OpenShift Dedicated 4

Web コンソール

Red Hat OpenShift Dedicated Web コンソールのスタートガイド

Last Updated: 2024-10-15

OpenShift Dedicated 4 Web コンソール

Red Hat OpenShift Dedicated Web コンソールのスタートガイド

法律上の通知

概要

このドキュメントでは、OpenShift Service Dedicated にアクセスし、カスタマイズする方法を説明します。

目次

第1章 WEB コンソールの概要 **3**
1.1. WEB コンソールの ADMINISTRATOR パースペクティブについて 3
1.2. WEB コンソールの DEVELOPER パースペクティブ 3
1.3. パースペクティブへのアクセス 4

第2章 WEB コンソールへのアクセス **6**
2.1. 前提条件 6
2.2. WEB コンソールの理解および WEB コンソールへのアクセス 6

第3章 OPENSHIFT DEDICATED ダッシュボードを使用したクラスター情報の取得 **7**
3.1. OPENSHIFT DEDICATED のダッシュボードページについて 7
3.2. リソースおよびプロジェクトの制限とクォータの認識 8

第4章 動的プラグイン **9**
4.1. 動的プラグインの概要 9
4.2. 動的プラグインを使い始める 12
4.3. クラスターへのプラグインのデプロイ 13
4.4. 動的プラグインの例 17
4.5. 動的プラグイン参照 19

第5章 WEB 端末 **92**
5.1. WEB 端末のインストール 92
5.2. WEB 端末の使用 93
5.3. WEB 端末のトラブルシューティング 93
5.4. WEB 端末のアンインストール 94

第6章 クイックスタートチュートリアルについて **97**
6.1. クイックスタートについて 97
6.2. クイックスタートのユーザーワークフロー 97
6.3. クイックスタートのコンポーネント 98

第1章 WEB コンソールの概要

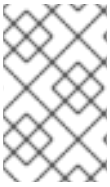
Red Hat OpenShift Dedicated の Web コンソールは、プロジェクトデータを視覚化し、管理およびトラブルシューティングタスクを実行するためのグラフィカルユーザーインターフェイスを備えています。Web コンソールは、openshift-console プロジェクトのコントロールプレーンノードで Pod として実行されます。これは **console-operator** Pod によって管理されます。**Administrator** および **Developer** パースペクティブの両方がサポートされます。

Administrator および **Developer** パースペクティブの両方で、OpenShift Dedicated のクイックスタートチュートリアルを作成できます。クイックスタートは、ユーザータスクに関するガイド付きチュートリアルで、アプリケーション、Operator、またはその他の製品オファリングを理解するのに役立ちます。

1.1. WEB コンソールの ADMINISTRATOR パースペクティブについて

Administrator パースペクティブでは、クラスターインベントリー、容量、全般的および特定の使用に関する情報、および重要なイベントのストリームを表示できます。これらはすべて、プランニングおよびトラブルシューティングの作業を簡素化するのに役立ちます。プロジェクト管理者とクラスター管理者の両方が **Administrator** パースペクティブを表示できます。

OpenShift Dedicated 4.7 以降の場合、クラスター管理者は Web Terminal Operator を使用して組み込みのコマンドラインターミナルインスタンスを開くこともできます。



注記

表示されるデフォルトの Web コンソールパースペクティブは、ユーザーのロールによって異なります。ユーザーが管理者として認識される場合、**Administrator** パースペクティブがデフォルトで表示されます。

Administrator パースペクティブは、以下を実行する機能などの管理者のユースケースに固有のワークフローを提供します。

- ワークロード、ストレージ、ネットワーク、およびクラスター設定を管理する。
- Operator Hub を使用して Operator をインストールし、管理する。
- ユーザーにログインを許可し、ロールおよびロールバインディングを使用してユーザーアクセスを管理できるようにするアイデンティティープロバイダーを追加する。
- クラスターの更新、部分的なクラスターの更新、クラスター Operator、カスタムリソース定義 (CRD)、ロールバインディング、リソースクォータなど、さまざまな高度な設定を表示および管理する。
- メトリクス、アラート、モニタリングダッシュボードなどのモニタリング機能にアクセスし、管理する。
- クラスターに関するロギング、メトリック、および高ステータスの情報を表示し、管理する。
- OpenShift Dedicated の **Administrator** パースペクティブに関連するアプリケーション、コンポーネント、およびサービスを視覚的に操作する。

1.2. WEB コンソールの DEVELOPER パースペクティブ

Developer パースペクティブは、アプリケーション、サービス、データベースをデプロイするために組み込まれたさまざまな手法を提供します。**Developer** パースペクティブでは、以下を実行できます。

- コンポーネントでのロールアウトのローリングおよび再作成をリアルタイムに可視化する。
- アプリケーションのステータス、リソースの使用状況、プロジェクトイベントのストリーミング、およびクォータの消費を表示する。
- プロジェクトを他のユーザーと共有する。
- プロジェクトで Prometheus Query Language (PromQL) クエリーを実行し、グラフに可視化されたメトリックを検査して、アプリケーションに関する問題のトラブルシューティングを行う。メトリックにより、クラスターの状態と、モニターしているユーザー定義のワークロードに関する情報が提供されます。

OpenShift Dedicated 4.7 以降の場合、クラスター管理者は Web コンソールで組み込みのコマンドラインターミナルインスタンスを開くこともできます。

注記

表示されるデフォルトの Web コンソールパースペクティブは、ユーザーのロールによって異なります。**Developer** パースペクティブは、ユーザーが開発者として認識される場合、デフォルトで表示されます。

Developer パースペクティブは、以下を実行する機能を含む、開発者のユースケースに固有のワークフローを提供します。

- 既存のコードベース、イメージ、コンテナーファイルをインポートして、OpenShift Dedicated でアプリケーションを作成およびデプロイします。
- アプリケーション、コンポーネント、およびプロジェクト内のこれらに関連付けられたサービスと視覚的に対話し、それらのデプロイメントとビルドステータスを監視します。
- アプリケーション内のコンポーネントをグループ化し、アプリケーション内およびアプリケーション間でコンポーネントを接続します。
- Serverless 機能 (テクノロジープレビュー) を統合します。
- Eclipse Che を使用してアプリケーションコードを編集するためのワークスペースを作成します。

Topology ビューを使用して、プロジェクトのアプリケーション、コンポーネント、およびワークロードを表示できます。プロジェクトにワークロードがない場合、**Topology** ビューにはワークロードを作成またはインポートするためのリンクがいくつか表示されます。**Quick Search** を使用してコンポーネントを直接インポートすることもできます。

関連情報

Developer パースペクティブで **Topology** ビューを使用する方法の詳細は、Topology ビューを使用したアプリケーション構成の表示 を参照してください。

1.3. パースペクティブへのアクセス

次のように、Web コンソールから **Administrator** および **Developer** パースペクティブにアクセスできます。

前提条件

パースペクティブにアクセスするには、Web コンソールにログインしていることを確認してください。デフォルトのパースペクティブは、ユーザーの権限によって自動的に決定されます。すべてのプロジェクトへのアクセス権を持つユーザーには Administrator パースペクティブが選択され、自分のプロジェクトへのアクセスが制限されているユーザーには Developer パースペクティブが選択されます。

関連情報

パースペクティブの変更の詳細は、ユーザー設定の追加 を参照してください。

手順

1. パースペクティブスイッチャーを使用して、Administrator パースペクティブまたは Developer パースペクティブに切り替えます。
2. Project ドロップダウンリストから既存のプロジェクトを選択します。このドロップダウンから新しいプロジェクトを作成することもできます。

注記

パースペクティブスイッチャーは、**cluster-admin** としてのみ使用できます。

関連情報

- クラスター情報の表示
- Web 端末の使用
- クイックスタートチュートリアルの作成

第2章 WEB コンソールへのアクセス

OpenShift Dedicated Web コンソールは、Web ブラウザーからアクセスできるユーザーインターフェイスです。開発者は Web コンソールを使用してプロジェクトのコンテンツを視覚的に把握し、参照し、管理することができます。

2.1. 前提条件

- Web コンソールを使用するために JavaScript が有効にされている必要があります。WebSocket をサポートする Web ブラウザーを使用することが最も推奨されます。
- OpenShift Container Platform 4.x のテスト済みインテグレーション のページを確認してから、クラスターのサポートされるインフラストラクチャーを作成します。

2.2. WEB コンソールの理解および WEB コンソールへのアクセス

Web コンソールは、コントロールプレーンノード上で Pod として実行されます。Web コンソールを実行するために必要な静的アセットは Pod によって提供されます。

手順

1. OpenShift Cluster Manager にログインし、クラスターの名前をクリックします。
2. クラスターの **Overview** タブで、**Open console** をクリックし、認証情報を使用してログインします。

または、**oc whoami --show-console** コマンドを使用して、Web コンソールの URL を取得します。

第3章 OPENSHIFT DEDICATED ダッシュボードを使用したクラスター情報の取得

OpenShift Dedicated の Web コンソールは、クラスターに関する概要情報を取得します。

3.1. OPENSHIFT DEDICATED のダッシュボードページについて

OpenShift Dedicated Web コンソールから **Home → Overview** に移動して、クラスターに関する概要情報を取得する OpenShift Dedicated ダッシュボードにアクセスします。

OpenShift Dedicated ダッシュボードでは、個々のダッシュボードカードに、取得されたさまざまなクラスター情報が表示されます。

OpenShift Dedicated ダッシュボードは次のカードで構成されています。

- **Details** は、クラスターの詳細情報の概要を表示します。
  ステータスには、**ok**、**error**、**warning**、**in progress**、および **unknown** が含まれます。リソースでは、カスタムのステータス名を追加できます。
  - クラスター
  - プロバイダー
  - バージョン
- **Cluster Inventory** は、リソースの数および関連付けられたステータスの詳細を表示します。これは、問題の解決に介入が必要な場合に役立ちます。以下に関する情報が含まれます。
  - ノード数
  - Pod 数
  - 永続ストレージボリューム要求
  - 状態別にリスト表示されたクラスター内のベアメタルホスト (**metal3** 環境でのみ利用可能)
- **Status** は、管理者がクラスターリソースの消費状況を把握するのに役立ちます。リソースをクリックし、指定されたクラスターリソース (CPU、メモリー、またはストレージ) の最大量を消費する Pod およびノードを一覧表示する詳細ページに切り替えます。
- **Cluster Utilization** には、指定期間におけるさまざまなリソースの容量が表示されます。これは、リソース消費量が多い場合に、管理者がその規模と頻度を把握するのに役立ちます。次の情報が表示されます。
  - CPU 時間
  - メモリー割り当て
  - 消費されたストレージ
  - 消費されたネットワークリソース
  - Pod 数
- **Activity** には、Pod の作成や別のホストへの仮想マシンの移行など、クラスター内の最近のアクティビティーに関連するメッセージがリスト表示されます。

3.2. リソースおよびプロジェクトの制限とクォータの認識

Web コンソールの **Developer** パースペクティブの **Topology** ビューで、利用可能なリソースのグラフィカル表示を使用できます。

リソースの制限やクォータに到達したことを示すメッセージがリソースにある場合は、リソース名の周囲に黄色の境界線が表示されます。メッセージを表示するには、リソースをクリックしてサイドパネルを開きます。**Topology** ビューがズームアウトされている場合、黄色の点はメッセージがあることを示します。

View Shortcuts メニューから **List View** を使用すると、リソースのリストが表示されます。**Alerts** 列は、メッセージがあるかどうかを示します。

第4章 動的プラグイン

4.1. 動的プラグインの概要

4.1.1. 動的プラグインについて

動的プラグインは実行時にリモートのソースからロードされ、解釈されます。動的プラグインをコンソールに提供して公開する 1 つの方法は、OLM Operator を使用することです。OLM Operator は、プラグインをホストする HTTP サーバーを備えたプラットフォーム上にデプロイメントを作成し、Kubernetes サービスを使用してそれを公開します。

動的プラグインを使用すると、実行時にカスタムページやその他の拡張機能をコンソールユーザーインターフェイスに追加できます。**ConsolePlugin** カスタムリソースはコンソールにプラグインを登録し、クラスター管理者はコンソールの Operator 設定でプラグインを有効にします。

4.1.2. 主な特長

動的プラグインを使用すると、OpenShift Dedicated のインターフェイスに次のカスタマイズを行うことができます。

- カスタムページの追加。
- 管理者と開発者を超えたパースペクティブを追加します。
- ナビゲーション項目の追加。
- リソースページへのタブおよびアクションの追加。

4.1.3. 全般的なガイドライン

プラグインの作成時には、以下の一般的なガイドラインに従ってください。

- プラグインをビルドして実行するには、**Node.js** と **yarn** が必要です。
- CSS クラス名の前にプラグイン名を付けて、競合を回避します。例: **my-plugin__heading** および **my-plugin__icon**
- 他のコンソールページとの一貫したルック、フィール、および動作を維持します。
- プラグインの作成時には、react-i18next のローカリゼーションガイドラインに従ってください。以下の例のように **useTranslation** フックを使用できます。

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

- 要素セレクターなど、プラグインコンポーネント外のマークアップに影響を与える可能性のあるセレクターは避けてください。これらは API ではなく、変更される可能性があります。これらを使用すると、プラグインが破損する可能性があります。プラグインコンポーネント外のマークアップに影響を与える可能性のある要素セレクターなどのセレクターを回避します。
- プラグイン Web サーバーが提供するすべてのアセットの **Content-Type** 応答ヘッダーを使用して、有効な JavaScript Multipurpose Internet Mail Extension (MIME) タイプを指定します。各プ

ラグインデプロイメントには、そのプラグインの生成済みアセットをホストする Web サーバーが含まれている必要があります。

- Webpack バージョン 5 以降を使用して、Webpack でプラグインをビルドします。
- 競合を避けるために、CSS クラス名の前にプラグイン名を付けます。例: **my-plugin__heading** および **my-plugin__icon**
- 他のコンソールページと一貫性のある外観、操作性、動作を維持します。
- 要素セレクターなど、プラグインコンポーネント外部のマークアップに影響を与える可能性のあるセレクターは避けてください。これらは API ではなく、変更される可能性があります。
- プラグイン Web サーバーによって提供されるすべてのアセットに対して、**Content-Type** 応答ヘッダーを使用して有効な JavaScript Multipurpose Internet Mail Extension (MIME) タイプを指定する必要があります。各プラグインデプロイメントには、そのプラグインの生成済みアセットをホストする Web サーバーが含まれている必要があります。

PatternFly ガイドライン
プラグインを作成する場合は、PatternFly の使用に関する以下のガイドラインに従ってください。

- PatternFly コンポーネンツと PatternFly CSS 変数を使用します。コア PatternFly コンポーネントは SDK から利用できます。PatternFly コンポーネントと変数を使用すると、将来のコンソールバージョンでプラグインが一貫しているように見えます。
  - OpenShift Dedicated バージョン 4.14 以前を使用している場合は、Patternfly 4.x を使用してください。
  - OpenShift Dedicated 4.15 以降を使用している場合は、Patternfly 5.x を使用してください。
- PatternFly's accessibility fundamentals に従って、プラグインにアクセスできるようにします。
- Bootstrap や Tailwind などの他の CSS ライブラリーは使用しないでください。これらは PatternFly と競合し、コンソールの他の部分とマッチしない可能性があります。プラグインには、基本の PatternFly スタイルに加え、評価対象のユーザーインターフェイスに固有のスタイルのみを含めます。**@patternfly/react-styles/*/.css** などのスタイルや **@patternfly/patternfly** パッケージからのスタイルは、プラグインにインポートしないでください。
- コンソールアプリケーションは、サポートされているすべての PatternFly バージョンの基本スタイルをロードします。

