Red Hat CodeReady Workspaces 2.14

管理ガイド

Administering Red Hat CodeReady Workspaces 2.14

Last Updated: 2023-02-03

Red Hat CodeReady Workspaces 2.14 管理ガイド

Administering Red Hat CodeReady Workspaces 2.14

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 CodeReady Workspaces の運用に関する情報

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージをご覧ください。

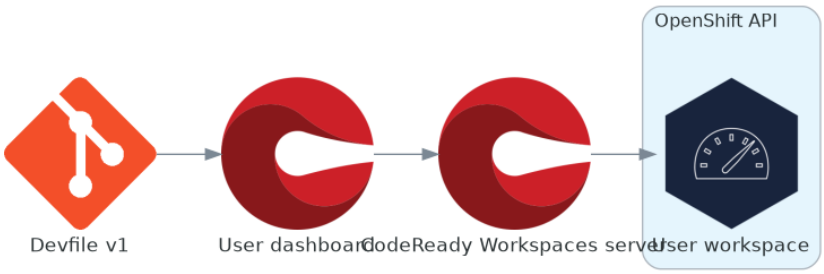
第1章 アーキテクチャーの概要

CodeReady Workspaces では、ワークスペースのライフサイクルを管理するためにワークスペースエンジンが必要です。2 つのワークスペースエンジンを使用できます。ワークスペースエンジンの選択は、アーキテクチャーを定義します。

「CodeReady Workspaces サーバーを使用した CodeReady Workspaces アーキテクチャー」

CodeReady Workspaces サーバーはデフォルトのワークスペースエンジンです。

図1.1 CodeReady Workspaces サーバーエンジンを使用した高レベルの CodeReady Workspaces アーキテクチャー



「Dev Workspace を使用した CodeReady Workspaces アーキテクチャー」

Dev Workspace Operator は新規ワークスペースエンジンです。

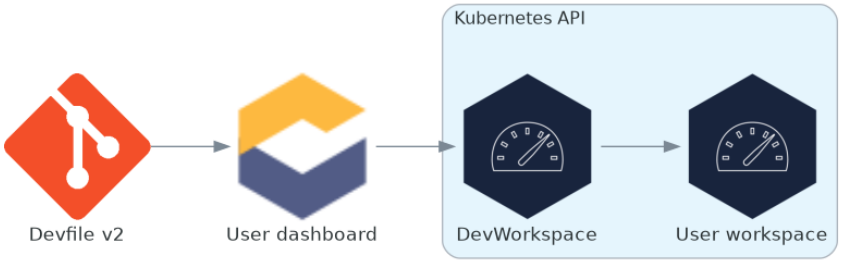
テクノロジープレビューの機能

Dev Workspace エンジンでのワークスペースの管理は実験的な機能です。このワークスペースエンジンを実稼働環境で使用しないでください。

既知の制限

ワークスペースのセキュリティーは保護されません。ワークスペースの URL がアクセスできることを確認し、ユーザーの認証情報をリークできるユーザーは常にあります。

図1.2 Dev Workspace Operator を使用した高レベルの CodeReady Workspaces アーキテクチャー



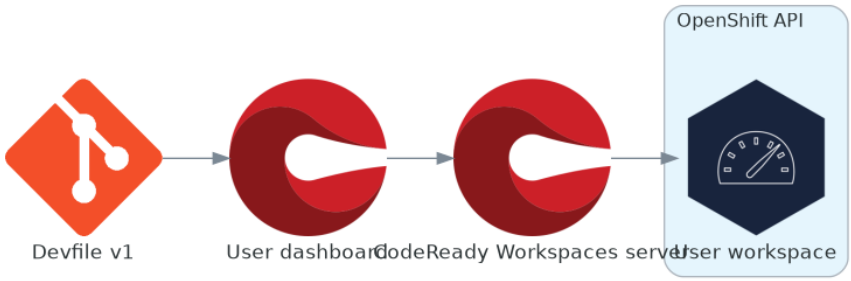
関連情報

- 「CodeReady Workspaces サーバーを使用した CodeReady Workspaces アーキテクチャー」
- 「Dev Workspace を使用した CodeReady Workspaces アーキテクチャー」
- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc
- Dev Workspace Operator GitHub リポジトリー

1.1. CODEREADY WORKSPACES サーバーを使用した CODEREADY WORKSPACES アーキテクチャー

CodeReady Workspaces サーバーはデフォルトのワークスペースエンジンです。

図1.3 CodeReady Workspaces サーバーエンジンを使用した高レベルの CodeReady Workspaces アーキテクチャー



Red Hat CodeReady Workspaces コンポーネントには以下が含まれます。

CodeReady Workspaces サーバー

OpenShift API でユーザーワークスペースを管理する always 実行サービス。

ユーザーワークスペース

ユーザー要求で実行されるコンテナーベースの IDE。

関連情報

- 「CodeReady Workspaces サーバーについて」
- 「CodeReady Workspaces ワークスペースアーキテクチャーについて」

1.2. CODEREADY WORKSPACES サーバーについて

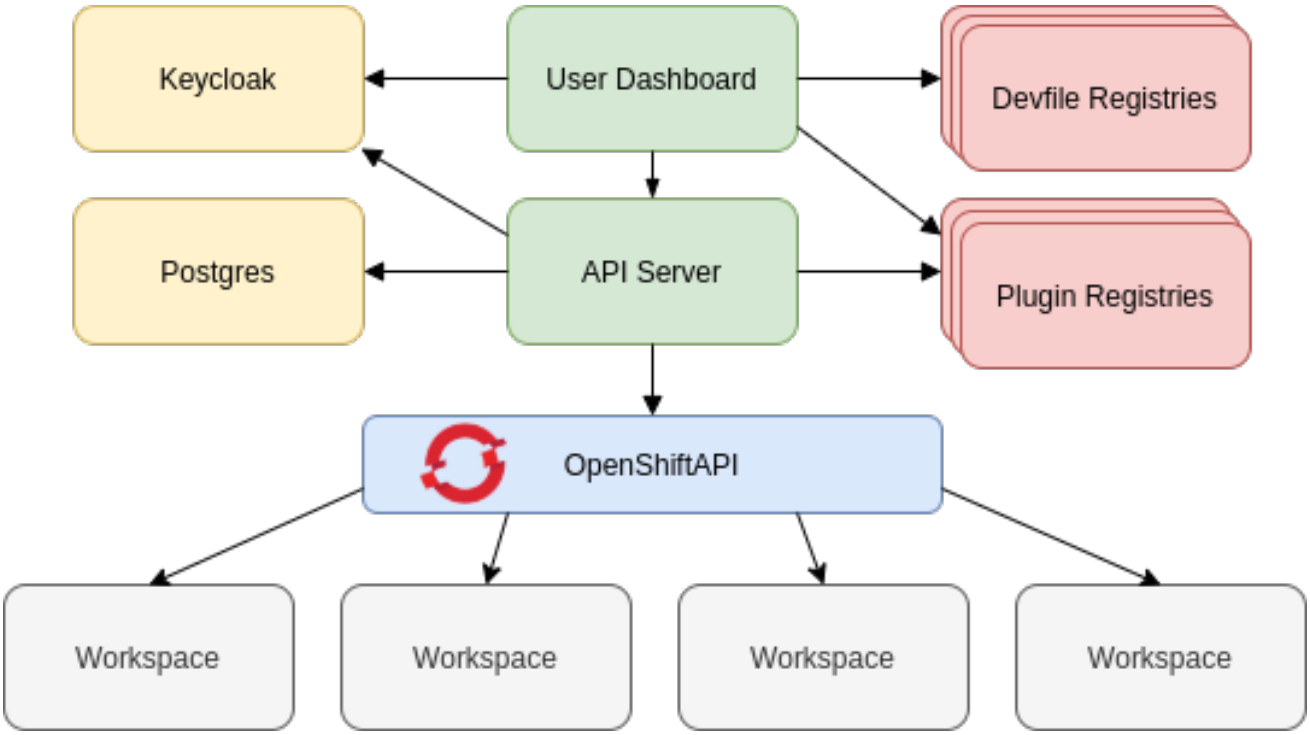
本章では、CodeReady Workspaces コントローラーとコントローラーの一部であるサービスを説明します。

1.2.1. CodeReady Workspaces サーバー

ワークスペースコントローラーは、コンテナーベースの開発環境を管理します。CodeReady Workspaces ワークスペース。認証で開発環境のセキュリティーを保護するために、デプロイメントは常にマルチユーザーおよびマルチテナントになります。

以下の図は、CodeReady Workspaces ワークスペースコントローラーの一部である各種サービスを示しています。

図1.4 CodeReady Workspaces ワークスペースコントローラー



関連情報

- 「ユーザーの認証」

1.2.2. CodeReady Workspaces サーバー

CodeReady Workspaces サーバーは、CodeReady Workspaces サーバー側のコンポーネントの中心的なサービスです。これは、HTTP REST API を公開して CodeReady Workspaces ワークスペースおよびユーザーを管理する Java Web サービスです。これはデフォルトのワークスペースエンジンです。

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc

1.2.3. CodeReady Workspaces 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 CodeReady Workspaces のランディングページです。これは React アプリケーションです。CodeReady Workspaces ユーザーはブラウザーからユーザーダッシュボードに移動し、CodeReady Workspaces ワークスペースを作成し、起動し、管理します。

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#navigating-che.adoc

1.2.4. CodeReady Workspaces devfile レジストリー

CodeReady Workspaces 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 CodeReady Workspaces サンプルの一覧を提供するサービスです。このサンプルのリストは、**Dashboard → Create Workspace** ウィンドウで使用されます。devfile レジストリーはコンテナー

で実行され、ユーザーダッシュボードが接続できる任意の場所にデプロイできます。

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#creating-a-workspace-from-a-code-sample.adoc
- CodeReady Workspaces devfile レジストリーリポジトリー

1.2.5. CodeReady Workspaces プラグインレジストリー

CodeReady Workspaces プラグインレジストリーは、CodeReady Workspaces ワークスペースのプラグインおよびエディターの一覧を提供するサービスです。devfile は、CodeReady Workspaces プラグインレジストリーに公開されるプラグインのみを参照します。これはコンテナーで実行され、CodeReady Workspaces サーバーが接続するすべての場所にデプロイできます。

1.2.6. CodeReady Workspaces および PostgreSQL

PostgreSQL データベースは、CodeReady Workspaces サーバーおよび RH-SSO の前提条件です。

CodeReady Workspaces 管理者は以下を選択できます。

- CodeReady Workspaces を既存の PostgreSQL インスタンスに接続します。
- CodeReady Workspaces デプロイメントで新規の専用 PostgreSQL インスタンスを起動します。

サービスは、以下の目的でデータベースを使用します。

CodeReady Workspaces サーバー
: ワークスペースメタデータや Git 認証情報などのユーザー設定を永続化します。

RH-SSO
: ユーザー情報を永続化します。

関連情報

- 「外部データベースの設定」
- **quay.io/eclipse/che-postgres** container image
- CodeReady Workspaces Postgres リポジトリー

1.2.7. CodeReady Workspaces および RH-SSO

RH-SSO は、CodeReady Workspaces を設定するための前提条件です。CodeReady Workspaces 管理者は、CodeReady Workspaces を既存の RH-SSO インスタンスに接続するか、または CodeReady Workspaces デプロイメントで新規の専用 RH-SSO インスタンスを起動することを選択できます。

CodeReady Workspaces サーバーは、OpenID Connect (OIDC) プロバイダーとして RH-SSO を使用して CodeReady Workspaces ユーザーの認証を行い、CodeReady Workspaces リソースへのアクセスのセキュリティーを保護します。

関連情報

- **quay.io/eclipse/che-keycloak** container image
- CodeReady Workspaces RH-SSO リポジトリー

1.3. CODEREADY WORKSPACES ワークスペースアーキテクチャーについて

本章では、CodeReady Workspaces のアーキテクチャーおよびコンポーネントを説明します。

1.3.1. CodeReady Workspaces ワークスペースアーキテクチャー

クラスターの CodeReady Workspaces デプロイメントは、CodeReady Workspaces サーバーコンポーネント、ユーザープロファイルおよび設定を保存するデータベース、およびワークスペースをホストするいつくかの追加のデプロイメントで構成されます。CodeReady Workspaces サーバーは、ワークスペースコンテナーと有効にされたプラグイン、以下のような関連するコンポーネントを含むデプロイメントで構成されるワークスペースの作成をオーケストレーションします。

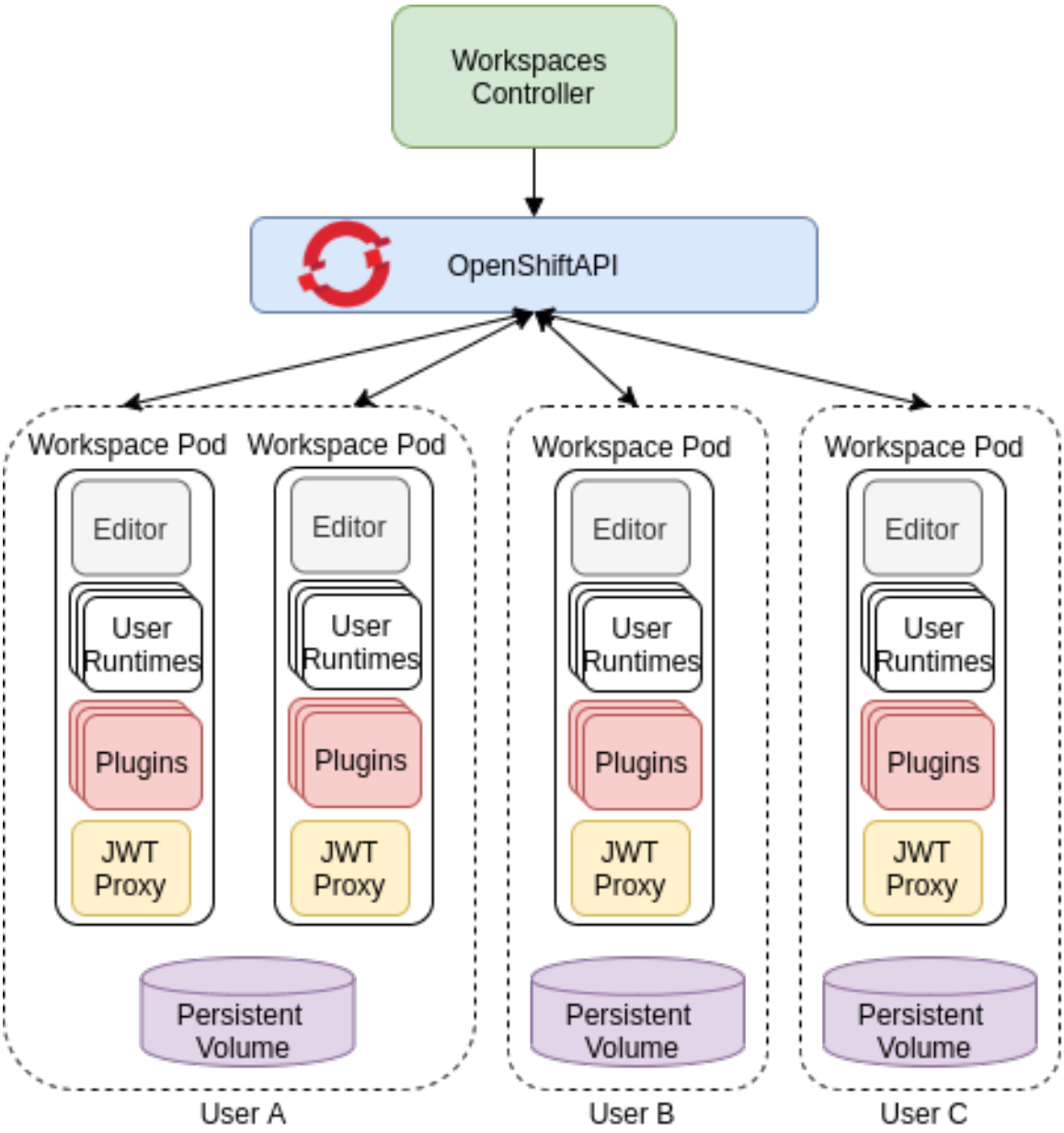
- ConfigMap
- service
- endpoints
- ingress またはルート
- secrets
- 永続ボリューム (PV)

CodeReady Workspaces ワークスペースは Web アプリケーションです。これは、エディター、言語の自動補完、デバッグツールなどの最新の IDE のすべてのサービスを提供するコンテナーで実行されるマイクロサービスで構成されます。IDE サービスは、OpenShift リソースとして定義されるコンテナーおよびユーザーランタイムアプリケーションにパッケージ化された開発ツールでデプロイされます。

CodeReady Workspaces ワークスペースのプロジェクトのソースコードは、OpenShift **PersistentVolume** で永続化されます。マイクロサービスは、ソースコード（IDE サービス、開発ツール) への読み取り/書き込みアクセスを持つコンテナー内で実行され、ランタイムアプリケーションはこの共有ディレクトリーへの読み取り/書き込みアクセスがあります。

以下の図は、CodeReady Workspaces ワークスペースの詳細なコンポーネントを示しています。

図1.5 CodeReady Workspaces 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 4 つあります。2 つは **User A** に属し、1 つは **User B** に属し、もう 1 つは **User C** に属します。

devfile 形式を使用して、CodeReady Workspaces 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。

1.3.2. CodeReady Workspaces ワークスペースコンポーネント

本セクションでは、CodeReady Workspaces ワークスペースのコンポーネントについて説明します。

1.3.2.1. Che Editor プラグイン

Che Editor プラグインは、CodeReady Workspaces ワークスペースプラグインです。これは、ワークスペースでエディターとして使用される Web アプリケーションを定義します。デフォルトの CodeReady Workspaces ワークスペースエディターは Che-Theia です。これは、Visual Studio Code (VS Code) と同様の Web ベースのソースコードエディターです。これには、VS Code 拡張機能をサポートするプラグインシステムがあります。

ソースコード	Che-Theia
コンテナーイメージ	**eclipse/che-theia**
エンドポイント	**theia**、**webviews**、**theia-dev**、**theia-redirect-1**、**theia-redirect-2**、 **theia-redirect-3**

関連情報

- Che-Theia
- Eclipse Theia オープンソースプロジェクト
- Visual Studio Code

1.3.2.2. CodeReady Workspaces ユーザーランタイム

ユーザーランタイムとして終了しないユーザーコンテナーを使用します。コンテナーイメージまたは OpenShift リソースのセットとして定義できるアプリケーションは、CodeReady Workspaces ワークスペースに追加できます。これにより、CodeReady Workspaces ワークスペースでのアプリケーションのテストが容易になります。

CodeReady Workspaces ワークスペースでアプリケーションをテストするには、ワークスペース仕様にステージまたは実稼働環境で使用するアプリケーションの YAML 定義を含めます。これは、12 要素のアプリケーション開発/実稼働パリティーです。

ユーザーランタイムの例は Node.js、SpringBoot または MongoDB、および MySQL です。

1.3.2.3. CodeReady Workspaces ワークスペース JWT プロキシー

JWT プロキシーは、CodeReady Workspaces ワークスペースサービスの通信のセキュリティーを保護します。

HTTP プロキシーは、ワークスペースサービスから CodeReady Workspaces サーバーへの送信要求に署名し、ブラウザーで実行されている IDE クライアントからの受信要求を認証するために使用されます。

ソースコード	JWT プロキシー
コンテナーイメージ	**eclipse/che-jwtproxy**

1.3.2.4. CodeReady Workspaces プラグインブローカー

プラグインブローカーは、プラグイン **meta.yaml** ファイルに関連した特別なサービスです。以下を実行します。

- CodeReady Workspaces サーバーが認識するプラグイン定義を提供するためにすべての情報を収集します。
- ワークスペースプロジェクトで準備の各種アクションを実行する (ダウンロード、ファイルの展開、プロセス設定)。

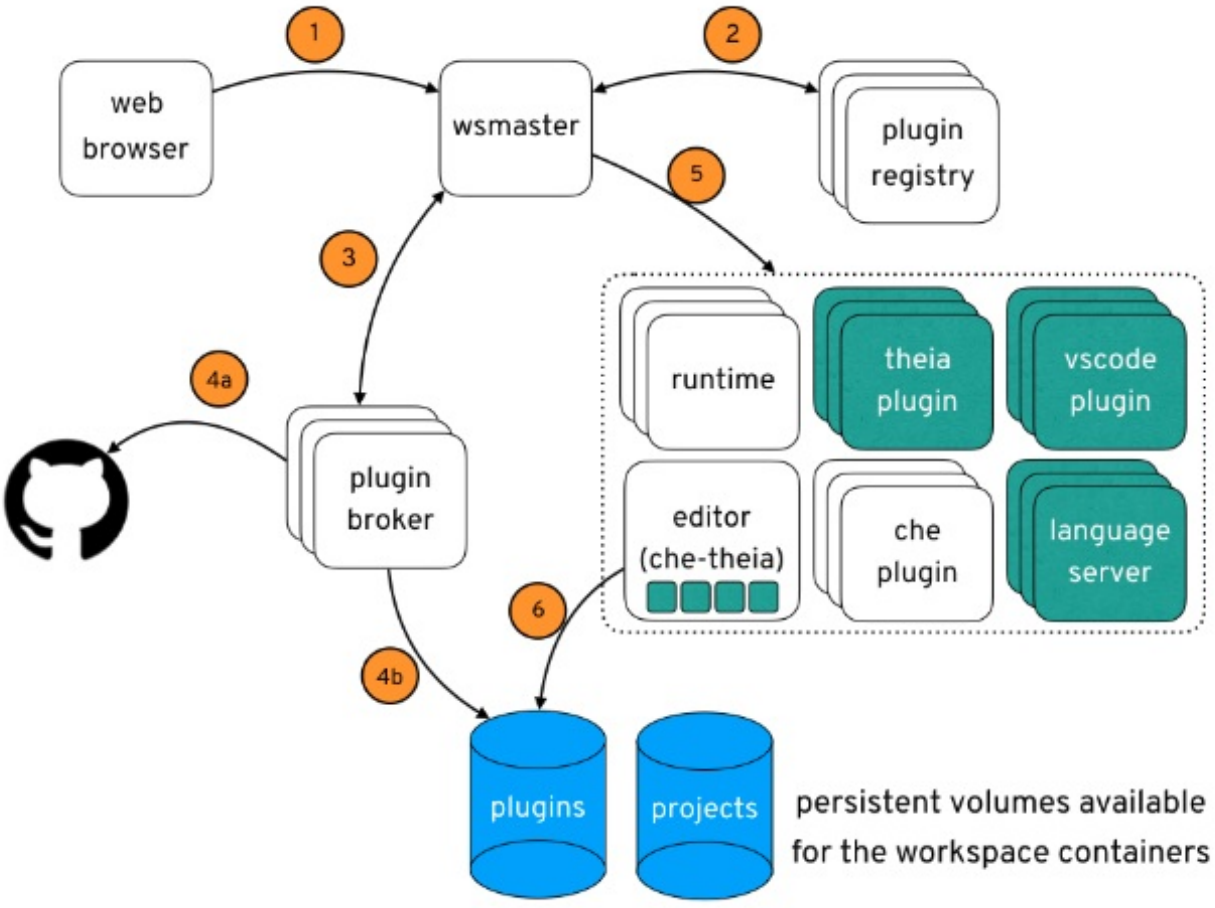
プラグインフローカーの主な目的は、CodeReady Workspaces がサポートできる実際のプラグインから CodeReady Workspaces プラグイン定義を切り離すことにあります。ブローカーでは、CodeReady Workspaces は CodeReady Workspaces サーバーを更新せずに異なるプラグインをサポートできます。

CodeReady Workspaces サーバーはプラグインブローカーを起動します。プラグインブローカーは、ワークスペースと同じ OpenShift プロジェクトで実行されます。これには、プラグインおよびプロジェクトの永続ボリュームへのアクセスがあります。

プラグインブローカーはコンテナーイメージとして定義されます (例: **eclipse/che-plugin-broker**)。プラグインタイプは、起動しているブローカーのタイプを判別します。2 種類のプラグインがサポートされます。**Che プラグインおよび Che エディター**。

ソースコード	CodeReady Workspaces プラグインブローカー
コンテナーイメージ	**quay.io/eclipse/che-plugin-artifacts-broker** **eclipse/che-plugin-metadata-broker**

1.3.3. CodeReady Workspaces ワークスペース作成フロー



以下は、CodeReady Workspaces ワークスペースの作成フローです。

1. ユーザーは、以下によって定義された CodeReady Workspaces ワークスペースを起動します。
    - エディター (デフォルトは Che-Theia)
    - プラグインの一覧（Java や OpenShift ツールなど）

- ランタイムアプリケーションの一覧

2. CodeReady Workspaces サーバーは、プラグインレジストリーからエディターおよびプラグインメタデータを取得します。
3. すべてのプラグインタイプに対して、CodeReady Workspaces サーバーは特定のプラグインブローカーを起動します。
4. CodeReady Workspaces プラグインブローカーは、プラグインのメタデータを Che Plugin 定義に変換します。これは以下の手順を実行します。
   a. プラグインをダウンロードし、そのコンテンツを展開します。
   b. プラグインの **meta.yaml** ファイルを処理し、これを Che Plugin の形式で CodeReady Workspaces サーバーに戻します。
5. CodeReady Workspaces サーバーはエディターとプラグインサイドカーを起動します。
6. エディターは、プラグインの永続ボリュームからプラグインを読み込みます。

1.4. DEV WORKSPACE を使用した CODEREADY WORKSPACES アーキテクチャー

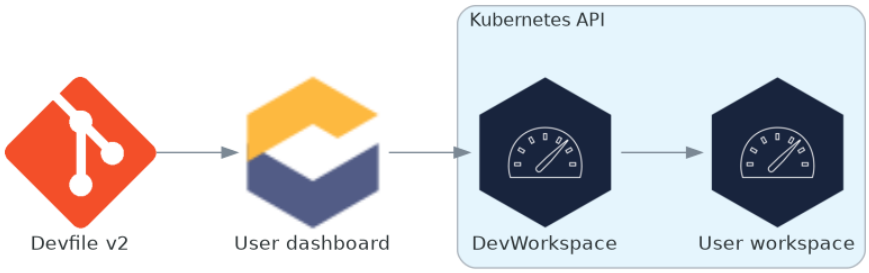
テクノロジープレビューの機能

Dev Workspace エンジンでのワークスペースの管理は実験的な機能です。このワークスペースエンジンを実稼働環境で使用しないでください。

既知の制限

ワークスペースのセキュリティーは保護されません。ワークスペースの URL がアクセスできることを確認し、ユーザーの認証情報をリークできるユーザーは常にあります。

図1.6 Dev Workspace Operator を使用した高レベルの CodeReady Workspaces アーキテクチャー



CodeReady Workspaces が Dev Workspace Operator で実行されている場合、これはコンポーネントの 3 つのグループで実行されます。

CodeReady Workspaces サーバーコンポーネント

User プロジェクトおよびワークスペースを管理します。メインコンポーネントは、ユーザーがワークスペースを制御する User ダッシュボードです。

dev Workspace 演算子

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。**Pod**、**サービス**、および **PeristentVolumes** が含まれます。

ユーザーワークスペース

コンテナーベースの開発環境（IDE を含む）

これらの OpenShift 機能の役割は中心的な要素です。

Dev Workspace 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、および CodeReady Workspaces で操作します。これは、3 つのコンポーネントのグループの通信チャネルです。

