Red Hat OpenShift Dev Spaces 3.2

管理ガイド

Red Hat OpenShift Dev Spaces 3.2 の管理

Last Updated: 2022-10-30

Red Hat OpenShift Dev Spaces 3.2 管理ガイド

Red Hat OpenShift Dev Spaces 3.2 の管理

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

管理者による Red Hat OpenShift Dev Spaces の操作に関する情報。

目次

第1章 インストールの準備

OpenShift Dev Spaces インストールを準備するには、OpenShift Dev Spaces エコシステムおよびデプロイメントの制約について確認します。

- 「サポートされるプラットフォーム」
- 「OpenShift Dev Spaces アーキテクチャー」
- 「OpenShift Dev Spaces リソース要件の計算」
- 「**CheCluster** カスタムリソースについて」

1.1. サポートされるプラットフォーム

OpenShift Dev Spaces 3.2 は、一覧表示されたサポートされるインストール方法を使用して、一覧表示されたプラットフォームで利用できます。

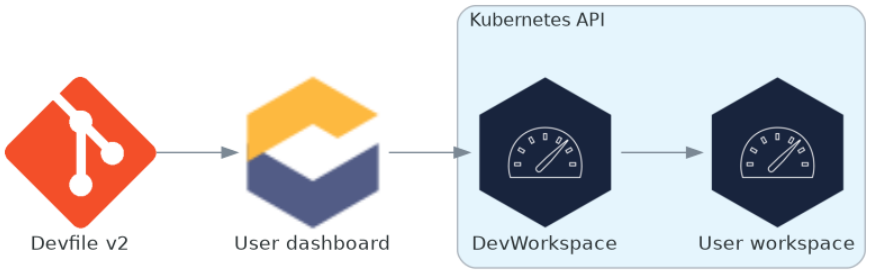
表1.1 OpenShift Dev Spaces 3.2 でサポートされるデプロイメント環境

プラットフォーム	アーキテクチャー	デプロイメント方法
OpenShift Container Platform 4.10	• AMD64 および Intel 64 (**x**86_64) • IBM Power (**ppc64le**) • IBM Z (**s390x**)	• OpenShift Web コンソール • **DSC** 管理ツール
OpenShift Container Platform 4.11	• AMD64 および Intel 64 (**x**86_64) • IBM Power (**ppc64le**) • IBM Z (**s390x**)	• OpenShift Web コンソール • **DSC** 管理ツール
OpenShift Dedicated 4.10	• AMD64 および Intel 64 (**x**86_64)	• OpenShift Web コンソール
OpenShift Dedicated 4.11	• AMD64 および Intel 64 (**x**86_64)	• OpenShift Web コンソール
Red Hat OpenShift Service on AWS (ROSA) 4.10	• AMD64 および Intel 64 (**x**86_64)	• OpenShift Web コンソール

プラットフォーム	アーキテクチャー	デプロイメント方法
Red Hat OpenShift Service on AWS (ROSA) 4.11	• AMD64 および Intel 64 (**x**86_64)	• OpenShift Web コンソール

1.2. OPENSHIFT DEV SPACES アーキテクチャー

図1.1 Dev Workspace Operator を使用した高度な OpenShift DevSpaces アーキテクチャー



OpenShift Dev Spaces は、3 つのコンポーネントのグループで実行されます。

OpenShift Dev Spaces サーバーコンポーネント

ユーザープロジェクトおよびワークスペースの管理。主な設定要素はユーザーダッシュボードで、ユーザーはここから自分のワークスペースを制御します。

DevWorkspace operator

User ワークスペースの実行に必要な OpenShift オブジェクトを作成し、制御します。**Pods**、**Services**、**PeristentVolumes** を含みます。

User ワークスペース

コンテナーベースの開発環境、IDE を含みます。

これらの OpenShift の機能のロールは中心的なものです。

DevWorkspace カスタムリソース

ユーザーワークスペースを表す有効な OpenShift オブジェクト。OpenShift Dev Spaces で操作します。3 つのグループのコンポーネントのコミュニケーションチャンネルとなります。

OpenShift のロールベースアクセスコントロール (RBAC)

すべてのリソースへのアクセスを制御します。

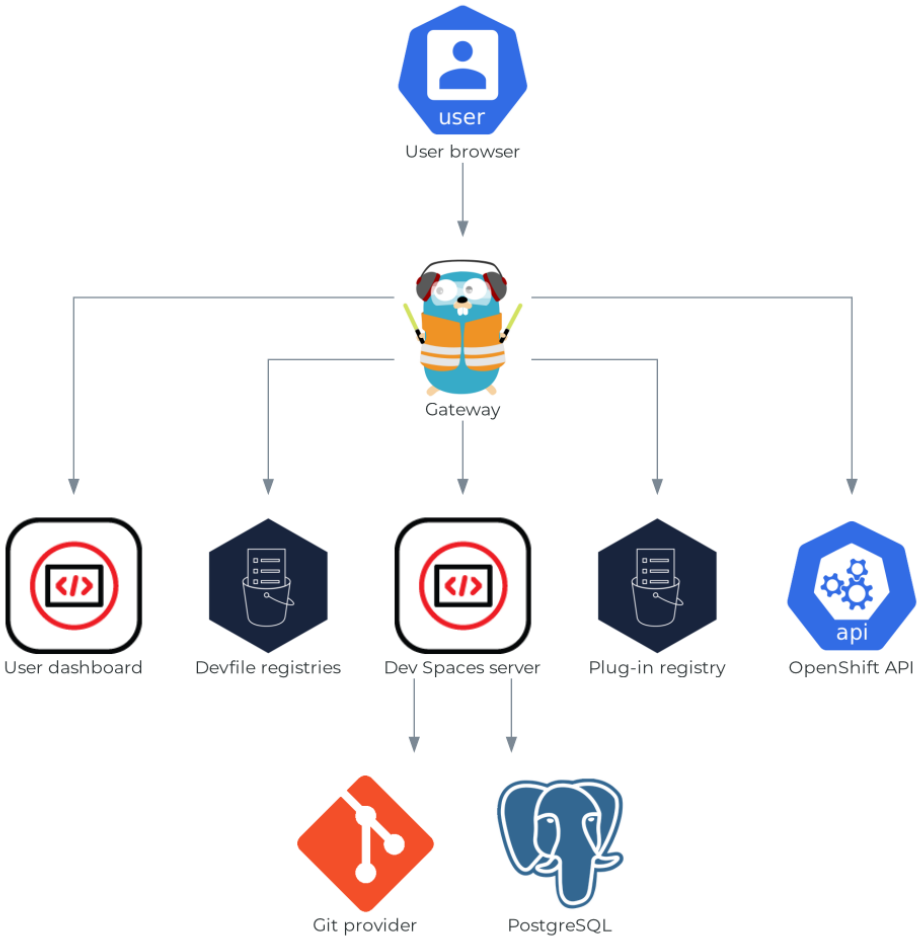
関連情報

- 「OpenShift Dev Spaces サーバーコンポーネント」
- 「DevWorkspace operator」
- 「User ワークスペース」
- DevWorkspace Operator repository
- Kubernetes のドキュメント - カスタムリソース

1.2.1. OpenShift Dev Spaces サーバーコンポーネント

OpenShift Dev Spaces サーバーコンポーネントにより、マルチテナンシーとワークスペースの管理が確保されます。

図1.2 Dev Workspace Operator と対話する OpenShift DevSpaces サーバーコンポーネント



関連情報

- 「OpenShift Dev Spaces 演算子」
- 「DevWorkspace operator」
- 「ゲートウェイ」
- 「ユーザーダッシュボード」
- 「Devfile レジストリー」
- 「OpenShift Dev Spaces サーバー」
- 「PostgreSQL」
- 「プラグインレジストリー」

1.2.1.1. OpenShift Dev Spaces 演算子

OpenShift Dev Spaces Operator は、OpenShift Dev Spaces サーバーコンポーネントの完全なライフサイクル管理を行います。これには、以下が含まれます。

CheCluster カスタムリソース定義 (CRD)

CheCluster OpenShift オブジェクトを定義します。

OpenShift Dev Spaces コントローラー

Pod、サービス、永続ボリュームなどの OpenShift Dev Space インスタンスを実行するために必要な OpenShift オブジェクトを作成し、制御します。

CheCluster カスタムリソース (CR)

OpenShift Dev Spaces Operator を持つクラスターでは、**CheCluster** カスタムリソース (CR) を作成できます。OpenShift Dev Spaces オペレーターは、この OpenShift Dev Spaces インスタンス上で OpenShift Dev Spaces サーバーコンポーネントの完全なライフサイクル管理を行います。

- 「DevWorkspace operator」
- 「ゲートウェイ」
- 「ユーザーダッシュボード」
- 「Devfile レジストリー」
- 「OpenShift Dev Spaces サーバー」
- 「PostgreSQL」
- 「プラグインレジストリー」

関連情報

- 「**CheCluster** カスタムリソースについて」
- *2章OpenShift Dev Space のインストール*

1.2.1.2. DevWorkspace operator

DevWorkspace Operator は OpenShift を拡張して DevWorkspace サポートを提供します。これには、以下が含まれます。

DevWorkspace のカスタムリソース定義

Devfile v2 仕様から DevWorkspace OpenShift オブジェクトを定義します。

DevWorkspace コントローラー

Pod、サービス、永続ボリュームなど、DevWorkspace の実行に必要な OpenShift オブジェクトを作成して制御します。

DevWorkspace カスタムリソース

DevWorkspace 演算子があるクラスターでは、DevWorkspace カスタムリソース (CR) を作成することができます。DevWorkspace CR は、Devfile を OpenShift で表現したものです。OpenShift クラスター内の User ワークスペースを定義します。

関連情報

- Devfile API リポジトリー

1.2.1.3. ゲートウェイ

OpenShift Dev Spaces ゲートウェイには、以下のロールがあります。

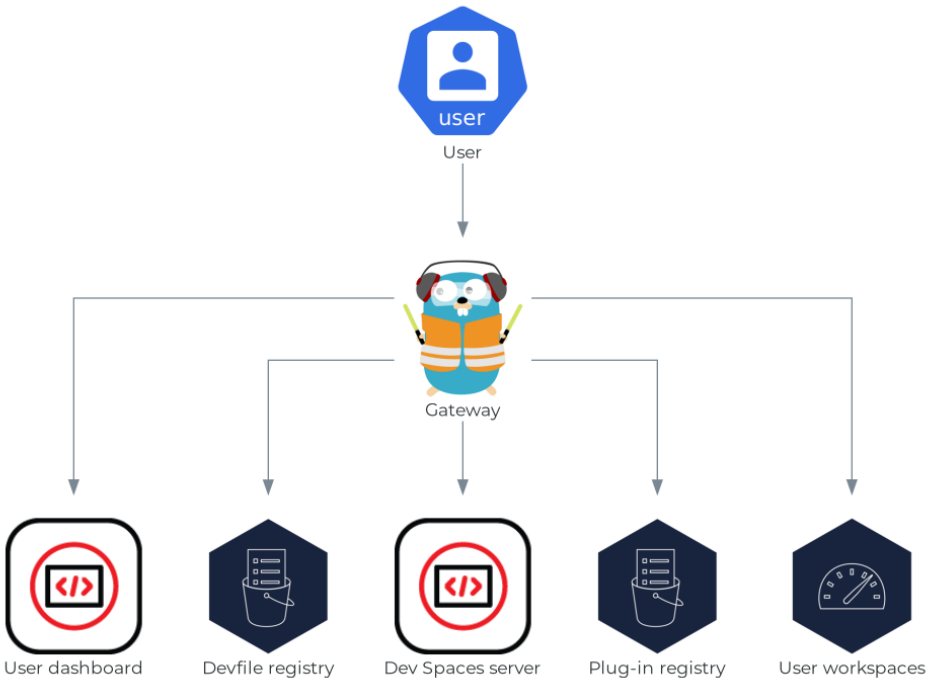
- 要求をルーティングする。Traefik を使用します。
- OpenID Connect(OIDC) でユーザーを認証する。OpenShift OAuth2 プロキシー を使用します。
- OpenShift RoleBased Access Control(RBAC) ポリシーを適用して、OpenShift Dev Spaces リソースへのアクセスを制御します。'kube-rbac-proxy' を使用します。

OpenShift Dev Spaces Operator はこれを **che-gateway** Deployment として管理します。

以下へのアクセスを制御します。

- 「ユーザーダッシュボード」
- 「Devfile レジストリー」
- 「OpenShift Dev Spaces サーバー」
- 「プラグインレジストリー」
- 「User ワークスペース」

図1.3 OpenShift Dev Spaces ゲートウェイと他のコンポーネントとの対話



関連情報

- 「ID および承認の管理」

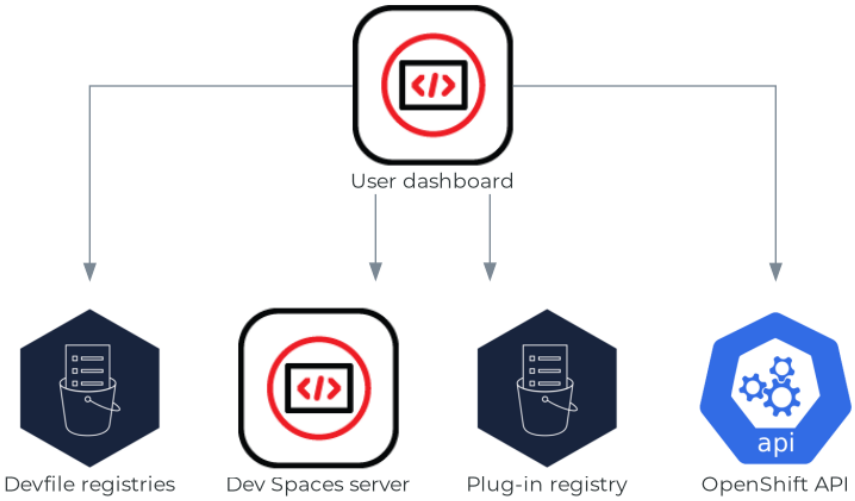
1.2.1.4. ユーザーダッシュボード

ユーザーダッシュボードは、Red Hat OpenShift Dev Spaces のランディングページです。OpenShift Dev Spaces ユーザーは、ユーザーダッシュボードを参照してワークスペースにアクセスし、管理します。これは React のアプリケーションです。OpenShift Dev Spaces デプロイメントは、**devspaces-dashboard** Deployment で起動します。

以下にアクセスする必要があります。

- 「Devfile レジストリー」
- 「OpenShift Dev Spaces サーバー」
- 「プラグインレジストリー」
- OpenShift API

図1.4 User ダッシュボードと他のコンポーネントとの対話



ユーザーがユーザーダッシュボードにワークスペースの起動を要求すると、ユーザーダッシュボードはこの一連のアクションを実行します。

1. ユーザーがコードサンプルからワークスペースを作成する際に、「Devfile レジストリー」 から devfile を収集します。
2. リポジトリー URL を 「OpenShift Dev Spaces サーバー」 に送信し、ユーザーがリモート devfile からワークスペースを作成する際に devfile が返されることを想定します。
3. ワークスペースを記述した devfile を読み込みます。
4. 「プラグインレジストリー」 から追加のメタデータを収集します。
5. その情報を DevWorkspace Custom Resource に変換します。
6. OpenShift API を使用して、ユーザープロジェクトに DevWorkspace Custom Resource を作成します。
7. DevWorkspace カスタムリソースのステータスを監視します。
8. 実行中のワークスペース IDE にユーザーをリダイレクトします。

1.2.1.5. Devfile レジストリー

関連情報

OpenShift Dev Spaces devfile レジストリーは、すぐに使用できるワークスペースを作成するためのサ

ンプル devfile の一覧を提供するサービスです。「ユーザーダッシュボード」 は、**Dashboard → Create Workspace** ページにサンプルリストを表示します。各サンプルには、Devfile v2 が含まれています。OpenShift Dev Spaces デプロイメントでは、**devfile-registry** デプロイメントで 1 つの devfile レジストリーインスタンスを起動します。

図1.5 他のコンポーネントとの相互作用を登録する Devfile

関連情報

- Devfile v2 ドキュメント
- devfile レジストリーの最新のコミュニティーバージョンのオンラインインスタンス
- OpenShift Dev Spaces devfile レジストリーリポジトリー

1.2.1.6. OpenShift Dev Spaces サーバー

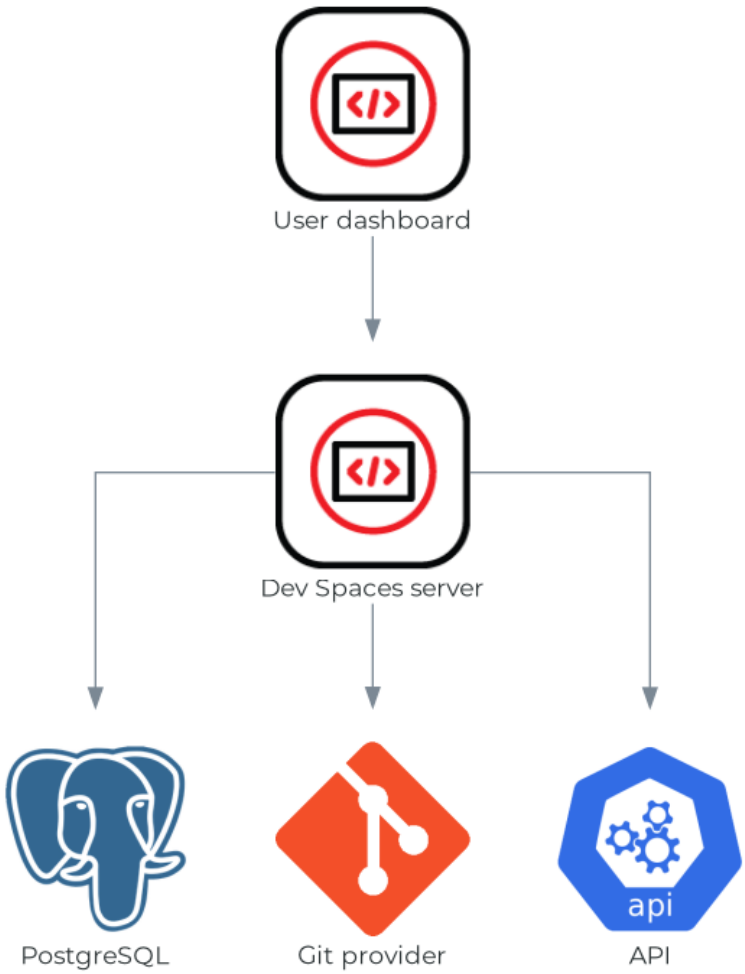
OpenShift Dev Spaces サーバーの主な機能は次のとおりです。

- ユーザーネームスペースの作成
- ユーザーネームスペースに必要なシークレットと設定マップのプロビジョニング
- Git サービスプロバイダーとの統合による devfile の取得および認証

OpenShift Dev Spaces サーバーは、HTTP REST API を公開する Java Web サービスで、以下へのアクセスが必要です。

- 「PostgreSQL」
- Git サービスプロバイダー
- OpenShift API

図1.6 OpenShift Dev Spaces サーバーと他のコンポーネントとの対話



関連情報

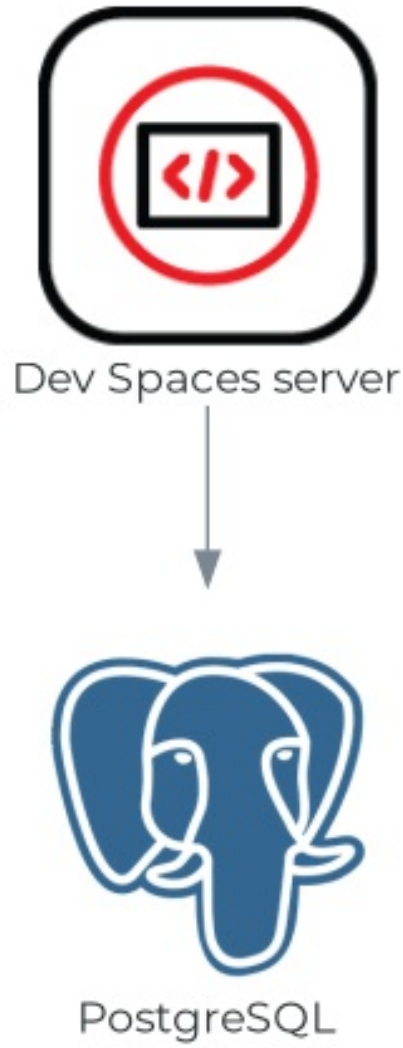
- 「OpenShift Dev Spaces サーバーコンポーネントの詳細な設定オプション」

1.2.1.7. PostgreSQL

OpenShift Dev Spaces サーバーは、PostgreSQL データベースを使用してワークスペースのメタデータなどのユーザー設定を永続化します。

OpenShift Dev Spaces デプロイメントでは、**postgres** Deployment で専用の PostgreSQL インスタンスを起動します。代わりに外部データベースを使用することができます。

図1.7 Postgre SQL と他のコンポーネントとの対話



関連情報

- **quay.io/eclipse/che—centos—postgresql-96-centos7** container image
- **quay.io/eclipse/che—centos—postgresql-13-centos7** container image

1.2.1.8. プラグインレジストリー

各 OpenShift Dev Spaces ワークスペースは、特定のエディターおよび関連する拡張機能のセットで始まります。OpenShift Dev Spaces プラグインレジストリーは、利用可能なエディターおよびエディターエクステンションの一覧を提供します。各エディターや拡張機能については、Devfile v2 に記載されています。

「ユーザーダッシュボード」 は、レジストリーの内容を読み取っています。

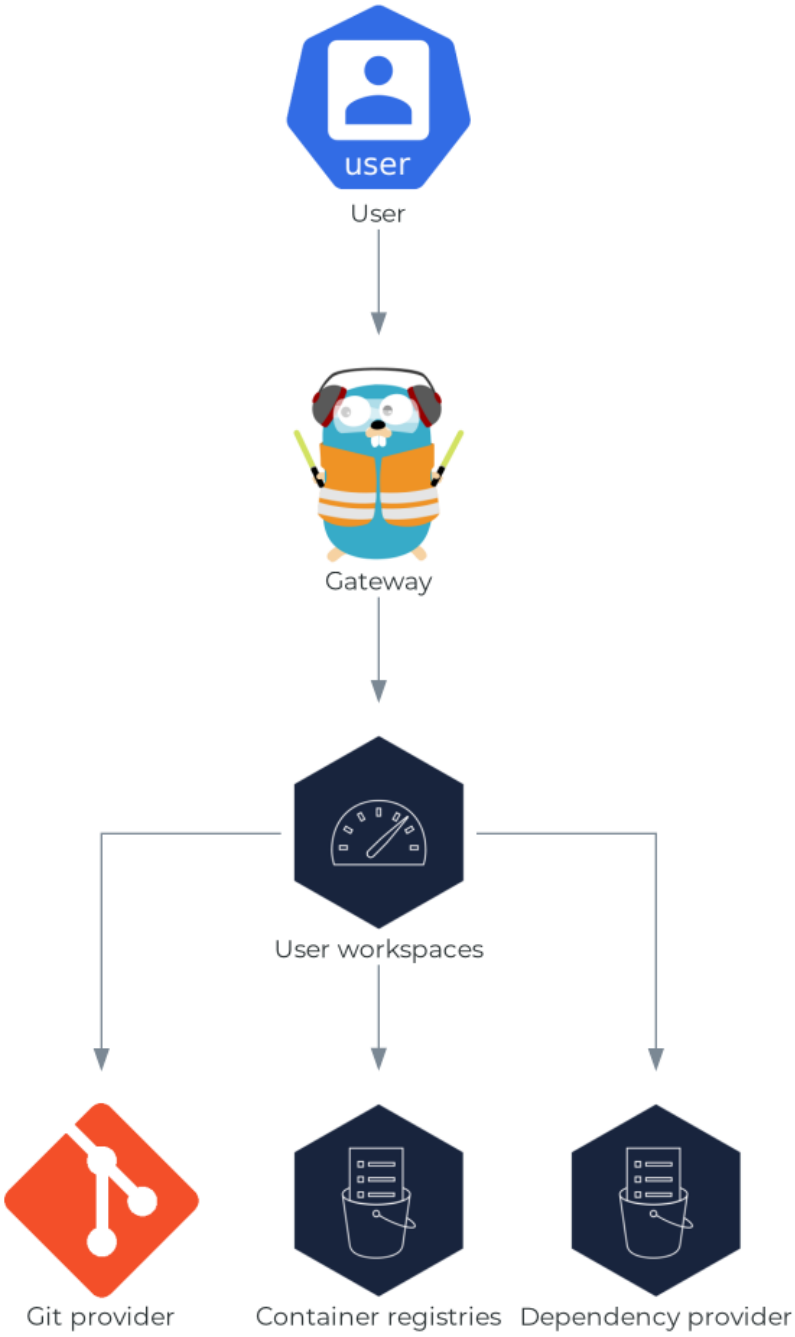
図1.8 プラグインは、他のコンポーネントとの相互作用を登録します。

関連情報

- OpenShift Dev Spaces プラグインレジストリーリポジトリーでのエディター定義
- OpenShift Dev Spaces プラグインレジストリーリポジトリーでのプラグイン定義
- Plug-in registry latest community version online instance

1.2.2. User ワークスペース

図1.9 User ワークスペースと他のコンポーネントとの対話



User ワークスペースは、コンテナー内で動作する Web IDE です。

User ワークスペースは、Web アプリケーションです。コンテナー内で動作するマイクロサービスで設定されており、ブラウザー上で動作する最新の IDE のすべてのサービスを提供します。

- エディター
- 言語オートコンプリート
- 言語サーバー
- デバッグツール
- プラグイン
- アプリケーションのランタイム

ワークスペースは、ワークスペースコンテナーと有効なプラグイン、および関連する OpenShift コンポーネントを含む 1 つの OpenShift Deployment です。

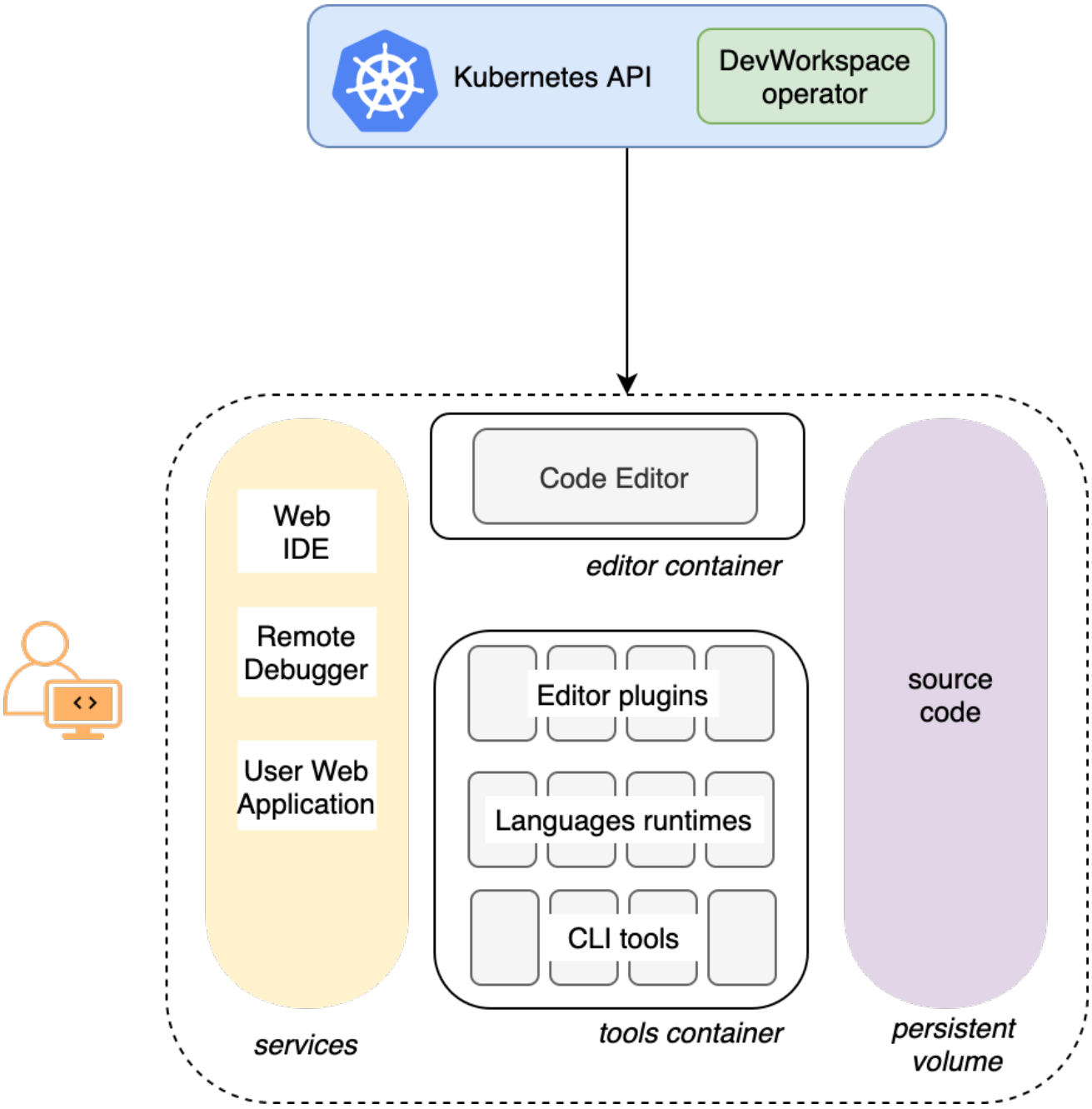
- コンテナー
- ConfigMap
- サービス
- エンドポイント
- ingress またはルート
- シークレット
- 永続ボリューム (PV)

OpenShift Dev Spaces ワークスペースには、OpenShift 永続ボリューム (PV) で永続化されるプロジェクトのソースコードが含まれます。マイクロサービスは、この共有ディレクトリーに読み書き可能なアクセス権があります。

devfile v2 形式を使用して、OpenShift Dev Spaces ワークスペースのツールおよびランタイムアプリケーションを指定します。

以下の図は、OpenShift Dev Spaces ワークスペースとそのコンポーネントを実行する 1 つを示しています。

図1.10 OpenShift Dev Spaces ワークスペースコンポーネント



この図では、実行中のワークスペースが 1 つあります。

1.3. OPENSHIFT DEV SPACES リソース要件の計算

OpenShift Dev Spaces Operator、DevWorkspace Controller、およびユーザーワークスペースは Pod のセットで設定されます。これらの Pod は、CPU および RAM の制限および要求の点でリソース消費に貢献します。Red Hat OpenShift Dev Spaces の実行に必要なメモリーや CPU などのリソースを計算する方法を説明します。

1.3.1. OpenShift Dev Spaces Operator の要件

OpenShift Dev Spaces Operator は、6 つの異なる Pod で実行される 6 つのオペランドを管理します。以下の表は、これらのオペランドのデフォルトのリソース要件を示しています。

表1.2 OpenShift Dev Spaces Operator オペランド

Pod	コンテナー名	デフォルトのメモリー制限	デフォルトのメモリー要求
OpenShift Dev Spaces Server	OpenShift Dev Spaces	1 Gi	512 MiB
OpenShift Dev Spaces Gateway	**gateway**、**configbump**、**oauth-proxy**、**kube-rbac-proxy**	4 Gi、256Mi、512Mi、512Mi	128 Mi、64Mi、64Mi、64Mi
OpenShift Dev Spaces Dashboard	**OpenShift Dev Spaces-dashboard**	256 Mi	32 Mi
PostgreSQL	**postgres**	1 Gi	512 Mi
devfile レジストリー	**che-devfile-registry**	256 Mi	32 Mi
プラグインレジストリー	**che-plugin-registry**	256 Mi	32 Mi

すべてのオペランドを有効にする OpenShift Dev Spaces Operator は、**64Mi** メモリー要求と **256Mi** 制限を持つ単一コンテナーで設定されます。これらのデフォルト値は、OpenShift Dev Spaces Operator が比較的大量の OpenShift Dev Spaces ワークスペースを管理する場合に十分です。大規模なデプロイメントでは、デフォルト値を増やすことを検討してください。

関連情報

- 「OpenShift Dev Spaces アーキテクチャー」

1.3.2. DevWorkspace Operator の要件

DevWorkspace Operator は 3 つの Pod で設定されます。以下の表は、これらの各 Pod のデフォルトのリソース要件を示しています。

表1.3 DevWorkspace Operator Pods

Pod	コンテナー名	デフォルトのメモリー制限	デフォルトのメモリー要求
DevWorkspace Controller Manager	**DevWorkspace-controller**、**kube-rbac-proxy**	1 Gi	100 Mi
DevWorkspace Operator Catalog	**registry-server**	該当なし	50 Mi
DevWorkspace Webhook Server	**webhook-server}**、kube-rbac-proxy	300 Mi	20 Mi

これらのデフォルト値は、DevWorkspace Controller が比較的大量の OpenShift DevSpaces ワークスペースを管理する場合に十分です。大規模なデプロイメントでは、デフォルト値を増やすことを検討してください。

関連情報

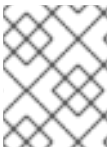
- 「OpenShift Dev Spaces アーキテクチャー」

1.3.3. ワークスペースの要件

本セクションでは、ワークスペースに必要なリソースを計算する方法を説明します。これは、ワークスペースの各コンテナーに必要なリソースの合計です。

手順

1. devfile の components セクションに明示的に指定されるワークスペース **components** を特定します。
2. 暗黙的なワークスペースコンポーネントを特定します。



注記

OpenShift Dev Spaces は、デフォルトの **theia-ide**、**che-machine-exec**、**che-gateway** コンテナーを暗黙的に読み込みます。

1. 各コンポーネントの要件を計算します。

関連情報

- 「OpenShift Dev Spaces アーキテクチャー」

1.3.4. ワークスペースの例

このセクションでは、OpenShift Dev Spaces ワークスペースの例について説明します。

以下の devfile は、OpenShift Dev Spaces ワークスペースを定義します。

```
schemaVersion: 2.1.0
metadata:
  name: bash
components:
  - name: tools
    container:
      image: quay.io/devfile/universal-developer-image:ubi8-0e189d9
      memoryLimit: 4Gi

commands:
  - id: run-main-script
    exec:
      label: "Run main.sh script"
      component: tools
      workingDir: '${PROJECT_SOURCE}'
      commandLine: |
        ./main.sh
      group:
        kind: run
        isDefault: true
```

