix. The secret is optional if the suffix matches.

**TIP**

You can create a TLS secret by using the `oc create secret tls` command.

**Procedure**

1. Edit the cluster **Ingress** configuration:

   ```
   $ oc edit ingress.config.openshift.io cluster
   ```

2. Set the custom hostname and optionally the serving certificate and key:

   ```
   apiVersion: config.openshift.io/v1
   kind: Ingress
   metadata:
     name: cluster
   spec:
     componentRoutes:
     - name: console
       namespace: openshift-console
       hostname: <custom_hostname>  
       servingCertKeyPairSecret:
         name: <secret_name>
   ```

   1. The custom hostname.
   2. Reference to a secret in the `openshift-config` namespace that contains a TLS certificate (`tls.crt`) and key (`tls.key`). This is required if the domain for the custom hostname suffix does not match the cluster domain suffix. The secret is optional if the suffix matches.

3. Save the file to apply the changes.

**6.3.2. Customizing the download route**

You can customize the download route by setting the custom hostname and TLS certificate in the `spec.componentRoutes` field of the cluster **Ingress** configuration.

**Prerequisites**

- You have logged in to the cluster as a user with administrative privileges.
- You have created a secret in the `openshift-config` namespace containing the TLS certificate and key. This is required if the domain for the custom hostname suffix does not match the cluster domain suffix. The secret is optional if the suffix matches.
TIP

You can create a TLS secret by using the `oc create secret tls` command.

Procedure

1. Edit the cluster **Ingress** configuration:

```
$ oc edit ingress.config.openshift.io cluster
```

2. Set the custom hostname and optionally the serving certificate and key:

```
apiVersion: config.openshift.io/v1
kind: Ingress
metadata:
  name: cluster
spec:
  componentRoutes:
    - name: downloads
      namespace: openshift-console
      hostname: <custom_hostname>  
      servingCertKeyPairSecret:
        name: <secret_name>  
```

1. The custom hostname.
2. Reference to a secret in the `openshift-config` namespace that contains a TLS certificate (`tls.crt`) and key (`tls.key`). This is required if the domain for the custom hostname suffix does not match the cluster domain suffix. The secret is optional if the suffix matches.

3. Save the file to apply the changes.

6.4. CUSTOMIZING THE LOGIN PAGE

Create Terms of Service information with custom login pages. Custom login pages can also be helpful if you use a third-party login provider, such as GitHub or Google, to show users a branded page that they trust and expect before being redirected to the authentication provider. You can also render custom error pages during the authentication process.

NOTE

Customizing the error template is limited to identity providers (IDPs) that use redirects, such as request header and OIDC-based IDPs. It does not have an effect on IDPs that use direct password authentication, such as LDAP and htpasswd.

Prerequisites

- You must have administrator privileges.

Procedure

1. Run the following commands to create templates you can modify:
Create the secrets:

```bash
$ oc adm create-login-template > login.html
$ oc adm create-provider-selection-template > providers.html
$ oc adm create-error-template > errors.html
```

2. Create the secrets:

```bash
$ oc create secret generic login-template --from-file=login.html -n openshift-config
$ oc create secret generic providers-template --from-file=providers.html -n openshift-config
$ oc create secret generic error-template --from-file=errors.html -n openshift-config
```

3. Run:

```bash
$ oc edit oauths cluster
```

4. Update the specification:

```yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
# ...
spec:
templates:
  error:
    name: error-template
  login:
    name: login-template
  providerSelection:
    name: providers-template
```

Run `oc explain oauths.spec.templates` to understand the options.

### 6.5. DEFINING A TEMPLATE FOR AN EXTERNAL LOG LINK

If you are connected to a service that helps you browse your logs, but you need to generate URLs in a particular way, then you can define a template for your link.

**Prerequisites**

- You must have administrator privileges.

**Procedure**

1. From Administration → Custom Resource Definitions, click on ConsoleExternalLogLink.
2. Select Instances tab
3. Click Create Console External Log Link and edit the file:
6.6. CREATING CUSTOM NOTIFICATION BANNERS

Prerequisites

- You must have administrator privileges.

Procedure

1. From Administration → Custom Resource Definitions, click on ConsoleNotification.
2. Select Instances tab
3. Click Create Console Notification and edit the file:

```yaml
apiVersion: console.openshift.io/v1
kind: ConsoleNotification
metadata:
  name: example
spec:
  text: This is an example notification message with an optional link.
  location: BannerTop
  link:
    href: 'https://www.example.com'
    text: Optional link text
    color: '#fff'
    backgroundColor: '#0088ce'
```

Valid location settings are BannerTop, BannerBottom, and BannerTopBottom.

4. Click Create to apply your changes.

6.7. CUSTOMIZING CLI DOWNLOADS

You can configure links for downloading the CLI with custom link text and URLs, which can point directly to file packages or to an external page that provides the packages.

Prerequisites

- You must have administrator privileges.

Procedure
1. Navigate to Administration → Custom Resource Definitions

2. Select ConsoleCLIDownload from the list of Custom Resource Definitions (CRDs).

3. Click the YAML tab, and then make your edits:

apiVersion: console.openshift.io/v1
kind: ConsoleCLIDownload
metadata:
  name: example-cli-download-links
spec:
  description: |
    This is an example of download links
displayName: example
links:
- href: 'https://www.example.com/public/example.tar'
  text: example for linux
- href: 'https://www.example.com/public/example.mac.zip'
  text: example for mac
- href: 'https://www.example.com/public/example.win.zip'
  text: example for windows

4. Click the Save button.

6.8. ADDING YAML EXAMPLES TO KUBERNETES RESOURCES

You can dynamically add YAML examples to any Kubernetes resources at any time.

Prerequisites

- You must have cluster administrator privileges.

Procedure

1. From Administration → Custom Resource Definitions, click on ConsoleYAMLSample.

2. Click YAML and edit the file:

apiVersion: console.openshift.io/v1
kind: ConsoleYAMLSample
metadata:
  name: example
spec:
targetResource:
  apiVersion: batch/v1
  kind: Job
title: Example Job
description: An example Job YAML sample
yaml: |
  apiVersion: batch/v1
  kind: Job
  metadata:
    name: countdown
  spec:
    template:
6.9. CUSTOMIZING USER PERSPECTIVES

The OpenShift Container Platform web console provides two perspectives by default, Administrator and Developer. You might have more perspectives available depending on installed console plugins. As a cluster administrator, you can show or hide a perspective for all users or for a specific user role. Customizing perspectives ensures that users can view only the perspectives that are applicable to their role and tasks. For example, you can hide the Administrator perspective from unprivileged users so that they cannot manage cluster resources, users, and projects. Similarly, you can show the Developer perspective to users with the developer role so that they can create, deploy, and monitor applications.

You can also customize the perspective visibility for users based on role-based access control (RBAC). For example, if you customize a perspective for monitoring purposes, which requires specific permissions, you can define that the perspective is visible only to users with required permissions.

Each perspective includes the following mandatory parameters, which you can edit in the YAML view:

- **id**: Defines the ID of the perspective to show or hide
- **visibility**: Defines the state of the perspective along with access review checks, if needed
- **state**: Defines whether the perspective is enabled, disabled, or needs an access review check

**NOTE**

By default, all perspectives are enabled. When you customize the user perspective, your changes are applicable to the entire cluster.

6.9.1. Customizing a perspective using YAML view

**Prerequisites**

- You must have administrator privileges.

**Procedure**

1. In the Administrator perspective, navigate to Administration → Cluster Settings.
2. Select the Configuration tab and click the Console (operator.openshift.io) resource.

```yaml
metadata:
  name: countdown
spec:
  containers:
    - name: counter
      image: centos:7
      command:
        - "bin/bash"
        - "-c"
        - "for i in 9 8 7 6 5 4 3 2 1 ; do echo $i ; done"
  restartPolicy: Never
```

Use `spec.snippet` to indicate that the YAML sample is not the full YAML resource definition, but a fragment that can be inserted into the existing YAML document at the user’s cursor.

3. Click Save.
3. Click the YAML tab and make your customization:

   a. To enable or disable a perspective, insert the snippet for **Add user perspectives** and edit the YAML code as needed:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Console
   metadata:
     name: cluster
   spec:
     customization:
       perspectives:
       - id: admin
         visibility:
           state: Enabled
       - id: dev
         visibility:
           state: Enabled
   ```

   b. To hide a perspective based on RBAC permissions, insert the snippet for **Hide user perspectives** and edit the YAML code as needed:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Console
   metadata:
     name: cluster
   spec:
     customization:
       perspectives:
       - id: admin
         requiresAccessReview:
           - group: rbac.authorization.k8s.io
             resource: clusterroles
             verb: list
       - id: dev
         state: Enabled
   ```

   c. To customize a perspective based on your needs, create your own YAML snippet:

   ```yaml
   apiVersion: operator.openshift.io/v1
   kind: Console
   metadata:
     name: cluster
   spec:
     customization:
       perspectives:
       - id: admin
         visibility:
           state: AccessReview
         accessReview:
           missing:
             - resource: deployment
             verb: list
           required:
             - resource: namespaces
   ```
6.9.2. Customizing a perspective using form view

**Prerequisites**
- You must have administrator privileges.

**Procedure**
1. In the **Administrator** perspective, navigate to **Administration → Cluster Settings**.
2. Select the **Configuration** tab and click the **Console (operator.openshift.io)** resource.
3. Click **Actions → Customize** on the right side of the page.
4. In the **General** settings, customize the perspective by selecting one of the following options from the dropdown list:
   - **Enabled**: Enables the perspective for all users
   - **Only visible for privileged users**: Enables the perspective for users who can list all namespaces
   - **Only visible for unprivileged users**: Enables the perspective for users who cannot list all namespaces
   - **Disabled**: Disables the perspective for all users

   A notification opens to confirm that your changes are saved.

**NOTE**
When you customize the user perspective, your changes are automatically saved and take effect after a browser refresh.

6.10. DEVELOPER CATALOG AND SUB-CATALOG CUSTOMIZATION
As a cluster administrator, you have the ability to organize and manage the Developer catalog or its sub-
catalogs. You can enable or disable the sub-catalog types or disable the entire developer catalog.

The `developerCatalog.types` object includes the following parameters that you must define in a
snippet to use them in the YAML view:

- **state**: Defines if a list of developer catalog types should be enabled or disabled.
- **enabled**: Defines a list of developer catalog types (sub-catalogs) that are visible to users.
- **disabled**: Defines a list of developer catalog types (sub-catalogs) that are not visible to users.

You can enable or disable the following developer catalog types (sub-catalogs) using the YAML view or
the form view.

- Builder Images
- Templates
- Devfiles
- Samples
- Helm Charts
- Event Sources
- Event Sinks
- Operator Backed

### 6.10.1. Customizing a developer catalog or its sub-catalogs using the YAML view

You can customize a developer catalog by editing the YAML content in the YAML view.

