Red Hat CodeReady Workspaces 2.9

管理ガイド

Red Hat CodeReady Workspaces 2.9 の管理

Last Updated: 2023-02-03

Red Hat CodeReady Workspaces 2.9 管理ガイド

Red Hat CodeReady Workspaces 2.9 の管理

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

法律上の通知

概要

Information for administrators operating Red Hat CodeReady Workspaces.

目次

多様性を受け入れるオープンソースの強化

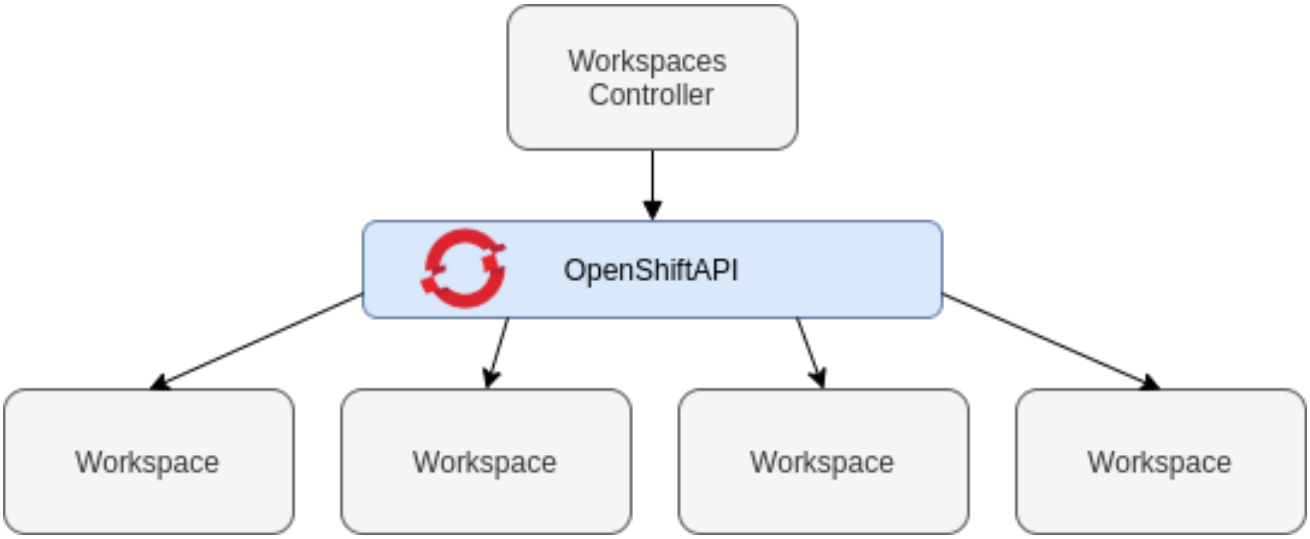
Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、弊社 の CTO、Chris Wright のメッセージを参照してください。

第1章 CODEREADY WORKSPACES アーキテクチャーの概要

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

- 中央のワークスペースコントローラー： OpenShift API でユーザーワークスペースを管理する常に実行されるサービス。
- ユーザーワークスペース： ユーザーがコーディングを停止する際にコントローラーが停止させるコンテナーベースの IDE。

図1.1 CodeReady Workspaces アーキテクチャーの概要



CodeReady Workspaces が OpenShift クラスターにインストールされる際、ワークスペースコントローラーはデプロイされている唯一のコンポーネントになります。CodeReady Workspaces ワークスペースは、ユーザーがこれをリクエストするとすぐに作成されます。

関連情報

- 「CodeReady Workspaces ワークスペースコントローラーについて」
- 「CodeReady Workspaces ワークスペースアーキテクチャーについて」

1.1. CODEREADY WORKSPACES ワークスペースコントローラーについて

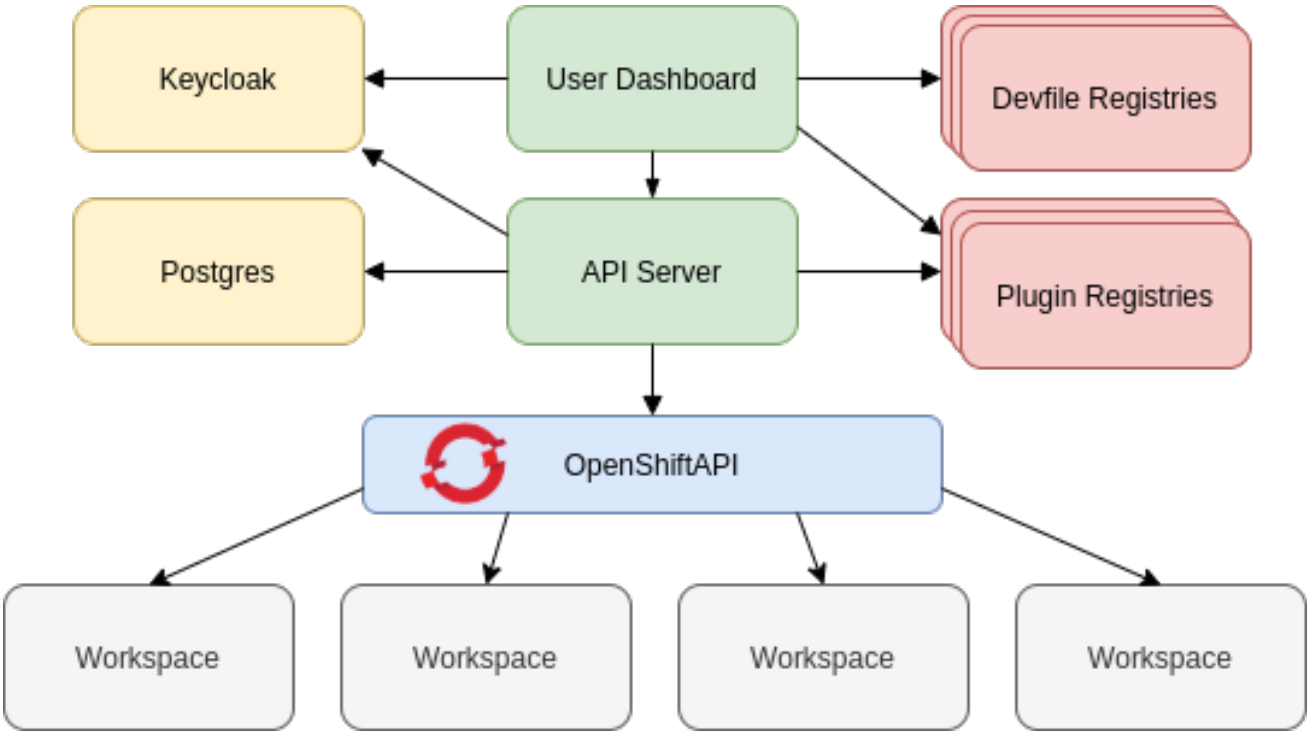
1.1.1. CodeReady Workspaces ワークスペースコントローラー

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 (CodeReady Workspaces ワークスペース) を管理します。以下のデプロイメントシナリオを利用できます。

- **Single-user**: デプロイメントには認証サービスは含まれません。開発環境のセキュリティーは保護されません。この設定に必要なリソースは少なくなります。これは、ローカルインストールにより適しています。
- **Multi-user**: これはマルチテナント設定です。開発環境のセキュリティーは保護され、この設定ではより多くのリソースが必要になります。クラウドインストールに適しています。

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。RH-SSO および PostgreSQL は、マルチユーザー設定でのみ必要となることに注意してください。

図1.2 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」

1.1.2. CodeReady Workspaces サーバー

CodeReady Workspaces サーバーは、ワークスペースコントローラーの中心となるサービスです。これは HTTP REST API を公開して CodeReady Workspaces ワークスペースを管理し、マルチユーザーモードで CodeReady Workspaces ユーザーを管理する Java Web サービスです。

コンテナーイメージ	**eclipse/che-server**

関連情報

- CodeReady Workspaces サーバーコンポーネントの詳細な設定オプション

1.1.3. 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 Angular フロントエンドアプリケーションです。CodeReady Workspaces ユーザーは、ユーザーダッシュボードでブラウザーから CodeReady Workspaces ワークスペースを作成し、起動し、管理します。

コンテナーイメージ	**eclipse/che-server**

1.1.4. CodeReady Workspaces Devfile レジストリー

CodeReady Workspaces devfile レジストリーは、そのまま使用できるワークスペースを作成するための CodeReady Workspaces スタックの一覧を提供するサービスです。このスタックの一覧

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。

devfile レジストリーのカスタマイズに関する詳細は、「devfile レジストリーのカスタマイズ」についてのセクションを参照してください。

コンテナーイメージ	**registry.redhat.io/codeready-workspaces/devfileregistry-rhel8:2.9**

1.1.5. 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。

コンテナーイメージ	**registry.redhat.io/codeready-workspaces/pluginregistry-rhel8:2.9**

1.1.6. CodeReady Workspaces および PostgreSQL

PostgreSQL データベースは、マルチユーザーモードで 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 PostgreSQL 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 PostgreSQL インスタンスを起動することを選択できます。

CodeReady Workspaces サーバーはデータベースを使用してユーザー設定（Workspaces メタデータ、Git 認証情報) を永続化させます。RH-SSO は、データベースをバックエンドとして使用し、ユーザー情報を永続化させます。

コンテナーイメージ	**registry.redhat.io/rhel8/postgresql-96:1**

1.1.7. CodeReady Workspaces および RH-SSO

RH-SSO は、マルチユーザーモードで 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。

コンテナーイメージ	**registry.redhat.io/rh-sso-7/sso74-openshift-rhel8:7.4**

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

1.2.1. 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複数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。