この表は、各ワークスペースコンポーネントのメモリー要件を示しています。

表1.4 ワークスペースメモリー要件および制限の合計

Pod	コンテナー名	デフォルトのメモリー制限	デフォルトのメモリー要求
ワークスペース	**theia-ide**	512 Mi	64 Mi
ワークスペース	**machine-exec**	128 Mi	32 Mi
ワークスペース	**tools**	4 Gi	64 Mi
ワークスペース	**che-gateway**	256 Mi	64 Mi
	合計	**4.9 Gi**	**224 Mi**

関連情報

- 「OpenShift Dev Spaces アーキテクチャー」
- 「**CheCluster** カスタムリソースについて」
- OpenShift Dev Spaces プラグインレジストリーリポジトリー
- Kubernetes resource management for pods and containers

第2章 OPENSHIFT DEV SPACE のインストール

このセクションでは、Red Hat OpenShift Dev Spaces をインストールする手順を説明します。

OpenShift Dev Spaces のインスタンスは、クラスターごとに 1 つだけデプロイできます。

- 「Web コンソールを使用した OpenShift への OpenShift Dev Spaces のインストール」
- 「**dsc** 管理ツールを使用した OpenShift への OpenShift Dev Spaces のインストール」
- 「OpenShift の制限された環境での OpenShift Dev Space のインストール」

2.1. DSC 管理ツールのインストール

Red Hat OpenShift Dev Spaces コマンドライン管理ツールである **dsc** は、Microsoft Windows、Apple MacOS、および Linux にインストールできます。**dsc** を使用すると、サーバーの起動、停止、更新、削除など、OpenShift Dev Spaces サーバーの操作を実行できます。

前提条件

- Linux または macOS の場合:

注記

Windows に **dsc** をインストールする場合は、以下のページを参照してください。

- https://developers.redhat.com/products/openshift-dev-spaces/download
- https://github.com/redhat-developer/devspaces-chectl

手順

1. https://developers.redhat.com/products/openshift-dev-spaces/download から **$HOME** などのディレクトリーにアーカイブをダウンロードします。
2. アーカイブで **tar xvzf** を実行して、**/dsc** ディレクトリーを展開します。
3. 展開した **/dsc/bin** サブディレクトリーを **$PATH** に追加します。

検証

- **dsc** を実行して、その情報を表示します。

```
$ dsc
```

関連情報

- DSC リファレンスドキュメント

2.2. DSC 管理ツールを使用した OPENSHIFT への OPENSHIFT DEV SPACES のインストール

OpenShift Dev Spaces を OpenShift にインストールできます。

前提条件

- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the OpenShift CLI を参照してください。
- **dsc**。「dsc 管理ツールのインストール」 を参照してください。

手順

1. オプション: この OpenShift クラスターに OpenShift Dev Spaces をデプロイした場合は、以前の OpenShift Dev Spaces インスタンスが削除されていることを確認してください。

```
$ dsc server:delete
```

2. OpenShift Dev Spaces インスタンスを作成します。

```
$ dsc server:deploy --platform openshift
```

検証手順

1. OpenShift Dev Spaces インスタンスのステータスを確認します。

```
$ dsc server:status
```

2. OpenShift Dev Spaces クラスターインスタンスに移動します。

```
$ dsc dashboard:open
```

2.3. WEB コンソールを使用した OPENSHIFT への OPENSHIFT DEV SPACES のインストール

このセクションでは、OpenShift Web コンソールを使用して OpenShift Dev Spaces をインストールする方法について説明します。代わりに 「**dsc** 管理ツールを使用した OpenShift への OpenShift Dev Spaces のインストール」 を検討してください。

前提条件

- クラスター管理者による OpenShift Web コンソールセッション。Accessing the web console を参照してください。

手順

1. オプション: この OpenShift クラスターに OpenShift Dev Spaces をデプロイした場合は、以前の OpenShift Dev Spaces インスタンスが削除されていることを確認してください。

```
$ dsc server:delete
```

2. Red Hat OpenShift Dev Spaces Operator をインストールします。Installing from OperatorHub using the web console を参照してください。

3. 次のように、OpenShift で **openshift-devspaces** プロジェクトを作成します。

```
oc create namespace openshift-devspaces
```

4. OpenShift Web コンソールの **Administrator** ビューで、**Operators → Installed Operators → Red Hat OpenShift Dev Spaces instance Specification→ Create CheCluster → YAML view** に移動します。

5. **YAML view** で、**namespace: openshift-operators** を **namespace: openshift-devspaces** に置き換えます。

6. **Create** を選択します。Creating applications from installed Operators を参照してください。

検証

1. OpenShift Dev Spaces インスタンスが正しくインストールされていることを確認するには、**Operator detail** ページの **Dev Spaces Cluster** タブに移動します。**Red Hat OpenShift Dev Spaces インスタンス仕様** ページには、Red Hat OpenShift Dev Spaces インスタンスとそのステータスのリストが表示されます。

2. **devspaces CheCluster** をクリックして、**Details** タブに移動します。

3. 以下のフィールドの内容を参照してください。

   - **Message** フィールドにはエラーメッセージが含まれます。予想される内容は **None** です。

   - **Red Hat OpenShift Dev Spaces URL**フィールドには、Red Hat OpenShift Dev Spaces インスタンスの URL が含まれます。デプロイメントが正常に終了すると、URL が表示されます。

4. **Resources** タブに移動します。OpenShift Dev Spaces デプロイメントに割り当てられたリソースの一覧とそのステータスを表示します。

2.4. OPENSHIFT の制限された環境での OPENSHIFT DEV SPACE のインストール

制限されたネットワークで動作する OpenShift クラスターでは、パブリックリソースは利用できません。

ただし、OpenShift Dev Spaces をデプロイしてワークスペースを実行するには、以下のパブリックリソースが必要です。

- Operator カタログ
- コンテナーイメージ
- サンプルプロジェクト

これらのリソースを使用可能にするには、OpenShift クラスターからアクセス可能なレジストリー内のそれらのコピーに置き換えます。

前提条件

- OpenShift クラスターに、少なくとも 64 GB のディスクスペースがある。

- OpenShift クラスターは制限されたネットワーク上で動作する準備ができており、OpenShift コントロールプレーンはパブリックインターネットにアクセスできる。非接続インストールのミラーリングについて、および ネットワークが制限された環境での Operator Lifecycle Manager の使用 を参照してください。
- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the OpenShift CLI を参照してください。
- **registry.redhat.io** Red Hat エコシステムカタログへのアクティブな **oc** レジストリーセッション。Red Hat Container Registry authentication を参照してください。
- **opm**。Installing the **opm** CLI を参照してください。
- **jq**。Downloading **jq** を参照してください。
- **podman**。Installing Podman を参照してください。
- **<my_registry>** レジストリーへの管理アクセス権を持つアクティブな **skopeo** セッション。Installing Skopeo、Authenticating to a registry、および Mirroring images for a disconnected installation を参照してください。
- OpenShift Dev Spaces バージョン 3.2 の **dsc**。「dsc 管理ツールのインストール」を参照してください。

手順

1. ミラーリングスクリプトをダウンロードして実行し、カスタム Operator カタログをインストールして、関連するイメージをミラーリングします: prepare-restricted-environment.sh。

```
$ bash prepare-restricted-environment.sh \
  --ocp_ver "4.11" \
  --devworkspace_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.10" \
  --devworkspace_operator_version "v0.15.2" \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.10" \
  --prod_operator_package_name "devspaces-operator" \
  --prod_operator_version "v3.2.0" \
  --my_registry "<my_registry>" \
  --my_catalog "<my_catalog>"
```

2. 前の手順で で **che-operator-cr-patch.yaml** に指定した設定で OpenShift Dev Spaces をインストールします。

```
$ dsc server:deploy --platform=openshift \
  --che-operator-cr-patch-yaml=che-operator-cr-patch.yaml
```

3. OpenShift Dev Spaces 名前空間からユーザープロジェクト内のすべての Pod への受信トラフィックを許可します。「ネットワークポリシーの設定」 を参照してください。

関連情報

- Red Hat が提供する Operator カタログ
- カスタムカタログの管理

第3章 OPENSHIFT DEV SPACE の設定

このセクションでは、Red Hat OpenShift Dev Spaces の設定方法とオプションについて説明します。

3.1. CHECLUSTER カスタムリソースについて

OpenShift Dev Spaces のデフォルトデプロイメントは、Red Hat OpenShift Dev Spaces Operator で定義される **CheCluster** カスタムリソースパラメーターで設定されます。

CheCluster カスタムリソースは Kubernetes オブジェクトです。 **CheCluster** カスタムリソース YAML ファイルを編集して設定できます。このファイルには、各コンポーネントを設定するためのセクションが含まれています:
devWorkspace、**cheServer**、**pluginRegistry**、**devfileRegistry**、**database**、**dashboard** および **imagePuller**。

Red Hat OpenShift Dev Spaces Operator は、**CheCluster** カスタムリソースを OpenShift Dev Spaces インストールの各コンポーネントで使用できる設定マップに変換します。

OpenShift プラットフォームは、設定を各コンポーネントに適用し、必要な Pod を作成します。OpenShift がコンポーネントの設定で変更を検知すると、Pod を適宜再起動します。

> **例3.1 OpenShift Dev Spaces サーバーコンポーネントの主なプロパティーの設定**
>
> 1. **cheServer** コンポーネントセクションで適切な変更を加えた **CheCluster** カスタムリソース YAML ファイルを適用します。
> 2. Operator は、**che ConfigMap** を生成します。
> 3. OpenShift は **ConfigMap** の変更を検出し、OpenShift Dev Spaces Pod の再起動をトリガーします。

関連情報

- Operator について
- カスタムリソースについて

3.1.1. dsc を使用したインストール時に CheCluster カスタムリソースの設定

適切な設定で OpenShift Dev Space をデプロイするには、OpenShift Dev Space のインストール時に **CheCluster** カスタムリソース YAML ファイルを編集します。それ以外の場合は、OpenShift Dev Spaces デプロイメントは Operator で設定されたデフォルト設定パラメーターを使用します。

前提条件

- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法 を参照してください。
- **dsc**。「dsc 管理ツールのインストール」 を参照してください。

手順

- 設定する **CheCluster** カスタムリソースのサブセットを含む **che-operator-cr-patch.yaml** YAML ファイルを作成します。

```
spec:
  <component>:
    <property-to-configure>: <value>
```

- OpenShift Dev Spaces をデプロイし、**che-operator-cr-patch.yaml** ファイルで説明されている変更を適用します。

```
$ dsc server:deploy \
--che-operator-cr-patch-yaml=che-operator-cr-patch.yaml \
--platform <chosen-platform>
```

検証

1. 設定されたプロパティーの値を確認します。

```
$ oc get configmap che -o jsonpath='{.data.<configured-property>}' \
-n openshift-devspaces
```

関連情報

- 「**CheCluster** カスタムリソースフィールドの参照」
- 「OpenShift Dev Spaces サーバーコンポーネントの詳細な設定オプション」

3.1.2. CLI を使用して CheCluster カスタムリソースの設定

OpenShift Dev Spaces の実行中のインスタンスを設定するには、**CheCluster** カスタムリソース YAML ファイルを編集します。

前提条件

- OpenShift 上の OpenShift Dev Spaces のインスタンス。
- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。

手順

1. クラスター上の CheCluster カスタムリソースを編集します。

```
$ oc edit checluster/devspaces -n openshift-devspaces
```

2. ファイルを保存して閉じ、変更を適用します。

検証

1. 設定されたプロパティーの値を確認します。

```
$ oc get configmap che -o jsonpath='{.data.<configured-property>}' \
-n openshift-devspaces
```

関連情報

- 「**CheCluster** カスタムリソースフィールドの参照」
- 「OpenShift Dev Spaces サーバーコンポーネントの詳細な設定オプション」

3.1.3. CheCluster カスタムリソースフィールドの参照

このセクションでは、**CheCluster** カスタムリソースのカスタマイズに使用できるすべてのフィールドについて説明します。

- 例3.2「最小の **CheCluster** カスタムリソースの例。」
- 表3.1「開発環境の設定オプション。」
- 表3.4「OpenShift Dev Spaces コンポーネントの設定。」
- 表3.5「DevWorkspace Operator コンポーネントの設定。」
- 表3.6「OpenShift Dev Spaces サーバーコンポーネントに関連する一般的な設定。」
- 表3.7「OpenShift Dev Spaces インストールで使用されるプラグインレジストリーコンポーネントに関連する設定。」
- 表3.8「OpenShift Dev Spaces インストールで使用される Devfile レジストリーコンポーネントに関連する設定。」
- 表3.9「OpenShift Dev Spaces インストールで使用されるデータベースコンポーネントに関連する設定。」
- 表3.10「OpenShift Dev Spaces インストールで使用される Dashboard コンポーネントに関連する設定。」
- 表3.11「Kubernetes Image Puller コンポーネントの設定。」
- 表3.12「OpenShift Dev Spaces サーバーメトリクスコンポーネントの設定。」
- 表3.13「ネットワーク、OpenShift Dev Spaces 認証、および TLS 設定。」
- 表3.14「OpenShift Dev Spaces イメージを格納する代替レジストリーの設定。」
- 表3.15「**CheCluster** カスタムリソースの **status** が OpenShift Dev Spaces インストールの観察される状態を定義します。」

例3.2 最小の CheCluster カスタムリソースの例。

```
apiVersion: org.eclipse.che/v2
kind: CheCluster
metadata:
  name: devspaces
spec:
  devEnvironments:
    defaultNamespace:
      template: '<username>-che'
    storage:
      pvcStrategy: 'common'
```

```
components:
  database:
    externalDb: false
  metrics:
    enable: true
```

表3.1 開発環境の設定オプション。

プロパティー	説明
defaultComponents	DevWorkspaces に適用されるデフォルトコンポーネント。デフォルトコンポーネントは、Devfile にコンポーネントが含まれていない場合に使用します。
defaultEditor	一緒に作成するワークスペースのデフォルトエディター。プラグイン ID または URI を指定できます。プラグイン ID には **publisher/plugin/version** 形式が必要です。URI は **http://** または **https://** で始まる必要があります。
defaultNamespace	ユーザーのデフォルトの名前空間。
defaultPlugins	DevWorkspaces に適用されるデフォルトのプラグイン。
nodeSelector	ノードセレクターは、ワークスペース Pod を実行できるノードを制限します。
secondsOfInactivityBeforeIdling	ワークスペースのアイドルタイムアウト (秒単位)。このタイムアウトは、アクティビティーがない場合にワークスペースがアイドル状態になるまでの期間です。非アクティブによるワークスペースのアイドリングを無効にするには、この値を -1 に設定します。
secondsOfRunBeforeIdling	ワークスペースのタイムアウトを秒単位で実行します。このタイムアウトは、ワークスペースが実行される最大期間です。ワークスペースの実行タイムアウトを無効にするには、この値を -1 に設定します。
storage	ワークスペースの永続ストレージ。
tolerations	ワークスペース Pod の Pod 許容範囲によって、ワークスペース Pod を実行できる場所が制限されます。
trustedCerts	信頼できる証明書の設定。

表3.2 開発環境の defaultNamespace オプション。

プロパティー	説明
template	ユーザー namespace を事前に作成しない場合には、このフィールドは最初のワークスペースの起動時に作成される Kubernetes namespace を定義します。che-workspace-<username> など、**<username>** と **<userid>** のプレースホルダーを使用できます。

表3.3 開発環境のストレージ オプション。

プロパティー	説明
perUserStrategyPvcConfig	**per-user** PVC ストラテジーを使用する場合の PVC 設定。
perWorkspaceStrategyPvcConfig	**per-workspace** PVC ストラテジーを使用する場合の PVC 設定。
pvcStrategy	Che サーバーの永続ボリューム要求ストラテジー。サポートされているストラテジー: **per-user** (1 つのボリュームに全ワークスペースの PVC が入る) と 'per-workspace' (各ワークスペースに個別の PVC が与えられる) です。詳細は、https://github.com/eclipse/che/issues/21185 を参照してください。

表3.4 OpenShift Dev Spaces コンポーネントの設定。

プロパティー	説明
cheServer	Che サーバーに関連する一般的な設定。
dashboard	Che インストールで使用されるダッシュボードに関連する設定。
database	Che インストールで使用されるデータベースに関連する設定。
devWorkspace	DevWorkspace Operator の設定。
devfileRegistry	Che インストールで使用される devfile レジストリーに関連する設定。
imagePuller	Kubernetes Image Puller の設定。
metrics	Che サーバーメトリクスの設定。
pluginRegistry	Che インストールで使用されるプラグインレジストリーに関連する設定。

表3.5 DevWorkspace Operator コンポーネントの設定。

プロパティー	説明
deployment	デプロイメントオーバーライドオプション。
runningLimit	ユーザーごとの実行中のワークスペースの最大数。

表3.6 OpenShift Dev Spaces サーバーコンポーネントに関連する一般的な設定。

プロパティー	説明

プロパティー	説明
clusterRoles	Che ServiceAccount に割り当てられた ClusterRoles。デフォルトのロールは、**<che-namespace>-cheworkspaces-namespaces-clusterrole** - **<che-namespace>-cheworkspaces-clusterrole** - **<che-namespace>-cheworkspaces-devworkspace-clusterrole** で、ここでの <che-namespace> は CheCluster CRD が作成される名前空間です。各ロールには、**app.kubernetes.io/part-of=che.eclipse.org** ラベルが必要です。Che Operator は、付与するためにはこれらの ClusterRoles のすべてのパーミッションをすでに持っていなければなりません。
debug	Che サーバーのデバッグモードを有効にします。
deployment	デプロイメントオーバーライドオプション。
extraProperties	**CheCluster** カスタムリソース (CR) の他のフィールドからすでに生成されている値に加えて、Che サーバーによって使用される生成された **che** ConfigMap に適用される追加の環境変数のマップ。**extraProperties** フィールドに、他の CR フィールドから **che** で通常生成されるプロパティーが含まれる場合、**extraProperties** で定義された値が代わりに使用されます。
logLevel	Che サーバーのログレベル: **INFO** または **DEBUG**。
proxy	Kubernetes クラスターのプロキシーサーバー設定。OpenShift クラスターに追加の設定は必要ありません。これらの設定を OpenShift クラスターに指定することで、OpenShift プロキシー設定をオーバーライドします。

表3.7 OpenShift Dev Spaces インストールで使用されるプラグインレジストリーコンポーネントに関連する設定。

プロパティー	説明
deployment	デプロイメントオーバーライドオプション。
disableInternalRegistry	内部プラグインレジストリーを無効にします。
externalPluginRegistries	外部プラグインレジストリー。
openVSXURL	VSX レジストリー URL を開きます。省略した場合は、埋め込みインスタンスが使用されます。

表3.8 OpenShift Dev Spaces インストールで使用される Devfile レジストリーコンポーネントに関連する設定。

プロパティー	説明
deployment	デプロイメントオーバーライドオプション。

プロパティー	説明
disableInternalRegistry	内部 devfile レジストリーを無効にします。
externalDevfileRegistries	サンプルのすぐに使用できる devfile を提供する外部 devfile レジストリー。

表3.9 OpenShift Dev Spaces インストールで使用されるデータベースコンポーネントに関連する設定。

プロパティー	説明
credentialsSecretName	Che サーバーがデータベースへの接続に使用する PostgreSQL **user** および **password** が含まれるシークレット。シークレットには、**app.kubernetes.io/part-of=che.eclipse.org** ラベルが必要です。
deployment	デプロイメントオーバーライドオプション。
externalDb	Operator に対して専用のデータベースをデプロイするよう指示します。デフォルトでは、専用の PostgreSQL データベースは Che インストールの一部としてデプロイされます。**externalDb** が **true** に設定されている場合、Operator によって専用データベースはデプロイされず、使用する外部データベースに関する接続の詳細を提供する必要があります。
postgresDb	Che サーバーがデータベースへの接続に使用する PostgreSQL データベース名。
postgresHostName	Che サーバーが接続する PostgreSQL データベースのホスト名。外部データベースを使用する場合、この値のみを上書きします。**externalDb** フィールドを参照してください。
postgresPort	Che サーバーが接続する PostgreSQL データベースのポート。外部データベースを使用する場合、この値のみを上書きします。**externalDb** フィールドを参照してください。
pvc	PostgreSQL データベースの PVC 設定。

表3.10 OpenShift Dev Spaces インストールで使用される Dashboard コンポーネントに関連する設定。

プロパティー	説明
deployment	デプロイメントオーバーライドオプション。
headerMessage	ダッシュボードのヘッダーメッセージ。

表3.11 Kubernetes Image Puller コンポーネントの設定。

プロパティー	説明
enable	コミュニティーでサポートされている Kubernetes Image Puller Operator をインストールして設定します。仕様を指定せずに値を **true** に設定すると、Operator によって管理されるデフォルトの Kubernetes Image Puller オブジェクトが作成されます。値を **false** に設定すると、仕様が提供されているかどうかに関係なく、Kubernetes Image Puller オブジェクトが削除され、Operator がアンインストールされます。**spec.images** フィールドを空のままにしておくと、推奨されるワークスペース関連のイメージのセットが自動的に検出され、インストール後に事前にプルされます。この Operator とその動作はコミュニティーによってサポートされていますが、そのペイロードは商用サポートされているイメージをプルするために商用サポートされている場合があることに注意してください。
spec	CheCluster で Image Puller を設定するための Kubernetes Image Puller 仕様。

表3.12 OpenShift Dev Spaces サーバーメトリクスコンポーネントの設定。

プロパティー	説明
enable	Che サーバーエンドポイントの **metrics** を有効にします。

表3.13 ネットワーク、OpenShift Dev Spaces 認証、および TLS 設定。

プロパティー	説明
annotations	Ingress (OpenShift プラットフォームのルート) に設定されるアノテーションを定義します。kubernetes プラットフォームのデフォルト: kubernetes.io/ingress.class: \nginx\ nginx.ingress.kubernetes.io/proxy-read-timeout: \3600\, nginx.ingress.kubernetes.io/proxy-connect-timeout: \3600\, nginx.ingress.kubernetes.io/ssl-redirect: \true\
auth	認証設定
domain	OpenShift クラスターの場合、Operator はドメインを使用してルートのホスト名を生成します。生成されたホスト名は、che-<che-namespace>.<domain> のパターンに従います。<che-namespace> は、CheCluster CRD が作成される名前空間です。ラベルと組み合わせて、デフォルト以外の Ingress コントローラーによって提供されるルートを作成します。Kubernetes クラスターの場合、グローバル Ingress ドメインが含まれます。デフォルト値はありません。指定する必要があります。
hostname	インストールされた Che サーバーの公開ホスト名。
labels	Ingress (OpenShift プラットフォームのルート) に設定されるラベルを定義します。

プロパティー	説明
tlsSecretName	Ingress TLS ターミネーションのセットアップに使用されるシークレットの名前。フィールドが空の文字列の場合、デフォルトのクラスター証明書が使用されます。シークレットには、**app.kubernetes.io/part-of=che.eclipse.org** ラベルが必要です。

表3.14 OpenShift Dev Spaces イメージを格納する代替レジストリーの設定。

プロパティー	説明
hostname	イメージのプルに使用する別のコンテナーレジストリーの任意のホスト名または URL。この値は、Che デプロイメントに関連するすべてのデフォルトコンテナーイメージで定義されるコンテナーレジストリーのホスト名を上書きします。これは、Che を制限された環境にインストールする場合に特に便利です。
organization	イメージのプルに使用する別のレジストリーのオプションのリポジトリー名。この値は、Che デプロイメントに関連するすべてのデフォルトコンテナーイメージで定義されるコンテナーレジストリーの組織を上書きします。これは、制限された環境で OpenShift Dev Spaces をインストールする場合にとくに役立ちます。

表3.15 CheCluster カスタムリソースの status が OpenShift Dev Spaces インストールの観察される状態を定義します。

プロパティー	説明
chePhase	Che デプロイメントの現在のフェーズを指定します。
cheURL	Che サーバーの公開 URL。
cheVersion	現在インストールされている Che バージョン。
devfileRegistryURL	内部 devfile レジストリーの公開 URL。
gatewayPhase	ゲートウェイデプロイメントの現在のフェーズを指定します。
message	Che デプロイメントが現在のフェーズにある理由の詳細を示す、人間が判読できるメッセージ。
pluginRegistryURL	内部プラグインレジストリーの公開 URL。
postgresVersion	使用中のイメージの PostgreSQL バージョン。
reason	Che デプロイメントが現在のフェーズにある理由の詳細を示す簡単な CamelCase メッセージ。

プロパティー	説明
workspaceBaseDomain	解決されたワークスペースベースドメイン。これは、仕様で明示的に定義された同じ名前のプロパティーのコピーか、仕様で定義されておらず、OpenShift で実行している場合は、ルートの自動的に解決されたベースドメインです。

3.2. ユーザープロジェクトプロビジョニングの設定

OpenShift Dev Spaces は、ユーザーごとに、プロジェクト内のワークスペースを分離します。OpenShift Dev Spaces は、ラベルとアノテーションの存在によってユーザープロジェクトを識別します。ワークスペースを起動する際に必要なプロジェクトが存在しない場合、OpenShift Dev Spaces はテンプレート名を使用してプロジェクトを作成します。

OpenShift Dev Spaces の動作は、次の方法で変更できます。

- 「自動プロビジョニング用のユーザープロジェクト名の設定」
- 「プロジェクトの事前プロビジョニング」

3.2.1. 自動プロビジョニング用のユーザープロジェクト名の設定

OpenShift Dev Spaces がワークスペース起動時に必要なプロジェクトを作成するために使用するプロジェクト名テンプレートを設定できます。

有効なプロジェクト名テンプレートは、次の規則に従います。

- **<username>** または **<userid>** プレースホルダーは必須です。
- ユーザー名と ID に無効な文字を含めることはできません。ユーザー名または ID のフォーマットが OpenShift オブジェクトの命名規則と互換性がない場合、OpenShift Dev Spaces は、互換性のない文字を **-** 記号に置き換えてユーザー名や ID を有効な名前に変更します。
- OpenShift Dev Spaces は、**<userid>** プレースホルダーを 14 文字の文字列と判断し、ID が衝突しないようにランダムな 6 文字の接尾辞を追加します。結果は、再利用のためにユーザー設定に保存されます。
- Kubernetes は、プロジェクト名の長さを 63 文字に制限しています。
- OpenShift は、長さをさらに 49 文字に制限しています。

手順

- **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    devEnvironments:
      defaultNamespace:
        template: <workspace_namespace_template_>
```

例3.3 ユーザーワークスペースプロジェクト名テンプレートの例

ユーザーワークスペースプロジェクト名テンプレート	結果のプロジェクト例
<username>-devspaces (デフォルト)	user1-devspaces
<userid>-namespace	**cge1egvsb2nhba-namespace-ul1411**
<userid>-aka-<username>-namespace	**cgezegvsb2nhba-aka-user1-namespace-6m2w2b**

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.2.2. プロジェクトの事前プロビジョニング

自動プロビジョニングに依存するのではなく、ワークスペースプロジェクトを事前にプロビジョニングできます。ユーザーごとに手順を繰り返します。

手順

- 次のラベルとアノテーションを使用して、<username> ユーザーの <project_name> プロジェクトを作成します。

```
kind: Namespace
apiVersion: v1
metadata:
  name: <project_name> (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspaces-namespace
  annotations:
    che.eclipse.org/username: <username>
```

(1) 選択したプロジェクト名を使用します。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.3. サーバーコンポーネントの設定

- 「シークレットまたは ConfigMap をファイルまたは環境変数として OpenShift Dev Spaces コンテナーにマウントする」

3.3.1. シークレットまたは ConfigMap をファイルまたは環境変数として OpenShift Dev Spaces コンテナーにマウントする

シークレットは、以下のような機密データを格納する OpenShift オブジェクトです。

- ユーザー名
- パスワード
- 認証トークン

(暗号化された形式)。

ユーザーは、機密データまたは OpenShift Dev Spaces で管理されるコンテナーの設定が含まれる ConfigMap を以下のようにマウントできます。

- ファイル
- 環境変数

マウントプロセスでは、標準の OpenShift マウントメカニズムを使用しますが、追加のアノテーションとラベル付けが必要です。

3.3.1.1. シークレットまたは ConfigMap を OpenShift Dev Spaces コンテナーにファイルとしてマウントする

前提条件

- Red Hat OpenShift Dev Spaces の実行中のインスタンス

手順

1. OpenShift Dev Spaces がデプロイされている OpenShift プロジェクトに新しい OpenShift シークレットまたは ConfigMap を作成します。作成される予定のオブジェクトのラベルは、ラベルのセットと一致する必要があります。
   - **app.kubernetes.io/part-of: che.eclipse.org**
   - **app.kubernetes.io/component: <DEPLOYMENT_NAME>-<OBJECT_KIND>**
   - **<DEPLOYMENT_NAME>** には、以下のデプロイメントのいずれかを使用します。
     - **postgres**
     - **keycloak**
     - **devfile-registry**
     - **plugin-registry**
     - **devspaces**
       および
   - **<jasper_KIND>** は以下のいずれかになります。
     - **secret**
       または

- **configmap**

例3.4 以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
...
```

または

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

アノテーションは、指定されるオブジェクトがファイルとしてマウントされていることを示す必要があります。

1. アノテーション値を設定します。

   - **che.eclipse.org/mount-as: file** - オブジェクトをファイルとしてマウントするように指定します。
   - **che.eclipse.org/mount-path: <TARGET_PATH>** - 必要なマウントパスを指定します。

例3.5 以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-data
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
  labels:
...
```

または

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-data
  annotations:
```

```
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
  labels:
...
```

OpenShift オブジェクトには複数の項目が含まれる可能性があり、その名前はコンテナーにマウントされる必要なファイル名と一致する必要があります。

例3.6 以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <base64 encoded data content here>
```

または

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-data
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
  annotations:
    che.eclipse.org/mount-as: file
    che.eclipse.org/mount-path: /data
data:
  ca.crt: <data content here>
```

これにより、**ca.crt** という名前のファイルが OpenShift Dev Spaces コンテナーの **/data** パスにマウントされます。

重要

OpenShift Dev Spaces コンテナーに変更を加えるには、オブジェクトを完全に再作成します。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.3.1.2. シークレットまたは ConfigMap を環境変数として OpenShift Dev Spaces コンテナーにマウントする

前提条件

- Red Hat OpenShift Dev Spaces の実行中のインスタンス

手順

1. OpenShift Dev Spaces がデプロイされている OpenShift プロジェクトに新しい OpenShift シークレットまたは ConfigMap を作成します。作成される予定のオブジェクトのラベルは、ラベルのセットと一致する必要があります。
   - **app.kubernetes.io/part-of: che.eclipse.org**
   - **app.kubernetes.io/component: <DEPLOYMENT_NAME>-<OBJECT_KIND>**
   - **<DEPLOYMENT_NAME>** には、以下のデプロイメントのいずれかを使用します。
     - **postgres**
     - **keycloak**
     - **devfile-registry**
     - **plugin-registry**
     - **devspaces**
       および
   - **<jasper_KIND>** は以下のいずれかになります。
     - **secret**
       または
     - **configmap**

例3.7 以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-secret
...
```

または

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  labels:
```

```
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: devspaces-configmap
...
```

アノテーションは、指定されるオブジェクトが環境変数としてマウントされていることを示す必要があります。

1. アノテーション値を設定します。
   - **che.eclipse.org/mount-as: env** –: オブジェクトを環境変数としてマウントするように指定します。
   - **che.eclipse.org/env-name: <FOOO_ENV>**: オブジェクトキー値のマウントに必要な環境変数名を指定します。

例3.8 以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
  labels:
    ...
data:
  mykey: myvalue
```

または

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
  labels:
    ...
data:
  mykey: myvalue
```

これにより、2 つの環境変数が

- **FOO_ENV**
- **myvalue**

OpenShift Dev Spaces コンテナーにプロビジョニングされている。

オブジェクトに複数のデータ項目がある場合、環境変数の名前は以下のようにそれぞれのデータキーについて指定される必要があります。

例3.9 以下に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/mount-as: env
    che.eclipse.org/mykey_env-name: FOO_ENV
    che.eclipse.org/otherkey_env-name: OTHER_ENV
  labels:
    ...
data:
  mykey: __<base64 encoded data content here>__
  otherkey: __<base64 encoded data content here>__
```

または

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: custom-settings
  annotations:
    che.eclipse.org/mount-as: env
    che.eclipse.org/mykey_env-name: FOO_ENV
    che.eclipse.org/otherkey_env-name: OTHER_ENV
  labels:
    ...
data:
  mykey: __<data content here>__
  otherkey: __<data content here>__
```

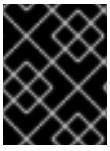
これにより、2 つの環境変数が

- **FOO_ENV**
- **OTHER_ENV**

OpenShift Dev Spaces コンテナーにプロビジョニングされている。

注記

OpenShift シークレットのアノテーション名の最大長さは 63 文字です。ここで、9 文字は、/ で終わる接頭辞用に予約されます。これは、オブジェクトに使用できるキーの最大長さの制限として機能します。



重要

OpenShift Dev Spaces コンテナーに変更を加えるには、オブジェクトを完全に再作成します。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.3.2. OpenShift Dev Spaces サーバーコンポーネントの詳細な設定オプション

以下のセクションでは、OpenShift Dev Spaces サーバーコンポーネントの詳細なデプロイメントおよび設定方法を説明します。

3.3.2.1. OpenShift Dev Spaces サーバーの詳細設定について

以下のセクションでは、OpenShift Dev Spaces サーバーコンポーネントの詳細設定方法について説明します。

詳細設定は以下を実行するために必要です。

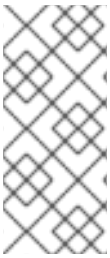
- 標準の **CheCluster** カスタムリソースフィールドから Operator によって自動的に生成されない環境変数を追加します。
- 標準の **CheCluster** カスタムリソースフィールドから Operator によって自動的に生成されるプロパティーを上書きします。

CheCluster Custom Resource **server** 設定の一部である **customCheProperties** フィールドには、OpenShift Dev Spaces サーバーコンポーネントに適用する追加の環境変数のマップが含まれます。