OpenShift のロールベースアクセス制御(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

- 「CodeReady Workspaces サーバーコンポーネント」
- 「dev Workspace 演算子」
- 「ユーザーワークスペース」
- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc

- dev Workspace Operator リポジトリー
- Kubernetes ドキュメント： カスタムリソース

1.5. CODEREADY WORKSPACES サーバーコンポーネント

テクノロジープレビューの機能

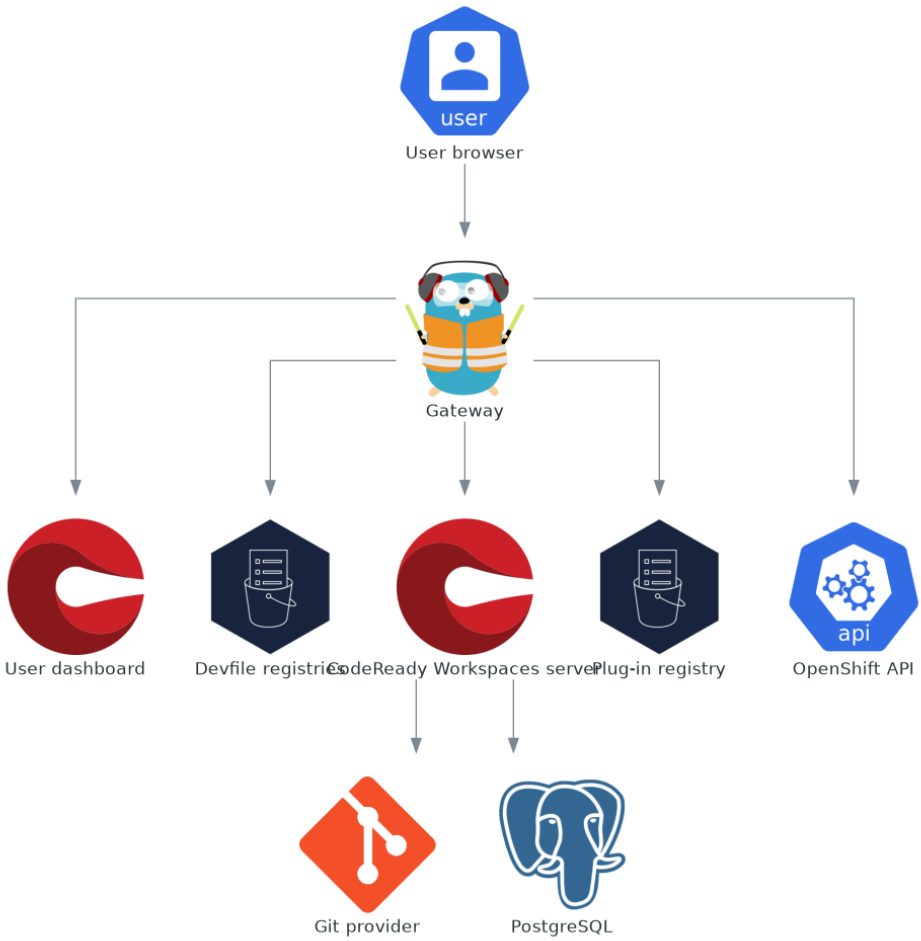
Dev Workspace エンジンでのワークスペースの管理は実験的な機能です。このワークスペースエンジンを実稼働環境で使用しないでください。

既知の制限

ワークスペースのセキュリティーは保護されません。ワークスペースの URL がアクセスできることを確認し、ユーザーの認証情報をリークできるユーザーは常にあります。

CodeReady Workspaces サーバーコンポーネントは、マルチテナンシーおよびワークスペース管理を確実にします。

図1.7 Dev Workspace Operator と対話する CodeReady Workspaces サーバーコンポーネント



関連情報

- 「CodeReady Workspaces Operator」
- 「dev Workspace 演算子」
- 「ゲートウェイ」
- 「ユーザーダッシュボード」
- 「devfile レジストリー」
- 「CodeReady Workspaces サーバー」
- 「PostgreSQL」

- 「プラグインレジストリー」

1.5.1. CodeReady Workspaces Operator

CodeReady Workspaces Operator は、CodeReady Workspaces 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確保します。以下が導入されています。

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。

CodeReady Workspaces コントローラー

Pod、サービス、永続ボリュームなどの CodeReady Workspaces インスタンスの実行に必要な 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)

CodeReady Workspaces Operator のクラスターで、**CheCluster カスタムリソース** (CR)を作成できます。CodeReady Workspaces Operator は、この CodeReady Workspaces インスタンスの CodeReady Workspaces 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可能にします。

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#configuring-the-che-installation.adoc
- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#installing-che.adoc

1.5.2. dev Workspace 演算子

テクノロジープレビューの機能

Dev Workspace エンジンでのワークスペースの管理は実験的な機能です。このワークスペースエンジンを実稼働環境で使用しないでください。

既知の制限

ワークスペースのセキュリティーは保護されません。ワークスペースの URL がアクセスできることを確認し、ユーザーの認証情報をリークできるユーザーは常にあります。

Dev Workspace Operator は OpenShift を拡張し、Dev Workspace サポートを提供します。以下が導入されています。

dev Workspace カスタムリソース定義

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

dev Workspace コントローラー

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

dev Workspace カスタムリソース

Dev Workspace Operator を使用するクラスターでは、Dev Workspace カスタムリソース(CR)を作成できます。Dev Workspace CR は Devfile の OpenShift 表現です。これは、OpenShift クラスターでユーザーワークスペースを定義します。

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc
- devfile API リポジトリー

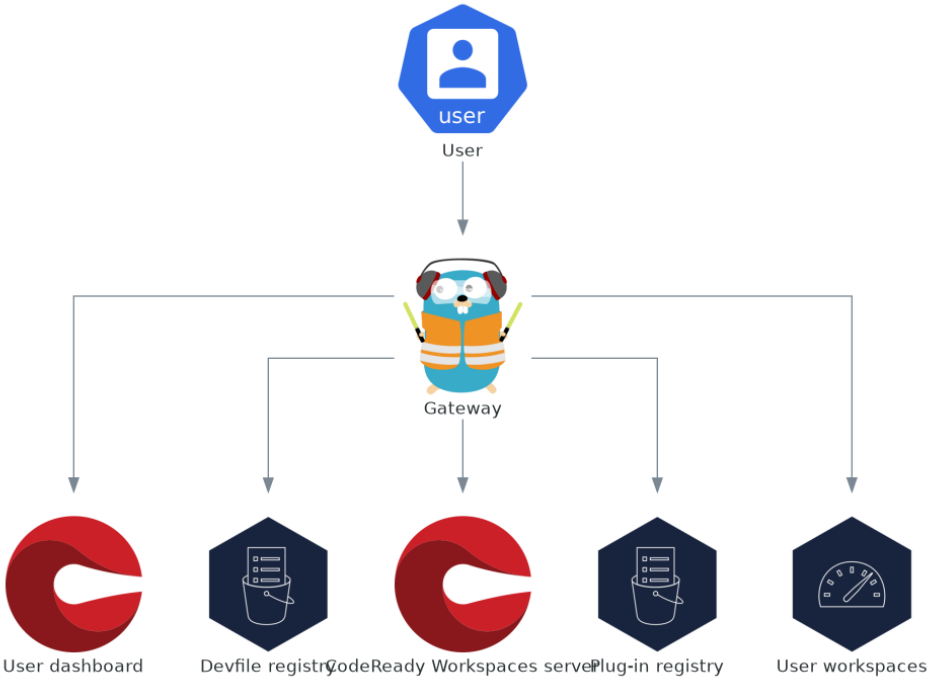
1.5.3. ゲートウェイ

CodeReady Workspaces ゲートウェイは、OpenShift Role based Access Control(RBAC)ポリシーを適用し、CodeReady Workspaces リソースへのアクセスを制御する Traefik インスタンスです。CodeReady Workspaces Operator はこれを **che-gateway** Deployment として管理します。

以下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」
- 「CodeReady Workspaces サーバー」
- 「プラグインレジストリー」
- 「ユーザーワークスペース」

図1.8 CodeReady Workspaces ゲートウェイと他のコンポーネントとの対話



関連情報

- *12章ID および承認の管理*

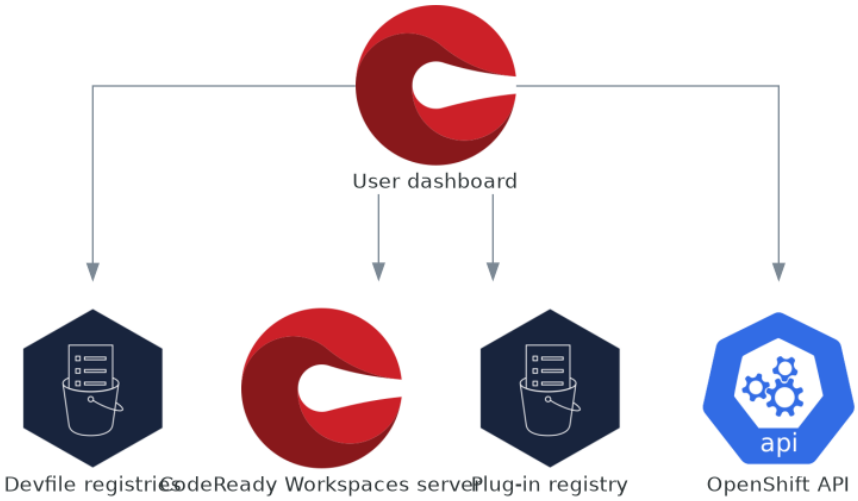
1.5.4. ユーザーダッシュボード

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

以下へのアクセスが必要です。

- 「devfile レジストリー」
- 「CodeReady Workspaces サーバー」
- 「プラグインレジストリー」
- OpenShift API

図1.9 ユーザーダッシュボードの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以下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時に、「devfile レジストリー」 から devfile を収集します。
2. リポジトリーの URL を 「CodeReady Workspaces 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時に devfile が返すことを想定します。
3. ワークスペースを記述する devfile を読み取ります。
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. 情報を Dev Workspace カスタムリソースに変換します。
6. OpenShift API を使用してユーザープロジェクトで Dev Workspace カスタムリソースを作成します。
7. Dev Workspace 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。

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#navigating-che.adoc

1.5.5. devfile レジストリー

CodeReady Workspaces 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** ページにサンプル一覧を表示します。各サンプルには Devfile v2 が含まれます。CodeReady Workspaces 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1.10 devfile レジストリーと他のコンポーネントとの対話

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#creating-a-workspace-from-a-code-sample.adoc
- devfile v2 ドキュメント
- devfile レジストリー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
- CodeReady Workspaces devfile レジストリーリポジトリー

1.5.6. CodeReady Workspaces サーバー

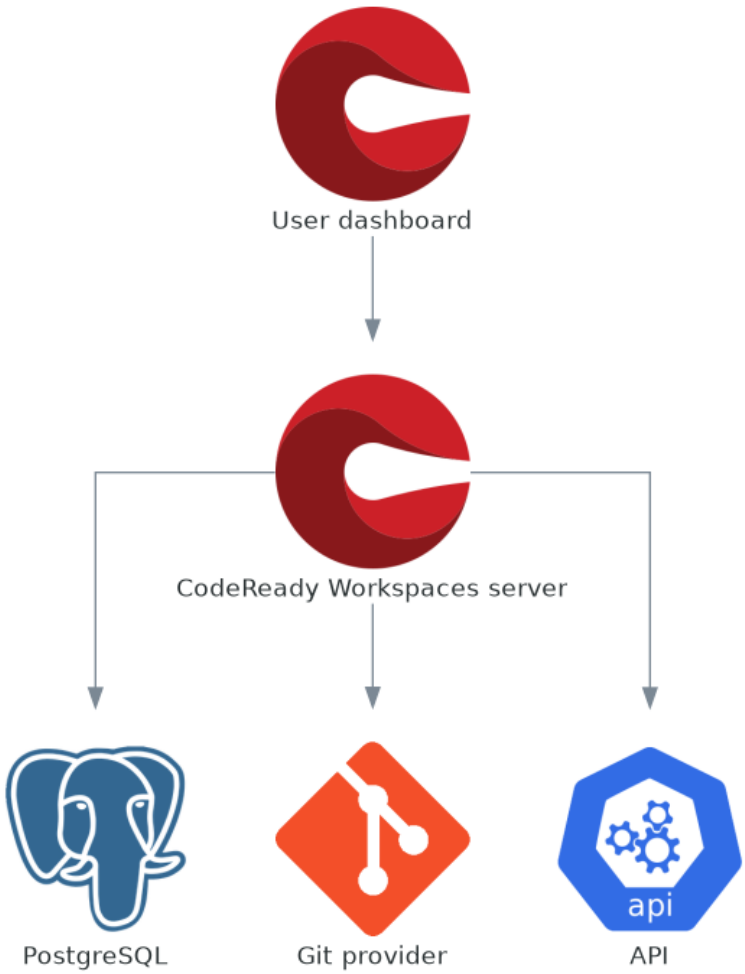
CodeReady Workspaces サーバーの主な機能は以下のとおりです。

- ユーザー namespace の作成。
- 必要なシークレットおよび設定マップを使用してユーザー namespace をプロビジョニングします。
- Git サービスプロバイダーとの統合により、devfile および認証を取得し、検証します。

CodeReady Workspaces サーバーは、HTTP REST API を公開する Java Web サービスであり、以下へのアクセスが必要になります。

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

図1.11 CodeReady Workspaces サーバーの他のコンポーネントとの対話



関連情報

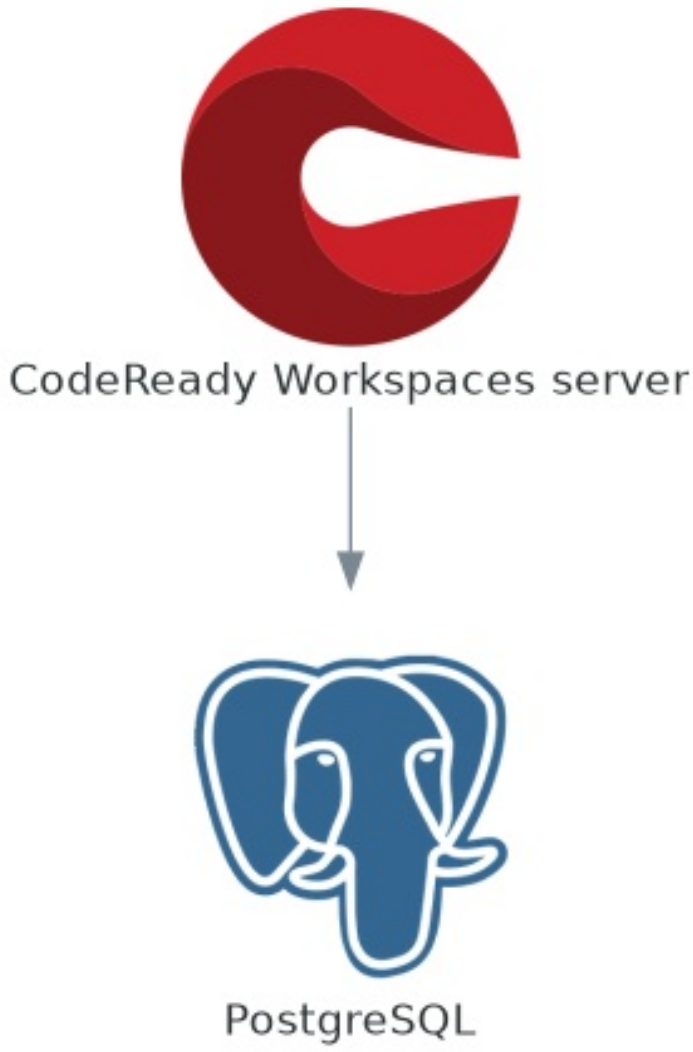
- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc

1.5.7. PostgreSQL

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

CodeReady Workspaces デプロイメントでは、postgres デプロイメントで専用の 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できます。**

図1.12 PostgreSQL の他のコンポーネントとの対話



関連情報

- 「外部データベースの設定」
- quay.io/eclipse/che-postgres container image
- CodeReady Workspaces Postgres リポジトリー

1.5.8. プラグインレジストリー

各 CodeReady Workspaces ワークスペースは、特定のエディターおよび関連付けられた拡張機能のセットで起動します。CodeReady Workspaces 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拡張の一覧を提供します。Devfile v2 は各エディターまたはエクステンションを記述します。

「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ります。

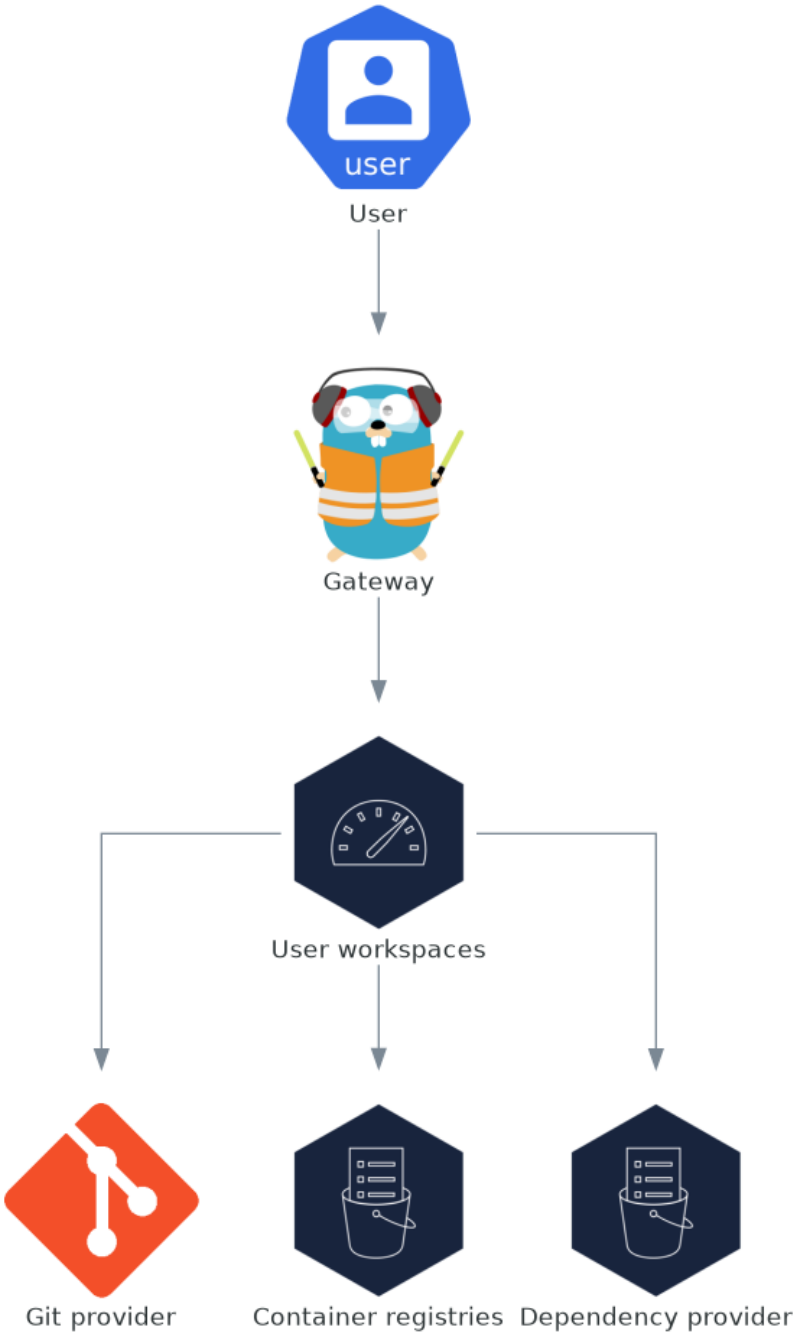
図1.13 プラグインレジストリーが他のコンポーネントと対話する

関連情報

- CodeReady Workspaces プラグインレジストリーリポジトリーのエディター定義
- CodeReady Workspaces プラグインレジストリーリポジトリーのプラグイン定義
- プラグインレジストリーの最新コミュニティーバージョンのオンラインインスタンス

1.6. ユーザーワークスペース

図1.14 ユーザーワークスペースの他のコンポーネントとの対話



ユーザーワークスペースは、コンテナーで実行される Web IDE です。

User ワークスペースは Web アプリケーションです。これは、ブラウザーで実行中の最新の IDE のすべてのサービスを提供するコンテナーで実行されるマイクロサービスで構成されます。

- エディター
- 言語の自動補完
- 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および有効にされたプラグイン、および関連する OpenShift コンポーネントが含まれる 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

- Secret

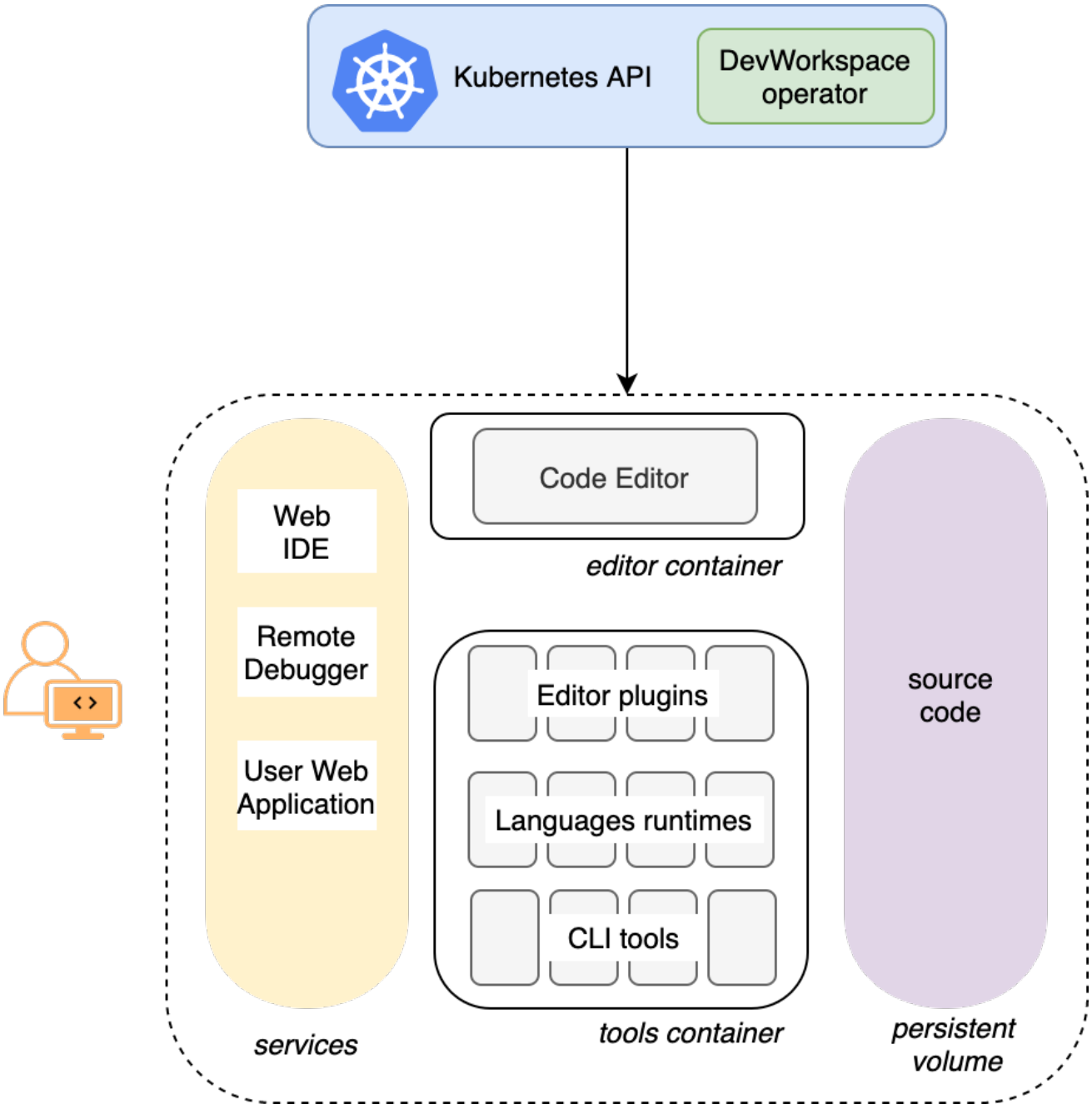
- 永続ボリューム (PV)

CodeReady Workspaces ワークスペースには、プロジェクトのソースコードが含まれ、OpenShift 永続ボリューム(PV)で永続化されます。マイクロサービスには、この共有ディレクトリーへの読み取り/書き込みアクセスがあります。

devfile v2 形式を使用して、CodeReady Workspaces 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、CodeReady Workspaces ワークスペースおよびそのコンポーネントを実行する 1 つを示しています。

図1.15 CodeReady Workspaces ワークスペースコンポーネント



この図には、ワークスペースが 1 つ実行されます。

第2章 CODEREADY WORKSPACES リソース要件の計算

関連情報

本セクションでは、Red Hat CodeReady Workspaces 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。

CodeReady Workspaces 中央コントローラーとユーザーワークスペースはいずれもコンテナーのセットで構成されます。これらのコンテナーは、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。

2.1. コントローラーの要件

Workspace コントローラーは、5 つの異なるコンテナーで実行される 5 つのサービスのセットで構成されます。以下の表は、これらの各サービスのデフォルトのリソース要件を示しています。

表2.1 ControllerServices

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
CodeReady Workspaces サーバーおよびダッシュボード	che	1 GiB	512 MiB
PostgreSQL	**postgres**	1 GiB	512 MiB
RH-SSO	**keycloak**	2 GiB	512 MiB
devfile レジストリー	**che-devfile-registry**	256 MiB	16 MiB
プラグインレジストリー	**che-plugin-registry**	256 MiB	16 MiB

これらのデフォルト値は、CodeReady Workspaces Workspace Controller が小規模な CodeReady Workspaces ワークスペースを管理する場合に問題なく使用できます。大規模なデプロイメントの場合は、メモリー制限を増やします。デフォルトの要求および制限を上書きする方法については、https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc の記事を参照してください。たとえば、https://workspaces.openshift.com で実行される Red Hat がホストする Eclipse Che は 1 GB のメモリーを使用します。

関連情報

- 「CodeReady Workspaces サーバーについて」.