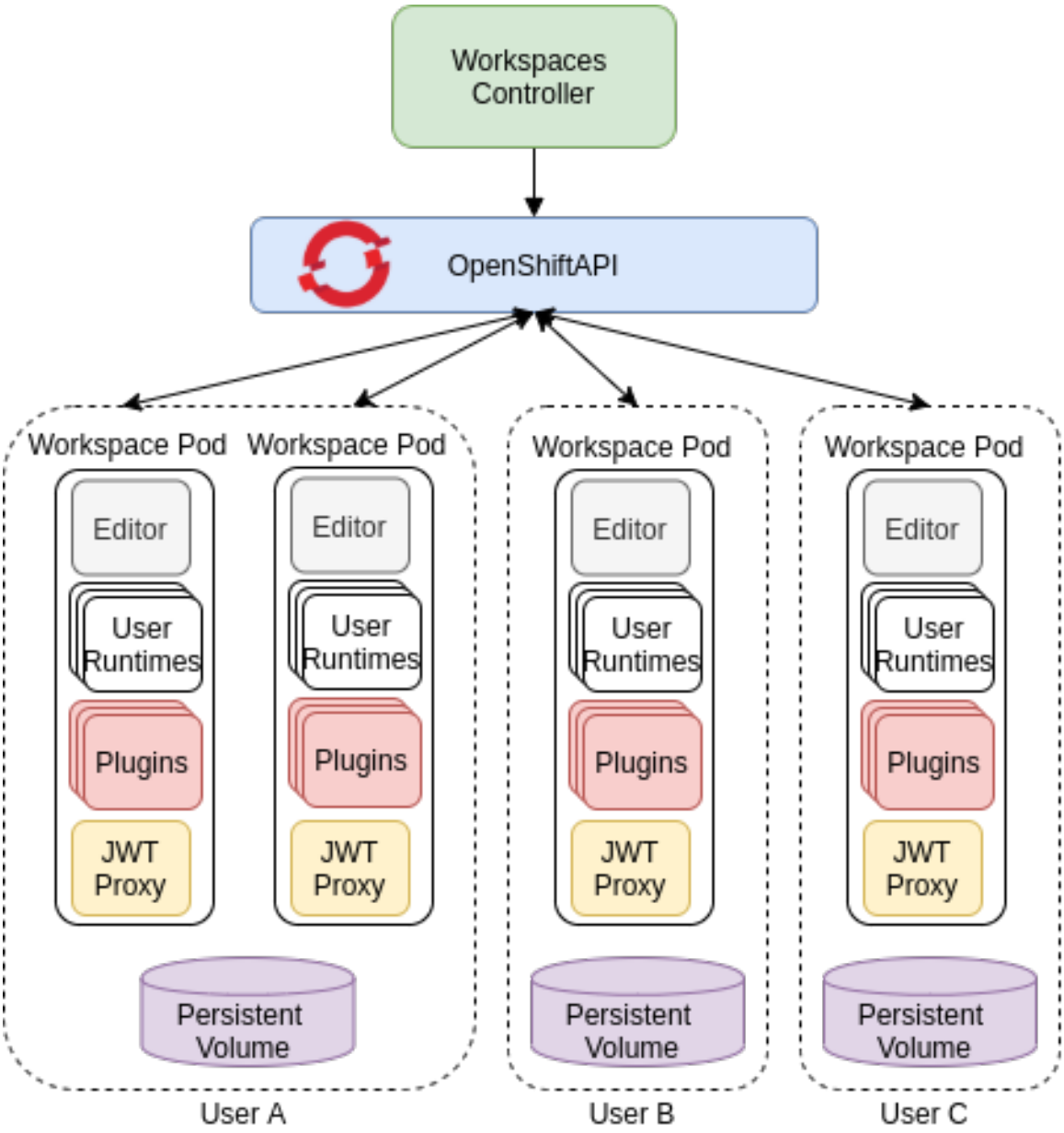
- ConfigMap
- service
- endpoint
- ingress/route
- Secret
- PV

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、OpenShiftPersistentVolume** 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1.3 CodeReady Workspaces ワークスペースコンポーネント



この図では、ユーザー A に属する 2 つのワークスペース、ユーザーB、もう 1 つは User C に所属するワークスペースという 4 つがあります。

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。

1.2.2. 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。

1.2.2.1. 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

1.2.2.2. 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 要素アプリケーション開発/実稼働環境のパリティーです。

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。

1.2.2.3. 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。CodeReady Workspaces ワークスペース JWT プロキシーは、CodeReady Workspaces サーバーがマルチユーザーモードで設定されている場合のみ、CodeReady Workspaces ワークスペースに含まれます。

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**

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

プラグインブローカーは、プラグインの **meta.yaml** ファイルを指定して、特別なサービスです。

- 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)。

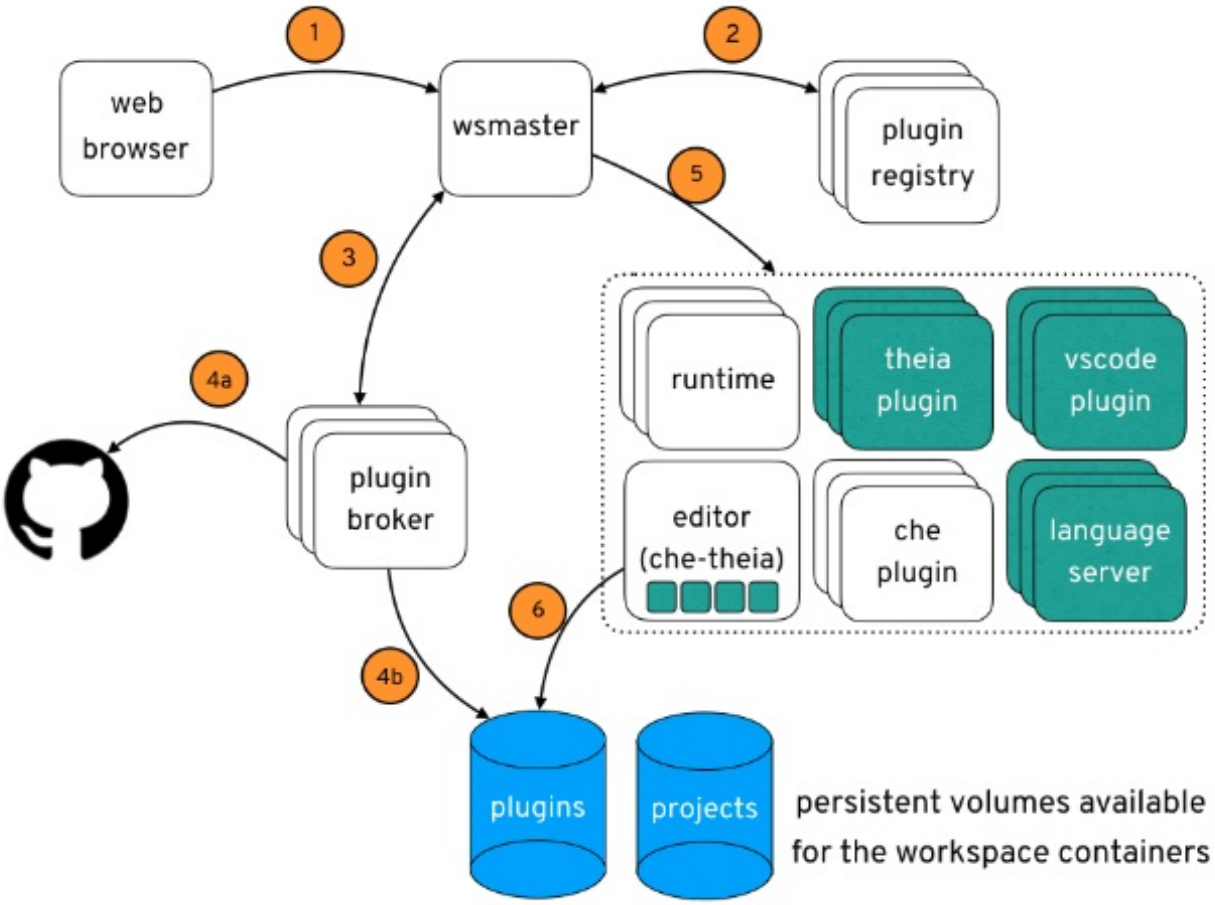
プラグインブローカーの主な目的は、CodeReady Workspaces がサポートできる実際のプラグインから CodeReady Workspaces プラグイン定義を切り離すことにあります。ブローカーでは、CodeReady Workspaces は CodeReady Workspaces サーバーを更新せずに異なるプラグインをサポートできます。

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（例： **eclipse/che-plugin-broker**）として定義されます。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。**Che Plugin** と **Che Editor** の 2 種類のプラグインがサポートされます。

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**

1.2.3. 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。

第2章 CODEREADY WORKSPACES リソース要件の計算

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、CodeReady Workspaces サーバーコンポーネント の Advanced configuration options 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 ワークスペースコントローラーについて」

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 のプラグインを使用す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. Authoring devfiles バージョン 2 の **components** セクションに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 は暗黙的にデフォルトの **cheEditor**:**che-theia**、およびコマンドの実行を許可する **chePlugin** を読み込みます (**che-machine-exec-plugin**)。デフォルトのエディターを変更するには、devfile に **cheEditor** コンポーネントセクションを追加します。

   b. CodeReady Workspaces **がマルチユーザーモードで実行されている場合は、JWT プロキシーコンポーネントを読み込みます**。JWT プロキシーは、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:
   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を示しています。また、デフォルトのクラスター全体を変更できるように対応する CodeReady Workspaces サーバープのロパティーも表示します。

   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

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
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 che Editors 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 属性は、chePlugins または che Editors コンポーネントのメモリー制限を定義します。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、コンテナーに以下のメモリー制限および要求が設定されます。

