# Red Hat Developer Hub 1.0

## Red Hat Developer Hub のスタートガイド

Last Updated: 2024-04-23

## 法律上の通知



## 概要

このドキュメントでは、Red Hat Developer Hub をインストールして設定するための要件と手順について説明します。

# 目次

# はじめに

開発者は Red Hat Developer Hub を使用して効率的な開発環境を利用できます。Red Hat Developer Hub は一元化されたソフトウェアカタログを採用しており、マイクロサービスとインフラストラクチャーの効率向上を実現します。これにより、製品チームは妥協することなく高品質のコードを提供できるようになります。

# RED HAT DEVELOPER HUB のサポート

本書で説明されている手順で問題が発生した場合は、Red Hat カスタマーポータル (http://access.redhat.com) にアクセスしてください。Red Hat Customer Portal を使用して、Red Hat 製品に関するテクニカルサポート記事の Red Hat ナレッジベースを検索または閲覧できます。Red Hat Global Support Services (GSS) の サポートケース を作成することもできます。製品として "Red Hat Developer Hub" を選択し、適切な製品バージョンを選択してください。

# 第1章 RED HAT DEVELOPER HUB の概要

Red Hat Developer Hub (Developer Hub) は、デベロッパーポータルの構築を目的としたオープンな開発者プラットフォームです。Developer Hub を使用すると、エンジニアリングチームは、開発プロセスを合理化し、高品質のソフトウェアを効率的に構築するためのさまざまなツールやリソースを提供する統合プラットフォームにアクセスできます。

Developer Hub の目標は、以下を使用してデベロッパーポータルの作成と維持に伴う問題に対処することです。

- 生産性を高める利用可能な開発者ツールとリソースをすべて確認できる一元化されたダッシュボード
- エンタープライズクラスのベストプラクティスに準拠したクラウドネイティブアプリケーション開発に役立つセルフサービス機能と guardrails
- 企業全体のすべての開発者に対する適切なセキュリティーとガバナンス

Red Hat Developer Hub は、内部で承認されたツール、プログラミング言語、およびさまざまな開発者リソースを、自己管理型のポータル内で厳選して開発者に提供し、意思決定を簡素化します。このアプローチは、イノベーションを促進しながら、アプリケーション開発の加速とコード品質の維持に貢献します。

# 第2章 HELM チャートを使用した RED HAT DEVELOPER HUB のインストール

柔軟性が高いインストール方法である Red Hat OpenShift (OpenShift) の Helm チャートを使用して、Red Hat Developer Hub をインストールできます。

Helm は OpenShift 上のパッケージマネージャーであり、次の特長を備えています。

- カスタムフックを使用して定期的なアプリケーション更新を適用します。
- 複雑なアプリケーションのインストールを管理します。
- パブリックおよびプライベートサーバーでホストできるチャートを提供します。
- アプリケーションの以前のバージョンへのロールバックをサポートします。

Red Hat Developer Hub の Helm チャートは、Red Hat OpenShift Dedicated、OpenShift Container Platform (OCP) の Helm カタログで入手できます。

**前提条件**

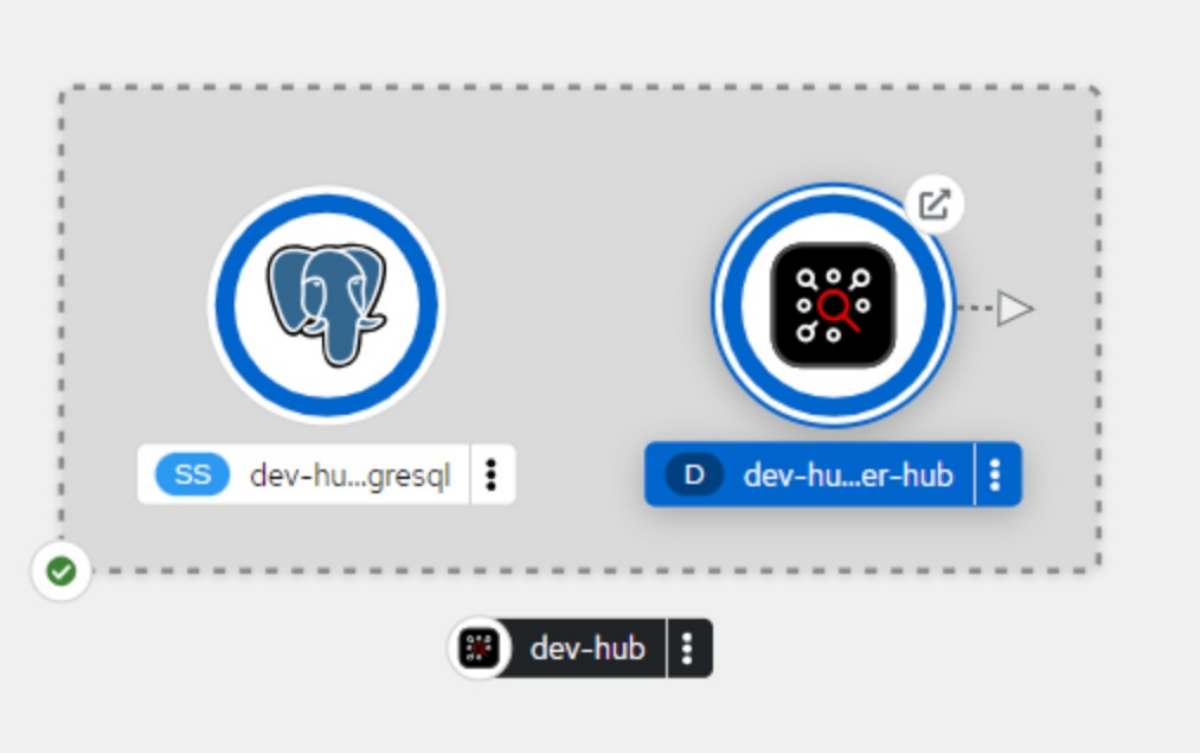
- OpenShift Web コンソールを使用して OCP にログインしている。
- アプリケーションを作成するために、プロジェクト内で適切なロールと権限を設定している。