4.1.3.1. react-i18next でメッセージを翻訳する

この プラグインテンプレート は、react-i18next を使用してメッセージを翻訳する方法を示しています。

前提条件

- プラグインテンプレートをローカルに複製する必要があります。
- オプション：プラグインをローカルでテストするには、コンテナーで OpenShift Dedicated Web コンソールを実行します。Docker または Podman 3.2.0 以降を使用できます。

手順

1. 名前の競合を避けるために、名前の前に **plugin__** を付けます。このプラグインテンプレートは、デフォルトで **plugin__console-plugin-template** namespace を使用します。プラグインの名前 (例: **plugin__my-plugin)** を変更する場合は更新する必要があります。たとえば、**useTranslation** フックを使用できます。

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

重要

i18n namespace を **ConsolePlugin** リソースの名前と一致させる必要があります。

2. 必要な動作に応じて **spec.i18n.loadType** フィールドを設定します。

例4.1 plugin__console-demo-plugin

```
spec:
  backend:
    service:
      basePath: /
      name: console-demo-plugin
      namespace: console-demo-plugin
      port: 9001
    type: Service
  displayName: OpenShift Console Demo Plugin
  i18n:
    loadType: Preload (1)
```

(1) ロード中、動的プラグインの後に、**i18n** namespace からプラグインのすべてのローカリゼーションリソースを読み込みます。

3. **console-extensions.json** 内のラベルに **%plugin__console-plugin-template~My Label%** という形式を使用します。コンソールは、値を **plugin__console-plugin-template** namespace の現在の言語のメッセージに置き換えます。以下に例を示します。

```
{
  "type": "console.navigation/section",
  "properties": {
    "id": "admin-demo-section",
    "perspective": "admin",
    "name": "%plugin__console-plugin-template~Plugin Template%"
  }
}
```

4. i18next-parser の TypeScript ファイルにコメントを追加して、**console-extensions.json** からのメッセージをメッセージカタログに追加します。以下に例を示します。

```
// t('plugin__console-demo-plugin~Demo Plugin')
```

5. メッセージを追加または変更するときにプラグインテンプレートの **locales** フォルダー内の JSON ファイルを更新するために、次のコマンドを実行します。

```
$ yarn i18n
```

4.2. 動的プラグインを使い始める

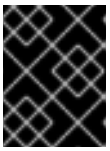
ダイナミックプラグインの使用を開始するには、OpenShift Dedicated の新しい動的プラグインを作成するように環境をセットアップする必要があります。新しいプラグインを作成する方法の例は、Pod ページへのタブの追加 を参照してください。

4.2.1. 動的プラグインの開発

ローカルの開発環境を使用してプラグインを実行できます。OpenShift Dedicated の Web コンソールは、ログインしているクラスターに接続されたコンテナーで実行されます。

前提条件

- プラグインを作成するためのテンプレートが含まれている **console-plugin-template** リポジトリーを複製しておく。



重要

Red Hat はカスタムプラグインコードをサポートしていません。プラグインで利用できるのは、共同コミュニティーのサポート のみです。

- OpenShift Dedicated クラスターが実行中である必要があります。
- OpenShift CLI (**oc**) がインストールされている。
- **yarn** がインストールされている。
- Docker v3.2.0 以降または Podman v3.2.0 以降がインストール済みで実行中である。

手順

1. ターミナルウィンドウを 2 つ開きます。
2. 一方のターミナルウィンドウで、次のコマンドを実行し、yarn を使用してプラグインの依存関係をインストールします。

```
$ yarn install
```

3. インストール後、以下のコマンドを実行して yarn を起動します。

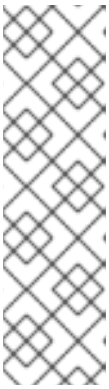
```
$ yarn run start
```

4. 別のターミナルウィンドウで、CLI を使用して OpenShift Dedicated にログインします。

```
$ oc login
```

5. 次のコマンドを実行して、ログインしているクラスターに接続されたコンテナーで OpenShift Dedicated の Web コンソールを実行します。

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



注記

yarn run start-console コマンドは、**amd64** イメージを実行します。Apple Silicon および Podman で実行すると、失敗する可能性があります。これは、以下のコマンドを実行して、**qemu-user-static** を使用すると回避できます。

```
$ podman machine ssh
$ sudo -i
$ rpm-ostree install qemu-user-static
$ systemctl reboot
```

検証

- localhost:9000 にアクセスして、実行中のプラグインを表示します。**window.SERVER_FLAGS.consolePlugins** の値を検査し、ランタイム時にロードされるプラグインの一覧を表示します。

4.3. クラスターへのプラグインのデプロイ

プラグインを OpenShift Dedicated クラスターにデプロイできます。

4.3.1. Docker を使用したイメージのビルド

クラスターにプラグインをデプロイするには、まずイメージをビルドし、それをイメージレジストリーにプッシュする必要があります。

手順

1. 以下のコマンドでイメージをビルドします。

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

2. オプション: イメージをテストする場合は、以下のコマンドを実行します。

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

3. 以下のコマンドを実行してイメージをプッシュします。

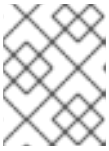
```
$ docker push quay.io/my-repository/my-plugin:latest
```

4.3.2. クラスターへのプラグインのデプロイ

変更を加えたイメージをレジストリーにプッシュした後、Helm チャートを使用してプラグインをクラスターにデプロイできます。

前提条件

- 以前にプッシュしたプラグインを含むイメージの場所が用意されている。



注記

プラグインのニーズに応じて追加のパラメーターを指定できます。**values.yaml** ファイルに、サポートされているパラメーターがすべて含まれています。

手順

1. プラグインをクラスターにデプロイするには、プラグインの名前を Helm リリース名として Helm チャートを、新しい namespace または **-n** コマンドラインオプションで指定された既存の namespace にインストールします。次のコマンドを使用して、**plugin.image** パラメーター内のイメージの場所を指定します。

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

ここでは、以下のようになります。

n <my-plugin-namespace>

プラグインをデプロイする既存の namespace を指定します。

--create-namespace

オプション: 新しい namespace にデプロイする場合は、このパラメーターを使用します。

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

plugin.image パラメーター内のイメージの場所を指定します。

注記

OpenShift Container Platform 4.10 以降にデプロイする場合は、**--set plugin.securityContext.enabled=false** パラメーターを追加して、Pod セキュリティーに関連する設定を除外することを推奨します。

2. オプション: **charts/openshift-console-plugin/values.yaml** ファイルでサポートされている一連のパラメーターを使用して、追加のパラメーターを指定できます。

```
plugin:
  name: ""
  description: ""
  image: ""
  imagePullPolicy: IfNotPresent
  replicas: 2
  port: 9443
  securityContext:
    enabled: true
  podSecurityContext:
    enabled: true
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containerSecurityContext:
    enabled: true
    allowPrivilegeEscalation: false
    capabilities:
      drop:
        - ALL
```

```
    resources:
      requests:
        cpu: 10m
        memory: 50Mi
    basePath: /
    certificateSecretName: ""
    serviceAccount:
      create: true
      annotations: {}
      name: ""
    patcherServiceAccount:
      create: true
      annotations: {}
      name: ""
    jobs:
      patchConsoles:
        enabled: true
        image: "registry.redhat.io/openshift4/ose-tools-
rhel8@sha256:e44074f21e0cca6464e50cb6ff934747e0bd11162ea01d522433a1a1ae116103"

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

検証

- 有効なプラグインのリストを表示するには、**Administration → Cluster Settings → Configuration → Console operator.openshift.io → Console plugins** に移動するか、**Overview** ページにアクセスします。

注記

新しいプラグイン設定が表示されるまで数分かかる場合があります。最近プラグインを有効にしたにもかかわらず、プラグインが表示されない場合は、ブラウザを更新する必要が生じる可能性があります。実行時にエラーが発生した場合は、ブラウザー開発者ツールの JS コンソールをチェックして、プラグインコードにエラーがないか調べてください。

4.3.3. プラグインサービスプロキシー

プラグインからクラスター内のサービスに HTTP リクエストを送信する必要がある場合は、**spec.proxy** 配列フィールドを使用して、**ConsolePlugin** リソースでサービスプロキシーを宣言できます。コンソールバックエンドは、プラグインとサービス間の通信をプロキシーするため

に、**/api/proxy/plugin/<plugin-name>/<proxy-alias>/<request-path>?<optional-query-parameters>** エンドポイントを公開します。プロキシーされたリクエストは、デフォルトで **サービス CA バンドル** を使用します。このサービスは HTTPS を使用する必要があります。

注記

プラグインは、JavaScript コードからリクエストを行うために **consolefetch** API を使用する必要があります。そうしないと、一部のリクエストが失敗する可能性があります。詳細は、「動的プラグイン API」を参照してください。

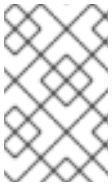
エントリーごとに、**endpoint** and **alias** フィールドでプロキシーのエンドポイントとエイリアスを指定する必要があります。Service プロキシータイプの場合、エンドポイント **type** フィールドを **Service** に設定し、**service** に **name**、**namespace**、および **port** フィールドの値を含める必要があります。たとえば、**/api/proxy/plugin/Helm/Helm-charts/releases?limit=10** は、10 個の Helm リリースをリストする **helm-charts** サービスが含まれる **helm** プラグインからのプロキシー要求パスです。

サービスプロキシーの例

```
apiVersion: console.openshift.io/v1
kind: ConsolePlugin
metadata:
  name:<plugin-name>
spec:
  proxy:
  - alias: helm-charts (1)
    authorization: UserToken (2)
    caCertificate: '-----BEGIN CERTIFICATE-----\nMIID....'en (3)
    endpoint: (4)
      service:
        name: <service-name>
        namespace: <service-namespace>
        port: <service-port>
      type: Service
```

(1) プロキシーのエイリアス。

(2) サービスプロキシー要求にログインしたユーザーの OpenShift Dedicated アクセストークンを含める必要がある場合は、authorization フィールドを **UserToken** に設定する必要があります。



注記

サービスプロキシー要求にログインしたユーザーの OpenShift Dedicated アクセストークンが含まれていない場合は、authorization フィールドを **None** に設定します。

(3) サービスがカスタムサービス CA を使用する場合、**caCertificate** フィールドに証明書バンドルが含まれている必要があります。

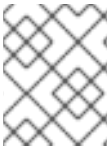
(4) プロキシーのエンドポイント。

4.3.4. ブラウザーでのプラグインの無効化

コンソールユーザーは、**disable-plugins** クエリーパラメーターを使用して、通常ランタイム時にロードされる特定またはすべての動的プラグインを無効にすることができます。

手順

- 特定のプラグインを無効にするには、プラグイン名のコンマ区切りリストから無効にするプラグインを削除します。
- すべてのプラグインを無効にするには、**disable-plugins** クエリーパラメーターを空の文字列のままにします。



注記

クラスター管理者は、Web コンソールの **Cluster Settings** ページでプラグインを無効にできます。

4.4. 動的プラグインの例

例を実行する前に、動的プラグイン開発 の手順に従って、プラグインが機能していることを確認してください。

4.4.1. Pod ページへのタブの追加

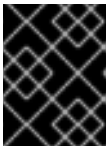
OpenShift Dedicated の Web コンソールには、さまざまなカスタマイズを行うことができます。以下の手順では、サンプルとしてプラグインにタブを **Pod details** ページに追加します。

注記

OpenShift Dedicated の Web コンソールは、ログインしているクラスターに接続されたコンテナーで実行されます。独自のプラグインを作成する前にプラグインをテストするための情報は、「動的プラグインの開発」を参照してください。

手順

1. 新しいタブでプラグインを作成するためのテンプレートを含む **console-plugin-template** リポジトリーにアクセスします。



重要

カスタムプラグインコードは、Red Hat ではサポートされていません。プラグインで利用できるのは、共同コミュニティーのサポート のみです。

2. **Use this template → Create new repository** をクリックして、テンプレートの GitHub リポジトリーを作成します。
3. プラグインの名前で新しいリポジトリーの名前を変更します。
4. コードを編集できるように、新しいリポジトリーのクローンをローカルマシンに作成します。
5. **package.json** ファイルを編集して、プラグインのメタデータを **consolePlugin** 宣言に追加します。以下に例を示します。

```
"consolePlugin": {
```

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

(1) プラグインの名前を更新します。

(2) バージョンを更新します。

(3) プラグインの表示名を更新します。

(4) プラグインの概要を使用して、説明を更新します。

6. **console-extensions.json** ファイルに以下を追加します。

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

7. **package.json** ファイルを編集して以下の変更を追加します。

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

8. 新しいファイル **src/components/ExampleTab.tsx** を作成し、以下のスクリプトを追加することで、**Pod** ページの新規カスタムタブに表示されるメッセージを作成します。

```
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. プラグインをクラスターにデプロイするには、プラグインの名前を Helm リリース名として Helm チャートを、新しい namespace または **-n** コマンドラインオプションで指定された既存の namespace にインストールします。次のコマンドを使用して、**plugin.image** パラメーター内のイメージの場所を指定します。

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

注記

クラスターへのプラグインのデプロイの詳細は、「クラスターへのプラグインのデプロイ」を参照してください。

検証

- Pod ページに移動し、追加されたタブを表示します。

4.5. 動的プラグイン参照

プラグインのカスタマイズを可能にするエクステンションを追加できます。これらのエクステンションは、ランタイム時にコンソールにロードされます。

4.5.1. 動的プラグインエクステンションのタイプ

console.action/filter
ActionFilter を使用してアクションを絞り込むことができます。

名前	値のタイプ	任意	説明
contextId	**string**	いいえ	コンテキスト ID は、提供したアクションのスコープをアプリケーションの特定のエリアに限定するのに役立ちますたとえば、**トポロジー** および **helm** などがあります。

名前	値のタイプ	任意	説明
filter	**CodeRef<(スコープ: any、action: Action) ⇒ boolean>**	いいえ	一部の条件に基づいてアクションをフィルターする関数。 **scope**: アクションを指定するスコープ。Horizontal Pod Autoscaler (HPA) のデプロイメントから **ModifyCount** アクションを削除する必要がある場合には、フックが必要になることがあります。

console.action/group
ActionGroup は、サブメニューに指定可能なアクショングループを提供します。

名前	値のタイプ	任意	説明
id	**string**	いいえ	アクションの選択を識別するための ID。
label	**string**	はい	UI に表示されるラベル。サブメニューに必要です。
submenu	**boolean**	はい	このグループをサブメニューとして表示するかどうか。
insertBefore	**string | string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。
insertAfter	**string | string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** 値が優先されます。

console.action/provider
ActionProvider は、特定のコンテキストに対するアクションのリストを返すフックを提供します。

名前	値のタイプ	任意	説明
contextId	**string**	いいえ	コンテキスト ID は、提供したアクションのスコープをアプリケーションの特定のエリアに限定するのに役立ちますたとえば、**トポロジー** および **helm** などがあります。
provider	**CodeRef<ExtensionHook<Action[], any>>**	いいえ	指定のスコープのアクションを返す React フック。**contextId** = **resource** の場合には、スコープは常に Kubernetes リソースオブジェクトになります。

console.action/resource-provider
ResourceActionProvider は、特定のリソースモデルに対するアクションのリストを返すフックを提供します。

