Red Hat Developer Hub 1.5

動的プラグインの設定

Red Hat Developer Hub での動的プラグインの設定

Last Updated: 2025-07-16

Red Hat Developer Hub 1.5 動的プラグインの設定

Red Hat Developer Hub での動的プラグインの設定

法律上の通知

概要

プラットフォームエンジニアは、RHDH で動的プラグインを設定して、開発インフラストラクチャーまたはソフトウェア開発ツールにアクセスできます。

目次

第1章 ANSIBLE PLUG-INS FOR RED HAT DEVELOPER HUB のインストール

Ansible plug-ins for Red Hat Developer Hub は、厳選されたラーニングパス、ボタン操作によるコンテンツ作成、統合開発ツール、その他の事前設定済みリソースを備えた Ansible 固有のポータルエクスペリエンスを提供します。

Ansible プラグインをインストールして設定するには、**Ansible plug-ins for Red Hat Developer Hub のインストール** を参照してください。

第2章 ARGO CD プラグインの有効化

Argo CD プラグインを使用すると、OpenShift GitOps の継続的デリバリー (CD) のワークフローを視覚化できます。このプラグインは、アプリケーションのステータス、デプロイメントの詳細、コミットメッセージ、コミットの作成者、環境にプロモートされたコンテナーイメージ、およびデプロイメントの履歴の概要を視覚的に提供します。

前提条件

- 次の例に示すように、Argo CD インスタンス情報を **app-config.yaml** 設定マップに追加します。

```
argocd:
  appLocatorMethods:
    - type: 'config'
      instances:
        - name: argoInstance1
          url: https://argoInstance1.com
          username: ${ARGOCD_USERNAME}
          password: ${ARGOCD_PASSWORD}
        - name: argoInstance2
          url: https://argoInstance2.com
          username: ${ARGOCD_USERNAME}
          password: ${ARGOCD_PASSWORD}
```

- Argo CD アプリケーションを特定するために、エンティティーの **catalog-info.yaml** ファイルに次のアノテーションを追加します。

```
annotations:
  ...
  # The label that Argo CD uses to fetch all the applications. The format to be used is
label.key=label.value. For example, rht-gitops.com/janus-argocd=quarkus-app.

  argocd/app-selector: '${ARGOCD_LABEL_SELECTOR}'
```

- (オプション) Argo CD インスタンスを切り替えるには、次の例に示すように、エンティティーの **catalog-info.yaml** ファイルに次のアノテーションを追加します。

```
annotations:
  ...
  # The Argo CD instance name used in `app-config.yaml`.

  argocd/instance-name: '${ARGOCD_INSTANCE}'
```

注記

このアノテーションを設定しなかった場合、Argo CD プラグインは、**app-config.yaml** で設定された最初の Argo CD インスタンスをデフォルトで使用します。

手順

1. dynamic-plugins ConfigMap に以下を追加して、Argo CD プラグインを有効にします。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/roadiehq-backstage-plugin-argo-cd-backend-dynamic
        disabled: false
      - package: ./dynamic-plugins/dist/backstage-community-plugin-redhat-argocd
        disabled: false
```

2.1. ARGO CD ロールアウトの有効化

オプションの Argo CD ロールアウト機能は、ブルーグリーンデプロイメントやカナリアデプロイメントなどの高度なデプロイメントストラテジーをアプリケーションに提供することで Kubernetes を強化します。バックステージ Kubernetes プラグインに統合すると、開発者と運用チームは、バックステージインターフェイス内で Argo CD ロールアウトをシームレスに視覚化および管理できるようになります。

前提条件

- Backstage Kubernetes プラグイン (**@backstage/plugin-kubernetes**) がインストールされ、設定されている。
  - Backstage で Kubernetes プラグインをインストールして設定するには、インストール および 設定 ガイドを参照してください。
- カスタムリソースと **ClusterRoles** の作成や管理に必要な権限があり、Kubernetes クラスターにアクセスできる。
- Kubernetes クラスターには、**argoproj.io** グループリソース (Rollouts や AnalysisRuns など) がインストールされている。

手順

1. Backstage インスタンスの **app-config.yaml** ファイルで、**kubernetes** 設定に次の **customResources** コンポーネントを追加して、Argo Rollouts と AnalysisRuns を有効にします。

```
kubernetes:
  ...
  customResources:
    - group: 'argoproj.io'
      apiVersion: 'v1alpha1'
      plural: 'Rollouts'
    - group: 'argoproj.io'
      apiVersion: 'v1alpha1'
      plural: 'analysisruns'
```

2. カスタムリソースに対する **ClusterRole** 権限を付与します。

注記

- Backstage Kubernetes プラグインがすでに設定されている場合は、Rollouts および AnalysisRuns の **ClusterRole** 権限がすでに付与されている可能性があります。
- 準備したマニフェスト を使用して、Kubernetes プラグインと ArgoCD プラグインの両方に読み取り専用の **ClusterRole** アクセスを割り当てます。

a. **ClusterRole** 権限が付与されていない場合は、次の YAML マニフェストを使用して **ClusterRole** を作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  - apiGroups:
      - argoproj.io
    resources:
      - rollouts
      - analysisruns
    verbs:
      - get
      - list
```

a. **kubectl** を使用してマニフェストをクラスターに適用します。

```
kubectl apply -f <your-clusterrole-file>.yaml
```

b. クラスターにアクセスする **ServiceAccount** にこの **ClusterRole** が割り当てられていることを確認します。

3. Backstage の Kubernetes リソースを識別するために、**catalog-info.yaml** にアノテーションを追加します。

a. エンティティー ID でリソースを識別する場合:

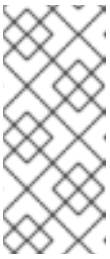
```
annotations:
  ...
  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

b. (オプション) 名前空間でリソースを識別する場合:

```
annotations:
  ...
  backstage.io/kubernetes-namespace: <RESOURCE_NAMESPACE>
```

c. エンティティー ID または名前空間によるリソース識別をオーバーライドするカスタムラベルセレクターを使用する場合:

```
annotations:
  ...
  backstage.io/kubernetes-label-selector: 'app=my-app,component=front-end'
```



注記

Kubernetes リソースの **backstage.io/kubernetes-label-selector** で宣言されたラベルを必ず指定してください。このアノテーションは、**backstage.io/kubernetes-id** や **backstage.io/kubernetes-namespace** などのエンティティーベースまたは名前空間ベースの識別アノテーションをオーバーライドします。

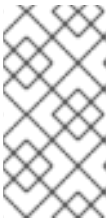
4. Backstage が適切な Kubernetes リソースを見つけられるように、Kubernetes リソースにラベルを追加します。

   a. Backstage Kubernetes プラグインラベル: このラベルを追加して、リソースを特定の Backstage エンティティーにマップします。

   ```
   labels:
     ...
     backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
   ```

   b. GitOps アプリケーションマッピング: このラベルを追加して、Argo CD ロールアウトを特定の GitOps アプリケーションにマッピングします。

   ```
   labels:
     ...
     app.kubernetes.io/instance: <GITOPS_APPLICATION_NAME>
   ```



注記

ラベルセレクターアノテーション (backstage.io/kubernetes-label-selector) を使用する場合は、指定されたラベルがリソースに存在することを確認します。ラベルセレクターは、kubernetes-id や kubernetes-namespace などの他のアノテーションをオーバーライドします。

検証

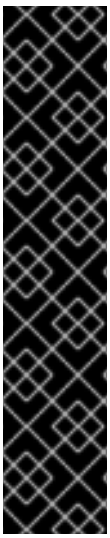
1. 更新された設定を GitOps リポジトリーにプッシュして、ロールアウトをトリガーします。
2. Red Hat Developer Hub インターフェイスを開き、設定したエンティティーに移動します。
3. **CD** タブを選択し、対象の **GitOps アプリケーション** を選択します。サイドパネルが開きます。
4. サイドパネルの **Resources** テーブルで、次のリソースが表示されていることを確認します。
   - Rollouts
   - AnalysisRuns (オプション)
5. ロールアウトリソースを展開し、次の詳細を確認します。
   - Revisions の行には、さまざまなロールアウトバージョンのトラフィックの分布が詳細に表示されます。
   - Analysis Runs の行には、ロールアウトの成功を評価する分析タスクのステータスが表示されます。

関連情報

- Red Hat ArgoCD プラグインのパッケージパス、スコープ、および名前は 1.2 以降変更されています。詳細は、**Red Hat Developer Hub リリースノート**の 互換性を失わせる変更点 を参照してください。
- 動的プラグインのインストールの詳細は、Red Hat Developer Hub でのプラグインのインストールと表示 を参照してください。

第3章 JFROG ARTIFACTORY プラグインのインストールと設定

JFrog Artifactory は、JFrog Artifactory リポジトリーに保存されているコンテナーイメージに関する情報を表示するフロントエンドプラグインです。JFrog Artifactory プラグインは Developer Hub に事前インストールされており、デフォルトで無効になっています。これを使用するには、まず有効化および設定を行う必要があります。



重要

JFrog Artifactory プラグインはテクノロジープレビュー機能です。

テクノロジープレビュー機能は、実稼働環境での Red Hat サービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートの詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

バンドルされたコミュニティーの動的プラグインに対する Red Hat のサポートの詳細は、Red Hat Developer サポートポリシー のページを参照してください。

3.1. インストール

JFrog Artifactory プラグインは、基本的な設定プロパティーを使用して Developer Hub に事前にインストールされています。これを有効にするには、次のように、disabled プロパティーを **false** に設定します。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-jfrog-artifactory
        disabled: false
```

3.2. 設定

1. 以下のように、**app-config.yaml** ファイルで目的の JFrog Artifactory サーバーにプロキシーを設定します。

```
proxy:
  endpoints:
    '/jfrog-artifactory/api':
      target: http://<hostname>:8082 # or https://<customer>.jfrog.io
      headers:
      # Authorization: 'Bearer <YOUR TOKEN>'
      # Change to "false" in case of using a self-hosted Artifactory instance with a self-signed certificate
      secure: true
```

2. 以下のアノテーションをエンティティーの **catalog-info.yaml** ファイルに追加して、RHDH コンポーネントで JFrog Artifactory プラグイン機能を有効にします。

```
metadata:
  annotations:
    'jfrog-artifactory/image-name': '<IMAGE-NAME>'
```

第4章 KEYCLOAK のインストールと設定

Keycloak を Developer Hub に統合する Keycloak バックエンドプラグインには、次の機能があります。

- レルム内の Keycloak ユーザーの同期
- レルム内の Keycloak グループとそのユーザーの同期

注記

サポートされている Red Hat build of Keycloak (RHBK) バージョンは **24.0** です。

4.1. インストール

Keycloak プラグインは、基本的な設定プロパティーとともに Developer Hub にプリロードされています。これを有効にするには、次のように、**disabled** プロパティーを **false** に設定します。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-catalog-backend-module-
keycloak-dynamic
        disabled: false
```

4.2. BASIC CONFIGURATION

Keycloak プラグインを有効にするには、次の環境変数を設定する必要があります。

- **KEYCLOAK_BASE_URL**
- **KEYCLOAK_LOGIN_REALM**
- **KEYCLOAK_REALM**
- **KEYCLOAK_CLIENT_ID**
- **KEYCLOAK_CLIENT_SECRET**

4.3. 詳細設定

スケジュール設定

次のように、**app-config.yaml** ファイルでスケジュールを設定できます。

```
  catalog:
    providers:
      keycloakOrg:
        default:
          # ...
          # highlight-add-start
          schedule: # optional; same options as in TaskScheduleDefinition
```

```
    # supports cron, ISO duration, "human duration" as used in code
    frequency: { minutes: 1 }
    # supports ISO duration, "human duration" as used in code
    timeout: { minutes: 1 }
    initialDelay: { seconds: 15 }
    # highlight-add-end
```

注記

app-config.yaml ファイルのスケジュールに変更を加えた場合は、再起動して変更を適用します。

Keycloak クエリーパラメーター

次のように、**app-config.yaml** ファイル内のデフォルトの Keycloak クエリーパラメーターをオーバーライドできます。

```
catalog:
  providers:
    keycloakOrg:
      default:
        # ...
        # highlight-add-start
        userQuerySize: 500 # Optional
        groupQuerySize: 250 # Optional
        # highlight-add-end
```

Developer Hub と Keycloak 間の通信は、Keycloak API を使用して有効になります。ユーザー名およびパスワード、またはクライアント認証情報は、認証方法に対応しています。

以下の表は、**app-config.yaml** ファイルの **catalog.providers.keycloakOrg.<ENVIRONMENT_NAME>** オブジェクトでプラグインを有効にするために設定できるパラメーターを説明しています。

名前	説明	デフォルト値	必須
baseUrl	Keycloak サーバーの場所 (例: **https://localhost:8443/auth**)。	""	はい
realm	同期するレルム	**master**	なし
loginRealm	認証に使用するレルム	**master**	なし
username	認証するユーザー名	""	パスワードベースの認証を使用している場合は Yes
password	認証するパスワード	""	パスワードベースの認証を使用している場合は Yes

名前	説明	デフォルト値	必須
clientId	認証するクライアント ID	""	クライアントクレデンシャルベースの認証を使用している場合は Yes
clientSecret	認証するクライアントシークレット	""	クライアントクレデンシャルベースの認証を使用している場合は Yes
userQuerySize	一度にクエリーするユーザーの数	**100**	なし
groupQuerySize	一度にクエリーするグループの数	**100**	なし

クライアントクレデンシャルを使用する場合は、アクセスタイプを **confidential** に設定し、サービスアカウントを有効にする必要があります。**realm-management** クライアントロールから以下のロールも追加する必要があります。

- **query-groups**
- **query-users**
- **view-users**

4.4. 制限

自己署名または企業証明書に問題がある場合は、Developer Hub を開始する前に以下の環境変数を設定できます。

NODE_TLS_REJECT_UNAUTHORIZED=0

注記

環境変数を設定するソリューションは推奨されません。

第5章 NEXUS REPOSITORY MANAGER プラグインのインストールおよび設定

Nexus Repository Manager プラグインは、Developer Hub アプリケーションのビルドアーティファクトに関する情報を表示します。ビルドアーティファクトは Nexus Repository Manager で入手できます。

重要

Nexus Repository Manager プラグインは、テクノロジープレビュー機能のみです。

テクノロジープレビュー機能は、実稼働環境での Red Hat サービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートの詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

バンドルされたコミュニティーの動的プラグインに対する Red Hat のサポートの詳細は、Red Hat Developer サポートポリシー のページを参照してください。

5.1. インストール

Nexus Repository Manager プラグインは、基本的な設定プロパティーとともに Developer Hub にプリロードされています。これを有効にするには、次のように、disabled プロパティーを **false** に設定します。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-nexus-repository-manager
        disabled: false
```