**手順**

1. OpenShift にプロジェクトが存在しない場合は、作成します。
   OpenShift でのプロジェクトの作成の詳細は、Red Hat OpenShift ドキュメント を参照してください。
2. Red Hat OpenShift Web コンソールで、**Developer** モードに切り替えます。
3. **+Add** をクリックします。
4. **Developer Catalog** パネルから、**Helm Chart** をクリックします。
5. 検索バーで **Developer Hub** を検索し、**Red Hat Developer Hub**カードを選択します。
6. **Create** をクリックします。
7. OpenShift ルーターホスト (例: **apps.<clusterName>.com**) を **Root Schema → global → Shorthand for users who do not want to specify a custom HOSTNAME.Used ONLY with the DEFAULT upstream.backstage.appConfig value and with OCP Route enabled.** にコピーし、必要に応じて他の値を調整します。
8. あるいは、OpenShift ルーターホスト (例: **apps.<clusterName>.com**) を **global.clusterRouterBase** にコピーし、必要に応じて他の値 (**global.clusterRouterBase: apps.example.com** など) を調整します。
   上記の手順で、ホストに関する情報がコピーされます。この情報に Developer Hub のバックエンドがアクセスします。

   OCP ルートが自動的に生成されると、ルートのホスト値が推測され、同じホスト情報が Developer Hub に送信されます。また、値を使用してホストを手動で設定することで Developer Hub がカスタムドメイン上に存在する場合は、カスタムホストが優先されます。

9. **Create** をクリックし、データベースと Red Hat Developer Hub が起動するまで待ちます。
10. **Open URL** オプションをクリックして、Red Hat Developer Hub プラットフォームの使用を開始します。




**注記**

**Developer Hub** Pod が **CrashLoopBackOff** でスタックし、次のログが表示されることがあります。

```
Loaded config from app-config-from-configmap.yaml, env
...
2023-07-24T19:44:46.223Z auth info Configuring "database" as KeyStore provider
type=plugin
Backend failed to start up Error: Missing required config value at
'backend.database.client'
```

この場合、設定ファイルを確認します。これは、設定ファイルが RHDH コンテナーによってアクセスされていないためです。

# 第3章 RED HAT DEVELOPER HUB でサポートされる設定

このセクションでは、Red Hat Developer Hub にアクセスするために必要な以下の設定について説明します。

- カスタムアプリケーション設定
- Developer Hub カタログのソース管理設定

## 3.1. RED HAT OPENSHIFT へのカスタムアプリケーション設定ファイルの追加

Red Hat Developer Hub にアクセスするには、OpenShift にカスタムアプリケーション設定ファイルを追加する必要があります。OpenShift では、次のコンテンツを基本テンプレートとして使用して、**app-config-rhdh** という名前の ConfigMap を作成できます。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    app:
      title: Red Hat Developer Hub
```

**前提条件**

- Red Hat OpenShift でアカウントを作成している。

**手順**

1. OpenShift に移動し、**ConfigMaps** タブを選択します。
2. **Create ConfigMap** をクリックします。
   **Create ConfigMap** ページが表示されます。
3. **Configure via** で **YAML view** オプションを選択し、必要に応じてファイルに変更を加えます。
4. **Create** をクリックします。
5. **Helm** タブに移動します。
   Helm リリースのリストがページに表示されます。
6. Helm リリースの 3 つの点をクリックし、**Upgrade** を選択します。
7. **Root Schema → Backstage Chart Schema → Backstage Parameters → Extra App Configuration files to inline into command arguments** で値を追加します。
   - ConfigMapRef: **app-config-rhdh**
   - filename: **app-config-rhdh.yaml**
8. **Upgrade** をクリックします。

9. あるいは、Helm 値の YAML ビューに移動し、**upstream.backstage.command** の値を **[]** に設定します。

10. **Upgrade** をクリックします。

## 3.2. RED HAT DEVELOPER HUB へのカタログのソース管理の追加

Red Hat Developer Hub にカタログを追加するには、ソフトウェアテンプレートを追加する必要があります。テンプレートを追加するには、GitHub、GitLab、BitBucket などのソース管理を有効にする必要があります。

### 3.2.1. GitHub の統合と認証の設定

Red Hat Developer Hub で GitHub プラグインを有効にするには、GitHub の統合と認証が必要です。

**前提条件**

- GitHub アカウントがある。
- Red Hat OpenShift にアカウントがある。

**手順**

1. OpenShift で、**ConfigMap** をクリックします。
2. 次のように、**app-config-rhdh** ファイルを変更して GitHub 設定を含めます。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    app:
      title: Red Hat Developer Hub
    integrations:
      github:
        - host: github.com
          apps:
            - appId: ${GITHUB_APP_APP_ID}
              clientId: ${GITHUB_APP_CLIENT_ID}
              clientSecret: ${GITHUB_APP_CLIENT_SECRET}
              webhookUrl: ${GITHUB_APP_WEBHOOK_URL}
              webhookSecret: ${GITHUB_APP_WEBHOOK_SECRET}
              privateKey: |
                ${GITHUB_APP_PRIVATE_KEY}
    auth:
      # see https://backstage.io/docs/auth/ to learn about auth providers
      environment: development
      providers:
        github:
          development:
            clientId: ${GITHUB_APP_CLIENT_ID}
            clientSecret: ${GITHUB_APP_CLIENT_SECRET}
```