名前	値のタイプ	任意	説明
model	**ExtensionK8sKindVersionModel**	いいえ	このプロバイダーがアクションを提供するモデル。
provider	**CodeRef<ExtensionHook<Action[], any>>**	いいえ	指定のリソースモデルに対するアクションを返す反応フック

console.alert-action
このエクステンションを使用すると、特定の Prometheus アラートが **rule.name** 値に基づいてコンソールで観察された場合に、特定のアクションをトリガーできます。

名前	値のタイプ	任意	説明
alert	**string**	いいえ	**alert.rule.name** プロパティーで定義されたアラート名
text	**string**	いいえ	
action	**CodeRef<(alert: any) ⇒ void>**	いいえ	副次的な影響を実行する関数

console.catalog/item-filter

このエクステンションは、特定のカタログ項目をフィルタリングできるハンドラーを追加するプラグインに使用できます。たとえばプラグインは、特定のプロバイダーからの Helm チャートをフィルタリングするフィルターを追加できます。

名前	値のタイプ	任意	説明
catalogId	**string | string[]**	いいえ	このプロバイダーが提供するカタログの一意の識別子。
type	**string**	いいえ	カタログ項目タイプのタイプ ID。
filter	**CodeRef<(item: CatalogItem) ⇒ boolean>**	いいえ	特定のタイプの項目をフィルタリングします。Value は、**CatalogItem[]** を受け取り、フィルター条件に基づいてサブセットを返す関数です。

console.catalog/item-metadata
このエクステンションを使用すると、特定のカタログ項目に追加のメタデータを追加するプロバイダーを追加できます。

名前	値のタイプ	任意	説明
catalogId	**string | string[]**	いいえ	このプロバイダーが提供するカタログの一意の識別子。
type	**string**	いいえ	カタログ項目タイプのタイプ ID。
provider	**CodeRef<ExtensionHook<CatalogItemMetadataProviderFunction, CatalogExtensionHookOptions>>**	いいえ	特定のタイプのカタログ項目にメタデータを提供するために使用される関数を返すフック。

console.catalog/item-provider
このエクステンションを使用すると、プラグインはカタログ項目タイプのプロバイダーを追加できます。たとえば、Helm プラグインは、すべての Helm チャートを取得するプロバイダーを追加できます。このエクステンションを他のプラグインで使用して、特定のカタログ項目タイプをさらに追加することもできます。

名前	値のタイプ	任意	説明
catalogId	**string | string[]**	いいえ	このプロバイダーが提供するカタログの一意の識別子。
type	**string**	いいえ	カタログ項目タイプのタイプ ID。
title	**string**	いいえ	カタログ項目プロバイダーのタイトル
provider	**CodeRef<ExtensionHook<CatalogItem<any>[], CatalogExtensionHookOptions>>**	いいえ	項目を取得し、これをカタログ用に正規化します。値は反応効果フックです。
priority	**number**	はい	このプロバイダーの優先順位。デフォルトは **0** です。優先度の高いプロバイダーは、他のプロバイダーが提供するカタログ項目を上書きする可能性があります。

console.catalog/item-type
このエクステンションを使用すると、プラグインはカタログ項目の新しいタイプを追加できます。たとえば Helm プラグインは、開発者カタログに追加する新しいカタログ項目タイプを HelmCharts として定義できます。

名前	値のタイプ	任意	説明
type	**string**	いいえ	カタログ項目をタイプ。
title	**string**	いいえ	カタログ項目のタイトル。
catalogDescription	**string | CodeRef<React.ReactNode>**	はい	カタログに固有のタイプの説明。
typeDescription	**string**	はい	カタログ項目タイプの説明。
filters	**CatalogItemAttribute[]**	はい	カタログ項目に固有のカスタムフィルター。

名前	値のタイプ	任意	説明
groupings	**CatalogItemAttribute []**	はい	カタログ項目に固有のカスタムグルーピング。

console.catalog/item-type-metadata
このエクステンションを使用すると、プラグインは任意のカタログ項目タイプのカスタムフィルターやグループ化などのメタデータを追加できます。たとえばプラグインは、チャートプロバイダーに基づきフィルタリングできる HelmCharts のカスタムフィルターをアタッチできます。

名前	値のタイプ	任意	説明
type	**string**	いいえ	カタログ項目をタイプ。
filters	**CatalogItemAttribute []**	はい	カタログ項目に固有のカスタムフィルター。
groupings	**CatalogItemAttribute []**	はい	カタログ項目に固有のカスタムグルーピング。

console.cluster-overview/inventory-item
新しいインベントリー項目をクラスターの概要ページに追加します。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<{}>>**	いいえ	レンダリングされるコンポーネント。

console.cluster-overview/multiline-utilization-item
新しいクラスター概要のマルチライン使用状況項目を追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	使用状況項目のタイトル。
getUtilizationQueries	**CodeRef<GetMultilineQueries>**	いいえ	Prometheus 使用状況クエリー。
humanize	**CodeRef<Humanize>**	いいえ	Prometheus データを人間が判読できる形式に変換します。

名前	値のタイプ	任意	説明
TopConsumerPopovers	**CodeRef<React.ComponentType<TopConsumerPopoverProps>[]>**	はい	プレーン値の代わりに Top コンシューマーポップオーバーを表示します。

console.cluster-overview/utilization-item
新しいクラスター概要の使用状況項目を追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	使用状況項目のタイトル。
getUtilizationQuery	**CodeRef<GetQuery>**	いいえ	Prometheus 使用状況クエリー。
humanize	**CodeRef<Humanize>**	いいえ	Prometheus データを人間が判読できる形式に変換します。
getTotalQuery	**CodeRef<GetQuery>**	はい	Prometheus 合計のクエリー。
getRequestQuery	**CodeRef<GetQuery>**	はい	Prometheus 要求のクエリー。
getLimitQuery	**CodeRef<GetQuery>**	はい	Prometheus 制限のクエリー。
TopConsumerPopover	**CodeRef<React.ComponentType<TopConsumerPopoverProps>>**	はい	プレーン値の代わりに Top コンシューマーポップオーバーを表示します。

console.context-provider
新しい React コンテキストプロバイダーを Web コンソールのアプリケーションルートに追加します。

名前	値のタイプ	任意	説明
provider	**CodeRef<Provider<T>>**	いいえ	Context プロバイダーコンポーネント。
useValueHook	**CodeRef<() ⇒ T>**	いいえ	コンテキスト値のフック。

console.dashboards/card
新しいダッシュボードカードを追加します。

名前	値のタイプ	任意	説明
tab	**string**	いいえ	カードを追加するダッシュボードタブの ID。
position	**'LEFT' | 'RIGHT' | 'MAIN'**	いいえ	ダッシュボードのカードのグリッド位置。
component	**CodeRef<React.ComponentType<{}>>**	いいえ	ダッシュボードカードのコンポーネント。
span	**OverviewCardSpan**	はい	列内のカードの垂直スパン。小さな画面では無視され、デフォルトは **12** です。

console.dashboards/custom/overview/detail/item
Overview ダッシュボードの Details カードに項目を追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	Details カードのタイトル
component	**CodeRef<React.ComponentType<{}>>**	いいえ	OverviewDetailItem コンポーネントによってレンダリングされる値
valueClassName	**string**	はい	className の値
isLoading	**CodeRef<() ⇒ boolean>**	はい	コンポーネントのロード中の状態を返す関数
error	**CodeRef<() ⇒ string>**	はい	コンポーネントごとに表示するエラーを返す関数

console.dashboards/overview/activity/resource
Kubernetes リソースの監視に基づいてアクティビティーをトリガーしている Overview ダッシュボードの Activity カードにアクティビティーを追加します。

名前	値のタイプ	任意	説明
k8sResource	**CodeRef<FirehoseResource & { isList: true; }>**	いいえ	置き換える使用状況項目。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<K8sActivityProps<T>>>**	いいえ	アクションコンポーネント。
isActivity	**CodeRef<(resource: T) ⇒ boolean>**	はい	指定のリソースがアクションを表すかどうかを判断する関数。定義されていない場合は、すべてのリソースがアクティビティーを表します。
getTimestamp	**CodeRef<(resource: T) ⇒ Date>**	はい	指定のアクションのタイムスタンプで、順序付けに使用されます。

console.dashboards/overview/health/operator
ステータスのソースが Kubernetes REST API である **Overview** ダッシュボードのステータスカードに health サブシステムを追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	ポップアップメニューの Operators セクションのタイトル。
resources	**CodeRef<FirehoseResource[]>**	いいえ	フェッチされ、**healthHandler** に渡される Kubernetes リソース。
getOperatorsWithStatuses	**CodeRef<GetOperatorsWithStatuses<T>>**	はい	Operator のステータスを解決します。
operatorRowLoader	**CodeRef<React.ComponentType<OperatorRowProps<T>>>**	はい	ポップアップ行コンポーネントのローダー。
viewAllLink	**string**	はい	すべてのリソースページへのリンク。指定しない場合は、resources prop から最初のリソースのリストページが使用されます。

console.dashboards/overview/health/prometheus
ステータスのソースが Prometheus である Overview ダッシュボードのステータスカードに health サブシステムを追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	サブシステムの表示名。
queries	**string[]**	いいえ	Prometheus クエリー
healthHandler	**CodeRef<PrometheusHealthHandler>**	いいえ	サブシステムの健全性を解決します。
additionalResource	**CodeRef<FirehoseResource>**	はい	フェッチされ、**healthHandler** に渡される追加のリソース。
popupComponent	**CodeRef<React.ComponentType<PrometheusHealthPopupProps>>**	はい	ポップアップメニューコンテンツのローダー。定義された場合、health 項目はリンクとして表され、指定のコンテンツを含むポップアップメニューが開きます。
popupTitle	**string**	はい	ポップオーバーのタイトル。
disallowedControlPlaneTopology	**string[]**	はい	サブシステムを非表示にする必要のあるコントロールプレーントポロジー。

console.dashboards/overview/health/resource
ステータスのソースが Kubernetes リソースである概要ダッシュボードのステータスカードに health サブシステムを追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	サブシステムの表示名。
resources	**CodeRef<WatchK8sResources<T>>**	いいえ	フェッチされ、**healthHandler** に渡される Kubernetes リソース。
healthHandler	**CodeRef<ResourceHealthHandler<T>>**	いいえ	サブシステムの健全性を解決します。

名前	値のタイプ	任意	説明
popupComponent	**CodeRef<WatchK8sResults<T>>**	はい	ポップアップメニューコンテンツのローダー。定義された場合、health 項目はリンクとして表され、指定のコンテンツを含むポップアップメニューが開きます。
popupTitle	**string**	はい	ポップオーバーのタイトル。

console.dashboards/overview/health/url
ステータスのソースが Kubernetes REST API である概要ダッシュボードのステータスカードに health サブシステムを追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	サブシステムの表示名。
url	**string**	いいえ	データの取得元の URL。これには、ベース Kubernetes URL が接頭辞として付けられます。
healthHandler	**CodeRef<URLHealthHandler<T, K8sResourceCommon | K8sResourceCommon[]>>**	いいえ	サブシステムの健全性を解決します。
additionalResource	**CodeRef<FirehoseResource>**	はい	フェッチされ、**healthHandler** に渡される追加のリソース。
popupComponent	**CodeRef<React.ComponentType<{ healthResult?: T; healthResultError?: any; k8sResult?: FirehoseResult<R>; }>>**	はい	ポップアップコンテンツのローダー。定義された場合、health 項目は指定のコンテンツのポップアップが開くリンクとして表示されます。
popupTitle	**string**	はい	ポップオーバーのタイトル。

console.dashboards/overview/inventory/item

概要インベントリーカードにリソースタイルを追加します。

名前	値のタイプ	任意	説明
model	**CodeRef<T>**	いいえ	取得する **resource** のモデル。モデルの **label** または **abbr** の取得に使用します。
mapper	**CodeRef<StatusGroupMapper<T, R>>**	はい	さまざまなステータスをグループにマッピングする関数。
additionalResources	**CodeRef<WatchK8sResources<R>>**	はい	フェッチされ、**mapper** 関数に渡される追加のリソース。

console.dashboards/overview/inventory/item/group
インベントリーのステータスグループを追加します。

名前	値のタイプ	任意	説明
id	**string**	いいえ	ステータスグループの ID。
icon	**CodeRef<React.ReactElement<any, string | React.JSXElementConstructor<any>>>**	いいえ	ステータスグループアイコンを表す React コンポーネント。

console.dashboards/overview/inventory/item/replacement
概要のインベントリーカードを置き換えます。

名前	値のタイプ	任意	説明
model	**CodeRef<T>**	いいえ	取得する **resource** のモデル。モデルの **label** または **abbr** の取得に使用します。
mapper	**CodeRef<StatusGroupMapper<T, R>>**	はい	さまざまなステータスをグループにマッピングする関数。
additionalResources	**CodeRef<WatchK8sResources<R>>**	はい	フェッチされ、**mapper** 関数に渡される追加のリソース。

console.dashboards/overview/prometheus/activity/resource
Kubernetes リソースの監視に基づいてアクティビティーをトリガーしている Prometheus Overview ダッシュボードの Activity カードにアクティビティーを追加します。

名前	値のタイプ	任意	説明
queries	**string[]**	いいえ	監視するクエリー。
component	**CodeRef<React.ComponentType<PrometheusActivityProps>>**	いいえ	アクションコンポーネント。
isActivity	**CodeRef<(results: PrometheusResponse[]) ⇒ boolean>**	はい	指定のリソースがアクションを表すかどうかを判断する関数。定義されていない場合は、すべてのリソースがアクティビティーを表します。

console.dashboards/project/overview/item
プロジェクトの概要インベントリーカードにリソースタイルを追加します。

名前	値のタイプ	任意	説明
model	**CodeRef<T>**	いいえ	取得する **resource** のモデル。モデルの **label** または **abbr** の取得に使用します。
mapper	**CodeRef<StatusGroupMapper<T, R>>**	はい	さまざまなステータスをグループにマッピングする関数。
additionalResources	**CodeRef<WatchK8sResources<R>>**	はい	フェッチされ、**mapper** 関数に渡される追加のリソース。

console.dashboards/tab
Overview タブの後に置かれた新規ダッシュボードタブを追加します。

名前	値のタイプ	任意	説明
id	**string**	いいえ	このタブにカードを追加する場合にタブリンク **href** として使用される一意のタブ ID。
navSection	**'home' | 'storage'**	いいえ	タブが属するナビゲーションセクション。