例3.10 ワークスペースのデフォルトのメモリー制限の上書き

- **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_WORKSPACE_DEFAULT__MEMORY__LIMIT__MB: "2048"
```



注記

OpenShift Dev Spaces Operator の以前のバージョンには、このロールを果たすために **custom** という名前の ConfigMap がありました。OpenShift Dev Spaces オペレーターが **custom** という名前の **configMap** を見つけると、それに含まれるデータを **customCheProperties** フィールドに追加し、OpenShift Dev Spaces を再デプロイして、**カスタムconfigMap** を削除します。

関連情報

- 「**CheCluster** カスタムリソースフィールドの参照」
- 「OpenShift Dev Spaces サーバーコンポーネントのシステムプロパティーの参照」

3.3.2.2. OpenShift Dev Spaces サーバーコンポーネントのシステムプロパティーの参照

以下のドキュメントでは、OpenShift Dev Spaces サーバーコンポーネントの可能な設定プロパティーをすべて説明します。

3.3.2.2.1. OpenShift Dev Spaces サーバー

3.3.2.2.1.1. CHE_API

API サービス。ブラウザーは、この URL を使用して OpenShift Dev Spaces サーバーへの REST 通信を開始します。

デフォルト

http://${CHE_HOST}:${CHE_PORT}/api

3.3.2.2.1.2. CHE_API_INTERNAL

API サービスの内部ネットワーク URL。バックエンドサービスは、この URL を使用して OpenShift Dev Spaces サーバーへの REST 通信を開始する必要があります。

デフォルト

NULL

3.3.2.2.1.3. CHE_WEBSOCKET_ENDPOINT

OpenShift Dev Spaces WebSocket の主なエンドポイント。主な websocket の対話とメッセージング用の基本的な通信エンドポイントを提供します。

デフォルト

ws://${CHE_HOST}:${CHE_PORT}/api/websocket

3.3.2.2.1.4. CHE_WEBSOCKET_INTERNAL_ENDPOINT

OpenShift Dev Spaces WebSocket の主な内部エンドポイント。主な websocket の対話とメッセージング用の基本的な通信エンドポイントを提供します。

デフォルト

NULL

3.3.2.2.1.5. CHE_WORKSPACE_PROJECTS_STORAGE

プロジェクトは、OpenShift Dev Spaces サーバーから、各ワークスペースを実行するマシンに同期されます。これは、プロジェクトが配置されているマシンのディレクトリーです。

デフォルト

/projects

3.3.2.2.1.6. CHE_WORKSPACE_LOGS_ROOT__DIR

すべてのワークスペースログが置かれるマシン内のディレクトリーを定義します。環境変数などの値として、この値をマシンに指定します。これは、エージェントの開発者がこのディレクトリーを使用してエージェントのログをバックアップできるようにするためのものです。

デフォルト

/workspace_logs

3.3.2.2.1.7. CHE_WORKSPACE_HTTP__PROXY

環境変数 HTTP_PROXY は、ワークスペースを起動するコンテナーで指定された値に設定します。

デフォルト

空

3.3.2.2.1.8. CHE_WORKSPACE_HTTPS__PROXY

環境変数 HTTPS_PROXY は、ワークスペースを起動するコンテナーで指定された値に設定します。

デフォルト

空

3.3.2.2.1.9. CHE_WORKSPACE_NO__PROXY

環境変数 NO_PROXY は、ワークスペースを起動するコンテナーで指定された値に設定します。

デフォルト

空

3.3.2.2.1.10. CHE_WORKSPACE_AUTO__START

デフォルトでは、ユーザーがこの URL を使用してワークスペースにアクセスすると、ワークスペースは自動的に起動します (現時点で停止している場合)。この動作を無効にするには、このパラメーターを **false** に設定します。

デフォルト

true

3.3.2.2.1.11. CHE_WORKSPACE_POOL_TYPE

ワークスペーススレッドプールの設定。このプールは、非同期の実行が必要なワークスペース関連の操作 (例: 起動/停止) に使用されます。設定可能な値は **fixed** および **cached** です。

デフォルト

固定:

3.3.2.2.1.12. CHE_WORKSPACE_POOL_EXACT__SIZE

プールタイプが **fixed** と異なる場合に、このプロパティーは無視されます。これはプールのサイズを設定します。設定されると、**multiplier** プロパティーは無視されます。このプロパティーが設定されていない場合 (**0**, **<0**, **NULL**)、プールサイズはコア数と等しくなります。**che.workspace.pool.cores_multiplier** も参照してください。

デフォルト

30

3.3.2.2.1.13. CHE_WORKSPACE_POOL_CORES__MULTIPLIER

プールタイプが **fixed** に設定されておらず、**che.workspace.pool.exact_size** が設定されている場合は、このプロパティーは無視されます。設定されている場合、プールサイズは **N_CORES * multiplier** になります。

デフォルト

2

3.3.2.2.1.14. CHE_WORKSPACE_PROBE__POOL__SIZE

このプロパティーは、ワークスペースサーバーの liveness プローブに使用するスレッドの数を指定します。

デフォルト

10

3.3.2.2.1.15. CHE_WORKSPACE_HTTP__PROXY__JAVA__OPTIONS

ワークスペース JVM の HTTP プロキシー設定。

デフォルト

NULL

3.3.2.2.1.16. CHE_WORKSPACE_JAVA__OPTIONS

ワークスペースで実行されている JVM に追加される Java コマンドラインオプション。

デフォルト

-XX:MaxRAM=150m-XX:MaxRAMFraction=2 -XX:+UseParallelGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true -Xms20m -Djava.security.egd=file:/dev/./urandom

3.3.2.2.1.17. CHE_WORKSPACE_MAVEN__OPTIONS

ワークスペースでエージェントを実行する JVM に追加される Maven コマンドラインオプション。

デフォルト

-XX:MaxRAM=150m-XX:MaxRAMFraction=2 -XX:+UseParallelGC -XX:MinHeapFreeRatio=10 -XX:MaxHeapFreeRatio=20 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true -Xms20m -Djava.security.egd=file:/dev/./urandom

3.3.2.2.1.18. CHE_WORKSPACE_DEFAULT__MEMORY__LIMIT__MB

環境に RAM 設定のない各マシンの RAM 制限のデフォルト。0 以下の値値は、制限を無効にするものとして解釈されます。

デフォルト

1024

3.3.2.2.1.19. CHE_WORKSPACE_DEFAULT__MEMORY__REQUEST__MB

環境内に明示的な RAM 設定のない各コンテナーの RAM 要求。この量はワークスペースコンテナーの作成時に割り当てられます。このプロパティーは、すべてのインフラストラクチャー実装でサポートされる訳ではありません。現時点で、これは OpenShift によってサポートされます。メモリー制限を超えるメモリー要求は無視され、制限サイズのみが使用されます。0 以下の値値は、制限を無効にするものとして解釈されます。

デフォルト

200

3.3.2.2.1.20. CHE_WORKSPACE_DEFAULT__CPU__LIMIT__CORES

環境に CPU 設定のない各コンテナーの CPU 制限。浮動小数点のコア数 (例: **0.125**) で、または Kubernetes 形式 (**125m** などの整数のミリコア数) を使用して指定します。0 以下の値値は、制限を無効にするものとして解釈されます。

デフォルト

-1

3.3.2.2.1.21. CHE_WORKSPACE_DEFAULT__CPU__REQUEST__CORES

環境内に CPU 設定のない各コンテナーの CPU 要求。CPU 制限を超える CPU 要求は無視され、制限の数値のみが使用されます。0 以下の値値は、制限を無効にするものとして解釈されます。

デフォルト

-1

3.3.2.2.1.22. CHE_WORKSPACE_SIDECAR_DEFAULT__MEMORY__LIMIT__MB

OpenShift Dev Spaces プラグイン設定に RAM 設定のない各サイドカーの RAM 制限。0 以下の値値は、制限を無効にするものとして解釈されます。

デフォルト

128

3.3.2.2.1.23. CHE_WORKSPACE_SIDECAR_DEFAULT__MEMORY__REQUEST__MB

OpenShift Dev Spaces プラグイン設定に RAM 設定のない各サイドカーの RAM 要求。

デフォルト

64

3.3.2.2.1.24. CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__LIMIT__CORES

OpenShift Dev Spaces プラグイン設定に CPU 設定のない各サイドカーの CPU 制限のデフォルト。浮動小数点のコア数 (例: **0.125**) で、または Kubernetes 形式 (**125m** などの整数のミリコア数) を使用して指定します。0 以下の値値は、制限を無効にするものとして解釈されます。

デフォルト

-1

3.3.2.2.1.25. CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__REQUEST__CORES

OpenShift Dev Spaces プラグイン設定に CPU 設定のない各サイドカーの CPU 要求のデフォルト。浮動小数点のコア数 (例: **0.125**) で、または Kubernetes 形式 (**125m** などの整数のミリコア数) を使用して指定します。

デフォルト

-1

3.3.2.2.1.26. CHE_WORKSPACE_SIDECAR_IMAGE__PULL__POLICY

サイドカーのイメージプルストラテジーを定義します。使用できる値は **Always**、**Never**、**IfNotPresent** です。その他の値については、**Always** は **:latest** タグが付いたイメージに、その他の場合は **IfNotPresent** が想定されます。

デフォルト

Always

3.3.2.2.1.27. CHE_WORKSPACE_ACTIVITY__CHECK__SCHEDULER__PERIOD__S

非アクティブなワークスペースの一時停止ジョブの実行期間。

デフォルト

60

3.3.2.2.1.28. CHE_WORKSPACE_ACTIVITY__CLEANUP__SCHEDULER__PERIOD__S

アクティビティーテーブルのクリーンアップ期間。アクティビティーテーブルには、サーバーが特定の時点で障害が発生するなどの予想されないエラーが生じる場合に、無効なデータまたは古いデータを含まれることがあります。デフォルトでは、クリーンアップジョブは 1 時間ごとに実行されます。

デフォルト

3600

3.3.2.2.1.29. CHE_WORKSPACE_ACTIVITY__CLEANUP__SCHEDULER__INITIAL__DELAY__S

サーバーの起動後から最初のアクティビティーのクリーンアップジョブを開始するまでの遅延。

デフォルト

60

3.3.2.2.1.30. CHE_WORKSPACE_ACTIVITY__CHECK__SCHEDULER__DELAY__S

OpenShift Dev Spaces サーバーが非アクティブタイムアウトに近い期間利用できない場合に、マスカレードチェックジョブが開始されるまでの遅延。

デフォルト

180

3.3.2.2.1.31. CHE_WORKSPACE_CLEANUP__TEMPORARY__INITIAL__DELAY__MIN

一時ワークスペースのクリーンアップジョブの最初の実行を遅延させる時間。

デフォルト

5

3.3.2.2.1.32. CHE_WORKSPACE_CLEANUP__TEMPORARY__PERIOD__MIN

実行を終了してから次の一時的なワークスペースのクリーンアップジョブの実行を開始するまでの間に遅延する時間

デフォルト

180

3.3.2.2.1.33. CHE_WORKSPACE_SERVER_PING__SUCCESS__THRESHOLD

サーバーへの正常に順次実行される ping の数。この数を超えると、サーバーは利用可能な状態にあるものとして処理されます。OpenShift Dev Sspaces Operator: このプロパティーは、ワークスペース、エージェント、ターミナル、exec などの全サーバーに共通します。

デフォルト

1

3.3.2.2.1.34. CHE_WORKSPACE_SERVER_PING__INTERVAL__MILLISECONDS

ワークスペースサーバーへの連続する ping の間隔 (ミリ秒単位)。

デフォルト

3000

3.3.2.2.1.35. CHE_WORKSPACE_SERVER_LIVENESS__PROBES

liveness プローブを必要とするサーバー名の一覧

デフォルト

wsagent/http,exec-agent/http,terminal,theia,jupyter,dirigible,cloud-shell,intellij

3.3.2.2.1.36. CHE_WORKSPACE_STARTUP__DEBUG__LOG__LIMIT__BYTES

ワークスペースの起動をデバッグする際に che-server で観察される単一コンテナーから収集されるログの制限サイズ。デフォルト値は 10MB=10485760 です。

デフォルト

10485760

3.3.2.2.1.37. CHE_WORKSPACE_STOP_ROLE_ENABLED

true の場合、OpenShift OAuth が有効な場合に、編集権限を持つ stop-workspace ロールが che ServiceAccount に付与されます。この設定は、OpenShift OAuth が有効な場合にワークスペースのアイドリングに主に必要になります。

デフォルト

true

3.3.2.2.1.38. CHE_DEVWORKSPACES_ENABLED

DevWorkspaces を有効にして OpenShift Dev Spaces をデプロイするかどうかを指定します。このプロパティーは、DevWorkspaces のサポートもインストールされている場合、OpenShift DevSpacesOperator によって設定されます。このプロパティーは、このファクトを OpenShift Dev Spaces ダッシュボードにアドバタイズするために使用されます。このプロパティーの値を手動で変更することは推奨されません。

デフォルト

false

3.3.2.2.2. 認証パラメーター

3.3.2.2.2.1. CHE_AUTH_USER__SELF__CREATION

OpenShift Dev Spaces には単一の ID 実装があるため、これによってユーザーエクスペリエンスが変わることはありません。true の場合、API レベルでのユーザー作成を有効にします。

デフォルト

false

3.3.2.2.2.2. CHE_AUTH_ACCESS__DENIED__ERROR__PAGE

認証エラーページアドレス

デフォルト

/error-oauth

3.3.2.2.2.3. CHE_AUTH_RESERVED__USER__NAMES

予約済みのユーザー名

デフォルト

空

3.3.2.2.2.4. CHE_OAUTH2_GITHUB_CLIENTID__FILEPATH

GitHub OAuth2 クライアントの設定。パーソナルアクセストークンの取得に使用されます。GitHub クライアント ID を持つファイルの場所。

デフォルト

NULL

3.3.2.2.2.5. CHE_OAUTH2_GITHUB_CLIENTSECRET__FILEPATH

GitHub クライアントシークレットを含むファイルの場所。

デフォルト

NULL

3.3.2.2.2.6. CHE_OAUTH_GITHUB_AUTHURI

GitHub OAuth 認証 URI。

デフォルト

https://github.com/login/oauth/authorize

3.3.2.2.2.7. CHE_OAUTH_GITHUB_TOKENURI

GitHub OAuth トークン URI。

デフォルト

https://github.com/login/oauth/access_token

3.3.2.2.2.8. CHE_INTEGRATION_GITHUB_OAUTH__ENDPOINT

GitHub サーバーアドレス。前提条件： OAuth 2 統合が GitHub サーバーに設定される。

デフォルト

NULL

3.3.2.2.2.9. CHE_INTEGRATION_GITHUB_DISABLE__SUBDOMAIN__ISOLATION

GitHub サーバーは、サブドメイン分離フラグを無効にします。

デフォルト

false

3.3.2.2.2.10. CHE_OAUTH_GITHUB_REDIRECTURIS

GitHub OAuth リダイレクト URI。複数の値をコンマで区切ります (例: URI,URI,URI)。

デフォルト

http://localhost:${CHE_PORT}/api/oauth/callback

3.3.2.2.2.11. CHE_OAUTH_OPENSHIFT_CLIENTID

OpenShift OAuth クライアントの設定。OpenShift OAuth トークンの取得に使用されます。OpenShift OAuth クライアント ID。

デフォルト

NULL

3.3.2.2.2.12. CHE_OAUTH_OPENSHIFT_CLIENTSECRET

OpenShift OAuth クライアントの設定。OpenShift OAuth トークンの取得に使用されます。OpenShift OAuth クライアント ID。OpenShift OAuth クライアントシークレット。

デフォルト

NULL

3.3.2.2.2.13. CHE_OAUTH_OPENSHIFT_OAUTH__ENDPOINT

Configurationof OpenShift OAuth クライアント。OpenShift OAuth トークンの取得に使用されます。OpenShift OAuth クライアント ID。OpenShift OAuth クライアントシークレット。OpenShift OAuth エンドポイント。

デフォルト

NULL

3.3.2.2.2.14. CHE_OAUTH_OPENSHIFT_VERIFY__TOKEN__URL

ConfigurationofOpenShiftOAuth クライアント。OpenShift OAuth トークンの取得に使用されます。OpenShift OAuth クライアント ID。OpenShift OAuth クライアントシークレット。OpenShift OAuth エンドポイント。OpenShift OAuth 検証トークン URL。

デフォルト

NULL

3.3.2.2.2.15. CHE_OAUTH1_BITBUCKET_CONSUMERKEYPATH

Bitbucket Server OAuth1 クライアントの設定。パーソナルアクセストークンの取得に使用されます。Bitbucket Server アプリケーションのコンシューマーキーが含まれるファイルの場所 (ユーザー名と同等)。

デフォルト

NULL

3.3.2.2.2.16. CHE_OAUTH1_BITBUCKET_PRIVATEKEYPATH

Bitbucket Server OAuth1 クライアントの設定パーソナルアクセストークンの取得に使用されます。Bitbucket Server アプリケーションのコンシューマーキーが含まれるファイルの場所 (ユーザー名と同等)。Bitbucket Server アプリケーションの秘密鍵が含まれるファイルの場所

デフォルト

NULL

3.3.2.2.2.17. CHE_OAUTH1_BITBUCKET_ENDPOINT

Bitbucket Server OAuth1 クライアントの設定パーソナルアクセストークンの取得に使用されます。Bitbucket Server アプリケーションのコンシューマーキーが含まれるファイルの場所 (ユーザー名と同等)。Bitbucket Server アプリケーションの秘密鍵の Bitbucket Server URL が含まれるファイルの場所ファクトリーと正しく連携するには、同じ URL を **che.integration.bitbucket.server_endpoints** に含める必要があります。

デフォルト

NULL

3.3.2.2.2.18. CHE_OAUTH2_BITBUCKET_CLIENTID__FILEPATH

Bitbucket OAuth2 クライアントの設定。パーソナルアクセストークンの取得に使用されます。Bitbucket クライアント ID を含むファイルの場所。

デフォルト

NULL

3.3.2.2.2.19. CHE_OAUTH2_BITBUCKET_CLIENTSECRET__FILEPATH

Bitbucket クライアントシークレットを含むファイルの場所。

デフォルト

NULL

3.3.2.2.2.20. CHE_OAUTH_BITBUCKET_AUTHURI

Bitbucket OAuth 承認 URI。

デフォルト

https://bitbucket.org/site/oauth2/authorize

3.3.2.2.2.21. CHE_OAUTH_BITBUCKET_TOKENURI

Bitbucket OAuth トークン URI。

デフォルト

https://bitbucket.org/site/oauth2/access_token

3.3.2.2.2.22. CHE_OAUTH_BITBUCKET_REDIRECTURIS

Bitbucket OAuth リダイレクト URI。複数の値をコンマで区切ります (例: URI,URI,URI)。

デフォルト

http://localhost:${CHE_PORT}/api/oauth/callback

3.3.2.2.3. 内部

3.3.2.2.3.1. SCHEDULE_CORE__POOL__SIZE

OpenShift Dev Spaces 拡張は、時間に基づいて実行をスケジュールできます。これにより、繰り返されるスケジュールで起動する拡張に割り当てられるスレッドプールのサイズが設定されます。

デフォルト

10

3.3.2.2.3.2. DB_SCHEMA_FLYWAY_BASELINE_ENABLED

DB の初期化および移行設定。true の場合には、baseline.version で設定 Si たバージョンのスクリプトを無視します。

デフォルト

true

3.3.2.2.3.3. DB_SCHEMA_FLYWAY_BASELINE_VERSION

これまでのバージョンを含むスクリプトは無視されます。ベースラインバージョンと同じバージョンのスクリプトも無視されることに注意してください。

デフォルト

5.0.0.8.1

3.3.2.2.3.4. DB_SCHEMA_FLYWAY_SCRIPTS_PREFIX

移行スクリプトの接頭辞

デフォルト

空

3.3.2.2.3.5. DB_SCHEMA_FLYWAY_SCRIPTS_SUFFIX

移行スクリプトの接尾辞。

デフォルト

.sql

3.3.2.2.3.6. DB_SCHEMA_FLYWAY_SCRIPTS_VERSION__SEPARATOR

スクリプト名を他の部分からバージョンを区切るための区切り文字。

デフォルト

—

3.3.2.2.3.7. DB_SCHEMA_FLYWAY_SCRIPTS_LOCATIONS

移行スクリプトを検索する場所。

デフォルト

classpath:che-schema

3.3.2.2.4. Kubernetes インフラパラメーター

3.3.2.2.4.1. CHE_INFRA_KUBERNETES_MASTER__URL

インフラが使用する Kubernetes クライアントマスター URL の設定。

デフォルト

空

3.3.2.2.4.2. CHE_INFRA_KUBERNETES_TRUST__CERTS

Kubernetes クライアントが信頼済みの証明書を使用するように設定するブール値。

デフォルト

false

3.3.2.2.4.3. CHE_INFRA_KUBERNETES_CLUSTER__DOMAIN

Kubernetes クラスタードメイン。設定されていない場合は、svc 名にはクラスタードメインに関する情報が含まれません。

デフォルト

NULL

3.3.2.2.4.4. CHE_INFRA_KUBERNETES_SERVER__STRATEGY

サーバーが Kubernetes インフラでグローバルに公開される方法を定義します。OpenShift Dev Spaces に実装されている戦略のリスト: **default-host**、**multi-host**、**single-host**。

デフォルト

multi-host

3.3.2.2.4.5. CHE_INFRA_KUBERNETES_SINGLEHOST_WORKSPACE_EXPOSURE

ワークスペースのプラグインとエディターを単一ホストモードで公開する方法を定義します。サポートされているエクスポージャー: **native** :KubernetesIngresses を使用してサーバーをエクスポージャーします。Kubernetes でのみ機能します。**gateway**: reverse-proxy ゲートウェイを使用してサーバーを公開します。

デフォルト

native

3.3.2.2.4.6. CHE_INFRA_KUBERNETES_SINGLEHOST_WORKSPACE_DEVFILE__ENDPOINT__EXPOSURE

single-host サーバーストラテジーで devfile エンドポイント、エンドユーザーのアプリケーションを公開する方法を定義します。これらは single-host ストラテジーに従い、サブパスで公開されるか、またはサブドメイン上で公開できます。**Multi-host**: サブドメインで公開されます。**single-host**: サブパスに公開されます。

デフォルト

multi-host

3.3.2.2.4.7. CHE_INFRA_KUBERNETES_SINGLEHOST_GATEWAY_CONFIGMAP__LABELS

single-host ゲートウェイを設定する ConfigMap に設定されるラベルを定義します。

デフォルト

app=che,component=che-gateway-config

3.3.2.2.4.8. CHE_INFRA_KUBERNETES_INGRESS_DOMAIN

che.infra.kubernetes.server_strategy プロパティーが **multi-host** に設定されている場合に、ワークスペースでサーバーのドメインを生成するために使用されます。

デフォルト

空

3.3.2.2.4.9. CHE_INFRA_KUBERNETES_NAMESPACE_CREATION__ALLOWED

OpenShift Dev Spaces サーバーがユーザーワークスペース用のプロジェクトの作成を許可されているか、クラスター管理者が手動で作成することを目的としているかを示します。このプロパティーは OpenShift infra によっても使用されます。

デフォルト

true

3.3.2.2.4.10. CHE_INFRA_KUBERNETES_NAMESPACE_DEFAULT

ユーザーがオーバーライドしない場合にユーザーのワークスペースが作成される Kubernetes のデフォルトの名前空間を定義します。**<username>** および **<userid>** プレースホルダー (例: **che-workspace-<username>**) を使用できます。この場合、ユーザーごとに新規 namespace が作成されます。OpenShift インフラでプロジェクトの指定にも使用されます。**<username>** または **<userid>** プレースホルダーは必須です。

デフォルト

<username>-che

3.3.2.2.4.11. CHE_INFRA_KUBERNETES_NAMESPACE_LABEL

che-server がワークスペース namespace にラベルを付けるかどうかを定義します。注: このプロパティーの値を true に設定しておくことを強くお勧めします。false の場合、新しいワークスペース

namespace は自動的にラベル付けされないため、OpenShift Dev Spaces Operator によって認識されず、DevWorkspaces の一部の機能が機能しません。false の場合、管理者は che.infra.kubernetes.namespace.labels で指定されたラベルを使用して、namespace に手動でラベルを付ける必要があります。namespace を自分で管理する場合は、必ず https://www.eclipse.org/che/docs/stable/administration-guide/provisioning-namespaces-in-advance/ に従ってください。namespace に存在する追加のラベルはそのまま保持され、機能に影響を与えません。また、このプロパティーが true の場合でも、管理者は手動で namespace を事前に作成してラベルを付けることができます。namespace がすでにラベル付け要件を満たしている場合、namespace の更新は行われません。

デフォルト

true

3.3.2.2.4.12. CHE_INFRA_KUBERNETES_NAMESPACE_ANNOTATE

che-server がワークスペース namespace にアノテーションを付けるかどうかを定義します。

デフォルト

true

3.3.2.2.4.13. CHE_INFRA_KUBERNETES_NAMESPACE_LABELS

OpenShift Dev Spaces Workspaces に使用されるプロジェクトを検索するラベルの一覧。これらは **che.infra.kubernetes.namespace.annotations** と組み合わせてユーザー用に準備されたプロジェクトを検索し、ワークスペースでプロジェクトをアクティブにラベル付けするのに使用されます。注: OpenShift Dev Spaces Operator は、DevWorkspaces を調整するときにこれらのラベルとその正確な値に依存するため、このプロパティーの値を変更しないことを強くお勧めします。この設定が変更された場合、デフォルトのラベルと値を使用して手動でラベル付けされない限り、namespace は OpenShift Dev Spaces Operator によってワークスペース namespace として自動的に認識されません。namespace に追加のラベルを付けても、機能には影響しません。

デフォルト

app.kubernetes.io/part-of=che.eclipse.org,app.kubernetes.io/component=workspaces-namespace

3.3.2.2.4.14. CHE_INFRA_KUBERNETES_NAMESPACE_ANNOTATIONS

OpenShift Dev Spaces ユーザーワークスペース用に準備されているプロジェクトを検索するアノテーションの一覧。これらのアノテーションと照合されるのは、**che.infra.kubernetes.namespace.labels** と一致するプロジェクトのみです。このプロジェクトは、**che.infra.kubernetes.namespace.labels** と **che.infra.kubernetes.namespace.annotations** 両方に一致するプロジェクトは、優先的にユーザーのワークスペースに使用されます。**<username>** プレースホルダーを使用して、具体的なユーザーにプロジェクトを指定できます。これらは **che.infra.kubernetes.namespace.labels** と組み合わせてユーザー用に準備されたプロジェクトを検索し、ワークスペースでプロジェクトにアクティブにアノテーションを付けるのに使用されます。

デフォルト

che.eclipse.org/username=<username>

3.3.2.2.4.15. CHE_INFRA_KUBERNETES_SERVICE__ACCOUNT__NAME

すべてのワークスペース Pod にバインドするように指定する必要がある Kubernetes サービスアカウント名を定義します。Kubernetes インフラストラクチャーがサービスアカウントを作成しない OpenShift Dev Spaces オペレーターであり、存在する必要があります。OpenShift インフラストラク

チャーは、プロジェクトが事前に定義されているかどうかをチェックします (**che.infra.openshift.project** が空でない場合)。これが事前に定義されている場合はサービスアカウントが存在するはずです。これが 'NULL' または空の文字列の場合、インフラストラクチャーはワークスペースごとに新しい OpenShift プロジェクトを作成し、必要なロールを持つワークスペーのスサービスアカウントをここに準備します。

デフォルト

NULL

3.3.2.2.4.16. CHE_INFRA_KUBERNETES_WORKSPACE__SA__CLUSTER__ROLES

ワークスペースサービスアカウントで使用するオプションの追加のクラスターロールを指定します。クラスターロール名がすでに存在している必要がある OpenShift Dev Spaces オペレーター、および OpenShift Dev Spaces サービスアカウントは、これらのクラスターロールをワークスペースサービスアカウントに関連付けるためのロールバインディングを作成できる必要があります。名前はコンマで区切られます。このプロパティーは **che.infra.kubernetes.cluster_role_name** を非推奨にします。

デフォルト

NULL

3.3.2.2.4.17. CHE_INFRA_KUBERNETES_USER__CLUSTER__ROLES

名前空間でユーザーに割り当てるクラスターのロール

デフォルト

NULL

3.3.2.2.4.18. CHE_INFRA_KUBERNETES_WORKSPACE__START__TIMEOUT__MIN

Kubernetes ワークスペースの開始時間を制限する待機時間を定義します。

デフォルト

8

3.3.2.2.4.19. CHE_INFRA_KUBERNETES_INGRESS__START__TIMEOUT__MIN

Kubernetes Ingress が準備状態になる期間を制限するタイムアウトを分単位で定義します。

デフォルト

5

3.3.2.2.4.20. CHE_INFRA_KUBERNETES_WORKSPACE__UNRECOVERABLE__EVENTS

ワークスペースの起動中に、プロパティーで定義されたリカバリー不可能なイベントが発生する場合は、タイムアウトまで待機するのではなく、ワークスペースをすぐ終了します。OpenShift Dev Spaces Operator: リカバリー不能なイベントが発生する可能性があるため Failed の理由だけを追加できません。失敗したコンテナーの起動は、OpenShift Dev Spaces サーバーによって明示的に処理されます。

デフォルト

FailedMount,FailedScheduling,MountVolume.SetUpfailed,Failed to pull image,FailedCreate,ReplicaSetCreateError

3.3.2.2.4.21. CHE_INFRA_KUBERNETES_INGRESS_ANNOTATIONS__JSON

サーバーを公開するために使用される Ingress のアノテーションを定義します。値は Ingress コントローラーの種類によって異なります。OpenShift インフラストラクチャーは Ingress ではなくルートを使用するため、このプロパティーは無視されます。単一ホストデプロイメントストラテジーが機能する OpenShift Dev Spaces Operator は、URL の書き換えをサポートするコントローラーを使用する必要があります (そのため、サーバーはアプリケーションルートの変更をサポートする必要がありません)。**che.infra.kubernetes.ingress.path.rewrite_transform** プロパティーは、Ingress のパスが URL の書き換えをサポートするよう変換する方法を定義します。このプロパティーは、選択した Ingress コントローラーに対して実際に URL の書き換えを実行するように指示する ingress 自体のアノテーションのセットを定義します (選択された Ingress コントローラーで必要な場合)。たとえば、Nginx Ingress Controller 0.22.0 以降の場合、次の値が推奨されます: **{"ingress.kubernetes.io/rewrite-target": "/$1"、"ingress.kubernetes.io/ssl-redirect": "false"、\ "ingress.kubernetes.io/proxy-connect-timeout": "3600"、"ingress.kubernetes.io/proxy-read-timeout": "3600"、"nginx.org/websocket-services": "< service-name>"}** および **che.infra.kubernetes.ingress.path.rewrite_transform** は **"%s(。*)"** に設定する必要があります。0.22.0 よりも古い nginx Ingress コントローラーの場合には、rewrite-target は / に設定するだけで、パスは **%s** に変換されます (**che.infra.kubernetes.ingress.path.rewrite_transform** プロパティーを参照)。Ingress コントローラーが Ingress パスにある正規表現を使用する方法と、URL の書き換えを実行する方法についての説明は、nginx Ingress コントローラーのドキュメントを参照してください。

デフォルト

NULL

3.3.2.2.4.22. CHE_INFRA_KUBERNETES_INGRESS_PATH__TRANSFORM

サーバーを公開する Ingress のパスを宣言する方法についての recipe(レシピ) を定義します。**%s** はサーバーのベース公開 URL を表し、スラッシュで終了することが保証されています。このプロパティーは **String.format()** メソッドへの有効な入力であり、**%s** への参照が 1 つだけ含まれる必要があります。Ingress のアノテーションとパスを指定する際にこれら 2 つのプロパティーの相互作用を確認するには、**che.infra.kubernetes.ingress.annotations_json** プロパティーの説明を参照してください。これが定義されていない場合、このプロパティーはデフォルトで **%s** (引用符なし) に設定されます。これは、パスが Ingress コントローラーで使用する場合に変換されないことを意味します。

デフォルト

NULL

3.3.2.2.4.23. CHE_INFRA_KUBERNETES_INGRESS_LABELS

明確化できるように、OpenShift Dev Spaces サーバーによって作成されるすべての Ingress に追加する追加のラベル。

デフォルト

NULL

3.3.2.2.4.24. CHE_INFRA_KUBERNETES_POD_SECURITY__CONTEXT_RUN__AS__USER

Kubernetes インフラストラクチャーによって作成される Pod のセキュリティーコンテキストを定義します。これは OpenShift インフラによって無視されます。

デフォルト

NULL

3.3.2.2.4.25. CHE_INFRA_KUBERNETES_POD_SECURITY__CONTEXT_FS__GROUP

Kubernetes インフラストラクチャーによって作成される Pod のセキュリティーコンテキストを定義し

Kubernetes インフラストラクチャーによって作成される Pod のセキュリティーコンテキストを定義します。Pod の全コンテナーに適用される特別な補助グループです。OpenShift インフラは、このグループを無視します。

デフォルト

NULL

3.3.2.2.4.26. CHE_INFRA_KUBERNETES_POD_TERMINATION__GRACE__PERIOD__SEC

OpenShift インフラストラクチャーによって作成される Pod の猶予期間を定義します。デフォルト値: **0**これにより、Pod をすぐに停止し、ワークスペースの停止に必要な時間を短縮できます。OpenShift Dev Spaces Operator: **terminationGracePeriodSeconds** が OpenShift レシピで明示的に設定されている場合は上書きされません。

デフォルト

0

3.3.2.2.4.27. CHE_INFRA_KUBERNETES_TLS__ENABLED

Transport Layer Security(TLS) を有効にして Ingress を作成します。OpenShift インフラストラクチャーではルートは TLS に対応します。

デフォルト

false

3.3.2.2.4.28. CHE_INFRA_KUBERNETES_TLS__SECRET

TLS でワークスペース Ingress を作成する際に使用すべきシークレットの名前。OpenShift インフラストラクチャーでは、このプロパティーは無視されます。

デフォルト

空

3.3.2.2.4.29. CHE_INFRA_KUBERNETES_TLS__KEY

ワークスペース Ingress に使用する必要のある TLS Secret のデータ。**cert** および **key** は Base64 アルゴリズムでエンコードする必要があります。OpenShift インフラストラクチャーでは、これらのプロパティーは無視されます。

デフォルト

NULL

3.3.2.2.4.30. CHE_INFRA_KUBERNETES_TLS__CERT

ワークスペース Ingress に使用する必要のある TLS Secret の証明書データ。証明書は、Base64 アルゴリズムでエンコードする必要があります。OpenShift インフラストラクチャーでは、このプロパティーは無視されます。

デフォルト

NULL

3.3.2.2.4.31. CHE_INFRA_KUBERNETES_RUNTIMES__CONSISTENCY__CHECK__PERIOD__MIN

ランタイムの整合性チェックが実行される期間を定義します。ランタイムに一貫性のない状態がある場

合、ランタイムは自動的に停止します。値は 0 をより大きな値、または **-1** である必要があります。ここで、**-1** はチェックが実行されないことを意味します。OpenShift Dev Spaces Server がユーザーが操作を呼び出していない場合に Kubernetes API と対話できない場合に、OpenShift Dev Spaces サーバー設定が可能な OpenShift Dev Spaces サーバー設定があるため、デフォルトでは無効になっています。以下の設定で機能します。- ワークスペースオブジェクトは、OpenShift Dev Spaces Server が配置されているのと同じ名前空間に作成されます。- **cluster-admin** サービスアカウントトークンが OpenShift Dev Spaces サーバー Pod にマウントされます。OpenShift Dev Spaces Server は、OAuth プロバイダーからのトークンを使用して Kubernetes API と通信します。

デフォルト

-1

3.3.2.2.4.32. CHE_INFRA_KUBERNETES_TRUSTED__CA_SRC__CONFIGMAP

すべてのユーザーのワークスペースに伝播される追加の CATLS 証明書を含む OpenShift Dev Spaces サーバー名前空間の ConfigMap の名前。プロパティーを OpenShift 4 インフラストラクチャーに設定し、**che.infra.openshift.trusted_ca.dest_configmap_labels** に **config.openshift.io/inject-trusted-cabundle=true** ラベルが含まれる場合に、クラスター CA バンドルも伝播されます。

デフォルト

NULL

3.3.2.2.4.33. CHE_INFRA_KUBERNETES_TRUSTED__CA_DEST__CONFIGMAP

追加の CA TLS 証明書を含むワークスペース namespace の設定マップの名前。ワークスペース namespace にある **che.infra.kubernetes.trusted_ca.src_configmap** のコピーを保持します。この設定マップの内容は、プラグインブローカーを含むすべてのワークスペースコンテナーにマウントされます。既存の ConfigMap と競合しない限り、ConfigMap 名を変更しないでください。結果の ConfigMap 名を最終的に調整してプロジェクト内で一意にすることができる OpenShift Dev SpacesOperator。元の名前は **che.original_name** ラベルに保存されます。

デフォルト

ca-certs

3.3.2.2.4.34. CHE_INFRA_KUBERNETES_TRUSTED__CA_MOUNT__PATH

CA バンドルがマウントされるワークスペースコンテナーでパスを設定します。**che.infra.kubernetes.trusted_ca.dest_configmap** で指定される設定マップの内容がマウントされます。

デフォルト

/public-certs

3.3.2.2.4.35. CHE_INFRA_KUBERNETES_TRUSTED__CA_DEST__CONFIGMAP__LABELS

ユーザーワークスペースの CA 証明書の設定マップに追加するラベルのコンマ区切りの一覧。**che.infra.kubernetes.trusted_ca.dest_configmap** プロパティーを参照してください。

デフォルト

空

3.3.2.2.5. OpenShift インフラパラメーター

3.3.2.2.5.1. CHE_INFRA_OPENSHIFT_TRUSTED__CA_DEST__CONFIGMAP__LABELS

ユーザーワークスペースの CA 証明書の設定マップに追加するラベルのコンマ区切りの一覧。**che.infra.kubernetes.trusted_ca.dest_configmap** プロパティーを参照してください。このデフォルト値は、OpenShift 4 でのクラスター CA バンドルの自動挿入に使用されます。

デフォルト

config.openshift.io/inject-trusted-cabundle=true

3.3.2.2.5.2. CHE_INFRA_OPENSHIFT_ROUTE_LABELS

OpenShift Dev Spaces サーバーによって作成されたすべてのルートに追加する追加のラベルにより、明確な識別が可能になります。

デフォルト

NULL

3.3.2.2.5.3. CHE_INFRA_OPENSHIFT_ROUTE_HOST_DOMAIN__SUFFIX

ワークスペースルートの接尾辞として使用する必要のあるホスト名。例: **domain_suffix=<devspaces-__<openshift_deployment_name>__.__<domain_name>__>** を仕様すると、ルートは **routed3qrtk.<devspaces-__<openshift_deployment_name>__.__<domain_name>__>** のようになります。有効な DNS 名である必要があります。

デフォルト

NULL

3.3.2.2.5.4. CHE_INFRA_OPENSHIFT_PROJECT_INIT__WITH__SERVER__SA

OpenShift OAuth が有効な場合に、OpenShift Dev Spaces サーバーのサービスアカウントで OpenShift プロジェクトを初期化します。

デフォルト

true

3.3.2.2.6. 実験的なプロパティー

3.3.2.2.6.1. CHE_WORKSPACE_PLUGIN__BROKER_METADATA_IMAGE

ワークスペースツール設定を解決し、プラグインの依存関係をワークスペースにコピーする OpenShift Dev Spaces プラグインブローカーアプリケーションの Docker イメージ。OpenShift Dev Spaces Operator はデフォルトでこれらのイメージを上書きします。OpenShift Dev Spaces が Operator を使用してインストールされている場合、ここでイメージを変更しても効果がありません。

デフォルト

quay.io/eclipse/che-plugin-metadata-broker:v3.4.0

3.3.2.2.6.2. CHE_WORKSPACE_PLUGIN__BROKER_ARTIFACTS_IMAGE

OpenShift Dev Spaces プラグインのアーティファクトブローカーの Docker イメージ。このブローカーは、ワークスペース Pod で init コンテナーとして実行されます。このジョブは、プラグインの ID (レジストリー内のプラグインへの参照または、プラグインの meta.yaml へのリンク) の一覧を取り、ワーク

スペース向けに要求されたプラグインごとに、正しい .vsix and .theia 拡張子が /plugins ディレクトリーにダウンロードされていることを確認します。

デフォルト

quay.io/eclipse/che-plugin-artifacts-broker:v3.4.0

3.3.2.2.6.3. CHE_WORKSPACE_PLUGIN__BROKER_DEFAULT__MERGE__PLUGINS

プラグインをワークスペースにプロビジョニングする際にプラグインブローカーのデフォルト動作を設定します。true に設定すると、プラグインブローカーは可能な場合にプラグインのマージを試行します (つまり、それらは同じサイドカーイメージで実行され、設定が競合することはありません)。この値は、devfile で **mergePlugins** 属性が指定されていない場合に使用されるデフォルト設定です。

デフォルト

false

3.3.2.2.6.4. CHE_WORKSPACE_PLUGIN__BROKER_PULL__POLICY

ワークスペースツール設定を解決し、プラグインの依存関係をワークスペースにコピーする OpenShift Dev Spaces プラグインブローカーアプリケーションの Docker イメージ

デフォルト

Always

3.3.2.2.6.5. CHE_WORKSPACE_PLUGIN__BROKER_WAIT__TIMEOUT__MIN

プラグインブローカーの待機中に結果の最大期間を制限するタイムアウトを分単位で定義します。

デフォルト

3

3.3.2.2.6.6. CHE_WORKSPACE_PLUGIN__REGISTRY__URL

ワークスペースプラグインレジストリーのエンドポイント。有効な HTTP URL でなければなりません。例: http://che-plugin-registry-eclipse-che.192.168.65.2.nip.io OpenShift Dev Spaces プラグインレジストリーが不要な場合、値 'NULL' を使用する必要があります。

デフォルト

https://che-plugin-registry.prod-preview.openshift.io/v3

3.3.2.2.6.7. CHE_WORKSPACE_PLUGIN__REGISTRY__INTERNAL__URL

ワークスペースプラグインレジストリーの内部エンドポイント。有効な HTTP URL でなければなりません。例: http://devfile-registry.che.svc.cluster.local:8080 OpenShift Dev Spaces プラグインレジストリーが不要な場合、値 'NULL' を使用する必要があります

デフォルト

NULL

3.3.2.2.6.8. CHE_WORKSPACE_DEVFILE__REGISTRY__URL

devfile レジストリーエンドポイント。有効な HTTP URL でなければなりません。例: http://che-devfile-registry-eclipse-che.192.168.65.2.nip.io OpenShift Dev Spaces プラグインレジストリーが不要な場合、値 'NULL' を使用する必要があります。

デフォルト

https://che-devfile-registry.prod-preview.openshift.io/

3.3.2.2.6.9. CHE_WORKSPACE_DEVFILE__REGISTRY__INTERNAL__URL

devfile レジストリー "internal" エンドポイント。有効な HTTP URL でなければなりません。例: http://plugin-registry.che.svc.cluster.local:8080 OpenShift Dev Spaces プラグインレジストリーが不要な場合、値 'NULL' を使用する必要があります

デフォルト

NULL

3.3.2.2.6.10. CHE_WORKSPACE_STORAGE_AVAILABLE__TYPES

ダッシュボードなどのクライアントがワークスペースの作成/更新時にユーザーに提案するストレージタイプに使用できる値を定義する設定プロパティー。使用できる値: - **persistent**: 永続ストレージの I/O は低速だが永続性がある。- **ephemeral**: 一時ストレージは、高速 I/O を可能にするが、ストレージには制限があり、永続性がない。- **async**: 実験的機能: 非同期ストレージは一時ストレージと永続ストレージの組み合わせ。高速な I/O を可能にし、変更を維持し、停止時にバックアップを実行し、ワークスペースの開始時に復元します。**che.infra.kubernetes.pvc.strategy='common'** - **che.limits.user.workspaces.run.count=1** - **che.infra.kubernetes.namespace.default** に **<username>** が含まれる場合にのみ機能します。それ他の場合は、一覧から **async** を削除します。

デフォルト

persistent,ephemeral,async

3.3.2.2.6.11. CHE_WORKSPACE_STORAGE_PREFERRED__TYPE

ダッシュボードなどのクライアントがワークスペースの作成/更新時にユーザーに提案するストレージタイプのデフォルト値を定義する設定プロパティー。**async** 値は実験的な機能であるため、デフォルトタイプとしての使用は推奨されません。

デフォルト

永続

3.3.2.2.6.12. CHE_SERVER_SECURE__EXPOSER

セキュアなサーバーが認証で保護される方法を設定します。適切な値: **default**: **jwtproxy** はパススルーモードで設定されます。そのため、サーバーは要求を認証する必要があります。**jwtproxy**: **jwtproxy** は要求を認証します。そのため、サーバーは認証済みの要求のみを受信します。

デフォルト

jwtproxy

3.3.2.2.6.13. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_TOKEN_ISSUER

署名のない要求をルーティングするための **Jwtproxy** 発行側の文字列、トークンの有効期間およびオプションの認証ページのパス。

デフォルト

wsmaster

3.3.2.2.6.14. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_TOKEN_TTL

jwtproxy 発行者トークンの有効期間。

デフォルト

8800h

3.3.2.2.6.15. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_AUTH_LOADER_PATH

署名なしの要求をルーティングする認証ページのパス (任意)。

デフォルト

/_app/loader.html

3.3.2.2.6.16. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_IMAGE

jwtproxy イメージ。

デフォルト

quay.io/eclipse/che-jwtproxy:0.10.0

3.3.2.2.6.17. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_MEMORY__REQUEST

jwtproxy メモリー要求。

デフォルト

15mb

3.3.2.2.6.18. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_MEMORY__LIMIT

jwtproxy メモリー制限。

デフォルト

128mb

3.3.2.2.6.19. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_CPU__REQUEST

jwtproxy CPU 要求。

デフォルト

0.03

3.3.2.2.6.20. CHE_SERVER_SECURE__EXPOSER_JWTPROXY_CPU__LIMIT

jwtproxy CPU 制限。

デフォルト

0.5

3.3.2.2.7. 主な WebSocket エンドポイントの設定

3.3.2.2.7.1. CHE_CORE_JSONRPC_PROCESSOR__MAX__POOL__SIZE

JSON RPC 処理プールの最大サイズ。プールサイズが超過すると、メッセージの実行が拒否されます。

デフォルト

50

3.3.2.2.7.2. CHE_CORE_JSONRPC_PROCESSOR__CORE__POOL__SIZE

初期 JSON 処理プール。主な JSON RPC メッセージを処理するために使用されるスレッドの最小数。

デフォルト

5

3.3.2.2.7.3. CHE_CORE_JSONRPC_PROCESSOR__QUEUE__CAPACITY

Json RPC メッセージの処理に使用するキューの設定。

デフォルト

100000

3.3.2.2.7.4. CHE_METRICS_PORT

Prometheus メトリクスで公開される HTTP サーバーエンドポイントのポート

デフォルト

8087

3.3.2.2.8. CORS 設定

3.3.2.2.8.1. CHE_CORS_ALLOWED__ORIGINS

許可される要求元を指定します。WS Master の CORS フィルターはデフォルトで無効にされます。環境変数 "CHE_CORS_ENABLED=true" を使用してオンにします。

デフォルト

*****

3.3.2.2.8.2. CHE_CORS_ALLOW__CREDENTIALS

認証情報 (cookie、ヘッダー、TLS クライアント証明書) を使用して要求の処理を許可するかどうかを示します。

デフォルト

false

3.3.2.2.9. Factory のデフォルト

3.3.2.2.9.1. CHE_FACTORY_DEFAULT__PLUGINS

OpenShift Dev Spaces 固有のワークスペース記述子が含まれないリモート git リポジトリーから作成される Factory 用に作成されるエディターおよびプラグイン。複数のプラグインは、以下のようにコンマ

で区切る必要があります。例:
pluginFooPublisher/pluginFooName/pluginFooVersion,pluginBarPublisher/pluginBarName/pluginBarVersion

デフォルト

redhat/vscode-commons/latest

3.3.2.2.9.2. CHE_FACTORY_DEFAULT__DEVFILE__FILENAMES

リポジトリーベースの Factory(GitHub など) を検索する devfile のファイル名。Factory は、プロパティーで列挙される順序でこれらのファイルの特定を試みます。

デフォルト

devfile.yaml,.devfile.yaml

3.3.2.2.10. devfile のデフォルト

3.3.2.2.10.1. CHE_FACTORY_DEFAULT__EDITOR

OpenShift Dev Spaces 固有のワークスペース記述子が含まれないリモート Git リポジトリーから作成される Factory に使用されるエディター。

デフォルト

eclipse/che-theia/latest

3.3.2.2.10.2. CHE_FACTORY_SCM__FILE__FETCHER__LIMIT__BYTES

SCM リポジトリーからファイルを取得する URL フェッチャーのファイルサイズ制限。

デフォルト

102400

3.3.2.2.10.3. CHE_FACTORY_DEVFILE2__FILES__RESOLUTION__LIST

devfile v2 を補完する追加ファイルで、リポジトリーに含まれる場合があり、取得するには Factory の SCM リゾルバーサービスへのリンクとして参照する必要があります。

デフォルト

.che/che-editor.yaml,.che/che-theia-plugins.yaml,.vscode/extensions.json

3.3.2.2.10.4. CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR

指定されていない場合に Devfile にプロビジョニングする必要があるデフォルトのエディター。エディター形式は、**editorPublisher/editorName/editorVersion** 値になります。**NULL** または値がない場合は、デフォルトのエディターはプロビジョニングされません。

デフォルト

eclipse/che-theia/latest

3.3.2.2.10.5. CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS

デフォルトのエディター用にプロビジョニングする必要があるデフォルトのプラグイン。ユーザー定義の devfile で明示的に参照されていないこの一覧のすべてのプラグインはプロビジョニングされます

が、これはデフォルトのエディターが使用されているか、またはユーザー定義のエディターが (異なるバージョンの場合でも) デフォルトと同じである場合に限ります。形式は、コンマ区切りの **pluginPublisher/pluginName/pluginVersion** 値および URL です。例: **eclipse/che-theia-exec-plugin/0.0.1,eclipse/che-theia-terminal-plugin/0.0.1,https://cdn.pluginregistry.com/vi-mode/meta.yaml** プラグインが URL の場合、プラグインの **meta.yaml** はその URL から取得されます。

デフォルト

NULL

3.3.2.2.10.6. CHE_WORKSPACE_PROVISION_SECRET_LABELS

ユーザー namespace からシークレットを選択するためにラベルのコンマ区切りの一覧を定義します。これは、ファイルまたは環境変数としてワークスペースコンテナーにマウントされます。すべての指定されるラベルに一致するシークレットのみが選択されます。

デフォルト

app.kubernetes.io/part-of=che.eclipse.org,app.kubernetes.io/component=workspace-secret

3.3.2.2.10.7. CHE_WORKSPACE_DEVFILE_ASYNC_STORAGE_PLUGIN

非同期ストレージ機能がワークスペース設定で有効にされ、環境でサポートされる場合に、プラグインが追加されます。

デフォルト

eclipse/che-async-pv-plugin/latest

3.3.2.2.10.8. CHE_WORKSPACE_POD_NODE__SELECTOR

オプションでワークスペース Pod のノードセレクターを設定します。形式は、コンマ区切りの key=value ペアです (例: **disktype=ssd,cpu=xlarge,foo=bar**)。

デフォルト

NULL

3.3.2.2.10.9. CHE_WORKSPACE_POD_TOLERATIONS__JSON

オプションでワークスペース Pod の容認を設定します。形式は、テイントの容認の JSON 配列を表す文字列か、または **NULL** の場合はこれを無効にします。配列に含まれるオブジェクトは、toleration v1 コア仕様 に準拠する必要があります。例: **[{"effect":"NoExecute","key":"aNodeTaint","operator":"Equal","value":"aValue"}]**

デフォルト

NULL

3.3.2.2.10.10. CHE_INTEGRATION_BITBUCKET_SERVER__ENDPOINTS

Factory の統合に使用される Bitbucket エンドポイント。bitbucket サーバー URL のコンマ区切りの一覧、または統合が予想されない場合は NULL。

デフォルト

NULL

3.3.2.2.10.11. CHE_INTEGRATION_GITLAB_SERVER__ENDPOINTS

Factory の統合に使用される GitLab エンドポイント。GitLab サーバー URL のコンマ区切りの一覧、または統合が予想されない場合は NULL。

デフォルト

NULL

3.3.2.2.10.12. CHE_INTEGRATION_GITLAB_OAUTH__ENDPOINT

OAuth 2 統合が設定された GitLab サーバーのアドレス

デフォルト

NULL

3.3.2.2.10.13. CHE_OAUTH2_GITLAB_CLIENTID__FILEPATH

GitLab OAuth2 クライアントの設定。パーソナルアクセストークンの取得に使用されます。GitLab クライアント ID を持つファイルの場所。

デフォルト

NULL

3.3.2.2.10.14. CHE_OAUTH2_GITLAB_CLIENTSECRET__FILEPATH

GitLab クライアントシークレットを含むファイルの場所。

デフォルト

NULL#

3.3.2.2.11. Che システム

3.3.2.2.11.1. CHE_SYSTEM_SUPER__PRIVILEGED__MODE

System Super Privileged Mode (システムのスーパー特権モード)。getByKey、getByNameSpace、stopWorkspaces、および getResources の manageSystem パーミッションの追加パーミッションをユーザーに付与します。これらは、デフォルトでは管理者には提供されず、これらのパーミッションにより、管理者は admin 権限でそれらのワークスペースに名前を指定し、ワークスペースへの可視性を得ることができます。

デフォルト

false

3.3.2.2.11.2. CHE_SYSTEM_ADMIN__NAME

che.admin.name ユーザーのシステムパーミッションを付与します。ユーザーがすでに存在する場合は、これはコンポーネントの起動時に生じます。ユーザーがすでに存在しない場合は、ユーザーがデータベースで永続化される初回のログイン時に発生します。

デフォルト

admin

3.3.2.2.12. Workspace の制限

3.3.2.2.12.1. CHE_LIMITS_WORKSPACE_ENV_RAM

ワークスペースは、開発を行う際のユーザー向けの基本的なランタイムです。ワークスペースの作成方法や、消費されるリソースを制限するパラメーターを設定できます。ユーザーが新規ワークスペースの作成時にワークスペースに割り当てることができる RAM の最大量。RAM スライダーは、この最大値に合わせて調整されます。

デフォルト

16gb

3.3.2.2.12.2. CHE_LIMITS_WORKSPACE_IDLE_TIMEOUT

システムがワークスペースを一時停止した後にこれを停止する際に、ユーザーがワークスペースでアイドル状態になる期間 (ミリ秒単位)。アイドル状態は、ユーザーがワークスペースと対話しない期間です。つまり、エージェントのいずれも対話を受け取っていない期間を意味します。ブラウザーウィンドウを開いたままにするとアイドル状態になります。

デフォルト

1800000

3.3.2.2.12.3. CHE_LIMITS_WORKSPACE_RUN_TIMEOUT

システムが一時停止するまでの、アクティビティーを問わず、ワークスペースが実行される期間 (ミリ秒単位)。一定期間後にワークスペースを自動的に停止する場合は、このプロパティーを設定します。デフォルトはゼロで、実行タイムアウトがないことを意味します。

デフォルト

0

3.3.2.2.13. ユーザーワークスペースの制限

3.3.2.2.13.1. CHE_LIMITS_USER_WORKSPACES_RAM

単一ユーザーがワークスペースの実行に割り当てることができる RAM の合計量。ユーザーは、この RAM を単一のワークスペースに割り当てるか、または複数のワークスペースに分散することができます。

デフォルト

-1

3.3.2.2.13.2. CHE_LIMITS_USER_WORKSPACES_COUNT

ユーザーが作成できるワークスペースの最大数。追加のワークスペースを作成しようとすると、ユーザーにはエラーメッセージが表示されます。これは、実行中および停止中のワークスペースの合計数に適用されます。

デフォルト

-1

3.3.2.2.13.3. CHE_LIMITS_USER_WORKSPACES_RUN_COUNT

単一ユーザーが持てる実行中のワークスペースの最大数。ユーザーがこのしきい値に達し、追加のワークスペースを開始しようとすると、エラーメッセージと共にプロンプトが表示されます。ユーザーは、実行中のワークスペースを停止してから別のワークスペースをアクティベートする必要があります。

デフォルト

1

3.3.2.2.14. 組織ワークスペースの制限

3.3.2.2.14.1. CHE_LIMITS_ORGANIZATION_WORKSPACES_RAM

単一組織 (チーム) がワークスペースの実行に割り当てることができる RAM の合計量。組織の所有者はこの RAM を割り当てることができますが、チームのワークスペース全体で適切に割り当てられているように見えます。

デフォルト

-1

3.3.2.2.14.2. CHE_LIMITS_ORGANIZATION_WORKSPACES_COUNT

組織が所有できるワークスペースの最大数。追加のワークスペースを作成しようとすると、組織にはエラーメッセージが表示されます。これは、実行中および停止中のワークスペースの合計数に適用されます。

デフォルト

-1

3.3.2.2.14.3. CHE_LIMITS_ORGANIZATION_WORKSPACES_RUN_COUNT

単一組織が持てる実行中のワークスペースの最大数。組織がこのしきい値に達し、追加のワークスペースを開始しようとすると、エラーメッセージと共にプロンプトが表示されます。組織は、実行中のワークスペースを停止してから別のワークスペースをアクティベートする必要があります。

デフォルト

-1

3.3.2.2.15. マルチユーザー固有の OpenShift インフラストラクチャー設定

3.3.2.2.15.1. CHE_INFRA_OPENSHIFT_OAUTH__IDENTITY__PROVIDER

Keycloak に登録されている OpenShiftID プロバイダーのエイリアス。現在の OpenShift Dev Spaces ユーザーが所有する OpenShift 名前空間にワークスペース OpenShift リソースを作成するために使用する必要があります。**che.infra.openshift.project** が空白以外の値に設定する場合は NULL に設定する必要があります。OpenShift アイデンティティープロバイダー を参照してください。

デフォルト

NULL

3.3.2.2.16. OIDC 設定

3.3.2.2.16.1. CHE_OIDC_AUTH__SERVER__URL

OIDC ID プロバイダーサーバーへの URL は、**che.oidc.oidcProvider** が使用されている場合にのみ NULL に設定できます。

デフォルト

http://${CHE_HOST}:5050/auth

3.3.2.2.16.2. CHE_OIDC_AUTH__INTERNAL__SERVER__URL

内部ネットワークサービス URL から OIDC ID プロバイダーサーバー

デフォルト

NULL

3.3.2.2.16.3. CHE_OIDC_ALLOWED__CLOCK__SKEW__SEC

exp または **nbf** 要求を検証する際にクロックスキューについて許容される秒数。

デフォルト

3

3.3.2.2.16.4. CHE_OIDC_USERNAME__CLAIM

ユーザー名は、JWT トークンを解析するときにユーザー表示名として使用されると主張します。定義されていない場合、フォールバック値は Keycloak インストールでは 'preferred_username' であり、Dex インストールでは **name** です。

デフォルト

NULL

3.3.2.2.16.5. CHE_OIDC_EMAIL__CLAIM

JWT トークンの解析時に使用される電子メールクレーム。定義されていない場合、フォールバック値は 'email' です。

デフォルト

NULL

3.3.2.2.16.6. CHE_OIDC_OIDC__PROVIDER

この仕様 (Obtaining OpenID Provider Configuration Information) で詳細に検出エンドポイントを指定する別の OIDC プロバイダーのベース URL。非推奨。代わりに **che.oidc.auth_server_url** および **che.oidc.auth_internal_server_url** を使用します。

デフォルト

NULL

3.3.2.2.17. Keycloak の設定

3.3.2.2.17.1. CHE_KEYCLOAK_REALM

Keycloak レルムを使用してユーザーを認証するために使用されます。**che.keycloak.oidcProvider** が使用している場合のみ NULL に設定できます。

デフォルト

che

3.3.2.2.17.2. CHE_KEYCLOAK_CLIENT__ID

ダッシュボード、IDE、および CLI でユーザーを認証する **che.keycloak.realm** の Keycloak クライアント識別子。

デフォルト

che-public

3.3.2.2.17.3. CHE_KEYCLOAK_OSO_ENDPOINT

OSO OAuth トークンにアクセスするための URL

デフォルト

NULL

3.3.2.2.17.4. CHE_KEYCLOAK_GITHUB_ENDPOINT

Github OAuth トークンにアクセスするための URL

デフォルト

NULL

3.3.2.2.17.5. CHE_KEYCLOAK_USE__NONCE

OIDC オプションの **nonce** 機能を使用して、セキュリティーを強化します。

デフォルト

true

3.3.2.2.17.6. CHE_KEYCLOAK_JS__ADAPTER__URL

使用する Keycloak Javascript アダプターの URL。NULL に設定すると、デフォルト値が **${che.keycloak.auth_server_url}/js/keycloak.js** になり、別の **oidc_provider** を使用する場合には、**<che-server>/api/keycloak/OIDCKeycloak.js** になります。

デフォルト

NULL

3.3.2.2.17.7. CHE_KEYCLOAK_USE__FIXED__REDIRECT__URLS

固定されたリダイレクト URL のみをサポートする別の OIDC プロバイダーを使用する場合は true に設定します。このプロパティーは、**che.keycloak.oidc_provider** が NULL の場合は無視されます。

デフォルト

false

3.3.2.2.17.8. CHE_OAUTH_SERVICE__MODE

"embedded" モードまたは "delegated" モードで使用できる OAuth 認証サービスの設定。埋め込みに設定されている場合、サービスは OpenShift Dev Spaces の OAuthAuthenticator へのラッパーとして機能

します (シングルユーザーモードの場合と同様)。"delegated" に設定すると、サービスは Keycloak IdentityProvider メカニズムを使用します。このプロパティーが正しく設定されていない場合は、ランタイム例外 **wii** が出力されます。

デフォルト

delegated

3.3.2.2.17.9. CHE_KEYCLOAK_CASCADE__USER__REMOVAL__ENABLED

OpenShift Dev Spaces データベースからユーザーを削除するときに Keycloak サーバーからユーザーを削除できるようにするための設定。デフォルトで、これは無効にされます。OpenShift Dev Spaces データベースでユーザーを削除するときに、Keycloak から関連ユーザーの削除を実行する必要がある特別な場合に有効にできます。適切に機能するには、管理ユーザー名 ${che.keycloak.admin_username} とパスワード ${che.keycloak.admin_password} を設定する必要があります。

デフォルト

false

3.3.2.2.17.10. CHE_KEYCLOAK_ADMIN__USERNAME

Keycloak 管理者のユーザー名。OpenShift Dev Spaces データベースからユーザーを削除する際に Keycloak からユーザーを削除するために使用されます。 ${che.keycloak.cascade_user_removal_enabled} が 'true' に設定されている場合にのみ機能します。

デフォルト

NULL

3.3.2.2.17.11. CHE_KEYCLOAK_ADMIN__PASSWORD

Keycloak 管理者パスワード。OpenShift Dev Spaces データベースからユーザーを削除する際に Keycloak からユーザーを削除するために使用されます。 ${che.keycloak.cascade_user_removal_enabled} が 'true' に設定されている場合にのみ機能します。

デフォルト

NULL

3.3.2.2.17.12. CHE_KEYCLOAK_USERNAME_REPLACEMENT__PATTERNS

ユーザー名の調整の設定。OpenShift Dev Spaces は、Kubernetes オブジェクト名とラベルの一部としてユーザー名を使用する必要があるため、ID プロバイダーが通常許可するよりもフォーマットに厳しい要件があります (DNS 準拠である必要があります)。この調整は、コンマ区切りのキー/値のペアで表されます。これらは元のユーザー名の String.replaceAll 関数への引数として順次使用されます。キーは正規表現で、値は正規表現に一致するユーザー名の文字を置き換える置換文字列です。変更されたユーザー名は OpenShift Dev Spaces データベースにのみ保存され、ID プロバイダーにアドバタイズされません。DNS に準拠する文字を代替文字列として使用することが推奨されます (キー/値のペアの値)。例:\\=-,@=-at- では \ は **-** に、@ は **-at-** に変更され、ユーザー名 **org\user@com** は **org-user-at-com.** になります。

デフォルト

NULL

3.4. ワークスペースのグローバル設定

このセクションでは、管理者がワークスペースをグローバルに設定する方法について説明します。

- 「ユーザーが保持できるワークスペースの数を制限する」
- 「ユーザーが複数のワークスペースを同時に実行できるようにする」
- 「自己署名証明書を使用した Git リポジトリーをサポートする OpenShift Dev Spaces のデプロイ」
- 「ワークスペース nodeSelector の設定」

3.4.1. ユーザーが保持できるワークスペースの数を制限する

デフォルトでは、ユーザーはダッシュボードに無制限の数のワークスペースを保持できますが、この数を制限してクラスターの需要を減らすことができます。

この設定は、**CheCluster** カスタムリソースの一部です。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LIMITS_USER_WORKSPACES_COUNT: "<kept_workspaces_limit>" (1)
```