3. **Save** をクリックします。

4. 以下の手順に従って、前のコードスニペットの環境変数をキーとして使用して、**rhdh-secrets** という名前の、キーまたは値のシークレットファイルを作成します。

   a. Red Hat OpenShift で、**Secrets** タブに移動し、**Create** をクリックします。

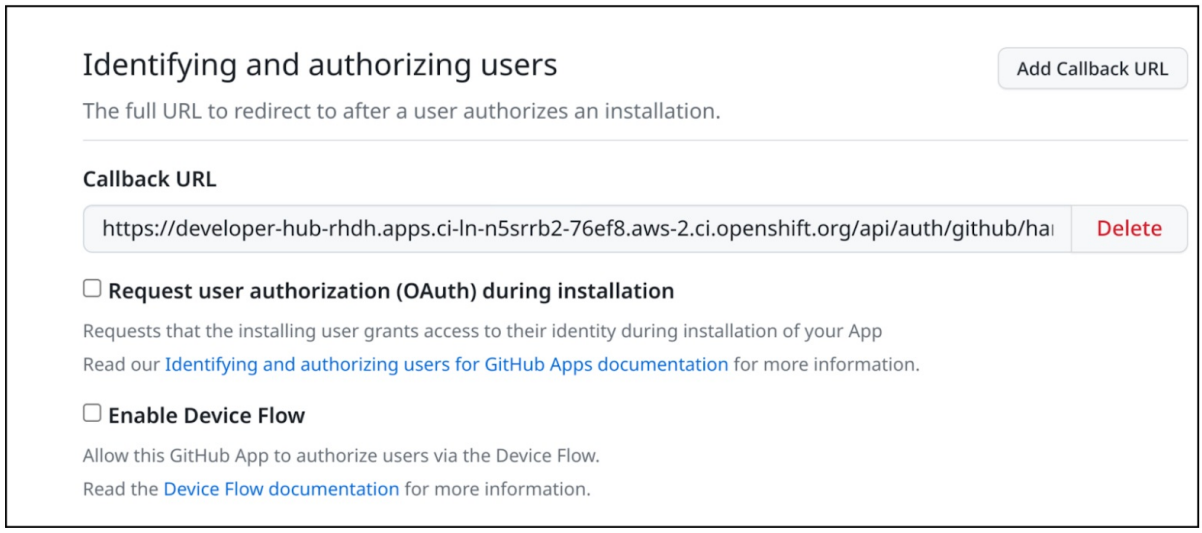
   b. **Secret name** に **rhdh-secrets** と入力します。

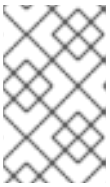
   c. 環境変数を **Key** と **Value** として追加し、**Create** をクリックします。

5. GitHub アプリケーションを使用してログインするには、GitHub アプリケーションの **Callback URL** が次のように設定されていることを確認します。
   **https://developer-hub-<NAMESPACE_NAME>.<OPENSHIFT_ROUTE_HOST>/api/auth/github/handler/frame**

   上記の コールバック URL の例に含まれる **OPENSHIFT_ROUTE_HOST** は、**Root Schema → global → clusterRouteBase** フィールドに追加した API URL です。

   以下はコールバック URL の例です。



**注記**

Developer Hub の概要ページにある GitHub セキュリティーインサイトウィジェットにアクセスするには、GitHub アプリケーションに Read-only Dependabot Alerts 権限があることを確認してください。

6. **Helm** タブに移動し、**Upgrade** を選択します。

7. **Root Schema → Backstage Chart Schema → Backstage Parameters → Backstage container environment variables from existing Secrets** で、値として **rhdh-secrets** を追加します。