Memory limit	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、フォルダー v3/plugins/${organization}/${name}/${version}/ の CodeReady Workspaces プラグインレジストリーリポジトリーで利用できます。

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 制限	CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__LIMIT__CORES
CPU 要求	CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__REQUEST__CORES

CodeReady Workspaces サーバーコンポーネントの高度な設定オプション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ます。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 および memoryRequest 属性（ある場合）は、dockerimage コンテナーのメモリー制限を定義します。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: guestbook-nodejs-sample-
projects:
 - name: guestbook-nodejs-sample
   source:
     type: git
     location: "https://github.com/l0rd/nodejs-sample"
components:
 - type: chePlugin
   id: che-incubator/typescript/latest
 - type: kubernetes
   alias: guestbook-frontend
   reference: https://raw.githubusercontent.com/l0rd/nodejs-sample/master/kubernetes-
manifests/guestbook-frontend.deployment.yaml
   mountSources: true
   entrypoints:
     - command: ['sleep']
       args: ['infinity']
```

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	
ワークスペース	Theia-ide（デフォルト **cheEditor**）	512MiB	512MiB
ワークスペース	machine-exec (default **chePlugin**)	128MiB	128MiB
ワークスペース	vscode-typescript (**chePlugin**)	512MiB	512MiB
ワークスペース	frontend(**kubernetes**)	1GiB	512MiB
JWT プロキシー	verifier	128MiB	128MiB

Pod	コンテナー名	デフォルトのメモリー制限
	合計	2..25GiB 1..75GiB

- theia-ide および machine-exec コンポーネントは、devfile に含まれていない場合でも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。
- ia-ide のリソースは、具体的に cheEditor meta.yaml で memoryLimit として 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、Kubernetes コンポーネントタイプのデフォルト （メモリー制限 1 GiB およびメモリー要求の 512 MiB）を適用します。これは、kubernetes コンポーネントが、リソースの制限や要求のないコンテナー仕様を持つ Deployment マニフェストを参照するためです。
- 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 1.75 GiB のメモリー要求が設定されます。

関連情報

- 1章*CodeReady Workspaces アーキテクチャーの概要*
- CodeReady Workspaces インストールの設定
- CodeReady Workspaces サーバーコンポーネントの詳細な設定オプション
- devfile のバージョン 2 のオーサリング
- 「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と、devfile で参照されているすべてのサンプルプロジェクトも zip ファイルとして含まれます。

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。「devfiles バージョン 2 の承認」を参照してください。

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-2.9-rhel-8
```

2. ./dependencies/che-devfile-registry/devfiles/ ディレクトリーで、<devfile-name>/ サブディレクトリーを作成し、devfile.yaml 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 形式の詳細は、「devfiles バージョン 2 のオーサリング」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することを確認します。

表3.1 devfile meta.yaml のパラメーター

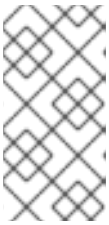
属性	説明
説明	ユーザーダッシュボードに表示される説明。
displayName	ユーザーダッシュボードに表示される名前。
globalMemoryLimit	devfile が起動するすべてのコンポーネントによって消費されることが予想されるメモリーの合計。この数字はユーザーダッシュボードに表示されます。これは情報を示唆するように提供されますが、CodeReady Workspaces サーバーでは考慮されません。
icon	ユーザーダッシュボードに表示される **an.svg** ファイルへのリンク。
tags	タグの一覧。タグには、通常、スタックに含まれるツールが含まれます。

例3.2 devfile meta.yaml の例

```
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 スクリプトの完全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

- **devfile のバージョン 2 のオーサリング。**
- **「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

- NodeJS 12.x
- yarn の実行中のバージョン。参照： Installing Yarn.
- ./node_modules/.bin は PATH 環境変数に置かれます。
- 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-2.9-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、「VS Code 拡張機能の Che プラグインレジストリーへの追加」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 スクリプトの完全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存在する ./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生成されるすべての meta.yaml ファイルはここに置かれます。**

```
./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 および my-devfile-registry イメージは、docker コマンドを使用してビルドされます。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/ ディレクトリーにあります。

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利用できます。**
  - **ワークスペースの詳細の Devfile タブの chePlugin コンポーネントへの完了**

- ワークスペースのプラグイン Che-Theia ビュー

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

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

次のセクションでは、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 - コマンドパレットコマンドのテキスト入力を使用して、新しいカスタムプラグインレジストリーを迅速に追加します。この方法では、ユーザーはプラグインレジストリーの URL や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する: 新規の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. Command Palette を使用したカスタムプラグインレジストリーの追加

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. 検索ボックスに Add Registry コマンドを入力し、Enter を一度入力します。

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、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および追加する主な CodeReady Workspaces 設定 メニューの使用方法について説明します。

前提条件

- CodeReady Workspaces のインスタンス

手順

1. メインの CodeReady Workspaces 画面で、Ctrl を押して Open Preferences を選択します。あるいは 、、Open Preferences を選択します。

2. Che プラグインを選択し 、設定.json リンクで編集を継続します。

   setting.json ファイルが表示されます。

3. 以下に示すように 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. 変更を保存して、カスタムプラグインレジストリーを既存の CodeReady Workspaces ワークスペースに追加します。

- 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 プラグインレジストリー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 に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リストで、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あり、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 ドキュメントのイベントの詳細な一覧について参照してください。

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 RunningPod

```
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、サーバーと **IDE** の **2** つの部分で構成されます。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 と入力し、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 and press Enter
   - component=postgres and press Enter

   OpenShift Web コンソールは、これらの 2 つのキーで検索を行い、PostgreSQL ロ

グを表示します。

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 は自動的に、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開始します。
- crwctl server:logs が既存の Red Hat CodeReady Workspaces サーバーログを収集します。
- crwctl workspace:logs が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。

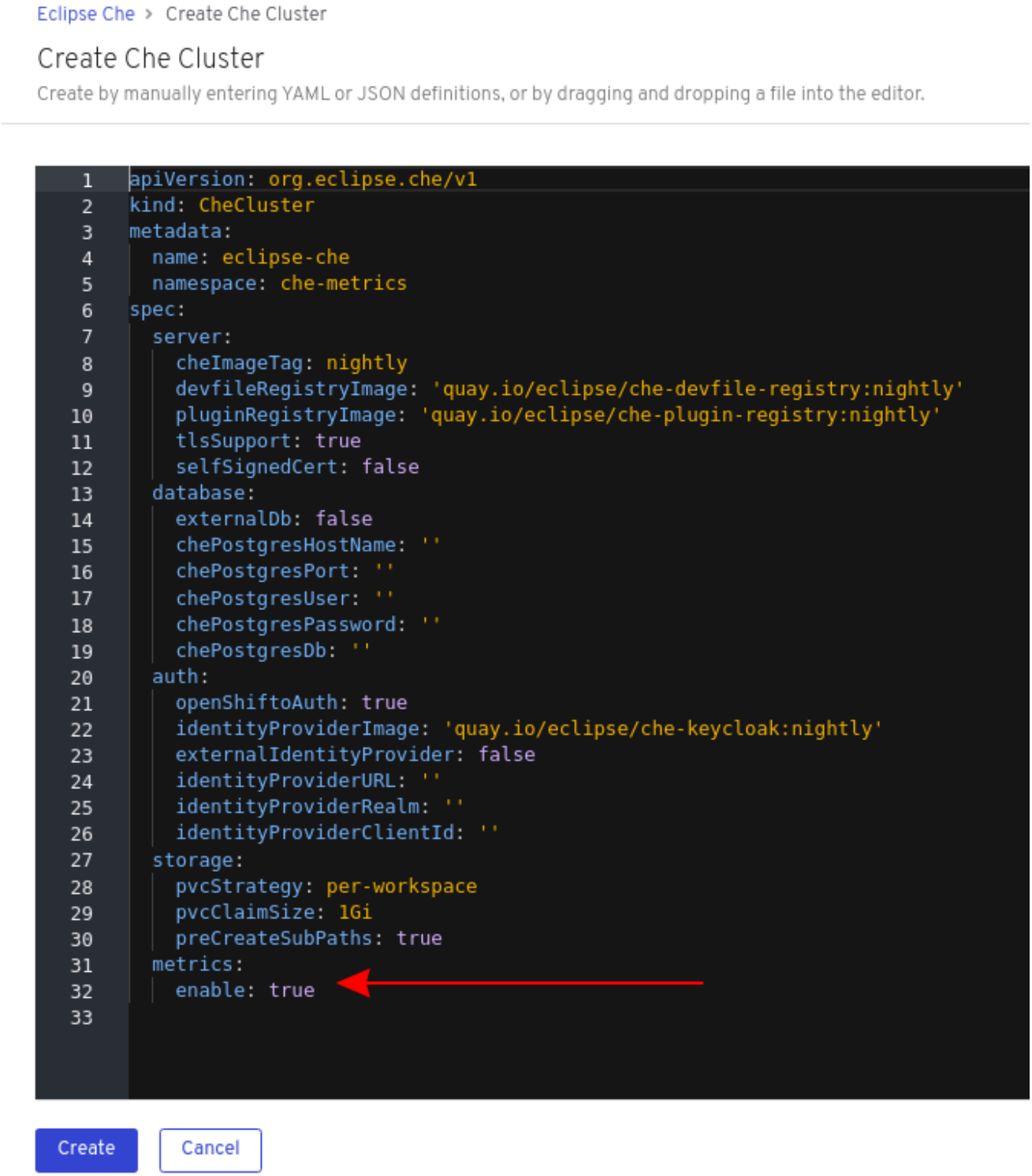
5.1. CODEREADY WORKSPACES メトリクスの有効化および公開

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 からインストールされると、デフォルトの CheCluster CR が使用され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

このセクションでは、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 がポート 8087 でメトリクスを公開している。che メトリクスの有効化および公開について参照してください。

- Prometheus 2.9.1 以降が実行中である。Prometheus コンソールは、対応するサービスとルートを持つポート 9090 で実行されている。「最初 steps with Prometheus」を参照してください。

手順

- 8087 ポートからメトリクスをスクレープするように Prometheus を設定します。

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 という単一のジョブがあります。これは、CodeReady Workspaces サーバーによって公開される時系列データを収集します。

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。

  メトリクスは http://<che-server-url>:9090/metrics で利用できます。

  詳細は、Prometheus ドキュメントの「Using the expression browser」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 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、ライブラリーをインストルメント化し、その使用がユーザーに適切ではありません。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。

6.1. トレース API

CodeReady Workspaces はインストルメンテーションに 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、新しい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ます。

6.2. 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。

6.3. 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 についての詳細は、Jaeger の Web サイトを参照してください。

6.3.1. 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 を入力します。
4. Jaeger Operator タイルをクリックします。

5. Jaeger Operator の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に特定のプロジェクトを選択し、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、エラーメッセージ「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る。

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. **Save** をクリックします。

19. **OpenShift は Jaeger cluster 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。

6.3.2. 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。

6.4. 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、以下の設定変数を spec.server.customCheProperties に追加します。