名前	値のタイプ	任意	説明
title	**string**	いいえ	タブのタイトル。

console.file-upload
このエクステンションを使用すると、特定のファイル拡張子に対するファイルドロップアクションのハンドラーを追加できます。

名前	値のタイプ	任意	説明
fileExtensions	**string[]**	いいえ	サポートされるファイル拡張子。
handler	**CodeRef<FileUploadHandler>**	いいえ	ファイルドロップアクションを処理する関数。

console.flag
Web コンソール機能フラグを完全に制御します。

名前	値のタイプ	任意	説明
handler	**CodeRef<FeatureFlagHandler>**	いいえ	任意の機能フラグを設定または設定解除するのに使用されます。

console.flag/hookProvider
フックハンドラーを使用して Web コンソール機能フラグを完全に制御します。

名前	値のタイプ	任意	説明
handler	**CodeRef<FeatureFlagHandler>**	いいえ	任意の機能フラグを設定または設定解除するのに使用されます。

console.flag/model
クラスター上の **CustomResourceDefinition** (CRD) オブジェクトの存在によって駆動される、新しい Web コンソール機能フラグを追加します。

名前	値のタイプ	任意	説明
flag	**string**	いいえ	CRD が検出された後に設定するフラグの名前。
model	**ExtensionK8sModel**	いいえ	CRD を指すモデル。

console.global-config

このエクステンションは、クラスターの設定を管理するために使用されるリソースを識別します。Administration → Cluster Settings → Configuration ページに、リソースへのリンクが追加されます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	クラスター設定リソースインスタンスの一意の識別子。
name	**string**	いいえ	クラスター設定リソースインスタンスの名前。
model	**ExtensionK8sModel**	いいえ	クラスター設定リソースを参照するモデル。
namespace	**string**	いいえ	クラスター設定リソースインスタンスの namespace。

console.model-metadata
API 検出で取得および生成される値を上書きして、モデルの表示をカスタマイズします。

名前	値のタイプ	任意	説明
model	**ExtensionK8sGroupModel**	いいえ	カスタマイズするモデル。グループのみ、またはオプションのバージョンおよび種類を指定できます。
badge	**ModelBadge**	はい	このモデル参照をテクノロジープレビューまたは開発者プレビューとみなすかどうか。
color	**string**	はい	このモデルに関連付ける色。
label	**string**	はい	ラベルをオーバーライドします。**kind** を指定する必要があります。
labelPlural	**string**	はい	複数形のラベルをオーバーライドします。**kind** を指定する必要があります。

名前	値のタイプ	任意	説明
abbr	**string**	はい	省略形をカスタマイズします。デフォルトは **kind** のすべての大文字 (最大 4 文字) です。その **kind** を指定する必要があります。

console.navigation/href
このエクステンションを使用すると、UI 内の特定のリンクを指すナビゲーション項目を追加できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	この項目の一意の識別子。
name	**string**	いいえ	この項目の名前。
href	**string**	いいえ	リンクの **href** の値。
perspective	**string**	はい	この項目が属するパースペクティブ ID。指定されていない場合は、デフォルトのパースペクティブに提供します。
section	**string**	はい	この項目が属するナビゲーションセクション。指定されていない場合は、この項目を最上位のリンクとしてレンダリングします。
dataAttributes	**{ [key: string]: string; }**	はい	データ属性を DOM に追加します。
startsWith	**string[]**	はい	URL がこのパスのいずれかで始まる場合は、この項目をアクティブと識別します。
insertBefore	**string | string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。

名前	値のタイプ	任意	説明
insertAfter	**string** | **string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** が優先されます。
namespaced	**boolean**	はい	**true** の場合、**/ns/active-namespace** を最後に追加します。
prefixNamespaced	**boolean**	はい	**true** の場合、先頭に **/k8s/ns/active-namespace** が追加されます。

console.navigation/resource-cluster
このエクステンションを使用すると、クラスターリソースの詳細ページを指すナビゲーションアイテムを追加できます。そのリソースの K8s モデルを使用して、ナビゲーション項目を定義できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	この項目の一意の識別子。
model	**ExtensionK8sModel**	いいえ	このナビゲーション項目がリンクするモデル。
perspective	**string**	はい	この項目が属するパースペクティブ ID。指定されていない場合は、デフォルトのパースペクティブに提供します。
section	**string**	はい	この項目が属するナビゲーションセクション。指定しない場合は、この項目をトップレベルのリンクとしてレンダリングします。
dataAttributes	**{ [key: string]: string; }**	はい	データ属性を DOM に追加します。

名前	値のタイプ	任意	説明
startsWith	**string[]**	はい	URL がこのパスのいずれかで始まる場合は、この項目をアクティブと識別します。
insertBefore	**string** | **string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。
insertAfter	**string** | **string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** が優先されます。
name	**string**	はい	デフォルト名をオーバーライドします。指定されていない場合、リンクの名前はモデルの複数形の値と同じになります。

console.navigation/resource-ns
このエクステンションを使用すると、namespaced リソースの詳細ページを指すナビゲーション項目を追加できます。そのリソースの K8s モデルを使用して、ナビゲーション項目を定義できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	この項目の一意の識別子。
model	**ExtensionK8sModel**	いいえ	このナビゲーション項目がリンクするモデル。
perspective	**string**	はい	この項目が属するパースペクティブ ID。指定されていない場合は、デフォルトのパースペクティブに提供します。

名前	値のタイプ	任意	説明
section	**string**	はい	この項目が属するナビゲーションセクション。指定しない場合は、この項目をトップレベルのリンクとしてレンダリングします。
dataAttributes	**{ [key: string]: string; }**	はい	データ属性を DOM に追加します。
startsWith	**string[]**	はい	URL がこのパスのいずれかで始まる場合は、この項目をアクティブと識別します。
insertBefore	**string | string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。
insertAfter	**string | string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** が優先されます。
name	**string**	はい	デフォルト名をオーバーライドします。指定されていない場合、リンクの名前はモデルの複数形の値と同じになります。

console.navigation/section
このエクステンションを使用すると、ナビゲーションタブ内の新しいナビゲーション項目セクションを定義できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	この項目の一意の識別子。

名前	値のタイプ	任意	説明
perspective	**string**	はい	この項目が属するパースペクティブ ID。指定されていない場合は、デフォルトのパースペクティブに提供します。
dataAttributes	**{ [key: string]: string; }**	はい	データ属性を DOM に追加します。
insertBefore	**string | string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。
insertAfter	**string | string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** が優先されます。
name	**string**	はい	このセクションの名前。指定しない場合は、セクションの上に区切り記号のみが表示されます。

console.navigation/separator
このエクステンションを使用すると、ナビゲーション内のナビゲーション項目間に区切り文字を追加できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	この項目の一意の識別子。
perspective	**string**	はい	この項目が属するパースペクティブ ID。指定されていない場合は、デフォルトのパースペクティブに提供します。

名前	値のタイプ	任意	説明
section	**string**	はい	この項目が属するナビゲーションセクション。指定されていない場合は、この項目を最上位のリンクとしてレンダリングします。
dataAttributes	**{ [key: string]: string; }**	はい	データ属性を DOM に追加します。
insertBefore	**string | string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。
insertAfter	**string | string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** が優先されます。

console.page/resource/details

名前	値のタイプ	任意	説明
model	**ExtensionK8sGroupKindModel**	いいえ	このリソースページがリンクするモデル。
component	**CodeRef<React.ComponentType<{ match: match<{}>; namespace: string; model: ExtensionK8sModel; }>>**	いいえ	ルートがマッチしたときにレンダリングされるコンポーネント。

console.page/resource/list
Console ルーターに新しいリソースリストのページを追加します。

名前	値のタイプ	任意	説明
model	**ExtensionK8sGroupKindModel**	いいえ	このリソースページがリンクするモデル。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<{ match: match<{}>; namespace: string; model: ExtensionK8sModel; }>>**	いいえ	ルートがマッチしたときにレンダリングされるコンポーネント。

console.page/route
Web コンソールルーターに新しいページを追加します。React Router を参照してください。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	いいえ	ルートがマッチしたときにレンダリングされるコンポーネント。
path	**string | string[]**	いいえ	**path-to-regexp@^1.7.0** が理解する有効な URL パスまたはパスの配列。
perspective	**string**	はい	このページが属するパースペクティブ。指定されていない場合は、すべてのパースペクティブに提供します。
exact	**boolean**	はい	true の場合、パスが **location.pathname** と完全に一致する場合にのみマッチします。

console.page/route/standalone
一般的なページレイアウトの外部でレンダリングされる新しいスタンドアロンページを Web コンソールルーターに追加します。React Router を参照してください。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	いいえ	ルートがマッチしたときにレンダリングされるコンポーネント。

名前	値のタイプ	任意	説明
path	**string | string[]**	いいえ	**path-to-regexp@^1.7.0** が理解する有効な URL パスまたはパスの配列。
exact	**boolean**	はい	true の場合、パスが **location.pathname** と完全に一致する場合にのみマッチします。

console.perspective
このエクステンションを使用すると、コンソールに新しいパースペクティブを追加してナビゲーションメニューをカスタマイズできます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	パースペクティブの識別子。
name	**string**	いいえ	パースペクティブの表示名。
icon	**CodeRef<LazyComponent>**	いいえ	パースペクティブの表示アイコン。
landingPageURL	**CodeRef<(flags: { [key: string]: boolean; }, isFirstVisit: boolean) ⇒ string>**	いいえ	パースペクティブのランディングページの URL を取得する関数。
importRedirectURL	**CodeRef<(namespace: string) ⇒ string>**	いいえ	インポートフローのリダイレクト URL を取得する関数。
default	**boolean**	はい	パースペクティブがデフォルトであるかどうか。デフォルトは 1 つのみです。
defaultPins	**ExtensionK8sModel[]**	はい	ナビゲーション上のデフォルトの固定されたリソース
usePerspectiveDetection	**CodeRef<() ⇒ [boolean, boolean]>**	はい	デフォルトのパースペクティブを検出するフック

console.project-overview/inventory-item
新しいインベントリー項目を **プロジェクトの概要** ページに追加します。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<{ projectName: string; }>>**	いいえ	レンダリングされるコンポーネント。

console.project-overview/utilization-item
新しいプロジェクト概要の使用状況項目を追加します。

名前	値のタイプ	任意	説明
title	**string**	いいえ	使用状況項目のタイトル。
getUtilizationQuery	**CodeRef<GetProjectQuery>**	いいえ	Prometheus 使用状況クエリー。
humanize	**CodeRef<Humanize>**	いいえ	Prometheus データを人間が判読できる形式に変換します。
getTotalQuery	**CodeRef<GetProjectQuery>**	はい	Prometheus 合計のクエリー。
getRequestQuery	**CodeRef<GetProjectQuery>**	はい	Prometheus 要求のクエリー。
getLimitQuery	**CodeRef<GetProjectQuery>**	はい	Prometheus 制限のクエリー。
TopConsumerPopover	**CodeRef<React.ComponentType<TopConsumerPopoverProps>>**	はい	プレーン値の代わりに最上位のコンシューマーポップオーバーを表示します。

console.pvc/alert
このエクステンションを使用すると、PVC 詳細ページにカスタムアラートを追加できます。

名前	値のタイプ	任意	説明
alert	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	いいえ	アラートコンポーネント。

console.pvc/create-prop
このエクステンションを使用すると、PVC リストページで PVC リソースを作成する際に使用される追加のプロパティーを指定できます。

名前	値のタイプ	任意	説明
label	**string**	いいえ	prop アクション作成のラベル。
path	**string**	いいえ	prop アクション作成のパス。

console.pvc/delete
このエクステンションを使用すると、PVC リソースの削除をフッキングできます。追加情報とカスタム PVC 削除ロジックを含むアラートを追加できます。

名前	値のタイプ	任意	説明
predicate	**CodeRef<(pvc: K8sResourceCommon) ⇒ boolean>**	いいえ	エクステンションを使用するかどうかを示す述語。
onPVCKill	**CodeRef<(pvc: K8sResourceCommon) ⇒ Promise<void>>**	いいえ	PVC 削除操作の方法。
alert	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	いいえ	追加情報を表示するアラートコンポーネント。

console.pvc/status

名前	値のタイプ	任意	説明
priority	**number**	いいえ	status コンポーネントの優先度。値が大きいほど優先度が高くなります。
status	**CodeRef<React.ComponentType<{ pvc: K8sResourceCommon; }>>**	いいえ	status コンポーネント。
predicate	**CodeRef<(pvc: K8sResourceCommon) ⇒ boolean>**	いいえ	ステータスコンポーネントをレンダリングするかどうかを示す述語。

console.redux-reducer
plugins.<scope> サブ状態で動作する Console Redux ストアに新しい reducer を追加します。

名前	値のタイプ	任意	説明
scope	**string**	いいえ	Redux 状態オブジェクト内の reducer が管理するサブ状態を表すキー。
reducer	**CodeRef<Reducer<any, AnyAction>>**	いいえ	reducer が管理するサブ状態で動作する reducer 関数

console.resource/create
このエクステンションを使用すると、プラグインは、ユーザーが新しいリソースインスタンスを作成しようとしたときにレンダリングされる特定のリソースのカスタムコンポーネント (つまりウィザードやフォーム) を追加できます。

名前	値のタイプ	任意	説明
model	**ExtensionK8sModel**	いいえ	この create resource ページがレンダリングされるモデル。
component	**CodeRef<React.ComponentType<CreateResourceComponentProps>>**	いいえ	モデルがマッチする場合にレンダリングされるコンポーネント

console.resource/details-item
詳細ページのデフォルトのリソース概要に、新しい詳細項目を追加します。

名前	値のタイプ	任意	説明
model	**ExtensionK8sModel**	いいえ	対象リソースの API グループ、バージョン、カインド。
id	**string**	いいえ	一意の ID
column	**DetailsItemColumn**	いいえ	項目を、詳細ページのリソース概要の '左列' と '右列' のどちらに表示するかを指定します。デフォルト: 'right'
title	**string**	いいえ	詳細項目のタイトル。

名前	値のタイプ	任意	説明
path	**string**	はい	詳細項目の値として使用されるリソースプロパティーへの完全修飾パス (オプション)。primitive type の値以外は直接レンダリングできません。他のデータ型を処理するには、コンポーネントプロパティーを使用します。
component	**CodeRef<React.ComponentType<DetailsItemComponentProps<K8sResourceCommon, any>>>**	はい	詳細項目の値をレンダリングする React コンポーネント (オプション)。
sortWeight	**number**	はい	同じ列内の他の詳細項目すべてに対する相対的な並べ替えの重み (オプション)。任意の有効な JavaScriptNumber で表されます。各列の項目は、低いものから高いものへと個別に並べ替えられます。並べ替えの重みがない項目は、並べ替えの重みがある項目の後に表示されます。

console.storage-class/provisioner
ストレージクラスの作成時に、新しいストレージクラスプロビジョナーをオプションとして追加します。

名前	値のタイプ	任意	説明
CSI	**ProvisionerDetails**	はい	Container Storage Interface プロビジョナータイプ
OTHERS	**ProvisionerDetails**	はい	Other プロビジョナータイプ

console.storage-provider
このエクステンションを使用すると、ストレージおよびプロバイダー固有のコンポーネントをアタッチする際に、新しいストレージプロバイダーを追加できます。