8. **Upgrade** をクリックします。

### 3.2.2. Red Hat Developer Hub での GitHub 検出の有効化

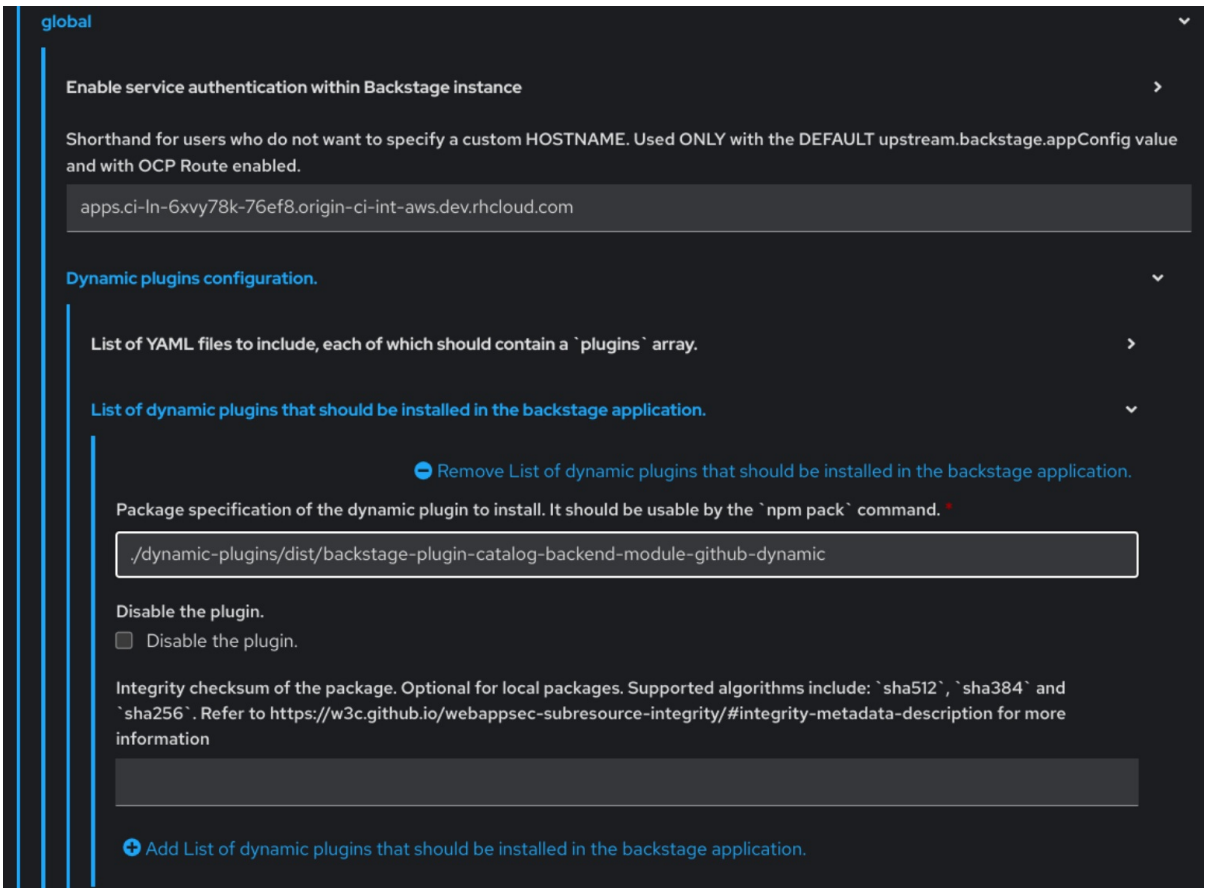
Developer Hub 内のコンポーネント (**catalog-info.yaml** ファイルを含むリポジトリーなど) に対して GitHub の検出機能を有効にすることができます。

**前提条件**

- GitHub 統合を設定している。詳細は、「GitHub の統合と認証の設定」 を参照してください。

**手順**

1. Red Hat Openshift で、**Helm** タブに移動し、**Developer Hub** リリースをアップグレードします。
2. **global → Dynamic plugins configuration → List of dynamic plugins that should be installed in the backstage application. → Package specification of the dynamic plugin to install. It should be usable by the npm pack command.** で、次の値を追加します。
**./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-dynamic**




3. **Upgrade** をクリックします。
4. ConfigMap に次のコードを追加します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    ...
      catalog:
        providers:
          github:
            providerId:
              organization: '${GITHUB_ORG}'
```

```
            schedule:
              frequency:
                minutes: 30
              initialDelay:
                seconds: 15
              timeout:
                minutes: 3
    ...
```

上記のコードの **${GITHUB_ORG}** は、コンポーネントの検出対象とする GitHub 組織に置き換えます。また、プロバイダーが 1 つだけの場合は、次のコードを ConfigMap に追加してください。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    ...
    catalog:
      providers:
        github:
          organization: ${GITHUB_ORG}
          schedule:
            frequency: { minutes: 1 }
            timeout: { minutes: 1 }
            initialDelay: { seconds: 100 }
    ...
```