(1) ユーザーごとのワークスペースの最大数を設定します。デフォルト値 **-1** では、ユーザーは無制限の数のワークスペースを保持できます。ユーザーごとのワークスペースの最大数を設定するには、正の整数を使用します。

手順

1. OpenShift Dev Spaces namespace の名前を取得します。デフォルトは **openshift-devspaces** です。

```
$ oc get checluster --all-namespaces \
  -o=jsonpath="{.items[*].metadata.namespace}"
```

2. **CHE_LIMITS_USER_WORKSPACES_COUNT** を設定します。

```
$ oc patch checluster/devspaces -n openshift-devspaces \ (1)
--type='merge' -p \
'{"spec":{"components":{"cheServer":{"extraProperties":
{"CHE_LIMITS_USER_WORKSPACES_COUNT": "<kept_workspaces_limit>"}}}}}' (2)
```

(1) ステップ 1 で取得した OpenShift Dev Spaces 名前空間。

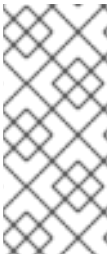
(2) **<kept_workspaces_limit>** の値を選択します。

関連資料

- 「CLI を使用して CheCluster カスタムリソースの設定」

3.4.2. ユーザーが複数のワークスペースを同時に実行できるようにする

デフォルトでは、ユーザーは一度に 1 つのワークスペースしか実行できません。ユーザーが複数のワークスペースを同時に実行できるようにすることができます。



注記

デフォルトのストレージ方法を使用している際、マルチノードクラスター内のノード全体に Pod が分散されている場合は、ワークスペースを同時に実行すると問題が発生する可能性があります。ユーザーごとの **common** ストレージストラテジーから **per-workspace** ストレージストラテジーに切り替えるか、**ephemeral** ストレージタイプを使用すると、これらの問題を回避または解決できます。

この設定は、**CheCluster** カスタムリソースの一部です。

```
spec:
  components:
    devWorkspace:
      runningLimit: "<running_workspaces_limit>" (1)
```

(1) ユーザーごとに同時に実行されるワークスペースの最大数を設定します。デフォルト値は **1** です。

手順

1. OpenShift Dev Spaces namespace の名前を取得します。デフォルトは **openshift-devspaces** です。

```
$ oc get checluster --all-namespaces \
  -o=jsonpath="{.items[*].metadata.namespace}"
```

2. **runningLimit** を設定します。

```
$ oc patch checluster/devspaces -n openshift-devspaces \ (1)
--type='merge' -p \
'{"spec":{"components":{"devWorkspace":{"runningLimit":
"<running_workspaces_limit>"}}}}' (2)
```

(1) ステップ 1 で取得した OpenShift Dev Spaces 名前空間。

(2) **<running_workspaces_limit>** の値を選択します。

関連情報

- 「CLI を使用して CheCluster カスタムリソースの設定」

3.4.3. 自己署名証明書を使用した Git リポジトリーをサポートする OpenShift Dev Spaces のデプロイ

自己署名証明書を使用する Git プロバイダーでの操作をサポートするように OpenShift Dev Spaces を設定できます。

前提条件

- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the OpenShift CLI を参照してください。
- Git バージョン 2 以降

手順

1. Git サーバーの詳細情報を使用して新規の **configMap** を作成します。

```
$ oc create configmap che-git-self-signed-cert \
  --from-file=ca.crt=<path_to_certificate> \ 1
  --from-literal=githost=<host:port> -n openshift-devspaces 2
```