```
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と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、コレクター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、CodeReady Workspaces サーバーコンポーネントの高度な設定オプションについて参照してください。
- 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。

6.5. 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 操作のトレースの概要を示します。

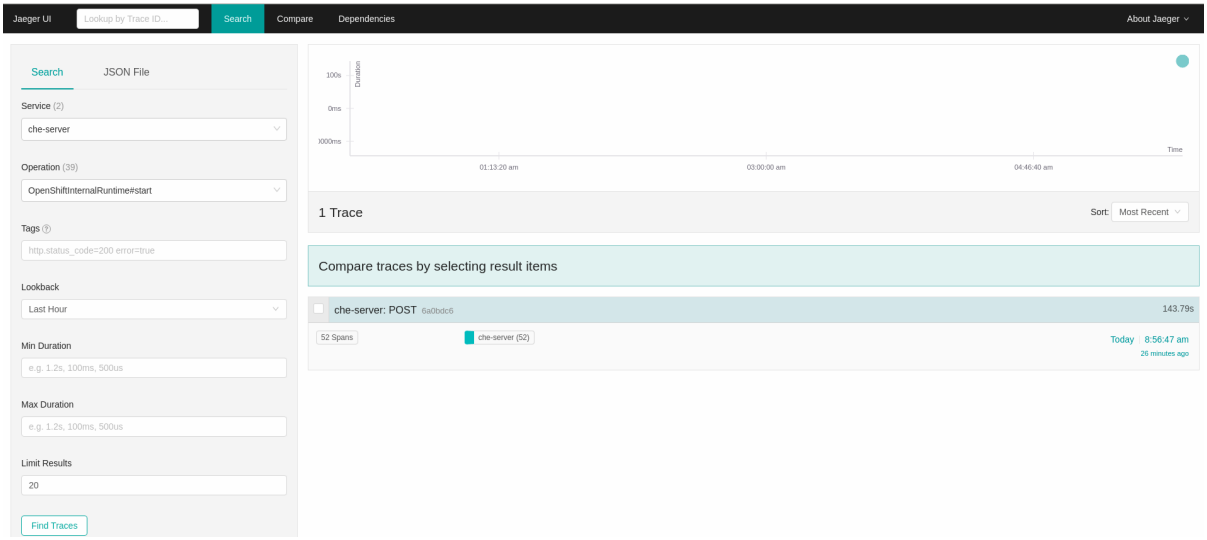
手順

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。

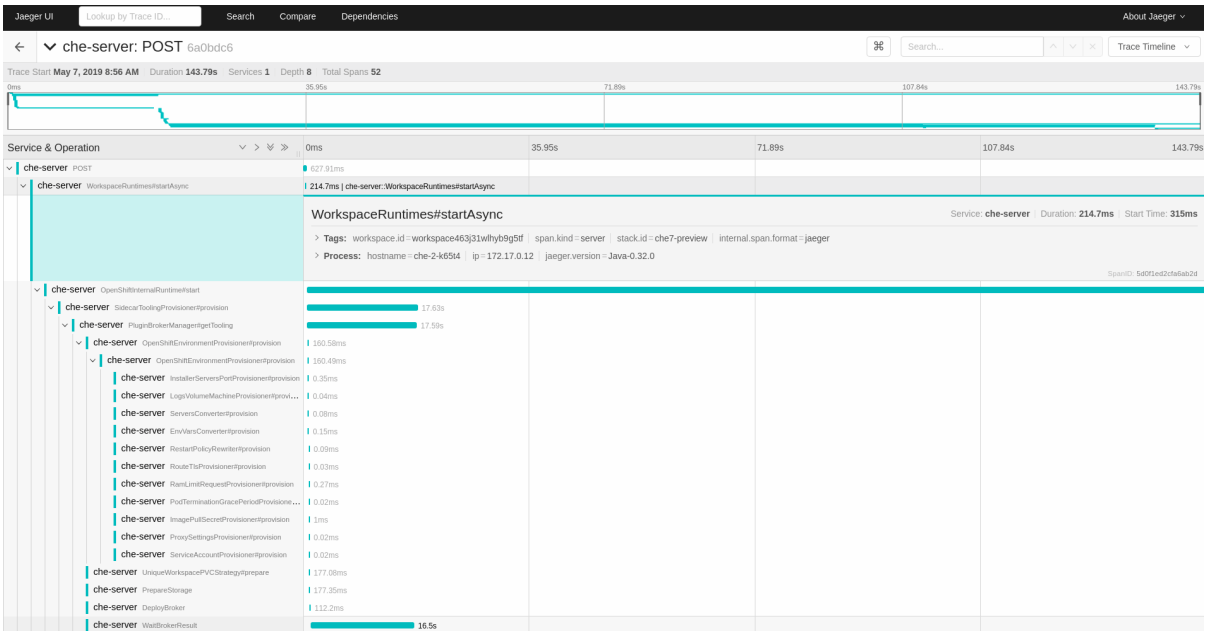
1. 左側の Search パネルで、操作名（スパン名)、タグ、または期間でスパンをフィルターします。

図6.1 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6.2 拡張されたトレースツリー



6.6. 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 のトレース実装の中核は、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 を使用して行います。@Traced アノテーションをメソッドに追加すると、トレースインターセプターはトレーススパンを追加します。

6.6.1. 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、workspace ID、および stackId などの追加のタグがあります。TracingFilter によって作成されるスパンには、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 の使用方法についての詳細は、Jaeger ドキュメントの『Jaeger Getting Started Guide』を参照してください。

第7章 バックアップおよび障害復旧

本セクションでは、CodeReady Workspaces のバックアップおよび障害復旧機能の複数の側面について説明します。

- 「外部データベースの設定」
- 「永続ボリュームのバックアップ」

7.1. 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)

7.1.1. 外部 PostgreSQL の設定

手順

1. 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などを永続化させます。

```
CREATE USER <database-user> WITH PASSWORD '<database-password>' 1 2
CREATE DATABASE <database>
GRANT ALL PRIVILEGES ON DATABASE <database> TO <database-user> 3
ALTER USER <database-user> WITH SUPERUSER
```

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

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

3 CodeReady Workspaces サーバーデータベースの名前

2.

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させます。

```
CREATE USER keycloak WITH PASSWORD '<identity-database-password>' (1)
CREATE DATABASE keycloak
GRANT ALL PRIVILEGES ON DATABASE keycloak TO keycloak
```

(1) RH-SSO データベースのパスワード

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

前提条件

- oc ツールが利用可能である。

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)

3. 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

4. パッチを適用して crwctl コマンドを実行して Red Hat CodeReady Workspaces をデプロイします。以下に例を示します。

```
$ 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.

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

7.2. 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。

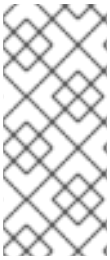
データの損失を防ぐには、PV を定期的にバックアップします。PV を含む OpenShift リソースのバックアップおよび復元には、ストレージに依存しないツールを使用することが推奨されます。

7.2.1. 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

第8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 を使用します。Image Puller は追加の OpenShift デプロイメントです。これは、各ノードで関連するコンテナーイメージをダウンロードし、実行する DaemonSet を作成します。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状態です。

Image Puller は、設定の以下のパラメーターを提供します。

表8.1 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**
イメージ	**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>** を参照してください。「プルするイメージの一覧の定義」	

パラメーター	使用法	デフォルト
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	**'{}'**
アフィニティー	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	DeamonSet によって作成される Pod に追加するには **pullsecret1;...** 形式で、イメージプルシークレットの一覧。これらのシークレットはイメージ puller の namespace にある必要があり、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

8.1. プルするイメージの一覧の定義

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. 関連するコンテナーイメージの一覧を取得します。

例8.1 CodeReady Workspaces 2.9 のすべてのイメージ一覧の取得