#### Prerequisites

- An OpenShift web console session with cluster administrator privileges.

#### Procedure

1. In the Administrator perspective of the web console, navigate to Administration → Cluster
   Settings.
2. Select the Configuration tab, click the Console (`operator.openshift.io`) resource and view the
   Details page.
3. Click the YAML tab to open the editor and edit the YAML content as needed.
   For example, to disable a developer catalog type, insert the following snippet that defines a list
   of disabled developer catalog resources:

```yaml
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
```
6.10.2. Customizing a developer catalog or its sub-catalogs using the form view

You can customize a developer catalog by using the form view in the Web Console.

Prerequisites

- An OpenShift web console session with cluster administrator privileges.
- The Developer perspective is enabled.

Procedure

1. In the Administrator perspective, navigate to Administration → Cluster Settings.
2. Select the Configuration tab and click the Console (operator.openshift.io) resource.
3. Click Actions → Customize.
4. Enable or disable items in the Pre-pinned navigation items, Add page, and Developer Catalog sections.

Verification

After you have customized the developer catalog, your changes are automatically saved in the system and take effect in the browser after a refresh.

NOTE

As an administrator, you can define the navigation items that appear by default for all users. You can also reorder the navigation items.
TIP

You can use a similar procedure to customize Web UI items such as Quick starts, Cluster roles, and Actions.

6.10.2.1. Example YAML file changes

You can dynamically add the following snippets in the YAML editor for customizing a developer catalog.

Use the following snippet to display all the sub-catalogs by setting the state type to **Enabled**.

```yaml
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
spec:
customization:
developerCatalog:
categories:
types:
state: Enabled
```

Use the following snippet to disable all sub-catalogs by setting the state type to **Disabled**:

```yaml
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
spec:
customization:
developerCatalog:
categories:
types:
state: Disabled
```

Use the following snippet when a cluster administrator defines a list of sub-catalogs, which are enabled in the Web Console.

```yaml
apiVersion: operator.openshift.io/v1
kind: Console
metadata:
  name: cluster
...
spec:
customization:
developerCatalog:
categories:
types:
state: Enabled
enabled:
- BuilderImage
```
- Devfile
- HelmChart
- ...
CHAPTER 7. DYNAMIC PLUGINS

7.1. OVERVIEW OF DYNAMIC PLUGINS

7.1.1. About dynamic plugins

A dynamic plugin allows you to add custom pages and other extensions to your interface at runtime. The `ConsolePlugin` custom resource registers plugins with the console, and a cluster administrator enables plugins in the `console-operator` configuration.

7.1.2. Key features

A dynamic plugin allows you to make the following customizations to the OpenShift Container Platform experience:

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

7.1.3. 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:

```javascript
const 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.

**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.
- 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.

### 7.2. GETTING STARTED WITH DYNAMIC PLUGINS

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

#### 7.2.1. Dynamic plugin development

You can run the plugin using a local development environment. The OpenShift Container Platform 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.

   ```bash
   $ yarn install
   ```

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

   ```bash
   $ yarn run start
   ```

3. In another terminal window, login to the OpenShift Container Platform through the CLI.

   ```bash
   $ oc login
   ```

4. Run the OpenShift Container Platform web console in a container connected to the cluster you have logged into by running the following command:

   ```bash
   $ 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.

### 7.3. DEPLOY YOUR PLUGIN ON A CLUSTER
You can deploy the plugin to a OpenShift Container Platform cluster.

### 7.3.1. 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-repository/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
   ```

### 7.3.2. 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.

   ```yaml
   plugin:
     name: ""
     description: ""
     image: ""
     imagePullPolicy: IfNotPresent
     replicas: 2
   ```
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.
7.3.3. 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.

7.3.4. Additional resources

- **Understanding Helm**

7.4. DYNAMIC PLUGIN EXAMPLE

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

7.4.1. Adding a tab to the pods page

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

**NOTE**

The OpenShift Container Platform 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:

```json
"consolePlugin": {
  "name": "my-plugin",  
  "version": "0.0.1",  
  "displayName": "My Plugin",  
  "description": "Enjoy this shiny, new console plugin!",
  "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:

```json
{
  "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:

```json
"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:

```jsx
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:

```bash
$ 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.

### 7.5. 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.

#### 7.5.1. Dynamic plugin extension types

**console.action/filter**

**ActionFilter** can be used to filter an action.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextId</td>
<td>string</td>
<td>no</td>
<td>The context ID helps to narrow the scope of contributed actions to a particular area of the application. Examples include <strong>topology</strong> and <strong>helm</strong>.</td>
</tr>
</tbody>
</table>
**filter**

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>ID used to identify the action section.</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>yes</td>
<td>The label to display in the UI. Required for submenus.</td>
</tr>
<tr>
<td>submenu</td>
<td>boolean</td>
<td>yes</td>
<td>Whether this group should be displayed as submenu.</td>
</tr>
<tr>
<td>insertBefore</td>
<td>string</td>
<td>string[]</td>
<td>yes</td>
</tr>
<tr>
<td>insertAfter</td>
<td>string</td>
<td>string[]</td>
<td>yes</td>
</tr>
</tbody>
</table>

---

**console.action/provider**

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

**ResourceActionProvider** contributes a hook that returns list of actions for specific resource model.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sKindVersionModel</td>
<td>no</td>
<td>The model for which this provider provides actions for.</td>
</tr>
<tr>
<td>provider</td>
<td>CodeRef&lt;Extension Hook&lt;Action[], any&gt;&gt;</td>
<td>no</td>
<td>A react hook which returns actions for the given resource model</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alert</td>
<td>string</td>
<td>no</td>
<td>Alert name as defined by <code>alert.rule.name</code> property</td>
</tr>
<tr>
<td>text</td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>action</td>
<td>CodeRef&lt;(alert: any) ⇒ void&gt;</td>
<td>no</td>
<td>Function to perform side effect</td>
</tr>
</tbody>
</table>

### console.catalog/item-filter

This extension can be used for plugins to contribute a handler that can filter specific catalog items. For more information, please refer to the documentation on [CHAPTER 7. DYNAMIC PLUGINS](#).
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalogId</td>
<td>string</td>
<td>string[]</td>
<td>no</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>filter</td>
<td>CodeRef&lt;item: CatalogItem&gt; ⇒ boolean&gt;</td>
<td>no</td>
<td>Filters items of a specific type. Value is a function that takes CatalogItem[] and returns a subset based on the filter criteria.</td>
</tr>
</tbody>
</table>

**console.catalog/item-metadata**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalogId</td>
<td>string</td>
<td>string[]</td>
<td>no</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td></td>
<td>no</td>
</tr>
<tr>
<td>provider</td>
<td>CodeRef&lt;Extension Hook&lt;CatalogItemMetadataProviderFunction, CatalogExtensionHookOptions&gt;&gt;</td>
<td>no</td>
<td>A hook which returns a function that will be used to provide metadata to catalog items of a specific type.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalogId</td>
<td>string</td>
<td>string[]</td>
<td>no</td>
</tr>
<tr>
<td>Name</td>
<td>Value Type</td>
<td>Optional</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td>no</td>
<td>Type ID for the catalog item type.</td>
</tr>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>Title for the catalog item.</td>
</tr>
<tr>
<td>provider</td>
<td>CodeRef&lt;ExtensionHook&lt;CatalogItem&lt;any&gt;[]&gt;, CatalogExtensionHookOptions&gt;</td>
<td>no</td>
<td>Fetch items and normalize it for the catalog. Value is a react effect hook.</td>
</tr>
<tr>
<td>priority</td>
<td>number</td>
<td>yes</td>
<td>Priority for this provider. Defaults to 0. Higher priority providers may override catalog items provided by other providers.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
<td>no</td>
<td>Type for the catalog item.</td>
</tr>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>Title for the catalog item.</td>
</tr>
<tr>
<td>catalogDescription</td>
<td>string</td>
<td>yes</td>
<td>Description for the type specific catalog.</td>
</tr>
<tr>
<td>typeDescription</td>
<td>string</td>
<td>yes</td>
<td>Description for the catalog item type.</td>
</tr>
<tr>
<td>filters</td>
<td>CatalogItemAttribute[]</td>
<td>yes</td>
<td>Custom filters specific to the catalog item.</td>
</tr>
<tr>
<td>groupings</td>
<td>CatalogItemAttribute[]</td>
<td>yes</td>
<td>Custom groupings specific to the catalog item.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
<td>no</td>
<td>Type for the catalog item.</td>
</tr>
<tr>
<td>filters</td>
<td>CatalogItemAttribute</td>
<td>yes</td>
<td>Custom filters specific to the catalog item.</td>
</tr>
<tr>
<td>groupings</td>
<td>CatalogItemAttribute</td>
<td>yes</td>
<td>Custom groupings specific to the catalog item.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;{}&gt;&gt;</td>
<td>no</td>
<td>The component to be rendered.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>The title of the utilization item.</td>
</tr>
<tr>
<td>getUtilizationQueries</td>
<td>CodeRef&lt;GetMultilineQueries&gt;</td>
<td>no</td>
<td>Prometheus utilization query.</td>
</tr>
<tr>
<td>humanize</td>
<td>CodeRef&lt;Humanize&gt;</td>
<td>no</td>
<td>Convert Prometheus data to human-readable form.</td>
</tr>
</tbody>
</table>

**console.cluster-overview/utilization-item**
Adds a new cluster overview utilization item.
<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>The title of the utilization item.</td>
</tr>
<tr>
<td>getUtilizationQuery</td>
<td>CodeRef&lt;GetQuery&gt;</td>
<td>no</td>
<td>Prometheus utilization query.</td>
</tr>
<tr>
<td>humanize</td>
<td>CodeRef&lt;Humanize&gt;</td>
<td>no</td>
<td>Convert Prometheus data to human-readable form.</td>
</tr>
<tr>
<td>getTotalQuery</td>
<td>CodeRef&lt;GetQuery&gt;</td>
<td>yes</td>
<td>Prometheus total query.</td>
</tr>
<tr>
<td>getRequestQuery</td>
<td>CodeRef&lt;GetQuery&gt;</td>
<td>yes</td>
<td>Prometheus request query.</td>
</tr>
<tr>
<td>getLimitQuery</td>
<td>CodeRef&lt;GetQuery&gt;</td>
<td>yes</td>
<td>Prometheus limit query.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>provider</td>
<td>CodeRef&lt;Provider&lt;T&gt;&gt;</td>
<td>no</td>
<td>Context Provider component.</td>
</tr>
<tr>
<td>useValueHook</td>
<td>CodeRef&lt;() =&gt; T&gt;</td>
<td>no</td>
<td>Hook for the Context value.</td>
</tr>
</tbody>
</table>