1 自己署名証明書へのパス

2 Git サーバーの HTTPS 接続のホストおよびポート (オプション)。

注記

- **githost** を指定しないと、指定された証明書がすべての HTTPS リポジトリーに使用されます。
- 証明書ファイルは、通常、以下のような Base64 ASCII ファイルとして保存されます。**.pem**, **.crt**, **.ca-bundle**.また、これらはバイナリーデータとしてエンコードすることもできます (例: **.cer**)。証明書ファイルを保持するすべての **Secrets** は、バイナリーデータ証明書ではなく、Base64 ASCII 証明書を使用する必要があります。

2. 必要なラベルを ConfigMap に追加します。

```
$ oc label configmap che-git-self-signed-cert \
  app.kubernetes.io/part-of=che.eclipse.org -n openshift-devspaces
```

3. Git リポジトリーに自己署名証明書を使用するように OpenShift Dev Spaces オペランドを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  devEnvironments:
    trustedCerts:
      gitTrustedCertsConfigMapName: che-git-self-signed-cert
```

検証手順

- 新規ワークスペースを作成および開始します。ワークスペースによって使用されるすべてのコンテナーは、自己署名証明書のあるファイルを含む特殊なボリュームをマウントします。コンテナーの **/etc/gitconfig** ファイルには、Git サーバーホスト (その URL) と **http** セクションの証明書へのパスについての情報が含まれます (git-config に関する Git ドキュメントを参照してください)。

例3.11 /etc/gitconfig ファイルの内容

```
[http "https://10.33.177.118:3000"]
sslCAInfo = /etc/config/che-git-tls-creds/certificate
```

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」
- 「信頼できない TLS 証明書の OpenShift Dev Space へのインポート」 の形式にする必要があります。

3.4.4. ワークスペース nodeSelector の設定

このセクションでは、OpenShift Dev Spaces ワークスペースの Pod に **nodeSelector** を設定する方法を説明します。

手順

OpenShift Dev Spaces は、**CHE_WORKSPACE_POD_NODE__SELECTOR** 環境変数を使用して **nodeSelector** を設定します。この変数には、nodeSelector ルールを形成するためにコンマ区切りの **key=value** ペアのセットが含まれるか、またはこれを無効にする **NULL** が含まれる場合があります。

```
CHE_WORKSPACE_POD_NODE__SELECTOR=disktype=ssd,cpu=xlarge,[key=value]
```

重要

nodeSelector は、OpenShift Dev Spaces のインストール時に設定する必要があります。これにより、既存のワークスペース PVC および Pod が異なるゾーンにスケジュールされることによってボリュームのアフィニティーの競合が生じ、既存のワークスペースが実行できなくなることを防ぐことができます。

大規模なマルチゾーンクラスターの異なるゾーンに Pod および PVC がスケジュールされないようにするには、PVC の作成プロセスを調整する追加の **StorageClass** オブジェクトを作成します (**allowedTopologies** フィールドに注目してください)。

新規に作成された **StorageClass** の名前を、**CHE_INFRA_KUBERNETES_PVC_STORAGE__CLASS__NAME** 環境変数で OpenShift Dev Spaces に指定します。この変数のデフォルトの空の値の場合、OpenShift Dev Spaces に対し、クラスターのデフォルト **StorageClass** を使用するように指示します。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.5. ワークスペースの起動を迅速化するイメージのキャッシュ

OpenShift Dev Spaces ワークスペースの起動時間のパフォーマンスを改善するには、Image Puller を使用して OpenShift クラスターのイメージの事前プルに使用できる OpenShift Dev Spaces に依存しないコンポーネントを使用します。Image Puller は、関連する OpenShift Dev Spaces ワークスペースイメージを各ノードで事前にプルするように設定できる **DaemonSet** を作成する追加の OpenShift デプロイメントです。これらのイメージは、OpenShift Dev Spaces ワークスペースの起動時にすでに利用可能なため、ワークスペースの開始時間が改善されています。

Image Puller は、設定用に以下のパラメーターを提供します。

表3.16 Image Puller パラメーター

パラメーター	使用方法	デフォルト
CACHING_INTERVAL_HOURS	デーモンセットのヘルスチェック間隔 (時間単位)	**"1"**
CACHING_MEMORY_REQUEST	Puller の実行中にキャッシュされる各イメージのメモリー要求。「Image Puller のメモリーパラメーターの定義」を参照してください。	**10Mi**
CACHING_MEMORY_LIMIT	Puller の実行中にキャッシュされる各イメージのメモリー制限。「Image Puller のメモリーパラメーターの定義」を参照してください。	**20Mi**
CACHING_CPU_REQUEST	Puller の実行中にキャッシュされる各イメージのプロセッサー要求	**.05** または 50 ミリコア
CACHING_CPU_LIMIT	Puller の実行中にキャッシュされる各イメージのプロセッサー制限	**.2** または 200 ミリコア
DAEMONSET_NAME	作成するデーモンセットの名前	**kubernetes-image-puller**
DEPLOYMENT_NAME	作成するデプロイメントの名前	**kubernetes-image-puller**
NAMESPACE	作成するデーモンセットが含まれる OpenShift プロジェクト	**k8s-image-puller**
IMAGES	プルするイメージのセミコロンで区切られた一覧 (**<name1>=<image1>;<name2>=<image2>** の形式)。「プルするイメージの一覧の定義」を参照してください。	
NODE_SELECTOR	デーモンセットによって作成される Pod に適用するノードセレクター	**'{}'**
AFFINITY	DaemonSet によって作成される Pod に適用されるアフィニティー	**'{}'**

パラメーター	使用方法	デフォルト
IMAGE_PULL_SECRETS	DeamonSet で作成される Pod に追加する **pullsecret1;...** 形式のイメージプルシークレットの一覧。これらのシークレットはイメージ puller の namespace に配置し、クラスター管理者はそれらを作成する必要があります。	""

関連情報

- 「プルするイメージの一覧の定義」
- 「Image Puller のメモリーパラメーターの定義」
- 「Web コンソールを使用した OpenShift への ImagePuller のインストール」
- 「CLI を使用した OpenShift への Image Puller のインストール」
- Kubernetes Image Puller ソースコードリポジトリー

3.5.1. プルするイメージの一覧の定義

Image Puller は、**che-machine-exec** などの scratch イメージを含むほとんどのイメージを事前プルできます。ただし、**traefik** などの Dockerfile にボリュームをマウントするイメージは、OpenShift 3.11 における事前プルではサポートされません。

手順

1. **https://devspaces-<openshift_deployment_name>.<domain_name>/plugin-registry/v3/external_images.txt** に移動して、プリプルに関連するコンテナーイメージのリストを収集します。
2. プル前の一覧からイメージを判別します。ワークスペースの起動時間を短縮するには、**che-theia**、**che-machine-exec**、**che-theia-endpoint-runtime-binary**、プラグインサイドカーイメージなどのワークスペース関連のイメージを事前にプルすることを検討してください。

関連情報

- 「Web コンソールを使用した OpenShift への ImagePuller のインストール」
- 「CLI を使用した OpenShift への Image Puller のインストール」

3.5.2. Image Puller のメモリーパラメーターの定義

メモリー要求および制限パラメーターを定義して、コンテナーをプルし、プラットフォームに実行するのに十分なメモリーがあることを確認します。

前提条件

- 「プルするイメージの一覧の定義」

手順

手順

1. **CACHING_MEMORY_REQUEST** または **CACHING_MEMORY_LIMIT** の最小値を定義するには、プルする各コンテナーイメージの実行に必要なメモリー容量を考慮してください。
2. **CACHING_MEMORY_REQUEST** または **CACHING_MEMORY_LIMIT** の最大値を定義するには、クラスターのデーモンセット Pod に割り当てられるメモリーの合計を考慮します。

   ```
   (memory limit) * (number of images) * (number of nodes in the cluster)
   ```

   コンテナーのメモリー制限が **20Mi** の 20 ノードで 5 つのイメージをプルする場合、 **2000Mi** のメモリーが必要です。

関連情報

- 「Web コンソールを使用した OpenShift への ImagePuller のインストール」
- 「CLI を使用した OpenShift への Image Puller のインストール」

3.5.3. Web コンソールを使用した OpenShift への ImagePuller のインストール

OpenShift Web コンソールを使用して、コミュニティーでサポートされている Kubernetes Image Puller Operator を OpenShift にインストールできます。

前提条件

- 「プルするイメージの一覧の定義」
- 「Image Puller のメモリーパラメーターの定義」
- クラスター管理者による OpenShift Web コンソールセッション。Accessing the web console を参照してください。

手順

1. コミュニティーでサポートされている Kubernetes Image Puller Operator をインストールします。Installing from OperatorHub using the web console を参照してください。
2. コミュニティーでサポートされている Kubernetes Image Puller Operator から **KubernetesImagePuller** オペランドを作成します。Creating applications from installed Operators を参照してください。

3.5.4. CLI を使用した OpenShift への Image Puller のインストール

OpenShift **oc** 管理ツールを使用して、OpenShift に Kubernetes Image Puller をインストールできます。

前提条件

- 「プルするイメージの一覧の定義」
- 「Image Puller のメモリーパラメーターの定義」
- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the OpenShift CLI を参照してください。

手順

1. Image Puller リポジトリーのクローンを作成し、OpenShift テンプレートが含まれるディレクトリーを取得します。

```
$ git clone https://github.com/che-incubator/kubernetes-image-puller
$ cd kubernetes-image-puller/deploy/openshift
```

2. 以下のパラメーターを使用して、**app.yaml**、**configmap.yaml** および **serviceaccount.yaml** OpenShift テンプレートを設定します。

表3.17 app.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用方法	デフォルト
DEPLOYMENT_NAME	ConfigMap の **DEPLOYMENT_NAME** の値	**kubernetes-image-puller**
IMAGE	**kubernetes-image-puller** デプロイメントに使用されるイメージ	**registry.redhat.io/devspaces/imagepuller-rhel8:3.2**
IMAGE_TAG	プルするイメージタグ	**latest**
SERVICEACCOUNT_NAME	デプロイメントで作成され、使用される ServiceAccount の名前	**kubernetes-image-puller**

表3.18 configmap.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用方法	デフォルト
CACHING_CPU_LIMIT	ConfigMap の **CACHING_CPU_LIMIT** の値	**.2**
CACHING_CPU_REQUEST	ConfigMap の **CACHING_CPU_REQUEST** の値	**.05**
CACHING_INTERVAL_HOURS	ConfigMap の **CACHING_INTERVAL_HOURS** の値	**"1"**
CACHING_MEMORY_LIMIT	ConfigMap の **CACHING_MEMORY_LIMIT** の値	**"20Mi"**
CACHING_MEMORY_REQUEST	ConfigMap の **CACHING_MEMORY_REQUEST** の値	**"10Mi"**

値	使用方法	デフォルト
DAEMONSET_NAME	ConfigMap の **DAEMONSET_NAME** の値	**kubernetes-image-puller**
DEPLOYMENT_NAME	ConfigMap の **DEPLOYMENT_NAME** の値	**kubernetes-image-puller**
IMAGES	ConfigMap の **IMAGES** の値	**"undefined"**
NAMESPACE	ConfigMap の **NAMESPACE** の値	**k8s-image-puller**
NODE_SELECTOR	ConfigMap の **NODE_SELECTOR** の値	**"{}"**

表3.19 serviceaccount.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用方法	デフォルト
SERVICEACCOUNT_NAME	デプロイメントで作成され、使用される ServiceAccount の名前	**kubernetes-image-puller**

3. Image Puller をホストする OpenShift プロジェクトを作成します。

```
$ oc new-project <k8s-image-puller>
```

4. テンプレートを処理してから適用し、Puller をインストールします。

```
$ oc process -f serviceaccount.yaml | oc apply -f -
$ oc process -f configmap.yaml | oc apply -f -
$ oc process -f app.yaml | oc apply -f -
```

検証手順

1. **<kubernetes-image-puller>** デプロイメントおよび **<kubernetes-image-puller>** デーモンセットがあることを確認します。デーモンセットでは、クラスター内の各ノードに Pod が必要です。

```
$ oc get deployment,daemonset,pod --namespace <k8s-image-puller>
```

2. **<kubernetes-image-puller> ConfigMap** の値を確認します。

```
$ oc get configmap <kubernetes-image-puller> --output yaml
```

3.6. 可観測性の設定

OpenShift Dev Spaces 可観測性機能を設定するには、以下を参照してください。

- 「Che-Theia ワークスペース」
- 「サーバーロギングの設定」
- 「dsc を使用したログの収集」
- 「DevWorkspace Operator のモニタリング」
- 「OpenShift Dev Spaces サーバーのモニタリング」

3.6.1. Che-Theia ワークスペース

3.6.1.1. Telemetry の概要

Telemetry は、操作データの明示的かつ論理的なコレクションです。デフォルトで、Telemetry は Red Hat OpenShift Dev Spaces では利用できませんが、Che-Theia エディターにはプラグインメカニズムを使用し、**chectl** コマンドラインツールの使用データをセグメントを使用して収集できる抽象 API があります。このアプローチは、すべての Che-Theia ワークスペースでテレメトリーが有効になっている Eclipse Che hosted by Red Hat サービスで使用されます。

以下では、Red Hat OpenShift Dev Spaces に独自の Telemetry クライアントを作成する方法について説明し、次に Red Hat OpenShift Dev Spaces Woopra Telemetry プラグイン の概要を示します。

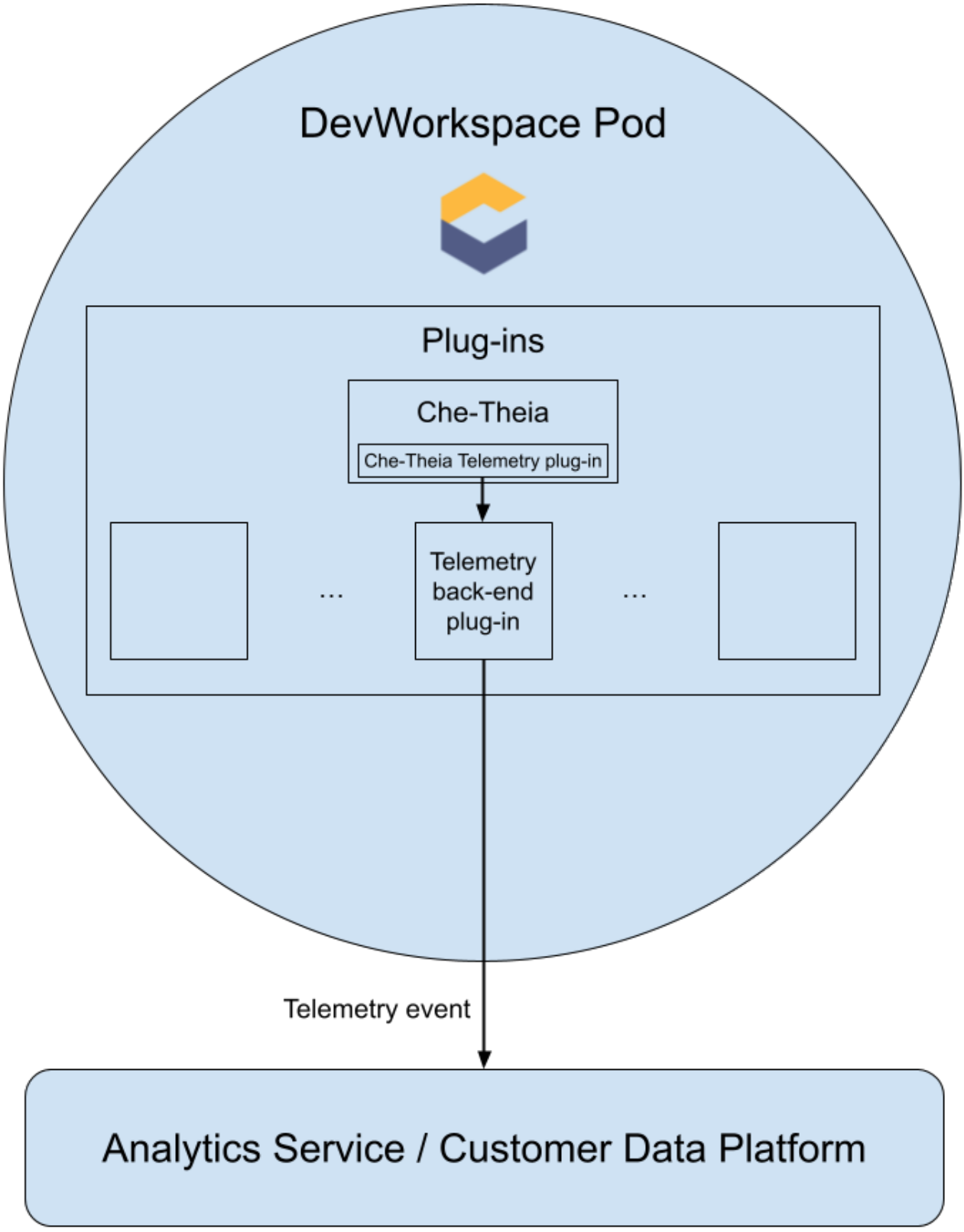
3.6.1.2. ユースケース

Red Hat OpenShift Dev Spaces Telemetry API では、以下の追跡が可能です。

- ワークスペース使用の期間
- ファイルの編集、コミット、およびリモートリポジトリーへのプッシュなどのユーザー駆動型アクション
- ワークスペースで使用されるプログラミング言語および devfile

3.6.1.3. 仕組み

DevWorkspace が起動すると、**che-theia** コンテナーは、テレメトリーイベントをバックエンドに送信するロールを担うテレメトリープラグインを起動します。**$DEVWORKSPACE_TELEMETRY_BACKEND_PORT** 環境変数が DevWorkspace Pod で設定されている場合、テレメトリープラグインはそのポートでリッスンしているバックエンドにイベントを送信します。バックエンドは、受信したイベントをイベントのバックエンド固有の表現に変換し、設定された分析バックエンド (Segment や Woopra など) に送信します。



3.6.1.4. Che-Theia Telemetry プラグインによってバックエンドに送信されるイベント

イベント	説明
WORKSPACE_OPENED	Che-Theia の起動時に送信されます。

イベント	説明
COMMIT_LOCALLY	**git.commitTheia** コマンドを使用してローカルでコミットが行われたときに送信されます
PUSH_TO_REMOTE	**git.push** Theia コマンドで Git のプッシュが実行される際に送信されます。
EDITOR_USED	エディターでファイルが変更されたときに送信されます

WORKSPACE_INACTIVE や **WORKSPACE_STOPPED** などの他のイベントは、バックエンドプラグイン内で検出できます。

3.6.1.5. Woopra Telemetry プラグイン

Woopra Telemetry プラグイン は、Telemetry を Red Hat OpenShift Dev Spaces インストールから Segment および Woopra に送信するためにビルドされたプラグインです。このプラグインは 、Red Hat によってホストされる Eclipse Che によって使用されますが、Red Hat OpenShift Dev Spaces デプロイメントはこのプラグインを利用できます。有効な Woopra ドメインおよびセグメント書き込みキー以外の依存関係はありません。プラグインである plugin.yaml の devfile v2 には、プラグインに渡すことのできる 4 つの環境変数があります。

- **WOOPRA_DOMAIN** - イベントの送信先となる Woopra ドメイン。
- **SEGMENT_WRITE_KEY** - セグメントおよび Woopra にイベントを送信するための書き込みキー。
- **WOOPRA_DOMAIN_ENDPOINT** - Woopra ドメインを直接渡さない場合、プラグインは Woopra ドメインを返す指定の HTTP エンドポイントからこれを取得します。
- **SEGMENT_WRITE_KEY_ENDPOINT** - セグメント書き込みキーを直接渡さない場合、プラグインはセグメント書き込みキーを返す指定された HTTP エンドポイントからこれを取得します。

Red Hat OpenShift Dev Spaces インストールで Woopra プラグインを有効にするには、以下を実行します。

手順

- **plugin.yaml** devfile v2 ファイルを、環境変数が正しく設定された HTTP サーバーにデプロイします。

1. **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  devEnvironments:
    defaultPlugins:
    - editor: eclipse/che-theia/next    (1)
      plugins:                          (2)
      - 'https://your-web-server/plugin.yaml'
```

1. Telemetry プラグインを設定するための **editorId**。
2. Telemetry プラグインの devfile v2 定義への URL。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.6.1.6. Telemetry プラグインの作成

本セクションでは、**AbstractAnalyticsManager** を拡張し、以下のメソッドを実装する **AnalyticsManager** クラスを作成する方法を説明します。

- **isEnabled()**: Telemetry バックエンドが正しく機能しているかどうかを判断します。これは、常に **true** を返すか、または接続プロパティーがない場合に **false** を返すなど、より複雑なチェックがあることを意味します。
- **destroy()**: Telemetry バックエンドをシャットダウンする前に実行されるクリーンアップ方法。このメソッドは、**WORKSPACE_STOPPED** イベントを送信します。
- **onActivity()** - 特定のユーザーについて一部のアクティビティーが依然として実行されていることを通知します。これは主に **WORKSPACE_INACTIVE** イベントを送信するために使用されます。
- **onEvent()** - Telemetry イベントを **WORKSPACE_USED** または **WORKSPACE_STARTED** などの Telemetry サーバーに送信します。
- **increaseDuration()** - 短時間に多くのイベントを送信するのではなく、現在のイベントの期間を長くします。

次のセクションでは、以下について説明します。

- Telemetry サーバーを作成してイベントを標準出力にエコーします。
- OpenShift Dev Spaces Telemetry クライアントを拡張して、ユーザーのカスタムバックエンドを実装します。
- カスタムバックエンドの DevWorkspace プラグインを表す **plugin.yaml** ファイルを作成します。
- **CheCluster** カスタムリソースから **workspacesDefaultPlugins** 属性を設定して、カスタムプラグインの場所を OpenShift Dev Spaces に指定します。

3.6.1.6.1. スタートガイド

以下では、OpenShift Dev Spaces Telemetry システムを拡張してカスタムバックエンドと通信するために必要な手順を説明します。

1. イベントを受信するサーバープロセスの作成
2. イベントをサーバーに送信するバックエンドを作成する OpenShift Dev Spaces ライブラリーの拡張

3. コンテナーでの Telemetry バックエンドのパッケージ化およびイメージレジストリーへのデプロイ

4. バックエンドのプラグインを追加し、OpenShift Dev Space に DevWorkspaces にプラグインを読み込むよう指示

Telemetry バックエンドの最終的な例については、here を参照してください。

イベントを受信するサーバーの作成
この例は、Telemetry プラグインからイベントを受信し、標準出力に書き込むサーバーを作成する方法を示しています。

実稼働環境のユースケースでは、独自の Telemetry サーバーを作成するのではなく、サードパーティーの Telemetry システム (Segment、Woopra など) との統合を検討してください。この場合、プロバイダーの API を使用してイベントをカスタムバックエンドからシステムに送信します。

以下の Go コードは、ポート **8080** でサーバーを起動し、イベントを標準出力に書き込みます。

例3.12 main.go

```
package main

import (
 "io/ioutil"
 "net/http"

 "go.uber.org/zap"
)

var logger *zap.SugaredLogger

func event(w http.ResponseWriter, req *http.Request) {
 switch req.Method {
 case "GET":
  logger.Info("GET /event")
 case "POST":
  logger.Info("POST /event")
 }
 body, err := req.GetBody()
 if err != nil {
  logger.With("err", err).Info("error getting body")
  return
 }
 responseBody, err := ioutil.ReadAll(body)
 if err != nil {
  logger.With("error", err).Info("error reading response body")
  return
 }
 logger.With("body", string(responseBody)).Info("got event")
}

func activity(w http.ResponseWriter, req *http.Request) {
 switch req.Method {
 case "GET":
  logger.Info("GET /activity, doing nothing")
 case "POST":
```

```
  logger.Info("POST /activity")
  body, err := req.GetBody()
  if err != nil {
   logger.With("error", err).Info("error getting body")
   return
  }
  responseBody, err := ioutil.ReadAll(body)
  if err != nil {
   logger.With("error", err).Info("error reading response body")
   return
  }
  logger.With("body", string(responseBody)).Info("got activity")
 }
}

func main() {

 log, _ := zap.NewProduction()
 logger = log.Sugar()

 http.HandleFunc("/event", event)
 http.HandleFunc("/activity", activity)
 logger.Info("Added Handlers")

 logger.Info("Starting to serve")
 http.ListenAndServe(":8080", nil)
}
```

このコードに基づいてコンテナーイメージを作成し、これを OpenShift の **openshift-devspaces** プロジェクトでデプロイメントとして公開します。サンプル Telemetry サーバーのコードは Telemetry-server-example で利用できます。Telemetry サーバーをデプロイするには、リポジトリーのクローンを作成し、コンテナーをビルドします。

```
$ git clone https://github.com/che-incubator/telemetry-server-example
$ cd telemetry-server-example
$ podman build -t registry/organization/telemetry-server-example:latest .
$ podman push registry/organization/telemetry-server-example:latest
```

manifest_with_ingress.yaml および **manifest_with_route** の両方には、Deployment およびサービスの定義が含まれます。また、前者は Kubernetes Ingress も定義しますが、後者は OpenShift Route を定義します。

マニフェストファイルで、プッシュした **image** に一致する image および **host** フィールドと、OpenShift クラスターのパブリックホスト名を置き換えます。次に、以下を実行します。

```
$ kubectl apply -f manifest_with_[ingress|route].yaml -n openshift-devspaces
```

3.6.1.6.2. バックエンドプロジェクトの作成

注記

開発時に迅速なフィードバックを得るには、DevWorkspace 内で開発を行うことが推奨されます。これにより、クラスターでアプリケーションを実行し、フロントエンドの Telemetry プラグインからイベントを受信できます。

1. Maven Quarkus プロジェクトのスキャフォールディング:

```
mvn io.quarkus:quarkus-maven-plugin:2.7.1.Final:create \
    -DprojectGroupId=mygroup -DprojectArtifactId=devworkspace-telemetry-example-plugin \
-DprojectVersion=1.0.0-SNAPSHOT
```

2. **src/main/java/mygroup** と **src/test/java/mygroup** の下にあるファイルを削除します。

3. **backend-base** の最新バージョンおよび Maven コーディネートについては、GitHub パッケージ を参照してください。

4. 以下の依存関係を **pom.xml** に追加します。

例3.13 pom.xml

```
<!-- Required -->
<dependency>
    <groupId>org.eclipse.che.incubator.workspace-telemetry</groupId>
    <artifactId>backend-base</artifactId>
    <version>LATEST VERSION FROM PREVIOUS STEP</version>
</dependency>


<!-- Used to make http requests to the telemetry server -->
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest-client</artifactId>
</dependency>
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-rest-client-jackson</artifactId>
</dependency>
```

5. **read:packages** パーミッションでパーソナルアクセストークンを作成し、GitHub パッケージから **org.eclipse.che.incubator.workspace-telemetry:backend-base** 依存関係をダウンロードします。

6. GitHub ユーザー名、個人アクセストークン、**che-incubator** リポジトリーの詳細を **~/.m2/settings.xml** ファイルに追加します。

例3.14 settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>che-incubator</id>
```

```
        <username>YOUR GITHUB USERNAME</username>
        <password>YOUR GITHUB TOKEN</password>
      </server>
    </servers>

    <profiles>
      <profile>
        <id>github</id>
        <activation>
          <activeByDefault>true</activeByDefault>
        </activation>
        <repositories>
          <repository>
            <id>central</id>
            <url>https://repo1.maven.org/maven2</url>
            <releases><enabled>true</enabled></releases>
            <snapshots><enabled>false</enabled></snapshots>
            </repository>
            <repository>
            <id>che-incubator</id>
            <url>https://maven.pkg.github.com/che-incubator/che-workspace-telemetry-
client</url>
          </repository>
        </repositories>
      </profile>
    </profiles>
</settings>
```

3.6.1.6.3. AnalyticsManager の具体的な実装の作成および特殊なロジックの追加

src/main/java/mygroup の下に、プロジェクトに 2 つのファイルを作成します。

- **MainConfiguration.java** - **AnalyticsManager** に提供される設定が含まれます。
- **AnalyticsManager.java**: Telemetry システム固有のロジックが含まれます。

例3.15 MainConfiguration.java

```
package org.my.group;

import java.util.Optional;

import javax.enterprise.context.Dependent;
import javax.enterprise.inject.Alternative;

import org.eclipse.che.incubator.workspace.telemetry.base.BaseConfiguration;
import org.eclipse.microprofile.config.inject.ConfigProperty;

@Dependent
@Alternative
public class MainConfiguration extends BaseConfiguration {
    @ConfigProperty(name = "welcome.message")  1
    Optional<String> welcomeMessage;           2
}
```

```
-
```

1 MicroProfile 設定アノテーションは、**welcome.message** 設定を注入するために使用されます。

バックエンドに固有の設定プロパティーを設定する方法の詳細は、Quarkus 設定リファレンスガイド を参照してください。

例3.16 AnalyticsManager.java

```
package org.my.group;

import java.util.HashMap;
import java.util.Map;

import javax.enterprise.context.Dependent;
import javax.enterprise.inject.Alternative;
import javax.inject.Inject;

import org.eclipse.che.incubator.workspace.telemetry.base.AbstractAnalyticsManager;
import org.eclipse.che.incubator.workspace.telemetry.base.AnalyticsEvent;
import org.eclipse.che.incubator.workspace.telemetry.finder.DevWorkspaceFinder;
import org.eclipse.che.incubator.workspace.telemetry.finder.UsernameFinder;
import org.eclipse.microprofile.rest.client.inject.RestClient;
import org.slf4j.Logger;

import static org.slf4j.LoggerFactory.getLogger;

@Dependent
@Alternative
public class AnalyticsManager extends AbstractAnalyticsManager {

    private static final Logger LOG = getLogger(AbstractAnalyticsManager.class);

    public AnalyticsManager(MainConfiguration mainConfiguration, DevWorkspaceFinder
devworkspaceFinder, UsernameFinder usernameFinder) {
        super(mainConfiguration, devworkspaceFinder, usernameFinder);

        mainConfiguration.welcomeMessage.ifPresentOrElse(    1
            (str) -> LOG.info("The welcome message is: {}", str),
            () -> LOG.info("No welcome message provided")
        );
    }

    @Override
    public boolean isEnabled() {
        return true;
    }

    @Override
    public void destroy() {}

    @Override
    public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent, String
resolution, Map<String, Object> properties) {
        LOG.info("The received event is: {}", event);    2
```

```
    }

    @Override
    public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) { }

    @Override
    public void onActivity() {}
}
```

1. 提供された場合は Welcome メッセージをログに記録します。
2. フロントエンドプラグインから受け取ったイベントをログに記録します。

org.my.group.AnalyticsManager と **org.my.group.MainConfiguration** は代替の Bean であるため、**src/main/resources/application.properties** の **quarkus.arc.selected-alternatives** プロパティーを使用して指定します。

例3.17 application.properties

```
quarkus.arc.selected-alternatives=MainConfiguration,AnalyticsManager
```

3.6.1.6.4. DevWorkspace 内でのアプリケーションの実行

1. DevWorkspace に **DEVWORKSPACE_TELEMETRY_BACKEND_PORT** 環境変数を設定します。ここで、値は **4167** に設定されます。

```
spec:
  template:
    attributes:
      workspaceEnv:
        - name: DEVWORKSPACE_TELEMETRY_BACKEND_PORT
          value: '4167'
```

2. Red Hat OpenShift DevSpaces ダッシュボードから DevWorkspace を再起動します。
3. DevWorkspace のターミナルウィンドウ内で以下のコマンドを実行し、アプリケーションを起動します。**--settings** フラグを使用して、GitHub アクセストークンが含まれる **settings.xml** ファイルの場所へのパスを指定します。

```
$ mvn --settings=settings.xml quarkus:dev -
Dquarkus.http.port=${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}
```

アプリケーションは、フロントエンドプラグインからポート **4167** を使用して Telemetry イベントを受け取るようになりました。

検証手順

1. 以下の出力がログに記録されていることを確認します。

```
INFO  [org.ecl.che.inc.AnalyticsManager] (Quarkus Main Thread) No welcome message
provided
```

```
INFO  [io.quarkus] (Quarkus Main Thread) devworkspace-telemetry-example-plugin 1.0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.2.Final) started in 0.323s. Listening on: http://localhost:4167
INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, kubernetes-client, rest-client, rest-client-jackson, resteasy, resteasy-jsonb, smallrye-context-propagation, smallrye-openapi, swagger-ui, vertx]
```

2. **AnalyticsManager** の **onEvent()** メソッドがフロントエンドプラグインからイベントを受信することを確認するには、**l** キーを押して Quarkus ライブコーディングを無効にし、IDE 内のファイルを編集します。以下の出力がログに記録されるはずです。

```
INFO  [io.qua.dep.dev.RuntimeUpdatesProcessor] (Aesh InputStream Reader) Live reload disabled
INFO  [org.ecl.che.inc.AnalyticsManager] (executor-thread-2) The received event is: Edit Workspace File in Che
```

3.6.1.6.5. isEnabled() の実装

この例では、このメソッドは呼び出されるたびに **true** を返します。

例3.18 AnalyticsManager.java

```
@Override
public boolean isEnabled() {
    return true;
}
```

より複雑なロジックを **isEnabled()** に設定することができます。たとえば、ホストされている OpenShift Dev Spaces Woopra バックエンド は、バックエンドが有効になっているかどうかを判断する前に、設定プロパティーが存在することを確認します。

3.6.1.6.6. onEvent() の実装

onEvent() は、バックエンドが受信したイベントを Telemetry システムに送信します。サンプルアプリケーションでは、HTTP POST ペイロードを Telemetry サーバーから **/event** エンドポイントに送信します。

3.6.1.6.6.1. サンプル Telemetry サーバーへの POST 要求の送信

以下の例では、Telemetry サーバーアプリケーションは **http://little-telemetry-server-che.apps-crc.testing** の URL で OpenShift にデプロイされます。ここで、 **apps-crc.testing** は OpenShift クラスターの Ingress ドメイン名です。

1. **TelemetryService.java** を作成して RESTEasy REST Client を設定します。

例3.19 TelemetryService.java

```
package org.my.group;

import java.util.Map;

import javax.ws.rs.Consumes;
```

```
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;

import org.eclipse.microprofile.rest.client.inject.RegisterRestClient;

@RegisterRestClient
public interface TelemetryService {
    @POST
    @Path("/event") (1)
    @Consumes(MediaType.APPLICATION_JSON)
    Response sendEvent(Map<String, Object> payload);
}
```

(1) **POST** リクエストを行うエンドポイント。

2. **src/main/resources/application.properties** ファイルで **TelemetryService** のベース URL を指定します。

例3.20 application.properties

```
org.my.group.TelemetryService/mp-rest/url=http://little-telemetry-server-che.apps-crc.testing
```

3. **TelemetryService** を **AnalyticsManager** に挿入し、**onEvent()** で **POST** リクエストを送信します

例3.21 AnalyticsManager.java

```
@Dependent
@Alternative
public class AnalyticsManager extends AbstractAnalyticsManager {
    @Inject
    @RestClient
    TelemetryService telemetryService;

    ...