2.2. 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、このワークスペースの各コンポーネントに必要なリソースの合計です。

以下の例は、適切な計算の必要性について示しています。

- アクティブな 10 のプラグインがあるワークスペースには、これより少ないプラグインを持つ同じワークスペースよりも多くのリソースが必要になります。
- 標準の Java ワークスペースでは、ビルド、テスト、およびアプリケーションのデバッグにより多くのリソースが必要になるため、標準の Node.js ワークスペースよりも多くのリソースが必要になります。

手順

1. https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc の components セクションで明示的に指定されるワークスペースコンポーネントを特定します。
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。
   a. CodeReady Workspaces はデフォルトの cheEditor: che-theia と、コマンドの実行を許可する chePlugin (che-machine-exec-plugin)を暗黙的に読み込みます。CodeReady Workspaces がマルチユーザーモードで実行されている場合は、cheEditor コンポーネントを読み込みます。
   b. JWT プロキシーコンポーネントは、ワークスペースコンポーネントの外部通信の認証および認可を行います。
3. 各コンポーネントの要件を計算します。
   a. デフォルト値:

デフォルト値:

以下の表は、すべてのワークスペースコンポーネントのデフォルト要件と、対応する CodeReady Workspaces サーバープロパティーを示しています。CodeReady Workspaces サーバープロパティーを使用して、デフォルトのクラスター全体を変更します。

表2.2 タイプ別のワークスペースコンポーネントのデフォルト要件

コンポーネントのタイプ	CodeReady Workspaces サーバープロパティー	デフォルトのメモリー制限	デフォルトのメモリー要求
chePlugin	che.workspace.sidecar.default_memory_limit_mb	128 MiB	64 MiB
cheEditor	che.workspace.sidecar.default_memory_limit_mb	128 MiB	64 MiB
kubernetes、openshift、dockerimage	che.workspace.default_memory_limit_mb, che.workspace.default_memory_request_mb	1 Gi	200 MiB
JWT プロキシー	che.server.secure_exposer.jwtproxy.memory_limit、che.server.secure_exposer.jwtproxy.memory_request	128 MiB	15 MiB

b. chePlugins および cheEditors コンポーネントのカスタム要件：

i. カスタムメモリー制限および要求:

meta.yaml ファイルの containers セクションの memoryLimit および memoryRequest 属性を定義して、chePlugins または cheEditors コンポーネントのメモリー制限を設定します。CodeReady Workspaces は、明示的に指定されていない場合に、メモリー制限に一致するようにメモリー要求を自動的に設定します。

> 例2.1 chePlugin che-incubator/typescript/latest
>
> meta.yaml 仕様セクション:

```
spec:
 containers:
   - image: docker.io/eclipse/che-remote-plugin-node:next
     name: vscode-typescript
     memoryLimit: 512Mi
     memoryRequest: 256Mi
```

これにより、コンテナーには以下のメモリー制限および要求が設定されます。

メモリー制限	512 MiB
メモリー要求	256 MiB

注記

IBM Power Systems (ppc64le) の場合は、一部のプラグインのメモリー制限が最大 1.5G まで増え、Pod が十分な RAM を実行できるようになりました。たとえば、IBM Power Systems (ppc64le) では、Theia エディター Pod には 2G が必要で、OpenShift コネクター Pod には 2.5G が必要です。AMD64 および Intel 64(x86_64)、 および IBM Z(s390x)の場合、メモリー要件は 512M と 1500M とそれぞれ低くなっています。ただし、一部の devfile は、AMD64 および Intel 64 (x86_64)、ならびに IBM Z (s390x) に有効である低い制限を設定するように設定されます。これを回避するため、デフォルトの memoryLimit を 1 - 1.5 GB 以上を増やすために、devfile を編集します。

注記

chePluginの meta.yaml ファイルの検索方法

コミュニティープラグイン は、v3/plugins/${organization}/${name}/${version}/ フォルダーの CodeReady Workspaces plug-ins registry repository で利用できます。

コミュニティー以外またはカスタマイズされたプラグインの場合、meta.yaml ファイルは ${pluginRegistryEndpoint}/v3/plugins/${organization}/${name}/${version}/meta.yaml のローカルの OpenShift クラスターで利用できます。

ii.

カスタム CPU の制限および要求:

CodeReady Workspaces は、デフォルトで CPU の制限および要求を設定しません。ただし、meta.yaml ファイルまたは devfile で chePlugin および cheEditor タイプの CPU 制限を、メモリー制限と同じ様に設定できます。

例2.2 chePlugin che-incubator/typescript/latest

meta.yaml 仕様セクション:

```
spec:
 containers:
   - image: docker.io/eclipse/che-remote-plugin-node:next
     name: vscode-typescript
     cpuLimit: 2000m
     cpuRequest: 500m
```

これにより、コンテナーに以下の CPU 制限および要求が設定されます。

CPU 制限	2 コア
CPU 要求	0.5 コア

CPU 制限および要求をグローバルに設定するには、以下の専用の環境変数を使用します。

CPU Limit	**CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__LIMIT__CORES**
CPU Request	**CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__REQUEST__CORES**

https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc も参照してください。

OpenShift プロジェクトの LimitRange オブジェクトは、クラスター管理者によって設定される CPU 制限および要求のデフォルト値を指定できることに注意してください。リソースのオーバーランによる開始エラーを防ぐために、アプリケーションやワークスペースレベルでの制限がこれらの設定に準拠している必要があります。

a. dockerimage コンポーネントのカスタム要件:

devfileのmemoryLimit属性とmemoryRequest属性を定義して、dockerimageコンテナのメモリ制限を設定します。CodeReady Workspaces は、明示的に指定されていない場合に、メモリー制限に一致するようにメモリー要求を自動的に設定します。

```
- alias: maven
  type: dockerimage
  image: eclipse/maven-jdk8:latest
  memoryLimit: 1536M
```

b. kubernetes または openshift コンポーネントのカスタム要件:

参照されるマニフェストは、メモリー要件および制限を定義できます。

1. 以前に計算された要件をすべて追加します。

関連情報

- 「CodeReady Workspaces ワークスペースアーキテクチャーについて」

2.3. ワークスペースの例

本セクションでは、CodeReady Workspaces ワークスペースの例を説明します。

以下の devfile は、CodeReady Workspaces ワークスペースを定義します。

```
apiVersion: 1.0.0
metadata:
  generateName: nodejs-configmap-
projects:
  - name: nodejs-configmap
    source:
```

```
      location: "https://github.com/crw-samples/nodejs-configmap.git"
      branch: 12.x
      type: git
components:
  - id: vscode/typescript-language-features/latest
    type: chePlugin
  - mountSources: true
    type: kubernetes
    entrypoints:
      - command:
          - sleep
        args:
          - infinity
    reference: 'https://raw.githubusercontent.com/crw-samples/nodejs-mongodb-
sample/master/kubernetes-manifests/guestbook-app.deployment.yaml'
    alias: guestbook-frontend
```

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2.3 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**（デフォルトの **cheEditor**）	512 MiB	512 MiB
ワークスペース	**machine-exec** (デフォルトの **chePlugin**)	128 MiB	32 MiB
ワークスペース	**vscode-typescript** (**chePlugin**)	512 MiB	512 MiB
ワークスペース	**nodejs** (**dockerimage**)	1 GiB	512 MiB
JWT プロキシー	verifier	128 MiB	128 MiB
	合計	2.25 GiB	1.38 GiB

- **devfile に含まれていない場合でも、theia-ide および machine-exec コンポーネントは暗黙的にワークスペースに追加されます。**
- **machine-exec で必要なリソースは、chePlugin のデフォルトです。**
-

theia-ide のリソースは、cheEditor meta.yaml で memoryLimit が 512 MiB に設定されます。

- Typescript VS Code 拡張は、デフォルトのメモリー制限もオーバーライドします。meta.yaml ファイルでは、制限は明示的に 512 MiB に指定されます。

- CodeReady Workspaces は、dockerimage コンポーネントタイプのデフォルト（メモリー制限 1 GiB とメモリー要求の 512 MiB) を適用します。

- JWT コンテナーには 128 MiB のメモリーが必要です。

すべての内容を追加すると、制限が 2.25 GiB の 1.38 GiB のメモリー要求が設定されます。

関連情報

- *1章 アーキテクチャーの概要*

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#configuring-the-che-installation.adoc

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc

- 「ユーザーの認証」

- CodeReady Workspaces プラグインレジストリーリポジトリー

第3章 レジストリーのカスタマイズ

本章では、CodeReady Workspaces のカスタムレジストリーをビルドし、実行する方法を説明します。

3.1. CODEREADY WORKSPACES レジストリーについて

CodeReady Workspaces は、プラグインレジストリーと devfile レジストリーの 2 つのレジストリーを使用します。これらは、CodeReady Workspaces プラグインおよび devfile のメタデータを公開する静的な Web サイトです。オフラインモードでビルドする場合には、アーティファクトも含まれます。

devfile およびプラグインレジストリーは、2 つの Pod で実行されます。これらのデプロイメントは、CodeReady Workspaces インストールに含まれます。

devfile およびプラグインレジストリー

devfile レジストリー

devfile レジストリーは、CodeReady Workspaces スタックの定義を保持します。スタックは、Create Workspace を選択すると CodeReady Workspaces ユーザーダッシュボードで利用できます。これには、サンプルプロジェクトを含む CodeReady Workspaces の技術的なスタックのサンプルの一覧が含まれます。オフラインモードでビルドする場合には、zip ファイルとして devfile で参照されるサンプルプロジェクトがすべて含まれます。

プラグインレジストリー

プラグインレジストリーを使用すると、CodeReady Workspaces の同じインスタンスのすべてのユーザーにプラグイン定義を共有できます。オフラインモードでビルドすると、すべてのプラグインまたは拡張アーティファクトも含まれます。

関連情報

- 「カスタムレジストリーイメージのビルド」
- 「カスタムレジストリーの実行」

3.2. カスタムレジストリーイメージのビルド

3.2.1. カスタム devfile レジストリーイメージのビルド

本セクションでは、カスタム devfile レジストリーイメージをビルドする方法を説明します。この手順では、devfile を追加する方法を説明します。このイメージには、devfile で参照されるすべてのサンプルプロジェクトが含まれます。

前提条件

- podman または docker の実行中のインストール。
- 追加する devfile の有効なコンテンツ。参照：https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc

手順

1. devfile レジストリー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。

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces
$ git checkout crw-2.14-rhel-8
```

2. ./dependencies/che-devfile-registry/devfiles/ ディレクトリーで、サブディレクトリー <devfile-name>/ を作成し、devfile.yaml ファイルおよび meta.yaml ファイルを追加します。

例3.1 devfile のファイル編成

```
./dependencies/che-devfile-registry/devfiles/
└── <devfile-name>
    ├── devfile.yaml
    └── meta.yaml
```

3. devfile.yaml ファイルに有効なコンテンツを追加します。devfile 形式の詳細については、https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc を参照してください。

4. meta.yaml ファイルが以下の構造に準拠していることを確認します。

表3.1 devfile meta.yaml のパラメーター

属性	説明
description	ユーザーダッシュボードに表示される説明。
displayName	ユーザーダッシュボードに表示される名前。
icon	ユーザーダッシュボードに表示される **.svg** ファイルへのリンク
tags	タグの一覧。タグには通常、スタックに含まれるツールが含まれます。
globalMemoryLimit	任意のパラメーター: devfile が起動するすべてのコンポーネントによって消費されることが予想されるメモリーの合計。この数字はユーザーダッシュボードに表示されます。これは情報を示唆するように提供されますが、CodeReady Workspaces サーバーでは考慮されません。

例3.2 devfile の例 meta.yaml

```
displayName: Rust
description: Rust Stack with Rust 1.39
tags: ["Rust"]
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
globalMemoryLimit: 1686Mi
```

5. **カスタム devfile レジストリーイメージをビルドします。**

```
$ cd dependencies/che-devfile-registry
$ ./build.sh --organization <my-org> \
        --registry <my-registry> \
        --tag <my-tag>
```

注記

build.sh スクリプトの詳細なオプションを表示するには --help パラメーターを使用します。

関連情報

- **https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#authoring-devfiles-version-2.adoc**
- **「カスタムレジストリーの実行」.**

3.2.2. カスタムプラグインレジストリーイメージのビルド

本セクションでは、カスタムプラグインレジストリーイメージをビルドする方法を説明します。この手順では、プラグインを追加する方法を説明します。イメージには、プラグインまたは拡張メタデータが含まれます。

前提条件

- Node.js 12.x
- yarn の実行中のバージョン。参照:Yarn のインストール。
- ./node_modules/.bin が PATH 環境変数にある。
- podman または docker の実行中のインストール。

手順

1. プラグインレジストリー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。

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces
$ git checkout crw-2.14-rhel-8
```

2. ./dependencies/che-plugin-registry/ ディレクトリーで、che-theia-plugins.yaml ファイルを編集します。

3. che-theia-plugins.yaml ファイルに有効なコンテンツを追加します。詳細は、https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#adding-a-vs-code-extension-to-the-che-plugin-registry.adoc を参照してください 。

4. カスタムプラグインレジストリーイメージをビルドします。

```
$ cd dependencies/che-plugin-registry
$ ./build.sh --organization <my-org> \
```

```
--registry <my-registry> \
--tag <my-tag>
```

注記

build.sh スクリプトの詳細なオプションを表示するには --help パラメーターを使用します。レジストリーイメージにプラグインバイナリーを含めるには、--offline パラメーターを追加します。

5. **レジストリーのビルド後にコンテナーに表示される ./dependencies/che-plugin-registry/v3/plugins/ の内容を確認します。正常なプラグインレジストリービルドから作成されるすべての meta.yaml ファイルがここにあります。**

```
./dependencies/che-plugin-registry/v3/plugins/
├── <publisher>
│   └── <plugin-name>
│       ├── latest
│       │   └── meta.yaml
│       └── latest.txt
```

関連情報

- 「カスタムレジストリーの実行」

3.3. カスタムレジストリーの実行

前提条件

このセクションで使用される my-plug-in-registry イメージおよび my-devfile-registry イメージは、docker コマンドを使用して構築されます。このセクションでは、これらのイメージが CodeReady Workspaces がデプロイされている OpenShift クラスターで利用できることを想定しています。

これらのイメージは以下にプッシュできます。

- **quay.io または DockerHub などのパブリックコンテナーレジストリー。**
- **プライベートレジストリー**

3.3.1. OpenShift でのレジストリーのデプロイ

手順

プラグインレジストリーをデプロイする OpenShift テンプレートは、GitHub リポジトリーの deploy/openshift/ ディレクトリーで利用できます。

1. OpenShift テンプレートを使用してプラグインレジストリーをデプロイするには、以下のコマンドを実行します。

```
NAMESPACE=<namespace-name> (1)
IMAGE_NAME="my-plug-in-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-plugin-registry.yml \
 -n "$\{NAMESPACE}" \
 -p IMAGE="$\{IMAGE_NAME}" \
 -p IMAGE_TAG="$\{IMAGE_TAG}" \
 -p PULL_POLICY="Always"
```

(1) crwctl を使用してインストールされている場合、デフォルトの CodeReady Workspaces プロジェクトは openshift-workspaces になります。この OperatorHub のインストール方法では、CodeReady Workspaces を現在のプロジェクトユーザーにデプロイします。

2. devfile レジストリーには、GitHub リポジトリーの deploy/openshift/ ディレクトリーに OpenShift テンプレートがあります。これをデプロイするには、以下のコマンドを実行します。

```
NAMESPACE=<namespace-name> (1)
IMAGE_NAME="my-devfile-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-devfile-registry.yml \
 -n "$\{NAMESPACE}" \
 -p IMAGE="$\{IMAGE_NAME}" \
 -p IMAGE_TAG="$\{IMAGE_TAG}" \
 -p PULL_POLICY="Always"
```

(1) crwctl を使用してインストールされている場合、デフォルトの CodeReady Workspaces プロジェクトは openshift-workspaces になります。この OperatorHub のインストール方法では、CodeReady Workspaces を現在のプロジェクトユーザーにデプロイします。

検証手順

1. **<plug-in> プラグインはプラグインレジストリーで利用できます。**

例3.3 プラグインレジストリー API を要求する <plug-in> を検索します。

```
$ URL=$(oc  get route -l app=che,component=plugin-registry \
  -o 'custom-columns=URL:.spec.host' --no-headers)
$ INDEX_JSON=$(curl -sSL http://${URL}/v3/plugins/index.json)
$ echo ${INDEX_JSON} | jq '.[] | select(.name == "<plug-in>")'
```

2. **<devfile> devfile は devfile レジストリーで利用できます。**

例3.4 devfile レジストリー API を要求する <devfile> を検索します。

```
$ URL=$(oc  get route -l app=che,component=devfile-registry \
  -o 'custom-columns=URL:.spec.host' --no-headers)
$ INDEX_JSON=$(curl -sSL http://${URL}/v3/plugins/index.json)
$ echo ${INDEX_JSON} | jq '.[] | select(.name == "<devfile>")'
```

3. **CodeReady Workspaces サーバーはプラグインレジストリーの URL を参照します。**

例3.5 che ConfigMap の CHE_WORKSPACE_PLUGIN__REGISTRY__URL パラメーターの値をプラグインレジストリールートの URL と比較します。

che ConfigMap の CHE_WORKSPACE_PLUGIN__REGISTRY__URL パラメーターの値を取得します。

```
$ oc  get cm/che \
  -o "custom-columns=URL:.data['CHE_WORKSPACE_PLUGIN__REGISTRY__URL']" \
  --no-headers
```

プラグインレジストリールートの URL を取得します。

```
$ oc  get route -l app=che,component=plugin-registry \
  -o 'custom-columns=URL: .spec.host' --no-headers
```

4. **CodeReady Workspaces サーバーは devfile レジストリーの URL を参照します。**

例3.6 che ConfigMap の CHE_WORKSPACE_DEVFILE__REGISTRY__URL パラメーターの値を devfile レジストリールートの URL と比較します。

che ConfigMap の CHE_WORKSPACE_DEVFILE__REGISTRY__URL パラメーターの値を取得します。

```
$ oc  get cm/che \
  -o "custom-columns=URL:.data['CHE_WORKSPACE_DEVFILE__REGISTRY__URL']" \
  --no-headers
```

devfile レジストリールートの URL を取得します。

```
$ oc  get route -l app=che,component=devfile-registry \
  -o 'custom-columns=URL: .spec.host' --no-headers
```

5. **値が一致しない場合は、ConfigMap を更新し、CodeReady Workspaces サーバーを再起動します。**

```
$ oc  edit cm/codeready
(...)
$ oc  scale --replicas=0 deployment/codeready
$ oc  scale --replicas=1 deployment/codeready
```

- **プラグインは以下で使用できます。**

- ワークスペースの詳細の Devfile タブの chePlugin コンポーネントの補完
- ワークスペースの Plugin Che-Theia ビュー

- devfile は、ユーザーダッシュボードの Create Workspace ページの Quick Add および Custom Workspace タブで利用できます。

3.3.2. 既存の CodeReady Workspaces ワークスペースでのカスタムプラグインレジストリーの追加

以下のセクションでは、既存の CodeReady Workspaces ワークスペースでのカスタムプラグインレジストリーを追加する 2 つの方法を説明します。

- コマンドパレットを使用したカスタムプラグインレジストリーの追加 - 新しいカスタムプラグインレジストリーを、コマンドパレットコマンドのテキスト入力を使用してすぐに追加します。この方法では、ユーザーがプラグインレジストリーの URL または名前などの既存の情報を編集することはできません。
- settings.json ファイルを使用してカスタムプラグインレジストリーを追加 - 新しいカスタムプラグインレジストリーを追加し、既存のエントリーを編集する場合。

3.3.2.1. コマンドパレットを使用してカスタムプラグインレジストリの追加

前提条件

- CodeReady Workspaces のインスタンス

手順

1. CodeReady Workspaces IDE で F1 キーを押してコマンドパレットを開くか、またはトップメニューで View → Find Command に移動します。

   コマンドパレット は、Ctrl+Shift+p (または macOS の Cmd+Shift+p) を押してアクティブにすることもできます。

2. Add Registry コマンドを検索ボックスに入力したら、Enter を一度押します。

3. 次の 2 つのコマンドプロンプトにレジストリー名とレジストリー URL を入力します。

- 新規プラグインレジストリーの追加後に、プラグイン ビューのプラグインの一覧が更新され、新規プラグインレジストリーが有効になっていないと、ユーザーに警告メッセージが表示されます。

3.3.2.2. settings.json ファイルを使用したカスタムプラグインレジストリーの追加

以下のセクションでは、メインの CodeReady Workspaces Settings メニューを使用して、settings.json ファイルを使用して新規プラグインレジストリーを編集し、追加する方法を説明します。

前提条件

- CodeReady Workspaces のインスタンス

手順

1. メインの CodeReady Workspaces 画面から、Ctrl+,を押すか、または左側のバーの歯車アイコンを使用して Open Preferences を選択します。

2. Che Plug-ins を選択し、Edit in setting.json リンクに進みます。

   setting.json ファイルが表示されます。

3. 以下のように chePlugins.repositories 属性を使用して、新しいプラグインレジストリーを追加します。

```
{
“application.confirmExit”: “never”,
“chePlugins.repositories”: {“test”: “https://test.com”}
}
```

4. 変更を保存し、既存の CodeReady Workspaces ワークスペースにカスタムプラグインレジストリーを追加します。

- 新たに追加されたプラグイン検証ツールは、**settings.json** ファイルの **chePlugins.repositories** フィールドに設定された **URL** 値の正確性をチェックします。

- 新規プラグインレジストリーの追加後に、プラグイン ビューのプラグインの一覧が更新され、新規プラグインレジストリーが有効になっていないと、ユーザーに警告メッセージが表示されます。このチェックは、コマンドパレットコマンド **Add plugin registry** を使用して追加されたプラグインにも機能します。

第4章 CODEREADY WORKSPACES ログの取得

CodeReady Workspaces で各種ログを取得する方法は、以下のセクションを参照してください。

- 「サーバーロギングの設定」
- 「OpenShift での OpenShift イベントへのアクセス」
- 「CodeReady Workspaces サーバーログの表示」
- 「外部サービスログの表示」
- 「プラグインブローカーログの表示」
- 「crwctl を使用したログの収集」

4.1. サーバーロギングの設定

CodeReady Workspaces 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。

CodeReady Workspaces 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。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、CodeReady Workspaces サーバーで個別のロガーのログレベルを設定できます。

4.1.1. ログレベルの設定

CHE_LOGGER_CONFIG プロパティーの値の形式は、コンマ区切りのキーと値のペアの一覧です。キーは、CodeReady Workspaces 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。

Operator ベースのデプロイメントでは、CHE_LOGGER_CONFIG 変数はカスタムリソースの customCheProperties の下に指定されます。

たとえば、以下のスニペットでは、WorkspaceManager が DEBUG ログメッセージを生成するようにします。

```
...
server:
  customCheProperties:
    CHE_LOGGER_CONFIG:
"org.eclipse.che.api.workspace.server.WorkspaceManager=DEBUG"
```

4.1.2. 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。

4.1.3. HTTP トラフィックのロギング

CodeReady Workspaces 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できます。これを実行するには、che.infra.request-logging ロガーを TRACE レベルに設定する必要があります。

```
...
server:
  customCheProperties:
    CHE_LOGGER_CONFIG: "che.infra.request-logging=TRACE"
```

4.2. OPENSHIFT での OPENSHIFT イベントへのアクセス

OpenShift プロジェクトのハイレベルのモニタリングについては、プロジェクトが実行する OpenShift イベントを表示します。

本セクションでは、OpenShift Web コンソールでこれらのイベントにアクセスする方法を説明します。

前提条件

- 実行中の OpenShift Web コンソール。

手順

1. OpenShift Web コンソールの左側のパネルで、Home → Events をクリックします。
2. 特定のプロジェクトのイベントの一覧を表示するには、一覧からプロジェクトを選択します。
3. 現在のプロジェクトのイベントの詳細が表示されます。

関連情報

- OpenShift イベントの一覧については、OpenShift ドキュメントのComprehensive List of Events in OpenShift documentationを参照してください。

4.3. OPENSHIFT 4 CLI ツールを使用した CODEREADY WORKSPACES クラスターデプロイメントの状態の表示

本セクションでは、OpenShift 4 CLI ツールを使用して CodeReady Workspaces クラスターのデプロイメントの状態を表示する方法を説明します。

前提条件

- OpenShift で実行している Red Hat CodeReady Workspaces のインスタンス。
- OpenShift コマンドラインツール oc のインストール

手順

1. 以下のコマンドを実行して crw プロジェクトを選択します。

```
$ oc project <project_name>
```

2. 以下のコマンドを実行して、選択したプロジェクトで実行されている Pod の名前およびステータスを取得します。

```
$ oc get pods
```

3. すべての **Pod** のステータスが **Running** であることを確認します。

例4.1 ステータスが Running の Pod

```
NAME                          READY     STATUS    RESTARTS   AGE
codeready-8495f4946b-jrzdc            0/1       Running   0          86s
codeready-operator-578765d954-99szc   1/1       Running   0          42m
keycloak-74fbfb9654-g9vp5       1/1       Running   0          4m32s
postgres-5d579c6847-w6wx5       1/1       Running   0          5m14s
```

4. **CodeReady Workspaces** クラスターデプロイメントの状態を表示するには、以下を実行します。

```
$ oc logs --tail=10 -f `(oc get pods -o name | grep operator)`
```

例4.2 Operator のログ:

```
time="2019-07-12T09:48:29Z" level=info msg="Exec successfully completed"
time="2019-07-12T09:48:29Z" level=info msg="Updating eclipse-che CR with status:
provisioned with OpenShift identity provider: true"
time="2019-07-12T09:48:29Z" level=info msg="Custom resource eclipse-che updated"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: ConfigMap, name:
che"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: ConfigMap, name:
custom"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: Deployment,
name: che"
time="2019-07-12T09:48:30Z" level=info msg="Updating eclipse-che CR with status:
CodeReady Workspaces API: Unavailable"
time="2019-07-12T09:48:30Z" level=info msg="Custom resource eclipse-che updated"
time="2019-07-12T09:48:30Z" level=info msg="Waiting for deployment che. Default
timeout: 420 seconds"
```

4.4. CODEREADY WORKSPACES サーバーログの表示

本セクションでは、コマンドラインを使用して **CodeReady Workspaces** サーバーログを表示する方法を説明します。

4.4.1. OpenShift CLI を使用した CodeReady Workspaces サーバーログの表示

本セクションでは、**OpenShift CLI (**コマンドラインインターフェース**)** を使用して **CodeReady Workspaces** サーバーログを表示する方法を説明します。

手順

1. ターミナルで以下のコマンドを実行し、**Pod** を取得します。

```
$ oc get pods
```

例

```
$ oc get pods
NAME           READY  STATUS   RESTARTS  AGE
codeready-11-j4w2b    1/1    Running  0         3m
```

2. デプロイメントのログを取得するには、以下のコマンドを実行します。

```
$ oc logs <name-of-pod>
```

例

```
$ oc logs codeready-11-j4w2b
```

4.5. 外部サービスログの表示

本セクションでは、**CodeReady Workspaces** サーバーに関連する外部サービスのログを表示する方法を説明します。

4.5.1. RH-SSO ログの表示

RH-SSO OpenID プロバイダーは、以下の **2** つの部分で構成されます。サーバーおよび **IDE**。これは診断またはエラー情報を複数のログに書き込みます。

4.5.1.1. RH-SSO サーバーログの表示

このセクションでは、RH-SSO OpenID プロバイダーサーバーのログを表示する方法について説明します。

手順

1. OpenShift Web コンソールで Deployments をクリックします。

2. Filter by label 検索フィールドに keycloak を入力し、RH-SSO ログを表示します。

.Deployment Configs セクションで、keycloak リンクをクリックしてこれを開きます。

1. History タブで、アクティブな RH-SSO デプロイメントについての View log リンクをクリックします。

2. RH-SSO ログが表示されます。

関連情報

- RH-SSO IDE サーバーに関連する診断およびエラーメッセージについては、「CodeReady Workspaces サーバーログの表示」を参照してください。

4.5.1.2. Firefox での RH-SSO クライアントログの表示

このセクションでは、Firefox WebConsole で RH-SSO IDE クライアント診断またはエラー情報を表示する方法を説明します。

手順

- Menu > WebDeveloper > WebConsole をクリックします。

4.5.1.3. Google Chrome での RH-SSO クライアントログの表示

このセクションでは、Google Chrome Console タブで RH-SSO IDE クライアントの診断またはエラー情報を表示する方法を説明します。

手順

1. Menu > More Tools > Developer Tools の順にクリックします。
2. Console タブをクリックします。

4.5.2. CodeReady Workspaces データベースログの表示

本セクションでは、PostgreSQL サーバーログなどのデータベースログを CodeReady Workspaces に表示する方法を説明します。

手順

1. OpenShift Web コンソールで Deployments をクリックします。
2. Find by label 検索フィールドに以下を入力します。
   - app=che および Enter を押してください。
   - component=postgres および Enter を押してください。