5.2. 設定

1. 次のように、**app-config.yaml** ファイルでプロキシーを目的の Nexus Repository Manager サーバーに設定します。

```
proxy:
    '/nexus-repository-manager':
    target: 'https://<NEXUS_REPOSITORY_MANAGER_URL>'
    headers:
        X-Requested-With: 'XMLHttpRequest'
        # Uncomment the following line to access a private Nexus Repository Manager using a
token
        # Authorization: 'Bearer <YOUR TOKEN>'
    changeOrigin: true
```

```
    # Change to "false" in case of using self hosted Nexus Repository Manager instance with a
self-signed certificate
    secure: true
```

2. オプション: Nexus Repository Manager プロキシーのベース URL を次のように変更します。

```
nexusRepositoryManager:
    # default path is `/nexus-repository-manager`
    proxyPath: /custom-path
```

3. オプション: 次の実験的アノテーションを有効にします。

```
nexusRepositoryManager:
    experimentalAnnotations: true
```

4. 以下のアノテーションを使用してエンティティーにアノテーションを付けます。

```
metadata:
    annotations:
    # insert the chosen annotations here
    # example
    nexus-repository-manager/docker.image-name: `<ORGANIZATION>/<REPOSITORY>`,
```

第6章 TEKTON プラグインのインストールと設定

Tekton プラグインを使用すると、Kubernetes または OpenShift クラスターでの CI/CD パイプライン実行の結果を視覚化できます。このプラグインを使用すると、ユーザーはアプリケーションのパイプラインに含まれるすべての関連タスクの概略ステータスを視覚的に確認できます。

6.1. インストール

前提条件

- **@backstage/plugin-kubernetes** および **@backstage/plugin-kubernetes-backend** 動的プラグインがインストールおよび設定されている。
- Kubernetes プラグインが、**ServiceAccount** を使用してクラスターに接続するように設定されている。
- **ServiceAccount** がクラスターにアクセスするように、**ClusterRole** がカスタムリソース (PipelineRuns および TaskRuns) に付与されている。

注記

RHDH Kubernetes プラグインが設定されている場合、**ClusterRole** はすでに付与されています。

- Pod ログを表示するために、**pods/log** の権限を付与している。
- 以下のコードを使用して、カスタムリソースおよび Pod ログに **ClusterRole** を付与することができます。

```
kubernetes:
  ...
  customResources:
    - group: 'tekton.dev'
      apiVersion: 'v1'
      plural: 'pipelineruns'
    - group: 'tekton.dev'
      apiVersion: 'v1'


...
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: backstage-read-only
  rules:
    - apiGroups:
        - ""
      resources:
        - pods/log
      verbs:
        - get
        - list
        - watch
    ...
```

```
  - apiGroups:
      - tekton.dev
    resources:
      - pipelineruns
      - taskruns
    verbs:
      - get
      - list
```

読み取り専用の **ClusterRole** 用に準備されたマニフェストを使用できます。これにより、Kubernetes プラグインと Tekton プラグインの両方にアクセスできるようになります。

- 以下のアノテーションをエンティティーの **catalog-info.yaml** ファイルに追加し、エンティティーに Kubernetes リソースが含まれているかどうかを特定します。

```
annotations:
  ...

  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

- また、**backstage.io/kubernetes-namespace** アノテーションを追加して、定義された namespace を使用して Kubernetes リソースを識別することもできます。

```
annotations:
  ...

  backstage.io/kubernetes-namespace: <RESOURCE_NS>
```

- 以下のアノテーションをエンティティーの **catalog-info.yaml** ファイルに追加して、RHDH で Tekton 関連の機能を有効にします。アノテーションの値は、RHDH エンティティーの名前を識別します。

```
annotations:
  ...

  janus-idp.io/tekton : <BACKSTAGE_ENTITY_NAME>
```

- RHDH が Kubernetes リソースを検索するために使用する、カスタムラベルセレクターを追加します。ラベルセレクターは ID アノテーションよりも優先されます。

```
annotations:
  ...

  backstage.io/kubernetes-label-selector: 'app=my-app,component=front-end'
```

- Kubernetes プラグインが要求されたエンティティーから Kubernetes リソースを取得できるように、以下のラベルをリソースに追加します。

```
labels:
  ...

  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

注記

ラベルセレクターを使用する場合は、上記のラベルがリソースに存在する必要があります。

手順

- Tekton プラグインは、基本的な設定プロパティーとともに RHDH にプリロードされています。これを有効にするには、次のように、disabled プロパティーを false に設定します。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-tekton
        disabled: false
```

第7章 トポロジープラグインのインストール

7.1. インストール

トポロジープラグインを使用すると、Kubernetes クラスター上のあらゆるサービスを動かすデプロイメント、ジョブ、デーモンセット、ステートフルセット、CronJob、Pod、仮想マシンなどのワークロードを視覚化できます。

前提条件

- @backstage/plugin-kubernetes-backend 動的プラグインがインストールおよび設定されている。
- Kubernetes プラグインが、ServiceAccount を使用してクラスターに接続するように設定されている。
- **ClusterRole** が、クラスターにアクセスする ServiceAccount に付与されている。

注記

Developer Hub Kubernetes プラグインが設定されている場合は、**ClusterRole** はすでに付与されています。

手順

- トポロジープラグインは、基本的な設定プロパティーとともに Developer Hub にプリロードされています。これを有効にするには、次のように、disabled プロパティーを false に設定します。

  app-config.yaml フラグメント

```
auth:
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-topology
        disabled: false
```

7.2. トポロジープラグインの設定

7.2.1. OpenShift ルートの表示

手順

1. OpenShift ルートを表示するには、Cluster Role のルートリソースへの読み取りアクセス権を付与します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
    name: backstage-read-only
  rules:
    ...
    - apiGroups:
        - route.openshift.io
      resources:
        - routes
      verbs:
        - get
        - list
```

2. また、**app-config.yaml** ファイルの **kubernetes.customResources** プロパティーに以下を追加します。

```
kubernetes:
  ...
  customResources:
    - group: 'route.openshift.io'
      apiVersion: 'v1'
        plural: 'routes'
```

7.2.2. Pod ログの表示

手順

- Pod ログを表示するには、**ClusterRole** に次の権限を付与する必要があります。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - ''
    resources:
      - pods
      - pods/log
    verbs:
      - get
      - list
      - watch
```

7.2.3. Tekton PipelineRuns の表示

手順

1. Tekton PipelineRuns を表示するには、**ClusterRole** の **pipelines**、**pipelinesruns**、および **taskruns** リソースへの読み取りアクセス権を付与します。

```
...
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
```

```
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - tekton.dev
    resources:
      - pipelines
      - pipelineruns
      - taskruns
    verbs:
      - get
      - list
```

2. サイドパネルで Tekton PipelineRuns リストを表示し、Topology ノードデコレーターで最新の PipelineRuns ステータスを表示するには、**app-config.yaml** ファイルの **kubernetes.customResources** プロパティーに次のコードを追加します。

```
kubernetes:
   ...
   customResources:
     - group: 'tekton.dev'
       apiVersion: 'v1'
       plural: 'pipelines'
     - group: 'tekton.dev'
       apiVersion: 'v1'
       plural: 'pipelineruns'
     - group: 'tekton.dev'
       apiVersion: 'v1'
       plural: 'taskruns'
```

7.2.4. 仮想マシンの表示

前提条件

1. OpenShift Virtualization Operator が Kubernetes クラスターにインストールされ、設定されます。

2. **ClusterRole** の **VirtualMachines** リソースへの読み取りアクセス権を付与します。

```
...
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - kubevirt.io
    resources:
      - virtualmachines
      - virtualmachineinstances
    verbs:
      - get
      - list
```

3. トポロジープラグイン上の仮想マシンノードを表示するには、**app-config.yaml** ファイルの **kubernetes.customResources** プロパティーに次のコードを追加します。