```
$ curl -sSLo- https://raw.githubusercontent.com/redhat-developer/codeready-workspaces-images/crw-2.9-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サポートされ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8.2 AMD64 および Intel 64(x86_64)のイメージ一覧 （openj9 イメージを除く）

```
che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-
workspaces/pluginbroker-artifacts-
rhel8@sha256:c0a096ed1829b21b0641d98c591139324affc0c866a5e011ae6627dd22d
316c6;
che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:cd977a7e55dff0d478be81762db17dad3e9daf2cc8f80d168f6e3adcccca3
986;
codeready_workspaces_machineexec=registry.redhat.io/codeready-
workspaces/machineexec-
rhel8@sha256:7bc2f046c1fcb00ab7656de7a7c31da73e71af3164314f77960061102f44
c9da;
codeready_workspaces_plugin_java11=registry.redhat.io/codeready-
workspaces/plugin-java11-
rhel8@sha256:e86be19ea5c4868f68188a46cec3f4131c1a18400f6800b27d5da92c95c
2418b;
codeready_workspaces_plugin_java8=registry.redhat.io/codeready-
workspaces/plugin-java8-
rhel8@sha256:bb8ec9c4964e39b335be589f027c4b5ec866fe5f742c7181473e70fff7176
55d;
codeready_workspaces_plugin_kubernetes=registry.redhat.io/codeready-
workspaces/plugin-kubernetes-
rhel8@sha256:f40c6cf67df5294f5d767312633a5ce5cf8b542f660f52f3252c61d6c5342f
9e;
codeready_workspaces_plugin_openshift=registry.redhat.io/codeready-
workspaces/plugin-openshift-
rhel8@sha256:a8f9e97a6946bbd5e0ac515621408cb361531bc33c99983f17aaf547a44
776a2;
codeready_workspaces_stacks_cpp=registry.redhat.io/codeready-
workspaces/stacks-cpp-
rhel8@sha256:8855029f8658060e79513bd51bad33e38134adf0586286ef2a02bdf27e9
dd526;
codeready_workspaces_stacks_dotnet=registry.redhat.io/codeready-
workspaces/stacks-dotnet-
rhel8@sha256:0422cfa23c5eefdda978bbec4f1fc2c24fccfa5a9568d5bda69e3232efefec
54;
codeready_workspaces_stacks_golang=registry.redhat.io/codeready-
workspaces/stacks-golang-
rhel8@sha256:de9daf2d87903ebabf03f64e4c15d6f4f7106525396f31bad555ce858864
219c;
codeready_workspaces_stacks_php=registry.redhat.io/codeready-
workspaces/stacks-php-
rhel8@sha256:59c41da505474c48617cec3759064db55bab69bf8cd6f9752bfa95b8756
dc710;
codeready_workspaces_theia=registry.redhat.io/codeready-workspaces/theia-
rhel8@sha256:925be51cad7d93eae735039cd76569ef14bdf546876584479a5b037758
```

```
defc7b;
codeready_workspaces_theia_endpoint=registry.redhat.io/codeready-
workspaces/theia-endpoint-
rhel8@sha256:42cd00002085280c39534ad856df6adac925218be07ee652b78d92c24cf
71573;
jboss_eap_7_eap73_openjdk8_openshift_rhel7=registry.redhat.io/jboss-eap-7/eap73-
openjdk8-openshift-
rhel7@sha256:d16cfe30eaf20a157cd5d5980a6c34f3fcbcfd2fd225e670a0138d81007dd
919;
jboss_eap_7_eap_xp2_openjdk11_openshift=registry.redhat.io/jboss-eap-7/eap-xp2-
openjdk11-openshift-
rhel8@sha256:647d092383a760edc083eafb2d7bc3208d6409097281bedbd5eaccde36
0e7e39;
pvc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:2f6b88c037c0503da7704bccd3fc73cb76324101af39ad28f16460e7bc
e98324;
rhscl_mongodb_36_rhel7=registry.redhat.io/rhscl/mongodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0
dc73;
```

- **IBM Z および IBM Power Systems の場合は、java8 および java 11 に openj9 バージョンを使用し、dotnet を除外します。**

例8.3 IBM Z および IBM Power Systems のイメージ一覧： java8 および java 11 に 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:c0a096ed1829b21b0641d98c591139324affc0c866a5e011ae6627dd22d
316c6;
che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:cd977a7e55dff0d478be81762db17dad3e9daf2cc8f80d168f6e3adcccca3
986;
codeready_workspaces_machineexec=registry.redhat.io/codeready-
workspaces/machineexec-
rhel8@sha256:7bc2f046c1fcb00ab7656de7a7c31da73e71af3164314f77960061102f44
c9da;
codeready_workspaces_plugin_java11_openj9=registry.redhat.io/codeready-
workspaces/plugin-java11-openj9-
rhel8@sha256:3fa85acf846cf774276bd243e58eaaf3ff26cc53e2fbaf69ad336ef8a054e7
41;
codeready_workspaces_plugin_java8_openj9=registry.redhat.io/codeready-
workspaces/plugin-java8-openj9-
rhel8@sha256:2593b8602800de8e0713cf31708fec8c68dcb0d92f8da770ebe89d9310c
04e6a;
codeready_workspaces_plugin_kubernetes=registry.redhat.io/codeready-
workspaces/plugin-kubernetes-
rhel8@sha256:f40c6cf67df5294f5d767312633a5ce5cf8b542f660f52f3252c61d6c5342f
9e;
codeready_workspaces_plugin_openshift=registry.redhat.io/codeready-
workspaces/plugin-openshift-
rhel8@sha256:a8f9e97a6946bbd5e0ac515621408cb361531bc33c99983f17aaf547a44
```

```
776a2;
codeready_workspaces_stacks_cpp=registry.redhat.io/codeready-
workspaces/stacks-cpp-
rhel8@sha256:8855029f8658060e79513bd51bad33e38134adf0586286ef2a02bdf27e9
dd526;
codeready_workspaces_stacks_golang=registry.redhat.io/codeready-
workspaces/stacks-golang-
rhel8@sha256:de9daf2d87903ebabf03f64e4c15d6f4f7106525396f31bad555ce858864
219c;
codeready_workspaces_stacks_php=registry.redhat.io/codeready-
workspaces/stacks-php-
rhel8@sha256:59c41da505474c48617cec3759064db55bab69bf8cd6f9752bfa95b8756
dc710;
codeready_workspaces_theia=registry.redhat.io/codeready-workspaces/theia-
rhel8@sha256:925be51cad7d93eae735039cd76569ef14bdf546876584479a5b037758
defc7b;
codeready_workspaces_theia_endpoint=registry.redhat.io/codeready-
workspaces/theia-endpoint-
rhel8@sha256:42cd00002085280c39534ad856df6adac925218be07ee652b78d92c24cf
71573;
jboss_eap_7_eap73_openjdk8_openshift_rhel7=registry.redhat.io/jboss-eap-7/eap73-
openjdk8-openshift-
rhel7@sha256:d16cfe30eaf20a157cd5d5980a6c34f3fcbcfd2fd225e670a0138d81007dd
919;
jboss_eap_7_eap_xp2_openj9_11_openshift=registry.redhat.io/jboss-eap-7/eap-xp2-
openj9-11-openshift-
rhel8@sha256:7cdfbf16587dd8688f26595df43c8c30e069625997581e898368203cb059
c88b;
jboss_eap_7_eap_xp2_openjdk11_openshift=registry.redhat.io/jboss-eap-7/eap-xp2-
openjdk11-openshift-
rhel8@sha256:647d092383a760edc083eafb2d7bc3208d6409097281bedbd5eaccde36
0e7e39;
pvc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:2f6b88c037c0503da7704bccd3fc73cb76324101af39ad28f16460e7bc
e98324;
rhscl_mongodb_36_rhel7=registry.redhat.io/rhscl/mongodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0
dc73;
```

4. **Dockerfile でボリュームをマウントするコンテナーイメージの一覧から除外します。**

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」

8.2. 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 DaemonSet 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には、約 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」

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

このセクションでは、CodeReady Workspaces Operator を使用して Image Puller をインストールする方法を説明します。これはコミュニティーがサポートするテクノロジープレビュー機能です。

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。「OperatorHub を使用した OpenShift 4 への CodeReady Workspaces のインストール」を参照してください。

手順

1. CheCluster カスタムリソースを編集し、 set.spec.imagePuller.enable を trueに設定します。

例8.4 CheCluster カスタムリソースでの Image Puller の有効化

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

CODEREADY WORKSPACES OPERATOR を使用した IMAGE PULLER のアンインストール

- CheCluster カスタムリソースを編集し、.spec.imagePuller.enable を false に設定します。

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

例8.5 CheCluster カスタムリソースでの Image Puller の設定

```
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:c0a096ed1829b21b0641d98c591139324affc0c866a5e011ae6627dd22d
316c6;che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:cd977a7e55dff0d478be81762db17dad3e9daf2cc8f80d168f6e3adcccc
a3986;codeready_workspaces_machineexec=registry.redhat.io/codeready-
workspaces/machineexec-
rhel8@sha256:7bc2f046c1fcb00ab7656de7a7c31da73e71af3164314f77960061102f44
c9da;codeready_workspaces_plugin_java11=registry.redhat.io/codeready-
workspaces/plugin-java11-
rhel8@sha256:e86be19ea5c4868f68188a46cec3f4131c1a18400f6800b27d5da92c95c2
418b;codeready_workspaces_plugin_java8=registry.redhat.io/codeready-
workspaces/plugin-java8-
rhel8@sha256:bb8ec9c4964e39b335be589f027c4b5ec866fe5f742c7181473e70fff7176
55d;codeready_workspaces_plugin_kubernetes=registry.redhat.io/codeready-
workspaces/plugin-kubernetes-
rhel8@sha256:f40c6cf67df5294f5d767312633a5ce5cf8b542f660f52f3252c61d6c5342f
9e;codeready_workspaces_plugin_openshift=registry.redhat.io/codeready-
workspaces/plugin-openshift-
rhel8@sha256:a8f9e97a6946bbd5e0ac515621408cb361531bc33c99983f17aaf547a447
76a2;codeready_workspaces_stacks_cpp=registry.redhat.io/codeready-
workspaces/stacks-cpp-
rhel8@sha256:8855029f8658060e79513bd51bad33e38134adf0586286ef2a02bdf27e9
dd526;codeready_workspaces_stacks_dotnet=registry.redhat.io/codeready-
workspaces/stacks-dotnet-
rhel8@sha256:0422cfa23c5eefdda978bbec4f1fc2c24fccfa5a9568d5bda69e3232efefe
c54;codeready_workspaces_stacks_golang=registry.redhat.io/codeready-
workspaces/stacks-golang-
rhel8@sha256:de9daf2d87903ebabf03f64e4c15d6f4f7106525396f31bad555ce858864
219c;codeready_workspaces_stacks_php=registry.redhat.io/codeready-
workspaces/stacks-php-
rhel8@sha256:59c41da505474c48617cec3759064db55bab69bf8cd6f9752bfa95b8756
dc710;codeready_workspaces_theia=registry.redhat.io/codeready-
workspaces/theia-
rhel8@sha256:925be51cad7d93eae735039cd76569ef14bdf546876584479a5b037758d
```