     OpenShift Web コンソールは、これら 2 つのキーでベースを検索し、PostgreSQL

ログを表示するようになりました。

3. postgres デプロイメントをクリックして開きます。

4. アクティブな PostgreSQL デプロイメントの View log リンクをクリックします。

OpenShift Web コンソールには、データベースログが表示されます。

関連情報

- PostgreSQL サーバーに関連する診断またはエラーメッセージは、アクティブな CodeReady Workspaces デプロイメントログにある場合があります。アクティブな CodeReady Workspaces デプロイメントログへのアクセスに関する詳細は、「CodeReady Workspaces サーバーログの表示」 セクションを参照してください。

4.6. プラグインブローカーログの表示

このセクションでは、プラグインブローカーログを表示する方法を説明します。

che-plugin-broker Pod 自体は作業が完了すると削除されます。そのため、イベントログはワークスペースの起動時にのみ利用できます。

手順

一時 Pod からのログイベントを表示するには、以下を実行します。

1. CodeReady Workspaces ワークスペースを起動します。

2. メインの OpenShift Container Platform 画面から、Workload → Pods に移動します。

3. Pod の Terminal タブにある OpenShift ターミナルコンソールを使用します。

検証手順

- ワークスペースの起動中に OpenShift ターミナルコンソールがプラグインブローカーログ

を表示します

4.7. CRWCTL を使用したログの収集

crwctl ツールを使用して OpenShift クラスターからすべての Red Hat CodeReady Workspaces ログを取得できます。

- crwctl server:deploy は、Red Hat CodeReady Workspaces のインストール時に Red Hat CodeReady Workspaces サーバーログの収集を自動的に開始します。
- crwctl server:logs は、既存の Red Hat CodeReady Workspaces サーバーログを収集します。
- crwctl workspace:logs はワークスペースログを収集します

第5章 CODEREADY WORKSPACES の監視

本章では、CodeReady Workspaces を設定してメトリクスを公開する方法と、CodeReady Workspaces でメトリクスとして公開されるデータを処理するために外部ツールを使用してモニタリングスタックのサンプルを構築する方法を説明します。

5.1. CODEREADY WORKSPACES メトリクスの有効化および公開

本セクションでは、CodeReady Workspaces メトリクスを有効にし、公開する方法を説明します。

手順

1. che-master ホストで 8087 ポートをサービスとして公開する CHE_METRICS_ENABLED=true 環境変数を設定します。

Red Hat CodeReady Workspaces が OperatorHub でインストールされると、デフォルトの CheCluster CR が使用されている場合に環境変数が自動的に設定されます。


```
spec:
  metrics:
    enable: true
```

5.2. PROMETHEUS を使用した CODEREADY WORKSPACES メトリクスの収集

本セクションでは、Prometheus モニタリングシステムを使用して、CodeReady Workspaces に関するメトリクスを収集し、保存し、クエリーする方法を説明します。

前提条件

- CodeReady Workspaces はポート 8087 でメトリクスを公開している。「CodeReady Workspaces メトリクスの有効化および公開」を参照してください。

- Prometheus 2.9.1 以降が実行中である。Prometheus コンソールは、対応する service と route のあるポート 9090 で実行されている。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

- 8087 ポートからメトリクスを収集するように Prometheus を設定する。

例5.1 Prometheus 設定の例

```
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'
        static_configs:
          - targets: ['[che-host]: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 (現時点ではシステムで使用されていません)。

3 Prometheus モニターするリソース。デフォルト設定では、che という単一ジョブは、CodeReady Workspaces サーバーによって公開される時系列データをスクレープします。

4

8087 ポートからメトリクスを収集します。

検証手順

- Prometheus コンソールを使用して、メトリクスをクエリーし、表示します。

  メトリックは、http://<che-server-url>:9090/metricsで入手できます。

  詳細は、「 式ブラウザーの使用 」を参照してください。

関連情報

- Prometheus での最初のステップ
- Prometheus の設定
- Prometheus のクエリー
- Prometheus メトリクスのタイプ

第6章 DEV WORKSPACE OPERATOR のモニタリング

本章では、Dev Workspace Operator によって公開されるメトリクスを処理するようにモニタリングスタックの例を設定する方法を説明します。本章の指示に従って、Dev Workspace Operator を有効にする必要があります。https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#enabling-dev-workspace-operator.adoc を参照してください。

6.1. PROMETHEUS を使用した DEV WORKSPACE OPERATOR メトリクスの収集

このセクションでは、Prometheus を使用して Dev Workspace Operator に関するメトリクスを収集し、保存し、クエリーする方法を説明します。

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ます。
- devworkspace-webhookserver サービスは、port 9443 でメトリクスを公開します。デフォルトで、サービスは port 9443 のメトリクスを公開します。
- Prometheus 2.26.0 以降が実行中である。Prometheus コンソールは、対応する service と route のあるポート 9090 で実行されている。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、Prometheus に関連付けられた ServiceAccount を devworkspace-controller-metrics-reader ClusterRole にバインドします。ClusterRoleBinding がないと、ロールベースのアクセス制御(RBAC)で保護されるため、Dev Workspace メトリクスにアクセスすることはできません。

例6.1 ClusterRole の例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: devworkspace-controller-metrics-reader
rules:
- nonResourceURLs:
  - /metrics
  verbs:
  - get
```

-

例6.2 ClusterRoleBinding の例

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: devworkspace-controller-metrics-binding
subjects:
  - kind: ServiceAccount
    name: <ServiceAccount name associated with the Prometheus Pod>
    namespace: <Prometheus namespace>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: devworkspace-controller-metrics-reader
```

2. devworkspace-controller-metrics サービスによって公開される 8443 ポート、および devworkspace- webhookserver サービスによって公開される 9443 ポートからメトリクスを収集するように Prometheus を設定します。

例6.3 Prometheus 設定の例

```
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: 'DevWorkspace'
        authorization:
          type: Bearer
          credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
        tls_config:
          insecure_skip_verify: true
        static_configs:
          - targets: ['devworkspace-controller-metrics:8443']  (4)
      - job_name: 'DevWorkspace webhooks'
        authorization:
          type: Bearer
          credentials_file: '/var/run/secrets/kubernetes.io/serviceaccount/token'
        tls_config:
          insecure_skip_verify: true
        static_configs:
          - targets: ['devworkspace-webhookserver:9443']  (5)
```

(1)

2

記録ルール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、2 つのジョブ（DevWorkspace およびDevWorkspace Webhook）は、dev workspace-controller-metrics および devworkspace-webhookserver サービスが公開する時系列データをスクレープします。

4

8443 ポートからメトリクスを収集します。

5

9443 ポートからメトリクスを収集します。

検証手順

- Prometheus コンソールを使用してターゲットおよびメトリクスを表示します。

  詳細は、「 式ブラウザーの使用 」を参照してください。

関連情報

- Prometheus での最初のステップ
- Prometheus の設定
- Prometheus のクエリー
- Prometheus メトリクスのタイプ

6.2. DEV WORKSPACE 固有のメトリクス

本セクションでは、dev workspace-controller-metrics サービスによって公開される Dev

Workspace 固有のメトリクスについて説明します。

表6.1 メトリック

Name (名前)	タイプ	説明	ラベル
devworkspace_started_total	カウンター	Dev Workspace の開始イベントの数。	**source**, **routingclass**
devworkspace_started_success_total	カウンター	**Running** フェーズに入る Dev Workspaces の数。	**source**, **routingclass**
devworkspace_fail_total	カウンター	失敗した Dev Workspaces の数。	**source**, **reason**
devworkspace_startup_time	ヒストグラム	Dev Workspace の起動にかかった合計時間（秒単位）。	**source**, **routingclass**

表6.2 ラベル

Name (名前)	説明	値
source	Dev Workspace の **controller.devfile.io/devworkspace-source** ラベル。	**文字列**
routingclass	Dev Workspace 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。	**"BadRequest|InfrastructureFailure|Unknown"**

表6.3 起動失敗の理由

Name (名前)	説明
BadRequest	Dev Workspace の作成に使用される無効な devfile が原因で起動に失敗します。
InfrastructureFailure	以下のエラーが原因で起動に失敗する。**CreateContainerError**, **RunContainerError**, **FailedScheduling**, **FailedMount**.
Unknown	不明な失敗の理由。

第7章 CODEREADY WORKSPACES のトレース

トレースは、マイクロサービスアーキテクチャーのレイテンシーの問題をトラブルシューティングするためにタイミングデータを収集するのに役立ち、分散システムで伝播されるため、完全なトランザクションまたはワークフローを理解するのに役立ちます。すべてのトランザクションでは、新規サービスが独立したチームによって導入されると、早い段階でパフォーマンスの異常を反映する可能性があります。

CodeReady Workspaces アプリケーションの追跡は、ワークスペースの作成、ワークスペースの起動、サブ操作の実行期間の分解、ボトルネックの特定、プラットフォーム全体の状態を改善など、さまざまな操作の実行を分析するのに役立ちます。

トレーサーはアプリケーションに存在します。これらは、発生する操作に関するタイミングとメタデータを記録します。多くの場合、それらはライブラリーをインストルメント化し、使用がユーザーに破棄されるようにします。たとえば、インストルメント化された Web サーバーは、要求の受信時や応答の送信時を記録します。収集されるトレースデータは スパン と呼ばれます。スパンには、トレースやスパン識別子などの情報や、次のステップに伝播できる他の種類の情報が含まれるコンテキストがあります。

7.1. トレース API

CodeReady Workspaces はインストルメント化に OpenTracing API (ベンダーに依存しないフレームワーク) を使用します。これは、開発者が別のトレースバックエンドを試す場合、新規の分散トレースシステムのインストルメンテーションプロセスを繰り返すのではなく、開発者は単にトレーサーのバックエンドの設定を変更できることを意味します。

7.2. バックエンドの追跡

デフォルトでは、CodeReady Workspaces は Jaeger をトレースバックエンドとして使用します。Jaeger は Dapper および OpenZipkin によって提供され、Uber Technologies によってオープンソースとしてリリースされた分散トレーシングシステムです。Jaeger は、大規模な要求およびパフォーマンスに対応する、より複雑なアーキテクチャーを拡張します。

7.3. JAEGER トレースツールのインストール

以下のセクションでは、Jaeger トレーシングツールのインストール方法を説明します。その後、Jaeger は CodeReady Workspaces でメトリクスを収集するために使用できます。

利用可能なインストール方法:

- 「OpenShift 4 での OperatorHub を使用した Jaeger のインストール」
- 「OpenShift 4 での CLI を使用した Jaeger のインストール」

Jaeger を使用して CodeReady Workspaces インスタンスをトレースするには、バージョン 1.12.0 以降が必要になります。Jaeger の詳細は、Jaeger の Web サイトを参照してください。

7.3.1. OpenShift 4 での OperatorHub を使用した Jaeger のインストール

このセクションでは、実稼働環境でテストおよび評価目的で Jaeger トレースツールを使用する方法についての情報を提供します。

OpenShift Container Platform の OperatorHub インターフェースから Jaeger トレースツールをインストールするには、以下の手順を実行します。

前提条件

- ユーザーが OpenShift Container Platform Web コンソールにログインしている。
- CodeReady Workspaces インスタンスはプロジェクトで利用できます。

手順

1. OpenShift Container Platform コンソールを開きます。
2. メインの OpenShift Container Platform 画面の左側のメニューから、Operators → OperatorHub に移動します。
3. Search by keyword 検索バーに、Jaeger Operator と入力します。
4. Jaeger Operator タイルをクリックします。

5. Jaeger Operator ポップアップウィンドウで Install ボタンをクリックします。

6. インストール方法を選択します。CodeReady Workspaces がデプロイされている クラスターの特定のプロジェクト であり、残りをデフォルト値のままにします。

7. Subscribe ボタンをクリックします。

8. メインの OpenShift Container Platform 画面の左側のメニューから、Operators → Installed Operators ページ に移動します。

9. Red Hat CodeReady Workspaces は、InstallSucceeded ステータスで示唆されるようにインストールされた Operator として表示されます。

10. インストールされた Operator の一覧で、 Jaeger Operator 名をクリックします。

11. Overview タブに移動します。

12. ページ下部の Conditions セクションで、メッセージ install strategy completed with no errors が表示されるのを待機します。

13. Jaeger Operator および追加の Elasticsearch Operator がインストールされています。

14. Operators → Installed Operators セクションに移動します。

15. インストールされた Operator の一覧で Jaeger Operator をクリックします。

16. Jaeger Cluster ページが表示されます。

17. ウィンドウの左下にある Create Instance をクリックします。

18. 保存 をクリックします。

19. OpenShift は Jaeger クラスター jaeger-all-in-one-inmemory を作成します。

20. メトリクスコレクションの有効化についての手順に従い、以下の手順を完了します。

7.3.2. OpenShift 4 での CLI を使用した Jaeger のインストール

このセクションでは、テストおよび評価の目的で Jaeger トレースツールを使用する方法について説明します。

OpenShift Container Platform の CodeReady Workspaces プロジェクトから Jaeger トレースツールをインストールするには、本セクションの手順に従います。

前提条件

- ユーザーが OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Container Platform クラスターの CodeReady Workspaces のインスタンス。

手順

1. OpenShift Container Platform クラスターの CodeReady Workspaces インストールプロジェクトで、oc クライアントを使用して Jaeger デプロイメントの新規アプリケーションを作成します。

```
$ oc new-app -f / ${CHE_LOCAL_GIT_REPO}/deploy/openshift/templates/jaeger-all-in-one-template.yml:

--> Deploying template "<project_name>/jaeger-template-all-in-one" for "/home/user/crw-projects/crw/deploy/openshift/templates/jaeger-all-in-one-template.yml" to project <project_name>

    Jaeger (all-in-one)
    ---------
    Jaeger Distributed Tracing Server (all-in-one)

    * With parameters:
      * Jaeger Service Name=jaeger
      * Image version=latest
```

```
        * Jaeger Zipkin Service Name=zipkin

--> Creating resources ...
    deployment.apps "jaeger" created
    service "jaeger-query" created
    service "jaeger-collector" created
    service "jaeger-agent" created
    service "zipkin" created
    route.route.openshift.io "jaeger-query" created
--> Success
    Access your application using the route: 'jaeger-query-<project_name>.apps.ci-ln-
whx0352-d5d6b.origin-ci-int-aws.dev.rhcloud.com'
    Run 'oc status' to view your app.
```

2. メインの OpenShift Container Platform 画面の左側のメニューから Workloads → Deployments を使用して、Jaeger デプロイメントが正常に終了するまで監視します。

3. メインの OpenShift Container Platform 画面の左側のメニューから Networking → Routes を選択し、URL リンクをクリックして Jaeger ダッシュボードにアクセスします。

4. メトリクスコレクションの有効化についての手順に従い、以下の手順を完了します。

7.4. メトリクス収集の有効化

前提条件

- Jaeger v1.12.0 以降がインストールされている。「Jaeger トレースツールのインストール」 の手順を参照してください。

手順

Jaeger トレースを機能させるには、CodeReady Workspaces デプロイメントで以下の環境変数を有効にします。

```
# Activating CodeReady Workspaces tracing modules
CHE_TRACING_ENABLED=true

# Following variables are the basic Jaeger client library configuration.
JAEGER_ENDPOINT="http://jaeger-collector:14268/api/traces"

# Service name
JAEGER_SERVICE_NAME="che-server"

# URL to remote sampler
JAEGER_SAMPLER_MANAGER_HOST_PORT="jaeger:5778"
```

```
# Type and param of sampler (constant sampler for all traces)
JAEGER_SAMPLER_TYPE="const"
JAEGER_SAMPLER_PARAM="1"

# Maximum queue size of reporter
JAEGER_REPORTER_MAX_QUEUE_SIZE="10000"
```

次の環境変数を有効にするには、以下を実行します。

1. CodeReady Workspaces デプロイメントの yaml ソースコードで、spec.server.customCheProperties に以下の設定変数を追加します。

```
customCheProperties:
    CHE_TRACING_ENABLED: 'true'
    JAEGER_SAMPLER_TYPE: const
    DEFAULT_JAEGER_REPORTER_MAX_QUEUE_SIZE: '10000'
    JAEGER_SERVICE_NAME: che-server
    JAEGER_ENDPOINT: 'http://jaeger-collector:14268/api/traces'
    JAEGER_SAMPLER_MANAGER_HOST_PORT: 'jaeger:5778'
    JAEGER_SAMPLER_PARAM: '1'
```

2. JAEGER_ENDPOINT の値を編集して、デプロイメントの Jaeger コレクターサービスの名前に一致するようにします。

メインの OpenShift Container Platform 画面の左側のメニューから、Networking → Services に移動して JAEGER_ENDPOINT の値を取得します。または、以下の oc コマンドを実行します。

```
$ oc get services
```

要求された値は collector 文字列が含まれるサービス名に含まれます。

関連情報

- カスタム環境プロパティーや CheCluster カスタムリソースでの定義方法に関する詳細は、https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc を参照してください。

- Jaeger のカスタム設定については、Jaeger クライアント環境変数の一覧を参照してください。

7.5. JAEGER UI での CODEREADY WORKSPACES トレースの表示

このセクションでは、Jaeger UI を使用して CodeReady Workspaces 操作の追跡についての概要を示す方法を説明します。

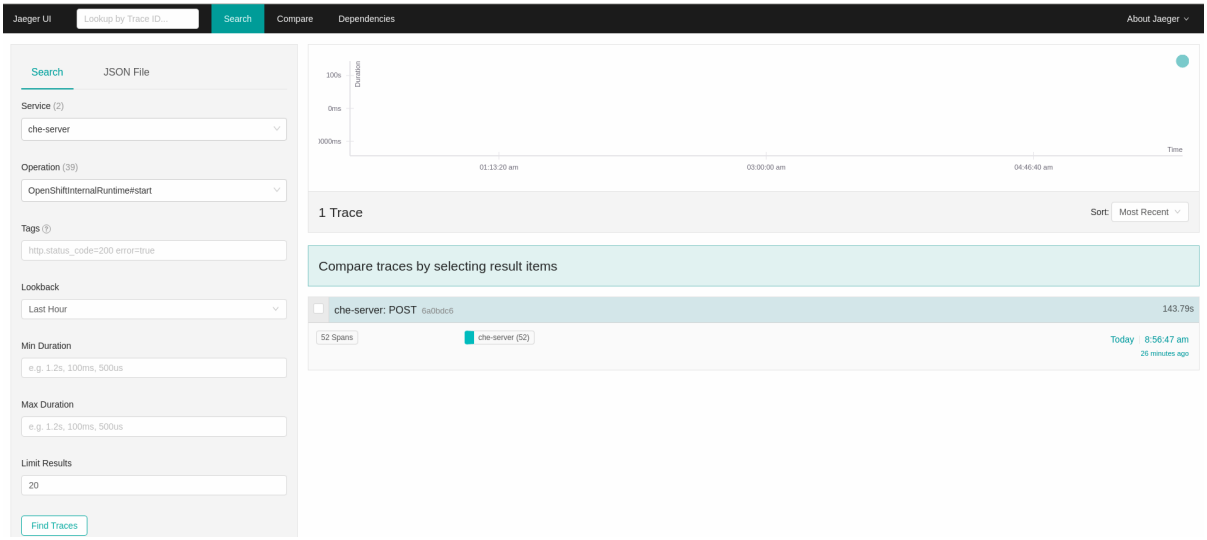
手順

この例では、CodeReady Workspaces インスタンスはしばらく実行されており、1 つのワークスペースが起動しています。

ワークスペース開始のトレースを検査するには、以下を実行します。

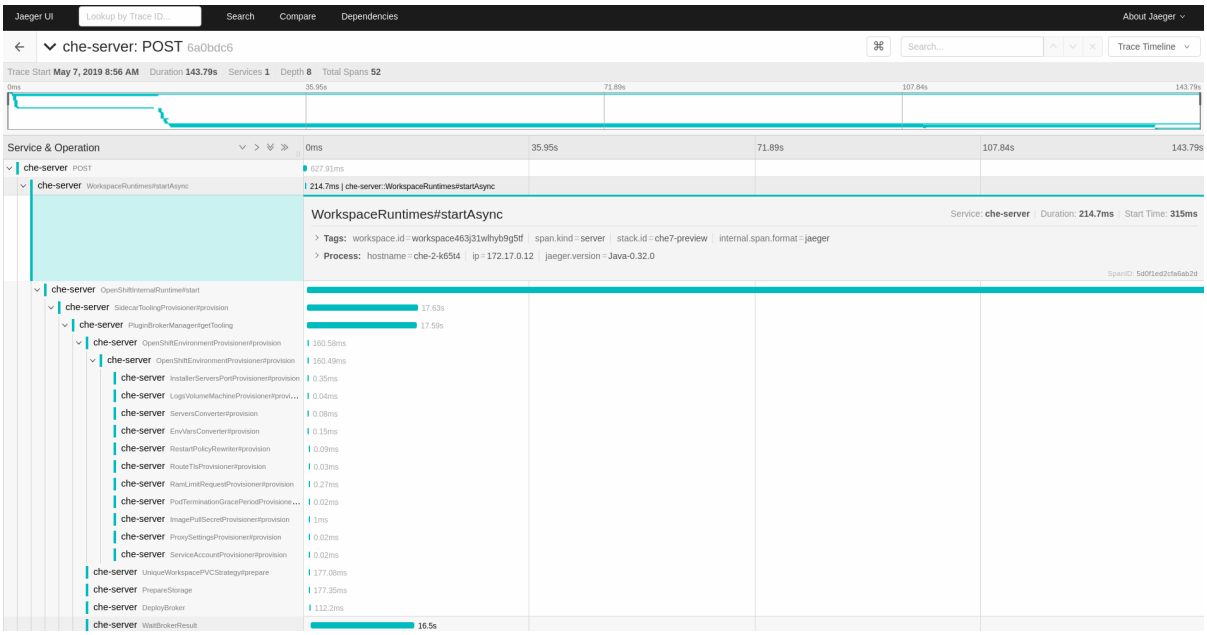
1. 左側の Search パネルで、操作名（スパン名)、タグ、または期間でスパンをフィルターします。

図7.1 Jaeger UI を使用した CodeReady Workspaces の追跡



2. トレースを選択して拡張し、ネストされたスパンのツリーと、タグや期間などの強調表示されたスパンに関する追加情報を表示します。

図7.2 拡張されたトレースツリー



7.6. CODEREADY WORKSPACES トレーシングコードベースの概要および拡張ガイド

CodeReady Workspaces のトレース実装のコアの部分は、che-core-tracing-core および che-core-tracing-web モジュールにあります。

トレース API へのすべての HTTP 要求には独自のトレースがあります。これは、サーバーアプリケーション全体にバインドされる OpenTracing ライブラリー から TracingFilter で実行されます。@Traced アノテーションをメソッドに追加すると、TracingInterceptor はトレーススパンを追加します。

7.6.1. タグ付け

スパンには、操作名、スパンの起点、エラー、およびユーザーのスパンのクエリーやフィルターに役立つその他のタグなど、標準のタグが含まれる場合があります。ワークスペース関連の操作（ワークスペースの開始または停止など）には、userId、workspaceID、stackId などの追加のタグが使用されます。TracingFilter によって作成されたスパンには、HTTP ステータスコードタグもあります。

トレースメソッドでのタグの宣言は、TracingTags クラスからフィールドを設定して静的に実行されます。

```
TracingTags.WORKSPACE_ID.set(workspace.getId());
```

TracingTags は、それぞれの AnnotationAware タグ実装のように、一般的に使用されるすべてのタグが宣言されるクラスです。

関連情報

- Jaeger UI の使用方法に関する詳細は、Jaeger ドキュメントを参照してください。Jaeger スタートガイド

第8章 バックアップおよび障害復旧

CodeReady Workspaces Operator は、CodeReady Workspaces インスタンスのバックアップを作成し、必要な場合はバックアップスナップショットから復元できます。本章では、このようなバックアップを準備し、そのバックアップの使用方法をフォローアップの復旧フェーズで使用する方法を説明します。

- 「crwctl を使用したバックアップの管理」
- 「カスタムリソースを使用したバックアップの管理」

注意

- CodeReady Workspaces の標準バックアップメカニズムは、ユーザーのワークスペースの内容をバックアップしません。ローカルの変更を保持するには、「永続ボリュームのバックアップ」 を参照してください。
- バックアップスナップショットは独自のクラスターにバインドされ、そこのみを使用する必要があります。
- CodeReady Workspaces Operator は、すべての CodeReady Workspaces 更新に新しいバックアップを作成します。
- 設定済みの バックアップサーバーは、バックアップの保存に自動的に使用されます。
- CodeReady Workspaces 管理者が複数のバックアップサーバーを設定する場合、CodeReady Workspaces Operator はデフォルトでサーバーを che.eclipse.org/backup-before-update: true アノテーションで使用します。
- CodeReady Workspaces Operator は内部バックアップサーバーを使用します。
  - CodeReady Workspaces 管理者がバックアップサーバーを設定しない時

- 複数のバックアップサーバーにアノテーションがない場合

関連情報

- 「外部データベースの設定」

8.1. バックアップサーバーの設定

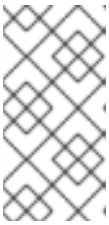
以下のセクションでは、サポートされる CodeReady Workspaces バックアップサーバーを説明し、その設定に関する情報を提供します。

注記

- Red Hat CodeReady Workspaces Operator は、同じクラスター内でバックアップサーバーを自動的に設定できますが、実稼働環境での使用には推奨されません。
- CodeReady Workspaces インストールと同じ OpenShift プロジェクト内のデータをバックアップするという決定に起因する制限に同意したユーザーは、このセクションをスキップできます。

CodeReady Workspaces は restic ツールを使用して以下を行うようにします。

- バックアップスナップショットの管理
- バックアップサーバーからバックアップデータをプッシュ、またはバックアップサーバーにプルします。



注記

restic バックアップツールは、BSD 2-Clause ライセンスでライセンスが適用されています。

現在、CodeReady Workspaces でバックアップサーバーがサポートされています。

REST

REST サーバーは、restic ツールと連携するソリューションです。REST サーバーのセットアップ方法を参照してください。

Amazon S3 および API と互換性のある代替手段

AWS S3 Simple Storage Serviceのドキュメント、またはAWSと互換性のあるAPIを持つ代替サービスのドキュメントを参照してください。

SFTP

How to configure an SFTP serverを参照してください。

8.2. CRWCTL を使用したバックアップの管理

次のセクションでは、CodeReady Workspacesインストールのバックアップを作成および使用して、crwctlを使用して以前のバージョンへのリカバリまたはロールバックを実行する方法について説明します。

前提条件

- Red Hat CodeReady Workspaces Operator は、同じクラスター内でバックアップサーバーを自動的に設定できますが、実稼働環境での使用には推奨されません。
- CodeReady Workspaces インストールと同じ OpenShift プロジェクト内のデータをバックアップするという決定に起因する制限に同意したユーザーは、このセクションをスキップできます。

- バックアップサーバーを設定する
- バックアップサーバーを使用するように crwctl を設定する

手順

- 「新規バックアップの作成」
- 「バックアップからの復元」

8.2.1. 新規バックアップの作成

1. バックアップスナップショットを作成して、事前に設定されたバックアップサーバーに送信するには、以下を実行します。