### console.dashboards/card
Adds a new dashboard card.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab</td>
<td>string</td>
<td>no</td>
<td>The ID of the dashboard tab to which the card will be added.</td>
</tr>
<tr>
<td>position</td>
<td>'LEFT'</td>
<td>'RIGHT'</td>
<td>'MAIN'</td>
</tr>
</tbody>
</table>
### console.dashboards/custom/overview/detail/item
Adds an item to the Details card of Overview Dashboard.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>Details card title</td>
</tr>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;{}&gt;&gt;</td>
<td>no</td>
<td>The value, rendered by the OverviewDetailItem component</td>
</tr>
<tr>
<td>valueClassName</td>
<td>string</td>
<td>yes</td>
<td>Value for a className</td>
</tr>
<tr>
<td>isLoading</td>
<td>CodeRef&lt;() ⇒ boolean&gt;</td>
<td>yes</td>
<td>Function returning the loading state of the component</td>
</tr>
<tr>
<td>error</td>
<td>CodeRef&lt;() ⇒ string&gt;</td>
<td>yes</td>
<td>Function returning errors to be displayed by the component</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>k8sResource</td>
<td>CodeRef&lt;FirehoseResource &amp; { isList: true; }&gt;</td>
<td>no</td>
<td>The utilization item to be replaced.</td>
</tr>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;K8sActivityProps&lt;T&gt;&gt;&gt;</td>
<td>no</td>
<td>The action component.</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>Title of Operators section in the pop-up menu.</td>
</tr>
<tr>
<td>resources</td>
<td>CodeRef&lt;FirehoseResource[]&gt;</td>
<td>no</td>
<td>Kubernetes resources which will be fetched and passed to healthHandler.</td>
</tr>
<tr>
<td>getOperatorsWithStatuses</td>
<td>CodeRef&lt;GetOperatorsWithStatuses&lt;T&gt; &gt;</td>
<td>yes</td>
<td>Resolves status for the Operators.</td>
</tr>
<tr>
<td>operatorRowLoader</td>
<td>CodeRef&lt;React.ComponentType&lt;OperatorRowProps&lt;T&gt;&gt;&gt;</td>
<td>yes</td>
<td>Loader for pop-up row component.</td>
</tr>
<tr>
<td>viewAllLink</td>
<td>string</td>
<td>yes</td>
<td>Links to all resources page. If not provided, then a list page of the first resource from resources prop is used.</td>
</tr>
</tbody>
</table>

### console.dashboards/overview/health/prometheus

Adds a health subsystem to the status card of Overview dashboard where the source of status is Prometheus.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
</table>
### `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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>title</code></td>
<td><code>string</code></td>
<td>no</td>
<td>The display name of the subsystem.</td>
</tr>
<tr>
<td><code>resources</code></td>
<td><code>CodeRef&lt;WatchK8sResources&lt;T&gt;&gt;</code></td>
<td>no</td>
<td>Kubernetes resources that will be fetched and passed to <code>healthHandler</code>.</td>
</tr>
<tr>
<td><code>healthHandler</code></td>
<td><code>CodeRef&lt;ResourceHealthHandler&lt;T&gt;&gt;</code></td>
<td>no</td>
<td>Resolve the subsystem’s health.</td>
</tr>
</tbody>
</table>

**Name**

- **title**: The display name of the subsystem.
- **queries**: The Prometheus queries.
- **healthHandler**: Resolve the subsystem’s health.
- **additionalResource**: Additional resource which will be fetched and passed to `healthHandler`.
- **popupComponent**: 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**: The title of the popover.
- **disallowedControlPlaneTopology**: Control plane topology for which the subsystem should be hidden.

**Value Type**

- `string`:必填项
- `string[]`:可选项

**Optional**

- no:必填项
- yes:可选项

---

**Description**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>title</code></td>
<td><code>string</code></td>
<td>no</td>
<td>The display name of the subsystem.</td>
</tr>
<tr>
<td><code>resources</code></td>
<td><code>CodeRef&lt;WatchK8sResources&lt;T&gt;&gt;</code></td>
<td>no</td>
<td>Kubernetes resources that will be fetched and passed to <code>healthHandler</code>.</td>
</tr>
<tr>
<td><code>healthHandler</code></td>
<td><code>CodeRef&lt;ResourceHealthHandler&lt;T&gt;&gt;</code></td>
<td>no</td>
<td>Resolve the subsystem’s health.</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>The display name of the subsystem.</td>
</tr>
<tr>
<td>url</td>
<td>string</td>
<td>no</td>
<td>The URL to fetch data from. It will be prefixed with base Kubernetes URL.</td>
</tr>
<tr>
<td>healthHandler</td>
<td>CodeRef&lt;URLHealthHandler&lt;T, K8sResourceCommon</td>
<td>no</td>
<td>Resolve the subsystem’s health.</td>
</tr>
<tr>
<td>additionalResource</td>
<td>CodeRef&lt;FirehoseResource&gt;</td>
<td>yes</td>
<td>Additional resource which will be fetched and passed to healthHandler.</td>
</tr>
<tr>
<td>popupComponent</td>
<td>CodeRef&lt;React.ComponentType&lt;{ healthResult?: T; healthResultError?: any; k8sResult?: FirehoseResult&lt;R&gt;; }&gt;&gt;</td>
<td>yes</td>
<td>Loader for popup content. If defined, a health item will be represented as a link which opens popup with given content.</td>
</tr>
<tr>
<td>popupTitle</td>
<td>string</td>
<td>yes</td>
<td>The title of the popover.</td>
</tr>
</tbody>
</table>

### console.dashboards/overview/inventory/item

Adds a resource tile to the overview inventory card.
<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>CodeRef&lt;T&gt;</td>
<td>no</td>
<td>The model for resource which will be fetched. Used to get the model's label or abbr.</td>
</tr>
<tr>
<td>mapper</td>
<td>CodeRef&lt;StatusGroupMapper&lt;T, R&gt;&gt;</td>
<td>yes</td>
<td>Function which maps various statuses to groups.</td>
</tr>
<tr>
<td>additionalResources</td>
<td>CodeRef&lt;WatchK8s Resources&lt;R&gt;&gt;</td>
<td>yes</td>
<td>Additional resources which will be fetched and passed to the mapper function.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>The ID of the status group.</td>
</tr>
<tr>
<td>icon</td>
<td>CodeRef&lt;React.ReactElement&lt;any, string</td>
<td>React.JSXElementConstructor&lt;any&gt;&gt;</td>
<td>no</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>CodeRef&lt;T&gt;</td>
<td>no</td>
<td>The model for resource which will be fetched. Used to get the model's label or abbr.</td>
</tr>
<tr>
<td>mapper</td>
<td>CodeRef&lt;StatusGroupMapper&lt;T, R&gt;&gt;</td>
<td>yes</td>
<td>Function which maps various statuses to groups.</td>
</tr>
<tr>
<td>additionalResources</td>
<td>CodeRef&lt;WatchK8s Resources&lt;R&gt;&gt;</td>
<td>yes</td>
<td>Additional resources which will be fetched and passed to the mapper function.</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queries</td>
<td>string[]</td>
<td>no</td>
<td>Queries to watch.</td>
</tr>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;PrometheusActivityProps&gt;&gt;</td>
<td>no</td>
<td>The action component.</td>
</tr>
<tr>
<td>isActivity</td>
<td>CodeRef&lt;(results: PrometheusResponse[]) ⇒ boolean&gt;</td>
<td>yes</td>
<td>Function which determines if the given resource represents the action. If not defined, every resource represents activity.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>CodeRef&lt;T&gt;</td>
<td>no</td>
<td>The model for resource which will be fetched. Used to get the model’s label or abbr.</td>
</tr>
<tr>
<td>mapper</td>
<td>CodeRef&lt;StatusGroupMapper&lt;T, R&gt;&gt;</td>
<td>yes</td>
<td>Function which maps various statuses to groups.</td>
</tr>
<tr>
<td>additionalResources</td>
<td>CodeRef&lt;WatchK8sResources&lt;R&gt;&gt;</td>
<td>yes</td>
<td>Additional resources which will be fetched and passed to the mapper function.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>A unique tab identifier, used as tab link href and when adding cards to this tab.</td>
</tr>
</tbody>
</table>
**console.file-upload**
This extension can be used to provide a handler for the file drop action on specific file extensions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileExtensions</td>
<td>string[]</td>
<td>no</td>
<td>Supported file extensions.</td>
</tr>
<tr>
<td>handler</td>
<td>CodeRef&lt;FileUploadHandler&gt;</td>
<td>no</td>
<td>Function which handles the file drop action.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>CodeRef&lt;FeatureFlagHandler&gt;</td>
<td>no</td>
<td>Used to set or unset arbitrary feature flags.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handler</td>
<td>CodeRef&lt;FeatureFlagHandler&gt;</td>
<td>no</td>
<td>Used to set or unset arbitrary feature flags.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>string</td>
<td>no</td>
<td>The name of the flag to set after the CRD is detected.</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>Unique identifier for the cluster config resource instance.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>no</td>
<td>The name of the cluster config resource instance.</td>
</tr>
<tr>
<td>model</td>
<td>ExtensionK8sModel</td>
<td>no</td>
<td>The model which refers to a cluster config resource.</td>
</tr>
<tr>
<td>namespace</td>
<td>string</td>
<td>no</td>
<td>The namespace of the cluster config resource instance.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sGroupModel</td>
<td>no</td>
<td>The model to customize. May specify only a group, or optional version and kind.</td>
</tr>
<tr>
<td>badge</td>
<td>ModelBadge</td>
<td>yes</td>
<td>Whether to consider this model reference as Technology Preview or Developer Preview.</td>
</tr>
<tr>
<td>color</td>
<td>string</td>
<td>yes</td>
<td>The color to associate to this model.</td>
</tr>
</tbody>
</table>
### label
- **Value Type**: string
- **Optional**: yes
- **Description**: Override the label. Requires `kind` be provided.

### labelPlural
- **Value Type**: string
- **Optional**: yes
- **Description**: Override the plural label. Requires `kind` be provided.

### abbr
- **Value Type**: string
- **Optional**: yes
- **Description**: 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong></td>
<td>string</td>
<td>no</td>
<td>A unique identifier for this item.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>string</td>
<td>no</td>
<td>The name of this item.</td>
</tr>
<tr>
<td><strong>href</strong></td>
<td>string</td>
<td>no</td>
<td>The link <code>href</code> value.</td>
</tr>
<tr>
<td><strong>perspective</strong></td>
<td>string</td>
<td>yes</td>
<td>The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.</td>
</tr>
<tr>
<td><strong>section</strong></td>
<td>string</td>
<td>yes</td>
<td>Navigation section to which this item belongs to. If not specified, render this item as a top level link.</td>
</tr>
<tr>
<td><strong>dataAttributes</strong></td>
<td>{ [key: string]: string; }</td>
<td>yes</td>
<td>Adds data attributes to the DOM.</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>A unique identifier for this item.</td>
</tr>
<tr>
<td>model</td>
<td>ExtensionK8sModel</td>
<td>no</td>
<td>The model for which this navigation item links to.</td>
</tr>
<tr>
<td>perspective</td>
<td>string</td>
<td>yes</td>
<td>The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.</td>
</tr>
</tbody>
</table>
### 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 | | Insert this item before the item referenced here. For arrays, the first one found in order is used.