```
kubernetes:
  ...
  customResources:
    - group: 'kubevirt.io'
      apiVersion: 'v1'
      plural: 'virtualmachines'
    - group: 'kubevirt.io'
      apiVersion: 'v1'
      plural: 'virtualmachineinstances'
```

7.2.5. ソースコードエディターの有効化

ソースコードエディターを有効にするには、次のサンプルコードに示すように、**ClusterRole** の CheClusters リソースへの読み取りアクセス権を付与する必要があります。

```
...
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: backstage-read-only
rules:
  ...
  - apiGroups:
      - org.eclipse.che
    resources:
      - checlusters
    verbs:
      - get
      - list
```

ソースコードエディターを使用するには、**app-config.yaml** ファイルの **kubernetes.customResources** プロパティーに次の設定を追加する必要があります。

```
kubernetes:
  ...
  customResources:
    - group: 'org.eclipse.che'
      apiVersion: 'v2'
      plural: 'checlusters'
```

7.3. TOPOLOGY プラグインのラベルとアノテーションの管理

7.3.1. ソースコードエディターまたはソースへのリンク

ソースコードエディターを使用して、関連付けられているアプリケーションの Git リポジトリーに移動するには、デプロイメントなどのワークロードリソースに次のアノテーションを追加します。

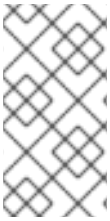
```
annotations:
  app.openshift.io/vcs-uri: <GIT_REPO_URL>
```

特定のブランチに移動するには、次のアノテーションを追加します。

```
annotations:
  app.openshift.io/vcs-ref: <GIT_REPO_BRANCH>
```

注記

Red Hat OpenShift Dev Spaces がインストールおよび設定されており、Git URL アノテーションもワークロード YAML ファイルに追加されている場合は、編集コードデコレーターをクリックすると、Red Hat OpenShift Dev Spaces インスタンスにリダイレクトされます。



注記

OCP Git インポートフローを使用してアプリケーションをデプロイする場合は、インポートフローによってラベルが追加されるため、独自に追加する必要はありません。それ以外の場合は、ワークロード YAML ファイルにラベルを手動で追加する必要があります。

デコレーターを使用してアクセスする編集 URL を含む **app.openshift.io/edit-url** アノテーションを追加することもできます。

7.3.2. エンティティーアノテーション/ラベル

RHDH がエンティティーに Kubernetes コンポーネントがあることを検出するには、エンティティーの **catalog-info.yaml** ファイルに次のアノテーションを追加します。

```
annotations:
  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>
```

Kubernetes プラグインが要求されたエンティティーから Kubernetes リソースを取得できるように、以下のラベルをリソースに追加します。

```
labels:
  backstage.io/kubernetes-id: <BACKSTAGE_ENTITY_NAME>`
```

注記

ラベルセレクターを使用する場合は、上記のラベルがリソースに存在する必要があります。

7.3.3. namespace アノテーション

手順

- 定義された namespace を使用して Kubernetes リソースを識別するには、**backstage.io/kubernetes-namespace** アノテーションを追加します。

  ```
  annotations:
    backstage.io/kubernetes-namespace: <RESOURCE_NS>
  ```

backstage.io/kubernetes-namespace アノテーションが **catalog-info.yaml** ファイルに追加されている場合、ソースコードエディターを使用して Red Hat OpenShift Dev Spaces インスタンスにアクセスすることはできません。

インスタンス URL を取得するには、CheCluster カスタムリソース (CR) が必要です。CheCluster CR は openshift-devspaces namespace に作成されるため、namespace のアノテーション値が openshift-devspaces でない場合、インスタンス URL は取得されません。

7.3.4. ラベルセレクタークエリーアノテーション

RHDH が Kubernetes リソースを見つけるために使用する独自のカスタムラベルを作成できます。ラベルセレクターは ID アノテーションよりも優先されます。

```
annotations:
  backstage.io/kubernetes-label-selector: 'app=my-app,component=front-end'
```

Red Hat Dev Spaces が設定されているときに複数のエンティティーがあり、複数のエンティティーで Red Hat Dev Spaces インスタンスにリダイレクトする編集コードデコレーターをサポートする必要がある場合は、各エンティティーの catalog-info.yaml ファイルに backstage.io/kubernetes-label-selector アノテーションを追加できます。

```
annotations:
  backstage.io/kubernetes-label-selector: 'component in (<BACKSTAGE_ENTITY_NAME>,che)'
```

以前のラベルセレクターを使用している場合は、Kubernetes プラグインが要求されたエンティティーから Kubernetes リソースを取得できるように、次のラベルをリソースに追加する必要があります。

```
labels:
  component: che # add this label to your che cluster instance
labels:
  component: <BACKSTAGE_ENTITY_NAME> # add this label to the other resources associated with your entity
```

エンティティーを区別するために、一意のラベルを使用してラベルセレクター用の独自のカスタムクエリーを作成することもできます。ただし、CheCluster インスタンスを含むエンティティーに関連付けられたリソースにこれらのラベルを確実に追加する必要があります。

7.3.5. ノードに表示されるアイコン

トポロジーノードにランタイムアイコンを表示するには、デプロイメントなどのワークロードリソースに次のラベルを追加します。

```
labels:
  app.openshift.io/runtime: <RUNTIME_NAME>
```

あるいは、次のラベルを含めてランタイムアイコンを表示することもできます。

```
labels:
  app.kubernetes.io/name: <RUNTIME_NAME>
```

<RUNTIME_NAME> でサポートされる値は次のとおりです。

- django

- dotnet
- drupal
- go-gopher
- golang
- grails
- jboss
- jruby
- js
- nginx
- nodejs
- openjdk
- perl
- phalcon
- php
- python
- quarkus
- rails
- redis
- rh-spring-boot
- rust
- java
- rh-openjdk
- ruby
- spring
- spring-boot

注記

その他の値の場合、ノードのアイコンはレンダリングされません。

7.3.6. アプリケーションのグループ化

ビジュアルグループ内のデプロイメントや Pod などのワークロードリソースを表示するには、次のラベルを追加します。

```
labels:
  app.kubernetes.io/part-of: <GROUP_NAME>
```

7.3.7. ノードコネクター

手順

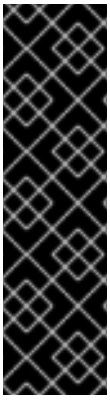
ビジュアルコネクターを使用してデプロイメントや Pod などのワークロードリソースを表示するには、次のアノテーションを追加します。