```
efc7b;codeready_workspaces_theia_endpoint=registry.redhat.io/codeready-
workspaces/theia-endpoint-
rhel8@sha256:42cd00002085280c39534ad856df6adac925218be07ee652b78d92c24cf
71573;jboss_eap_7_eap73_openjdk8_openshift_rhel7=registry.redhat.io/jboss-eap-
7/eap73-openjdk8-openshift-
rhel7@sha256:d16cfe30eaf20a157cd5d5980a6c34f3fcbcfd2fd225e670a0138d81007d
d919;jboss_eap_7_eap_xp2_openjdk11_openshift=registry.redhat.io/jboss-eap-
7/eap-xp2-openjdk11-openshift-
rhel8@sha256:647d092383a760edc083eafb2d7bc3208d6409097281bedbd5eaccde36
0e7e39;pvc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:2f6b88c037c0503da7704bccd3fc73cb76324101af39ad28f16460e7b
ce98324;rhscl_mongodb_36_rhel7=registry.redhat.io/rhscl/mongodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0
dc73;'
```

注記

サポートされる 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 の公式ビルドの利点は、Red Hat が提供する追加のテストおよび検証の利点です。

CodeReady Workspaces のインストール時に Operator Hub での KubernetesImagePuller の使用を有効にして、コミュニティーがサポートす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

検証手順

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

2. eclipse-che namespace には、コミュニティーがサポートする Kubernetes Image Puller Operator ClusterServiceVersion が含まれます。

```
$ oc get clusterserviceversions
```

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

```
$ oc get deployments
```

4. 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
```

8.4. 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コミュニティーがサポートする 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たウィンドウで、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。

8.5. 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 ツールが利用可能である。
- 「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8.2 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-rhel8:2.9**
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8.3 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"**

値	使用法	デフォルト
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**
イメージ	ConfigMap の **IMAGES** の値	**'che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-workspaces/pluginbroker-artifacts-rhel8@sha256:c0a096ed1829b21b0641d98c591139324affc0c866a5e011ae6627dd22d316c6;che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-workspaces/pluginbroker-metadata-rhel8@sha256:cd977a7e55dff0d478be81762db17dad3e9daf2cc8f80d168f6e3adcccca3986; codeready_workspaces_machineexec=registry.redhat.io/codeready-workspaces/machineexec-rhel8@sha256:7bc2f046c1fcb00ab7656de7a7c31da73e71af3164314f77960061102f44c9da;codeready_workspaces_plugin_java11=registry.redhat.io/codeready-workspaces/plugin-java11-rhel8@sha256:e86be19ea5c4868f68188a46cec3f4131c1a18400f6800b27d5da92c95c2418b; codeready_workspaces_p**

値	使用法	デフォルト
		lugin_java8=registry.redhat.io/codeready-workspaces/plugin-java8-rhel8@sha256:bb8ec9c4964e39b335be589f027c4b5ec866fe5f742c7181473e70fff717655d;codeready_workspaces_plugin_kubernetes=registry.redhat.io/codeready-workspaces/plugin-kubernetes-rhel8@sha256:f40c6cf67df5294f5d767312633a5ce5cf8b542f660f52f3252c61d6c5342f9e;codeready_workspaces_plugin_openshift=registry.redhat.io/codeready-workspaces/plugin-openshift-rhel8@sha256:a8f9e97a6946bbd5e0ac515621408cb361531bc33c99983f17aaf547a44776a2;codeready_workspaces_stacks_cpp=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:8855029f8658060e79513bd51bad33e38134adf0586286ef2a02bdf27e9dd526;codeready_workspaces_stacks_dotnet=registry.redhat.io/codeready-workspaces/stacks-dotnet-rhel8@sha256:0422cfa23c5eefdda978bbec4f1fc2c24fccfa5a9568d5bda69e3232efefec54;codeready_workspace_stacks_golang=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:de9daf2d87903ebabf03f64e4c15d6f4f7106525396f31bad555ce858864219c;codeready_workspaces_stacks_php=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:59c41da505474c48617cec3759064db5

値	使用法	デフォルト
		5bab69bf8cd6f9752bfa95b3756d710;codeready_workspaces_Theia=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:925be51cad7d93eae735039cd76569ef14bdf546876584479a5b037758defc7b;codeready_workspaces_theia_endpoint=registry.redhat.io/codeready-workspaces/theia-endpoint-rhel8@sha256:42cd00002085280c39534ad856df6adac925218be07ee652b78d92c24cf71573;jboss_eap_7_eap73_openjdk8_openshift_rhel7=registry.redhat.io/jboss-eap-7/eap73-openjdk8-openshift-rhel7@sha256:d16cfe30eaf20a157cd5d5980a6c34f3fcbcfd2fd225e670a0138d81007dd919;jboss_eap_7_eap_xp2_openjdk11_openshift=registry.redhat.io/jboss-eap-7/eap-xp2-openjdk11-openshift-rhel8@sha256:647d092383a760edc083eafb2d7bc3208d6409097281bedbd5eaccde360e7e39;pvc_jobs=registry.redhat.io/ubi8/ubi-minimal@sha256:2f6b88c037c0503da7704bccd3fc73cb76324101af39ad28f16460e7bce98324;rhscl_mongodb_36_rhel7=registry.redhat.io/rhscl/mongodb -36-rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73; '
NODE_SELECTOR	ConfigMap の **NODE_SELECTOR** の値	**"{}"**

表8.4 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 <kubernetes-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 <kubernetes-image-puller>
```

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

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

第9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」
- 「Configuring OpenShift OAuth」

9.1. 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、ユーザーアイデンティティー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います。

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

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

9.1.1.1. 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、サブジェクト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。

9.1.1.2. 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 つの実装間の切り替えには、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方法 /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 /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 /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 /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 /。

9.1.1.3. 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。

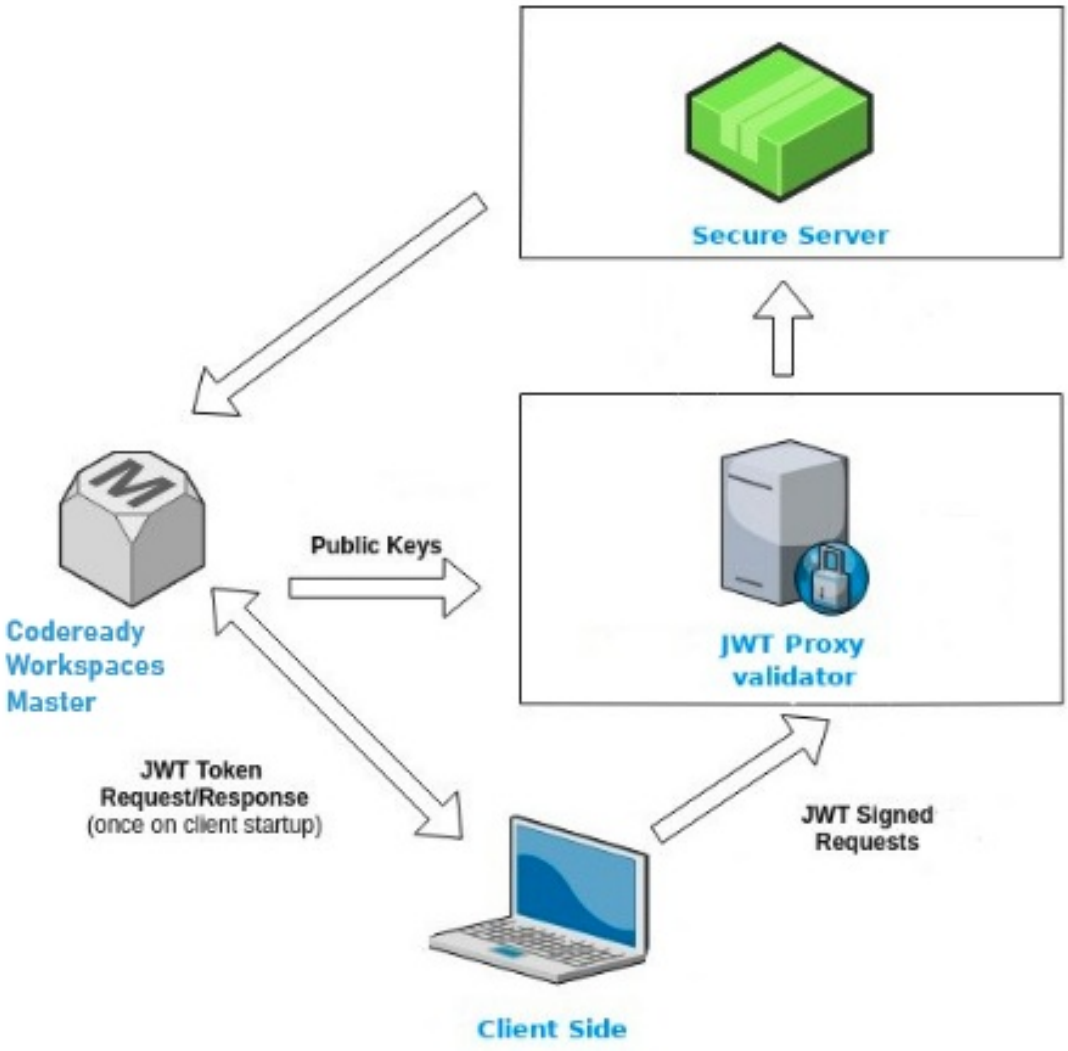
CodeReady Workspaces Swagger インターフェース (\https://codeready-<openshift_deployment_name>.<domain_name>/swagger) にアクセスします。アクセストークンが Request ヘッダーに含まれるように、ユーザーは RH-SSO で署名する必要があります。

9.1.2. 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9.1 ワークスペース内の認証



9.1.2.1. 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、エンドポイントの secure 属性を devfile の dockerimage type コンポーネントで 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'
```