名前	値のタイプ	任意	説明
name	**string**	いいえ	プロバイダーの表示名。
コンポーネント	**CodeRef<React.ComponentType<Partial<RouteComponentProps<{}, StaticContext, any>>>>**	いいえ	レンダリングするプロバイダー固有のコンポーネント。

console.tab
水平ナビゲーションに、**contextId** に一致するタブを追加します。

名前	値のタイプ	任意	説明
contextId	**string**	いいえ	タブが挿入される水平ナビゲーションに割り当てられるコンテキスト ID。使用できる値: **dev-console-observe**
name	**string**	いいえ	タブの表示ラベル
href	**string**	いいえ	既存の URL に追加される **href**
component	**CodeRef<React.ComponentType<PageComponentProps<K8sResourceCommon>>>**	いいえ	タブコンテンツのコンポーネント。

console.tab/horizontalNav
このエクステンションを使用すると、リソースの詳細ページにタブを追加できます。

名前	値のタイプ	任意	説明
model	**ExtensionK8sKindVersionModel**	いいえ	このプロバイダーがタブを表示するモデル。
page	**{ name: string; href: string; }**	いいえ	水平タブに表示されるページ。名前としてタブ名およびタブの href を取ります。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<PageComponentProps<K8sResourceCommon>>>**	いいえ	ルートがマッチしたときにレンダリングされるコンポーネント。

console.telemetry/listener
このコンポーネントは、テレメトリーイベントを受信するリスナー関数を登録するために使用できます。これらのイベントには、ユーザー識別、ページナビゲーション、その他のアプリケーション固有のイベントが含まれます。リスナーは、このデータをレポートと分析のために使用できます。

名前	値のタイプ	任意	説明
listener	**CodeRef<TelemetryEventListener>**	いいえ	テレメトリーイベントをリッスンします

console.topology/adapter/build
BuildAdapter は、Build コンポーネントで使用できるデータに要素を適応させるアダプターを追加します。

名前	値のタイプ	任意	説明
adapt	**CodeRef<(element: GraphElement) ⇒ AdapterDataType<BuildConfigData> | undefined>**	いいえ	Build コンポーネントで使用できるデータに要素を適応させるアダプター。

console.topology/adapter/network
NetworkAdapater は、**Networking** コンポーネントで使用できるデータに要素を適応させるアダプターを提供します。

名前	値のタイプ	任意	説明
adapt	**CodeRef<(element: GraphElement) ⇒ NetworkAdapterType | undefined>**	いいえ	Networking コンポーネントで使用できるデータに要素を適応させるアダプター。

console.topology/adapter/pod
PodAdapter はアダプターを提供し、**Pod** コンポーネントで使用できるデータに要素を適合させます。

名前	値のタイプ	任意	説明
adapt	**CodeRef<(element: GraphElement) ⇒ AdapterDataType<PodsAdapterDataType> | undefined>**	いいえ	Pod コンポーネントで使用できるデータに要素を適応させるアダプター。

console.topology/component/factory
ViewComponentFactory の Getter。

名前	値のタイプ	任意	説明
getFactory	**CodeRef<ViewComponentFactory>**	いいえ	**ViewComponentFactory** の Getter。

console.topology/create/connector
コネクター作成関数の getter。

名前	値のタイプ	任意	説明
getCreateConnector	**CodeRef<CreateConnectionGetter>**	いいえ	コネクター作成関数の getter。

console.topology/data/factory
トポロジーデータモデルファクトリーエクステンション

名前	値のタイプ	任意	説明
id	**string**	いいえ	ファクトリーの一意の ID。
priority	**number**	いいえ	ファクトリーの優先度
resources	**WatchK8sResourcesGeneric**	はい	**useK8sWatchResources** フックから取得されるリソース。
workloadKeys	**string[]**	はい	ワークロードが含まれるリソースのキー。
getDataModel	**CodeRef<TopologyDataModelGetter>**	はい	データモデルファクトリーの Getter。

名前	値のタイプ	任意	説明
isResourceDepicted	**CodeRef<TopologyDataModelDepicted>**	はい	リソースがこのモデルファクトリーによって記述されているかどうかを判断する関数の Getter。
getDataModelReconciler	**CodeRef<TopologyDataModelReconciler>**	はい	すべてのエクステンションのモデルがロードされた後にデータモデルを調整する関数の Getter。

console.topology/decorator/provider
トポロジーデコレータープロバイダーエクステンション

名前	値のタイプ	任意	説明
id	**string**	いいえ	エクステンション固有のトポロジーデコレーターの ID
priority	**number**	いいえ	エクステンション固有のトポロジーデコレーターの優先順位
quadrant	**TopologyQuadrant**	いいえ	エクステンション固有のトポロジーデコレーターのクアドラント
decorator	**CodeRef<TopologyDecoratorGetter>**	いいえ	エクステンション固有のデコレーター

console.topology/details/resource-alert
DetailsResourceAlert は、特定のトポロジーコンテキストまたはグラフ要素のアラートを提供します。

名前	値のタイプ	任意	説明
id	**string**	いいえ	このアラートの ID。アラートの破棄後に表示しない場合に状態を保存するために使用されます。

名前	値のタイプ	任意	説明
contentProvider	**CodeRef<(element: GraphElement) ⇒ DetailsResourceAlertContent | null>**	いいえ	アラートの内容を返すフック。

console.topology/details/resource-link
DetailsResourceLink は、特定のトポロジーコンテキストまたはグラフ要素のリンクを提供します。

名前	値のタイプ	任意	説明
link	**CodeRef<(element: GraphElement) ⇒ React.Component | undefined>**	いいえ	指定された場合はリソースリンクを返し、指定されない場合は未定義を返します。スタイルには **ResourceIcon** および **ResourceLink** プロパティーを使用します。
priority	**number**	はい	優先度の高いファクトリーからリンクを作成します。

console.topology/details/tab
DetailsTab は、トポロジーの詳細パネルのタブを提供します。

名前	値のタイプ	任意	説明
id	**string**	いいえ	この詳細タブの一意の識別子。
label	**string**	いいえ	UI に表示されるタブのラベル。
insertBefore	**string | string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。
insertAfter	**string | string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** 値が優先されます。

console.topology/details/tab-section
DetailsTabSection は、トポロジーの詳細パネルの特定タブのセクションを提供します。

名前	値のタイプ	任意	説明
id	**string**	いいえ	この詳細タブセクションの一意の識別子。
tab	**string**	いいえ	このセクションが提供する必要のある親タブ ID。
provider	**CodeRef<DetailsTabSectionExtensionHook>**	いいえ	コンポーネントを返すフック、または null か未定義の場合、トポロジーサイドバーにレンダリングされます。SDK コンポーネント: **<Section title=\{}>…** パディング領域
section	**CodeRef<(element: GraphElement, renderNull?: () ⇒ null) ⇒ React.Component | undefined>**	いいえ	非推奨: プロバイダーが定義されていない場合はフォールバックします。renderNull はすでに no-op です。
insertBefore	**string | string[]**	はい	ここで参照される項目の前に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。
insertAfter	**string | string[]**	はい	ここで参照される項目の後に、この項目を挿入します。配列の場合は、最初に見つかったものが順番に使用されます。**insertBefore** 値が優先されます。

console.topology/display/filters
トポロジー表示フィルターエクステンション

名前	値のタイプ	任意	説明
getTopologyFilters	**CodeRef<() ⇒ TopologyDisplayOption[]>**	いいえ	エクステンション固有のトポロジーフィルターのゲッター

名前	値のタイプ	任意	説明
applyDisplayOptions	**CodeRef<TopologyApplyDisplayOptions>**	いいえ	モデルにフィルターを適用する関数

console.topology/relationship/provider
トポロジー関係プロバイダーコネクターエクステンション

名前	値のタイプ	任意	説明
provides	**CodeRef<RelationshipProviderProvides>**	いいえ	ソースノードとターゲットノード間に接続を作成できるか判断するために使用
ヒント	**string**	いいえ	コネクター操作がドロップターゲット上に移動したときに表示されるツールヒント (例: "Create a Visual Connector")
create	**CodeRef<RelationshipProviderCreate>**	いいえ	接続を作成するためにコネクターがターゲットノード上にドロップされると実行されるコールバック
priority	**number**	いいえ	関係の優先順位。複数の場合は高い方が優先されます

console.user-preference/group
このエクステンションを使用して、console user-preferences ページにグループを追加できます。console user-preferences ページの垂直タブのオプションとして表示されます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	ユーザー設定グループを識別するのに使用されるID。
label	**string**	いいえ	ユーザー設定グループのラベル
insertBefore	**string**	はい	このユーザー設定グループの後に配置しなければならないグループの ID

名前	値のタイプ	任意	説明
insertAfter	**string**	はい	このユーザー設定グループの前に配置しなければならないグループの ID

console.user-preference/item
このエクステンションを使用して、console user-preferences ページのユーザー設定グループに項目を追加できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	ユーザー設定項目を特定するために使用され、項目の順序を定義するために insertAfter および insertBefore で参照される ID
label	**string**	いいえ	ユーザー設定のラベル
description	**string**	いいえ	ユーザー設定の説明
field	**UserPreferenceField**	いいえ	ユーザー設定を定義するために値をレンダリングするために使用される入力フィールドのオプション
groupId	**string**	はい	項目が属するユーザー優先グループを識別するために使用される ID
insertBefore	**string**	はい	このユーザー設定項目の後に配置しなければならない項目の ID
insertAfter	**string**	はい	このユーザー設定項目の前に配置しなければならない項目の ID

console.yaml-template
yaml エディターを使用してリソースを編集するための YAML テンプレート。

名前	値のタイプ	任意	説明

名前	値のタイプ	任意	説明
model	**ExtensionK8sModel**	いいえ	テンプレートに関連付けられたモデル。
template	**CodeRef<string>**	いいえ	YAML テンプレート。
name	**string**	いいえ	テンプレートの名前。名前 **default** を使用して、これをデフォルトテンプレートと識別します。

dev-console.add/action
このエクステンションを使用すると、プラグインは開発者パースペクティブの add ページに追加アクション項目を追加できます。たとえば、Serverless プラグインは、開発者コンソールの add ページにサーバーレス関数の新しい追加項目を追加できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	アクションを識別するための ID。
label	**string**	いいえ	アクションのラベル。
description	**string**	いいえ	アクションの説明。
href	**string**	いいえ	移動先の **href**。
groupId	**string**	はい	アクションが属するアクショングループを識別するのに使用される ID。
icon	**CodeRef<React.ReactNode>**	はい	パースペクティブの表示アイコン。
accessReview	**AccessReviewResourceAttributes[]**	はい	アクションの可視性または有効化を制御するオプションのアクセスレビュー。

dev-console.add/action-group
この拡張機能を使用すると、プラグインは開発者コンソールの add ページにグループを追加できます。グループはアクションが参照でき、アクションはエクステンションの定義に基づき add action ページでグループ化されます。たとえば、Serverless プラグインは、Serverless グループと複数の追加アクションを追加できます。

名前	値のタイプ	任意	説明
id	**string**	いいえ	アクショングループを識別するために使用される ID
name	**string**	いいえ	アクショングループのタイトル
insertBefore	**string**	はい	このアクショングループの後に配置しなければならないグループの ID
insertAfter	**string**	はい	このアクショングループの前に配置しなければならないグループの ID

dev-console.import/environment
このエクステンションを使用すると、開発者コンソール git インポートフォームのビルダーイメージセレクターで追加のビルド環境変数フィールドを指定できます。これを設定すると、フィールドはビルドセクション内の同じ名前の環境変数をオーバーライドします。

名前	値のタイプ	任意	説明
imageStreamName	**string**	いいえ	カスタム環境変数を指定するイメージストリームの名前
imageStreamTags	**string[]**	いいえ	サポートされるイメージストリームタグのリスト
environments	**ImageEnvironment[]**	いいえ	環境変数のリスト

console.dashboards/overview/detail/item
非推奨になりました。代わりに **CustomOverviewDetailItem** タイプを使用してください。

名前	値のタイプ	任意	説明
component	**CodeRef<React.ComponentType<{}>>**	いいえ	**DetailItem** コンポーネントに基づく値

console.page/resource/tab
非推奨。代わりに **console.tab/horizontalNav** を使用してください。Console ルーターに新しいリソースタブページを追加します。

名前	値のタイプ	任意	説明
model	**ExtensionK8sGroupKindModel**	いいえ	このリソースページがリンクするモデル。
component	**CodeRef<React.ComponentType<RouteComponentProps<{}, StaticContext, any>>>**	いいえ	ルートがマッチしたときにレンダリングされるコンポーネント。
name	**string**	いいえ	タブの名前。
href	**string**	はい	タブリンクのオプション **href**。指定しない場合は、最初の **path** が使用されます。
exact	**boolean**	はい	true の場合、パスが **location.pathname** と完全に一致する場合にのみマッチします。

4.5.2. 動的プラグイン API

useActivePerspective
現在アクティブなパースペクティブとアクティブなパースペクティブを設定するためのコールバックを提供するフック。現在アクティブなパースペクティブとセッターコールバックを含むタプルを返します。

例

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

GreenCheckCircleIcon
緑色のチェックマークの円形アイコンを表示するためのコンポーネント。

例

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

パラメーター名	説明
className	(オプション) コンポーネントの追加クラス名
title	(オプション) アイコンのタイトル
size	(オプション) アイコンのサイズ: (**sm**、**md**、**lg**、**xl**)

RedExclamationCircleIcon
赤い感嘆符の円形アイコンを表示するためのコンポーネント。

例

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

パラメーター名	説明
className	(オプション) コンポーネントの追加クラス名
title	(オプション) アイコンのタイトル
size	(オプション) アイコンのサイズ: (**sm**、**md**、**lg**、**xl**)

YellowExclamationTriangleIcon
黄色の三角形の感嘆符アイコンを表示するためのコンポーネント。

例

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

パラメーター名	説明
className	(オプション) コンポーネントの追加クラス名
title	(オプション) アイコンのタイトル
size	(オプション) アイコンのサイズ: (**sm**、**md**、**lg**、**xl**)

BlueInfoCircleIcon
青い情報円形アイコンを表示するためのコンポーネント。

例

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

パラメーター名	説明
className	(オプション) コンポーネントの追加クラス名
title	(オプション) アイコンのタイトル
size	(オプション) アイコンのサイズ: ('sm'、'md'、'lg'、'xl')

ErrorStatus
エラーステータスのポップオーバーを表示するためのコンポーネント。

例

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

パラメーター名	説明
title	(オプション) ステータステキスト
iconOnly	(オプション) true の場合、アイコンのみを表示します
noTooltip	(オプション) true の場合、ツールチップは表示されません
className	(オプション) コンポーネントの追加クラス名
popoverTitle	(オプション) ポップオーバーのタイトル

InfoStatus
情報ステータスのポップオーバーを表示するためのコンポーネント。

例

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

パラメーター名	説明
title	(オプション) ステータステキスト
iconOnly	(オプション) true の場合、アイコンのみを表示します
noTooltip	(オプション) true の場合、ツールチップは表示されません

パラメーター名	説明
className	(オプション) コンポーネントの追加クラス名
popoverTitle	(オプション) ポップオーバーのタイトル

ProgressStatus
進行状況のポップオーバーを表示するためのコンポーネント。

例

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

パラメーター名	説明
title	(オプション) ステータステキスト
iconOnly	(オプション) true の場合、アイコンのみを表示します
noTooltip	(オプション) true の場合、ツールチップは表示されません
className	(オプション) コンポーネントの追加クラス名
popoverTitle	(オプション) ポップオーバーのタイトル

SuccessStatus
成功ステータスのポップオーバーを表示するためのコンポーネント。

例

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

パラメーター名	説明
title	(オプション) ステータステキスト
iconOnly	(オプション) true の場合、アイコンのみを表示します
noTooltip	(オプション) true の場合、ツールチップは表示されません
className	(オプション) コンポーネントの追加クラス名