    @Override
    public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent,
    String resolution, Map<String, Object> properties) {
        Map<String, Object> payload = new HashMap<String, Object>(properties);
        payload.put("event", event);
        telemetryService.sendEvent(payload);
    }
```

これにより、HTTP 要求が Telemetry サーバーに送信され、短期間同じイベントが自動的に遅延します。デフォルトの期間は 1500 ミリ秒です。

3.6.1.6.7. increaseDuration() の実装

多くの Telemetry システムはイベント期間を認識します。**AbstractAnalyticsManager** は、同じ期間内で発生する同様のイベントを 1 つのイベントにマージします。**increaseDuration()** のこの実装は no-op です。この方法では、ユーザーの Telemetry プロバイダーの API を使用してイベントまたはイベントプロパティーを変更し、イベントの延長期間を反映します。

例3.22 AnalyticsManager.java

```
@Override
public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) {}
```

3.6.1.6.8. onActivity() の実装

非アクティブなタイムアウトの制限を設定し、最後のイベント時間がタイムアウトよりも長くなる場合は、**onActivity()** を使用して **WORKSPACE_INACTIVE** イベントを送信します。

例3.23 AnalyticsManager.java

```
public class AnalyticsManager extends AbstractAnalyticsManager {

  ...

  private long inactiveTimeLimit = 60000 * 3;

  ...

  @Override
  public void onActivity() {
    if (System.currentTimeMillis() - lastEventTime >= inactiveTimeLimit) {
      onEvent(WORKSPACE_INACTIVE, lastOwnerId, lastIp, lastUserAgent, lastResolution,
commonProperties);
    }
  }
```

3.6.1.6.9. destroy() の実装

destroy() が呼び出される際に、**WORKSPACE_STOPPED** イベントを送信し、接続プールなどのリソースをシャットダウンします。

例3.24 AnalyticsManager.java

```
@Override
public void destroy() {
  onEvent(WORKSPACE_STOPPED, lastOwnerId, lastIp, lastUserAgent, lastResolution,
commonProperties);
}
```

「DevWorkspace 内でのアプリケーションの実行」 で説明されているように **mvnquarkus:dev** を実行する**Ctrl**+**C** を使用してアプリケーションを終了すると、**WORKSPACE_STOPPED** イベントがサーバーに送信されます。

3.6.1.6.10. Quarkus アプリケーションのパッケージ化

アプリケーションをコンテナーにパッケージ化する最適な方法については、Quarkus ドキュメント を参照してください。コンテナーをビルドし、選択したコンテナーレジストリーにプッシュします。

3.6.1.6.10.1. JVM で実行する Quarkus イメージをビルドするための Dockerfile の例

例3.25 Dockerfile.jvm

```
FROM registry.access.redhat.com/ubi8/openjdk-11:1.11

ENV LANG='en_US.UTF-8' LANGUAGE='en_US:en'

COPY --chown=185 target/quarkus-app/lib/ /deployments/lib/
COPY --chown=185 target/quarkus-app/*.jar /deployments/
COPY --chown=185 target/quarkus-app/app/ /deployments/app/
COPY --chown=185 target/quarkus-app/quarkus/ /deployments/quarkus/

EXPOSE 8080
USER 185

ENTRYPOINT ["java", "-Dquarkus.http.host=0.0.0.0", "-
Djava.util.logging.manager=org.jboss.logmanager.LogManager", "-
Dquarkus.http.port=${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}", "-jar",
"/deployments/quarkus-run.jar"]
```

イメージをビルドするには、以下を実行します。

```
mvn package && \
podman build -f src/main/docker/Dockerfile.jvm -t image:tag .
```

3.6.1.6.10.2. Quarkus ネイティブイメージをビルドするための Dockerfile の例

例3.26 Dockerfile.native

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.5
WORKDIR /work/
RUN chown 1001 /work \
    && chmod "g+rwX" /work \
    && chown 1001:root /work
COPY --chown=1001:root target/*-runner /work/application

EXPOSE 8080
USER 1001

CMD ["./application", "-Dquarkus.http.host=0.0.0.0", "-
Dquarkus.http.port=$DEVWORKSPACE_TELEMETRY_BACKEND_PORT}"]
```

イメージをビルドするには、以下を実行します。

```
mvn package -Pnative -Dquarkus.native.container-build=true && \
podman build -f src/main/docker/Dockerfile.native -t image:tag .
```

3.6.1.6.11. プラグインの plugin.yaml の作成

DevWorkspacePod でカスタムバックエンドを実行する **DevWorkspace** プラグインを表す plugin.yamldevfilev2 ファイルを作成します。devfile v2 の詳細については、Devfile v2 documentation を参照してください。

例3.27 plugin.yaml

```
schemaVersion: 2.1.0
metadata:
  name: devworkspace-telemetry-backend-plugin
  version: 0.0.1
  description: A Demo telemetry backend
  displayName: Devworkspace Telemetry Backend
components:
  - name: devworkspace-telemetry-backend-plugin
    attributes:
      workspaceEnv:
        - name: DEVWORKSPACE_TELEMETRY_BACKEND_PORT
          value: '4167'
    container:
      image: YOUR IMAGE          (1)
      env:
        - name: WELCOME_MESSAGE  (2)
          value: 'hello world!'
```

(1) 「Quarkus アプリケーションのパッケージ化」 からビルドされたコンテナーイメージを指定します。

(2) Example 4 から **welcome.message** オプションの設定プロパティーの値を設定します。

通常、ユーザーはこのファイルを企業 Web サーバーにデプロイします。本書では、OpenShift で Apache Web サーバーを作成し、そこでプラグインをホストする方法を説明します。

新規 **plugin.yaml** ファイルを参照する **ConfigMap** オブジェクトを作成します。

```
$ oc create configmap --from-file=plugin.yaml -n openshift-devspaces telemetry-plugin-yaml
```

Web サーバーを公開するためにデプロイメント、サービス、およびルートを作成します。デプロイメントはこの **ConfigMap** オブジェクトを参照し、これを **/var/www/html** ディレクトリーに配置します。

例3.28 manifest.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
spec:
  replicas: 1
```

```
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      volumes:
        - name: plugin-yaml
          configMap:
            name: telemetry-plugin-yaml
            defaultMode: 420
      containers:
        - name: apache
          image: 'registry.redhat.io/rhscl/httpd-24-rhel7:latest'
          ports:
            - containerPort: 8080
              protocol: TCP
          resources: {}
          volumeMounts:
            - name: plugin-yaml
              mountPath: /var/www/html
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
---
kind: Service
apiVersion: v1
metadata:
  name: apache
spec:
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  selector:
    app: apache
  type: ClusterIP
---
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: apache
spec:
  host: apache-che.apps-crc.testing
  to:
    kind: Service
    name: apache
    weight: 100
```

```
    port:
      targetPort: 8080
    wildcardPolicy: None
```

```
$ oc apply -f manifest.yaml
```

検証手順

デプロイメントが開始されたら、Web サーバーで **plugin.yaml** が利用できることを確認します。

```
$ curl apache-che.apps-crc.testing/plugin.yaml
```

3.6.1.6.12. DevWorkspace での Telemetry プラグインの指定

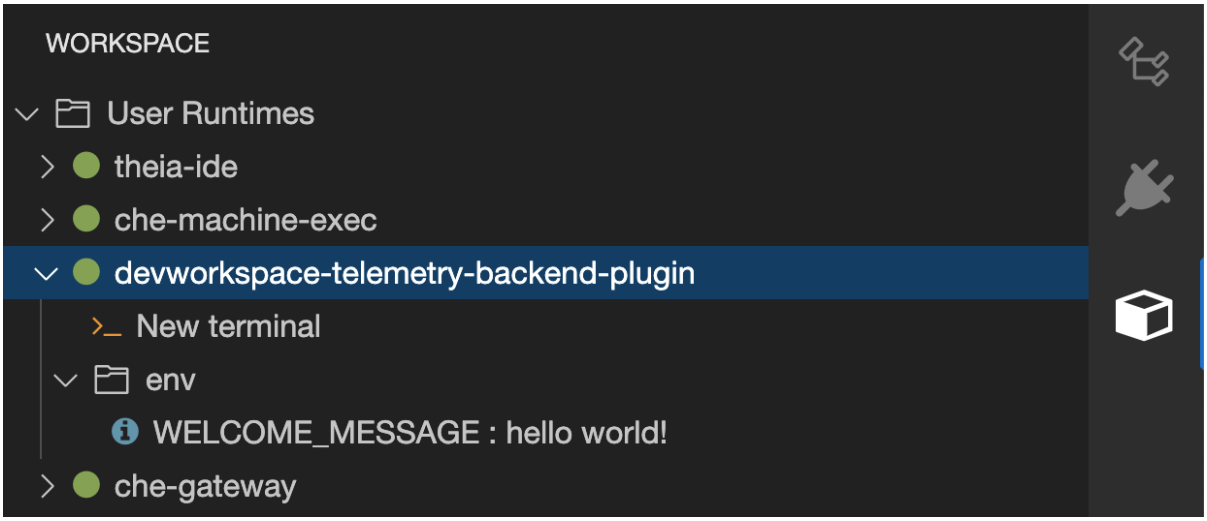
1. 以下を既存の DevWorkspace の **components** フィールドに追加します。

```
components:
  ...
  - name: telemetry-plug-in
    plugin:
      uri: http://apache-che.apps-crc.testing/plugin.yaml
```

2. OpenShift DevSpaces ダッシュボードから DevWorkspace を起動します。

検証手順

1. **Telemetry-plug-in** コンテナーが DevWorkspace Pod で稼働していることを確認します。ここでは、これはエディターで Workspace ビューをチェックして検証されます。



2. エディター内のファイルを編集し、Telemetry サーバーのログのサンプルでイベントを確認します。

3.6.1.6.13. すべての DevWorkspaces の Telemetry プラグインの適用

テレメトリープラグインをデフォルトのプラグインとして設定します。デフォルトのプラグインは、新規および既存の DevWorkspaces の DevWorkspace 起動時に適用されます。

- CheCluster カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソース

- **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  devEnvironments:
    defaultPlugins:
    - editor: eclipse/che-theia/next    (1)
      plugins:                          (2)
      - 'http://apache-che.apps-crc.testing/plugin.yaml'
```

(1) デフォルトのプラグインを設定するための editorId。

(2) devfile v2 プラグインへの URL の一覧。

関連情報

- 「CLI を使用して CheCluster カスタムリソースの設定」

検証手順

1. Red Hat OpenShift DevSpaces ダッシュボードから新規または既存の DevWorkspace を起動します。
2. 「DevWorkspace での Telemetry プラグインの指定」 の検証手順に従って、Telemetry プラグインが機能していることを確認します。

3.6.2. サーバーロギングの設定

OpenShift Dev Spaces サーバーで利用可能な個別のロガーのログレベルを微調整できます。

OpenShift Dev Spaces サーバー全体のログレベルは、Operator の **cheLogLevel** 設定プロパティーを使用してグローバルに設定されます。「**CheCluster** カスタムリソースフィールドの参照」を参照してください。Operator によって管理されないインストールでグローバルログレベルを設定するには、**che** ConfigMap で **CHE_LOG_LEVEL** 環境変数を指定します。

CHE_LOGGER_CONFIG 環境変数を使用して、OpenShift Dev Spaces サーバーの個々のロガーのログレベルを設定することができます。

3.6.2.1. ログレベルの設定

手順

- **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LOGGER_CONFIG: "<key1=value1,key2=value2>"    (1)
```

(1) キーと値のペアのコンマ区切りリスト。キーは OpenShift Dev Spaces サーバーログ出力に表示されるロガーの名前で、値は必要なログレベルになります。

例3.29 WorkspaceManager のデバッグモードの設定

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LOGGER_CONFIG:
"org.eclipse.che.api.workspace.server.WorkspaceManager=DEBUG"
```

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.6.2.2. ロガーの命名

ロガーの名前は、それらのロガーを使用する内部サーバークラスのクラス名に従います。

3.6.2.3. HTTP トラフィックのロギング

手順

- OpenShift Dev Spaces サーバーと Kubernetes または OpenShift クラスターの API サーバー間の HTTP トラフィックをログに記録するには、**CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_LOGGER_CONFIG: "che.infra.request-logging=TRACE"
```

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.6.3. dsc を使用したログの収集

Red Hat OpenShift Dev Spaces のインストールは、OpenShift クラスターで実行されている複数のコンテナーで設定されます。実行中の各コンテナーからログを手動で収集できますが、**dsc** はプロセスを自動化するコマンドを提供します。

以下のコマンドを使用すると、**dsc** ツールを使用して OpenShift クラスターから Red Hat OpenShift Dev Spaces ログを収集します。

dsc server:logs

既存の Red Hat OpenShift Dev Spaces サーバーログを収集し、ローカルマシンのディレクトリーに保存します。デフォルトでは、ログはマシンの一時ディレクトリーにダウンロードされます。ただし、**-d** パラメーターを指定すると上書きできます。たとえば、Che ログを **/home/user/che-logs/**

ディレクトリーにダウンロードするには、以下のコマンドを使用します。

```
dsc server:logs -d /home/user/che-logs/
```

実行すると、**dsc server:logs** はログファイルを保存するディレクトリーを指定するコンソールにメッセージを出力します。

```
Red Hat OpenShift Dev Spaces logs will be available in '/tmp/chectl-logs/1648575098344'
```

Red Hat OpenShift Dev Spaces がデフォルト以外のプロジェクトにインストールされている場合、**dsc server:logs** には **-n <NAMESPACE>** パラメーターが必要です。ここで、**<NAMESPACE>** は Red Hat OpenShift Dev Spaces がインストールされた OpenShift プロジェクトです。たとえば、**my-namespace** プロジェクトの OpenShift Dev Spaces からログを取得するには、以下のコマンドを使用します。

```
dsc server:logs -n my-namespace
```

dsc server:deploy

ログは、**dsc** を使用してインストール時に OpenShift Dev Spaces のインストール時に自動的に収集されます。**dsc server:logs** と同様に、ディレクトリーのログは **-d** パラメーターを使用して指定できます。

関連情報

- DSC リファレンスドキュメント

3.6.4. Prometheus と Grafana を使用した OpenShift Dev Spaces のモニターリング

クラスター上で実行中の Prometheus および Grafana のインスタンスを使用して、OpenShift Dev Spaces メトリクスを収集および表示できます。

- 「Prometheus と Grafana のインストール」
- 「DevWorkspace Operator のモニタリング」
- 「OpenShift Dev Spaces サーバーのモニタリング」

3.6.4.1. Prometheus と Grafana のインストール

template.yaml を適用して Prometheus および Grafana をインストールできます。この例の **template.yaml** ファイルは、Prometheus および Grafana を使い始めるための基本的な設定、Deployments および Services のモニタリングスタックを提供します。

または、Prometheus Operator と Grafana Operator を使用することもできます。

前提条件

- oc

手順

template.yaml を使用して Prometheus と Grafana をインストールするには、以下を実行します。

1. Prometheus および Grafana の新規プロジェクト **monitoring** を作成します。

```
$ oc new-project monitoring
```

2. **monitoring** プロジェクトで **template.yaml** を適用します。

```
$ oc apply -f template.yaml -n monitoring
```

例3.30 template.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  name: grafana
  labels:
    app: grafana
spec:
  ports:
  - name: 3000-tcp
    port: 3000
    protocol: TCP
    targetPort: 3000
  selector:
    app: grafana
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  labels:
    app: prometheus
spec:
  ports:
  - name: 9090-tcp
    port: 9090
    protocol: TCP
    targetPort: 9090
  selector:
    app: prometheus
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: grafana
  name: grafana
spec:
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
```

```
      - image: registry.redhat.io/rhel8/grafana:7
        name: grafana
        ports:
        - containerPort: 3000
          protocol: TCP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus
  name: prometheus
spec:
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      serviceAccountName: prometheus
      containers:
      - image: quay.io/prometheus/prometheus:v2.36.0
        name: prometheus
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
        - mountPath: /prometheus
          name: volume-data
        - mountPath: /etc/prometheus/prometheus.yml
          name: volume-config
          subPath: prometheus.yml
      volumes:
      - emptyDir: {}
        name: volume-data
      - configMap:
          defaultMode: 420
          name: prometheus-config
        name: volume-config
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: ""
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
---
```

関連情報

- First steps with Prometheus
- Installing Grafana

3.6.4.2. DevWorkspace Operator のモニタリング

DevWorkspace Operator が公開するメトリクスを処理するために、モニタリングスタックの例を設定できます。

3.6.4.2.1. Prometheus による DevWorkspace Operator メトリクスの収集

Prometheus を使用して、DevWorkspace Operator に関するメトリクスを収集、保存、および照会するには、以下を実行します。

前提条件

- **devworkspace-controller-metrics** サービスは、ポート **8443** でメトリクスを公開している。これはデフォルトで事前設定されています。
- **devworkspace-webhookserver** サービスは、ポート **9443** でメトリクスを公開している。これはデフォルトで事前設定されています。
- Prometheus 2.26.0 以降が動作している。Prometheus コンソールは、ポート **9090** で実行されており、対応するサービスがあります。Prometheus を初めて実行するための手順 について参照してください。

手順

1. ClusterRoleBinding を作成して、Prometheus に関連付けられた ServiceAccount を devworkspace-controller-metrics-reader ClusterRole にバインドします。 モニターリングスタックのサンプル では、使用される ServiceAccount の名前は **prometheus** です。

注記

DevWorkspace メトリクスへのアクセスはロールベースアクセスコントロール (RBAC) で保護されているため、ClusterRoleBinding がないとアクセスできません。

例3.31 clusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: devworkspace-controller-metrics-binding
subjects:
  - kind: ServiceAccount
    name: prometheus
    namespace: monitoring
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: devworkspace-controller-metrics-reader
```

2. Prometheus は、**devworkspace-controller-metrics** サービスが公開するポート **8443** と、**devworkspace-webhookserver** サービスが公開するポート **9443** からメトリクスを収集するように設定します。

注記

モニターリングスタックのサンプル では、空の設定で **prometheus-config** ConfigMap がすでに作成されています。Prometheus 設定の詳細を指定するには、ConfigMap の **data** フィールドを編集します。

例3.32 Prometheus の設定

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.yml: |-
    global:
      scrape_interval: 5s (1)
      evaluation_interval: 5s (2)
    scrape_configs: (3)
      - job_name: 'DevWorkspace'
        scheme: https
        authorization:
          type: Bearer
          credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
        tls_config:
          insecure_skip_verify: true
        static_configs:
          - targets: ['devworkspace-controller-metrics.<DWO_project>:8443'] (4)
      - job_name: 'DevWorkspace webhooks'
        scheme: https
        authorization:
          type: Bearer
          credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
        tls_config:
          insecure_skip_verify: true
        static_configs:
          - targets: ['devworkspace-webhookserver.<DWO_project>:9443'] (5)
```

(1) ターゲットが収集されるレート。

(2) 記録およびアラートルールを再チェックするレート。

(3) Prometheus が監視するリソースデフォルトの設定では、2 つのジョブ (**DevWorkspace** および **DevWorkspace webhooks**) が、**devworkspace-controller-metrics** サービスおよび **devworkspace-webhookserver** サービスによって公開された時系列データを収集します。

(4) ポート **8443** からのメトリクスのスクレイプターゲット。**<DWO_project>**

> 5 ポート **9443** からのメトリクスのスクレイプターゲット。**<DWO_project>** を、**devworkspace-webhookserver Service** が置かれているプロジェクトに置き換え

3. **Prometheus** Deployment をスケールダウンおよびスケールアップし、直前の手順で更新された ConfigMap を読み取ります。

```
$ oc scale --replicas=0 deployment/prometheus -n monitoring && oc scale --replicas=1 deployment/prometheus -n monitoring
```

検証

1. ポート転送を使用して、ローカルで **Prometheus** サービスにアクセスします。

```
$ oc port-forward svc/prometheus 9090:9090 -n monitoring
```

2. **localhost:9090/targets** でターゲットエンドポイントを表示して、すべてのターゲットが稼働していることを確認します。
3. Prometheus コンソールを使用して、メトリクスを表示および照会します。
   - **localhost:9090/metrics** でメトリクスを表示します。
   - **localhost:9090/graph** からメトリクスをクエリーします。詳細は、Using the expression browser を参照してください。

関連情報

- Prometheus の設定
- Querying Prometheus
- Prometheus metric types

3.6.4.2.2. DevWorkspace 固有のメトリクス

次の表は、**devworkspace-controller-metrics** サービスによって公開される DevWorkspace 固有のメトリックについて説明しています。

表3.20 メトリクス

名前	タイプ	説明	ラベル
devworkspace_started_total	カウンター	DevWorkspace の開始イベントの数。	**source**、**routingclass**
devworkspace_started_success_total	カウンター	**Running** 段階に移行した DevWorkspaces の数。	**source**、**routingclass**
devworkspace_fail_total	カウンター	失敗した DevWorkspaces の数。	**source**、**reason**

名前	タイプ	説明	ラベル
devworkspace_startup_time	ヒストグラム	DevWorkspace の起動にかかった総時間 (秒)。	**source**、**routingclass**

表3.21 ラベル

名前	説明	値
source	DevWorkspace の **controller.devfile.io/devworkspace-source** ラベルです。	**string**
routingclass	DevWorkspace の **spec.routingclass**。	**"basic|cluster|cluster-tls|web-terminal"**
reason	ワークスペースの起動失敗の理由です。	**"BadRequest|InfrastructureFailure|Unknown"**

表3.22 スタートアップ失敗の理由

名前	説明
BadRequest	DevWorkspace の作成に使用された devfile が無効であるため、起動に失敗しました。
InfrastructureFailure	**CreateContainerError**、**RunContainerError**、**FailedScheduling**、**FailedMount** のエラーによる起動の失敗。
Unknown	不明な失敗理由。

3.6.4.2.3. Grafana ダッシュボードでの DevWorkspace Operator メトリクスの表示

ダッシュボードの例を使用して Grafana の DevWorkspace Operator メトリクスを表示するには、以下を実行します。

前提条件

- Prometheus はメトリクスを収集している。「Prometheus による DevWorkspace Operator メトリクスの収集」を参照してください。
- Grafana バージョン 7.5.3 以降。
- Grafana はポート **3000** で実行されており、対応するサービスがあります。Installing Grafana を参照してください。

手順

1. Prometheus インスタンスのデータソースを追加します。Creating a Prometheus data source を参照してください。
2. example **grafana-dashboard.json** ダッシュボードをインポートします。

検証手順

- Grafana コンソールを使用して、DevWorkspace Operator メトリクスダッシュボードを表示します。「DevWorkspace Operator の Grafana ダッシュボード」を参照してください。

関連情報

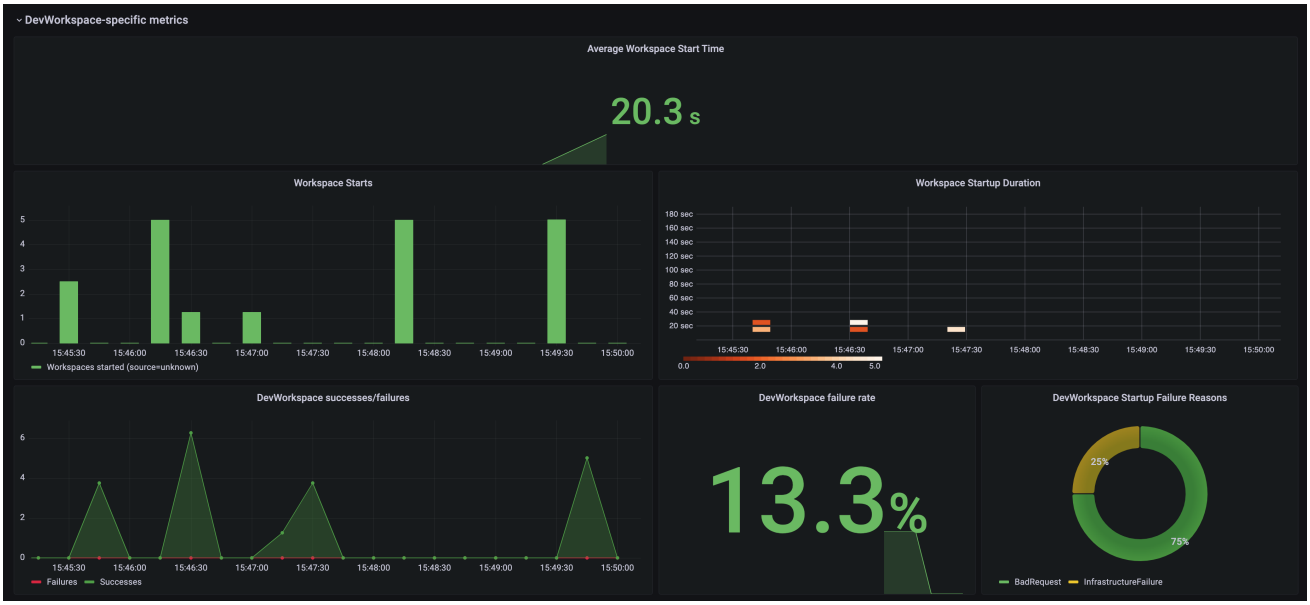
- Prometheus data source
- Import dashboard

3.6.4.2.4. DevWorkspace Operator の Grafana ダッシュボード

grafana-dashboard.json に基づく サンプルの Grafana ダッシュボードには、DevWorkspace Operator から次のメトリクスが表示されます。

3.6.4.2.4.1. DevWorkspace-specific metrics パネル

図3.1 DevWorkspace-specific metrics パネル



ワークスペースの平均起動時間

ワークスペースの平均起動時間。

ワークスペースの起動

ワークスペースの起動の成功と失敗の回数。

ワークスペースの起動時間

ワークスペースの起動時間を表示するヒートマップ。

DevWorkspace の成功/失敗

DevWorkspace の起動の成功と失敗の比較。

DevWorkspace の失敗率

ワークスペースの起動失敗回数と総起動回数の比率。

DevWorkspace 起動失敗の理由

ワークスペース起動失敗の分布を表示する円グラフ:

- **BadRequest**
- **InfrastructureFailure**
- **Unknown**

3.6.4.2.4.2. Operator metrics パネル (パート 1)

図3.2 Operator metrics パネル (パート 1)

進行中の Webhook

さまざまな Webhook リクエストの数の比較。

作業キューの期間

調整リクエストが処理される前にワークキューにとどまる時間を表示するヒートマップ。

Webhook のレイテンシー (/mutate)

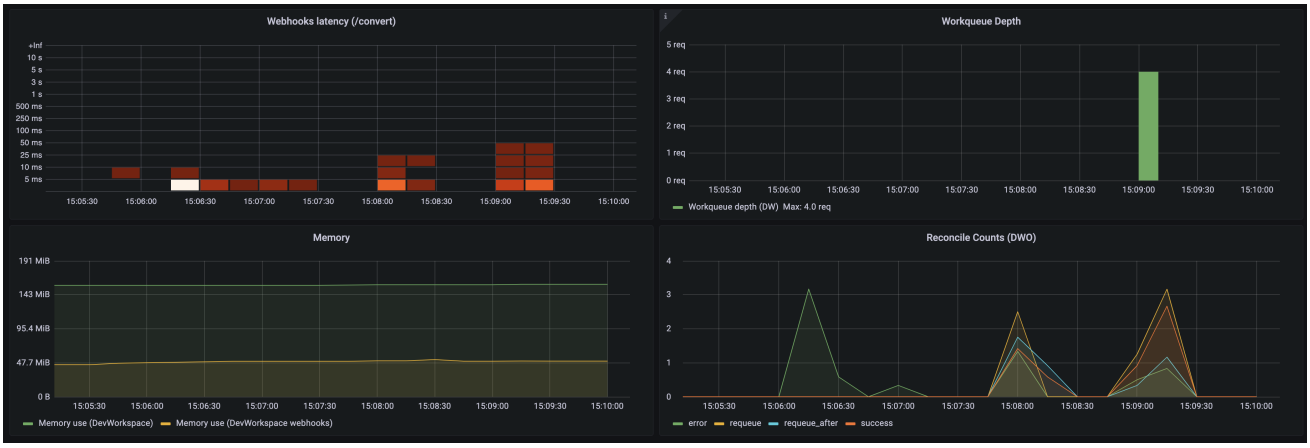
/**mutate** Webhook レイテンシーを表示するヒートマップ。

調整時間

調整期間を表示するヒートマップ。

3.6.4.2.4.3. Operator metrics パネル (パート 2)

図3.3 Operator metrics パネル (パート 2)



Webhook のレイテンシー (/convert)

/convert Webhook レイテンシーを表示するヒートマップ。

作業キューの深さ

作業キューにある調整リクエストの数。

メモリー

DevWorkspace コントローラーおよび DevWorkspace Webhook サーバーのメモリー使用量。

調整数 (DWO)

DevWorkspace コントローラーの 1 秒あたりの平均調整回数。

3.6.4.3. OpenShift Dev Spaces サーバーのモニタリング

OpenShift Dev Spaces サーバーの JVM メモリーやクラ出力ディングなどの JVM メトリクスを公開するように OpenShift Dev Spaces を設定できます。

3.6.4.3.1. OpenShift Dev Spaces サーバーメトリクスの有効化と公開

OpenShift Dev Spaces は、**che-host** サービスのポート **8087** で JVM メトリクスを公開します。この動作を設定できます。

手順

- **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    metrics:
      enable: <boolean> 1
```

1 有効にするには **true**、無効にするには **false**。

3.6.4.3.2. Prometheus を使用した OpenShift Dev Spaces メトリクスの収集

Prometheus を使用して、OpenShift Dev Spaces サーバーの JVM メトリクスを収集、保存、および照会するには、以下を実行します。

前提条件

- OpenShift Dev Spaces は、ポート **8087** にメトリクスを公開しています。Enabling and exposing OpenShift Dev Spaces server JVM metrics を参照してください。
- Prometheus 2.26.0 以降が動作している。Prometheus コンソールは、ポート **9090** で実行されており、対応するサービスがあります。Prometheus を初めて実行するための手順 について参照してください。

手順

1. ポート **8087** からメトリクスを収集するように Prometheus を設定します。

注記

モニターリングスタックのサンプル では、空の設定で **prometheus-config** ConfigMap がすでに作成されています。Prometheus 設定の詳細を指定するには、ConfigMap の **data** フィールドを編集します。

例3.33 Prometheus の設定

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |-
    global:
      scrape_interval:     5s          1
      evaluation_interval: 5s          2
    scrape_configs:                    3
      - job_name: 'Che Server'
        static_configs:
          - targets: ['che-host.<OpenShift Dev Spaces_project>:8087']  4
```

1. ターゲットが収集されるレート。
2. 記録およびアラートルールを再チェックするレート。
3. Prometheus が監視するリソースデフォルト設定では、単一のジョブ **Che Server** が、OpenShift Dev Spaces Server によって公開された時系列データをスクレイピングします。
4. ポート **8087** からのメトリクスのスクレイプターゲット。**<OpenShift Dev Spaces_project>** を OpenShift Dev Spaces プロジェクトに置き換えます。デフォルトの OpenShift Dev Spaces プロジェクトは **openshift-devspaces** です。

2. **Prometheus** Deployment をスケールダウンおよびスケールアップし、直前の手順で更新された ConfigMap を読み取ります。

```
$ oc scale --replicas=0 deployment/prometheus -n monitoring && oc scale --replicas=1 deployment/prometheus -n monitoring
```

検証

1. ポート転送を使用して、ローカルで **Prometheus** サービスにアクセスします。

```
$ oc port-forward svc/prometheus 9090:9090 -n monitoring
```

2. **localhost:9090/targets** で **targets** エンドポイントを表示して、すべてのターゲットが稼働していることを確認します。
3. Prometheus コンソールを使用して、メトリクスを表示および照会します。
   - **localhost:9090/metrics** でメトリクスを表示します。

- **localhost:9090/graph** からメトリクスをクエリーします。
  詳細は、Using the expression browser を参照してください。

関連情報

- Prometheus の設定
- Querying Prometheus
- Prometheus metric types

3.6.4.3.3. Grafana ダッシュボードでの OpenShift Dev Spaces サーバーメトリクスの表示

Grafana で OpenShift Dev Spaces サーバーメトリクスを表示するには、以下を実行します。

前提条件

- Prometheus は、OpenShift Dev Spaces クラスターでメトリクスを収集している。「Prometheus と Grafana を使用した OpenShift Dev Spaces のモニターリング」 を参照してください。
- Grafana 6.0 以降がポート **3000** で実行されており、対応するサービスがあります。Installing Grafana を参照してください。

手順

1. Prometheus インスタンスのデータソースを追加します。Creating a Prometheus data source を参照してください。
2. サンプル ダッシュボード をインポートします。Import dashboard を参照してください。
3. Grafana コンソールで OpenShift Dev Spaces メトリクスを表示します。