9.1.2.2. 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。

9.1.2.3. 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 サーバーにクエリーするには、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。

9.2. 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。

9.2.1. 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、read、use、run、 set Permissions、および delete などのパーミッションが含まれます。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9.1 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。
delete	ワークスペースの削除を許可します。

9.2.2. 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9.2 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、CHE_SYSTEM_ADMIN__NAME プロパティーで設定した管理者ユーザーに付与されます（デフォルトは 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。ユーザーが CodeReady Workspaces ユーザーデータベースにない場合は、最初のユーザーのログイン後に表示されます。

9.2.3. 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。

9.2.4. 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。

9.2.5. 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。

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} 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。

9.2.6. 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。

9.2.7. CodeReady Workspaces パーミッションの共有

setPermissions 権限を持つユーザーはワークスペースを共有し 、他のユーザーの読み取り、使用、実行、設定、または setPermissions 権限を付与できます。

手順

ワークスペースパーミッションを共有するには、以下を実行します。

1. ユーザーダッシュボードでワークスペースを選択します。

2. Share タブに移動して、ユーザーのメール ID を入力します。複数のメールの区切り文字としてコンマまたはスペースを使用します。

9.3. 認証の設定

9.3.1. 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。

9.3.2. 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指定して、codeready-public クライアントを作成します。

- Valid Redirect URIs: 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

エイリアスである 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。

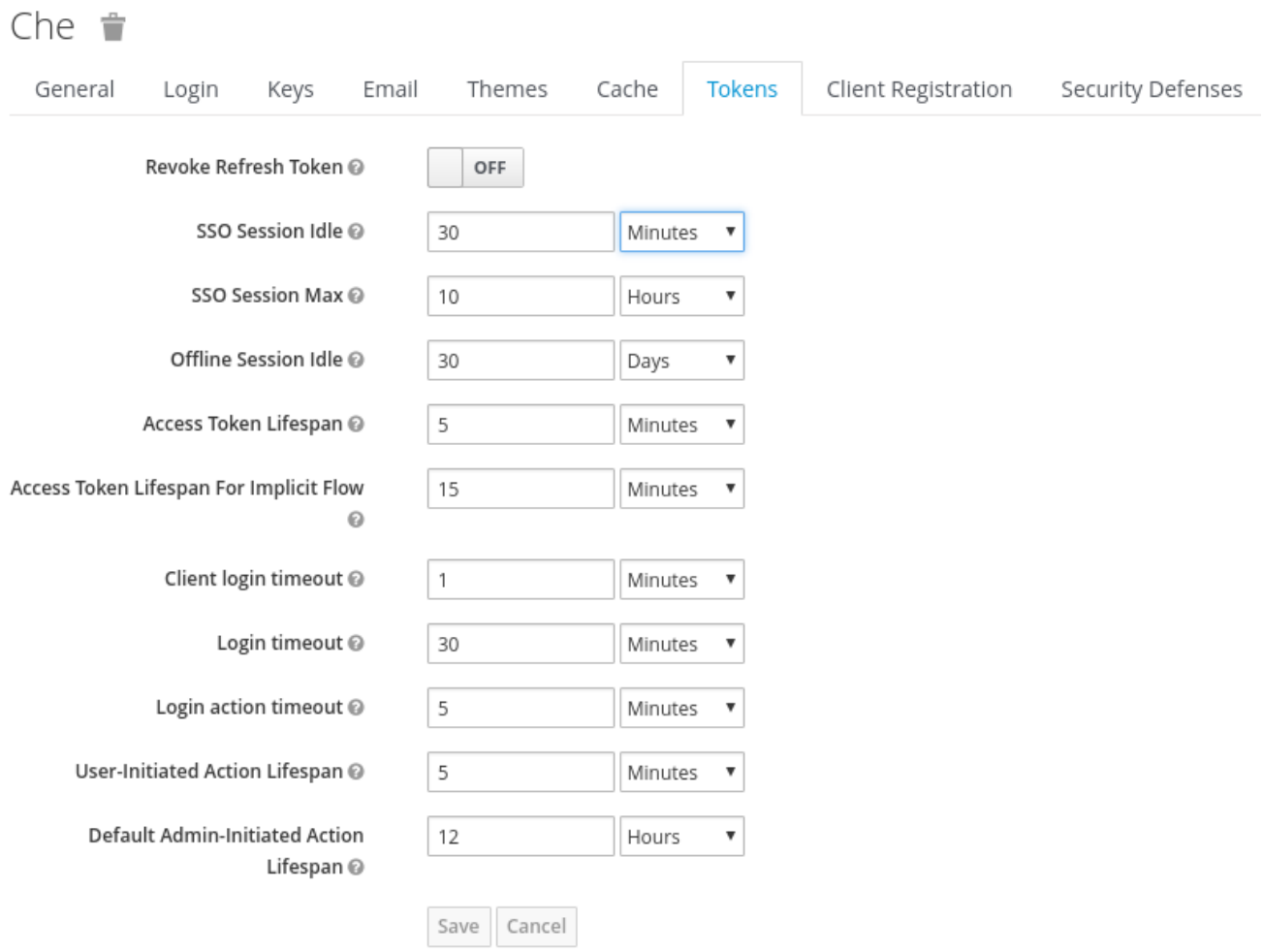
CORS エラー

無効な Web Origin がある場合に発生します。

9.3.3. 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。



9.3.4. 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 ページを参照してください。

9.3.5. 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。

9.3.5.1. 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 ツールが利用可能である。

手順

- CodeReady Workspaces URL をアプリケーションホームページ URL の値として使用し、GitHub で OAuth アプリケーションを作成し、RH-SSO GitHub エンドポイント URL を認可コールバック URL の値として作成します。デフォルト値は https://codeready-openshift-workspacesです。<DOMAIN>/ と https://keycloak-openshift-workspaces<DOMAIN>/auth/realms/codeready/broker/github/endpoint はそれぞれ<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。

9.3.5.2. Bitbucket サーバーの設定

Bitbucket 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、CHCHE_INTEGRATION_BITBUCKET_SERVER__ENDPOINTS プロパティーを使用して、Bitbucket サーバー URL を Red Hat CodeReady Workspaces に登録する必要があります。プロパティーの値には、登録するサーバーのホスト名が含まれる必要があります。Helm または Operator を使用して設定オプションを変更する方法の例は、以下を参照してください。

- CodeReady Workspaces サーバーコンポーネントの詳細な設定オプション

9.3.5.3. Bitbucket Server OAuth 1 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。
- SCM サーバーのプライベートリポジトリーでユーザーを認証する。

これにより、CodeReady Workspaces は 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 ツールが利用可能である。
- Bitbucket サーバーは、CodeReady Workspaces サーバーから利用できます。

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. **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: <...> 1
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: <...> 2
type: Opaque
data:
  private.key: <...> 3
  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。

4. 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。

a. 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。

a. 「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。

a. 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

b. **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。

関連情報

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
- SCM サーバーのプライベートリポジトリーでユーザーを認証する。