+

```
annotations:
  app.openshift.io/connects-to: '[{"apiVersion": <RESOURCE_APIVERSION>,"kind":
<RESOURCE_KIND>,"name": <RESOURCE_NAME>}]'
```

ラベルとアノテーションの詳細は、**Guidelines for labels and annotations for OpenShift applications** を参照してください。

第8章 GITHUB リポジトリーの一括インポート



重要

この章の機能はテクノロジープレビュー機能です。テクノロジープレビュー機能は、実稼働環境での Red Hat サービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートの詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Red Hat Developer Hub は、GitHub リポジトリーのオンボーディングを自動化し、インポートステータスを追跡できます。

8.1. 一括インポート機能の有効化とアクセス権の付与

ユーザーに対して一括インポート機能を有効にし、アクセスするために必要な権限を付与できます。

前提条件

- GitHub インテグレーションが設定されている。

手順

1. Bulk Import プラグインはインストールされていますが、デフォルトでは無効になっています。**./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import-backend-dynamic** および **./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import** プラグインを有効にするには、**dynamic-plugins.yaml** を次の内容で編集します。

   dynamic-plugins.yaml フラグメント

   ```
   plugins:
     - package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import-backend-dynamic
       disabled: false
     - package: ./dynamic-plugins/dist/red-hat-developer-hub-backstage-plugin-bulk-import
       disabled: false
   ```

   Red Hat Developer Hub でのプラグインのインストールと表示 を参照してください。

2. 管理者ではないユーザーに対して必要な **bulk.import** RBAC 権限を次のように設定します。

   rbac-policy.csv フラグメント

   ```
   p, role:default/bulk-import, bulk.import, use, allow
   g, user:default/<your_user>, role:default/bulk-import
   ```

   一括インポート機能を使用できるのは、Developer Hub 管理者または **bulk.import** 権限を持つユーザーのみであることに注意してください。Red Hat Developer Hub の権限ポリシー を参照してください。

検証

- サイドバーに **Bulk Import** オプションが表示されます。
- **Bulk Import** ページには、**Added Repositories** のリストが表示されます。

8.2. 複数の GITHUB リポジトリーのインポート

Red Hat Developer Hub では、GitHub リポジトリーを選択し、Developer Hub カタログへのオンボーディングを自動化できます。

前提条件

- 一括インポート機能を有効にし、アクセス権が付与されている。

手順

1. 左側のサイドバーで **Bulk Import** をクリックします。
2. 右上隅の **Add** ボタンをクリックすると、設定された GitHub インテグレーションからアクセスできるすべてのリポジトリーのリストが表示されます。
   a. **Repositories** ビューから、任意のリポジトリーを選択したり、アクセス可能なリポジトリーを検索したりできます。選択されたリポジトリーごとに、**catalog-info.yaml** が生成されます。
   b. **Organizations** ビューでは、3 番目の列の **Select** をクリックして任意の組織を選択できます。このオプションを使用すると、選択した組織から 1 つ以上のリポジトリーを選択できます。
3. 各リポジトリーのプルリクエストの詳細を表示または編集するには、**Preview file** をクリックします。
   a. プルリクエストの説明と **catalog-info.yaml** ファイルの内容を確認します。
   b. オプション: リポジトリーに **.github/CODEOWNERS** ファイルがある場合は、**content-info.yaml** に特定のエンティティー所有者を含めるのではなく、**Use CODEOWNERS file as Entity Owner** チェックボックスを選択してそのファイルを使用できます。
   c. **Save** をクリックします。
4. **Create pull requests** をクリックします。この時点で、選択したリポジトリーに対して一連のドライランチェックが実行され、次のようなインポートの要件を満たしているかどうかが確認されます。
   a. リポジトリー **catalog-info.yaml** で指定された名前のエンティティーが Developer Hub カタログに存在しないことを確認します。
   b. リポジトリーが空でないことを確認します。
   c. リポジトリーの **Use CODEOWNERS file as Entity Owner**チェックボックスが選択されている場合、リポジトリーに **.github/CODEOWNERS** ファイルが含まれていることを確認します。
      - エラーが発生した場合、プルリクエストは作成されず、問題の詳細を示す **Failed to create PR** のエラーメッセージが表示されます。理由の詳細を表示するには、**Edit** をクリックします。

- エラーがなければ、プルリクエストが作成され、追加されたリポジトリーのリストにリダイレクトされます。

5. **catalog-info.yml** ファイルを作成する各プルリクエストを確認し、マージします。

検証

- **Added repositories** リストには、インポートしたリポジトリーが、それぞれ適切なステータス **Waiting for approval** または **Added** とともに表示されます。
- リストされている **Waiting for approval** のインポートジョブごとに、対応するリポジトリーに **catalog-info.yaml** ファイルを追加するための、対応するプルリクエストがあります。

8.3. 追加されたリポジトリーの管理

Developer Hub にインポートされたリポジトリーを監視および管理できます。

前提条件

- GitHub リポジトリーがインポートされている。

手順

1. 左側のサイドバーで **Bulk Import** をクリックして、Import ジョブとして追跡されている現在のすべてのリポジトリーとそのステータスを表示します。

   Added

   インポートプルリクエストがマージされた後、または一括インポート中にリポジトリーに **catalog-info.yaml** ファイルがすでに含まれていた場合、リポジトリーは Developer Hub カタログに追加されます。カタログでエンティティーが利用可能になるまで数分かかる場合があります。

   Waiting for approval

   catalog-info.yaml ファイルをリポジトリーに追加するオープンプルリクエストがあります。これにより、以下が可能になります。

   - 右側の **鉛筆アイコン** をクリックして、プルリクエストの詳細を表示するか、Developer Hub からプルリクエストのコンテンツを編集します。
   - インポートジョブを削除します。この操作により、インポート PR も閉じられます。
   - インポートジョブを **Added** 状態に移行するには、Git リポジトリーからのインポートプルリクエストをマージします。

   Empty

   リポジトリーは他のソースからインポートされていますが、**catalog-info.yaml** ファイルがなく、それを追加するインポートプルリクエストもないため、Developer Hub はインポートジョブのステータスを判別できません。

注記

- インポートプルリクエストがマージされると、インポートステータスは Added Repositories のリストで **Added** としてマークされますが、対応するエンティティーが Developer Hub カタログに表示されるまでに数秒かかる場合があります。
- 他のソースから追加された場所 (**app-config.yaml** ファイルで静的に追加された場所、GitHub 検出を有効にする ときに動的に追加された場所、または「既存のコンポーネントの登録」ページを使用して手動で登録された場所など) は、次の条件が満たされていると、追加されたリポジトリーの一括インポートリストに表示されることがあります。
  - ターゲットリポジトリーには、設定された GitHub インテグレーションからアクセスできます。
  - 場所 URL は、リポジトリーのデフォルトブランチのルートにある **catalog-info.yaml** ファイルを指します。

8.4. 一括インポート監査ログについて

Bulk Import backend プラグインは、Developer Hub 監査ログに次のイベントを追加します。監査ログの設定方法と表示方法の詳細は、Red Hat Developer Hub の監査ログ を参照してください。

一括インポートイベント:

BulkImportUnknownEndpoint

不明なエンドポイントへの要求を追跡します。

BulkImportPing

/ping エンドポイントへの **GET** リクエストを追跡し、一括インポートバックエンドが稼働していることを確認できます。

BulkImportFindAllOrganizations

/organizations エンドポイントへの **GET** リクエストを追跡します。これにより、設定されたすべての GitHub インテグレーションからアクセス可能な組織のリストが返されます。

BulkImportFindRepositoriesByOrganization

/organizations/:orgName/repositories エンドポイントへの **GET** リクエストを追跡します。このエンドポイントは、指定された組織のリポジトリーのリストを返します (設定された GitHub インテグレーションのいずれかからアクセス可能)。

BulkImportFindAllRepositories

/repositories エンドポイントへの GET リクエストを追跡し、設定されたすべての GitHub インテグレーションからアクセス可能なリポジトリーのリストを返します。

BulkImportFindAllImports

/imports エンドポイントへの **GET** リクエストを追跡し、既存のインポートジョブのリストとそのステータスを返します。

BulkImportCreateImportJobs

/imports エンドポイントへの **POST** リクエストを追跡します。これにより、ターゲットリポジトリーにインポートプルリクエストを作成して、1 つまたは複数のリポジトリーを Developer Hub カタログに一括インポートするリクエストを送信できます。

BulkImportFindImportStatusByRepo

指定されたリポジトリーのインポートジョブの詳細を取得する **/import/by-repo** エンドポイントへの **GET** リクエストを追跡します。

BulkImportDeleteImportByRepo

/import/by-repo エンドポイントへの **DELETE** リクエストを追跡します。これにより、作成された可能性があり、未対応のインポートプルリクエストをすべて終了し、指定されたリポジトリーの既存のインポートジョブがすべて削除されます。

監査ログの一括インポートの例