パラメーター名	説明
popoverTitle	(オプション) ポップオーバーのタイトル

checkAccess
特定のリソースへのユーザーアクセスに関する情報を提供します。リソースアクセス情報を含むオブジェクトを返します。

パラメーター名	説明
resourceAttributes	アクセスレビューのリソース属性
impersonate	権限借用の詳細

useAccessReview
特定のリソースへのユーザーアクセスに関する情報を提供するフック。**isAllowed** と **loading** 値を含む配列を返します。

パラメーター名	説明
resourceAttributes	アクセスレビューのリソース属性
impersonate	権限借用の詳細

useResolvedExtensions
解決された **CodeRef** プロパティーで Console 拡張機能を使用するための React フック。このフックは、**useExtensions** フックと同じ引数を受け入れ、拡張インスタンスの適合したリストを返し、各拡張のプロパティー内のすべてのコード参照を解決します。

最初に、フックは空の配列を返します。解決が完了すると、React コンポーネントが再レンダリングされ、適合した拡張機能のリストが返されます。一致する拡張子のリストが変更されると、解決が再開されます。フックは解決が完了するまで前の結果を返し続けます。

フックの結果要素は、再レンダリング全体で参照的に安定していることが保証されています。解決されたコード参照、解決が完了したかどうかを示すブール値フラグ、および解決中に検出されたエラーのリストを含む適応拡張インスタンスのリストを含むタプルを返します。

例

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

パラメーター名	説明

パラメーター名	説明
typeGuards	それぞれが動的プラグイン拡張機能を引数として受け入れ、拡張機能が目的の型制約を満たしているかどうかを示すブール値フラグを返すコールバックのリスト

HorizontalNav
ページのナビゲーションバーを作成するコンポーネント。ルーティングはコンポーネントの一部として処理されます。**console.tab/horizontalNav** を使用すると、水平ナビゲーションにコンテンツを追加できます。

例

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

パラメーター名	説明
resource	K8sResourceCommon タイプのオブジェクトである、このナビゲーションに関連付けられたリソース
pages	ページオブジェクトの配列
match	React Router が提供する match オブジェクト

VirtualizedTable
仮想化されたテーブルを作成するためのコンポーネント。

例

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

パラメーター名	説明
data	テーブルのデータ
loaded	データがロードされたことを示すフラグ
loadError	データのロードで問題が発生した場合のエラーオブジェクト
列	列の設定
行	行の設定
unfilteredData	フィルターなしの元のデータ
NoDataEmptyMsg	(オプション) データのない空のメッセージコンポーネント
EmptyMsg	(オプション) 空のメッセージコンポーネント
scrollNode	(オプション) スクロールを処理する関数
label	(オプション) テーブルのラベル
ariaLabel	(オプション) aria ラベル
gridBreakPoint	応答性のためにグリッドを分割する方法のサイジング
onSelect	(オプション) テーブルの選択を処理する関数
rowData	(オプション) 行に固有のデータ

TableData
テーブル行内にテーブルデータを表示するためのコンポーネント。

例

```
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>
```

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

パラメーター名	説明
id	テーブルの一意の ID
activeColumnIDs	アクティブな列
className	(オプション) スタイリングのオプションクラス名

useActiveColumns
ユーザーが選択したアクティブな TableColumns のリストを提供するフック。

例

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

パラメーター名	説明
options	キーと値のマップとして渡されるもの。
\{TableColumn[]} options.columns	使用可能なすべての TableColumn の配列
{boolean} [options.showNamespaceOverride]	(オプション) true の場合、列管理の選択に関係なく、名前空間列が含まれます。
{string} [options.columnManagementID]	(オプション) ユーザー設定との間で列管理の選択を保持および取得するために使用される一意の ID。通常は、リソースのグループ/バージョン/種類 (GVK) の文字列です。

現在のユーザーが選択したアクティブな列 (options.columns のサブセット) と、ユーザー設定がロードされたかどうかを示すブール値フラグを含むタプル。

ListPageHeader
ページヘッダーを生成するためのコンポーネント。

例

```
const exampleList: React.FC = () => {
  return (
```

```
    <>
      <ListPageHeader title="Example List Page"/>
    </>
  );
};
```

パラメーター名	説明
title	見出しタイトル
helpText	(オプション) 反応ノードとしてのヘルプセクション
badge	(オプション) 反応ノードとしてのバッジアイコン

ListPageCreate
特定のリソースの種類に対して、そのリソースの作成用 YAML へのリンクを自動的に生成する作成ボタンを追加するためのコンポーネント。

例

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

パラメーター名	説明
groupVersionKind	表すためのリソースグループ/バージョン/種類

ListPageCreateLink
定型化されたリンクを作成するためのコンポーネント。

例

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

パラメーター名	説明
to	リンク先の文字列の場所
createAccessReview	(オプション) アクセスを決定するために使用される namespace と種類を持つオブジェクト
children	(オプション) コンポーネントの子

ListPageCreateButton
ボタンを作成するためのコンポーネント。

例

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

パラメーター名	説明
createAccessReview	(オプション) アクセスを決定するために使用される namespace と種類を持つオブジェクト
pfButtonProps	(オプション) Patternfly Button のプロパティー

ListPageCreateDropdown
権限チェックでラップされたドロップダウンを作成するためのコンポーネント。

例

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

パラメーター名	説明
items	key: ドロップダウンコンポーネントに表示する項目の ReactNode のペア
onClick	ドロップダウン項目をクリックするためのコールバック関数
createAccessReview	(オプション) アクセスを決定するために使用される namespace と種類を持つオブジェクト
children	(オプション) ドロップダウントグルの子

ListPageFilter
リストページのフィルターを生成するコンポーネント。

例

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

パラメーター名	説明
data	データポイントの配列
loaded	データがロードされたことを示します
onFilterChange	フィルター更新時のコールバック関数
rowFilters	(オプション) 利用可能なフィルターオプションを定義する RowFilter 要素の配列
nameFilterPlaceholder	(オプション) 名前フィルターのプレースホルダー

パラメーター名	説明
labelFilterPlaceholder	(オプション) ラベルフィルターのプレースホルダー
hideLabelFilter	(オプション) 名前フィルターとラベルフィルターの両方ではなく、名前フィルターのみを表示します。
hideNameLabelFilter	(オプション) 名前フィルターとラベルフィルターの両方を非表示にします。
columnLayout	(オプション) 列レイアウトオブジェクト
hideColumnManagement	(オプション) 列管理を非表示にするフラグ

useListPageFilter
ListPageFilter コンポーネントのフィルター状態を管理するフック。すべての静的フィルターによってフィルター処理されたデータ、すべての静的フィルターと行フィルターによってフィルター処理されたデータ、および rowFilters を更新するコールバックを含むタプルを返します。

例

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

パラメーター名	説明
data	データポイントの配列
rowFilters	(オプション) 利用可能なフィルターオプションを定義する RowFilter 要素の配列
staticFilters	(オプション) データに静的に適用される FilterValue 要素の配列

ResourceLink
アイコンバッジを使用して特定のリソースタイプへのリンクを作成するコンポーネント。

例

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

パラメーター名	説明
kind	(オプション) リソースの種類、つまり Pod、Deployment、Namespace
groupVersionKind	(オプション) グループ、バージョン、および種類を含むオブジェクト
className	(オプション) コンポーネントのクラススタイル
displayName	(オプション) コンポーネントの表示名。設定されている場合は、リソース名を上書きします。
inline	(オプション) アイコンバッジを作成し、子とインラインで名前を付けるためのフラグ
linkTo	(オプション) Link オブジェクトを作成するためのフラグ - デフォルトは true
name	(オプション) リソースの名前
namesapce	(オプション) リンク先の種類のリソースの特定の namespace
hideIcon	(オプション) アイコンバッジを非表示にするフラグ
title	(オプション) リンクオブジェクトのタイトル (非表示)
dataTest	(オプション) テスト用の識別子
onClick	(オプション) コンポーネントがクリックされたときのコールバック関数
truncate	(オプション) リンクが長すぎる場合に切り捨てるフラグ

ResourceIcon

特定のリソースタイプのアイコンバッジを作成するコンポーネント。

例

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

パラメーター名	説明
kind	(オプション) リソースの種類、つまり Pod、Deployment、Namespace
groupVersionKind	(オプション) グループ、バージョン、および種類を含むオブジェクト
className	(オプション) コンポーネントのクラススタイル

useK8sModel
指定された K8sGroupVersionKind の k8s モデルを redux から取得するフック。最初の項目が k8s モデル、2 番目の項目が **inFlight** ステータスの配列を返します。

例

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

パラメーター名	説明
groupVersionKind	k8s リソースのグループ、バージョン、種類。K8sGroupVersionKind が推奨されます。もしくは、グループ、バージョン、種類の参照 (例: group/version/kind (GVK) K8sResourceKindReference.) を渡すこともできますが、これは非推奨です。

useK8sModels
redux から現在のすべての k8s モデルを取得するフック。最初の項目が k8s モデルのリストで、2 番目の項目が **inFlight** ステータスの配列を返します。

例

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

useK8sWatchResource

ロード済みおよびエラーのステータスとともに k8s リソースを取得するフック。最初の項目がリソース、2 番目の項目がロード済みステータス、3 番目の項目がエラー状態 (存在する場合) の配列を返します。

例

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

パラメーター名	説明
initResource	リソースを監視するために必要なオプション。

useK8sWatchResources
ロード済みおよびエラーのそれぞれのステータスとともに k8s リソースを取得するフック。キーが initResouces で提供され、値が data、loaded、error の 3 つのプロパティーを持つマップを返します。

例

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

パラメーター名	説明
initResources	リソースはキーと値のペアとして監視する必要があります。ここで、キーはリソースに固有であり、値はそれぞれのリソースを監視するために必要なオプションです。

consoleFetch
コンソール固有のヘッダーを追加し、再試行とタイムアウトを可能にする **fetch** のカスタムラッパー。また、応答ステータスコードを検証し、適切なエラーを出力するか、必要に応じてユーザーをログアウトします。レスポンスに解決される promise を返します。

パラメーター名	説明
url	取得する URL

パラメーター名	説明
options	フェッチに渡すオプション
timeout	ミリ秒単位のタイムアウト

consoleFetchJSON
コンソール固有のヘッダーを追加し、再試行とタイムアウトを可能にする **fetch** のカスタムラッパー。また、応答ステータスコードを検証し、適切なエラーを出力するか、必要に応じてユーザーをログアウトします。応答を JSON オブジェクトとして返します。内部で **consoleFetch** を使用します。JSON オブジェクトとして応答に解決される promise を返します。

パラメーター名	説明
url	取得する URL
method	使用する HTTP メソッドデフォルトは GET です。
options	フェッチに渡すオプション
timeout	ミリ秒単位のタイムアウト
cluster	リクエストを行うクラスターの名前。デフォルトは、ユーザーが選択したアクティブなクラスターです

consoleFetchText
コンソール固有のヘッダーを追加し、再試行とタイムアウトを可能にする **fetch** のカスタムラッパー。また、応答ステータスコードを検証し、適切なエラーを出力するか、必要に応じてユーザーをログアウトします。応答をテキストとして返します。内部で **consoleFetch** を使用します。テキストとして応答に解決される promise を返します。

パラメーター名	説明
url	取得する URL
options	フェッチに渡すオプション
timeout	ミリ秒単位のタイムアウト
cluster	リクエストを行うクラスターの名前。デフォルトは、ユーザーが選択したアクティブなクラスターです

getConsoleRequestHeaders
redux の現在の状態を使用して、API 要求の偽装およびマルチクラスター関連のヘッダーを作成する関数。redux の状態に基づき、適切な偽装とクラスター要求ヘッダーを含むオブジェクトを返します。

パラメーター名	説明
targetCluster	指定された targetCluster で現在アクティブなクラスターをオーバーライドします

k8sGetResource
指定されたオプションに基づいて、クラスターからリソースを取得します。名前が指定されている場合は、1つのリソースが返されます。それ以外の場合は、モデルに一致するすべてのリソースが返されます。名前が指定されている場合、リソースを含む JSON オブジェクトとして応答に解決される promise を返します。それ以外の場合は、モデルに一致するすべてのリソースを返します。失敗した場合、promise は HTTP エラー応答で拒否されます。

パラメーター名	説明
options	マップでキーと値のペアとして渡されるもの。
options.model	k8s モデル
options.name	リソースの名前。指定されていない場合は、モデルに一致するすべてのリソースが検索されます。
options.ns	検索先の namespace。cluster-scoped リソースには指定しないでください。
options.path	指定されている場合はサブパスとして追加します
options.queryParams	URL に含めるクエリーパラメーター。
options.requestInit	使用する fetch init オブジェクト。これには、リクエストヘッダー、メソッド、リダイレクトなどを含めることができます。詳細は、Interface RequestInit を参照してください。

k8sCreateResource
指定されたオプションに基づいて、クラスター内にリソースを作成します。作成されたリソースの応答に解決される promise を返します。失敗した場合、promise は HTTP エラー応答で拒否されます。

パラメーター名	説明
options	マップでキーと値のペアとして渡されるもの。
options.model	k8s モデル
options.data	作成されるリソースのペイロード
options.path	指定されている場合はサブパスとして追加します

パラメーター名	説明
options.queryParams	URL に含めるクエリーパラメーター。

k8sUpdateResource
指定されたオプションに基づいて、クラスター内のリソース全体を更新します。クライアントが既存のリソースを完全に置き換える必要がある場合、k8sUpdate を使用できます。または、k8sPatch を使用して部分的な更新を実行することもできます。更新されたリソースの応答に解決される promise を返します。失敗した場合、promise は HTTP エラー応答で拒否されます。

パラメーター名	説明
options	マップでキーと値のペアとして渡されます
options.model	k8s モデル
options.data	更新する k8s リソースのペイロード
options.ns	検索先の namespace。cluster-scoped リソースには指定しないでください。
options.name	更新するリソース名。
options.path	指定されている場合はサブパスとして追加します
options.queryParams	URL に含めるクエリーパラメーター。

k8sPatchResource
指定されたオプションに基づいて、クラスター内の任意のリソースにパッチを適用します。クライアントが部分的な更新を実行する必要がある場合、k8sPatch を使用できます。または、k8sUpdate を使用して、既存のリソースを完全に置き換えることもできます。詳細は、Data Tracker を参照してください。パッチが適用されたリソースの応答に解決される promise を返します。失敗した場合、promise は HTTP エラー応答で拒否されます。

パラメーター名	説明
options	マップでキーと値のペアとして渡されるもの。
options.model	k8s モデル
options.resource	パッチを適用するリソース。
options.data	操作、パス、および値を含む既存のリソースにパッチを適用するデータのみ。
options.path	指定されている場合はサブパスとして追加します。

パラメーター名	説明
options.queryParams	URL に含めるクエリーパラメーター。

k8sDeleteResource
指定されたモデル、リソースに基づいて、クラスターからリソースを削除します。ガベージコレクションは **Foreground|Background** に基づいて機能し、指定されたモデルの propagationPolicy プロパティーで設定するか、json で渡すことができます。種類が Status のレスポンスに解決される promise を返します。失敗した場合、promise は HTTP エラー応答で拒否されます。