図3.4 OpenShift Dev Spaces サーバーの JVM ダッシュボード

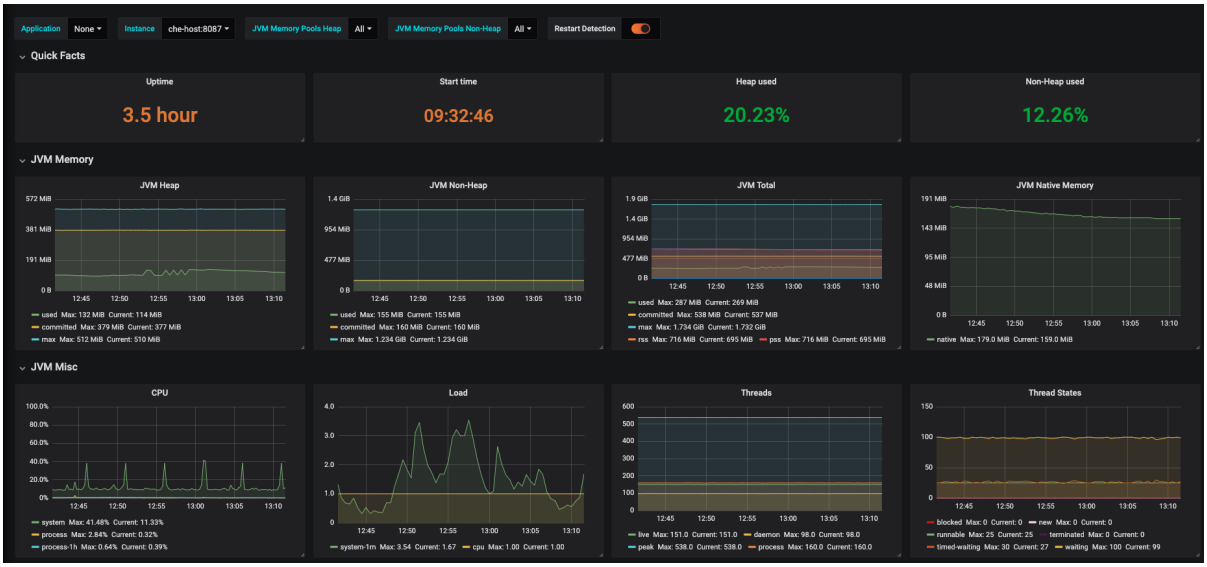


図3.5 クイックファクト

図3.6 JVM メモリー

図3.7 JVM Misc

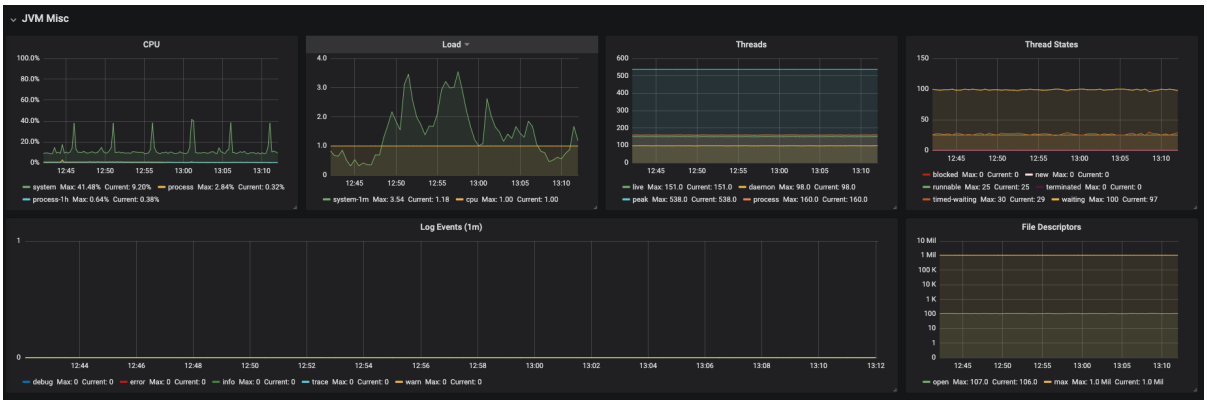


図3.8 JVM メモリープール (ヒープ)

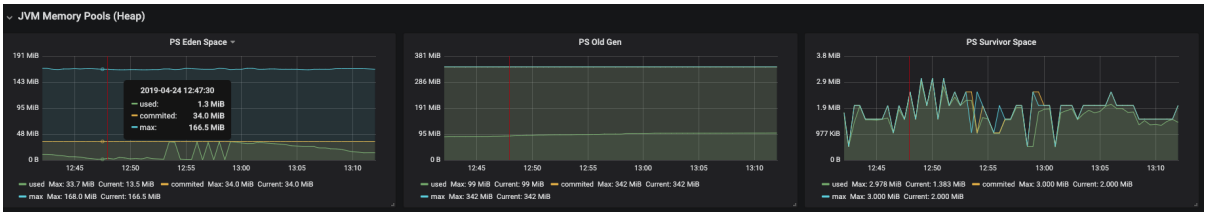


図3.9 JVM メモリープール (非ヒープ)

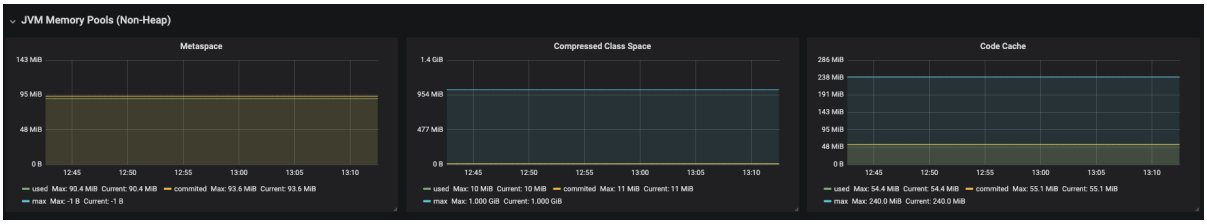


図3.10 ガベージコレクション

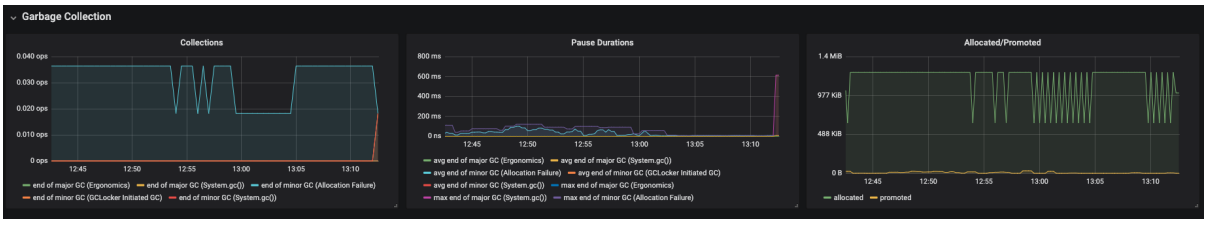


図3.11 クラ出力ディング

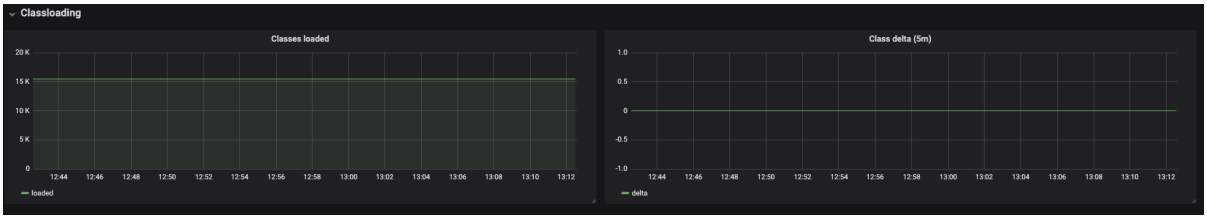


図3.12 バッファープール

3.7. ネットワークの設定

- 「ネットワークポリシーの設定」
- 「Red Hat OpenShift Dev Spaces サーバーのホスト名の設定」
- 「信頼できない TLS 証明書の OpenShift Dev Space へのインポート」
- 「ラベルおよびアノテーションの OpenShift ルートへの追加」
- 「ルーターのシャード化と連携するように OpenShift ルートを設定」

3.7.1. ネットワークポリシーの設定

デフォルトでは、OpenShift クラスター内のすべての Pod は、異なる名前空間にある場合でも相互に通信できます。OpenShift Dev Spaces のコンテキストでは、これにより、あるユーザープロジェクトのワークスペース Pod が別のユーザープロジェクトの別のワークスペース Pod にトラフィックを送信できるようになります。

セキュリティーのために、NetworkPolicy オブジェクトを使用してマルチテナント分離を設定し、すべての着信通信をユーザープロジェクト内の Pod に制限することができます。ただし、OpenShift Dev Spaces プロジェクトの Pod は、ユーザープロジェクトの Pod と通信できる必要があります。

前提条件

- OpenShift クラスターには、マルチテナント分離などのネットワーク制限があります。

手順

- **allow-from-openshift-devspaces** NetworkPolicy を各ユーザープロジェクトに適用します。**allow-from-openshift-devspaces** NetworkPolicy は、OpenShift Dev Spaces 名前空間からユーザープロジェクト内のすべての Pod への受信トラフィックを許可します。

例3.34 allow-from-openshift-devspaces.yaml

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-openshift-devspaces
spec:
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          kubernetes.io/metadata.name: openshift-devspaces  (1)
```

```
  podSelector: {}
  policyTypes:
  - Ingress
```

1. OpenShift Dev Spaces 名前空間。デフォルトは **openshift-devspaces** です。
2. 空の **podSelector** は、プロジェクト内のすべての Pod を選択します。

関連情報

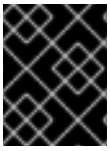
- 「ユーザープロジェクトプロビジョニングの設定」
- ネットワーク分離
- ネットワークポリシーを使用したマルチテナント分離の設定

3.7.2. Red Hat OpenShift Dev Spaces サーバーのホスト名の設定

この手順では、カスタムホスト名を使用するように OpenShift Dev Space を設定する方法を説明します。

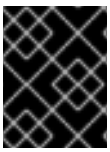
前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。
- 証明書とプライベートキーファイルが生成されます。



重要

秘密鍵と証明書のペアを生成するには、他の OpenShift Dev Spaces ホストと同じ認証局 (CA) を使用する必要があります。



重要

DNS プロバイダーに対し、カスタムホスト名をクラスター Ingress を参照するよう要求します。

手順

1. OpenShift Dev Spaces のプロジェクトを作成します。

```
$ oc create project openshift-devspaces
```

2. TLS Secret を作成します。

```
$ oc create secret TLS <tls-secret-name> \
--key <key-file> \
--cert <cert-file> \
-n openshift-devspaces
```

1. TLS Secret 名
2. プライベートキーを含むファイル
3. 証明書を含むファイル

3. シークレットに必要なラベルを追加します。

```
$ oc label secret <tls-secret-name> \ 1
app.kubernetes.io/part-of=che.eclipse.org -n openshift-devspaces
```

1. TLS Secret 名

4. **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  networking:
    hostname: <hostname> 1
    tlsSecretName: <secret> 2
```

1. カスタム Red Hat OpenShift Dev Spaces サーバーのホスト名
2. TLS Secret 名

5. OpenShift Dev Spaces がすでにデプロイされている場合は、すべての OpenShift Dev Spaces コンポーネントのロールアウトが完了するまで待ちます。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

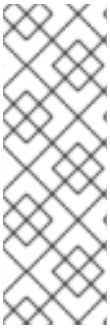
3.7.3. 信頼できない TLS 証明書の OpenShift Dev Space へのインポート

デフォルトでは、OpenShift Dev Spaces コンポーネント間の外部通信は TLS で暗号化されます。プロキシー、ソースコードリポジトリー、ID プロバイダーなどの外部サービスとの OpenShift Dev Spaces コンポーネントの通信にも、TLS が必要になる場合があります。TLS で暗号化されたすべての通信では、信頼できる認証局 (CA) によって署名された TLS 証明書を使用する必要があります。

OpenShift Dev Spaces コンポーネントまたは外部サービスが使用する証明書が信頼できない CA によって署名されている場合、CA 証明書を OpenShift Dev Spaces インスタンスにインポートして、すべての OpenShift Dev Spaces コンポーネントが証明書を信頼された CA によって署名されたものとして扱うようにする必要があります。次の場合にこれを行う必要があります。

- 基盤となる OpenShift クラスターが、信頼されていない CA によって署名された TLS 証明書を使用している場合。OpenShift Dev Spaces サーバーまたはワークスペースのコンポーネントが、信頼できない CA によって署名された TLS 証明書を使用する外部 OIDC プロバイダーまたは Git サーバーに接続している場合。

OpenShift Dev Spaces は、プロジェクトのラベルが付いた **ConfigMap** オブジェクトを TLS 証明書のソースとして使用します。**ConfigMap** オブジェクトには、それぞれ証明書数の乱数を持つ任意の数のキーを指定できます。



注記

OpenShift クラスターにクラスター全体の信頼できる CA 証明書が クラスター全体のプロキシー設定 を通じて追加されている場合、OpenShift DevSpaces Operator はそれらを検出し、この **ConfigMap** オブジェクトに自動的に挿入します。OpenShift Dev Spaces は、 **ConfigMap** オブジェクトに **config.openshift.io/inject-trusted-cabundle="true"** ラベルを自動的に付けます。このアノテーションに基づいて、OpenShift は **ConfigMap** オブジェクトの **ca-bundle.crt** キー内にクラスター全体で信頼される CA 証明書を自動的に挿入します。

重要

一部の OpenShift Dev Spaces コンポーネントでは、エンドポイントを信頼するために完全な証明書チェーンが必要です。クラスターが中間証明書で設定されている場合は、自己署名ルートを含むチェーン全体を OpenShift Dev Spaces に追加します。

3.7.3.1. OpenShift Dev Spaces への新しい CA 証明書の追加

通常、証明書ファイルは Base64 ファイルとして保存され、エクステンションは **.pem**、**.crt**、**.ca-bundle** などになります。証明書ファイルを保持するすべてのシークレットでは、バイナリーでエンコードされる証明書ではなく、Base64 でエンコードされた証明書を使用する必要があります。次の手順は、すでにインストールおよび実行されているインスタンスと、インストールされるインスタンスに適用されます。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法 を参照してください。
- OpenShift Dev Space の namespace が存在します。
- OpenShift Dev Spaces は、すでに予約済みファイル名を使用して証明書を **ConfigMap** オブジェクトに自動的に挿入します。証明書の保存には、以下の予約済みファイル名を使用しないでください。
  - **ca-bundle.crt**
  - **ca.crt**

手順

1. インポートする必要のある証明書をローカルファイルシステムに保存します。

注意

- 導入句が **BEGIN TRUSTED CERTIFICATE** の証明書は、Java でサポートされていない PEM **TRUSTED CERTIFICATE** 形式である可能性があります。次のコマンドを使用して、サポートされている **CERTIFICATE** 形式に変換します。
  - **openssl x509 -in cert.pem -out cert.cer**

2. 必要な TLS 証明書で新規 **ConfigMap** オブジェクトを作成します。

```
$ oc create configmap custom-certs --from-file=<bundle-file-path> -n=openshift-devspaces
```

複数のバンドルを適用するには、別の **-from-file=<bundle_file_path>** を追加します。または、別の **ConfigMap** オブジェクトを作成します。

3. 作成した **ConfigMap** オブジェクトに **app.kubernetes.io/part-of=che.eclipse.org** および **app.kubernetes.io/component=ca-bundle** ラベルを付けます。

```
$ oc label configmap custom-certs app.kubernetes.io/part-of=che.eclipse.org
app.kubernetes.io/component=ca-bundle -n openshift-devspaces
```

4. 以前にデプロイされていない場合は、OpenShift Dev Spaces をデプロイします。それ以外の場合は、OpenShift Dev Spaces コンポーネントのロールアウトが完了するまで待ちます。

5. 変更を有効にするには、実行中のワークスペースを再起動します。

3.7.3.2. インポートされた証明書に関する問題のトラブルシューティング

証明書を追加した後に問題が発生した場合は、OpenShift Dev Spaces のインスタンスレベルおよびワークスペースレベルで指定された値を確認します。

OpenShift Dev Spaces インスタンスレベルでのインポート済み証明書の検証

- OpenShift Dev Spaces Operator デプロイメントの場合、**CheCluster** が配置されている namespace には、正しいコンテンツを含む、ラベル付きの **ConfigMap** オブジェクトが含まれます。

```
$ oc get cm --selector=app.kubernetes.io/component=ca-bundle,app.kubernetes.io/part-
of=che.eclipse.org -n openshift-devspaces
```

以下を入力して **ConfigMap** オブジェクトの内容を確認します。

```
$ oc get cm <name> -n openshift-devspaces -o yaml
```

- OpenShift Dev Spaces Pod ボリューム一覧には、**ca-certs-merged ConfigMap** オブジェクトをデータソースとして使用するボリュームが含まれます。OpenShift Dev Spaces Pod のボリュームのリストを取得するには:

```
$ oc get pod -l app.kubernetes.io/component=devspaces -o "jsonpath=
{.items[0].spec.volumes}" -n openshift-devspaces
```

- OpenShift Dev Spaces は、証明書を OpenShift Dev Spaces サーバーコンテナーの **/public-certs/** フォルダーにマウントします。このフォルダー内のファイルのリストを表示するには、次のように入力します。

```
$ oc exec -t deploy/devspaces -n openshift-devspaces -- ls /public-certs/
```

- OpenShift Dev Spaces サーバーログに、設定された OpenShift Dev Spaces 証明書など、Java トラストストアに追加されたすべての証明書に対する行があります。次の表示:

```
$ oc logs deploy/devspaces -n openshift-devspaces
```

- OpenShift Dev Spaces サーバーの Java トラストストアに証明書が含まれます。証明書の SHA1 フィンガープリントは、トラストストアに含まれる証明書の SHA1 リストにあります。リストを表示します。

```
$ oc exec -t deploy/devspaces -n openshift-devspaces -- keytool -list -keystore
/home/user/cacerts
Your keystore contains 141 entries:
+
(...)
```

ローカルファイルシステムの証明書の SHA1 ハッシュを取得するには、以下のコマンドを実行します。

```
$ openssl x509 -in <certificate-file-path> -fingerprint -noout
SHA1 Fingerprint=3F:DA:BF:E7:A7:A7:90:62:CA:CF:C7:55:0E:1D:7D:05:16:7D:45:60
```

ワークスペースレベルでのインポート済み証明書の検証

- ワークスペースを起動し、これが作成されたプロジェクト名を取得し、ワークスペースが開始するまで待機します。
- ワークスペース Pod の名前を取得します。

```
$ oc get pods -o=jsonpath='{.items[0].metadata.name}' -n <workspace namespace> | grep
'^workspace.*'
```

- ワークスペース Pod 内の Che-Theia IDE コンテナーの名前を取得します。

```
$ oc get -o json pod <workspace pod name>  -n <workspace namespace> | \
    jq -r '.spec.containers[] | select(.name | startswith("theia-ide")).name'
```

- ワークスペース namespace 内で **che-trusted-ca-certs ConfigMap** オブジェクトを検索します。

```
$ oc get cm che-trusted-ca-certs -n <workspace namespace>
```

- **che-trusted-ca-certs ConfigMap** オブジェクトのエントリーに、以前に追加した別のエントリーがすべて含まれていることを確認します。さらに、予約されている **ca-bundle.crt** エントリーが含まれる場合があります。エントリーを表示します。

```
$ oc get cm che-trusted-ca-certs -n <workspace namespace> -o json | jq -r '.data | keys[]'
ca-bundle.crt
source-config-map-name.data-key.crt
```

- **che-trusted-ca-certs ConfigMap** オブジェクトがワークスペース Pod のボリュームとして追加されていることを確認します。

```
$ oc get -o json pod <workspace pod name> -n <workspace namespace> | \
    jq '.spec.volumes[] | select(.configMap.name == "che-trusted-ca-certs")'
{
  "configMap": {
    "defaultMode": 420,
    "name": "ca-certs"
```

```
  },
  "name": "che-self-signed-certs"
}
```

- ボリュームがコンテナー (とくに Che-Theia IDE コンテナー) にマウントされていることを確認します。

```
$ oc get -o json pod <workspace pod name> -n <workspace namespace> | \
    jq '.spec.containers[] | select(.name == "<theia ide container name>").volumeMounts[] |
select(.name == "che-trusted-ca-certs")'
{
  "mountPath": "/public-certs",
  "name": "che-trusted-ca-certs",
  "readOnly": true
}
```

- Che-Theia IDE コンテナーの **/public-certs** フォルダーを検査し、その内容が **custom-certs ConfigMap** オブジェクトのエントリーの一覧と一致するかどうかを確認します。

```
$ oc exec <workspace pod name> -c <theia ide container name> -n <workspace
namespace> -- ls /public-certs
ca-bundle.crt
source-config-map-name.data-key.crt
```

関連情報

- 「自己署名証明書を使用した Git リポジトリーをサポートする OpenShift Dev Spaces のデプロイ」

3.7.4. ラベルおよびアノテーションの OpenShift ルートへの追加

組織で必要な場合は、OpenShift Route のラベルとアノテーションを設定できます。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。
- OpenShift で実行される OpenShift Dev Spaces のインスタンス。

手順

- **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_INFRA_KUBERNETES_INGRESS_LABELS: <labels> 1
        CHE_INFRA_KUBERNETES_INGRESS_ANNOTATIONS__JSON: "<annotations>"
2
```

```
    networking:
      labels: <labels> 3
      annotations: <annotations> 4
```

1 3 OpenShift Route のラベルのコンマ区切りリスト: **key1=value1,key2=value2**。

2 4 JSON 形式の OpenShift Route のアノテーション: **{"key1": "value1", "key2" : "value2"}**。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.7.5. ルーターのシャード化と連携するように OpenShift ルートを設定

OpenShift Route が ルーターのシャード化 と連携するようにラベル、アノテーション、およびドメインを設定できます。

前提条件

- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the OpenShift CLI を参照してください。
- **dsc**。「dsc 管理ツールのインストール」 を参照してください。

手順

- **CheCluster** カスタムリソースを設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_INFRA_OPENSHIFT_ROUTE_LABELS: <labels> 1
        CHE_INFRA_OPENSHIFT_ROUTE_HOST_DOMAIN__SUFFIX: <domain> 2
    networking:
      labels: <labels> 3
      domain: <domain> 4
      annotations: <annotations> 5
```

1 3 ターゲット Ingress コントローラーがルートからサービスのセットをフィルターリングする時に使用するラベルのコンマ区切りリスト。

2 4 ターゲット Ingress コントローラーが提供する DNS 名。

5 リソースと共に格納される構造化されていないキー値マップ。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.8. ストレージの設定

- 「ストレージクラスの設定」

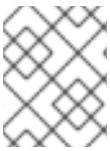
3.8.1. ストレージクラスの設定

設定されたインフラストラクチャーストレージを使用するように OpenShift Dev Spaces を設定するには、ストレージクラスを使用して OpenShift Dev Spaces をインストールします。これは、ユーザーがデフォルト以外のプロビジョナーによって提供される永続ボリュームをバインドする必要がある場合にとくに役立ちます。そのために、ユーザーはこのストレージを OpenShift Dev Spaces データ保存用にバインドし、そのストレージのパラメーターを設定します。これらのパラメーターは、以下を決定します。

- 特殊なホストパス
- ストレージ容量
- ボリューム mod
- マウントオプション
- ファイルシステム
- アクセスモード
- ストレージタイプ
- その他多数

OpenShift Dev Spaces には、データを格納するために永続ボリュームを必要とする 2 つのコンポーネントがあります。

- PostgreSQL データベース。
- OpenShift Dev Spaces ワークスペース。OpenShift Dev Spaces ワークスペースは、**/projects** ボリュームなどのボリュームを使用してソースコードを保存します。



注記

OpenShift Dev Spaces ワークスペースソースコードは、ワークスペースが一時的ではない場合にのみ永続ボリュームに保存されます。

永続ボリューム要求 (PVC) のファクト:

- OpenShift Dev Spaces は、インフラストラクチャーに永続ボリュームを作成しません。
- OpenShift Dev Spaces は、永続ボリュームクレーム (PVC) を使用して永続ボリュームをマウントします。
- OpenShift Dev Spaces サーバーは永続ボリューム要求を作成します。
  ユーザーは、OpenShift Dev Spaces PVC でストレージクラス機能を使用するために、
  OpenShift Dev Spaces 設定でストレージクラス名を定義します。ストレージクラスを使用する

と、ユーザーは追加のストレージパラメーターを使用してインフラストラクチャーストレージを柔軟に設定します。クラス名を使用して、静的にプロビジョニングされた永続ボリュームを OpenShift Dev Spaces PVC にバインドすることもできます。

手順

CheCluster カスタムリソース定義を使用してストレージクラスを定義します。

1. ストレージクラス名を定義します。**CheCluster** カスタムリソースを設定し、OpenShift Dev Spaces をインストールします。「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」を参照してください。

```
spec:
  components:
    database:
      pvc:
        # keep blank unless you need to use a non default storage class for PostgreSQL PVC
        storageClass: 'postgres-storage'
  devEnvironments:
    storage:
      pvc:
        # keep blank unless you need to use a non default storage class for workspace PVC(s)
        storageClass: 'workspace-storage'
```

2. **che-postgres-pv.yaml** ファイルで PostgreSQL データベースの永続ボリュームを定義します。

che-postgres-pv.yaml file

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: postgres-pv-volume
  labels:
    type: local
spec:
  storageClassName: postgres-storage
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/che/postgres"
```

3. **che-postgres-pv.yaml** ファイルで OpenShift Dev Spaces ワークスペースの永続ボリュームを定義します。

che-workspace-pv.yaml file

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: workspace-pv-volume
  labels:
    type: local
```

```
spec:
  storageClassName: workspace-storage
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/data/che/workspace"
```

4. 2 つの永続ボリュームをバインドします。

```
$ kubectl apply -f che-workspace-pv.yaml -f che-postgres-pv.yaml
```

注記

ボリュームの有効なファイルパーミッションを指定する必要があります。これは、ストレージクラスの設定を使用して実行することも、手動で実行することもできます。パーミッションを手動で定義するには、**storageClass#mountOptions uid** と **gid** を定義します。PostgreSQL ボリュームには **uid=26** と **gid=26** が必要です。

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.9. ブランド化

- 「Che-Theia のブランディング」

3.9.1. Che-Theia のブランディング

本章では、Che-Theia インターフェイスおよびブランディングをカスタマイズする方法について説明します。以下の要素のカスタマイズが可能です。

- **Welcome** ページと **About** ダイアログ:
  - プロダクト名
  - 製品ロゴ
  - 説明
  - 役立つリソースのリスト (**ウェルカム** ページの **ヘルプ** セクション)

カスタマイズされた Che-Theia の使用を開始するには、以下を実行します。

1. カスタマイズされた Che-Theia でコンテナーイメージをビルドします。
2. カスタムイメージを使用するエディターの **meta.yaml** を定義します。
3. カスタムエディターを使用して devfile からワークスペースを作成します。

3.9.1.1. Che-Theia のカスタムブランディング値の定義

本セクションでは、Che-Theia の基本的なブランディング要素の定義をカスタマイズする方法を説明します。

手順

ウェルカム ページで、製品の新しい名前、ロゴ、説明、およびハイパーリンクのリストを使用して **product.json** ファイルを作成します (**product.json** の例:

```
{
    "name": "Red Hat OpenShift Dev Spaces", (1)
    "icon": "icon.png", (2)
    "logo": { (3)
        "dark": "logo-light.png",
        "light": "logo-dark.png"
    },
    "welcome": { (4)
        "title": "Welcome to Your Workspace",
        "links": ["website", "documentation"]
    },
    "links": { (5)
        "website": {
            "name": "Discover Red Hat OpenShift Dev Spaces",
            "url": "https://developers.redhat.com/products/openshift-dev-spaces/overview"
        },
        "documentation": {
            "name": "Browse Documentation",
            "url": "https://www.redhat.com/docs"
        }
    }
}
```

(1) **Welcome** ページと **About** ダイアログの **name**: タブタイトル。

(2) **icon** : **Welcome** ページタブタイトルのアイコン。

(3) **logo**: **ウェルカム** ページ (最大高さ 80 ピクセル) および **バージョン** 情報ダイアログ (最大高さ 100 ピクセル) の暗いテーマと明るいテーマの製品ロゴ。透過的な背景でイメージを使用します。相対パス、絶対パス、またはイメージへの URL を定義します。

(4) **Welcome**: **Welcome** ページの動作。招待タイトルと **Help** セクションのリンクをカスタマイズします。**welcome/links** プロパティーが定義されていない場合は、**Welcome** ページに **link** セクションへのリンクが表示されます。

(5) **link**: プロダクトの便利なリソースの一覧。タグを使用してリンクをグループ化し、検索を簡素化します。

注記

ダブルアンドライトの両方には、1 つのロゴイメージのみを使用するには、以下を実行します。

```
{
    ...
    "logo": "product-logo.png"
    ...
}
```

3.9.1.2. カスタムブランディングを使用した Che-Theia コンテナーイメージのビルド

本セクションでは、カスタムブランディングを適用して Che-Theia コンテナーイメージをビルドする方法を説明します。

前提条件

- カスタムブランディング定義が含まれる **product.json** ファイル。

手順

1. サンプル **Dockerfile** をダウンロードします。
2. **Dockerfile** ディレクトリーで、**branding/** サブディレクトリーを作成します。カスタム **product.json** ファイルとロゴイメージを **branding/** ディレクトリーに置きます。
3. Che-Theia でコンテナーイメージをビルドし、イメージをコンテナーレジストリーにプッシュします。

```
$ podman build -t username/che-theia-devspaces:next .
$ podman push username/che-theia-devspaces:next
```

3.9.1.3. カスタムブランディングを使用した Che-Theia のテスト

本セクションでは、カスタムブランディングで新規ワークスペースを開き、カスタマイズされた Che-Theia をテストする方法を説明します。

前提条件

- カスタムブランディング定義でビルドされた Che-Theia コンテナーイメージ。

手順

カスタムの Che-Theia イメージをテストするには、カスタム **cheEditor** を記述する新規の **meta.yaml** ファイルを作成し、テストワークスペースに devfile で使用します。

1. **che-plugin-registry** リポジトリーのクローンを作成し、デプロイするバージョンをチェックアウトします。以下を参照してください。
   administration-guide:examples/snip_devspaces-clone-the-plug-in-registry-repository.adoc
2. **che-editors.yaml** ファイルを開きます。
3. **id** が **eclipse/che-theia/next** であるエントリーを編集し、**containers** セクションの **image** の値をカスタム Che-Theia コンテナーイメージを参照するように置き換えます。

4. レジストリーをビルドします。
   administration-guide:examples/snip_devspaces-build-a-custom-plug-in-registry.adoc

5. **./dependencies/che-plugin-registry/v3/plugins/eclipse/che-theia/next** ディレクトリーに移動します。

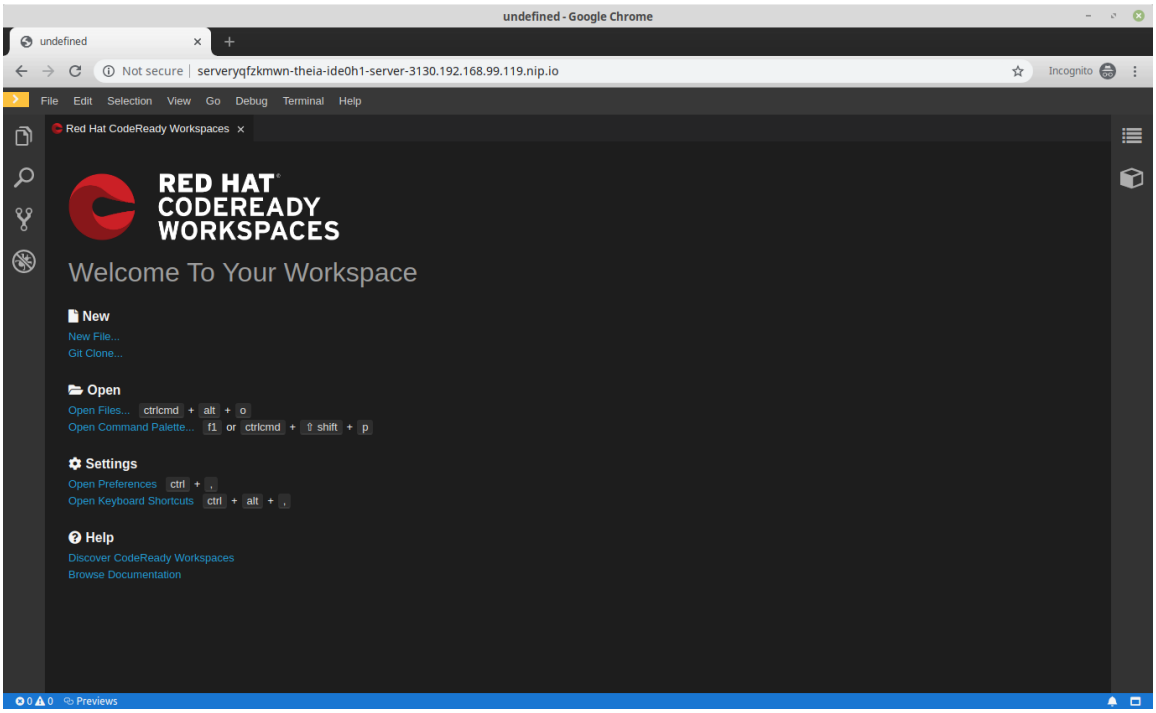
6. このディレクトリーの **meta.yaml** ファイルを、HTTP リソースとして使用できる一般にアクセス可能な場所に公開します。