```
$ crwctl server:backup --repository-url=<repository-url> --repository-password=<repository-password>
```

- 引数なしで server:backup コマンドを使用して、同じバックアップサーバーに他のバックアップを作成できます。

- 初めて引数のない server:backup コマンドを使用すると、内部バックアップサーバーを設定して使用します。

8.2.2. バックアップからの復元

CodeReady Workspaces 管理者は、特定の CodeReady Workspaces バージョンの既存のスナップショットを使用して、必要な状態またはバージョンを復元できます。次の手順は、復元コマンドについていくつかのバリエーションを説明します。ユースケースに合わせて、コマンド引数を調整します。

- 同じバージョンの CodeReady Workspaces の以前の機能状態を復元するには、以下を実行します。

```
$ crwctl server:restore --repository-url=<repository-url> --repository-password=<repository-password> --snapshot-id=<snapshot-id>
```

- 現行バージョンの CodeReady Workspaces とは異なるバージョンにロールバックするには、以下を実行します。

```
$ crwctl server:restore --version=<version> --snapshot-id=<snapshot-id> --repository-url=<repository-url> --repository-password=<repository-password>
```

これにより、バージョンの CodeReady Workspaces から作成されたスナップショットがロールバックを実行し、復元されます。指定したスナップショットは、ロールバックする CodeReady Workspaces のバージョンから作成する必要があります。

注記

各 CodeReady Workspaces バージョンに専用のバックアップリポジトリーがあり、バージョンに最新のバックアップを使用する必要がある場合は、最新の 引数をスナップショット ID として指定できます。これにより、最新の 引数は指定されたリポジトリーの最新の既知の ID に変換されます。これは、CodeReady Workspaces Operator によってリカバリーに使用されます。

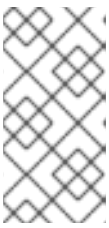
- 既存のバックアップカスタムリソースによって記述される状態を復元するには、以下を実行します。

```
$ crwctl server:restore --backup-cr-name=<CheClusterBackupCRName>
```

- CodeReady Workspaces のバージョンアップグレードをロールバックするには、以下を実行します。

```
$ crwctl server:restore --rollback
```

これにより、CodeReady Workspaces が後続バージョンにアップグレードする前に使用したバージョンを復元します。



注記

CodeReady Workspaces Operator は、アップグレードごとにバックアップを自動的に作成します。

8.3. バックアップサーバーを使用するように CRWCTL を設定

次のセクションでは、crwctl ツールを使用して、特定のバックアップサーバーの環境変数を定義する方法を説明します。

手順

1. バックアップサーバータイプとサーバー URL を確認します。restic リポジトリードキュメント は参考として使用してください。

URL は -r パラメーターで指定するか、BACKUP_REPOSITORY_URL 環境変数を使用して定義できます。

2. バックアップリポジトリーのパスワードを取得または作成します。

パスワードは、-p パラメーターで指定するか、BACKUP_REPOSITORY_PASSWORD 環境変数を使用して定義できます。

警告

バックアップデータは、このパスワードで暗号化されます。バックアップリポジトリーのパスワードが失われると、データが失われます。

3. 選択した バックアップサーバータイプ に、以下の環境変数を設定します。

REST

任意の認証が有効な場合は、REST_SERVER_USERNAME および REST_SERVER_PASSWORD 環境変数をエクスポートします。

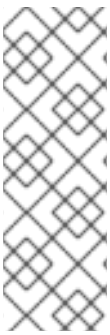
AWS S3

AWS ユーザー認証情報で AWS_ACCESS_KEY_ID および AWS_SECRET_ACCESS_KEY 環境変数をエクスポートします。

SFTP

パスワードなしのログインの場合は、対応する SSH 鍵を持つファイルへのパスを保持する SSH_KEY_FILE 環境変数をエクスポートするか、--ssh-key-file パラメーターを指定します。

または、SSH キー自体を保持する SSH_KEY 環境変数を使用することもできます。



注記

--backup-server-config-name パラメーターまたは BACKUP_SERVER_CONFIG_NAME 環境変数を使用して、バックアップサーバー設定オブジェクトを直接参照することができます。この場合、上記のすべての設定は必要ありません。詳細は、「カスタムリソースを使用したバックアップの管理」 を参照してください。

8.4. カスタムリソースを使用したバックアップの管理

以下のセクションでは、CodeReady Workspaces インストールのバックアップを作成し、カスタムリソースオブジェクトを使用して直接復元する方法を説明します。

注記

- Red Hat CodeReady Workspaces Operator は、同じクラスター内でバックアップサーバーを自動的に設定できますが、実稼働環境での使用には推奨されません。
- CodeReady Workspaces インストールと同じ OpenShift プロジェクト内のデータをバックアップするという決定に起因する制限に同意したユーザーは、このセクションをスキップできます。

前提条件

- バックアップサーバーの設定
- バックアップサーバーを使用するように Red Hat CodeReady Workspaces を設定

手順

- 「新規バックアップの作成」
- 「バックアップからの復元」

8.4.1. 新規バックアップの作成

1. CheClusterBackup オブジェクトを作成し、新規バックアップを作成します。

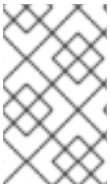
```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: CodeReady Workspaces-backup
spec:
  backupServerConfigRef: backup-server-configuration (1)
```

1

使用するバックアップサーバーを定義する CheBackupServerConfiguration オブジェクトの名前。

- CheClusterBackup オブジェクトの作成は、新規バックアップを開始します。
- 新規バックアップオブジェクトに同じ名前を再利用する前に、古いオブジェクトを削除します。

```
oc delete CheClusterBackup <name> -n openshift-workspaces
```



注記

CheClusterBackup オブジェクトの編集は機能しません。

代替方法

内部バックアップサーバーを使用するには、CodeReady Workspaces Operator から自動設定を要求します。上記の準備は必要ありません。

- 自動設定を構成し、バックアップを内部バックアップサーバーに送信します。

```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: CodeReady Workspaces-backup
spec:
  useInternalBackupServer: true
```

8.4.2. バックアップからの復元

注記

本章で説明する方法は、異なるバージョンの CodeReady Workspaces への復元には使用できません。CodeReady Workspaces を別のバージョンに復元するには、crwctl ツールを使用します。詳細は、「crwctl を使用したバックアップの管理」 の章を参照してください。

1. CheClusterRestore の新規オブジェクトを作成し、バックアップから CodeReady Workspaces インストールを復元します。

```
apiVersion: org.eclipse.che/v1
kind: CheClusterRestore
metadata:
  name: CodeReady Workspaces-restore
spec:
  backupServerConfigRef: backup-server-configuration (1)
  snapshotId: ba92c7e0 (2)
```

(1) 使用するバックアップサーバーを定義する CheBackupServerConfiguration オブジェクトの名前。

(2) 復元元となるスナップショット ID を定義するオプションパラメーター。デフォルト値は、バックアップサーバーの最後のスナップショットです。

1. 新規リカバリーを要求する新しい CheClusterRestore オブジェクトを作成します。

- 新規バックアップオブジェクトに同じ名前を再利用する前に、古いオブジェクトを削除します。

```
oc delete CheClusterBackup <name> -n openshift-workspaces
```

2. リカバリープロセスが完了するまで待ちます。

リカバリー後にブラウザーでエラーが発生した場合は、CodeReady Workspaces ドメインのブラウザーデータをクリーンアップします。

注記

CheClusterRestore オブジェクトの編集は機能しません。

検証

1.

バックアッププロセスの状態を確認します。

a.

CheClusterBackup オブジェクトの status セクションを読み取り、バックアッププロセスを確認します。

```
status:
  message: 'Backup is in progress. Start time: <timestamp>' 1
  stage: Collecting CodeReady Workspaces installation data 2
  state: InProgress 3
  snapshotId: ba92c7e0 4
```

1

全体の状態またはエラーメッセージを表示します。

2

バックアッププロセスの現在のフェーズは、人間が判読できる形式で行います。

3

バックアッププロセスの状態。InProgress、Succeeded、または Failed のいずれか。

4

作成されたバックアップスナップショットの ID。フィールドは、state が Succeeded の場合にのみ表示されます。

2.

復元プロセスの状態の確認

a.

CheClusterRestore オブジェクトの status セクションを読み取り、復元プロセスを確認します。

```
status:
  message: 'Restore is in progress. Start time: <timestamp>' 1
  stage: Restoring CodeReady Workspaces related cluster objects 2
  state: InProgress 3
```

1

2

3

復元プロセスの状態。InProgress、Succeeded、または Failed のいずれか。

8.5. バックアップサーバーを使用するように CODEREADY WORKSPACES を設定

CodeReady Workspaces のバックアップサーバーを設定するには、ユーザーは openshift-workspaces namespace に CheBackupServerConfiguration カスタムリソースオブジェクトを作成する必要があります。オブジェクトの spec プロパティーは複数のセクションに分割され、それぞれは特定の バックアップサーバータイプ に対応します。

- REST
- AWS S3 または API 互換
- SFTP

注記

- openshift-workspaces namespace に保存されるカスタムリソースオブジェクトには、spec プロパティーに 1 つのセクションのみを設定する必要があります。
- 必要な数だけバックアップサーバーを設定できますが、各バックアップサーバーは個別の Custom Resource で設定できます。
- 各サーバータイプの参照シークレットが存在し、必須フィールドを指定する必要があります。対応するサーバータイプの章で各シークレットの説明を参照してください。

8.5.1. REST サーバーの設定

```
apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
```

```
metadata:
  name: backup-server-configuration
spec:
  rest:
    protocol: http                                                    (1)
    hostname: my-domain.net                                           (2)
    port: 1234                                                        (3)
    repositoryPath: CodeReady Workspaces-backups                      (4)
    repositoryPasswordSecretRef: backup-encryption-password-secret    (5)
    credentialsSecretRef: rest-server-auth-secret                     (6)
```

(1) 使用するプロトコルを指定する任意のプロパティー。デフォルト値はhttpsで、2番目に許可されるオプションはhttpです。

(2) バックアップサーバーのホスト名。

(3) バックアップサーバーが実行しているポートを指定する任意のプロパティー。デフォルト値は8000です。

(4) バックアップスナップショットが保存されるバックアップサーバーのパス。

(5) repo-password フィールドに保存されるリポジトリーパスワードを含むシークレット名。シークレットに 1 つのフィールドのみが含まれる場合、その名前は任意です。パスワードは、バックアップスナップショットデータを暗号化および復号するために使用されます。

(6) username フィールドおよび password フィールドに保存されている REST サーバーユーザー認証情報でシークレットの名前を指定する任意のプロパティー。

8.5.2. AWS S3 または API 互換サーバーの設定

```
apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
spec:
  awss3:
```

```
protocol: https                                                    1
hostname: my-domain.net                                              2
port: 1234                                                         3
repositoryPath: CodeReady Workspaces-backups                                 4
repositoryPasswordSecretRef: backup-encryption-password-secret   5
awsAccessKeySecretRef: aws-user-credentials-secret                 6
```

1

使用するプロトコルを指定する任意のプロパティー。デフォルト値はhttpsで、2番目に許可されるオプションはhttpです。

2

S3 ホスト名を指定する任意のプロパティー。デフォルト値は s3.amazonaws.com です。

3

バックアップサーバーが実行しているポートを指定する任意のプロパティー。

4

バックアップスナップショットが保存されるバケットリソースの名前。バケットリソースは手動で事前に作成する必要があります。

5

repo-password フィールドに保存されるリポジトリーパスワードが含まれるシークレットの名前。シークレットに 1 つのフィールドのみが含まれる場合、この名前は任意です。パスワードは、バックアップスナップショットデータを暗号化および復号するために使用されます。

6

awsAccessKeyId フィールドおよび awsSecretAccessKey フィールドに保存されているユーザー認証情報が含まれるシークレットの名前。

8.5.3. SFTP サーバーの設定

```
apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
spec:
  awss3:
    username: user                                                 1
    hostname: my-domain.net                                          2
    port: 1234                                                     3
```

```
repositoryPath: CodeReady Workspaces-backups                    4
repositoryPasswordSecretRef: backup-encryption-password-secret  5
sshKeySecretRef: ssh-key-secret                                 6
```

1

SSH プロトコルを使用してログインするためのリモートサーバーのユーザー名。

2

リモートサーバーのホスト名。

3

SFTP サーバーが実行しているポートを指定する任意のプロパティー。デフォルト値は22です。

4

バックアップスナップショットが保存されるサーバー上の絶対パスまたは相対パス。

5

repo-password フィールドに保存されるリポジトリーパスワードが含まれるシークレットの名前。シークレットに 1 つのフィールドのみが含まれる場合、この名前は任意です。パスワードは、バックアップスナップショットデータを暗号化および復号するために使用されます。

6

ssh-privatekey フィールドに保存される SSH 秘密鍵が含まれるシークレットの名前。この SSH キーは、SFTP サーバーにパスワードなしでログインを実行するために使用できます。

8.6. 永続ボリュームのバックアップ

永続ボリューム (PV) は、ローカルのハードドライブのデスクトップ IDE 用にワークスペースのデータを保存する方法と同様に、CodeReady Workspaces ワークスペースデータを保存します。

データの損失を防ぐには、PV を定期的にバックアップします。PV を含む OpenShift リソースのバックアップおよび復元には、ストレージに依存しないツールを使用することが推奨されます。

8.6.1. 推奨されるバックアップツール：Velero

Velero は、OpenShift アプリケーションおよびそれらの PV をバックアップするオープンソース

ツールです。Velero を使用すると、以下を実行できます。

- クラウドまたはオンプレミスでデプロイします。
- データ損失が発生した場合にクラスターをバックアップし、復元します。
- クラスターリソースを他のクラスターに移行します。
- 実稼働クラスターを開発およびテスト用に複製します。

注記

または、基礎となるストレージシステムに依存するバックアップソリューションを使用できます。たとえば、Gluster や Ceph 固有のソリューションなどがこれに含まれます。

関連情報

- 永続ボリュームのドキュメント
- Gluster ドキュメント
- Ceph ドキュメント
- Velero on GitHub

8.7. 外部データベースの設定

PostgreSQL データベースは、CodeReady Workspaces の状態に関するデータを永続化させるために、CodeReady Workspaces サーバーによって使用されます。これには、ユーザーアカウント、ワークスペース、設定についての情報、およびその他の詳細情報が含まれます。

デフォルトで、CodeReady Workspaces Operator はデータベースデプロイメントを作成し、管理

します。

ただし、CodeReady Workspaces Operator はバックアップやリカバリーなどの完全なライフサイクル機能をサポートしません。

ビジネスに不可欠な環境では、以下の推奨される障害復旧オプションを使用して外部データベースを設定します。

- 高可用性 (HA)
- PITR (Point In Time Recovery)

オンプレミスの外部 PostgreSQL インスタンスを設定するか、または Amazon Relational Database Service (Amazon RDS) などのクラウドサービスを使用します。Amazon RDS を使用すると、通常の、およびオンデマンドのスナップショットを使用して、回復性のある障害復旧ストラテジーの Multi-Availability Zone 設定で実稼働データベースをデプロイできます。

データベースサンプルの設定例は以下のようになります。

パラメーター	値
インスタンスクラス	db.t2.small
vCPU	1
RAM	2 GB
Multi-az	true、2 つのレプリカ
エンジンのバージョン	9.6.11
TLS	enabled
自動化されたバックアップ	有効 (30 日)

8.7.1. 外部 PostgreSQL の設定

外部 PostgreSQL を設定すると、ワークスペースメタデータおよびユーザー情報を永続化できま

す。

手順

1. 以下のプレースホルダーの値を定義します。

   - <database-user> は、CodeReady Workspaces サーバーデータベースのユーザー名です。
   - <database-password> は、CodeReady Workspaces サーバーデータベースのパスワードです。
   - <database> は、CodeReady Workspaces サーバーデータベースの名前です。

2. 以下の SQL スクリプトを使用して、CodeReady Workspaces サーバーのユーザーおよびデータベースを作成し、ワークスペースのメタデータを永続化します。

```
CREATE USER <database-user> WITH PASSWORD '<database-password>'
CREATE DATABASE <database>
GRANT ALL PRIVILEGES ON DATABASE <database> TO <database-user>
ALTER USER <database-user> WITH SUPERUSER
```

3. 以下のプレースホルダーの値を定義します。

   - <identity-database-password> は RH-SSO データベースのパスワードです。

4. 以下の SQL スクリプトを使用して、RH-SSO バックエンドのデータベースを作成して、ユーザー情報を永続化します。

```
CREATE USER keycloak WITH PASSWORD '<identity-database-password>'
CREATE DATABASE keycloak
GRANT ALL PRIVILEGES ON DATABASE keycloak TO keycloak
```

8.7.2. 外部 PostgreSQL と連携するように CodeReady Workspaces を設定する

前提条件

- **oc ツールが利用できる。**

手順

1. **CodeReady Workspaces のプロジェクトを事前に作成します。**

```
$ oc create namespace openshift-workspaces
```

2. **CodeReady Workspaces サーバーデータベースの認証情報を保存するためにシークレットを作成します。**

```
$ oc create secret generic <server-database-credentials> \ 1
--from-literal=user=<database-user> \ 2
--from-literal=password=<database-password> \ 3
-n openshift-workspaces
```

1. **CodeReady Workspaces サーバーデータベースの認証情報を保存するためのシークレットの名前**

2. **CodeReady Workspaces サーバーデータベースのユーザー名**

3. **CodeReady Workspaces サーバーデータベースのパスワード**

3. **必要なラベルを CodeReady Workspaces サーバーデータベース認証情報シークレットに追加します。**

```
$ oc label secret <server-database-credentials> \ 1
 app.kubernetes.io/part-of=che.eclipse.org -n openshift-workspaces
```

1. **CodeReady Workspaces サーバーデータベースの認証情報を保存するためのシークレットの名前**

4. **RH-SSO データベース認証情報を保存するためのシークレットを作成します。**

```
$ oc create secret generic <identity-database-credentials> \ 1
--from-literal=password=<identity-database-password> \ 2
-n openshift-workspaces
```

1 **RH-SSO データベースの認証情報を保存するためのシークレット名**

2 **RH-SSO データベースのパスワード**

5. **必要なラベルを RH-SSO データベース認証情報シークレットに追加します。**

```
$ oc label secret <identity-database-credentials> \ 1
app.kubernetes.io/part-of=che.eclipse.org -n openshift-workspaces
```

1 **RH-SSO データベースの認証情報を保存するためのシークレット名**

6. **パッチを適用して crwctl コマンドを実行し、Red Hat CodeReady Workspaces をデプロイします。以下は例になります。**

```
$ crwctl server:deploy --che-operator-cr-patch-yaml=patch.yaml ...
```

patch.yaml には、Operator がデータベースのデプロイを省略し、既存のデータベースの接続詳細を CodeReady Workspaces サーバーに渡すことができるようにするために以下が含まれる必要があります。

```
spec:
  database:
    externalDb: true
    chePostgresHostName: <hostname>                1
    chePostgresPort: <port>                        2
    chePostgresSecret: <server-database-credentials>  3
    chePostgresDb: <database>                      4
```

```
spec:
  auth:
    identityProviderPostgresSecret: <identity-database-credentials> 5
```

1

外部データベースのホスト名

2

外部データベースポート

3

CodeReady Workspaces サーバーデータベースの認証情報を含むシークレットの名前

4

CodeReady Workspaces サーバーデータベースの名前

5

RH-SSO データベースの認証情報が含まれるシークレットの名前

関連情報

- PostgreSQL
- RDS

第9章 POSTGRESQL 9 から POSTGRESQL 13 への移行

2021 年 11 月 11 日以降、PostgreSQL バージョン 9.6 はサポートされなくなり、CodeReady Workspaces チームはすべてのユーザーがバージョン 13 に移行することを推奨しています。

以下の手順に従って、データの損失なしに、新しいバージョンの PostgreSQL に正常に移行してください。

前提条件

- **oc ツールが利用できる。**
- **OpenShift で実行される CodeReady Workspaces のインスタンス。**

手順

1. **CodeReady Workspaces インスタンスの実行中のすべてのワークスペースの Git リポジトリーに変更を保存し、プッシュします。**
2. **CodeReady Workspaces インスタンスのすべてのワークスペースを停止します。**
3. **CodeReady Workspaces および RH-SSO デプロイメントをスケールダウンします。**

```
oc scale deployment codeready --replicas=0 -n openshift-workspaces
oc scale deployment keycloak --replicas=0 -n openshift-workspaces
```

4. **利用可能なデータベースのバックアップを作成します。**

```
POSTGRES_POD=$(oc get pods -n openshift-workspaces | grep postgres | awk '{print $1}')
CHE_POSTGRES_DB=$(oc get checluster/codeready-workspaces -n openshift-workspaces
-o json  | jq '.spec.database.chePostgresDb')
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "pg_dump
$CHE_POSTGRES_DB > /tmp/che.sql"
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "pg_dump keycloak >
/tmp/keycloak.sql"
```

5. **取得したバックアップをローカルファイルシステムにコピーします。**

```
oc cp openshift-workspaces/$POSTGRES_POD:/tmp/che.sql che.sql
oc cp openshift-workspaces/$POSTGRES_POD:/tmp/keycloak.sql keycloak.sql
```

6. **PostgreSQL デプロイメントをスケールダウンします。**

```
oc scale deployment postgres --replicas=0 -n openshift-workspaces
```

7. **古いデータをクリーンアップするために、対応する PVC ユニットを削除します。**

```
oc delete pvc postgres-data -n openshift-workspaces
```

上記のステップから PVC を削除した後に、新規 PVC は自動的に数秒で表示されます。

8. **新規 PostgreSQL データベースのバージョンを 13.3 に設定します。**

```
oc patch checluster codeready-workspaces -n openshift-workspaces --type=json -p '[{"op":
"replace", "path": "/spec/database/postgresVersion", "value": "13.3"}]'
```

9. **PostgreSQL デプロイメントをスケールアップします。**

```
oc scale deployment postgres --replicas=1 -n openshift-workspaces
oc wait --for=condition=ready pod -l app.kubernetes.io/component=postgres -n openshift-
workspaces --timeout=120s
```

10. **データベースをプロビジョニングします。**

```
POSTGRES_POD=$(oc get pods -n openshift-workspaces | grep postgres | awk '{print $1}')
OPERATOR_POD=$(oc get pods -n openshift-workspaces | grep codeready-operator | awk
'{print $1}')

IDENTITY_POSTGRES_SECRET=$(oc get checluster/codeready-workspaces -n openshift-
workspaces -o json | jq -r '.spec.auth.identityProviderPostgresSecret')
IDENTITY_POSTGRES_PASSWORD=$(if [ -z "$IDENTITY_POSTGRES_SECRET" ] || [
$IDENTITY_POSTGRES_SECRET = "null" ]; then oc get checluster/codeready-workspaces
-n openshift-workspaces -o json | jq -r '.spec.auth.identityProviderPostgresPassword'; else oc
get secret $IDENTITY_POSTGRES_SECRET -n openshift-workspaces -o json | jq -r
'.data.password' | base64 -d; fi)

oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"CREATE USER keycloak WITH PASSWORD '$IDENTITY_POSTGRES_PASSWORD'\""
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"CREATE DATABASE keycloak\""
```

```
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"GRANT ALL PRIVILEGES ON DATABASE keycloak TO keycloak\""

POSTGRES_SECRET=$(oc get checluster/codeready-workspaces -n openshift-workspaces
-o json | jq -r '.spec.database.chePostgresSecret')
CHE_USER=$(if [ -z "$POSTGRES_SECRET" ] || [ $POSTGRES_SECRET = "null" ]; then
oc get checluster/codeready-workspaces  -n openshift-workspaces -o json | jq -r
'.spec.database.chePostgresUser'; else oc get secret $POSTGRES_SECRET -n openshift-
workspaces -o json | jq -r '.data.user' | base64 -d; fi)

oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql postgres -tAc
\"ALTER USER $CHE_USER WITH SUPERUSER\""
```

11. **バックアップを PostgreSQL Pod にコピーします。**

```
oc cp che.sql openshift-workspaces/$POSTGRES_POD:/tmp/che.sql
oc cp keycloak.sql openshift-workspaces/$POSTGRES_POD:/tmp/keycloak.sql
```

12. **データベースを復元します。**

```
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql keycloak <
/tmp/keycloak.sql"
oc exec -it $POSTGRES_POD -n openshift-workspaces  -- bash  -c "psql
$CHE_POSTGRES_DB < /tmp/che.sql"
```

13. **RH-SSO および CodeReady Workspaces デプロイメントをスケールアップします。**

```
oc scale deployment keycloak --replicas=1 -n openshift-workspaces
oc wait --for=condition=ready pod -l app.kubernetes.io/component=keycloak -n openshift-
workspaces --timeout=120s
oc scale deployment codeready --replicas=1 -n openshift-workspaces
oc wait --for=condition=ready pod -l app.kubernetes.io/component=codeready -n openshift-
workspaces --timeout=120s
```

第10章 READINESS INIT コンテナー

CodeReady Workspaces Operator は CodeReady Workspaces をインストールし、そのコンテナーを正しい順序で起動します。CodeReady Workspaces を持つノードが再起動され、すべての CodeReady Workspaces コンテナーを同時に起動する場合、一部のコンテナーに依存する他のコンポーネントが準備状態にならないため、一部のコンテナーが失敗する場合があります。このような障害を回避するには、readiness init コンテナーは CodeReady Workspaces コンポーネントのコンテナーを正しい順序でキューに入れます。

readiness init コンテナーはデフォルトで無効にされます。有効にする場合は、CodeReady Workspaces のインストールに使用するインストール方法に従って続行します。

- 「Operator インストーラーの readiness init コンテナーの有効化および無効化」
- 「OLM インストーラーの readiness init コンテナーの有効化および無効化」

10.1. OPERATOR インストーラーの READINESS INIT コンテナーの有効化および無効化

Operator インストーラーによってインストールされた CodeReady Workspaces の readiness init コンテナーを有効および無効にできます。

- Operator インストーラーの readiness init コンテナーの有効化
- Operator インストーラーの readiness init コンテナーの無効化

10.2. OPERATOR インストーラーの READINESS INIT コンテナーの有効化

readiness init コンテナーはデフォルトでは有効になっていないため、まずそれらを有効にする必要があります。Operator インストーラーによってインストールされた CodeReady Workspaces の readiness init コンテナーを有効にするには、以下を実行します。

前提条件

- Operator インストーラーによってインストールされる Red Hat CodeReady Workspaces。

手順

手順

1. CodeReady Workspaces Operator Deployment の名前を見つけます。通常、これは codeready-workspaces-operator になります。

```
$ oc get deployments -n openshift-workspaces
```

2. 以下のように Deployment を編集します。Operator Deployment の spec.template.spec.containers[0].env の下に以下の行を挿入します。

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS (1)
  value: "true"
```

(1) ADD_COMPONENT_READINESS_INIT_CONTAINERS は環境変数です。

3. CodeReady Workspaces Operator が一部のコンポーネントを再起動する間待機します。

注記

新規 Operator Deployment の作成時に各 CodeReady Workspaces のアップグレード後にこれらの手順を繰り返します。

10.3. OPERATOR インストーラーの READINESS INIT コンテナーの無効化

Operator インストーラーによってインストールされた CodeReady Workspaces の以前に有効にされた readiness init コンテナーを無効にするには、以下を実行します。