例

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

パラメーター名	説明
options	マップでキーと値のペアとして渡されるもの。
options.model	k8s モデル
options.resource	削除するリソース。
options.path	指定されている場合はサブパスとして追加します
options.queryParams	URL に含めるクエリーパラメーター。
options.requestInit	使用する fetch init オブジェクト。これには、リクエストヘッダー、メソッド、リダイレクトなどを含めることができます。詳細は、Interface RequestInit を参照してください。
options.json	指定されている場合はリソースのガベージコレクションを明示的に制御できます。指定されていない場合は、モデルの "propagationPolicy" がデフォルトになります。

k8sListResource
指定されたオプションに基づいて、リソースをクラスター内の配列として一覧表示します。レスポンスに解決される promise を返します。

パラメーター名	説明
options	マップでキーと値のペアとして渡されるもの。
options.model	k8s モデル

パラメーター名	説明
options.queryParams	URL に含めるクエリーパラメーター。ラベルセレクターおよび "labelSelector" キーと併せて渡すことができます。
options.requestInit	使用する fetch init オブジェクト。これには、リクエストヘッダー、メソッド、リダイレクトなどを含めることができます。詳細は、Interface RequestInit を参照してください。

k8sListResourceItems
k8sListResource と同じインターフェイスですが、サブ項目を返します。モデルの apiVersion、つまり **group/version** を返します。

getAPIVersionForModel
k8s モデルの apiVersion を提供します。

パラメーター名	説明
model	k8s モデル

getGroupVersionKindForResource
リソースのグループ、バージョン、および種類を提供します。指定されたリソースのグループ、バージョン、種類を返します。リソースに API グループがない場合、グループ "core" が返されます。リソースの apiVersion が無効な場合は、エラーが出力されます。

パラメーター名	説明
resource	k8s リソース

getGroupVersionKindForModel
k8s モデルのグループ、バージョン、および種類を提供します。これは、提供されたモデルのグループ、バージョン、種類を返します。モデルに apiGroup がない場合は、グループ "core" が返されます。

パラメーター名	説明
model	k8s モデル

StatusPopupSection
ポップアップウィンドウでステータスを表示するコンポーネント。**console.dashboards/overview/health/resource** 拡張機能を構築するための便利なコンポーネント。

例

-

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

パラメーター名	説明
firstColumn	ポップアップの最初の列の値
secondColumn	(オプション) ポップアップの 2 列目の値
children	(オプション) ポップアップの子

StatusPopupItem
ステータスポップアップで使用されるステータス要素。**StatusPopupSection** で使用されます。

例

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

パラメーター名	説明
value	(オプション) 表示するテキスト値
icon	(オプション) 表示するアイコン
children	子要素

概要
ダッシュボードのラッパーコンポーネントを作成します。

例

-

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

パラメーター名	説明
className	(オプション) div のスタイルクラス
children	(オプション) ダッシュボードの要素

OverviewGrid
ダッシュボードのカード要素のグリッドを作成します。**Overview** 内で使用されます。

例

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

パラメーター名	説明
mainCards	グリッド用カード
leftCards	(オプション) グリッドの左側のカード
rightCards	(オプション) グリッドの右側のカード

InventoryItem
インベントリーカード項目を作成します。

例

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

パラメーター名	説明
children	項目内でレンダリングする要素

InventoryItemTitle
インベントリーカード項目のタイトルを作成します。**InventoryItem** 内で使用されます。

例

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

パラメーター名	説明
children	タイトル内にレンダリングする要素

InventoryItemBody
インベントリーカードの本文を作成します。**InventoryCard** 内で使用され、**InventoryTitle** と使用できます。

例

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

パラメーター名	説明
children	インベントリーカードまたはタイトル内でレンダリングする要素
error	div の要素

InventoryItemStatus
オプションのリンクアドレスを使用してインベントリーカードのカウントとアイコンを作成します。**InventoryItemBody** 内で使用されます。

例

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

```
    </InventoryItemBody>
  </InventoryItem>
)
```

パラメーター名	説明
count	表示用カウント
icon	表示用アイコン
linkTo	(オプション) リンクアドレス

InventoryItemLoading
インベントリーカードのロード時にスケルトンコンテナーを作成します。**InventoryItem** および関連コンポーネントで使用されます。

例

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

useFlag
FLAGS redux 状態から指定された機能フラグを返すフック。要求された機能フラグまたは未定義のブール値を返します。

パラメーター名	説明
flag	返す機能フラグ

CodeEditor
ホバーヘルプと補完機能を備えた基本的な遅延ロード Code エディター。

例

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

パラメーター名	説明
value	レンダリングする yaml コードを表す文字列。
言語	エディターの言語を表す文字列。
options	Monaco エディターのオプション。詳細は、インターフェイス IStandAloneEditorConstructionOptions を参照してください。
minHeight	有効な CSS の高さの値における最小のエディターの高さ。
showShortcuts	エディターの上にショートカットを表示するためのブール値。
toolbarLinks	エディター上部のツールバーリンクセクションにレンダリングされる ReactNode の配列。
onChange	コード変更イベントのコールバック。
onSave	コマンド CTRL / CMD + S がトリガーされたときに呼び出されるコールバック。
ref	**{ editor?: IStandaloneCodeEditor }** への参照に反応します。**editor** プロパティーを使用すると、エディターを制御するすべてのメソッドにアクセスできます。詳細は、インターフェイス IStandaloneCodeEditor を参照してください。

ResourceYAMLEditor
ホバーヘルプと補完機能を備えた Kubernetes リソース用の遅延ロード YAML エディター。このコンポーネントは YAMLEditor を使用し、その上にリソースの更新処理、アラート、保存、キャンセル、リロードボタン、アクセシビリティなどの機能を追加します。**onSave** コールバックが指定されないかぎり、リソースの更新は自動的に処理されます。**React.Suspense** コンポーネントでラップする必要があります。

例

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

パラメーター名	説明
initialResource	エディターによって表示されるリソースを表す YAML/オブジェクト。この prop は、最初のレンダリング中にのみ使用されます
header	YAML エディターの上にヘッダーを追加する
onSave	Save ボタンのコールバック。これを渡すと、エディターによってリソースに対して実行されるデフォルトの更新が上書きされます。

ResourceEventStream
特定のリソースに関連するイベントを表示するコンポーネント。

例

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

パラメーター名	説明
resource	関連イベントを表示するオブジェクト。

usePrometheusPoll
単一のクエリーに対して Prometheus へのポーリングを設定します。クエリー応答、応答が完了したかどうかを示すブール値フラグ、および要求中または要求の後処理中に発生したエラーを含むタプルを返します。

パラメーター名	説明
{PrometheusEndpoint} props.endpoint	PrometheusEndpoint (ラベル、クエリー、範囲、ルール、ターゲット) のいずれか
{string} [props.query]	(オプション) Prometheus クエリー文字列。空または未定義の場合、ポーリングは開始されません。
{number} [props.delay]	(オプション) ポーリング遅延間隔 (ミリ秒)
{number} [props.endTime]	(オプション) QUERY_RANGE エンドポイントの場合、クエリー範囲の終わり
{number} [props.samples]	(オプション) QUERY_RANGE エンドポイント用
{number} [options.timespan]	(オプション) QUERY_RANGE エンドポイント用
{string} [options.namespace]	(オプション) 追加する検索パラメーター

パラメーター名	説明
{string} [options.timeout]	(オプション) 追加する検索パラメーター

Timestamp
タイムスタンプをレンダリングするコンポーネント。タイムスタンプは、Timestamp コンポーネントの個々のインスタンス間で同期されます。指定されたタイムスタンプは、ユーザーロケールに従ってフォーマットされます。

パラメーター名	説明
timestamp	レンダリングするタイムスタンプ。形式は、ISO 8601 (Kubernetes で使用)、エポックタイムスタンプ、または日付のインスタンスであることが期待されます。
simple	アイコンとツールチップを省略したシンプルなバージョンのコンポーネントをレンダリングします。
omitSuffix	接尾辞を省略して日付をフォーマットします。
className	コンポーネントの追加のクラス名。

useModal
モーダルを起動するためのフック。

例

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

ActionServiceProvider
console.action/provider 拡張タイプの他のプラグインからのコントリビューションを受け取ることを可能にするコンポーネント。

例

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

  ...

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

パラメーター名	説明
context	contextId とオプションのプラグインデータを含むオブジェクト

NamespaceBar
namespace のドロップダウンメニューが左端にある水平ツールバーをレンダリングするコンポーネント。追加のコンポーネントは子として渡すことができ、namespace ドロップダウンの右側にレンダリングされます。このコンポーネントは、ページの上部で使用するように設計されています。k8s リソースを含むページなど、ユーザーがアクティブな namespace を変更できる必要があるページで使用する必要があります。

例

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

...

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

  ...
```

パラメーター名	説明
onNamespaceChange	(オプション) namespace オプションが選択されたときに実行される関数。唯一の引数として、文字列の形式で新しい namespace を受け入れます。オプションが選択されると、アクティブな namespace が自動的に更新されますが、この関数を介して追加のロジックを適用できます。namespace が変更されると、URL の namespace パラメーターが以前の namespace から新しく選択された namespace に変更されます。
isDisabled	(オプション) true に設定されている場合、namespace のドロップダウンを無効にするブール値フラグ。このオプションは namespace ドロップダウンにのみ適用され、子コンポーネントには影響しません。
children	(オプション) namespace ドロップダウンの右側にあるツールバー内にレンダリングされる追加の要素。

ErrorBoundaryFallbackPage
フルページの ErrorBoundaryFallbackPage コンポーネントを作成して、"Oh no!Something went wrong." というメッセージと、スタックトレースおよびその他の役立つデバッグ情報を表示します。これは、コンポーネントと組み合わせて使用されます。

例

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

   return this.props.children;
)
```

パラメーター名	説明
errorMessage	エラーメッセージのテキスト説明
componentStack	例外のコンポーネントトレース
stack	例外のスタックトレース
title	エラー境界ページのヘッダーとしてレンダリングするタイトル

QueryBrowser
Prometheus PromQL クエリーからの結果のグラフを、グラフと対話するためのコントロールとともにレンダリングするコンポーネント。

例

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

パラメーター名	説明
customDataSource	(オプション) PromQL クエリーを処理する API エンドポイントのベース URL。指定した場合、これはデータをフェッチするためのデフォルト API の代わりに使用されます。

パラメーター名	説明
defaultSamples	(オプション) 各データ系列に対してプロットされるデータサンプルのデフォルトの数。データ系列が多い場合、QueryBrowser はここで指定した数よりも少ない数のデータサンプルを自動的に選択することがあります。
defaultTimespan	(オプション) グラフのデフォルトのタイムスパン (ミリ秒単位) - デフォルトは 1,800,000 (30 分) です。
disabledSeries	(オプション) これらの正確なラベルと値のペアを持つデータシリーズを無効にします (表示しません)。
disableZoom	(オプション) グラフのズームコントロールを無効にするフラグ。
filterLabels	(オプション) 必要に応じて、返されたデータ系列をこれらのラベルと値のペアに一致するデータ系列のみにフィルタリングします。
fixedEndTime	(オプション) 現在の時刻までのデータを表示するのではなく、表示される時間範囲の終了時刻を設定します。
formatSeriesTitle	(オプション) 単一のデータ系列のタイトルとして使用する文字列を返す関数。
GraphLink	(オプション) 別のページへのリンクをレンダリングするためのコンポーネント (たとえば、このクエリーに関する詳細情報を取得する)。
hideControls	(オプション) グラフのタイムスパンなどを変更するためのグラフコントロールを非表示にするフラグ。
isStack	(オプション) 折れ線グラフの代わりに積み上げグラフを表示するフラグ。showStackedControl が設定されている場合でも、ユーザーは折れ線グラフに切り替えることができます。
namespace	(オプション) 指定した場合、この namespace のデータのみが返されます (この namespace ラベルを持つシリーズのみ)。
onZoom	(オプション) グラフがズームされたときに呼び出されるコールバック。

パラメーター名	説明
pollInterval	(オプション) 設定すると、最新のデータを表示するためにグラフが更新される頻度 (ミリ秒単位) が決まります。
queries	実行して結果をグラフに表示する PromQL クエリーの配列。
showLegend	(オプション) グラフの下に凡例を表示できるようにするフラグ。
showStackedControl	積み上げグラフモードと折れ線グラフモードを切り替えるためのグラフコントロールの表示を有効にするフラグ。
timespan	(オプション) グラフがカバーするタイムスパン (ミリ秒単位)。
units	(オプション) Y 軸およびツールチップに表示する単位。

useAnnotationsModal
Kubernetes リソースのアノテーションを編集するためのモーダルを起動するコールバックを提供するフック。

例

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

パラメーター名	説明
resource	K8sResourceCommon タイプのオブジェクトのアノテーションを編集するためのリソース。

戻り値

リソースのアノテーションを編集するためのモーダルを起動する関数。

useDeleteModal
リソースを削除するためのモーダルを起動するコールバックを提供するフック。

例

```
const DeletePodButton = ({ pod }) => {
  const { t } = useTranslation();
```

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

パラメーター名	説明
resource	削除するリソース。
redirectTo	(オプション) リソースを削除した後にリダイレクトする場所。
message	(オプション) モーダルに表示するメッセージ。
btnText	(オプション) 削除ボタンに表示するテキスト。
deleteAllResources	(オプション) 同じ種類のリソースをすべて削除する機能。

戻り値

リソースを削除するためのモーダルを起動する関数。

useLabelsModel
Kubernetes リソースラベルを編集するためのモーダルを起動するコールバックを提供するフック。

例

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

パラメーター名	説明
resource	ラベルを編集するリソース (K8sResourceCommon タイプのオブジェクト)。

戻り値

リソースのラベルを編集するためのモーダルを起動する関数。

useActiveNamespace
現在アクティブな namespace と、アクティブな namespace を設定するためのコールバックを提供するフック。

例

```
const Component: React.FC = (props) => {
   const [activeNamespace, setActiveNamespace] = useActiveNamespace();
```

```
    return <select
      value={activeNamespace}
      onChange={(e) => setActiveNamespace(e.target.value)}
    >
      {
        // ...namespace options
      }
    </select>
  }
```

戻り値

現在アクティブな namespace とセッターコールバックを含むタプル。

useUserSettings
ユーザー設定値と、ユーザー設定値を設定するためのコールバックを提供するフック。

例

```
const Component: React.FC = (props) => {
  const [state, setState, loaded] = useUserSettings(
    'devconsole.addPage.showDetails',
    true,
    true,
  );
  return loaded ? (
    <WrappedComponent {...props} userSettingState={state} setUserSettingState={setState} />
   ) : null;
};
```

戻り値

ユーザー設定値、セッターコールバック、およびロードされたブール値を含むタプル。

useQuickStartContext
現在のクイックスタートコンテキスト値を提供するフック。これにより、プラグインがコンソールのクイックスタート機能と相互運用できるようになります。

例

```
const OpenQuickStartButton = ({ quickStartId }) => {
  const { setActiveQuickStart } = useQuickStartContext();
  const onClick = React.useCallback(() => {
     setActiveQuickStart(quickStartId);
  }, [quickStartId]);
  return <button onClick={onClick}>{t('Open Quick Start')}</button>
};
```

戻り値

クイックスタートコンテキスト値オブジェクト。

PerspectiveContext
非推奨: 代わりに、指定された **usePerspectiveContext** を使用してください。パースペクティブコンテキストを作成します。