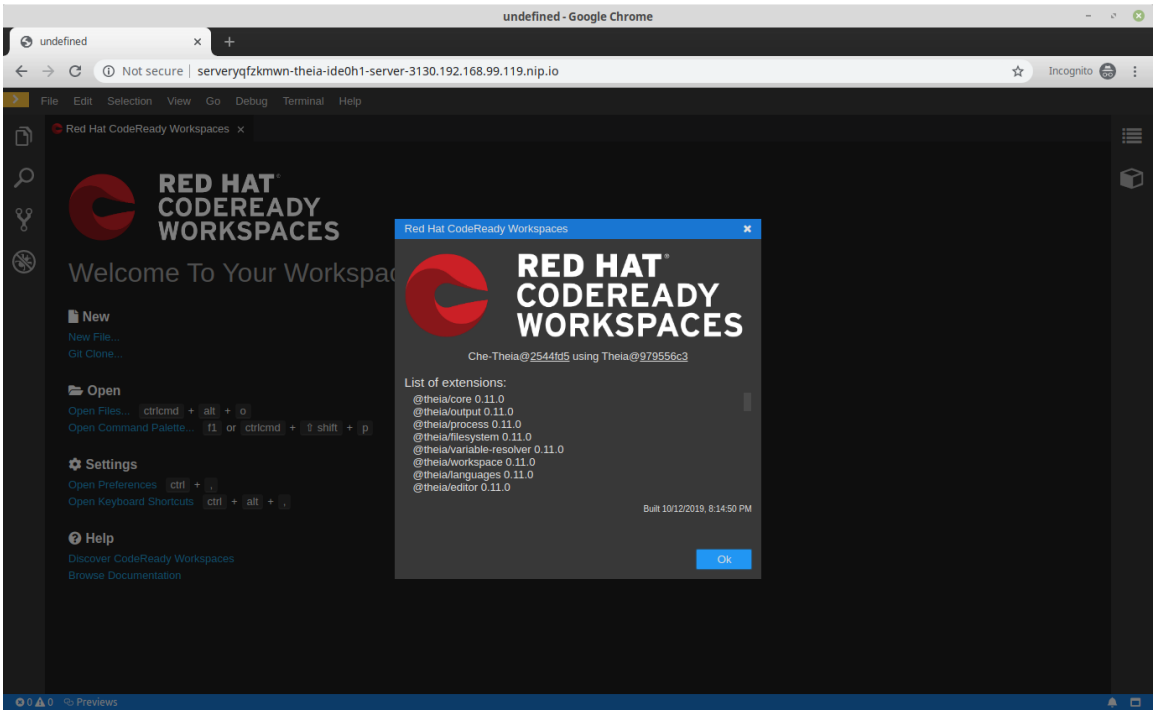
7. サンプル che-theia-branding-example devfile を使用してワークスペースを作成し、変更を適用します。
   reference フィールドが公開された **meta.yaml** ファイルを参照することを確認します。


```
metadata:
  name: che-theia-all
projects:
  - name: che-cheia-branding-example
    source:
      location: 'https://github.com/che-samples/che-theia-branding-example.git'
      type: git
      branch: master
components:
  - type: cheEditor
    reference: >-
      https://raw.githubusercontent.com/che-samples/che-theia-branding-example/master/che-
editor.meta.yaml
apiVersion: 1.0.0
```

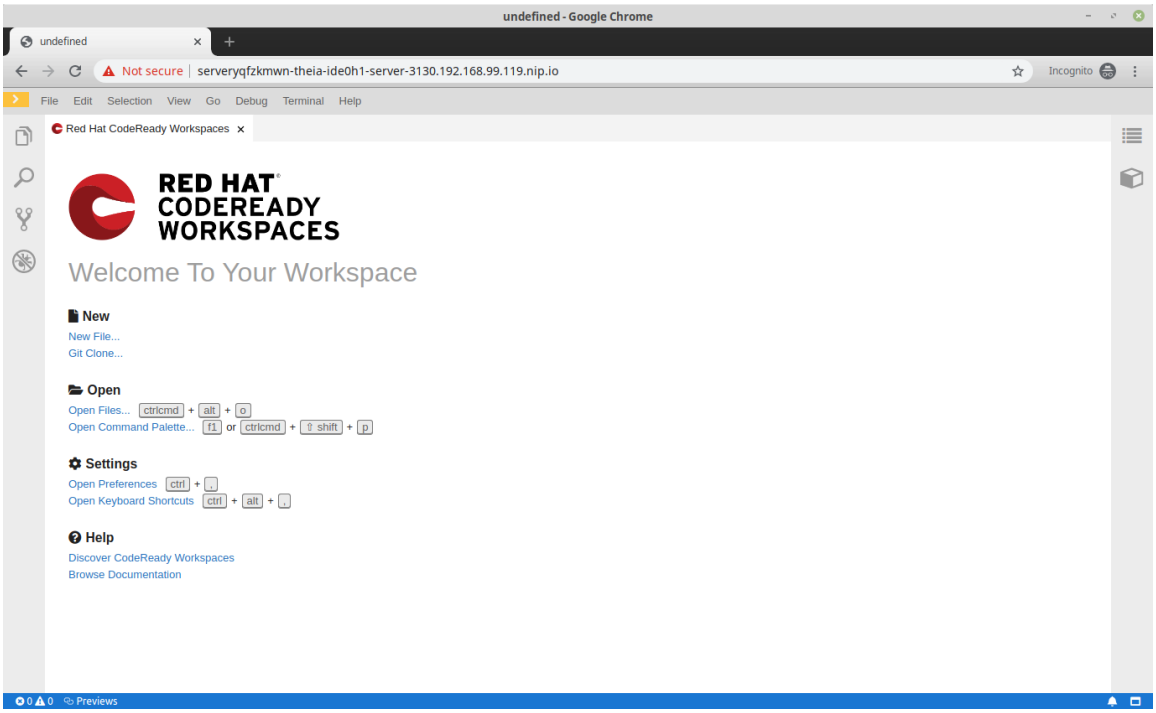
8. ワークスペースを実行して変更を表示します。

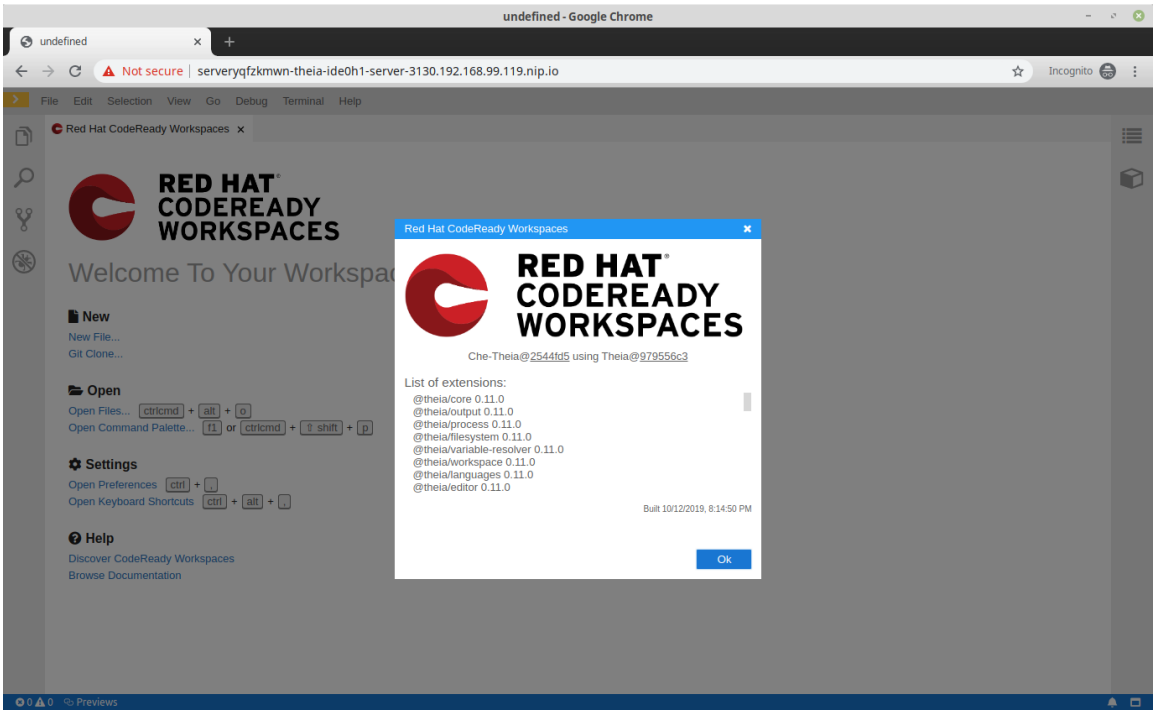
   - Che-Theia のテーマ:





- Che-Theia の軽量テーマ:





3.10. ID および承認の管理

このセクションでは、Red Hat OpenShift Dev Spaces の ID および承認の管理のさまざまな側面について説明します。

- 「ユーザーデータの削除」

3.10.1. GitHub、GitLab、または Bitbucket の OAuth

ユーザーがリモート Git リポジトリーと連携できるようにするには、以下を実行します。

- 「Configuring OAuth 2.0 for GitHub」
- 「GitLab の OAuth 2.0 の設定」

- OAuth 1.0 for a Bitbucket Server または OAuth 2.0 for the Bitbucket Cloud の設定

3.10.1.1. Configuring OAuth 2.0 for GitHub

ユーザーが GitHub でホストされるリモート Git リポジトリーと連携できるようにするには、以下を実行します。

1. GitHub OAuth アプリ (OAuth 2.0) をセットアップします。
2. GitHub OAuth アプリケーションシークレットを適用します。

3.10.1.1.1. GitHub OAuth アプリケーションの設定

Set up a GitHub OAuth App using OAuth 2.0.

前提条件

- GitHub にログインしている。
- **base64** が使用しているオペレーティングシステムにインストールされている。

手順

1. https://github.com/settings/applications/new にアクセスします。
2. 以下の値を設定します。
   a. **アプリケーション名**: **OpenShift Dev Spaces**
   b. **ホームページの URL**: **https://devspaces-<openshift_deployment_name>.<domain_name>/**
   c. **認証コールバック URL**: **https://devspaces-<openshift_deployment_name>.<domain_name>/api/oauth/callback**
3. **Register application** をクリックします。
4. **Generate new client secret** をクリックします。
5. **GitHub OAuth アプリケーションシークレットを適用する際に使用する GitHub OAuth クライアント ID** をコピーし、これを Base64 にエンコードします。

```
$ echo -n '<github_oauth_client_id>' | base64
```

6. **GitHub OAuth クライアントシークレットをコピーし、GitHub OAuth App Secretを適用する際に使用する** Base64 にエンコードします。

```
$ echo -n '<github_oauth_client_secret>' | base64
```

関連情報

- GitHub Docs: OAuth アプリケーションの作成

3.10.1.1.2. GitHub OAuth アプリケーションシークレットの適用

GitHub OAuth App Secret を準備し、これを適用します。

前提条件

- GitHub OAuth アプリケーションの設定が完了します。
- GitHub OAuth アプリケーションの設定時に生成された Base64 でエンコードされた値が作成されます。
  - **GitHub OAuth Client ID**
  - **GitHub OAuth Client Secret**
- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。

手順

1. Secret を準備します。

```
kind: Secret
apiVersion: v1
metadata:
  name: github-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: github
type: Opaque
data:
  id: <Base64_GitHub_OAuth_Client_ID> (2)
  secret: <Base64_GitHub_OAuth_Client_Secret> (3)
```

(1) OpenShift Dev Spaces 名前空間。デフォルトは **openshift-devspaces** です。

(2) Base64 でエンコードされた **GitHubOAuth クライアント ID**。

(3) base64 でエンコードされた **GitHub OAuth クライアントシークレット**。

2. シークレットを適用します。

```
$ oc apply -f - <<EOF
<Secret_prepared_in_the_previous_step>
EOF
```

3. 出力に Secret が作成されたことを確認します。

3.10.1.2. GitLab の OAuth 2.0 の設定

ユーザーが GitLab インスタンスを使用してホストされるリモート Git リポジトリーと連携できるようにするには、以下を実行します。

1. GitLab 認定アプリケーション (OAuth 2.0) をセットアップします。
2. GitLab で承認されたアプリケーションシークレットを適用します。

3.10.1.2.1. GitLab で承認されたアプリケーションの設定

OAuth 2.0 を使用して GitLab で承認されたアプリケーションを設定します。

前提条件

- GitLab にログインしている。
- **base64** が使用しているオペレーティングシステムにインストールされている。

手順

1. アバターをクリックして、**プロファイル → アプリケーション** の編集に移動します。
2. **Name** に **OpenShift Dev Spaces** を入力します。
3. **https://devspaces-<openshift_deployment_name>.<domain_name>/api/oauth/callback** を **リダイレクト URI** として指定します。
4. **Confidential** および **Expire access tokens** のチェックボックスを選択します。
5. **Scopes** の下で、**api**、**write_repository**、および **openid** のチェックボックスにチェックを入れます。
6. **Save application** をクリックします。
7. **GitLab アプリケーション ID** をコピーし、GitLab で承認されたアプリケーションシークレットを適用するときに使用する Base64 にエンコードします。

```
$ echo -n '<gitlab_application_id>' | base64
```

8. **GitLab クライアントシークレット** をコピーし、GitLab で承認されたアプリケーションシークレットを適用するときに使用する Base64 にエンコードします。

```
$ echo -n '<gitlab_client_secret>' | base64
```

関連情報

- GitLab Docs: 認証アプリケーション

3.10.1.2.2. GitLab で承認されるアプリケーションシークレットの適用

GitLab で承認されるアプリケーションシークレットを準備し、これを適用します。

前提条件

- GitLab 認証アプリケーションの設定が完了します。
- GitLab で承認されるアプリケーションの設定時に生成された Base64 でエンコードされた値が作成されます。

- GitLab Application ID
- GitLab Client Secret

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。

手順

1. Secret を準備します。

```
kind: Secret
apiVersion: v1
metadata:
  name: gitlab-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: gitlab
    che.eclipse.org/scm-server-endpoint: <gitlab_server_url> (2)
type: Opaque
data:
  id: <Base64_GitLab_Application_ID> (3)
  secret: <Base64_GitLab_Client_Secret> (4)
```

(1) OpenShift Dev Spaces 名前空間。デフォルトは **openshift-devspaces** です。

(2) **GitLab サーバーの URL** です。**SAAS** バージョンには **https://gitlab.com** を使用します。

(3) Base64 でエンコードされた **GitLab アプリケーション ID**。

(4) Base64 でエンコードされた **GitLab クライアントシークレット**。

2. シークレットを適用します。

```
$ oc apply -f - <<EOF
<Secret_prepared_in_the_previous_step>
EOF
```

3. 出力に Secret が作成されたことを確認します。

3.10.1.3. Bitbucket サーバー向け OAuth 1.0 の設定

ユーザーが Bitbucket サーバーでホストされるリモート Git リポジトリーと連携できるようにするには、以下を実行します。

1. Bitbucket Server でアプリケーションリンク (OAuth 1.0) を設定します。
2. Bitbucket Server のアプリケーションリンクシークレットを適用します。

3.10.1.3.1. Bitbucket Server でのアプリケーションリンクの設定

Bitbucket Server で OAuth 1.0 のアプリケーションリンクをセットアップします。

前提条件

- Bitbucket Server にログインしている。
- **openssl** は、使用しているオペレーティングシステムにインストールされている。
- **base64** が使用しているオペレーティングシステムにインストールされている。

手順

1. コマンドラインでコマンドを実行して、次の手順に必要なファイルを作成し、アプリケーションリンクシークレットを適用するときに使用します。

```
$ openssl genrsa -out private.pem 2048 && \
openssl pkcs8 -topk8 -inform pem -outform pem -nocrypt -in private.pem -out
privatepkcs8.pem && \
cat privatepkcs8.pem | sed 's/-----BEGIN PRIVATE KEY-----//g' | sed 's/-----END PRIVATE
KEY-----//g' | tr -d '\n' | base64 | tr -d '\n' > privatepkcs8-stripped.pem && \
openssl rsa -in private.pem -pubout > public.pub && \
cat public.pub | sed 's/-----BEGIN PUBLIC KEY-----//g' | sed 's/-----END PUBLIC KEY-----//g'
| tr -d '\n' > public-stripped.pub && \
openssl rand -base64 24 > bitbucket-consumer-key && \
openssl rand -base64 24 > bitbucket-shared-secret
```

2. **Administration → Application Links** に移動します。
3. URL フィールドに **https://devspaces-<openshift_deployment_name>.<domain_name>/** と入力し、**Create new link** をクリックします。
4. **提供されたアプリケーション URL が一度リダイレクトされました** で、**この URL を使用する** チェックボックスをオンにして、**続行** をクリックします。
5. **Application Name** に **OpenShift Dev Spaces** を入力します。
6. **Application Type** として **Generic Application** を選択します。
7. **OpenShift Dev Spaces** を **Service Provider Name**として入力します。
8. **bitbucket-consumer-key** ファイルの内容を **Consumer キー** として貼り付けます。
9. **bitbucket-shared-secret** ファイルの内容を **Shared シークレット** として貼り付けます。
10. **リクエストトークンの URL** として **<bitbucket_server_url>/plugins/servlet/oauth/request-token** と入力します。
11. **アクセストークンの URL** として **<bitbucket_server_url>/plugins/servlet/oauth/access-token** と入力します。
12. **Authorize URL** として **<bitbucket_server_url>/plugins/servlet/oauth/authorize** と入力します。
13. **受信リンクの作成** チェックボックスをオンにして、**続行** をクリックします。
14. **bitbucket_consumer_key** ファイルの内容を **Consumer キー** として貼り付けます。

15. **コンシューマー名** として **OpenShift Dev Spaces** を入力します。
16. **public-stripped.pub** ファイルの内容を **公開鍵** として貼り付け、**Continue** をクリックします。

関連情報

- Atlassian Documentation: 他のアプリケーションへのリンク

3.10.1.3.2. Bitbucket Server 用にアプリケーションリンクシークレットを適用する

Bitbucket Server のアプリケーションリンクシークレットを準備して適用します。

前提条件

- アプリケーションリンクが Bitbucket Server にセットアップされている。
- アプリケーションリンクのセットアップ時に作成された以下の Base64 でエンコードされたファイルが用意されている。
  - **privatepkcs8-stripped.pem**
  - **bitbucket_consumer_key**
  - **bitbucket-shared-secret**
- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。

手順

1. Secret を準備します。

```
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/component: oauth-scm-configuration
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: <bitbucket_server_url> (2)
type: Opaque
data:
  private.key: <Base64_content_of_privatepkcs8-stripped.pem> (3)
  consumer.key: <Base64_content_of_bitbucket_server_consumer_key> (4)
  shared_secret: <Base64_content_of_bitbucket-shared-secret> (5)
```

(1) OpenShift Dev Spaces 名前空間。デフォルトは **openshift-devspaces** です。

(2) Bitbucket Server の URL。

(3) **privatepkcs8-stripped.pem** ファイルの Base64 でエンコードされたコンテンツ。

4 **bitbucket_consumer_key** ファイルの Base64 でエンコードされたコンテンツ。

5 **bitbucket-shared-secret** ファイルの Base64 でエンコードされたコンテンツ。

2. シークレットを適用します。

```
$ oc apply -f - <<EOF
<Secret_prepared_in_the_previous_step>
EOF
```

3. 出力に Secret が作成されたことを確認します。

3.10.1.4. Bitbucket Cloud 向け OAuth 2.0 の設定

Bitbucket Cloud でホストされているリモート Git リポジトリーをユーザーが操作できるようにすることができます。

1. Bitbucket Cloud で OAuth コンシューマー (OAuth 2.0) をセットアップします。
2. Bitbucket Cloud に OAuth コンシューマーシークレットを適用します。

3.10.1.4.1. Bitbucket Cloud で OAuth コンシューマーを設定する

Bitbucket Cloud で OAuth 2.0 の OAuth コンシューマーをセットアップします。

前提条件

- Bitbucket Cloud にログインしている。
- base64 が使用しているオペレーティングシステムにインストールされている。

手順

1. アバターをクリックして、**All workspaces** ページに移動します。
2. ワークスペースを選択してクリックします。
3. **Settings → OAuth consumers → Add consumer** に移動します。
4. **Name** に **OpenShift Dev Spaces** を入力します。
5. **https://devspaces-<openshift_deployment_name>.<domain_name>/api/oauth/callback** を **コールバック URL** として入力します。
6. **Permissions** で、**Account** と **Repositories** のすべてのチェックボックスをオンにして、**Save** をクリックします。
7. 追加したコンシューマーを展開してから、**キー** 値をコピーして Base64 にエンコードし、Bitbucket OAuth コンシューマーシークレットを適用するときに使用します。

```
$ echo -n '<bitbucket_oauth_consumer_key>' | base64
```

8. Bitbucket OAuth コンシューマーシークレットを適用するときに使用するために、**Secret** 値をコピーして Base64 にエンコードします。

```
$ echo -n '<bitbucket_oauth_consumer_secret>' | base64
```

関連情報

- Bitbucket Docs: Use OAuth on Bitbucket Cloud

3.10.1.4.2. Bitbucket Cloud に OAuth コンシューマーシークレットを適用する

Bitbucket Cloud の OAuth コンシューマーシークレットを準備して適用します。

前提条件

- OAuth コンシューマーは Bitbucket Cloud でセットアップされます。
- Bitbucket OAuth コンシューマーのセットアップ時に生成された Base64 でエンコードされた値が準備されます。
  - Bitbucket OAuth コンシューマーキー
  - Bitbucket OAuth コンシューマーシークレット
- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。

手順

1. Secret を準備します。

```
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: openshift-devspaces (1)
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
type: Opaque
data:
  id: <Base64_Bitbucket_Oauth_Consumer_Key> (2)
  secret: <Base64_Bitbucket_Oauth_Consumer_Secret> (3)
```

(1) OpenShift Dev Spaces 名前空間。デフォルトは **openshift-devspaces** です。

(2) Base64 でエンコードされた **Bitbucket OAuth コンシューマーキー**。

(3) Base64 でエンコードされた **Bitbucket OAuth コンシューマーシークレット**。

2. シークレットを適用します。

```
$ oc apply -f - <<EOF
<Secret_prepared_in_the_previous_step>
EOF
```

3. 出力に Secret が作成されたことを確認します。

3.10.2. 管理ユーザーの設定

ユーザーデータの削除など、OpenShift Dev Spaces サーバーで管理者権限を必要とするアクションを実行するには、管理者権限を持つユーザーをアクティブ化します。デフォルトのインストールでは、OpenShift に存在するかどうかに関係なく、**admin** ユーザーの管理者権限が有効になります。

手順

- **CheCluster** カスタムリソースを設定して、<admin> ユーザーに管理者権限を設定します。「CLI を使用して CheCluster カスタムリソースの設定」を参照してください。

```
spec:
  components:
    cheServer:
      extraProperties:
        CHE_SYSTEM_ADMIN__NAME: '<admin>'
```

関連情報

- 「dsc を使用したインストール時に **CheCluster** カスタムリソースの設定」
- 「CLI を使用して CheCluster カスタムリソースの設定」

3.10.3. ユーザーデータの削除

3.10.3.1. GDPR に準拠したユーザーデータの削除

OpenShift Dev Spaces API を使用して、OpenShift Dev Spaces ユーザーのデータを削除できます。この手順に従うことで、個人による個人データの消去権利を課す EU 一般データ保護規則 (GDPR) にサービスが準拠するようになります。

前提条件

- OpenShift Dev Spaces への管理権限を持つアクティブなセッション。「管理ユーザーの設定」を参照してください。
- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the OpenShift CLI を参照してください。

手順

1. **<username>** ユーザー **<id> id** を取得します。https://**<devspaces-**<openshift_deployment_name>.<domain_name>>/swagger/#/user/find_1 に移動し、**Try it out** をクリックし、**name**: **<username>** を設定して **Execute** をクリックします。**レスポンス本文** を下にスクロールして、**id** 値を見つけます。
2. ユーザー設定など、OpenShift Dev Spaces サーバーが管理する **<id>** ユーザーデータを削除し

ます。
https://**<devspaces-**<openshift_deployment_name>.<domain_name>>/swagger/#/user/remove に移動し、**Try it out** をクリックし、**id**: **<id>** を設定して **Execute** をクリックします。**204** レスポンスコードが予想されます。

3. ユーザープロジェクトを削除して、ユーザーにバインドされているすべての OpenShift リソース (ワークスペース、シークレット、configmap など) を削除します。

```
$ oc delete namespace <username>-devspaces
```

関連情報

- *4章OpenShift Dev Spaces サーバー API を使用した OpenShift Dev Spaces サーバーワークロードの管理*
- 「自動プロビジョニング用のユーザープロジェクト名の設定」
- すべてのユーザーのデータを削除する場合、*6章OpenShift Dev Space のアンインストール* を参照してください。

第4章 OPENSHIFT DEV SPACES サーバー API を使用した OPENSHIFT DEV SPACES サーバーワークロードの管理

OpenShift Dev Spaces サーバーのワークロードを管理するには、Swagger Web ユーザーインターフェイスを使用して OpenShift Dev Spaces サーバー API をナビゲートします。

手順

- Swagger API Web ユーザーインターフェイス (**https://devspaces-<openshift_deployment_name>.<domain_name>/swagger**.) に移動します。

関連情報

- Swagger

第5章 OPENSHIFT DEV SPACE のアップグレード

この章では、CodeReady Workspaces 3.1 から OpenShift Dev Spaces 3.2 にアップグレードする方法について説明します。

5.1. DSC 管理ツールのアップグレード

このセクションでは、**dsc** 管理ツールをアップグレードする方法について説明します。

手順

- 「dsc 管理ツールのインストール」

5.2. RED HAT OPENSHIFT DEV SPACES OPERATOR の更新承認ストラテジーの指定

Red Hat OpenShift Dev Spaces Operator は、2 つのアップグレード戦略をサポートしています。

自動

Operator は、新しい更新が利用可能になったときにそれらをインストールします。

Manual

インストールを開始する前に、新しい更新を手動で承認する必要があります。

OpenShift Web コンソールを使用して、Red Hat OpenShift Dev Spaces Operator の更新承認ストラテジーを指定できます。

前提条件

- クラスター管理者による OpenShift Web コンソールセッション。Accessing the web console を参照してください。
- Red Hat エコシステムカタログを使用してインストールされた OpenShift Dev Spaces のインスタンス。

手順

1. OpenShift Web コンソールで、**Operators → Installed Operators** に移動します。
2. インストールされているオペレーターのリストで **Red Hat OpenShift Dev Spaces**をクリックします。
3. **Subscription** タブに移動します。
4. **更新承認** ストラテジーを **Automatic** または **Manual** に設定します。

関連情報

- Operator の更新チャネルの変更

5.3. OPENSHIFT WEB コンソールを使用した OPENSHIFT DEV SPACES のアップグレード

OpenShift Web コンソールの Red Hat エコシステムカタログにある Red Hat OpenShift Dev Spaces Operator を使用して、以前のマイナーバージョンからのアップグレードを手動で承認できます。

前提条件

- クラスター管理者による OpenShift Web コンソールセッション。Accessing the web console を参照してください。
- Red Hat エコシステムカタログを使用してインストールされた OpenShift Dev Spaces のインスタンス。
- サブスクリプションの承認ストラテジーは **Manual** になります。「Red Hat OpenShift Dev Spaces Operator の更新承認ストラテジーの指定」を参照してください。

手順

- 保留中の Red Hat OpenShift Dev Spaces Operator のアップグレードを手動で承認します。Manually approving a pending Operator upgrade を参照してください。

検証手順

1. OpenShift Dev Spaces インスタンスに移動します。
2. 3.2 のバージョン番号がページ下部に表示されます。

関連情報

- 保留中の Operator アップグレードの手動による承認

5.4. CLI 管理ツールを使用した OPENSHIFT DEV SPACE のアップグレード

本セクションでは、CLI 管理ツールを使用して以前のマイナーバージョンからアップグレードする方法を説明します。

前提条件

- OpenShift の管理者アカウント。
- 以前のマイナーバージョンの CodeReady Workspaces の実行中のインスタンス。これは **openshift-devspaces** OpenShift プロジェクトの同じ OpenShift インスタンス上で CLI 管理ツールを使用してインストールします。
- OpenShift Dev Spaces バージョン 3.2 の **dsc**。「dsc 管理ツールのインストール」 を参照してください。

手順

1. 実行中のすべての CodeReady Workspaces 3.1 ワークスペースの変更を保存し、Git リポジトリーにプッシュします。
2. CodeReady Workspaces 3.1 インスタンスのすべてのワークスペースをシャットダウンします。
3. OpenShift Dev Spaces をアップグレードします。

```
$ dsc server:update -n openshift-devspaces
```

注記

低速なシステムまたはインターネット接続の場合は、**--k8spodwaittimeout=1800000** フラグオプションを追加して、Pod のタイムアウト期間を 1800000 ms 以上に拡張します。

検証手順

1. OpenShift Dev Spaces インスタンスに移動します。
2. 3.2 のバージョン番号がページ下部に表示されます。

5.5. 制限された環境での CLI 管理ツールを使用した OPENSHIFT DEV SPACES のアップグレード

このセクションでは、Red Hat OpenShift Dev Spaces をアップグレードして、制限された環境で CLI 管理ツールを使用してマイナーバージョンの更新を実行する方法を説明します。

前提条件

- OpenShift Dev Spaces インスタンスは、**openshift-devspaces** プロジェクトの **dsc --installer operator** メソッドを使用して OpenShift にインストールされている。「OpenShift の制限された環境での OpenShift Dev Space のインストール」を参照してください。
- OpenShift クラスターに、少なくとも 64 GB のディスクスペースがある。
- OpenShift クラスターは制限されたネットワーク上で動作する準備ができており、OpenShift コントロールプレーンはパブリックインターネットにアクセスできる。非接続インストールのミラーリングについて、および ネットワークが制限された環境での Operator Lifecycle Manager の使用 を参照してください。
- OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the OpenShift CLI を参照してください。
- **registry.redhat.io** Red Hat エコシステムカタログへのアクティブな **oc** レジストリーセッション。Red Hat Container Registry authentication を参照してください。
- **opm**。Installing the **opm** CLI を参照してください。
- **jq**。Downloading **jq** を参照してください。
- **podman**。Installing Podman を参照してください。
- **<my_registry>** レジストリーへの管理アクセス権を持つアクティブな **skopeo** セッション。Installing Skopeo、Authenticating to a registry、および Mirroring images for a disconnected installation を参照してください。
- OpenShift Dev Spaces バージョン 3.2 の **dsc**。「dsc 管理ツールのインストール」を参照してください。

手順

1. ミラーリングスクリプトをダウンロードして実行し、カスタム Operator カタログをインストールして、関連するイメージをミラーリングします: prepare-restricted-environment.sh。

```
$ bash prepare-restricted-environment.sh \
  --ocp_ver "4.11" \
  --devworkspace_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.10" \
  --devworkspace_operator_version "v0.15.2" \
  --prod_operator_index "registry.redhat.io/redhat/redhat-operator-index:v4.10" \
  --prod_operator_package_name "devspaces-operator" \
  --prod_operator_version "v3.2.0" \
  --my_registry "<my_registry>" \
  --my_catalog "<my_catalog>"
```

2. CodeReady Workspaces 3.1 インスタンスで実行されているすべてのワークスペースで、変更を保存し、Git リポジトリーに再度プッシュします。

3. CodeReady Workspaces 3.1 インスタンスのすべてのワークスペースを停止します。

4. 以下のコマンドを実行します。

```
$ dsc server:update --che-operator-image="$TAG" -n openshift-devspaces --
k8spodwaittimeout=1800000
```

検証手順

1. OpenShift Dev Spaces インスタンスに移動します。

2. 3.2 のバージョン番号がページ下部に表示されます。

関連情報

- Red Hat が提供する Operator カタログ
- カスタムカタログの管理

5.6. OPENSHIFT での DEVWORKSPACE OPERATOR の修復

OLM の再起動やクラスターのアップグレードなど特定の条件下で、Dev Spaces Operator for OpenShift Dev Spaces Operator がすでにクラスターに存在する場合でも、自動的にインストールされる場合があります。その場合、次のように OpenShift で DevWorkspace Operator を修復できます。

前提条件

- 宛先 OpenShift クラスターへのクラスター管理者としてのアクティブな **oc** セッション。CLI の使用方法 を参照してください。
- OpenShift Web コンソールの **Installed Operators** ページに、DevWorkspace Operator の複数のエントリーが表示されるか、または 1 つのエントリーが **Replaceing** と **Pending** のループに陥っています。

手順

1. 失敗した Pod を含む **devworkspace-controller** namespace を削除します。

2. **DevWorkspace** および **DevWorkspaceTemplate** カスタムリソース定義 (CRD) を更新するには、変換戦略を **None** に設定し、**webhook** セクション全体を削除します。

```
spec:
  ...
  conversion:
    strategy: None
status:
...
```

ヒント

Administration → CustomResourceDefinitions で **DevWorkspace** を検索することにより、OpenShift Web コンソールの **Administrator** パースペクティブで **DevWorkspace** および **DevWorkspaceTemplate** CRD を見つけて編集できます。

注記

DevWorkspaceOperatorConfig および **DevWorkspaceRouting** CRD の変換ストラテジーは、デフォルトで **None** に設定されています。

3. DevWorkspace Operator サブスクリプションを削除します。

```
$ oc delete sub devworkspace-operator \
-n openshift-operators 1
```

1 **openshift-operators** または DevWorkspace Operator がインストールされている OpenShift プロジェクト。

4. **<devworkspace-operator.vX.Y.Z>** 形式で DevWorkspace Operator CSV を取得します。

```
$ oc get csv | grep devworkspace
```

5. 各 DevWorkspace Operator CSV を削除します。

```
$ oc delete csv <devworkspace-operator.vX.Y.Z> \
-n openshift-operators 1
```

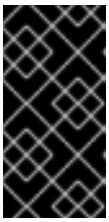
1 **openshift-operators** または DevWorkspace Operator がインストールされている OpenShift プロジェクト。

6. DevWorkspace Operator サブスクリプションを再作成します。

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: devworkspace-operator
  namespace: openshift-operators
spec:
  channel: fast
```

```
  name: devworkspace-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic (1)
  startingCSV: devworkspace-operator.v0.15.2
EOF
```

(1) **Automatic** または **Manual**。



重要

installPlanApproval: Manual の場合、OpenShift Web コンソールの **Administrator** パースペクティブで **Operator → Installed Operators** に移動し、**DevWorkspace Operator**: **Upgrade available → Preview InstallPlan → Approve** に対して以下を選択します。

7. OpenShift Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動し、**DevWorkspace Operator** の **Succeeded** ステータスを確認します。

第6章 OPENSHIFT DEV SPACE のアンインストール

警告

OpenShift Dev Spaces をアンインストールすると、OpenShift Dev Spaces 関連のすべてのユーザーデータが削除されます。

Red Hat OpenShift Dev Spaces 3.2 のインスタンスをアンインストールするには、以下を実行します。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。CLI の使用方法を参照してください。
- **dsc**。「dsc 管理ツールのインストール」 を参照してください。

手順

1. OpenShift Dev Spaces プロジェクトの名前を取得します。デフォルトは **openshift-devspaces** です。

```
$ oc get checluster --all-namespaces \
  -o=jsonpath="{.items[*].metadata.namespace}"
```

2. OpenShift Dev Spaces インスタンスを **openshift-devspaces** プロジェクトから削除します。

```
$ dsc server:delete -n openshift-devspaces 1
```

1 (手順 1 で取得したプロジェクト)。

注記

OpenShift Dev Spaces が OpenShift Web コンソールからインストールされた場合、**dsc** は DevWorkspace Operator をアンインストールしません。DevWorkspace Operator をアンインストールするには、Deleting the DevWorkspace Operator dependency を参照してください。