前提条件

- Operator インストーラーによってインストールされる Red Hat CodeReady Workspaces。

手順

1. CodeReady Workspaces Operator Deployment の名前を見つけます。通常、これは codeready-workspaces-operator になります。

```
$ oc get deployments -n openshift-workspaces
```

2. 以下のように Deployment を編集します。Operator Deployment の spec.template.spec.containers[0].env の下で、以下の行を削除します。

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS (1)
  value: "true"
```

(1) ADD_COMPONENT_READINESS_INIT_CONTAINERS は環境変数です。

3. CodeReady Workspaces Operator が一部のコンポーネントを再起動する間待機します。

10.4. OLM インストーラーの READINESS INIT コンテナーの有効化および無効化

OLM インストーラーによってインストールされた CodeReady Workspaces の readiness init コンテナーを有効および無効にできます。（crw ctl で利用可能）、OLM インストーラーは Operator Lifecycle Manager を使用して CodeReady Workspaces をインストールします。

- OLM インストーラーの readiness init コンテナーの有効化
- OLM インストーラーの readiness init コンテナーの無効化

10.5. OLM インストーラーの READINESS INIT コンテナーの有効化

readiness init コンテナーはデフォルトでは有効になっていないため、まずそれらを有効にする必要があります。OLM インストーラーによってインストールされた CodeReady Workspaces の readiness init コンテナーを有効にするには、以下を実行します。

前提条件

- CodeReady Workspaces は OLM インストーラーによってインストールされます。

手順

1. CodeReady Workspaces Operator サブスクリプション名を検索します。

```
$ oc get subscriptions -n openshift-workspaces
```

2. CodeReady Workspaces Operator サブスクリプションから CSV(Cluster Service Version)名を取得します。

```
$ oc get subscription <subscription-name> -n openshift-workspaces -o yaml | grep installedCSV
```

3. ClusterServiceVersion YAML マニフェストを編集します。

```
$ oc edit csv <csv-name> -n openshift-workspaces
```

4. 以下の環境変数を Operator Deployment 仕様に追加します。

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS
  value: "true"
```

5. CodeReady Workspaces Operator が再起動するまで待機します。その後、再起動された Operator はそのコンポーネントの一部の再起動を続行します。

注記

OLM で新規 CSV が作成されると、各 CodeReady Workspaces のアップグレードの後にこれらの手順を繰り返します。

10.6. OLM インストーラーの READINESS INIT コンテナーの無効化

OLM インストーラーによってインストールされた CodeReady Workspaces の以前に有効にされた readiness init コンテナーを無効にするには、以下を実行します。

前提条件

- CodeReady Workspaces は OLM インストーラーによってインストールされます。

手順

1. CodeReady Workspaces Operator サブスクリプション名を検索します。

```
$ oc get subscriptions -n openshift-workspaces
```

2. CodeReady Workspaces Operator サブスクリプションから CSV(Cluster Service Version)名を取得します。

```
$ oc get subscription <subscription-name> -n openshift-workspaces -o yaml | grep installedCSV
```

3. ClusterServiceVersion YAML マニフェストを編集します。

```
$ oc edit csv <csv-name> -n openshift-workspaces
```

4. 以下の環境変数を Operator Deployment 仕様から削除します。

```
- name: ADD_COMPONENT_READINESS_INIT_CONTAINERS
  value: "true"
```

5. CodeReady Workspaces Operator が再起動するまで待機します。その後、再起動された Operator はそのコンポーネントの一部の再起動を続行します。

第11章 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

CodeReady Workspaces 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 CodeReady Workspaces 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 CodeReady Workspaces 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、CodeReady Workspaces 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11.1 Image Puller パラメーター

パラメーター	使用法	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 の実行時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 の実行時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 の実行時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 の実行時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**

パラメーター	使用法	デフォルト
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	**'{}'**
IMAGE_PULL_SECRETS	イメージプルシークレットの一覧。形式は **pullsecret='{...** DaemonSet によって作成される Pod に追加するには、以下を実行します。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」**
- **「CodeReady Workspaces Operator を使用した Image Puller のインストール」**
- **「OperatorHub を使用した OpenShift 4 での Image Puller のインストール」**
- **「OpenShift テンプレートを使用した 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**

11.1. 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。

ワークスペースの起動に関連するイメージの事前プルが行われると、ワークスペースの起動時間が短縮されます。以下は例になります。

- Che-Theia
- ブローカーイメージ
- プラグインサイドカーイメージ

前提条件

- curl ツールが利用できる。curl ホームページを参照してください。
- jq ツールが利用できる。jq ホームページを参照してください。
- yq ツールが利用できる。yq ホームページを参照してください。

手順

1. OpenShift プラットフォームの関連するコンテナーイメージの一覧を収集します。

例11.1 CodeReady Workspaces 2.14 のすべてのイメージの一覧の取得

```
$ curl -sSLo- https://raw.githubusercontent.com/redhat-developer/codeready-workspaces-images/crw-2.14-rhel-8/codeready-workspaces-operator-metadata-generated/manifests/codeready-workspaces.csv.yaml \
  | yq -r '.spec.relatedImages[]'
```

2. ワークスペースの起動フェーズに関連するイメージを保持します。

- **eap**
- **machineexec**
- **mongodb**
- **pluginbroker**
- **plugin-**
- **stacks**
- **theia**
- **ubi-minimal**

3. **ターゲットプラットフォームで対応していないコンテナーイメージの一覧から除外します。**

- **AMD64 および Intel 64 (x86_64)の場合は、 openj9 イメージを除外します。**

例11.2 openj9 イメージを除く AMD64 および Intel 64 (x86_64)のイメージ一覧

```
che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-
workspaces/pluginbroker-artifacts-
rhel8@sha256:bde2f4c7c21d7cd7d826d4f4bbd2ee9f31b2119e2d2aa10253592099598
cf5ba;
che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:457dd2db3d72cc1d823e1219d657ae32e3a9da26f7dd420e0185d1cbe8
72a792;
codeready_workspaces_machineexec_plugin_registry_image_gixdcnak=registry.redhat.
o/codeready-workspaces/machineexec-
rhel8@sha256:1e25377fe0538ef380030a898fcfcff9493ff0bdbaa4db77d648cdcb003681
6b;
codeready_workspaces_plugin_java11_devfile_registry_image_gixdcnak=registry.redha
.io/codeready-workspaces/plugin-java11-
rhel8@sha256:2036cbb70aae5f5d507657bd4b820e340ee0bacf3d4b520d80dbd21aad
```

```
85e13a;
codeready_workspaces_plugin_java11_plugin_registry_image_gixdcnak=registry.redhat
.io/codeready-workspaces/plugin-java11-
rhel8@sha256:2036cbb70aae5f5d507657bd4b820e340ee0bacf3d4b520d80dbd21aad
85e13a;
codeready_workspaces_plugin_java8_devfile_registry_image_gixdcnak=registry.redhat.
o/codeready-workspaces/plugin-java8-
rhel8@sha256:f0ecc1812888611407c23ede1d3952dfb7b9bd597c336f22995cc4d8d9c2
3edd;
codeready_workspaces_plugin_java8_plugin_registry_image_gixdcnak=registry.redhat.i
o/codeready-workspaces/plugin-java8-
rhel8@sha256:f0ecc1812888611407c23ede1d3952dfb7b9bd597c336f22995cc4d8d9c2
3edd;
codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnak=registry.re
dhat.io/codeready-workspaces/plugin-kubernetes-
rhel8@sha256:5f40400fb032b419e90bb334c8748470eb50e9dc4662b487364e494ccf8
a3f05;
codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnak=registry.red
hat.io/codeready-workspaces/plugin-openshift-
rhel8@sha256:c4be840840349bb647e6ace19b519b8b3e9676da42bb094512be1fafd4
11ae37;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnak=registry.redhat.ic
/codeready-workspaces/stacks-cpp-
rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837
891c1;
codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcnak=registry.redhat.io/
codeready-workspaces/stacks-cpp-
rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837
891c1;
codeready_workspaces_stacks_dotnet_devfile_registry_image_gixdcnak=registry.redha
.io/codeready-workspaces/stacks-dotnet-
rhel8@sha256:88134d9fd6b7c81e237e6295183d59cfe3e546762315e93f4d6fb547ecdf
aeba;
codeready_workspaces_stacks_dotnet_plugin_registry_image_gixdcnak=registry.redhat
.io/codeready-workspaces/stacks-dotnet-
rhel8@sha256:88134d9fd6b7c81e237e6295183d59cfe3e546762315e93f4d6fb547ecdf
aeba;
codeready_workspaces_stacks_golang_devfile_registry_image_gixdcnak=registry.redha
t.io/codeready-workspaces/stacks-golang-
rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e
82462;
codeready_workspaces_stacks_golang_plugin_registry_image_gixdcnak=registry.redha
.io/codeready-workspaces/stacks-golang-
rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e
82462;
codeready_workspaces_stacks_php_devfile_registry_image_gixdcnak=registry.redhat.ic
/codeready-workspaces/stacks-php-
rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a
35;
codeready_workspaces_stacks_php_plugin_registry_image_gixdcnak=registry.redhat.io/
codeready-workspaces/stacks-php-
rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a
35;
codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnak=registry.redha
t.io/codeready-workspaces/theia-endpoint-
rhel8@sha256:128e281bceaccfcb3f9c3aebdd218b6bb6381f9c41cff2259eba47dd49d95
```

```
c4d;
codeready_workspaces_theia_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:928f5792cc39e6b7785f4f92ec0d6a5b9cd36fb285c1f72d12239beb05d8696e;
jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi_____=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi_____=registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-rhel8@sha256:3875b2ee2826a6d8134aa3b80ac0c8b5ebc4a7f718335d76dfc3461b79f93d19;
pvc_jobs=registry.redhat.io/ubi8/ubi-minimal@sha256:c536d4c63253318fdfc1db499f8f4bb0881db7fbd6f3d1554b4d54c812f85cc7;
rhscl_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhscl/mongodb-36-rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73;
```

- **IBM Z および IBM Power Systems の場合は、java8 および java11 の openj9 バージョンを使用しdotnet を除外します。**

例11.3 IBM Z および IBM Power Systems のイメージ一覧: java8 および java11 の openj9 バージョンを使用し、dotnet を除く

```
che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-workspaces/pluginbroker-artifacts-rhel8@sha256:bde2f4c7c21d7cd7d826d4f4bbd2ee9f31b2119e2d2aa10253592099598cf5ba;
che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-workspaces/pluginbroker-metadata-rhel8@sha256:457dd2db3d72cc1d823e1219d657ae32e3a9da26f7dd420e0185d1cbe872a792;
codeready_workspaces_machineexec_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/machineexec-rhel8@sha256:1e25377fe0538ef380030a898fcfcff9493ff0bdbaa4db77d648cdcb0036816b;
codeready_workspaces_plugin_java11_openj9_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-java11-openj9-rhel8@sha256:fc5e110243a8e30d23705897a1766de20ec637db4442d419ba05ace3b874c27f;
codeready_workspaces_plugin_java11_openj9_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-java11-openj9-rhel8@sha256:fc5e110243a8e30d23705897a1766de20ec637db4442d419ba05ace3b874c27f;
codeready_workspaces_plugin_java8_openj9_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-java8-openj9-rhel8@sha256:27fe438df6cfccdfb5d1e927cfa2f360b3bed3fbc409e923e68714a1ef586461;
codeready_workspaces_plugin_java8_openj9_plugin_registry_image_gixdcnak=registry
redhat.io/codeready-workspaces/plugin-java8-openj9-
```

rhel8@sha256:27fe438df6cfccdfb5d1e927cfa2f360b3bed3fbc409e923e68714a1ef586461;
codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-kubernetes-rhel8@sha256:5f40400fb032b419e90bb334c8748470eb50e9dc4662b487364e494ccf8a3f05;
codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-openshift-rhel8@sha256:c4be840840349bb647e6ace19b519b8b3e9676da42bb094512be1fafd411ae37;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837891c1;
codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837891c1;
codeready_workspaces_stacks_golang_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e82462;
codeready_workspaces_stacks_golang_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e82462;
codeready_workspaces_stacks_php_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a35;
codeready_workspaces_stacks_php_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a35;
codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/theia-endpoint-rhel8@sha256:128e281bceaccfcb3f9c3aebdd218b6bb6381f9c41cff2259eba47dd49d95c4d;
codeready_workspaces_theia_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:928f5792cc39e6b7785f4f92ec0d6a5b9cd36fb285c1f72d12239beb05d8696e;
jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi_____=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d;
jboss_eap_7_eap_xp3_openj9_11_openshift_devfile_registry_image_gmxdacq_=registry.redhat.io/jboss-eap-7/eap-xp3-openj9-11-openshift-rhel8@sha256:44f82c43a730acbfb4ce2be81ca32197099c370eeb85cedbee3d1e89e9ac7684;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi_____=registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-rhel8@sha256:3875b2ee2826a6d8134aa3b80ac0c8b5ebc4a7f718335d76dfc3461b79f93d19;
pvc_jobs=registry.redhat.io/ubi8/ubi-minimal@sha256:c536d4c63253318fdfc1db499f8f4bb0881db7fbd6f3d1554b4d54c812f

```
85cc7;
rhscl_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhscl/mongodb-36-rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73;
```

4. **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、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。

- **theia-rhel8**
- **theia-endpoint-rhel8**
- **pluginbroker-artifacts-rhel8**
- **pluginbroker-metadata-rhel8**
- **stacks-*-rhel8**
- **plugin-*-rhel8**
  - スタックイメージの一覧：コンテナーイメージ： スタック
  - プラグインイメージの一覧：コンテナーイメージ： プラグイン

関連情報

- 「Image Puller のメモリーパラメーターの定義」

- 「OperatorHub を使用した OpenShift 4 での Image Puller のインストール」
- 「OpenShift テンプレートを使用した OpenShift への Image Puller のインストール」

11.2. 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

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

- 「OperatorHub を使用した OpenShift 4 での Image Puller のインストール」
- 「OpenShift テンプレートを使用した OpenShift への Image Puller のインストール」

11.3. CODEREADY WORKSPACES OPERATOR を使用した IMAGE PULLER のインストール

本セクションでは、CodeReady Workspaces Operatorを使用して、テクノロジープレビュー状態でコミュニティがサポートする機能であるImage Pullerをインストールする方法を説明します。

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」
- Operator Lifecycle Manager および OperatorHub が OpenShift インスタンスで利用できる。OpenShift は、バージョン 4.2 以降のバージョンを提供します。
- CodeReady Workspaces Operator が利用できる。https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#installing-che-on-openshift-4-using-operatorhub.adocを参照してください。

手順

1. .spec.imagePuller.enable を true に設定して、CheCluster カスタムリソースの Image Puller を有効にします。

```
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  name: codeready-workspaces
spec:
  # ...
  imagePuller:
    enable: true
```

2. CheCluster カスタムリソースに Image Puller を設定します。

```
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  name: codeready-workspaces
spec:
  ...
```

```
  imagePuller:
    enable: true
    spec:
      configMapName: <kubernetes-image-puller>
      daemonsetName: <kubernetes-image-puller>
      deploymentName: <kubernetes-image-puller>
      images: 'che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-
workspaces/pluginbroker-artifacts-
rhel8@sha256:bde2f4c7c21d7cd7d826d4f4bbd2ee9f31b2119e2d2aa10253592099598cf5
ba;che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:457dd2db3d72cc1d823e1219d657ae32e3a9da26f7dd420e0185d1cbe872
a792;codeready_workspaces_machineexec_plugin_registry_image_gixdcnak=registry
.redhat.io/codeready-workspaces/machineexec-
rhel8@sha256:1e25377fe0538ef380030a898fcfcff9493ff0bdbaa4db77d648cdcb0036816
b;codeready_workspaces_plugin_java11_devfile_registry_image_gixdcnak=registry.re
dhat.io/codeready-workspaces/plugin-java11-
rhel8@sha256:2036cbb70aae5f5d507657bd4b820e340ee0bacf3d4b520d80dbd21aad85
e13a;codeready_workspaces_plugin_java11_plugin_registry_image_gixdcnak=registr
y.redhat.io/codeready-workspaces/plugin-java11-
rhel8@sha256:2036cbb70aae5f5d507657bd4b820e340ee0bacf3d4b520d80dbd21aad85
e13a;codeready_workspaces_plugin_java8_devfile_registry_image_gixdcnak=registry.
redhat.io/codeready-workspaces/plugin-java8-
rhel8@sha256:f0ecc1812888611407c23ede1d3952dfb7b9bd597c336f22995cc4d8d9c23e
dd;codeready_workspaces_plugin_java8_plugin_registry_image_gixdcnak=registry.re
dhat.io/codeready-workspaces/plugin-java8-
rhel8@sha256:f0ecc1812888611407c23ede1d3952dfb7b9bd597c336f22995cc4d8d9c23e
dd;codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnak=regi
stry.redhat.io/codeready-workspaces/plugin-kubernetes-
rhel8@sha256:5f40400fb032b419e90bb334c8748470eb50e9dc4662b487364e494ccf8a3f
05;codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnak=registr
y.redhat.io/codeready-workspaces/plugin-openshift-
rhel8@sha256:c4be840840349bb647e6ace19b519b8b3e9676da42bb094512be1fafd411a
e37;codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnak=registry.re
dhat.io/codeready-workspaces/stacks-cpp-
rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837891
c1;codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcnak=registry.red
hat.io/codeready-workspaces/stacks-cpp-
rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837891
c1;codeready_workspaces_stacks_dotnet_devfile_registry_image_gixdcnak=registry.r
edhat.io/codeready-workspaces/stacks-dotnet-
rhel8@sha256:88134d9fd6b7c81e237e6295183d59cfe3e546762315e93f4d6fb547ecdfae
ba;codeready_workspaces_stacks_dotnet_plugin_registry_image_gixdcnak=registry.r
edhat.io/codeready-workspaces/stacks-dotnet-
rhel8@sha256:88134d9fd6b7c81e237e6295183d59cfe3e546762315e93f4d6fb547ecdfae
ba;codeready_workspaces_stacks_golang_devfile_registry_image_gixdcnak=registry.
redhat.io/codeready-workspaces/stacks-golang-
rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e82
462;codeready_workspaces_stacks_golang_plugin_registry_image_gixdcnak=registry
.redhat.io/codeready-workspaces/stacks-golang-
rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e82
462;codeready_workspaces_stacks_php_devfile_registry_image_gixdcnak=registry.re
dhat.io/codeready-workspaces/stacks-php-
rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a35;
codeready_workspaces_stacks_php_plugin_registry_image_gixdcnak=registry.redhat
.io/codeready-workspaces/stacks-php-
```

```
rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a35;
codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnak=registry.red
hat.io/codeready-workspaces/theia-endpoint-
rhel8@sha256:128e281bceaccfcb3f9c3aebdd218b6bb6381f9c41cff2259eba47dd49d95c
4d;codeready_workspaces_theia_plugin_registry_image_gixdcnak=registry.redhat.io/
codeready-workspaces/theia-
rhel8@sha256:928f5792cc39e6b7785f4f92ec0d6a5b9cd36fb285c1f72d12239beb05d869
6e;jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi_
_____=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-
rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c8923586
32d;jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi_
_____=registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-
rhel8@sha256:3875b2ee2826a6d8134aa3b80ac0c8b5ebc4a7f718335d76dfc3461b79f93d
19;pvc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:c536d4c63253318fdfc1db499f8f4bb0881db7fbd6f3d1554b4d54c812f8
5cc7;rhscl_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/r
hscl/mongodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc
73;'
```

注記

サポートされている Image Puller を使用するには、これを KubernetesImagePuller Operator とは別にインストールします。Red Hat が提供する追加のテストおよび検証により、Red Hat の公式ビルドの利点。

CodeReady Workspaces のインストール時に Operator Hub での KubernetesImagePuller の使用を有効にし、使用するコミュニティーでサポートされるバージョンを設定します。

- コミュニティービルド
- Red Hat の公式ビルド

デフォルトイメージ

- CodeReady Workspaces Operator は、CheCluster カスタムリソースを作成する前にイメージがこのフィールドに追加されていない場合に、ワークスペースの起動（Theia イメージ、プラグインブローカーイメージ、サイドカープラグインイメージ）に使用されるデフォルトイメージを.spec.imagePuller.spec.imagesフィールドに入力します。CodeReady Workspaces Operator は、CodeReady Workspaces のロールアウト更新後に .spec.imagePuller.spec.images フィールドのデフォルトイメージを更新します。ただし、CheCluster カスタムリソースを作成する前に、イメージが .spec.imagePuller.spec.images フィールドに追加された場合、CodeReady Workspaces Operator はデフォルトイメージを追加しません。

- CheCluster カスタムリソースの作成 後 にユーザーによって提供されるイメージが .spec.imagePuller.spec.images フィールドに追加されると、CodeReady Workspaces Operator は後続の CodeReady Workspaces ロールアウトの更新時にデフォルトイメージを更新します。ロールアウトの更新後も、デフォルト以外のイメージは .spec.imagePuller.spec.images フィールドで変更されないままになります。

検証

- OpenShift は kubernetes-image-puller-operator サブスクリプションを作成します。

- eclipse-che namespace には コミュニティーでサポートされる community supported Kubernetes Image Puller Operator ClusterServiceVersion が含まれます。

```
$ oc get clusterserviceversions
```

- eclipse-che namespace には kubernetes-image-pullerおよび kubernetes-image-puller-operator デプロイメントが含まれます。

```
$ oc get deployments
```

- コミュニティーがサポートする Kubernetes Image Puller Operator は KubernetesImagePuller カスタムリソースを作成します。

```
$ oc get kubernetesimagepullers
```

CodeReady Workspaces Operator を使用した Image Puller のアンインストール

1. CheCluster カスタムリソースを編集し、.spec.imagePuller.enable を false に編集します。

2. CheCluster カスタムリソースを編集し、.spec.imagePuller.spec を、 CodeReady Workspaces Operator のオプションの Image Puller パラメーターを設定できるように設定します。

11.4. OPERATORHUB を使用した OPENSHIFT 4 での IMAGE PULLER のインストール

この手順では、Operator を使用してコミュニティーでサポートされる Kubernetes Image Puller Operator を OpenShift 4 にインストールする方法について説明します。

前提条件

- OpenShift 4 の実行中のインスタンスの管理者アカウント
- 「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」

手順

1. Image Puller をホストする OpenShift プロジェクト <kubernetes-image-puller> を作成するには、OpenShift Web コンソールを開き、Home → Projects セクションに移動し、Create Project をクリックします。
2. プロジェクトの詳細を指定します。
   - Name: <kubernetes-image-puller>
   - Display Name: <Image Puller>
   - Description: <Kubernetes Image Puller>
3. Operators → OperatorHub に移動します。
4. Filter by keywordボックスを使用して、community supported Kubernetes Image Puller Operatorを検索します。community supported Kubernetes Image Puller Operator をクリックします。
5. Operator の説明を確認します。Continue → Install をクリックします。
6. Installation Mode について A specific project on the cluster を選択します。ドロップダウンで、OpenShift プロジェクト <kubernetes-image-puller> を見つけます。Subscribe をクリックします。

7. コミュニティーがサポートする Kubernetes Image Puller Operator のインストールを待機します。KubernetesImagePuller → Create instance をクリックします。

8. YAML エディターのあるリダイレクトされるウィンドウで、KubernetesImagePuller カスタムリソースに変更を加え、Create をクリックします。

9. <kubernetes-image-puller> OpenShift プロジェクトの Workloads および Pods メニューに移動します。Image Puller が利用可能であることを確認します。

11.5. OPENSHIFT テンプレートを使用した OPENSHIFT への IMAGE PULLER のインストール

この手順では、OpenShift テンプレートを使用して Kubernetes Image Puller を OpenShift にインストールする方法を説明します。

前提条件

- 実行中の OpenShift クラスター。
- oc ツールが利用できる。
- 「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」

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

値	使用法	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/codeready-workspaces/imagepuller-rhel8:2.14**
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11.3 configmap.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用法	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"**
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	**'che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-**

値	使用法	デフォルト
		workspaces/pluginbroker-artifacts-rhel8@sha256:bde2f4c7c21d7cd7d826d4f4bbd2ee9f31b2119e2d2aa10253592099598cf5ba; che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-workspaces/pluginbroker-metadata-rhel8@sha256:457dd2db3d72cc1d823e1219d657ae32e3a9da26f7dd420e0185d1cbe872a792;codeready_workspaces_machineexec_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/machineexec-rhel8@sha256:1e25377fe0538ef380030a898fcfcff9493ff0bdbaa4db77d648cdcb0036816b; codeready_workspaces_plugin_java11_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-java11-rhel8@sha256:2036cbb70aae5f5d507657bd4b820e340ee0bacf3d4b520d80dbd21aad85e13a; codeready_workspaces_plugin_java11_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-java11-rhel8@sha256:2036cbb70aae5f5d507657bd4b820e340ee0bacf3d4b520d80dbd21aad85e13a; codeready_workspaces_plugin_java8_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-java8-rhel8@sha256:f0ecc1812888611407c23ede1d3952dfb7b9bd597c336f22995cc4d8d9c23edd; codeready_workspaces_plugin_java8_plugin_registry_image_gixdcnak=registr

値	使用法	デフォルト
		y.redhat.io/codeready-workspaces/plugin-java8-rhel8@sha256:f0ecc1812888611407c23ede1d3952dfb7b9bd597c336f22995cc4d8d9c23edd; codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-kubernetes-rhel8@sha256:5f40400fb032b419e90bb334c8748470eb50e9dc4662b487364e494ccf8a3f05; codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/plugin-openshift-rhel8@sha256:c4be840840349bb647e6ace19b519b8b3e9676da42bb094512be1fafd411ae37; codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837891c Professional codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:fc621b59be72465ab82cfa293b5b190521eecfed9c353051a7e72592837891c1; codeready_workspaces_stacks_dotnet_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-dotnet-rhel8@sha256:88134d9fd6b7c81e237e6295183d59cfe3e546762315e93f4d6fb547ecdfaeba; codeready_workspaces_stacks_dotnet_plugin_regis

値	使用法	デフォルト
		try_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-dotnet-rhel8@sha256:88134d9fd6b7c81e237e6295183d59cfe3e546762315e93f4d6fb547ecdfaeba; codeready_workspaces_stacks_golang_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e82462; codeready_workspaces_stacks_golang_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:ef135a05399a4d5f58bcb059b6634498bee5adbbcf8ddb2956abf25819e82462; codeready_workspaces_stacks_php_devfile_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a35; codeready_workspaces_stacks_php_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:f2ee2cf24f649092568f932977193f585caac19ef23892968d0fe4dbc90f4a35; codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/theia-endpoint-rhel8@sha256:128e281bceaccfcb3f9c3aebdd218b6bb6381f9c41cff2259eba47dd49d95c4d;

値	使用法	デフォルト
		codeready_workspaces_theia_plugin_registry_image_gixdcnak=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:928f5792cc39e6b7785f4f92ec0d6a5b9cd36fb285c1f72d12239beb05d8696e; jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqbi___=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d; jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi___=registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-rhel8@sha256:3875b2ee2826a6d8134aa3b80ac0c8b5ebc4a7f718335d76dfc3461b79f93d19; pvc_jobs=registry.redhat.io/ubi8/ubi-minimal@sha256:c536d4c63253318fdfc1db499f8f4bb0881db7fbd6f3d1554b4d54c812f85cc7;rhscl_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhscl/mongodb-36-rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73; '
NAMESPACE	ConfigMap の **NAMESPACE** の値	**k8s-image-puller**
NODE_SELECTOR	ConfigMap の **NODE_SELECTOR** の値	**"{}"**