プロバイダーがリストとなっている場合は、プロバイダーを特定するには、以前のコードの **providerId** が必要です。

5. **Save** をクリックします。

### 3.2.3. Red Hat Developer Hub での GitHub 組織メンバー検出の有効化

GitHub 組織のメンバーに対して GitHub の検出機能を有効にすることもできます。

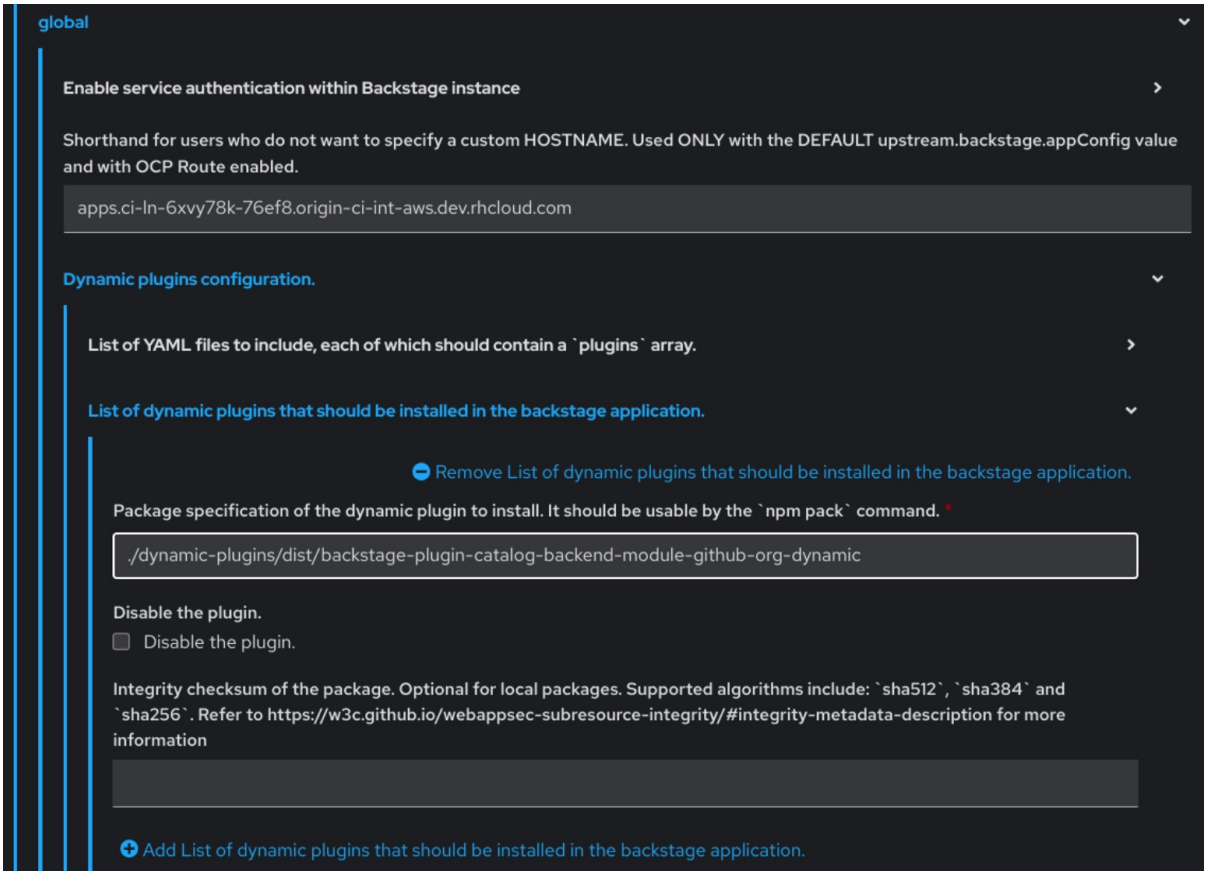
**前提条件**

- GitHub 統合を設定している。詳細は、「GitHub の統合と認証の設定」 を参照してください。

**手順**

1. Red Hat Openshift で、**Helm** タブに移動し、**Developer Hub** リリースをアップグレードします。

2. **global → Dynamic plugins configuration → List of dynamic plugins that should be installed in the backstage application. → Package specification of the dynamic plugin to install. It should be usable by the npm pack command.** で、次の値を追加します。
**./dynamic-plugins/dist/backstage-plugin-catalog-backend-module-github-org-dynamic**




3. **Upgrade** をクリックします。
4. ConfigMap に次のコードを追加します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    ...
    catalog:
      providers:
        githubOrg:
          default:
            id: production
            orgUrl: '${GITHUB_ORG_URL}'
    ...
```

上記のコードの **${GITHUB_ORG_URL}** は、ユーザーの取り込み元の GitHub 組織に置き換えます。

5. **Save** をクリックします。

# 第4章 RED HAT DEVELOPER HUB の HOME ページのカスタマイズ

Red Hat Developer Hub では、Home ページのデータを設定できます。Home ページのデータはプロキシーとして **app-config.yaml** ファイルに渡すことができます。Home ページのデータは、次の方法を使用して提供できます。

- ホストされている JSON ファイル、GitHub または GitLab を使用します。JSON ファイルのデータにアクセスするには、**app-config.yaml** ファイルに次のコードを追加します。