**insertAfter** | string | | 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>id</strong></td>
<td>string</td>
<td>no</td>
<td>A unique identifier for this item.</td>
</tr>
<tr>
<td><strong>model</strong></td>
<td>ExtensionK8sModel</td>
<td>no</td>
<td>The model for which this navigation item links to.</td>
</tr>
</tbody>
</table>
### console.navigation/section

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>A unique identifier for this item.</td>
</tr>
</tbody>
</table>
The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.

Adds data attributes to the DOM.

Insert this item before the item referenced here. For arrays, the first one found in order is used.

Insert this item after the item referenced here. For arrays, the first one found in order is used. `insertBefore` takes precedence.

Name of this section. If not supplied, only a separator will be shown above the section.

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

A unique identifier for this item.

The perspective ID to which this item belongs to. If not specified, contributes to the default perspective.

Navigation section to which this item belongs to. If not specified, render this item as a top level link.
## console.page/resource/details

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sGroup KindModel</td>
<td>no</td>
<td>The model for which this resource page links to.</td>
</tr>
</tbody>
</table>
| 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sGroup KindModel</td>
<td>no</td>
<td>The model for which this resource page links to.</td>
</tr>
</tbody>
</table>
| 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;RouteComponentProps&lt;{}, StaticContext, any&gt;&gt;&gt;</td>
<td>no</td>
<td>The component to be rendered when the route matches.</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>string[]</td>
<td>no</td>
</tr>
<tr>
<td>perspective</td>
<td>string</td>
<td>yes</td>
<td>The perspective to which this page belongs to. If not specified, contributes to all perspectives.</td>
</tr>
<tr>
<td>exact</td>
<td>boolean</td>
<td>yes</td>
<td>When true, will only match if the path matches the location.pathname exactly.</td>
</tr>
</tbody>
</table>

### console.page/route/standalone

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;RouteComponentProps&lt;{}, StaticContext, any&gt;&gt;&gt;</td>
<td>no</td>
<td>The component to be rendered when the route matches.</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>string[]</td>
<td>no</td>
</tr>
<tr>
<td>exact</td>
<td>boolean</td>
<td>yes</td>
<td>When true, will only match if the path matches the location.pathname exactly.</td>
</tr>
</tbody>
</table>
**console.perspective**
This extension contributes a new perspective to the console, which enables customization of the navigation menu.

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

**console.project-overview/inventory-item**
Adds a new inventory item into the Project Overview page.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;{ projectName: string; }&gt;&gt;</td>
<td>no</td>
<td>The component to be rendered.</td>
</tr>
</tbody>
</table>

**console.project-overview/utilization-item**
Adds a new project overview utilization item.
<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>string</td>
<td>no</td>
<td>The title of the utilization item.</td>
</tr>
<tr>
<td>getUtilizationQuery</td>
<td>CodeRef&lt;GetProjectQuery&gt;</td>
<td>no</td>
<td>Prometheus utilization query.</td>
</tr>
<tr>
<td>humanize</td>
<td>CodeRef&lt;Humanize&gt;</td>
<td>no</td>
<td>Convert Prometheus data to human-readable form.</td>
</tr>
<tr>
<td>getTotalQuery</td>
<td>CodeRef&lt;GetProjectQuery&gt;</td>
<td>yes</td>
<td>Prometheus total query.</td>
</tr>
<tr>
<td>getRequestQuery</td>
<td>CodeRef&lt;GetProjectQuery&gt;</td>
<td>yes</td>
<td>Prometheus request query.</td>
</tr>
<tr>
<td>getLimitQuery</td>
<td>CodeRef&lt;GetProjectQuery&gt;</td>
<td>yes</td>
<td>Prometheus limit query.</td>
</tr>
<tr>
<td>TopConsumerPopover</td>
<td>CodeRef&lt;React.ComponentType&lt;TopConsumerPopoverProps&gt;&gt;</td>
<td>yes</td>
<td>Shows the top consumer popover instead of plain value.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alert</td>
<td>CodeRef&lt;React.ComponentType&lt;{ pvc: K8sResourceCommon; }&gt;&gt;</td>
<td>no</td>
<td>The alert component.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>string</td>
<td>no</td>
<td>Label for the create prop action.</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>no</td>
<td>Path for the create prop action.</td>
</tr>
</tbody>
</table>
console.pvc/delete
This extension allows hooking into deleting PVC resources. It can provide an alert with additional information and custom PVC delete logic.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>predicate</td>
<td>CodeRef&lt;(pvc: K8sResourceComm on) ⇒ boolean&gt;</td>
<td>no</td>
<td>Predicate that tells whether to use the extension or not.</td>
</tr>
<tr>
<td>onPVCKill</td>
<td>CodeRef&lt;(pvc: K8sResourceComm on) ⇒ Promise&lt;void&gt;&gt;</td>
<td>no</td>
<td>Method for the PVC delete operation.</td>
</tr>
<tr>
<td>alert</td>
<td>CodeRef&lt;React.ComponentType&lt;{ pvc: K8sResourceComm on; }&gt;&gt;</td>
<td>no</td>
<td>Alert component to show additional information.</td>
</tr>
</tbody>
</table>

console.pvc/status

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td>number</td>
<td>no</td>
<td>Priority for the status component. A larger value means higher priority.</td>
</tr>
<tr>
<td>status</td>
<td>CodeRef&lt;React.ComponentType&lt;{ pvc: K8sResourceComm on; }&gt;&gt;</td>
<td>no</td>
<td>The status component.</td>
</tr>
<tr>
<td>predicate</td>
<td>CodeRef&lt;(pvc: K8sResourceComm on) ⇒ boolean&gt;</td>
<td>no</td>
<td>Predicate that tells whether to render the status component or not.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope</td>
<td>string</td>
<td>no</td>
<td>The key to represent the reducer-managed substate within the Redux state object.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sModel</td>
<td>no</td>
<td>The model for which this create resource page will be rendered</td>
</tr>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;Create ResourceComponentProps&gt;&gt;</td>
<td>no</td>
<td>The component to be rendered when the model matches</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSI</td>
<td>ProvisionerDetails</td>
<td>yes</td>
<td>Container Storage Interface provisioner type</td>
</tr>
<tr>
<td>OTHERS</td>
<td>ProvisionerDetails</td>
<td>yes</td>
<td>Other provisioner type</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>no</td>
<td>Displayed name of the provider.</td>
</tr>
</tbody>
</table>
## console.tab

Adds a tab to a horizontal nav matching the `contextId`.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextId</td>
<td>string</td>
<td>no</td>
<td>Context ID assigned to the horizontal nav in which the tab will be injected. Possible values: <code>dev-console-observe</code></td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>no</td>
<td>The display label of the tab</td>
</tr>
<tr>
<td>href</td>
<td>string</td>
<td>no</td>
<td>The <code>href</code> appended to the existing URL</td>
</tr>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;Partial&lt;RouteComponentProps&lt;{}, StaticContext, any&gt;&gt;&gt;</td>
<td>no</td>
<td>Tab content component.</td>
</tr>
</tbody>
</table>

## console.tab/horizontalNav

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sKindVersionModel</td>
<td>no</td>
<td>The model for which this provider show tab.</td>
</tr>
<tr>
<td>page</td>
<td>{ name: string; href: string; }</td>
<td>no</td>
<td>The page to be show in horizontal tab. It takes tab name as name and href of the tab.</td>
</tr>
</tbody>
</table>
## Name | Value Type | Optional | Description
--- | --- | --- | ---
**component** | CodeRef<React.ComponentType<PageComponentProps<K8s ResourceCommon>> | 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>listener</strong></td>
<td>CodeRef&lt;TelemetryEventListener&gt;</td>
<td>no</td>
<td>Listen for telemetry events</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>adapt</strong></td>
<td>CodeRef&lt;(element: GraphElement) ⇒ AdapterDataType&lt;BuildConfigData&gt;</td>
<td>no</td>
<td>Adapter to adapt element to data that can be used by Build component.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>adapt</strong></td>
<td>CodeRef&lt;(element: GraphElement) ⇒ NetworkAdapterType</td>
<td>no</td>
<td>Adapter to adapt element to data that can be used by Networking component.</td>
</tr>
</tbody>
</table>

### console.topology/adapter/pod
**PodAdapter** contributes an adapter to adapt element to data that can be used by the Pod component.
### Chapter 7: Dynamic Plugins

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCreateConnector</td>
<td>CodeRef&lt;CreateConnectionGetter&gt;</td>
<td>no</td>
<td>Getter for the create connector function.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>Unique ID for the factory.</td>
</tr>
<tr>
<td>priority</td>
<td>number</td>
<td>no</td>
<td>Priority for the factory</td>
</tr>
<tr>
<td>resources</td>
<td>WatchK8sResources Generic</td>
<td>yes</td>
<td>Resources to be fetched from useK8sWatchResources hook.</td>
</tr>
<tr>
<td>workloadKeys</td>
<td>string[]</td>
<td>yes</td>
<td>Keys in resources containing workloads.</td>
</tr>
<tr>
<td>getDataModel</td>
<td>CodeRef&lt;TopologyDataModelGetter&gt;</td>
<td>yes</td>
<td>Getter for the data model factory.</td>
</tr>
</tbody>
</table>
### isResourceDepicted
- **Value Type**: `CodeRef<TopologyDataModelDepicted>`
- **Optional**: yes
- **Description**: Getter for function to determine if a resource is depicted by this model factory.

### getDataModelReconciler
- **Value Type**: `CodeRef<TopologyDataModelReconciler>`
- **Optional**: yes
- **Description**: Getter for function to reconcile data model after all extensions’ models have loaded.