表11.4 serviceaccount.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用法	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
```

第12章 ID および承認の管理

このセクションでは、Red Hat CodeReady Workspaces の ID および認証の管理についての各種の側面について説明します。

- 「ユーザーの認証」
- 「ユーザーの認証」
- 「認証の設定」
- 「ユーザーデータの削除」
- 「OpenShift OAuth の設定」

12.1. ユーザーの認証

以下では、CodeReady Workspaces サーバー上とワークスペース内の両方で、Red Hat CodeReady Workspaces のユーザー認証のすべての側面について説明します。これには、すべての REST API エンドポイント、WebSocket または JSON RPC 接続、一部の Web リソースのセキュリティーを保護することが含まれます。

すべての認証タイプは、JWT オープン標準を、ユーザー ID 情報を転送するコンテナーとして使用します。さらに、CodeReady Workspaces サーバー認証は、RH-SSO によってデフォルトで提供される OpenID Connect プロトコル実装に基づいて行われます。

ワークスペースでの認証は、自己署名されたワークごとの JWT トークンの発行と、JWTProxy に基づく専用サービスでの検証について示唆します。

12.1.1. CodeReady Workspaces サーバーに対する認証

12.1.1.1. 他の認証の実装を使用した CodeReady Workspaces サーバーに対する認証

この手順では、RH-SSO 以外の OpenID Connect (OIDC) 認証の実装を使用する方法について説明します。

手順

1. multiuser.properties ファイルに保存されている認証設定パラメーターを更新します (例: クライアント ID、認証 URL、レルム名)。

2. 単一のフィルターまたはフィルターチェーンを作成してトークンを検証し、CodeReady Workspaces ダッシュボードでユーザーを作成し、subject オブジェクトを作成します。

3. 新規の認証プロバイダーが OpenID プロトコルをサポートする場合、設定エンドポイントで利用可能な OIDC JS クライアントライブラリーを使用してください。これは特定の実装から分離されるためです。

4. 選択されたプロバイダーがユーザー (名前および姓、肩書き) についての追加データを保存する場合、この情報を提供するプロバイダーに固有の ProfileDao 実装を作成することが推奨されます。

12.1.1.2. OAuth を使用した CodeReady Workspaces サーバーに対する認証

サードパーティーサービスとのユーザーの対話を容易にするために、CodeReady Workspaces サーバーは OAuth 認証をサポートします。OAuth トークンは、GitHub 関連のプラグインにも使用されます。

OAuth 認証には、2 つの主要なフローがあります。

delegated

デフォルトです。OAuth 認証を RH-SSO サーバーに委譲します。

embedded

ビルトイン CodeReady Workspaces サーバーメカニズムを使用して OAuth プロバイダーと通信します。

2 つの実装間で切り替えるには、che.oauth.service_mode=<embedded|delegated> 設定プロパティーを使用します。

OAuth API の主な REST エンドポイントは /api/oauth であり、以下が含まれます。

- OAuth 認証フローを開始できる認証メソッドの /authenticate。
- プロバイダーからのコールバックを処理するコールバックメソッドの /callback。
- 現行ユーザーの OAuth トークンを取得するためのトークン GET メソッドの /token。
- 現行ユーザーの OAuth トークンを無効にするためのトークン DELETE メソッドの /token。
- 設定済みのアイデンティティープロバイダーの一覧を取得する GET メソッドの /。

12.1.1.3. Swagger または REST クライアントを使用したクエリーの実行

ユーザーの RH-SSO トークンを使用して、REST クライアントでユーザーの代わりにセキュアな API に対してクエリーを実行します。有効なトークンは、Request ヘッダーまたは ?token=$token クエリーパラメーターとして割り当てる必要があります。

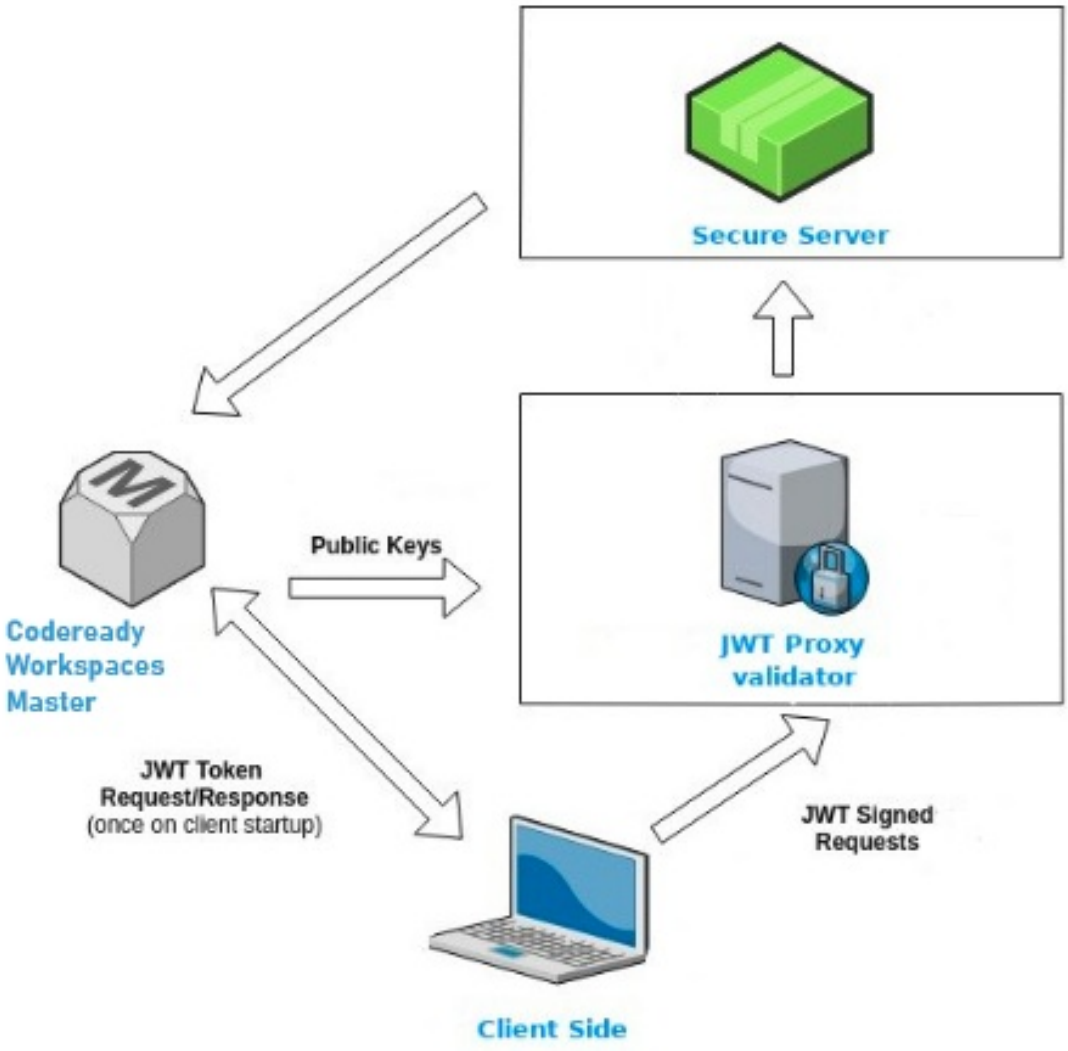
CodeReady Workspaces Swagger インターフェース \https://codeready-<openshift_deployment_name>.<domain_name>/swagger にアクセスします。アクセストークンが Request ヘッダーに含まれるように、ユーザーは RH-SSO で署名する必要があります。

12.1.2. CodeReady Workspaces ワークスペースでの認証

ワークスペースコンテナーには、認証で保護される必要のあるサービスが含まれる場合があります。このように保護されるサービスは、セキュア なサービスと呼ばれます。これらのサービスのセキュリティーを保護するには、マシンの認証メカニズムを使用します。

JWT トークンを使用すると、RH-SSO トークンをワークスペースコンテナーに渡す必要がなくなります（セキュアではなくなる可能性があります）。また、RH-SSO トークンの有効期間は比較的短く、定期的な更新またはリフレッシュが必要になる場合があります。このため、クライアントの同じユーザーセッショントークンを管理し、これらとの同期を維持するのは容易ではありません。

図12.1 ワークスペース内の認証



12.1.2.1. セキュアなサーバーの作成

CodeReady Workspaces ワークスペースでセキュアなサーバーを作成するには、devfile の dockerimage タイプコンポーネントで、エンドポイントの secure 属性を true に設定します。

セキュアなサーバーの devfile スニペット

```
components:
  - type: dockerimage
    endpoints:
      - attributes:
          secure: 'true'
```

12.1.2.2. ワークスペース JWT トークン

ワークスペーストークンは、要求に以下の情報が含まれる JSON Web トークン (JWT) です。

- uid:このトークンを所有するユーザーの ID
- Uname:このトークンを所有するユーザーの名前
- wsid:このトークンでクエリーできるワークスペースの ID

すべてのユーザーには、各ワークスペースに固有の個人用トークンが提供されます。トークンと署名の構造は、RH-SSO の場合とは異なります。以下は、トークンビューの例です。

```
# Header
{
  "alg": "RS512",
  "kind": "machine_token"
}
# Payload
{
  "wsid": "workspacekrh99xjenek3h571",
  "uid": "b07e3a58-ed50-4a6e-be17-fcf49ff8b242",
  "uname": "john",
  "jti": "06c73349-2242-45f8-a94c-722e081bb6fd"
}
# Signature
{
  "value": "RSASHA256(base64UrlEncode(header) + . +  base64UrlEncode(payload))"
}
```

RSA アルゴリズムを使用した SHA-256 暗号は、JWT トークンの署名に使用されます。これは設定不可です。また、トークンの署名に使用されるキーペアの公開部分を配布するパブリックサービスはありません。

12.1.2.3. マシントークンの検証

マシントークン (JWT トークン) の検証は、別の Pod で実行される JWTProxy と共に専用の per-workspace サービスを使用して実行されます。ワークスペースが起動すると、このサービスは CodeReady Workspaces サーバーから SHA キーの公開部分を受信します。セキュアなサーバーごとに個別の検証エンドポイントが作成されます。トラフィックがそのエンドポイントに到達すると、JWTProxy は cookie またはヘッダーからトークンを抽出し、公開鍵の部分を使用して検証します。

CodeReady Workspaces サーバーをクエリーするには、ワークスペースサーバーは CHE_MACHINE_TOKEN 環境変数で提供されるマシントークンを使用できます。このトークンは、ワークスペースを起動するユーザーのトークンです。このような要求の範囲は、現在のワークスペースにのみ制限されます。許可される操作の一覧も厳密に制限されます。

12.2. ユーザーの認証

CodeReady Workspaces でのユーザー認証は、パーミッションモデルに基づいて行われます。パーミッションは、ユーザーの許可されるアクションを制御し、セキュリティーモデルを確立するために使用されます。すべての要求は、認証にパスした後に、現行ユーザーのサブジェクトに必要なパーミッションがあるかどうかについて検証されます。CodeReady Workspaces が管理するリソースを制御し、ユーザーにパーミッションを割り当てることで特定のアクションを許可できます。

パーミッションは以下のエンティティーに適用できます。

- ワークスペース
- システム

すべてのパーミッションは、提供される REST API を使用して管理できます。API は、\https://codeready-<openshift_deployment_name>.<domain_name>/swagger/#!/permissions で Swagger を使用して文書化されます。

12.2.1. CodeReady Workspaces ワークスペースパーミッション

ワークスペースを作成するユーザーはワークスペースの所有者です。デフォルトで、ワークスペースの所有者には、パーミッション read、use、run、configure、setPermissions、および delete があります。ワークスペースの所有者は、ユーザーをワークスペースに招待し、他のユーザーのワークスペースのパーミッションを制御できます。

ワークスペースには以下のパーミッションが関連付けられています。

表12.1 CodeReady Workspaces ワークスペースパーミッション

パーミッション	説明
read	ワークスペース設定の読み取りを許可します。

パーミッション	説明
use	ワークスペースの使用や、これとの対話を許可します。
run	ワークスペースの開始および停止を許可します。
configure	ワークスペース設定の定義および変更を許可します。
setPermissions	その他のユーザーのワークスペースパーミッションの更新を許可します。
削除	ワークスペースの削除を許可します。

12.2.2. CodeReady Workspaces システムパーミッション

CodeReady Workspaces のシステムパーミッションは、CodeReady Workspaces インストール全体のさまざまな側面を制御します。以下のパーミッションがシステムに適用されます。

表12.2 CodeReady Workspaces システムパーミッション

パーミッション	説明
manageSystem	システムおよびワークスペースの制御を許可します。
setPermissions	システムでのユーザーのパーミッションの更新を許可します。
manageUsers	ユーザーの作成および管理を許可します。
monitorSystem	サーバーの状態の監視に使用するエンドポイントへのアクセスを許可します。

すべてのシステムパーミッションは管理ユーザーに付与されます。管理ユーザーを設定するには、CHE_SYSTEM_ADMIN__NAME プロパティーを使用します。デフォルト値は admin です。システムのパーミッションは CodeReady Workspaces サーバーの起動時に付与されます。ユーザーのレコードが CodeReady Workspaces ユーザーデータベースにない場合、ユーザーには最初のログイン後にパーミッションが付与されます。

12.2.3. manageSystem パーミッション

manageSystem パーミッションを 持つ ユーザーは、以下のサービスにアクセスできます。

パス	HTTP メソッド	説明
/resource/free/	GET	空きリソース制限を取得します。
/resource/free/{accountId}	GET	指定のアカウントの空きリソース制限を取得します。
/resource/free/{accountId}	POST	指定のアカウントの空きリソース制限を編集します。
/resource/free/{accountId}	DELETE	指定のアカウントの空きリソース制限を削除します。
/installer/	POST	インストーラーをレジストリーに追加します。
/installer/{key}	PUT	レジストリーでインストーラーを更新します。
/installer/{key}	DELETE	レジストリーからインストーラーを削除します。
/logger/	GET	CodeReady Workspaces サーバーのロギング設定を取得します。
/logger/{name}	GET	CodeReady Workspaces サーバーで、ロガーの名前でロガーの設定を取得します。
/logger/{name}	PUT	CodeReady Workspaces サーバーでロガーを作成します。
/logger/{name}	POST	CodeReady Workspaces サーバーでロガーを編集します。
/resource/{accountId}/details	GET	指定のアカウントのリソースに関する詳細情報を取得します。
/system/stop	POST	すべてのシステムサービスをシャットダウンし、CodeReady Workspaces の停止に向けて準備します。

12.2.4. monitorSystem パーミッション

monitorSystem パーミッションを持つユーザーは、以下のサービスにアクセスできます。

パス	HTTP メソッド	説明
/activity	GET	一定期間に特定の状態に置かれているワークスペースを取得します。

12.2.5. CodeReady Workspaces パーミッションの一覧表示

特定の リソース に適用される CodeReady Workspaces パーミッションを一覧表示するには、GET /permissions 要求を実行します。

user に適用されるパーミッションを一覧表示するには、GET /permissions/{domain} 要求を実行します。

すべてのユーザー に適用されるパーミッションを一覧表示するには、GET /permissions/{domain}/all 要求を実行します。この情報を表示するには、ユーザーに manageSystem パーミッションが必要です。

適切なドメイン値は次のとおりです。

- **system**
- **organization**
- **workspace**

注記

ドメインは任意です。ドメインを指定しないと、API はすべてのドメインについて想定されるすべてのパーミッションを返します。

12.2.6. CodeReady Workspaces パーミッションの割り当て

リソースにパーミッションを割り当てるには、POST /permissions 要求を実行します。適切なドメイン値は次のとおりです。

- system
- organization
- workspace

以下は、userId を持つユーザーの workspaceID を持つワークスペースに対するパーミッションを要求するメッセージの本体です。

CodeReady Workspaces ユーザーパーミッションの要求

```
{
  "actions": [
    "read",
    "use",
    "run",
    "configure",
    "setPermissions"
  ],
  "userId": "userID",          1
  "domainId": "workspace",
  "instanceId": "workspaceID"  2
}
```

1 userId パラメーターは、特定のパーミッションが付与されたユーザーの ID です。

2 instanceId パラメーターは、すべてのユーザーのパーミッションを取得するリソースの ID です。

12.3. 認証の設定

CodeReady Workspaces は、ユーザー認可のパーミッションモデルを使用します。

12.3.1. 認証およびユーザー管理

Red Hat CodeReady Workspaces は RH-SSO を使用してユーザーの作成、インポート、管理、削除、および認証を行います。RH-SSO は、ビルトイン認証メカニズムとユーザーストレージを使用します。サードパーティーのアイデンティティー管理システムを使用してユーザーを作成し、認証できます。CodeReady Workspaces リソースへのアクセスを要求する場合に、Red Hat CodeReady Workspaces には RH-SSO トークンが必要です。

ローカルユーザーおよびインポートされたフェデレーションユーザーは、プロファイルにメールアドレスが必要です。

デフォルトの RH-SSO 認証情報は admin:admin です。Red Hat CodeReady Workspaces への初回ログイン時に、admin:admin 認証情報を使用できます。これにはシステム権限があります。

RH-SSO URL を特定します。

OpenShift Web コンソールおよび RH-SSO プロジェクトに移動します。

12.3.2. RH-SSO と連携する CodeReady Workspaces の設定

デプロイメントスクリプトは RH-SSO を設定します。以下のフィールドを使用して codeready-public クライアントを作成します。

- 有効なリダイレクト URI:この URL を使用して CodeReady Workspaces にアクセスします。
- Web Origins

以下は、RH-SSO と連携するように CodeReady Workspaces を設定する場合の一般的なエラーです。

無効な redirectURI エラー

エイリアスの myhost で CodeReady Workspaces にアクセスし、元の CHE_HOST が 1.1.1.1 の場合に発生します。このエラーが発生した場合は、RH-SSO 管理コンソールに移動し、有効なリダイレクト URI が設定されていることを確認してください。

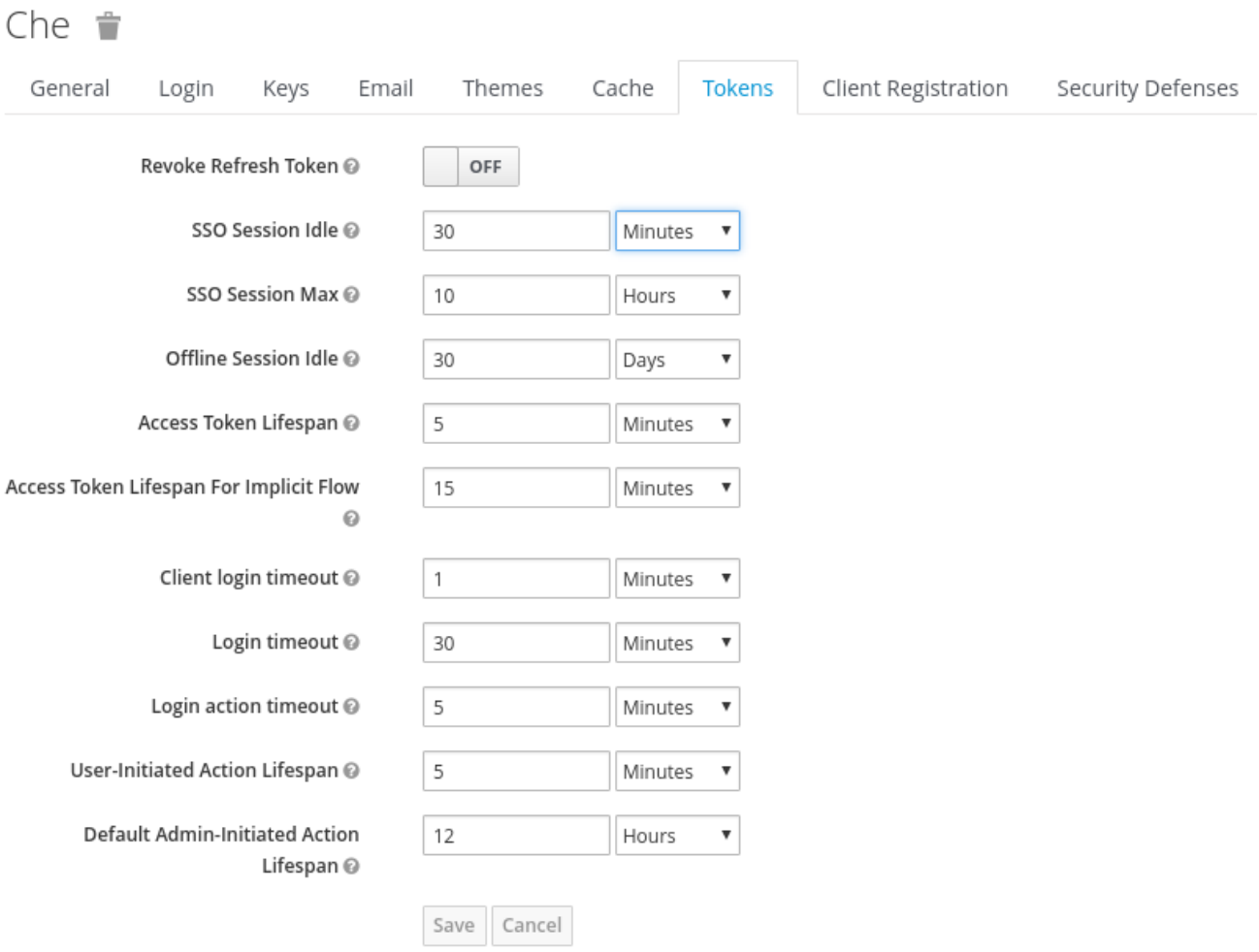
CORS エラー

無効な Web Origin がある場合に発生します。

12.3.3. RH-SSO トークンの設定

ユーザートークンの有効期間は、デフォルトで 30 分後に切れます。

以下の RH-SSO トークン設定を変更することができます。



12.3.4. ユーザーフェデレーションの設定

RH-SSO は、外部ユーザーデータベースのフェデレーションを行い、LDAP および Active Directory をサポートします。ストレージプロバイダーを選択する前に、接続をテストし、ユーザーを認証できます。

プロバイダーの追加方法については、RH-SSO ドキュメントのユーザーストレージのフェデーレーションについてのページを参照してください。

複数の LDAP サーバーを指定するには、RH-SSO ドキュメントの LDAP および Active Directory についてのページを参照してください。

12.3.5. ソーシャルアカウントおよびブローカーを使用した認証の有効化

RH-SSO は、GitHub、OpenShift、および Facebook や Twitter などの最も一般的に使用されるソーシャルネットワークの組み込みサポートを提供します。GitHub でログインを有効にする方法については、RH-SSO ドキュメントを参照してください。

12.3.5.1. GitHub OAuth の設定

GitHub の OAuth では、GitHub への SSH キーの自動アップロードを許可します。

前提条件

- oc ツールが利用できる。

手順

- CodeReady Workspaces URL をアプリケーションの Homepage URL の値として、また RH-SSO GitHub エンドポイント URL を認証コールバック URL の値として使用して GitHub に OAuth アプリケーションを作成します。デフォルト値は、それぞれhttps://codeready-openshift-workspaces.<DOMAIN>/およびhttps://keycloak-openshift-workspaces.<DOMAIN>/auth/realms/codeready/broker/github/endpointです。<DOMAIN>はOpenShiftクラスタードメインです。

1. CodeReady Workspaces がデプロイされているプロジェクトで新規シークレットを作成します。

```
$ oc apply -f - <<EOF
kind: Secret
apiVersion: v1
metadata:
  name: github-oauth-config
  namespace: <...> 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: <...> 2
  secret: <...> 3
EOF
```

1 CodeReady Workspaces namespace。デフォルトは openshift-workspaces です。

2 base64 でエンコードされた GitHub OAuth 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. CodeReady Workspaces がすでにインストールされている場合は、RH-SSO コンポーネントのロールアウトが完了するまで待機します。

12.3.5.2. 自己署名 TLS 証明書を使用する Bitbucket サーバーの設定

本章では、CodeReady Workspaces サーバーおよびワークスペースコンポーネントが BB で信頼される接続を確立できるように、自己署名 TLS 証明書を使用する Bitbucket (BB)サーバーを設定する方法を説明します。

- 追加の TLS 証明書および gitSelfSign 証明書用の ConfigMap の作成。これにより、以下が可能になります。
  - devfile URL を使用したファクトリーの起動。
  - プロジェクトのインポートおよびクローン作成。

注記

- BB サーバー側で OAuth 1 認証を設定します。詳細は、Bitbucket Server OAuth 1 の設定を参照してください。
- BB サーバーが自己署名 TLS 証明書で設定されている場合にのみ、追加の証明書をインポートする ConfigMap を作成する必要があります。これらの証明書は、CodeReady Workspaces サーバーおよびワークスペース内の適切な機能に必要であり、特定のリポジトリーに関連する Git 操作を実行するために使用します。

前提条件

- Base64 ASCII 形式でエクスポートされた BB サーバーの認証局(CA)の値。また、ca.crt ファイルに保存されます。
- CodeReady Workspaces のインスタンス

手順

1. BB サーバーの CA を CodeReady Workspaces サーバーにプロビジョニングし、BB サーバーに保存されている devfile を読み取ることができるようにします。これを実行するには、以下の ConfigMap を openshift-workspaces プロジェクトに追加します。

```
$ oc create configmap bitbucket-ca-cert-for-factory --from-file=ca.crt -n openshift-workspaces
```

```
$ oc label configmap bitbucket-ca-cert-for-factory app.kubernetes.io/part-of=che.eclipse.org
app.kubernetes.io/component=ca-bundle -n openshift-workspaces
```

2. Git 操作を使用できるように、BB サーバーの CA を CodeReady Workspaces サーバーにプロビジョニングします。これを実行するには、新規 ConfigMap を openshift-workspaces プロジェクトに追加します。

```
$ oc create configmap che-git-self-signed-cert --from-file=ca.crt --from-
literal=githost=<bitbucket_server_url> -n openshift-workspaces
```

3. CheCluster カスタムリソース(CR)を編集して、CodeReady Workspaces サーバーを設定します。

```
spec:
```

```
server:
 # ...
 gitSelfSignedCert: <boolean> 1
```

1

自己署名証明書を使用する BB サーバーには true を使用します。デフォルト値は false です。

- 詳細は、https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc を参照してください。

参照資料

- Bitbucket CA 証明書を CodeReady Workspaces に追加する場合は、https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc を参照してください。

12.3.5.3. OAuth1 を使用するように Bitbucket および CodeReady Workspaces 統合を設定

以下のセクションでは、Bitbucket (BB)リポジトリーで読み取りおよび書き込み操作を実行するのに必要な OAuth 1 認証の設定を説明します。clone、push などの許可された Git 操作で BB リポジトリーを使用するには、最初に CodeReady Workspaces で BB エンドポイントを登録し、OAuth 1 認証を設定します。

注記

この手順では、以下が必要です。

- RSA 鍵ペアの生成
- コンシューマーのキーペアの生成
- BB 側でのアプリケーションリンクの作成
- CodeReady Workspaces サーバー側での BB の設定

この手順では、Bitbucket Server の OAuth 1 をアクティベートして以下を実行する方法を説明します。

- Bitbucket Server でホストされる devfile を使用します。
- これは CodeReady Workspaces が Bitbucket Server Personal アクセストークンを取得し、更新できるようにします。

前提条件

- oc ツールが利用できる。
- Bitbucket サーバーは、CodeReady Workspaces サーバーから利用できます。
- CodeReady Workspaces のインスタンス

手順

1. RSA キーペアと公開鍵の省略バージョンを生成します。