9.3.5.4. 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 CodeReady Workspaces に GitLab サーバー URL を登録し、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/
```

以下を使用して GitLab サーバーを設定する他の例については、以下を実行します。

- Operator - Operator を使用した CodeReady Workspaces サーバーの詳細設定について参照してください。

関連情報

- CodeReady Workspaces サーバーコンポーネントの詳細な設定オプション

9.3.5.5. Configuring GitLab OAuth2

GitLab の OAuth2 では、プライベート GitLab リポジトリーからの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 から利用できる。

手順

- CodeReady Workspaces をアプリケーション Name に、RH-SSO GitLab エンドポイント URL をリダイレクト URI の値として使用し、GitLab に Authorized 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です。<DOMAIN>/auth/realms/codeready/broker/gitlab/endpoint 。ここで<DOMAIN> は OpenShift クラスタードメインです。アプリケーション ID および Secret の値を保存します。3 つのタイプの GitLab OAuth2 アプリケーションはすべてサポートされます（ユーザーが所有するグループ所有、インスタンス全体）。

1. GitLab サーバーを参照する RH-SSO にカスタム 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 サーバーによって提供される Application ID フィールドの値

   Client Secret

   直前の手順で GitLab サーバーによって提供される Secret フィールドの値。

   認証 URL

   https://<GITLAB_DOMAIN>/oauth/authorize 形式を持つ URL

   トークン URL

   https://<GITLAB_DOMAIN>/oauth/token 形式を持つ URL

   スコープ

API write_repository openid など、以下のセットが含まれる必要があるスコープのセット（ただし、これらに限定されません）

トークンの保存

有効にする必要があります。

ストアトークンが読み取り可能

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。

関連情報

TLS キーに関連する GitLab へのアクセスが CodeReady Workspaces 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。

- 信頼できない TLS 証明書の CodeReady Workspaces へのインポート。
- 自己署名証明書を使用した Git リポジトリーをサポートする CodeReady Workspaces のデプロイ

9.3.6. 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。

9.3.7. 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 」を参照してください。

9.3.8. 外部の RH-SSO インストールを使用する CodeReady Workspaces の設定

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

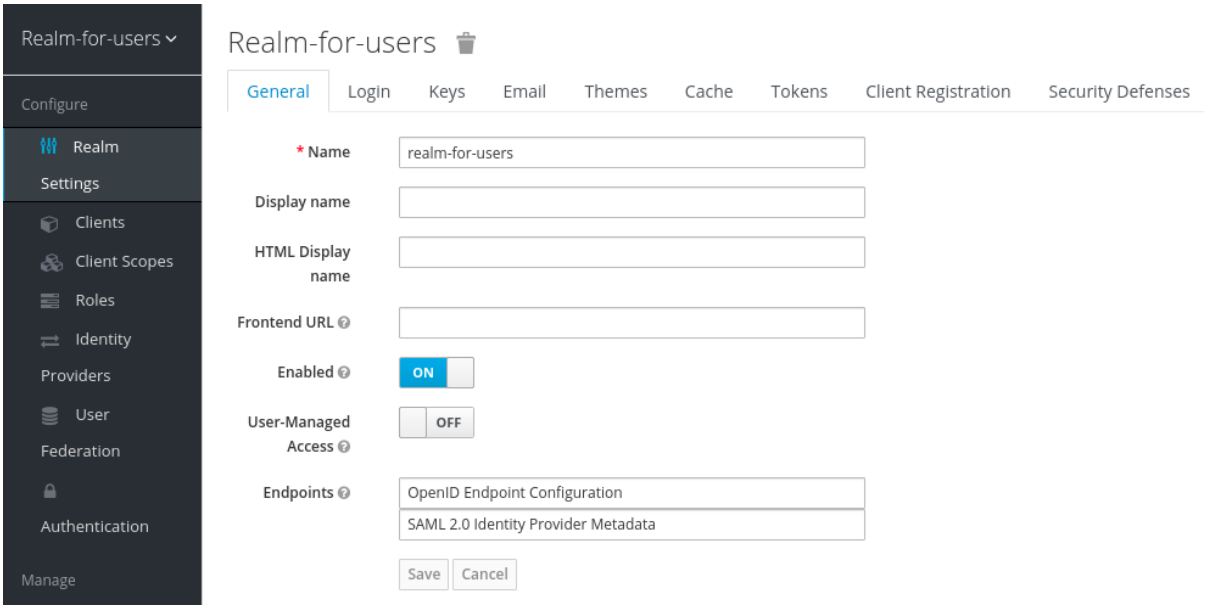
は、すでに定義されたユーザーを含む既存の RH-SSO インスタンス (複数のアプリケーションが使用する会社全体の RH-SSO サーバーなど) がある場合に役立ちます。

表9.3 サンプルで使用されるプレースホルダー

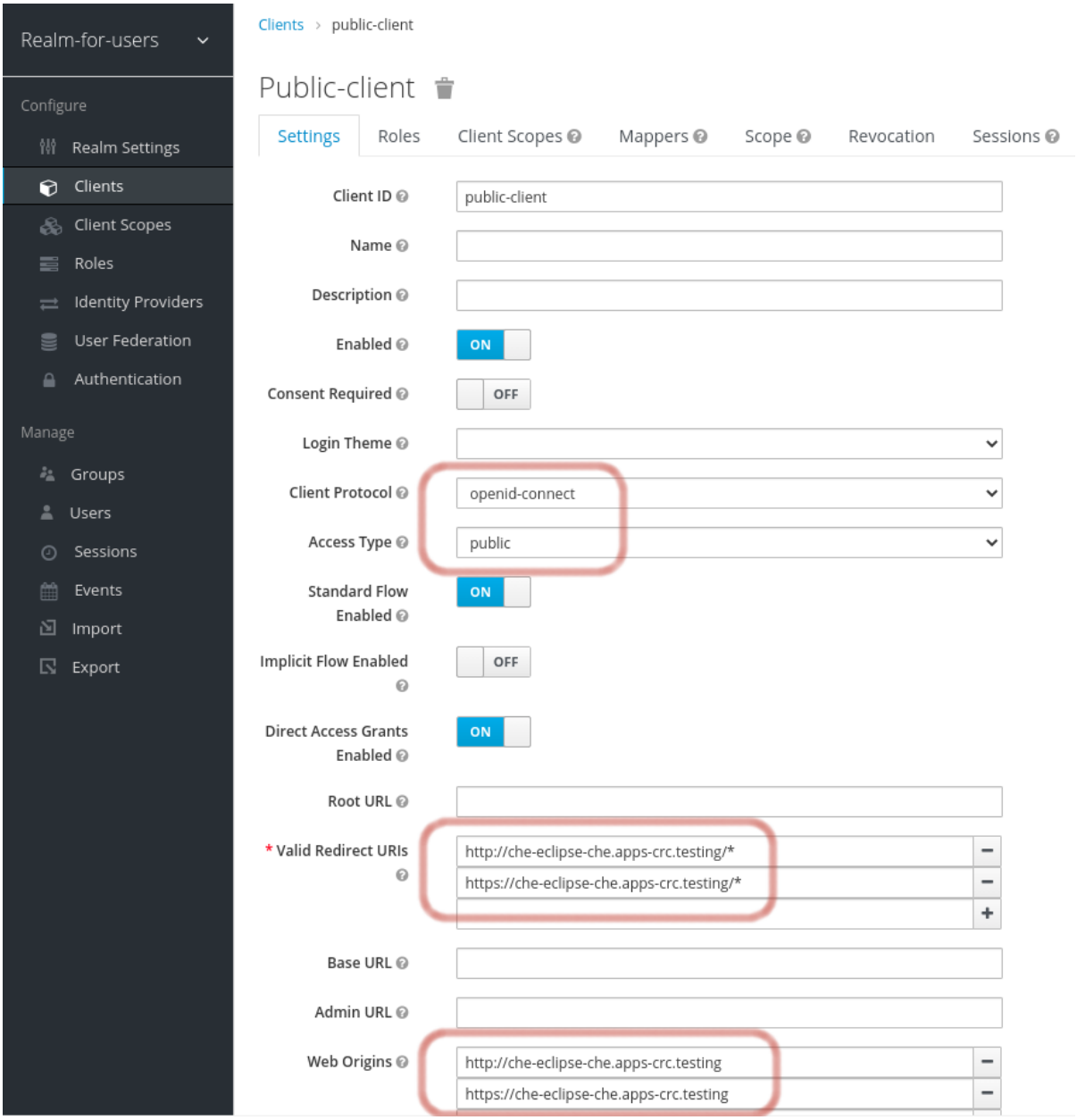
<provider-realm-name>	CodeReady Workspaces で使用することが意図された RH-SSO レルム名
<oidc-client-name>	**<provider-realm-name>**で定義される **oidc** クライアントの名前
<auth-base-url>	外部 RH-SSO サーバーのベース URL

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 に接続することが意図されたユーザーが含まれるレルムを定義します。**



- **このレルムで、CodeReady Workspaces がユーザーの認証に使用される 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 にする必要があります。
- Access Type は public でなければなりません。CodeReady Workspaces はパブリックアクセスタイプのみをサポートします。
- Valid Redirect URIs には、http プロトコルと他の 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 が含まれ、その後に /* ワイルドカードが必要です。
- Web Origins には、http プロトコルと他の 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 "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、RH-SSO を手動で設定します。手順については、OpenShift 3 [OpenShift3] または OpenShift4 [OpenShift4] の適切な RH-SSO ドキュメントを参照してください。

- 設定した RH-SSO では、ストアトークンとストア トークンの読み取り専用オプションが有効になります。

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>
```

9.3.9. 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 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。

9.3.10. 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。

9.4. CONFIGURING OPENSHIFT OAUTH

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。

OpenShift Connector の概要による認証は、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。

9.4.1. 初期ユーザーを使用した OpenShift OAuth の設定

前提条件

- oc ツールが利用可能である。
- crwctl 管理ツールが利用できる。「crwctl 管理ツールの使用」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 アイデンティティープロバイダーの設定ステップを省略し、OpenShift クラスターには設定済みの 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、crwctlserver:deploy 仕様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。

9.4.2. OpenShift 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、OpenShift の初期 OAuth ユーザーをプロビジョニングせずに OpenShift OAuth を設定する方法を説明します。

前提条件

- crwctl 管理ツールが利用できる。「crwctl 管理ツールの使用」を参照してください。

手順

1. OperatorHub を使用して CodeReady Workspaces をデプロイするために使用される場合、以下の値を codeready-workspaces カスタムリソース(CR)に設定します。

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

2. crwctl ツールを使用して CodeReady Workspaces のデプロイに使用される場合、--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 には以下が含まれます。

```
spec:
  auth:
    openShiftoAuth: true
    initialOpenShiftOAuthUser: ''
```

9.4.3. 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}]'
```

9.5. ユーザーデータの削除

9.5.1. 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)は、個人データを消去できるように個人の権利を強制します。

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、デフォルトの OpenShift プロジェクト openshift-workspaces を、-n オプションのユーザーの例として使用します。

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、管理者権限が必要です。管理者は、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、**CHE_SYSTEM_ADMIN__NAME** プロパティーで設定した管理者ユーザーに付与されます（デフォルトは **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。ユーザーが **CodeReady Workspaces** ユーザーデータベースにない場合は、最初のユーザーのログイン後に表示されます。

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、vparfono をローカルユーザー名として使用します。

```
$ 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 Resource(CR)定義を更新し、RH-SSO データベースからユーザーのデータを削除できるようにします。**

```
$ 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、CodeReady Workspaces のアンインストール手順に従います。