```
proxy:
  endpoints:
    # Other Proxies
    # customize developer hub instance
    '/developer-hub':
      target: <DOMAIN_URL> # i.e https://raw.githubusercontent.com/
      pathRewrite:
        '^/api/proxy/developer-hub': <path to json file> # i.e /janus-idp/backstage-showcase/main/packages/app/public/homepage/data.json
      changeOrigin: true
      secure: true
      # Change to "false" in case of using self hosted cluster with a self-signed certificate
      headers:
 <HEADER_KEY>: <HEADER_VALUE> # optional and can be passed as needed i.e Authorization can be passed for private GitHub repo and PRIVATE-TOKEN can be passed for private GitLab repo
```

- API を使用して JSON 形式の Home ページのデータを提供する別のサービスを使用します。

**注記**

Home ページのデータと Tech Radar のデータを同じサービスで提供する必要はありません。

ホームページと Tech Radar の両方にデータを提供するサンプルサービスとして、**red-hat-developer-hub-customization-provider** を使用できます。**red-hat-developer-hub-customization-provider** サービスは、デフォルトの Developer Hub データと同じデータを提供します。必要に応じて、GitHub から **red-hat-developer-hub-customization-provider** サービスリポジトリーをフォークし、独自のデータで変更できます。

このセクションでは、Developer Hub Helm チャートがデプロイされているクラスターに **red-hat-developer-hub-customization-provider** サービスをデプロイする方法について説明します。

**前提条件**

- Helm チャートを使用して Red Hat Developer Hub をインストールしている。詳細は、*2章Helm チャートを使用した Red Hat Developer Hub のインストール* を参照してください。

**手順**

1. Red Hat OpenShift で、**+Add** を選択し、**Import from Git** オプションをクリックします。
2. Git リポジトリーの URL を **Git Repo URL** フィールドに追加します。

**red-hat-developer-hub-customization-provider** サービスを使用するには、red-hat-developer-hub-customization-provider リポジトリーの URL を追加できます。

3. **General** セクションで、**Name** フィールドの値の名前を **rhdh-customization-provider** に変更し、**Create** をクリックします。

4. **Advanced Options** に移動し、**Target Port** から値をコピーします。
**Target Port** は、通信する Kubernetes または OpenShift サービスを自動的に生成するために使用されます。

5. サービスを表示するには、**OpenShift Administrator** ビューに移動し、**Networking → Service** セクションに移動します。
Topology ビューで **Service Resources** を表示することもできます。

   例を使用してこの手順に従うと、**rhdh-customization-provider** サービスが呼び出され、8080 ポートが追加されます。次の例に示すように、指定した Home ページの API URL から、JSON 形式でデータが返される必要があります。

```
[
  {
    "title": "Dropdown 1",
    "isExpanded": false,
    "links": [
      {
        "iconUrl": "https://imagehost.com/image.png",
        "label": "Dropdown 1 Item 1",
        "url": "https://example.com/"
      },
      {
        "iconUrl": "https://imagehost2.org/icon.png",
        "label": "Dropdown 1 Item 2",
        "url": ""
      }
    ]
  },
  {
    "title": "Dropdown 2",
    "isExpanded": true,
    "links": [
      {
        "iconUrl": "http://imagehost3.edu/img.jpg",
        "label": "Dropdown 2 Item 1",
        "url": "http://example.com"
      }
    ]
  }
]
```

   要求の呼び出しが失敗した場合や、設定されていない場合、Developer Hub インスタンスはデフォルトのローカルデータにフォールバックします。

   Red Hat Developer Hub の Home ページにアクセスするには、ベース URL に **/developer-hub** プロキシーが含まれている必要があります。

6. 次のコードを **app-config-rhdh.yaml** ファイルに追加します。

```
proxy:
  endpoints:
    # Other Proxies
    # customize developer hub instance
    '/developer-hub':
      target: ${HOMEPAGE_DATA_URL}
      changeOrigin: true
      # Change to "false" in case of using self-hosted cluster with a self-signed certificate
      secure: true
```

API 要求の呼び出しが JSON 形式で応答を返すことを確認します。

7. **HOMEPAGE_DATA_URL** を http://<SERVICE_NAME>:8080 と定義します。たとえば、**http://rhdh-customization-provider:8080** とします。
**HOMEPAGE_DATA_URL** を置き換えるには、URL を **rhdh-secrets** に追加するか、カスタム ConfigMap で直接置き換えます。

8. Developer Hub Pod を削除して、変更を取り込みます。
イメージまたはアイコンが読み込まれない場合は、次のように、カスタム ConfigMap のコンテンツセキュリティーポリシー (csp) の **img-src** にイメージまたはアイコンのホスト URL を追加して、イメージまたはアイコンをホワイトリストに登録します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: app-config-rhdh
data:
  app-config-rhdh.yaml: |
    app:
      title: Red Hat Developer Hub
    backend:
      csp:
        connect-src:
          - "'self'"
          - 'http:'
          - 'https:'
        img-src:
          - "'self'"
          - 'data:'
          - <image host url 1>
          - <image host url 2>
          - <image host url 3>
    # Other Configurations
```

その後、Pod を削除して、新しい設定が正しくロードされていることを確認します。

# 第5章 RED HAT DEVELOPER HUB の TECH RADAR ページのカスタマイズ

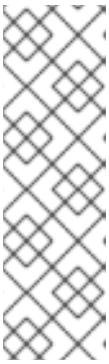
Red Hat Developer Hub では、Helm チャートの動的プラグイン機能を使用しても、Tech Radar ページが有効になりません。