```
{
  "actor": {
    "actorId": "user:default/myuser",
    "hostname": "localhost",
    "ip": "::1",
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/128.0.0.0 Safari/537.36"
  },
  "eventName": "BulkImportFindAllOrganizations",
  "isAuditLog": true,
  "level": "info",
  "message": "'get /organizations' endpoint hit by user:default/myuser",
  "meta": {},
  "plugin": "bulk-import",
  "request": {
    "body": {},
    "method": "GET",
    "params": {},
    "query": {
      "pagePerIntegration": "1",
      "sizePerIntegration": "5"
    },
    "url": "/api/bulk-import/organizations?pagePerIntegration=1&sizePerIntegration=5"
  },
  "response": {
    "status": 200
  },
  "service": "backstage",
  "stage": "completion",
  "status": "succeeded",
  "timestamp": "2024-08-26 16:41:02"
}
```

第9章 RED HAT DEVELOPER HUB の SERVICENOW カスタムアクション

重要

この章の機能はテクノロジープレビュー機能です。テクノロジープレビュー機能は、実稼働環境での Red Hat サービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートの詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Red Hat Developer Hub では、カタログ内のリソースを取得して登録する ServiceNow カスタムアクション (カスタムアクション) にアクセスできます。

Developer Hub のカスタムアクションを使用すると、レコードの管理を容易化、自動化できます。カスタムアクションを使用すると、次の操作を実行できます。

- レコードの作成、更新、または削除
- 1 つまたは複数のレコードに関する情報の取得

9.1. RED HAT DEVELOPER HUB の SERVICENOW カスタムアクションプラグインの有効化

Red Hat Developer Hub では、ServiceNow カスタムアクションはプリロードされたプラグインとして提供されますが、デフォルトでは無効になっています。次の手順でカスタムアクションプラグインを有効にできます。

前提条件

- Red Hat Developer Hub がインストールされ、実行されている。Developer Hub のインストールの詳細は、Helm チャートを使用した OpenShift Container Platform への Red Hat Developer Hub のインストール を参照してください。
- Developer Hub でプロジェクトを作成した。

手順

1. カスタムアクションプラグインを有効にするには、プラグイン名を持つ **package** を追加し、Helm チャートの **disabled** フィールドを次のように更新します。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-scaffolder-backend-module-servicenow-dynamic
        disabled: false
```

注記

プラグインのデフォルト設定は、**dynamic-plugins.default.yaml** ファイルから抽出されます。ただし、**pluginConfig** エントリーを使用すると、デフォルト設定をオーバーライドできます。

2. カスタムアクションにアクセスするには、Helm チャートで次の変数を設定します。

```
servicenow:
  # The base url of the ServiceNow instance.
  baseUrl: ${SERVICENOW_BASE_URL}
  # The username to use for authentication.
  username: ${SERVICENOW_USERNAME}
  # The password to use for authentication.
  password: ${SERVICENOW_PASSWORD}
```

9.2. RED HAT DEVELOPER HUB でサポートされている SERVICENOW カスタムアクション

ServiceNow カスタムアクションを使用すると、Red Hat Developer Hub でレコードを管理できます。カスタムアクションは、API 要求で次の HTTP メソッドをサポートします。

- **GET**: 指定したリソースのエンドポイントから指定した情報を取得する
- **POST**: リソースを作成または更新する
- **PUT** : リソースを変更する
- **PATCH** : リソースを更新する
- **DELETE**: リソースを削除する

9.2.1. ServiceNow カスタムアクション

[GET] servicenow:now:table:retrieveRecord

Developer Hub のテーブルから指定のレコードの情報を取得します。

表9.1 入力パラメーター

名前	型	要件	説明
tableName	**string**	必須	レコードの取得元のテーブル名
sysId	**string**	必須	取得するレコードの一意の識別子
sysparmDisplayValue	**enum("true", "false", "all")**	任意	**true** に設定してフィールド表示値を返すか、**false** に設定して実際の値を返すか、またはその両方を返します。デフォルト値は **false** です。

名前	型	要件	説明
sysparmExcludeReferenceLink	**boolean**	任意	参照フィールドのテーブル API リンクを除外するには、**true** に設定します。デフォルト値は **false** です。
sysparmFields	**string[]**	任意	レスポンスで返されるフィールドの配列
sysparmView	**string**	任意	指定された UI ビューに従ってレスポンスを表示します。**sysparm_fields** を使用してこのパラメーターをオーバーライドできます。
sysparmQueryNoDomain	**boolean**	任意	ドメイン間でデータにアクセスするには (許可されている場合)、**true** に設定します。デフォルト値は **false** です。

表9.2 出力パラメーター

名前	型	説明
result	**Record<PropertyKey, unknown>**	要求のレスポンスボディー

[GET] servicenow:now:table:retrieveRecords

Developer Hub のテーブルから複数のレコードに関する情報を取得します。

表9.3 入力パラメーター

名前	型	要件	説明
tableName	**string**	必須	レコードの取得元のテーブル名
sysparamQuery	**string**	任意	結果をフィルタリングするために使用するエンコードされたクエリー文字列
sysparmDisplayValue	**enum("true", "false", "all")**	任意	**true** に設定してフィールド表示値を返すか、**false** に設定して実際の値を返すか、またはその両方を返します。デフォルト値は **false** です。
sysparmExcludeReferenceLink	**boolean**	任意	参照フィールドのテーブル API リンクを除外するには、**true** に設定します。デフォルト値は **false** です。

名前	型	要件	説明
sysparmSuppressPaginationHeader	**boolean**	任意	ページネーションヘッダーを抑制するには、**true** に設定します。デフォルト値は **false** です。
sysparmFields	**string[]**	任意	レスポンスで返されるフィールドの配列
sysparmLimit	**int**	任意	ページごとに返される結果の最大数。デフォルト値は **10,000** です。
sysparmView	**string**	任意	指定された UI ビューに従ってレスポンスを表示します。**sysparm_fields** を使用してこのパラメーターをオーバーライドできます。
sysparmQueryCategory	**string**	任意	クエリーに使用するクエリーカテゴリーの名前
sysparmQueryNoDomain	**boolean**	任意	ドメイン間でデータにアクセスするには (許可されている場合)、**true** に設定します。デフォルト値は **false** です。
sysparmNoCount	**boolean**	任意	テーブルに対して select count (*) を実行しません。デフォルト値は **false** です。

表9.4 出力パラメーター

名前	型	説明
result	**Record<PropertyKey, unknown>**	要求のレスポンスボディー

[POST] servicenow:now:table:createRecord

Developer Hub のテーブルにレコードを作成します。

表9.5 入力パラメーター

名前	型	要件	説明
tableName	**string**	必須	レコードの保存先のテーブル名
requestBody	**Record<PropertyKey, unknown>**	任意	指定のレコードで定義する各パラメーターのフィールド名と関連する値