```
$ openssl genrsa -out <private.pem> 2048
```

```
$ openssl rsa -in <private.pem> -pubout > <public.pub>
```

```
$ openssl pkcs8 -topk8 -inform pem -outform pem -nocrypt -in <private.pem> -out
<privatepkcs8.pem>
```

```
$ cat <public.pub> | sed 's/-----BEGIN PUBLIC KEY-----//g' | sed 's/-----END PUBLIC KEY--
---//g' | tr -d '\n' > <public-stripped.pub>
```

2. コンシューマーキーと共有シークレットを生成します。

```
$ openssl rand -base64 24 > <bitbucket_server_consumer_key>
```

```
$ openssl rand -base64 24 > <bitbucket_shared_secret>
```

3. Bitbucket でアプリケーションリンクを設定し、CodeReady Workspaces から Bitbucket サーバーへの通信を有効にします。

   a. Bitbucket Server で上部のナビゲーションバーをクリックし、Administration > Application Links に移動します。

   b. アプリケーション URL: \https://codeready-<openshift_deployment_name>.<domain_name> を入力し、Create new link ボタンをクリックします。

   c. No response was received from the URLという警告メッセージが表示されたら Continue ボタンをクリックします。

   d. Link Applications フォームを入力し、Continue ボタンをクリックします。

   Application Name

   <CodeReady Workspaces>

   Application Type

   Generic Application

   Service Provider Name

<CodeReady Workspaces>

Consumer Key

<bitbucket_server_consumer_key> ファイルの内容を貼り付けます。

Shared secret

<bitbucket_shared_secret> ファイルの内容を貼り付けます。

Request Token URL

<Bitbucket Server URL>/plugins/servlet/oauth/request-token

アクセストークン URL

<Bitbucket Server URL>/plugins/servlet/oauth/access-token

Authorize URL

<Bitbucket Server URL>/plugins/servlet/oauth/access-token

Create incoming link

Enabled

e. **Link Applications フォームを入力し、Continue ボタンをクリックします。**

Consumer Key

<bitbucket_server_consumer_key> ファイルの内容を貼り付けます。

Consumer name

<CodeReady Workspaces>

Public Key

<public-stripped.pub> ファイルの内容を貼り付けます。

4. **コンシューマーおよびプライベートキーが含まれる OpenShift シークレットを CodeReady Workspaces プロジェクトに作成します。**

```
$ oc apply -f - <<EOF
```

```
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  namespace: <CodeReady Workspaces-namespace> 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: '<scm-server-endpoint>' 2
type: Opaque
data:
  private.key: '<user-private-key>' 3
  consumer.key: '<bitbucket_server_consumer_key>' 4
EOF
```

1 CodeReady Workspaces namespace。デフォルトは openshift-workspaces です。

2 Bitbucket サーバー URL

3 最初の行と最後の行のない <privatepkcs8.pem> ファイルの base64 でエンコードされたコンテンツ。

4 <bitbucket_server_consumer_key> ファイルの base64 でエンコードされたコンテンツ。

例

```
#!/usr/bin/env bash

NS=${1:-eclipse-che}
CONSUMER_KEY=$(cat ./certs/bitbucket_server_consumer_key)
PRIVATE_KEY=$(cat ./certs/privatepkcs8.pem | sed 's/-----BEGIN PRIVATE KEY-----//g' |
sed 's/-----END PRIVATE KEY-----//g' | tr -d '\n')
BITBUCKET_HOST='<your-bitbucket-host-here>'
unameOut="$(uname -s)"

case "${unameOut}" in
```

```
    Linux*)     BASE64_FUNC='base64 -w 0';;
    Darwin*)    BASE64_FUNC='base64';;
    CYGWIN*)    BASE64_FUNC='base64 -w 0';;
    MINGW*)     BASE64_FUNC='base64 -w 0';;
    *)          BASE64_FUNC='base64 -w 0'
esac

cat <<EOF | oc apply -n $NS -f -
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: https://$BITBUCKET_HOST
type: Opaque
data:
  private.key: $(echo -n $PRIVATE_KEY | $BASE64_FUNC)
  consumer.key: $(echo -n $CONSUMER_KEY | $BASE64_FUNC)
EOF
```

- この GitHub サンプル のスクリプト全体を参照してください。

関連情報

- Bitbucket Server overview
- Download Bitbucket Server
- Bitbucket Server Personal access tokens
- How to generate public key to application link 3rd party applications
- Using AppLinks to link to other applications
- https://access.redhat.com/documentation/ja-

jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#authenticating-on-scm-server-with-a-personal-access-token.adoc

12.3.5.4. GitLab サーバーの設定

GitLab サーバーをプロジェクトソースサプライヤーとして使用するには、CHE_INTEGRATION_GITLAB_SERVER__ENDPOINTS プロパティーを使用して GitLab サーバーの URL を CodeReady Workspaces に登録し、登録するサーバーのホスト名を指定します。

例

```
https://gitlab.apps.cluster-2ab2.2ab2.example.opentlc.com/
```

GitLab サーバーの設定に関する追加の例については、「Operator を使用した CodeReady Workspaces サーバーの詳細設定について」を参照してください。

関連情報

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#advanced-configuration-options-for-the-che-server-component.adoc

12.3.5.5. Configuring GitLab OAuth2

GitLab の OAuth2 では、プライベート GitLab リポジトリーからファクトリーを受け入れることができます。

前提条件

- GitLab サーバーが実行中であり、CodeReady Workspaces から利用可能

手順

- CodeReady Workspaces をアプリケーションの Name として使用し、RH-SSO GitLab エンドポイント URL を Redirect URI の値として使用して、 GitLab に 承認済み OAuth2 アプリケーション を作成します。コールバック URL のデフォルト値は https://keycloak-

openshift-workspaces.<DOMAIN>/auth/realms/codeready/broker/gitlab/endpoint です。ここで、<DOMAIN> は OpenShift クラスタードメインです。Application ID および Secret の値を保存します。3 種類の GitLab OAuth 2 アプリケーションがすべてサポートされます。ユーザーの所有者、グループの所有者、およびインスタンス全体

1. GitLab サーバーを参照する RH-SSO でカスタム OIDC プロバイダーリンクを作成します。以下のフィールドに入力します。

クライアント ID

直前の手順で GitLab サーバーにより提供される Application ID フィールドの値。

クライアントのシークレット

直前の手順で GitLab サーバーにより提供される Secret フィールドの値。

認証 URL

https://<GITLAB_DOMAIN>/oauth/authorize 形式の URL

トークン URL

https://<GITLAB_DOMAIN>/oauth/token 形式の URL

スコープ

api write_repository openid のセットを含める必要のある（ただしこれに限定されない）一連のスコープ

トークンの保存

有効にする必要があります。

読み取り可能なトークンの保存

有効にする必要があります。

注記

- <GITLAB_DOMAIN> を GitLab インストールの URL およびポートに置き換えます。

2. CHE_INTEGRATION_GITLAB_OAUTH__ENDPOINT プロパティーを使用して、

GitLab インスタンス URL を CodeReady Workspaces で有効にされた OAuth 2 サポートに登録します。

警告

- GitLab インスタンスの URL は、設定された GitLab 統合エンドポイントの一覧に存在し、CHE_INTEGRATION_GITLAB_SERVER_ENDPOINTS プロパティーで設定します。

関連情報

CodeReady Workspaces が TLS キーに関連する GitLab にアクセスする場合は、以下のドキュメントを参照してください。

- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#importing-untrusted-tls-certificates.adoc
- https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#deploying-che-with-support-for-git-repositories-with-self-signed-certificates.adoc

12.3.6. プロトコルベースのプロバイダーの使用

RH-SSO は SAML v2.0 プロトコルおよび OpenID Connect v1.0 プロトコルをサポートします。

12.3.7. RH-SSO を使用したユーザーの管理

ユーザーインターフェースでユーザーを追加し、削除し、編集できます。詳細は、RH-SSO ユーザー管理について参照してください。

12.3.8. 外部 RH-SSO インストールを使用するように CodeReady Workspaces を設定する

デフォルトでは、CodeReady Workspaces インストールには、専用の RH-SSO インスタンスのデプロイメントが含まれます。ただし、外部の RH-SSO を使用することも可能です。このオプション

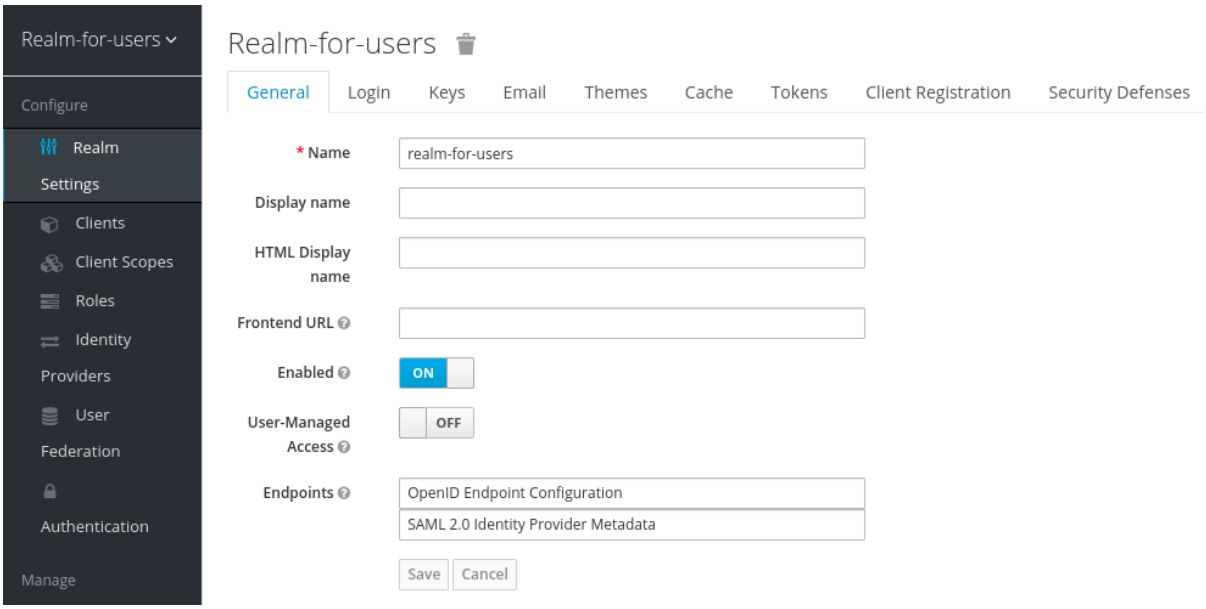
は、すでに定義されたユーザーを含む既存の RH-SSO インスタンス (複数のアプリケーションが使用する会社全体の RH-SSO サーバーなど) がある場合に役立ちます。

表12.3 サンプルで使用されるプレースホルダー

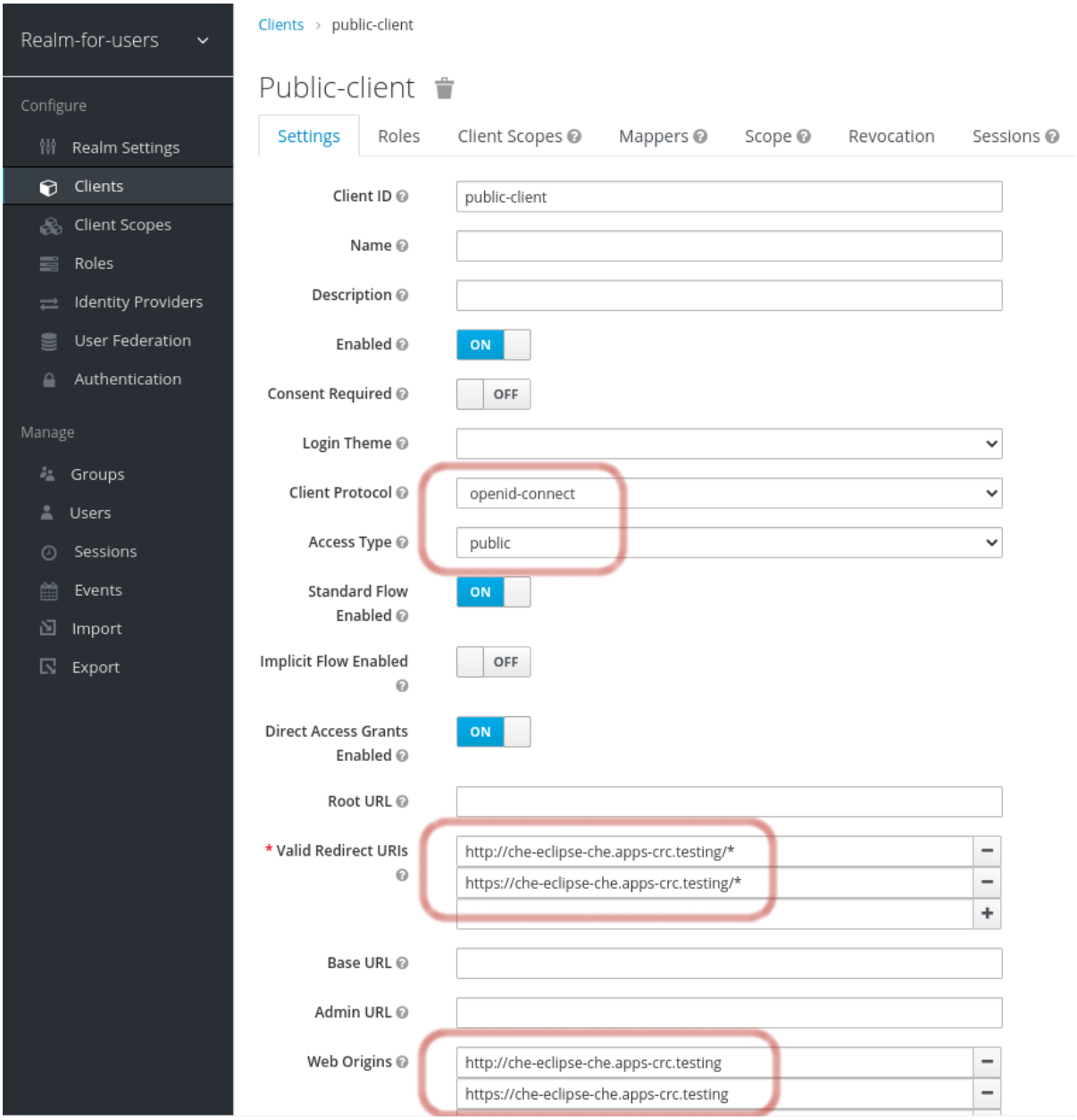
<provider-realm-name>	CodeReady Workspaces で使用することが意図された RH-SSO レルム名
<oidc-client-name>	**<provider-realm-name>** で定義された **oidc** クライアントの名前
<auth-base-url>	外部 RH-SSO サーバーのベース URL

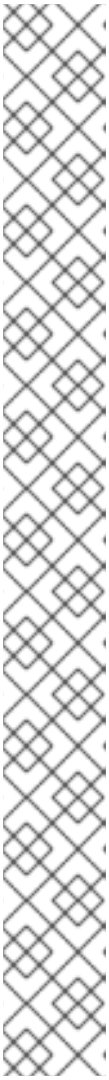
前提条件

- RH-SSO の外部インストールの管理コンソールで、CodeReady Workspaces に接続するユーザーが含まれる レルム を定義します。



- この realm では、CodeReady Workspaces がユーザーの認証に使用する OIDC クライアントを定義します。以下は、正しい設定のあるクライアントの例です。


Realm-for-users
Configure
Realm Settings
Clients
Client Scopes
Roles
Identity Providers
User Federation
Authentication
Manage
Groups
Users
Sessions
Events
Import
Export
Clients › public-client
Public-client
Settings
Roles
Client Scopes
Mappers
Scope
Revocation
Sessions
Client ID
public-client
Name
Description
Enabled
ON
Consent Required
OFF
Login Theme
Client Protocol
openid-connect
Access Type
public
Standard Flow Enabled
ON
Implicit Flow Enabled
OFF
Direct Access Grants Enabled
ON
Root URL
* Valid Redirect URIs
http://che-eclipse-che.apps-crc.testing/*
https://che-eclipse-che.apps-crc.testing/*
Base URL
Admin URL
Web Origins
http://che-eclipse-che.apps-crc.testing
https://che-eclipse-che.apps-crc.testing



注記

- Client Protocol は openid-connect である必要があります。
- Access Type は public である必要があります。CodeReady Workspaces は public アクセスタイプのみをサポートします。
- Valid Redirect URIs には、http プロトコルを使用する URL と https を使用する URL の 2 つ以上の CodeReady Workspaces サーバーに関連する URI が含まれる必要があります。これらの URI には CodeReady Workspaces サーバーのベース URL (この後に /* ワイルドカードが続く) が含まれる必要があります。
- Web Origins には、http プロトコルを使用する URI と https を使用する URI の 2 つ以上の CodeReady Workspaces サーバーに関連する URI が含まれる必要があります。これらの URI には、CodeReady Workspaces サーバーのベース URL (ホストの後はパスがない) が含まれる必要があります。

URI の数は、インストールされている製品ツールの数によって異なります。

- デフォルトの OpenShift OAuth サポートを使用する CodeReady Workspaces では、ユーザー認証は、OpenShift OAuth と RH-SSO の統合に依存します。これにより、ユーザーは OpenShift ログインで CodeReady Workspaces にログインでき、独自のワークスペースを個人の OpenShift プロジェクトに作成することができます。

  これには、OpenShift の「RH-SSO Identity Provider」を設定する必要があります。外部 RH-SSO を使用する場合は、手動で RH-SSO を設定します。手順については、OpenShift 3 または OpenShift 4 のいずれかに該当する RH-SSO ドキュメントを参照してください。

- 設定した RH-SSO のオプションには、Store Tokens および Stored Tokens Readable が有効になっています。

手順

1. CheCluster カスタムリソース (CR) に以下のプロパティーを設定します。

```
spec:
  auth:
```

```
    externalIdentityProvider: true
    identityProviderURL: <auth-base-url>
    identityProviderRealm: <provider-realm-name>
    identityProviderClientId: <oidc-client-name>
```

2. OpenShift OAuth サポートを有効にして CodeReady Workspaces をインストールする場合は、CheCluster カスタムリソース (CR) に以下のプロパティーを設定します。

```
spec:
  auth:
    openShiftoAuth: true
# Note: only if the OpenShift "RH-SSO Identity Provider" alias is different from 'openshift-v3' or 'openshift-v4'
server:
  customCheProperties:
    CHE_INFRA_OPENSHIFT_OAUTHIDENTITYPROVIDER: <OpenShift "RH-SSO Identity Provider" alias>
```

12.3.9. SMTP およびメール通知の設定

Red Hat CodeReady Workspaces は事前に設定された MTP サーバーを提供しません。

RH-SSO で SMTP サーバーを有効にするには、以下を実行します。

1. che realm settings > Email の順に移動します。
2. ホスト、ポート、ユーザー名、およびパスワードを指定します。

Red Hat CodeReady Workspaces は、登録、メールの確認、パスワードの復旧、およびログインの失敗についてのデフォルトのテーマを使用します。

12.3.10. 自己登録の有効化

自己登録により、ユーザーは CodeReady Workspaces サーバー URL にアクセスして、CodeReady Workspaces インスタンスに自己登録できます。

OpenShift OAuth サポートなしでインストールされた CodeReady Workspaces では、自己登録はデフォルトで無効にされるため、ログインページで新規ユーザーを登録するオプションは利用できません。

前提条件

- 管理者としてログインしている。

手順

ユーザーの自己登録を有効にするには、以下を実行します。

1. 左側の Realm Settings メニューに移動し、Login タブを開きます。
2. User registration オプションを On に設定します。

12.4. OPENSHIFT OAUTH の設定

ユーザーが OpenShift と対話できるようにするには、まず OpenShift クラスターに対して認証する必要があります。OpenShift OAuth は、ユーザーが取得された OAuth アクセストークンを使って API 経由でクラスターに対して自らを証明するプロセスです。

https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/end-user_guide/index#openshift-connector-overview.adoc を使用した認証は、CodeReady Workspaces ユーザーが OpenShift クラスターで認証できる方法です。

以下のセクションでは、OpenShift OAuth 設定オプションと、CodeReady Workspaces での使用方法について説明します。

12.4.1. 初期ユーザーでの OpenShift OAuth の設定

前提条件

- oc ツールが利用できる。
- crwctl 管理ツールが利用可能である。https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#using-the-chectl-management-tool.adoc を参照してください。

手順

- クラスターで OpenShift アイデンティティープロバイダーを設定します。アイデンティティープロバイダー設定について参照してください。

  ユーザーが OpenShift「RH-SSO Identity Provider」の設定ステップを省略し、OpenShift クラスターに設定済みの RH-SSO が含まれていない場合、CodeReady Workspaces は HTPasswd アイデンティティープロバイダーの初期 OpenShift ユーザーを作成します。このユーザーの認証情報は、openshift-config namespace にある openshift-oauth-user-credentials シークレットに保存されます。

  OpenShift クラスターおよび CodeReady Workspaces インスタンスにログインするための認証情報を取得します。

  1. OpenShift ユーザー名を取得します。

     ```
     $ oc get secret openshift-oauth-user-credentials -n openshift-config -o json | jq -r
     '.data.user' | base64 -d
     ```

  2. OpenShift ユーザーパスワードを取得します。

     ```
     $ oc get secret openshift-oauth-user-credentials -n openshift-config -o json | jq -r
     '.data.password' | base64 -d
     ```

- OperatorHub または crwctl を使用して CodeReady Workspaces をデプロイする場合は、crwctl server:deploy 仕様についての章を参照してください。OpenShift OAuth はデフォルトで有効にされます。

12.4.2. OpenShift の初期 OAuth ユーザーをプロビジョニングせずに OpenShift OAuth を設定する

以下の手順では、初期 OAuth ユーザーをプロビジョニングせずに OpenShift OAuth を設定する方法を説明します。

前提条件

- crwctl 管理ツールが利用可能である。https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#using-the-chectl-management-tool.adoc を参照してください。

手順

1. Operator を使用して CodeReady Workspaces をインストールしている場合は、codeready-workspaces カスタムリソースに以下の値を設定します。

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

2. crwctl ツールを使用して CodeReady Workspaces をインストールしている場合は、--che-operator-cr-patch-yaml フラグを使用します。

```
$ crwctl server:deploy --che-operator-cr-patch-yaml=patch.yaml ...
```

patch.yaml ファイルには以下を含める必要があります。

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

12.4.3. OpenShift 初期 OAuth ユーザーの削除

以下の手順では、Red Hat CodeReady Workspaces がプロビジョニングする OpenShift の初期の OAuth ユーザーを削除する方法を説明します。

前提条件

- oc ツールがインストールされている。
- OpenShift で実行している Red Hat CodeReady Workspaces のインスタンス。
- oc ツールを使用して OpenShift クラスターにログインしている。

手順

1. codeready-workspaces カスタムリソースを更新します。

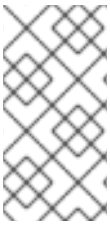
```
$ oc patch checluster/codeready-workspaces -n openshift-workspaces --type=json -p \
'[{"op": "replace", "path": "/spec/auth/initialOpenShiftOAuthUser", "value": false}]'
```

12.5. ユーザーデータの削除

12.5.1. GDPR に準拠したユーザーデータの削除

一般データ保護規則 (GDPR: General Data Protection Regulation) では、個人データの消去を求める個人の権利が施行されています。

以下の手順では、クラスターおよび RH-SSO データベースからユーザーのデータを削除する方法を説明します。



注記

以下のコマンドは、-n オプションのユーザー例として、デフォルトの OpenShift プロジェクト openshift-workspaces を使用します。

前提条件

- ユーザーまたは管理者の認証トークン。ユーザーアカウントにバインドされているデータ以外のデータを削除する場合は、admin 権限が必要になります。admin は、CHE_SYSTEM_ADMIN__NAME および CHE_SYSTEM_SUPER__PRIVILEGED__MODE = true カスタムリソース定義を使用して事前に作成され、有効にされる特別な CodeReady Workspaces 管理者アカウントです。

```
spec:
 server:
   customCheProperties:
     CHE_SYSTEM_SUPER__PRIVILEGED__MODE: 'true'
     CHE_SYSTEM_ADMIN__NAME: '<admin-name>'
```

必要に応じて、以下のコマンドを使用して admin ユーザーを作成します。

```
$ oc patch checluster/codeready-workspaces \
  --type merge \
  -p '{ "spec": { "server": {"customCheProperties":
{"CHE_SYSTEM_SUPER__PRIVILEGED__MODE": "true"} } }}' \
  -n openshift-workspaces
```

```
$ oc patch checluster/codeready-workspaces \
  --type merge \
```

```
  -p '{ "spec": { "server": {"customCheProperties": {"CHE_SYSTEM_ADMIN__NAME":
"<admin-name>"} } }}' \
  -n openshift-workspaces
```

注記

すべてのシステムパーミッションは管理ユーザーに付与されます。管理ユーザーを設定するには、CHE_SYSTEM_ADMIN__NAME プロパティーを使用します。デフォルト値は admin です。システムのパーミッションは CodeReady Workspaces サーバーの起動時に付与されます。ユーザーレコードが CodeReady Workspaces ユーザーデータベースにない場合は、ユーザーの最初のログイン後にパーミッションが付与されます。

認証トークンの権限:

- **admin: すべてのユーザーのすべての個人データを削除できます。**
- **user: ユーザーに関連するデータのみを削除できます。**

- **ユーザーまたは管理者が、CodeReady Workspaces がデプロイされた状態で OpenShift クラスターにログインしている。**

- **ユーザー ID が取得されます。以下のコマンドを使用してユーザー ID を取得します。**

  - **現行ユーザーの場合:**

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user'
```

  - **名前でユーザーを検索するには、以下を実行します。**

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/find?
name=<username>'
```

  - **メールでユーザーを検索するには、以下を実行します。**

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/find?
email=<email>'
```

ユーザー ID を取得する例

この例では、ローカルユーザーの名前として vparfono を使用します。

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://che-vp-che.apps.che-dev.x6e0.p1.openshiftapps.com/api/user/find?
name=vparfono'
```

ユーザー ID は、curl コマンド出力の下部にあります。

```
{
 "name": "vparfono",
 "links": [
   {
.
.
.
   }
 ],
 "email": "vparfono@redhat.com",
 "id": "921b6f33-2657-407e-93a6-fb14cf2329ce"
}
```

手順

1. **codeready-workspaces CheCluster Custom リソース (CR) 定義を更新して、RH-SSO データベースからユーザーのデータの削除を許可します。**

```
$ oc patch checluster/codeready-workspaces \
  --patch "{\"spec\":{\"server\":{\"customCheProperties\":
{\"CHE_KEYCLOAK_CASCADE__USER__REMOVAL__ENABLED\": \"true\"}}}}" \
  --type=merge -n openshift-workspaces
```

2. **API を使用してデータを削除します。**

```
$ curl -i -X DELETE \
  --header 'Authorization: Bearer <user-token>' \
  https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/<user-id>
```

検証

以下のコマンドを実行すると、コード 204 が API 応答として返されます。

```
$ curl -i -X DELETE \
  --header 'Authorization: Bearer <user-token>' \
  https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/<user-id>
```

関連情報

すべてのユーザーのデータを削除するには、https://access.redhat.com/documentation/ja-jp/red_hat_codeready_workspaces/2.14/html-single/installation_guide/index#uninstalling-che.adoc の手順に従います。