パラメーター名	説明
PerspectiveContextType	アクティブなパースペクティブとセッターを含むオブジェクト

useAccessReviewAllowed
非推奨: 代わりに **@console/dynamic-plugin-sdk** の **useAccessReview** を使用してください。指定されたリソースへのユーザーアクセスに関する使用可能なステータスを指定するフック。**isAllowed** ブール値を返します。

パラメーター名	説明
resourceAttributes	アクセスレビューのリソース属性
impersonate	権限借用の詳細

useSafetyFirst
非推奨: このフックはコンソールの機能とは関係ありません。指定されたコンポーネントがアンマウントされた場合に備えて、React 状態の安全な非同期設定を保証するフック。状態値とその set 関数のペアを含む配列を返します。

パラメーター名	説明
initialState	初期状態値

YAMLEditor
非推奨: ホバーヘルプと補完を備えた基本的な遅延ロード YAML エディター。

例

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

パラメーター名	説明
value	レンダリングする yaml コードを表す文字列。
options	Monaco エディターのオプション。
minHeight	有効な CSS の高さの値における最小のエディターの高さ。
showShortcuts	エディターの上にショートカットを表示するためのブール値。

パラメーター名	説明
toolbarLinks	エディター上部のツールバーリンクセクションにレンダリングされる ReactNode の配列。
onChange	コード変更イベントのコールバック。
onSave	コマンド CTRL / CMD + S がトリガーされたときに呼び出されるコールバック。
ref	**{ editor?: IStandaloneCodeEditor }** への参照に反応します。**editor** プロパティーを使用すると、エディターを制御するすべてのメソッドにアクセスできます。

4.5.3. 動的プラグインのトラブルシューティング

プラグインのロードで問題が発生した場合は、このトラブルシューティングのヒントのリストを参照してください。

- 以下のコマンドを実行して、コンソールの Operator 設定でプラグインが有効になっており、プラグイン名が出力されていることを確認します。

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

  - **Administrator perspective** の **Overview** ページのステータスカードで、有効なプラグインを確認します。プラグインが最近有効になった場合は、ブラウザーを更新する必要があります。

- 次の方法で、プラグインサービスが正常であることを確認します。
  - プラグイン Pod のステータスが実行中であり、コンテナーの準備が整っていることを確認します。
  - サービスラベルセレクターが Pod と一致し、ターゲットポートが正しいことを確認します。
  - コンソール Pod またはクラスター上の別の Pod のターミナルで、サービスから **plugin-manifest.json** をカールします。
- **ConsolePlugin** リソース名 (**consolePlugin.name**) が **package.json** で使用されているプラグイン名と一致することを確認します。
- サービス名、namespace、ポート、およびパスが **ConsolePlugin** リソースで正しく宣言されていることを確認します。
- プラグインサービスが HTTPS とサービス提供証明書を使用していることを確認します。
- コンソール Pod ログで証明書または接続エラーを確認します。
- プラグインが依存する機能フラグが無効になっていないことを確認します。
- プラグインの **package.json** に一致しない **consolePlugin.dependencies** がないことを確認します。

    - これには、コンソールバージョンの依存関係または他のプラグインへの依存関係が含まれる場合があります。ブラウザーで JS コンソールをプラグインの名前でフィルタリングして、ログに記録されたメッセージを表示します。

- ナビゲーション拡張パースペクティブまたはセクション ID にタイプミスがないことを確認します。

    - プラグインはロードされている可能性がありますが、ID が正しくない場合、ナビゲーション項目が表示されません。URL を編集して、プラグインページに直接移動してみてください。

- コンソール Pod からプラグインサービスへのトラフィックをブロックしているネットワークポリシーがないことを確認します。

    - 必要に応じて、ネットワークポリシーを調整して、openshift-console namespace のコンソール Pod がサービスにリクエストを送信できるようにします。

- 開発者ツールブラウザーの **Console** タブで、ブラウザーにロードされる動的プラグインのリストを確認します。

    - **window.SERVER_FLAGS.consolePlugins** を評価して、コンソールフロントエンドの動的プラグインを確認します。

第5章 WEB 端末

5.1. WEB 端末のインストール

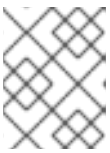
Web 端末は、OpenShift Dedicated Operator Hub に登録されている Web Terminal Operator を使用してインストールできます。Web 端末 Operator をインストールする際に、**DevWorkspace** CRD などのコマンドラインの設定に必要なカスタムリソース定義 (CRD) が自動的にインストールされます。Web コンソールでは、Web 端末を開く際に必要なリソースを作成します。

前提条件

- OpenShift Dedicated Web コンソールにログインしている。
- クラスター管理者パーミッションがある。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → OperatorHub** に移動します。
2. **Filter by keyword** ボックスを使用してカタログで Web Terminal Operator を検索し、**Web Terminal** タイルをクリックします。
3. **Web Terminal** ページで Operator に関する簡単な説明を確認してから、**Install** をクリックします。
4. **Install Operator** ページで、すべてのフィールドのデフォルト値を保持します。
   - **Update Channel** メニューの **fast** オプションは、Web 端末 Operator の最新リリースのインストールを可能にします。
   - **Installation Mode** メニューの **All namespaces on the cluster**オプションにより、Operator にクラスターのすべての namespace を監視され、Operator をこれらの namespace で利用可能にすることができます。
   - **Installed Namespace** メニューの **openshift-operators** オプションは、Operator をデフォルトの **openshift-operators** namespace にインストールします。
   - **Approval Strategy** メニューの **Automatic** オプションにより、Operator への今後のアップグレードは Operator Lifecycle Manager によって自動的に処理されます。
5. **Install** をクリックします。
6. **Installed Operators** ページで、**View Operator** をクリックし、Operator が **Installed Operators** ページにリスト表示されていることを確認します。



注記

Web Terminal Operator は、DevWorkspace Operator を依存関係としてインストールします。

7. Operator のインストール後に、ページを更新し、コンソールのマストヘッドにあるコマンドラインターミナルアイコン () を確認します。

5.2. WEB 端末の使用

Web コンソールで組み込みコマンドラインターミナルインスタンスを起動できます。この端末のインスタンスは、**oc**、**kubectl**、**odo**、**kn**、**tkn**、**helm**、**subctl** など、クラスターと対話するための一般的な CLI ツールと共に事前にインストールされます。また、これには作業しているプロジェクトのコンテキストが含まれ、ユーザーの認証情報を使用してユーザーのログインを自動的に行います。

5.2.1. Web 端末へのアクセス

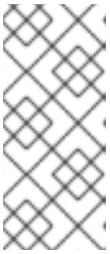
Web Terminal Operator をインストールすると、Web 端末にアクセスできます。Web 端末を初期化した後に、Web 端末で **oc**、**kubectl**、**odo**、**kn**、**tkn**、**helm**、**subctl** などの事前インストールされた CLI ツールを使用できます。ターミナルで実行したコマンドのリストからコマンドを選択して、コマンドを再実行することができます。これらのコマンドは、複数のターミナルセッション間で保持されます。Web 端末を閉じるまで、またはブラウザーウィンドウかタブを閉じるまで、Web 端末は表示されたままになります。

前提条件

- OpenShift Dedicated クラスターにアクセスでき、Web コンソールにログインしている。
- Web Terminal Operator がクラスターにインストールされている。

手順

1. Web 端末を起動するには、コンソールのマストヘッドにあるコマンドラインターミナルアイコン () をクリックします。Web 端末インスタンスが、**Command line terminal** ペインに表示されます。このインスタンスは、お使いの認証情報を使用して自動的にログインします。
2. 現在のセッションでプロジェクトが選択されていない場合は、**DevWorkspace** CR を作成する必要があるプロジェクトを **Project** ドロップダウンリストから選択します。デフォルトでは、現在のプロジェクトが選択されます。



注記

- 1 つの **DevWorkspace** CR は、1 ユーザーの Web 端末を定義します。この CR には、ユーザーの Web 端末ステータスおよびコンテナーイメージコンポーネントに関する詳細が含まれています。
- **DevWorkspace** CR は存在しない場合にのみ作成されます。

3. **Start** をクリックし、選択したプロジェクトを使用して Web 端末を初期化します。
4. **+** をクリックして、コンソールの Web 端末で複数のタブを開きます。

5.3. WEB 端末のトラブルシューティング

5.3.1. Web 端末とネットワークポリシー

クラスターにネットワークポリシーが設定されていると、Web 端末の起動に失敗する可能性があります。Web 端末のインスタンスを初期化するには、Web Terminal Operator が Web 端末の Pod と通信して Pod が実行中であることを確認する必要があります。また、OpenShift Dedicated の Web コンソー

ルが、端末内のクラスターへの自動ログイン情報を送信する必要があります。いずれのステップも失敗すると、Web 端末は起動に失敗し、**context deadline exceeded error** が発生するまで、端末パネルはロード状態になります。

この問題を回避するには、端末に使用される namespace のネットワークポリシーが **openshift-console** および **openshift-operators** namespace からの ingress を許可していることを確認してください。

以下のサンプルは、**openshift-console** および **openshift-operators** namespace からの Ingress を許可する **NetworkPolicy** オブジェクトを示しています。

openshift-console namespace からの Ingress の許可

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-console
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-console
  podSelector: {}
  policyTypes:
  - Ingress
```

openshift-operators namespace からの Ingress の許可

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-operators
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-operators
  podSelector: {}
  policyTypes:
  - Ingress
```

5.4. WEB 端末のアンインストール

Web Terminal Operator をアンインストールしても、Operator のインストール時に作成されるカスタムリソース定義 (CRD) または管理リソースは削除されません。セキュリティー上の理由から、これらのコンポーネントは手動でアンインストールする必要があります。これらのコンポーネントを削除すると、Operator をアンインストールしても端末はアイドル状態にならないため、クラスターリソースが保存されます。

Web 端末のアンインストールは 2 つの手順で実行されます。

1. Operator のインストール時に追加された Web 端末 Operator および関連するカスタムリソース (CR) をアンインストールします。
2. Web 端末 Operator の依存関係として追加された DevWorkspace Operator とそれに関連するカスタムリソースをアンインストールします。

5.4.1. Web Terminal Operator の削除

Web 端末をアンインストールするには、Operator が使用する Web Terminal Operator とカスタムリソースを削除します。

前提条件

- クラスター管理者権限を持つ OpenShift Dedicated クラスターにアクセスできる。
- **oc** CLI がインストールされている。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
2. フィルターリストをスクロールするか、**Filter by name** ボックスにキーワードを入力して Web Terminal Operator を見つけます。
3. Web Terminal Operator の Options メニュー をクリックし、**Uninstall Operator** を選択します。
4. **Uninstall Operator** 確認ダイアログボックスで、**Uninstall** をクリックし、Operator、Operator デプロイメント、および Pod をクラスターから削除します。この Operator は実行を停止し、更新を受信しなくなります。

5.4.2. DevWorkspace Operator の削除

Web 端末を完全にアンインストールするには、Operator が使用する DevWorkspace Operator とカスタムリソースも削除する必要があります。

重要

DevWorkspace Operator はスタンドアロン Operator であり、クラスターにインストールされている他の Operator の依存関係として必要になる場合があります。
DevWorkspace Operator が不要であることが確実な場合にのみ、以下の手順を実行してください。

前提条件

- クラスター管理者権限を持つ OpenShift Dedicated クラスターにアクセスできる。
- **oc** CLI がインストールされている。

手順

1. Operator が使用する **DevWorkspace** カスタムリソースと関連する Kubernetes オブジェクトを削除します。

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

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

> **警告**
>
> この手順が完了していない場合、ファイナライザーにより Operator を完全にアンインストールすることが困難になります。

2. 残りのサービス、シークレット、および設定マップを削除します。インストールによっては、以下のコマンドに含まれる一部のリソースがクラスターに存在しない場合があります。

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

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

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

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

3. DevWorkspace Operator をアンインストールします。
   a. Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
   b. フィルターリストをスクロールするか、**Filter by name** ボックスにキーワードを入力して DevWorkspace Operator を見つけます。
   c. Operator のオプションメニュー  をクリックし、**Uninstall Operator** を選択します。
   d. **Uninstall Operator** 確認ダイアログボックスで、**Uninstall** をクリックし、Operator、Operator デプロイメント、および Pod をクラスターから削除します。この Operator は実行を停止し、更新を受信しなくなります。

第6章 クイックスタートチュートリアルについて

OpenShift Dedicated Web コンソールのクイックスタートチュートリアルを作成する場合は、以下のガイドラインに従って、すべてのクイックスタートで一貫したユーザーエクスペリエンスを維持するようにしてください。

6.1. クイックスタートについて

クイックスタートは、ユーザータスクに関するガイド付きチュートリアルです。Web コンソールでは、**Help** メニューでクリックスタートにアクセスできます。これらは、アプリケーション、Operator、または他の製品オファリングを使用する場合に役立ちます。

クイックスタートは、主にタスクとステップで構成されます。タスクごとに複数のステップがあり、各クイックスタートには複数のタスクがあります。以下に例を示します。

- タスク 1
  - ステップ 1
  - ステップ 2
  - ステップ 3
- タスク 2
  - ステップ 1
  - ステップ 2
  - ステップ 3
- タスク 3
  - ステップ 1
  - ステップ 2
  - ステップ 3

6.2. クイックスタートのユーザーワークフロー

既存のクイックスタートチュートリアルと対話する場合、以下が想定されるワークフローエクスペリエンスになります。

1. **Administrator** または **Developer** パースペクティブで、**Help アイコン** をクリックし、**Quick Starts** を選択します。
2. クイックスタートカードをクリックします。
3. 表示されるパネルで **Start** をクリックします。
4. 画面上の手順を実行し、**Next** をクリックします。
5. 表示される **Check your work** モジュールで質問に回答し、タスクが正常に完了したことを確認します。
   a. **Yes** を選択した場合には、**Next** をクリックして次のタスクに進みます。

b. **No** を選択した場合は、タスクの手順を繰り返して作業を再度確認します。

6. 上記の手順 1 から 6 を繰り返し、クイックスタートの残りのタスクを完了します。
7. 最終タスクが完了したら、**Close** をクリックしてクイックスタートを閉じます。

6.3. クイックスタートのコンポーネント

クイックスタートは、以下のセクションで構成されます。

- **Card**: タイトル、説明、時間 (time commitment)、完了ステータスなどの、クイックスタートの基本情報を提供するカタログタイル
- **Introduction**: クイックスタートの目的およびタスクの概要
- **Task headings**: クイックスタートの各タスクのハイパーリンクタイトル
- **Check your work module** ユーザーがクイックスタートの次のタスクに進む前に、タスクが正常に完了したことを確認するためのモジュール
- **Hints**: ユーザーによる製品の特定の機能を識別するのに役立つアニメーション
- **Buttons**
  - **Next and back buttons** クイックスタートの各タスク内のステップおよびモジュールに移動するためのボタン
  - **Final screen buttons** クイックスタートを閉じたり、クイックスタート内の前のタスクに戻ったり、クイックスタートをすべて表示したりするためのボタン

クイックスタートの主なコンテンツエリアには、以下のセクションが含まれます。

- **Card copy**
- **はじめに**
- **タスクの手順**
- **Modals and in-app messaging**
- **作業モジュールの確認**