Home ページのカスタマイズと同様に、Tech Radar のベース URL に **/developer-hub/tech-radar** プロキシーが含まれている必要があります。Tech Radar ページのデータは、次の方法を使用して提供できます。

- ホストされている JSON ファイル、GitHub または GitLab を使用します。JSON ファイルのデータにアクセスするには、**app-config.yaml** ファイルに次のコードを追加します。

```
proxy:
  endpoints:
    # Other Proxies
    # customize developer hub instance
    '/developer-hub':
      target: <DOMAIN_URL> # i.e https://raw.githubusercontent.com/
      pathRewrite:
        '^/api/proxy/developer-hub/tech-radar': <path to json file> # i.e /janus-idp/backstage-showcase/main/packages/app/public/tech-radar/data-default.json
  '^/api/proxy/developer-hub': <path to json file> # i.e /janus-idp/backstage-showcase/main/packages/app/public/homepage/data.json
      changeOrigin: true
      secure: true

      # Change to "false" in case of using self hosted cluster with a self-signed certificate
      headers:
 <HEADER_KEY>: <HEADER_VALUE> # optional and can be passed as needed i.e Authorization can be passed for private GitHub repo and PRIVATE-TOKEN can be passed for private GitLab repo
```



**注記**

**tech-radar** と **homepage** のクイックアクセスプロキシーに使用される **pathRewrites** の間には重複が存在するため、**tech-radar** の設定 (**^api/proxy/developer-hub/tech-radar**) は **homepage** の設定 (**^/api/proxy/developer-hub**) よりも前にある必要があります。

Red Hat Developer Hub の Home ページをカスタマイズする方法の詳細は、4章*Red Hat Developer Hub の Home ページのカスタマイズ* を参照してください。

- API を使用して JSON 形式の Tech Radar のデータを提供する別のサービスを使用します。

**前提条件**

- Helm チャートを使用して Red Hat Developer Hub をインストールしている。詳細は、2章*Helm チャートを使用した Red Hat Developer Hub のインストール* を参照してください。

**手順**

1. 次のコードを **app-config-rhdh.yaml** ファイルに追加します。

```
proxy:
  endpoints:
    # Other Proxies
    '/developer-hub/tech-radar':
      target: ${TECHRADAR_DATA_URL}
      changeOrigin: true
      # Change to "false" in case of using self hosted cluster with a self-signed certificate
      secure: true
```

API 要求の呼び出しが JSON 形式で応答を返すことを確認します。

2. **TECHRADAR_DATA_URL** を **http://<SERVICE_NAME>/tech-radar** と定義します (例: **http://rhdh-customization-provider/tech-radar**)。

**注記**

**TECHRADAR_DATA_URL** を定義するには、URL を **rhdh-secrets** に追加するか、カスタム ConfigMap 内の URL 値を直接置き換えます。

3. Developer Hub Pod を削除して、変更を取り込みます。

# 第6章 RED HAT DEVELOPER HUB の追加のカスタマイズ

このセクションでは、Red Hat Developer Hub に適用できる追加のカスタマイズオプションについて説明します。

### タブのツールチップのカスタマイズ

タブのツールチップをカスタマイズするには、次の内容を **app-config-rhdh.yaml** ファイルに追加します。

```
app:
  title: My custom developer hub
```

### Developer Hub インスタンスのブランディングのカスタマイズ

Developer Hub インスタンスのブランディングをカスタマイズするには、次の内容を **app-config-rhdh.yaml** ファイルに追加します。

```
app:
  branding:
    fullLogo: ${BASE64_EMBEDDED_FULL_LOGO}
    iconLogo: ${BASE64_EMBEDDED_ICON_LOGO}
    theme:
      light:
        primaryColor: ${PRIMARY_LIGHT_COLOR}
        headerColor1: ${HEADER_LIGHT_COLOR_1}
        headerColor2: ${HEADER_LIGHT_COLOR_2}
        navigationIndicatorColor: ${NAV_INDICATOR_LIGHT_COLOR}
      dark:
        primaryColor: ${PRIMARY_DARK_COLOR}
        headerColor1: ${HEADER_DARK_COLOR_1}
        headerColor2: ${HEADER_DARK_COLOR_2}
        navigationIndicatorColor: ${NAV_INDICATOR_DARK_COLOR}
```

上記の設定の各部分について説明します。

- **fullLogo** は、展開された (固定された) サイドバー上のロゴであり、base64 でエンコードされた画像を想定しています。
- **iconLogo** は、折りたたまれた (固定されていない) サイドバーのロゴであり、base64 でエンコードされた画像を想定しています。
- **primaryColor** は、リンクとほとんどのボタンの色を、入力された色に設定します。**primaryColor** でサポートされている形式は次のとおりです。
  - **#nnn**
  - **#nnnnnn**
  - **rgb()**
  - **rgba()**
  - **hsl()**
  - **hsla()**