---

**console.topology/decorator/provider**

Topology Decorator Provider Extension

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>ID for topology decorator specific to the extension</td>
</tr>
<tr>
<td>priority</td>
<td>number</td>
<td>no</td>
<td>Priority for topology decorator specific to the extension</td>
</tr>
<tr>
<td>quadrant</td>
<td><code>TopologyQuadrant</code></td>
<td>no</td>
<td>Quadrant for topology decorator specific to the extension</td>
</tr>
<tr>
<td>decorator</td>
<td><code>CodeRef&lt;TopologyDecoratorGetter&gt;</code></td>
<td>no</td>
<td>Decorator specific to the extension</td>
</tr>
</tbody>
</table>

---

**console.topology/details/resource-alert**

DetailsResourceAlert contributes an alert for specific topology context or graph element.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>The ID of this alert. Used to save state if the alert should not be shown after dismissed.</td>
</tr>
<tr>
<td>contentProvider</td>
<td>`CodeRef&lt;element: GraphElement) ⇒ DetailsResourceAlertContent</td>
<td>no</td>
<td>Hook to return the contents of the alert.</td>
</tr>
</tbody>
</table>
**console.topology/details/resource-link**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>CodeRef&lt;(element: GraphElement) ⇒ React.Component</td>
<td>no</td>
<td>Return the resource link if provided, otherwise undefined. Use the ResourceIcon and ResourceLink properties for styles.</td>
</tr>
<tr>
<td>priority</td>
<td>number</td>
<td>yes</td>
<td>A higher priority factory will get the first chance to create the link.</td>
</tr>
</tbody>
</table>

**console.topology/details/tab**

DetailsTab contributes a tab for the topology details panel.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>A unique identifier for this details tab.</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>no</td>
<td>The tab label to display in the UI.</td>
</tr>
<tr>
<td>insertBefore</td>
<td>string</td>
<td>yes</td>
<td>Insert this item before the item referenced here. For arrays, the first one found in order is used.</td>
</tr>
<tr>
<td>insertAfter</td>
<td>string</td>
<td>yes</td>
<td>Insert this item after the item referenced here. For arrays, the first one found in order is used. The insertBefore value takes precedence.</td>
</tr>
</tbody>
</table>

**console.topology/details/tab-section**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>A unique identifier for this details tab section.</td>
</tr>
</tbody>
</table>
### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getTopologyFilters</td>
<td>CodeRef&lt;() ⇒ TopologyDisplayOpt ion[]&gt;</td>
<td>no</td>
<td>Getter for topology filters specific to the extension</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applyDisplayOptions</td>
<td>CodeRef&lt;TopologyA pplyDisplayOptions&gt;</td>
<td>no</td>
<td>Function to apply filters to the model</td>
</tr>
</tbody>
</table>

### console.topology/relationship/provider
Topology relationship provider connector extension
<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>provides</td>
<td>CodeRef&lt;RelationshipProviderProvides&gt;</td>
<td>no</td>
<td>Use to determine if a connection can be created between the source and target node</td>
</tr>
<tr>
<td>tooltip</td>
<td>string</td>
<td>no</td>
<td>Tooltip to show when connector operation is hovering over the drop target, for example, “Create a Visual Connector”</td>
</tr>
<tr>
<td>create</td>
<td>CodeRef&lt;RelationshipProviderCreate&gt;</td>
<td>no</td>
<td>Callback to execute when connector is drop over target node to create a connection</td>
</tr>
<tr>
<td>priority</td>
<td>number</td>
<td>no</td>
<td>Priority for relationship, higher will be preferred in case of multiple</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>ID used to identify the user preference group.</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>no</td>
<td>The label of the user preference group</td>
</tr>
<tr>
<td>insertBefore</td>
<td>string</td>
<td>yes</td>
<td>ID of user preference group before which this group should be placed</td>
</tr>
<tr>
<td>insertAfter</td>
<td>string</td>
<td>yes</td>
<td>ID of user preference group after which this group should be placed</td>
</tr>
</tbody>
</table>

**console.user-preference/item**
This extension can be used to add an item to the user preferences group on the console user preferences page.
### User Preference Item Schema

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

**console.yaml-template**

YAML templates for editing resources via the yaml editor.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sModel</td>
<td>no</td>
<td>Model associated with the template.</td>
</tr>
<tr>
<td>template</td>
<td>CodeRef&lt;string&gt;</td>
<td>no</td>
<td>The YAML template.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>no</td>
<td>The name of the template. Use the name default to mark this as the default template.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>ID used to identify the action.</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>no</td>
<td>The label of the action.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>no</td>
<td>The description of the action.</td>
</tr>
<tr>
<td>href</td>
<td>string</td>
<td>no</td>
<td>The href to navigate to.</td>
</tr>
<tr>
<td>groupId</td>
<td>string</td>
<td>yes</td>
<td>IDs used to identify the action groups the action would belong to.</td>
</tr>
<tr>
<td>icon</td>
<td>CodeRef&lt;React.ReactNode&gt;</td>
<td>yes</td>
<td>The perspective display icon.</td>
</tr>
<tr>
<td>accessReview</td>
<td>AccessReviewResourceAttributes[]</td>
<td>yes</td>
<td>Optional access review to control the visibility or enablement of the action.</td>
</tr>
</tbody>
</table>

`dev-console.add/action-group`
This extension allows plugins to contribute 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>no</td>
<td>ID used to identify the action group</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>no</td>
<td>The title of the action group</td>
</tr>
<tr>
<td>insertBefore</td>
<td>string</td>
<td>yes</td>
<td>ID of action group before which this group should be placed</td>
</tr>
<tr>
<td>insertAfter</td>
<td>string</td>
<td>yes</td>
<td>ID of action group after which this group should be placed</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imageStreamName</td>
<td>string</td>
<td>no</td>
<td>Name of the image stream to provide custom environment variables for</td>
</tr>
<tr>
<td>imageStreamTags</td>
<td>string[]</td>
<td>no</td>
<td>List of supported image stream tags</td>
</tr>
<tr>
<td>environments</td>
<td>ImageEnvironment[]</td>
<td>no</td>
<td>List of environment variables</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;{}&gt;&gt;</td>
<td>no</td>
<td>The value, based on the DetailItem component</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value Type</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>ExtensionK8sGroupKindModel</td>
<td>no</td>
<td>The model for which this resource page links to.</td>
</tr>
<tr>
<td>component</td>
<td>CodeRef&lt;React.ComponentType&lt;RouteComponentProps&lt;{}, StaticContext, any&gt;&gt;, any&gt;&gt;</td>
<td>no</td>
<td>The component to be rendered when the route matches.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>no</td>
<td>The name of the tab.</td>
</tr>
<tr>
<td>href</td>
<td>string</td>
<td>yes</td>
<td>The optional href for the tab link. If not provided, the first path is used.</td>
</tr>
<tr>
<td>Name</td>
<td>Value Type</td>
<td>Optional</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>exact</td>
<td>boolean</td>
<td>yes</td>
<td>When true, will only match if the path matches the location.pathname exactly.</td>
</tr>
</tbody>
</table>

**7.5.2. OpenShift Container Platform 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**

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

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

**GreenCheckCircleIcon**
Component for displaying a green check mark circle icon.

**Example**

```jsx```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>(optional) additional class name for the component</td>
</tr>
<tr>
<td>title</td>
<td>(optional) icon title</td>
</tr>
<tr>
<td>size</td>
<td>(optional) icon size: (sm, md, lg, xl)</td>
</tr>
</tbody>
</table>

**RedExclamationCircleIcon**
Component for displaying a red exclamation mark circle icon.

**Example**

```jsx```
YellowExclamationTriangleIcon
Component for displaying a yellow triangle exclamation icon.

Example

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

BluelInfoCircleIcon
Component for displaying a blue info circle icon.

Example

```html
<BluelInfoCircleIcon title="Info" />
```

ErrorStatus
Component for displaying an error status popover.

Example

```html
>ErrorStatus title={errorMsg} />
```
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>(optional) status text</td>
</tr>
<tr>
<td>iconOnly</td>
<td>(optional) if true, only displays icon</td>
</tr>
<tr>
<td>noTooltip</td>
<td>(optional) if true, tooltip won’t be displayed</td>
</tr>
<tr>
<td>className</td>
<td>(optional) additional class name for the component</td>
</tr>
<tr>
<td>popoverTitle</td>
<td>(optional) title for popover</td>
</tr>
</tbody>
</table>

**InfoStatus**  
Component for displaying an information status popover.

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>(optional) status text</td>
</tr>
<tr>
<td>iconOnly</td>
<td>(optional) if true, only displays icon</td>
</tr>
<tr>
<td>noTooltip</td>
<td>(optional) if true, tooltip won’t be displayed</td>
</tr>
<tr>
<td>className</td>
<td>(optional) additional class name for the component</td>
</tr>
<tr>
<td>popoverTitle</td>
<td>(optional) title for popover</td>
</tr>
</tbody>
</table>

**ProgressStatus**  
Component for displaying a progressing status popover.

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>(optional) status text</td>
</tr>
<tr>
<td>iconOnly</td>
<td>(optional) if true, only displays icon</td>
</tr>
<tr>
<td>noTooltip</td>
<td>(optional) if true, tooltip won’t be displayed</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>className</td>
<td>(optional) additional class name for the component</td>
</tr>
<tr>
<td>popoverTitle</td>
<td>(optional) title for popover</td>
</tr>
</tbody>
</table>

**SuccessStatus**  
Component for displaying a success status popover.

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>(optional) status text</td>
</tr>
<tr>
<td>iconOnly</td>
<td>(optional) if true, only displays icon</td>
</tr>
<tr>
<td>noTooltip</td>
<td>(optional) if true, tooltip won’t be displayed</td>
</tr>
<tr>
<td>className</td>
<td>(optional) additional class name for the component</td>
</tr>
<tr>
<td>popoverTitle</td>
<td>(optional) title for popover</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceAttributes</td>
<td>resource attributes for access review</td>
</tr>
<tr>
<td>impersonate</td>
<td>impersonation details</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceAttributes</td>
<td>resource attributes for access review</td>
</tr>
<tr>
<td>impersonate</td>
<td>impersonation details</td>
</tr>
</tbody>
</table>
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

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typeGuards</td>
<td>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</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource</td>
<td>The resource associated with this Navigation, an object of K8sResourceCommon type</td>
</tr>
<tr>
<td>pages</td>
<td>An array of page objects</td>
</tr>
<tr>
<td>match</td>
<td>match object provided by React Router</td>
</tr>
</tbody>
</table>
VirtualizedTable
A component for making virtualized tables.