名前	型	要件	説明
sysparmDisplayValue	**enum("true", "false", "all")**	任意	**true** に設定してフィールド表示値を返すか、**false** に設定して実際の値を返すか、またはその両方を返します。デフォルト値は **false** です。
sysparmExcludeReferenceLink	**boolean**	任意	参照フィールドのテーブル API リンクを除外するには、**true** に設定します。デフォルト値は **false** です。
sysparmFields	**string[]**	任意	レスポンスで返されるフィールドの配列
sysparmInputDisplayValue	**boolean**	任意	**true** に設定して表示値を使用するか、**false** に設定して実際の値を使用してフィールド値を設定します。デフォルト値は **false** です。
sysparmSuppressAutoSysField	**boolean**	任意	システムフィールドの自動生成を抑制するには、**true** に設定します。デフォルト値は **false** です。
sysparmView	**string**	任意	指定された UI ビューに従ってレスポンスを表示します。**sysparm_fields** を使用してこのパラメーターをオーバーライドできます。

表9.6 出力パラメーター

名前	型	説明
result	**Record<PropertyKey, unknown>**	要求のレスポンスボディー

[PUT] servicenow:now:table:modifyRecord

Developer Hub のテーブルのレコードを変更します。

表9.7 入力パラメーター

名前	型	要件	説明
tableName	**string**	必須	レコードを変更するテーブルの名前
sysId	**string**	必須	変更するレコードの一意の識別子

名前	型	要件	説明
requestBody	**Record<PropertyKey, unknown>**	任意	指定のレコードで定義する各パラメーターのフィールド名と関連する値
sysparmDisplayValue	**enum("true", "false", "all")**	任意	**true** に設定してフィールド表示値を返すか、**false** に設定して実際の値を返すか、またはその両方を返します。デフォルト値は **false** です。
sysparmExcludeReferenceLink	**boolean**	任意	参照フィールドのテーブル API リンクを除外するには、**true** に設定します。デフォルト値は **false** です。
sysparmFields	**string[]**	任意	レスポンスで返されるフィールドの配列
sysparmInputDisplayValue	**boolean**	任意	**true** に設定して表示値を使用するか、**false** に設定して実際の値を使用してフィールド値を設定します。デフォルト値は **false** です。
sysparmSuppressAutoSysField	**boolean**	任意	システムフィールドの自動生成を抑制するには、**true** に設定します。デフォルト値は **false** です。
sysparmView	**string**	任意	指定された UI ビューに従ってレスポンスを表示します。**sysparm_fields** を使用してこのパラメーターをオーバーライドできます。
sysparmQueryNoDomain	**boolean**	任意	ドメイン間でデータにアクセスするには (許可されている場合)、**true** に設定します。デフォルト値は **false** です。

表9.8 出力パラメーター

名前	型	説明
result	**Record<PropertyKey, unknown>**	要求のレスポンスボディー

[PATCH] servicenow:now:table:updateRecord

Developer Hub のテーブルのレコードを更新します。

表9.9 入力パラメーター

名前	型	要件	説明

名前	型	要件	説明
tableName	**string**	必須	レコードを更新するテーブルの名前
sysId	**string**	必須	更新するレコードの一意の識別子
requestBody	**Record<PropertyKey, unknown>**	任意	指定のレコードで定義する各パラメーターのフィールド名と関連する値
sysparmDisplayValue	**enum("true", "false", "all")**	任意	**true** に設定してフィールド表示値を返すか、**false** に設定して実際の値を返すか、またはその両方を返します。デフォルト値は **false** です。
sysparmExcludeReferenceLink	**boolean**	任意	参照フィールドのテーブル API リンクを除外するには、**true** に設定します。デフォルト値は **false** です。
sysparmFields	**string[]**	任意	レスポンスで返されるフィールドの配列
sysparmInputDisplayValue	**boolean**	任意	**true** に設定して表示値を使用するか、**false** に設定して実際の値を使用してフィールド値を設定します。デフォルト値は **false** です。
sysparmSuppressAutoSysField	**boolean**	任意	システムフィールドの自動生成を抑制するには、**true** に設定します。デフォルト値は **false** です。
sysparmView	**string**	任意	指定された UI ビューに従ってレスポンスを表示します。**sysparm_fields** を使用してこのパラメーターをオーバーライドできます。
sysparmQueryNoDomain	**boolean**	任意	ドメイン間でデータにアクセスするには (許可されている場合)、**true** に設定します。デフォルト値は **false** です。

表9.10 出力パラメーター

名前	型	説明
result	**Record<PropertyKey, unknown>**	要求のレスポンスボディー

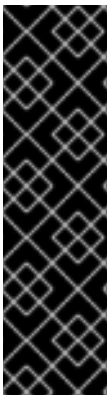
[DELETE] servicenow:now:table:deleteRecord

Developer Hub のテーブルからレコードを削除します。

表9.11 入力パラメーター

名前	型	要件	説明
tableName	**string**	必須	レコードを削除するテーブルの名前
sysId	**string**	必須	削除するレコードの一意の識別子
sysparmQueryNoDomain	**boolean**	任意	ドメイン間でデータにアクセスするには (許可されている場合)、**true** に設定します。デフォルト値は **false** です。

第10章 RED HAT DEVELOPER HUB の KUBERNETES カスタムアクション



重要

この章の機能はテクノロジープレビュー機能です。テクノロジープレビュー機能は、実稼働環境での Red Hat サービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、Red Hat では実稼働環境での使用を推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポートの詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Kubernetes カスタムアクションを使用すると、Kubernetes リソースを作成および管理できます。

Kubernetes カスタムアクションプラグインは、Developer Hub インスタンスにデフォルトでプリインストールされ、無効になっています。Red Hat Developer Hub の Helm チャートを設定することで、Kubernetes カスタムアクションプラグインを無効または有効にしたり、その他のパラメーターを変更したりできます。

注記

Kubernetes scaffolder アクションと Kubernetes カスタムアクションは、このドキュメント全体で同じ概念を指します。

10.1. RED HAT DEVELOPER HUB での KUBERNETES カスタムアクションプラグインの有効化

Red Hat Developer Hub では、Kubernetes カスタムアクションはプリインストールされたプラグインとして提供されており、デフォルトでは無効になっています。Helm チャート内の **disabled** キー値を更新することで、Kubernetes カスタムアクションプラグインを有効にできます。

前提条件

- Helm チャートを使用して Red Hat Developer Hub をインストールした。

手順

Kubernetes カスタムアクションプラグインを有効にするには、次の手順を実行します。

- Helm チャートで、Kubernetes カスタムアクションプラグイン名を含む **package** を追加し、**disabled** フィールドを更新します。以下に例を示します。

```
global:
  dynamic:
    includes:
      - dynamic-plugins.default.yaml
    plugins:
      - package: ./dynamic-plugins/dist/backstage-community-plugin-scaffolder-backend-module-kubernetes-dynamic
        disabled: false
```

注記

プラグインのデフォルト設定は、**dynamic-plugins.default.yaml** ファイルから抽出されます。ただし、**pluginConfig** エントリーを使用すると、デフォルト設定をオーバーライドできます。

10.2. RED HAT DEVELOPER HUB での KUBERNETES カスタムアクションプラグインの使用

Red Hat Developer Hub では、Kubernetes カスタムアクションを使用して Kubernetes のテンプレートアクションを実行できます。

手順

- カスタムテンプレートで Kubernetes カスタムアクションを使用するには、次の Kubernetes アクションをテンプレートに追加します。

```
action: kubernetes:create-namespace
id: create-kubernetes-namespace
name: Create kubernetes namespace
input:
  namespace: my-rhdh-project
  clusterRef: bar
  token: TOKEN
  skipTLSVerify: false
  caData: Zm9v
  labels: app.io/type=ns; app.io/managed-by=org;
```