- **color()**

- **headerColor1** (バナーの左側) と **headerColor2** (バナーの右側) は、各ページのヘッダーバナーの色と、テンプレートカードのバナーの色を変更します。**headerColor1** と **headerColor2** でサポートされている形式は次のとおりです。
  - **#nnn**
  - **#nnnnnn**
  - **rgb()**
  - **rgba()**
  - **hsl()**
  - **hsla()**
  - **color()**

- **navigationIndicatorColor** は、現在どのタブにいるかを示すサイドバーのインジケーターの色を変更します。**navigationIndicatorColor** でサポートされている形式は次のとおりです。
  - **#nnn**
  - **#nnnnnn**
  - **rgb()**
  - **rgba()**
  - **hsl()**
  - **hsla()**
  - **color()**

# 第7章 RED HAT DEVELOPER HUB のテーマのカスタマイズ

Red Hat Developer Hub (Developer Hub) のテーマモードをカスタマイズできます。

RHDH は次のテーマモードをサポートしています。

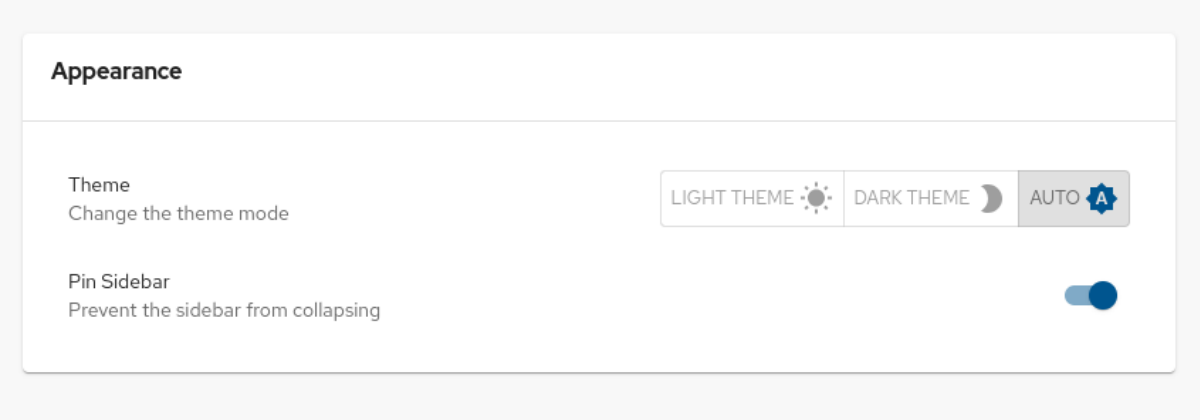
- ライトテーマ (デフォルト)
- ダークテーマ
- 自動

**前提条件**

- RHDH Web コンソールにログインしている。

**手順**

1. **Settings** をクリックします。
2. **Appearance** パネルで、**LIGHT THEME**、**DARK THEME**、または **AUTO** をクリックしてテーマモードを変更します。



# 第8章 RED HAT DEVELOPER HUB の SERVICENOW カスタムアクション

**重要**

本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、Red Hat の実稼働環境での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。これらの機能は、今後の製品機能への早期アクセスを提供することで、お客様が機能をテストし、開発プロセス中にフィードバックを提供できるようにしています。

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

## 8.1. 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。

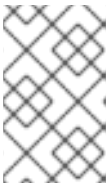
**前提条件**

- 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、*2章Helm チャートを使用した 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ている。

**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/janus-idp-backstage-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}
```

## 8.2. 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

### 8.2.1. 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8.1 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 | 応答で返されるフィールドの配列 |
| **sysparmView** | **string** | 任意 | 指定された UI ビューに従って応答を表示します。**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8.2 出力パラメーター**

| 名前 | 型 | 説明 |
| --- | --- | --- |
| **result** | **Record<PropertyKey, unknown>** | 要求の応答ボディー |

[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8.3 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 | 応答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 ビューに従って応答を表示します。**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8.4 出力パラメーター**

| 名前 | 型 | 説明 |
|---|---|---|
| **result** | **Record<PropertyKey, unknown>** | 要求の応答ボディー |

[POST] servicenow:now:table:createRecord

Developer Hub のテーブルにレコードを作成します。

**表8.5 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 | 応答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 ビューに従って応答を表示します。**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8.6 出力パラメーター**

| 名前 | 型 | 説明 |
| --- | --- | --- |
| **result** | **Record<PropertyKey, unknown>** | 要求の応答ボディー |

[PUT] servicenow:now:table:modifyRecord

Developer Hub のテーブルのレコードを変更します。

**表8.7 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 | 応答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 ビューに従って応答を表示します。**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8.8 出力パラメーター**

| 名前 | 型 | 説明 |
|---|---|---|
| **result** | **Record<PropertyKey, unknown>** | 要求の応答ボディー |

[PATCH] servicenow:now:table:updateRecord

Developer Hub のテーブルのレコードを更新します。

**表8.9 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 | 応答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 ビューに従って応答を表示します。**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8.10 出力パラメーター**

| 名前 | 型 | 説明 |
|---|---|---|
| **result** | **Record<PropertyKey, unknown>** | 要求の応答ボディー |

[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8.11 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。 |