Example

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

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

TableData
Component for displaying table data within a table row.
Example

```javascript
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>
    </>
  );
};
```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>unique ID for table</td>
</tr>
<tr>
<td>activeColumnIDs</td>
<td>active columns</td>
</tr>
<tr>
<td>className</td>
<td>(optional) option class name for styling</td>
</tr>
</tbody>
</table>

**useActiveColumns**

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

Example

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Which are passed as a key-value map</td>
</tr>
<tr>
<td>{TableColumn[]} options.columns</td>
<td>An array of all available TableColumns</td>
</tr>
<tr>
<td>{boolean} [options.showNamespaceOverride]</td>
<td>(optional) If true, a namespace column will be included, regardless of column management selections</td>
</tr>
</tbody>
</table>
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**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>heading title</td>
</tr>
<tr>
<td>helpText</td>
<td>(optional) help section as react node</td>
</tr>
<tr>
<td>badge</td>
<td>(optional) badge icon as react node</td>
</tr>
</tbody>
</table>

**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**

```jsx
const exampleList: React.FC<MyProps> = () => {
  return (
    <ListPageHeader title="Example Pod List Page"/>
    <ListPageCreate groupVersionKind="Pod">Create Pod</ListPageCreate>
  );
};
```
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupVersionKind</td>
<td>the resource group/version/kind to represent</td>
</tr>
</tbody>
</table>

**ListPageCreateLink**
Component for creating a stylized link.

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>string location where link should direct</td>
</tr>
<tr>
<td>createAccessReview</td>
<td>(optional) object with namespace and kind used to determine access</td>
</tr>
<tr>
<td>children</td>
<td>(optional) children for the component</td>
</tr>
</tbody>
</table>

**ListPageCreateButton**
Component for creating button.

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createAccessReview</td>
<td>(optional) object with namespace and kind used to determine access</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>pfButtonProps</td>
<td>(optional) Patternfly Button props</td>
</tr>
</tbody>
</table>

**ListPageCreateDropdown**
Component for creating a dropdown wrapped with permissions check.

**Example**

```javascript
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>
    </>
  );
};
```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>items</td>
<td>key:ReactNode pairs of items to display in dropdown component</td>
</tr>
<tr>
<td>onClick</td>
<td>callback function for click on dropdown items</td>
</tr>
<tr>
<td>createAccessReview</td>
<td>(optional) object with namespace and kind used to determine access</td>
</tr>
<tr>
<td>children</td>
<td>(optional) children for the dropdown toggle</td>
</tr>
</tbody>
</table>

**ListPageFilter**
Component that generates filter for list page.

**Example**

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

```javascript
// 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 (  
```
### Parameter Name

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>An array of data points</td>
</tr>
<tr>
<td>rowFilters</td>
<td>(optional) An array of RowFilter elements that define the available filter options</td>
</tr>
<tr>
<td>staticFilters</td>
<td>(optional) An array of FilterValue elements that are statically applied to the data</td>
</tr>
</tbody>
</table>

### ResourceLink

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

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>(optional) the kind of resource i.e. Pod, Deployment, Namespace</td>
</tr>
<tr>
<td>groupVersionKind</td>
<td>(optional) object with group, version, and kind</td>
</tr>
<tr>
<td>className</td>
<td>(optional) class style for component</td>
</tr>
<tr>
<td>displayName</td>
<td>(optional) display name for component, overwrites the resource name if set</td>
</tr>
<tr>
<td>inline</td>
<td>(optional) flag to create icon badge and name inline with children</td>
</tr>
<tr>
<td>linkTo</td>
<td>(optional) flag to create a Link object - defaults to true</td>
</tr>
<tr>
<td>name</td>
<td>(optional) name of resource</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>namespace</td>
<td>(optional) specific namespace for the kind resource to link to</td>
</tr>
<tr>
<td>hideIcon</td>
<td>(optional) flag to hide the icon badge</td>
</tr>
<tr>
<td>title</td>
<td>(optional) title for the link object (not displayed)</td>
</tr>
<tr>
<td>dataTest</td>
<td>(optional) identifier for testing</td>
</tr>
<tr>
<td>onClick</td>
<td>(optional) callback function for when component is clicked</td>
</tr>
<tr>
<td>truncate</td>
<td>(optional) flag to truncate the link if too long</td>
</tr>
</tbody>
</table>

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

**Example**
```js
<ResourceIcon kind="Pod"/>
```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kind</td>
<td>(optional) the kind of resource i.e. Pod, Deployment, Namespace</td>
</tr>
<tr>
<td>groupVersionKind</td>
<td>(optional) object with group, version, and kind</td>
</tr>
<tr>
<td>className</td>
<td>(optional) class style for component</td>
</tr>
</tbody>
</table>

**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**
```js
const Component: React.FC = () => {
    const [model, inFlight] = useK8sModel({ group: 'app'; version: 'v1'; kind: 'Deployment' });
    return ...
}
```
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupVersionKind</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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**

```jsx
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**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initResource</td>
<td>options needed to watch for resource.</td>
</tr>
</tbody>
</table>

**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 initResources and value has three properties data, loaded and error.

**Example**

```jsx
const Component: React.FC = () => {
  const watchResources = {
    'deployment': {...},
    'pod': {...}
  ...
```
const {deployment, pod} = useK8sWatchResources(watchResources)
return ...

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initResources</td>
<td>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.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL to fetch</td>
</tr>
<tr>
<td>options</td>
<td>The options to pass to fetch</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout in milliseconds</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL to fetch</td>
</tr>
<tr>
<td>method</td>
<td>The HTTP method to use. Defaults to GET</td>
</tr>
<tr>
<td>options</td>
<td>The options to pass to fetch</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout in milliseconds</td>
</tr>
<tr>
<td>cluster</td>
<td>The name of the cluster to make the request to. Defaults to the active cluster the user has selected</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The URL to fetch</td>
</tr>
<tr>
<td>options</td>
<td>The options to pass to fetch</td>
</tr>
<tr>
<td>timeout</td>
<td>The timeout in milliseconds</td>
</tr>
<tr>
<td>cluster</td>
<td>The name of the cluster to make the request to. Defaults to the active cluster the user has selected</td>
</tr>
</tbody>
</table>

**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 request headers, based on redux state.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetCluster</td>
<td>Override the current active cluster with the provided targetCluster</td>
</tr>
</tbody>
</table>

**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 provided else it returns all the resources matching the model. In case of failure, the promise gets rejected with HTTP error response.

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

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

**k8sUpdateResource**

It updates the entire resource in the cluster, based on provided options. When a client needs to replace an existing resource entirely, they can use k8sUpdate. Alternatively, they 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.

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

**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.

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

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Which are passed as key-value pair in the map.</td>
</tr>
<tr>
<td>options.model</td>
<td>k8s model</td>
</tr>
<tr>
<td>options.resource</td>
<td>The resource to be deleted.</td>
</tr>
<tr>
<td>options.path</td>
<td>Appends as subpath if provided</td>
</tr>
<tr>
<td>options.queryParams</td>
<td>The query parameters to be included in the URL.</td>
</tr>
<tr>
<td>options.requestInit</td>
<td>The fetch init object to use. This can have request headers, method, redirect, etc. See Interface RequestInit for more.</td>
</tr>
<tr>
<td>options.json</td>
<td>Can control garbage collection of resources explicitly if provided else will default to model’s “propagationPolicy”.</td>
</tr>
</tbody>
</table>
**k8sListResource**
Lists the resources as an array in the cluster, based on provided options. It returns a promise that resolves to the response.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Which are passed as key-value pairs in the map</td>
</tr>
<tr>
<td>options.model</td>
<td>k8s model</td>
</tr>
<tr>
<td>options.queryParams</td>
<td>The query parameters to be included in the URL and can pass label selector’s as well with key &quot;labelSelector&quot;.</td>
</tr>
<tr>
<td>options.requestInit</td>
<td>The fetch init object to use. This can have request headers, method, redirect, etc. See Interface RequestInit for more.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>k8s model</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resource</td>
<td>k8s resource</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>model</td>
<td>k8s model</td>
</tr>
</tbody>
</table>

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

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>firstColumn</code></td>
<td>values for first column of popup</td>
</tr>
<tr>
<td><code>secondColumn</code></td>
<td>(optional) values for second column of popup</td>
</tr>
<tr>
<td><code>children</code></td>
<td>(optional) children for the popup</td>
</tr>
</tbody>
</table>

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

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>value</code></td>
<td>(optional) text value to display</td>
</tr>
<tr>
<td><code>icon</code></td>
<td>(optional) icon to display</td>
</tr>
<tr>
<td><code>children</code></td>
<td>child elements</td>
</tr>
</tbody>
</table>
Overview
Creates a wrapper component for a dashboard.

Example

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>(optional) style class for div</td>
</tr>
<tr>
<td>children</td>
<td>(optional) elements of the dashboard</td>
</tr>
</tbody>
</table>

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

Example

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mainCards</td>
<td>cards for grid</td>
</tr>
<tr>
<td>leftCards</td>
<td>(optional) cards for left side of grid</td>
</tr>
<tr>
<td>rightCards</td>
<td>(optional) cards for right side of grid</td>
</tr>
</tbody>
</table>

InventoryItem
Creates an inventory card item.

Example

```jsx
return (
  <InventoryItem>
    <InventoryItemTitle>{title}</InventoryItemTitle>
    <InventoryItemBody error={loadError}>
      {loaded && <InventoryItemStatus count={workerNodes.length} icon={<MonitoringIcon />} />}
    </InventoryItemBody>
  </InventoryItem>
)
```
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>elements to render inside the item</td>
</tr>
</tbody>
</table>

**InventoryItemTitle**

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

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>elements to render inside the title</td>
</tr>
</tbody>
</table>

**InventoryItemBody**

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

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>children</td>
<td>elements to render inside the Inventory Card or title</td>
</tr>
<tr>
<td>error</td>
<td>elements of the div</td>
</tr>
</tbody>
</table>

**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>
)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>count for display</td>
</tr>
<tr>
<td>icon</td>
<td>icon for display</td>
</tr>
<tr>
<td>linkTo</td>
<td>(optional) link address</td>
</tr>
</tbody>
</table>

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

Example

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>The feature flag to return</td>
</tr>
</tbody>
</table>

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

Example

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

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

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

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

A component to show events related to a particular resource.

Example

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

An object whose related events should be shown.

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.

one of the PrometheusEndpoint (label, query, range, rules, targets)

(optional) Prometheus query string. If empty or undefined, polling is not started.

(optional) polling delay interval (ms)

(optional) for QUERY_RANGE endpoint, end of the query range

(optional) for QUERY_RANGE endpoint

(optional) a search param to append

(optional) a search param to append

A component to render timestamp. The timestamps are synchronized between individual instances of the Timestamp component. The provided timestamp is formatted according to user locale.
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>the timestamp to render. Format is expected to be ISO 8601 (used by Kubernetes), epoch timestamp, or an instance of a Date.</td>
</tr>
<tr>
<td>simple</td>
<td>render simple version of the component omitting icon and tooltip.</td>
</tr>
<tr>
<td>omitSuffix</td>
<td>formats the date omitting the suffix.</td>
</tr>
<tr>
<td>className</td>
<td>additional class name for the component.</td>
</tr>
</tbody>
</table>

**useModal**
A hook to launch Modals.

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

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

**Example**
```jsx
const context: ActionContext = { 'a-context-id': { dataFromDynamicPlugin } }; ...