関連情報

- テンプレートの設定

10.3. RED HAT DEVELOPER HUB での KUBERNETES カスタムアクションを使用したテンプレートの作成

Template オブジェクトを YAML ファイルとして定義することでテンプレートを作成できます。

Template オブジェクトは、テンプレートとそのメタデータを記述します。また、必要な入力変数と、スキャフォールディングサービスによって実行されるアクションのリストも含まれています。

+

```
apiVersion: scaffolder.backstage.io/v1beta3
kind: Template
metadata:
  name: create-kubernetes-namespace
  title: Create a kubernetes namespace
  description: Create a kubernetes namespace

spec:
  type: service
  parameters:
    - title: Information
```

```
      required: [namespace, token]
      properties:
        namespace:
          title: Namespace name
          type: string
          description: Name of the namespace to be created
        clusterRef:
          title: Cluster reference
          type: string
          description: Cluster resource entity reference from the catalog
          ui:field: EntityPicker
          ui:options:
            catalogFilter:
              kind: Resource
        url:
          title: Url
          type: string
          description: Url of the kubernetes API, will be used if clusterRef is not provided
        token:
          title: Token
          type: string
          ui:field: Secret
          description: Bearer token to authenticate with
        skipTLSVerify:
          title: Skip TLS verification
          type: boolean
          description: Skip TLS certificate verification, not recommended to use in production
environment, default to false
        caData:
          title: CA data
          type: string
          ui:field: Secret
          description: Certificate Authority base64 encoded certificate
        labels:
          title: Labels
          type: string
          description: Labels to be applied to the namespace
          ui:widget: textarea
          ui:options:
            rows: 3
          ui:help: 'Hint: Separate multiple labels with a semicolon!'
          ui:placeholder: 'kubernetes.io/type=namespace; app.io/managed-by=org'

    steps:
      - id: create-kubernetes-namespace
        name: Create kubernetes namespace
        action: kubernetes:create-namespace
        input:
          namespace: ${{ parameters.namespace }}
          clusterRef: ${{ parameters.clusterRef }}
          url: ${{ parameters.url }}
          token: ${{ secrets.token }}
          skipTLSVerify: ${{ parameters.skipTLSVerify }}
          caData: ${{ secrets.caData }}
          labels: ${{ parameters.labels }}
```

10.3.1. Red Hat Developer Hub でサポートされている Kubernetes カスタムアクション

Red Hat Developer Hub では、scaffolder テンプレートでカスタム Kubernetes アクションを使用できます。

カスタムの Kubernetes scaffolder アクション

アクション: kubernetes:create-namespace

Developer Hub に Kubernetes クラスターの namespace を作成します。

パラメーター名	型	要件	説明	例
namespace	**string**	必須	Kubernetes namespace の名前	**my-rhdh-project**
clusterRef	**string**	**url** が定義されていない場合にのみ必須です。**url** と **clusterRef** を両方とも指定することはできません。	カタログからのクラスターリソースエンティティーの参照	**bar**
url	**string**	**clusterRef** が定義されていない場合にのみ必要です。**url** と **clusterRef** を両方とも指定することはできません。	Kubernetes クラスターの API URL	**https://api.foo.redhat.com:6443**
token	**String**	必須	認証に使用する Kubernetes API ベアラートークン	
skipTLSVerify	**boolean**	任意	true の場合、証明書の検証がスキップされます。	false
caData	**string**	任意	Base64 でエンコードされた証明書データ	
label	**string**	任意	namespace に適用されるラベル	app.io/type=ns; app.io/managed-by=org;

第11章 コアバックエンドサービス設定のオーバーライド

Red Hat Developer Hub (RHDH) バックエンドプラットフォームは、適切にカプセル化された多数のコアサービスで構成されます。RHDH バックエンドは、初期化中にこれらのデフォルトのコアサービスを静的にインストールします。

バックエンドソースコードをカスタマイズし、Developer Hub アプリケーションを再構築することで、これらのコアサービスを設定できます。または、動的プラグイン機能を使用してコアサービスを **BackendFeature** としてインストールし、カスタマイズできます。

動的プラグイン機能を使用して RHDH アプリケーションのコアサービスをカスタマイズするには、特定のデフォルトのコアサービスが静的にインストールされないようにバックエンドを設定する必要があります。

たとえば、すべての受信要求を処理するミドルウェア関数を追加するには、基礎となる Express アプリケーションへのアクセスを許可する root **HTTP** ルーターバックエンドサービスのカスタム **configure** 関数をインストールします。

受信 HTTP 要求を処理する BackendFeature ミドルウェア関数の例

```
// Create the BackendFeature
export const customRootHttpServerFactory: BackendFeature =
  rootHttpRouterServiceFactory({
    configure: ({ app, routes, middleware, logger }) => {
      logger.info(
        'Using custom root HttpRouterServiceFactory configure function',
      );
      app.use(middleware.helmet());
      app.use(middleware.cors());
      app.use(middleware.compression());
      app.use(middleware.logging());
      // Add a the custom middleware function before all
      // of the route handlers
      app.use(addTestHeaderMiddleware({ logger }));
      app.use(routes);
      app.use(middleware.notFound());
      app.use(middleware.error());
    },
  });

// Export the BackendFeature as the default entrypoint
export default customRootHttpServerFactory;
```

上記の例では、**BackendFeature** が HTTP ルーターサービスのデフォルト実装をオーバーライドするため、Developer Hub がデフォルトの実装を自動的にインストールしないように、**ENABLE_CORE_ROOTHTTPROUTER_OVERRIDE** 環境変数を **true** に設定する必要があります。

11.1. 環境変数のオーバーライド

動的プラグインがコアサービスのオーバーライドをロードできるようにするには、対応するコアサービス ID 環境変数を **true** に設定して Developer Hub バックエンドを起動する必要があります。

表11.1 環境変数およびコアサービス ID

変数	説明
ENABLE_CORE_AUTH_OVERRIDE	**core.auth** サービスをオーバーライドします。
ENABLE_CORE_CACHE_OVERRIDE	**core.cache** サービスをオーバーライドします。
ENABLE_CORE_ROOTCONFIG_OVERRIDE	**core.rootConfig** サービスをオーバーライドします。
ENABLE_CORE_DATABASE_OVERRIDE	**core.database** サービスをオーバーライドします。
ENABLE_CORE_DISCOVERY_OVERRIDE	**core.discovery** サービスをオーバーライドします。
ENABLE_CORE_HTTPAUTH_OVERRIDE	**core.httpAuth** サービスをオーバーライドします。
ENABLE_CORE_HTTPROUTER_OVERRIDE	**core.httpRouter** サービスをオーバーライドします。
ENABLE_CORE_LIFECYCLE_OVERRIDE	**core.lifecycle** サービスをオーバーライドします。
ENABLE_CORE_LOGGER_OVERRIDE	**core.logger** サービスをオーバーライドします。
ENABLE_CORE_PERMISSIONS_OVERRIDE	**core.permissions** サービスをオーバーライドします。
ENABLE_CORE_ROOTHEALTH_OVERRIDE	**core.rootHealth** サービスをオーバーライドします。
ENABLE_CORE_ROOTHTTPROUTER_OVERRIDE	**core.rootHttpRouter** サービスをオーバーライドします。
ENABLE_CORE_ROOTLIFECYCLE_OVERRIDE	**core.rootLifecycle** サービスをオーバーライドします。
ENABLE_CORE_SCHEDULER_OVERRIDE	**core.scheduler** サービスをオーバーライドします。
ENABLE_CORE_USERINFO_OVERRIDE	**core.userInfo** サービスをオーバーライドします。
ENABLE_CORE_URLREADER_OVERRIDE	**core.urlReader** サービスをオーバーライドします。
ENABLE_EVENTS_SERVICE_OVERRIDE	**events.service** サービスをオーバーライドします。