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

</ActionServiceProvider>
```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>context</td>
<td>Object with contextId and optional plugin data</td>
</tr>
</tbody>
</table>

**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**

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

_namespaceBar onNamespaceChange={logNamespaceChange}>
_namespaceBarApplicationSelector />
</NamespaceBar>
</Page>
...
```

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>onNamespaceChange</strong></td>
<td>(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.</td>
</tr>
<tr>
<td><strong>isDisabled</strong></td>
<td>(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.</td>
</tr>
<tr>
<td><strong>children</strong></td>
<td>(optional) Additional elements to be rendered inside the toolbar to the right of the namespace dropdown.</td>
</tr>
</tbody>
</table>

**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 in conjunction with an component.

**Example**

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

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorMessage</td>
<td>text description of the error message</td>
</tr>
<tr>
<td>componentStack</td>
<td>component trace of the exception</td>
</tr>
<tr>
<td>stack</td>
<td>stack trace of the exception</td>
</tr>
<tr>
<td>title</td>
<td>title to render as the header of the error boundary page</td>
</tr>
</tbody>
</table>

**PerspectiveContext**

Deprecated: Use the provided `usePerspectiveContext` instead. Creates the perspective context.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PerspectiveContextType</td>
<td>object with active perspective and setter</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceAttributes</td>
<td>resource attributes for access review</td>
</tr>
<tr>
<td>impersonate</td>
<td>impersonation details</td>
</tr>
</tbody>
</table>

**useSafetyFirst**

Deprecated: This hook is not related to console functionality. Hook that ensures a safe asynchronous 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.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialState</td>
<td>initial state value</td>
</tr>
</tbody>
</table>

### 7.5.3. 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:
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 relies 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.

Additional resources

- Understanding service serving certificates
CHAPTER 8. WEB TERMINAL

8.1. INSTALLING THE WEB TERMINAL

You can install the web terminal by using the Web Terminal Operator listed in the OpenShift Container Platform 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 OpenShift Container Platform 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.

8.2. CONFIGURING THE WEB TERMINAL
You can configure timeout and image settings for the web terminal for your current session.

**8.2.1. Configuring the web terminal timeout for a session**

You can change the default timeout period for the web terminal for your current session.

**Prerequisites**

- You have access to an OpenShift Container Platform cluster that has the Web Terminal Operator installed.
- You are logged into the web console.

**Procedure**

1. Click the web terminal icon ( ).
2. Optional: Set the web terminal timeout for the current session:
   a. Click Timeout.
   b. In the field that appears, enter the timeout value.
   c. From the drop-down list, select a timeout interval of **Seconds**, **Minutes**, **Hours**, or **Milliseconds**.
3. Optional: Select a custom image for the web terminal to use.
   a. Click Image.
   b. In the field that appears, enter the URL of the image that you want to use.
4. Click Start to begin a terminal instance using the specified timeout setting.

**8.2.2. Configuring the web terminal image for a session**

You can change the default image for the web terminal for your current session.

**Prerequisites**

- You have access to an OpenShift Container Platform cluster that has the Web Terminal Operator installed.
- You are logged into the web console.

**Procedure**

1. Click the web terminal icon ( ).
2. Click Image to display advanced configuration options for the web terminal image.
3. Enter the URL of the image that you want to use.
4. Click **Start** to begin a terminal instance using the specified image setting.

### 8.3. 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`, `kubens`, `subctl`, and `kubectx`. It also has the context of the project you are working on and automatically logs you in using your credentials.

#### 8.3.1. 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`, `kubens`, `subctl`, and `kubectx` 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 OpenShift Container Platform 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.

   - The **openshift-terminal** project is the default project used for cluster administrators. They do not have the option to choose another project. The Web Terminal Operator installs the DevWorkspace Operator as a dependency.

3. Optional: Set the web terminal timeout for the current session:
   a. Click **Timeout**.
   b. In the field that appears, enter the timeout value.
   c. From the drop-down list, select a timeout interval of **Seconds**, **Minutes**, **Hours**, or **Milliseconds**.

4. Optional: Select a custom image for the web terminal to use.
a. Click Image.

b. In the field that appears, enter the URL of the image that you want to use.

5. Click **Start** to initialize the web terminal using the selected project.

6. Click + to open multiple tabs within the web terminal in the console.

### 8.4. TROUBLESHOOTING THE WEB TERMINAL

#### 8.4.1. 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 OpenShift Container Platform 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.

### 8.5. 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.

#### 8.5.1. 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 OpenShift Container Platform 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.

### 8.5.2. 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 OpenShift Container Platform 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 the CRDs used by the Operator:
WARNING

The DevWorkspace Operator provides custom resource definitions (CRDs) that use conversion webhooks. Failing to remove these CRDs can cause issues in the cluster.

3. Verify that all involved custom resource definitions are removed. The following command should not display any output:

```bash
$ oc get customresourcedefinitions.apiextensions.k8s.io | grep "devfile.io"
```

4. Remove the devworkspace-webhook-server deployment, mutating, and validating webhooks:

```bash
$ oc delete deployment/devworkspace-webhook-server -n openshift-operators
$ oc delete mutatingwebhookconfigurations controller.devfile.io
$ oc delete validatingwebhookconfigurations controller.devfile.io
```

NOTE

If you remove the devworkspace-webhook-server deployment without removing the mutating and validating webhooks, you cannot use oc exec commands to run commands in a container in the cluster. After you remove the webhooks you can use the oc exec commands again.

5. 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.

```bash
$ 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

6. 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 9. DISABLING THE WEB CONSOLE IN OPENSHIFT CONTAINER PLATFORM

You can disable the OpenShift Container Platform web console.

9.1. PREREQUISITES

- Deploy an OpenShift Container Platform cluster.

9.2. 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
  
  apiVersion: operator.openshift.io/v1
  kind: Console
  metadata:
    name: cluster
  spec:
    managementState: Removed
  
  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 10. CREATING QUICK START TUTORIALS IN THE WEB CONSOLE

If you are creating quick start tutorials for the OpenShift Container Platform web console, follow these guidelines to maintain a consistent user experience across all quick starts.

10.1. 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

10.2. 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.

10.3. 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**

10.4. CONTRIBUTING QUICK STARTS

OpenShift Container Platform introduces the quick start custom resource, which is defined by a `ConsoleQuickStart` object. Operators and administrators can use this resource to contribute quick starts to the cluster.

**Prerequisites**

- You must have cluster administrator privileges.

**Procedure**

1. To create a new quick start, run:
$ oc get -o yaml consolequickstart spring-with-s2i > my-quick-start.yaml

2. Run:

$ oc create -f my-quick-start.yaml

3. Update the YAML file using the guidance outlined in this documentation.

4. Save your edits.

**10.4.1. Viewing the quick start API documentation**

**Procedure**

- To see the quick start API documentation, run:

  ```bash
  $ oc explain consolequickstarts
  ```

Run `oc explain -h` for more information about `oc explain` usage.

**10.4.2. Mapping the elements in the quick start to the quick start CR**

This section helps you visually map parts of the quick start custom resource (CR) with where they appear in the quick start within the web console.

**10.4.2.1. conclusion element**

**Viewing the conclusion element in the YAML file**

```yaml
...
  summary:
    failed: Try the steps again.
    success: Your Spring application is running.
  title: Run the Spring application
  conclusion:
    Your Spring application is deployed and ready.

1 conclusion text
```

**Viewing the conclusion element in the web console**

The conclusion appears in the last section of the quick start.
10.4.2.2. description element

Viewing the description element in the YAML file

```yaml
apiVersion: console.openshift.io/v1
kind: ConsoleQuickStart
metadata:
  name: spring-with-s2i
spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.'
...
```

Viewing the description element in the web console

The description appears on the introductory tile of the quick start on the Quick Starts page.
10.4.2.3. `displayName` element

Viewing the `displayName` element in the YAML file

```yaml
apiVersion: console.openshift.io/v1
kind: ConsoleQuickStart
metadata:
  name: spring-with-s2i
spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.'
displayName: Get started with Spring
durationMinutes: 10
```

`displayName` text.

Viewing the `displayName` element in the web console

The display name appears on the introductory tile of the quick start on the Quick Starts page.
10.4.2.4. durationMinutes element

Viewing the durationMinutes element in the YAML file

```yaml
apiVersion: console.openshift.io/v1
kind: ConsoleQuickStart
metadata:
  name: spring-with-s2i
spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.'
  displayName: Get started with Spring
durationMinutes: 10
```

1 **durationMinutes** value, in minutes. This value defines how long the quick start should take to complete.

Viewing the durationMinutes element in the web console

The duration minutes element appears on the introductory tile of the quick start on the Quick Starts page.
10.4.2.5. icon element

Viewing the icon element in the YAML file

```yaml
... spec:
  description: 'Import a Spring Application from git, build, and deploy it onto OpenShift.'
  displayName: Get started with Spring
  durationMinutes: 10
  icon:
    - data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy5vcmcvMjAwMC9zdmcii8
```

OpenShift Container Platform 4.13 Web console
The icon defined as a base64 value.

Viewing the icon element in the web console
The icon appears on the introductory tile of the quick start on the Quick Starts page.
Viewing the introduction element in the YAML file

```
...  introduction: >-
  **Spring** is a Java framework for building applications based on a distributed microservices architecture.

- Spring enables easy packaging and configuration of Spring applications into a self-contained executable application which can be easily deployed as a container to OpenShift.

- Spring applications can integrate OpenShift capabilities to provide a natural "Spring on OpenShift" developer experience for both existing and net-new Spring applications. For example:

  - Externalized configuration using Kubernetes ConfigMaps and integration with Spring Cloud Kubernetes
  - Service discovery using Kubernetes Services
  - Load balancing with Replication Controllers
  - Kubernetes health probes and integration with Spring Actuator
  - Metrics: Prometheus, Grafana, and integration with Spring Cloud Sleuth
  - Distributed tracing with Istio & Jaeger tracing
  - Developer tooling through Red Hat OpenShift and Red Hat CodeReady developer tooling to quickly scaffold new Spring projects, gain access to familiar Spring APIs in your favorite IDE, and deploy to Red Hat OpenShift
```

The introduction introduces the quick start and lists the tasks within it.

Viewing the introduction element in the web console

After clicking a quick start card, a side panel slides in that introduces the quick start and lists the tasks within it.
Spring is a Java framework for building applications based on a distributed microservices architecture.

- Spring enables easy packaging and configuration of Spring applications into a self-contained executable application which can be easily deployed as a container to OpenShift.
- Spring applications can integrate OpenShift capabilities to provide a natural "Spring on OpenShift" developer experience for both existing and net-new Spring applications. For example:
  - Externalized configuration using Kubernetes ConfigMaps and integration with Spring Cloud Kubernetes
  - Service discovery using Kubernetes Services
  - Load balancing with Replication Controllers
  - Kubernetes health probes and integration with Spring Actuator
  - Metrics: Prometheus, Grafana, and integration with Spring Cloud Sleuth
  - Distributed tracing with Istio & Jaeger tracing
  - Developer tooling through Red Hat OpenShift and Red Hat CodeReady developer tooling to quickly scaffold new Spring projects, gain access to familiar Spring APIs in your favorite IDE, and deploy to Red Hat OpenShift

In this quick start, you will complete 6 tasks:

1. Create a Spring application
2. View the build status
3. View the associated Git repository
4. View the pod status
5. Change the deployment icon to Spring
6. Run the Spring application
10.4.3. Adding a custom icon to a quick start

A default icon is provided for all quick starts. You can provide your own custom icon.

**Procedure**

1. Find the `.svg` file that you want to use as your custom icon.
2. Use an online tool to convert the text to base64.
3. In the YAML file, add `icon: >-`, then on the next line include `data:image/svg+xml;base64` followed by the output from the base64 conversion. For example:

```yaml
icon: >-
   data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC9zdmciIHJvbGU9ImltZyIgdmlld.
```

10.4.4. Limiting access to a quick start

Not all quick starts should be available for everyone. The `accessReviewResources` section of the YAML file provides the ability to limit access to the quick start.

To only allow the user to access the quick start if they have the ability to create `HelmChartRepository` resources, use the following configuration:

```yaml
accessReviewResources:
  - group: helm.openshift.io
    resource: helmchartrepositories
    verb: create
```

To only allow the user to access the quick start if they have the ability to list Operator groups and package manifests, thus ability to install Operators, use the following configuration:

```yaml
accessReviewResources:
  - group: operators.coreos.com
    resource: operatorgroups
    verb: list
  - group: packages/operators.coreos.com
    resource: packagemanifests
    verb: list
```

10.4.5. Linking to other quick starts

**Procedure**

- In the `nextQuickStart` section of the YAML file, provide the `name`, not the `displayName`, of the quick start to which you want to link. For example:

```yaml
nextQuickStart:
  - add-healthchecks
```
10.4.6. Supported tags for quick starts

Write your quick start content in markdown using these tags. The markdown is converted to HTML.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'b',</td>
<td>Defines bold text.</td>
</tr>
<tr>
<td>'img',</td>
<td>Embeds an image.</td>
</tr>
<tr>
<td>'i',</td>
<td>Defines italic text.</td>
</tr>
<tr>
<td>'strike',</td>
<td>Defines strike-through text.</td>
</tr>
<tr>
<td>'s',</td>
<td>Defines smaller text</td>
</tr>
<tr>
<td>'del',</td>
<td>Defines smaller text.</td>
</tr>
<tr>
<td>'em',</td>
<td>Defines emphasized text.</td>
</tr>
<tr>
<td>'strong',</td>
<td>Defines important text.</td>
</tr>
<tr>
<td>'a',</td>
<td>Defines an anchor tag.</td>
</tr>
<tr>
<td>'p',</td>
<td>Defines paragraph text.</td>
</tr>
<tr>
<td>'h1',</td>
<td>Defines a level 1 heading.</td>
</tr>
<tr>
<td>'h2',</td>
<td>Defines a level 2 heading.</td>
</tr>
<tr>
<td>'h3',</td>
<td>Defines a level 3 heading.</td>
</tr>
<tr>
<td>'h4',</td>
<td>Defines a level 4 heading.</td>
</tr>
<tr>
<td>'ul',</td>
<td>Defines an unordered list.</td>
</tr>
<tr>
<td>'ol',</td>
<td>Defines an ordered list.</td>
</tr>
<tr>
<td>'li',</td>
<td>Defines a list item.</td>
</tr>
<tr>
<td>'code',</td>
<td>Defines a text as code.</td>
</tr>
<tr>
<td>'pre',</td>
<td>Defines a block of preformatted text.</td>
</tr>
<tr>
<td>'button',</td>
<td>Defines a button in text.</td>
</tr>
</tbody>
</table>

10.4.7. Quick start highlighting markdown reference
The highlighting, or hint, feature enables Quick Starts to contain a link that can highlight and animate a component of the web console.

The markdown syntax contains:

- Bracketed link text
- The `highlight` keyword, followed by the ID of the element that you want to animate

### 10.4.7.1. Perspective switcher

```
[Perspective switcher]{{highlight qs-perspective-switcher}}
```

### 10.4.7.2. Administrator perspective navigation links

```
[Home]{{highlight qs-nav-home}}
[Operators]{{highlight qs-nav-operators}}
[Workloads]{{highlight qs-nav-workloads}}
[Serverless]{{highlight qs-nav-serverless}}
[Networking]{{highlight qs-nav-networking}}
[Storage]{{highlight qs-nav-storage}}
[Service catalog]{{highlight qs-nav-servicecatalog}}
[Compute]{{highlight qs-nav-compute}}
[User management]{{highlight qs-nav-usermanagement}}
[Administration]{{highlight qs-nav-administration}}
```

### 10.4.7.3. Developer perspective navigation links

```
[Add]{{highlight qs-nav-add}}
[Topology]{{highlight qs-nav-topology}}
[Search]{{highlight qs-nav-search}}
[Project]{{highlight qs-nav-project}}
[Helm]{{highlight qs-nav-helm}}
```

### 10.4.7.4. Common navigation links

```
[Builds]{{highlight qs-nav-builds}}
[Pipelines]{{highlight qs-nav-pipelines}}
[Monitoring]{{highlight qs-nav-monitoring}}
```

### 10.4.7.5. Masthead links

```
[CloudShell]{{highlight qs-masthead-cloudshell}}
[Utility Menu]{{highlight qs-masthead-utilitymenu}}
[User Menu]{{highlight qs-masthead-usermenu}}
[Applications]{{highlight qs-masthead-applications}}
[Import]{{highlight qs-masthead-import}}
[Help]{{highlight qs-masthead-help}}
[Notifications]{{highlight qs-masthead-notifications}}
```

### 10.4.8. Code snippet markdown reference
You can execute a CLI code snippet when it is included in a quick start from the web console. To use this feature, you must first install the Web Terminal Operator. The web terminal and code snippet actions that execute in the web terminal are not present if you do not install the Web Terminal Operator. Alternatively, you can copy a code snippet to the clipboard regardless of whether you have the Web Terminal Operator installed or not.

10.4.8.1. Syntax for inline code snippets

```
code block`{(copy)}
code block`{(execute)}
```

**NOTE**

If the `execute` syntax is used, the Copy to clipboard action is present whether you have the Web Terminal Operator installed or not.

10.4.8.2. Syntax for multi-line code snippets

```
```multi line code block```
```
```multi line code block```
```
```

10.5. QUICK START CONTENT GUIDELINES

10.5.1. Card copy

You can customize the title and description on a quick start card, but you cannot customize the status.

- Keep your description to one to two sentences.
- Start with a verb and communicate the goal of the user. Correct example:

```
Create a serverless application.
```

10.5.2. Introduction

After clicking a quick start card, a side panel slides in that introduces the quick start and lists the tasks within it.

- Make your introduction content clear, concise, informative, and friendly.
- State the outcome of the quick start. A user should understand the purpose of the quick start before they begin.
- Give action to the user, not the quick start.
  - Correct example:
In this quick start, you will deploy a sample application to {product-title}.

- Incorrect example:

  This quick start shows you how to deploy a sample application to {product-title}.

- The introduction should be a maximum of four to five sentences, depending on the complexity of the feature. A long introduction can overwhelm the user.

- List the quick start tasks after the introduction content, and start each task with a verb. Do not specify the number of tasks because the copy would need to be updated every time a task is added or removed.

  - Correct example:

    Tasks to complete: Create a serverless application; Connect an event source; Force a new revision

  - Incorrect example:

    You will complete these 3 tasks: Creating a serverless application; Connecting an event source; Forcing a new revision

10.5.3. Task steps

After the user clicks Start, a series of steps appears that they must perform to complete the quick start.

Follow these general guidelines when writing task steps:

- Use “Click” for buttons and labels. Use “Select” for checkboxes, radio buttons, and drop-down menus.

- Use “Click” instead of “Click on”

  - Correct example:

    Click OK.

  - Incorrect example:

    Click on the OK button.

- Tell users how to navigate between Administrator and Developer perspectives. Even if you think a user might already be in the appropriate perspective, give them instructions on how to get there so that they are definitely where they need to be.

  Examples:

    Enter the Developer perspective: In the main navigation, click the dropdown menu and select Developer.
    Enter the Administrator perspective: In the main navigation, click the dropdown menu and select Admin.

- Use the “Location, action” structure. Tell a user where to go before telling them what to do.
• Correct example:
  In the node.js deployment, hover over the icon.

• Incorrect example:
  Hover over the icon in the node.js deployment.

• Keep your product terminology capitalization consistent.

• If you must specify a menu type or list as a dropdown, write “dropdown” as one word without a hyphen.

• Clearly distinguish between a user action and additional information on product functionality.
  • User action:
    Change the time range of the dashboard by clicking the dropdown menu and selecting time range.

  • Additional information:
    To look at data in a specific time frame, you can change the time range of the dashboard.

• Avoid directional language, like "In the top-right corner, click the icon". Directional language becomes outdated every time UI layouts change. Also, a direction for desktop users might not be accurate for users with a different screen size. Instead, identify something using its name.
  • Correct example:
    In the navigation menu, click Settings.

  • Incorrect example:
    In the left-hand menu, click Settings.

• Do not identify items by color alone, like "Click the gray circle". Color identifiers are not useful for sight-limited users, especially colorblind users. Instead, identify an item using its name or copy, like button copy.
  • Correct example:
    The success message indicates a connection.

  • Incorrect example:
    The message with a green icon indicates a connection.

• Use the second-person point of view, you, consistently:
  • Correct example:
    Set up your environment.
Incorrect example:

Let's set up our environment.

10.5.4. Check your work module

- After a user completes a step, a Check your work module appears. This module prompts the user to answer a yes or no question about the step results, which gives them the opportunity to review their work. For this module, you only need to write a single yes or no question.
  - If the user answers Yes, a check mark will appear.
  - If the user answers No, an error message appears with a link to relevant documentation, if necessary. The user then has the opportunity to go back and try again.

10.5.5. Formatting UI elements

Format UI elements using these guidelines:

- Copy for buttons, dropdowns, tabs, fields, and other UI controls: Write the copy as it appears in the UI and bold it.

- All other UI elements—including page, window, and panel names: Write the copy as it appears in the UI and bold it.

- Code or user-entered text: Use monospaced font.

- Hints: If a hint to a navigation or masthead element is included, style the text as you would a link.

- CLI commands: Use monospaced font.

- In running text, use a bold, monospaced font for a command.

- If a parameter or option is a variable value, use an italic monospaced font.

- Use a bold, monospaced font for the parameter and a monospaced font for the option.

10.6. ADDITIONAL RESOURCES

- For voice and tone requirements, refer to PatternFly’s brand voice and tone guidelines.

- For other UX content guidance, refer to all areas of PatternFly's UX writing style guide.