Red Hat CodeReady Workspaces 2.6

エンドユーザーガイド

Red Hat CodeReady Workspaces 2.6 の使用

Last Updated: 2023-02-03

Red Hat CodeReady Workspaces 2.6 エンドユーザーガイド

Red Hat CodeReady Workspaces 2.6 の使用

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

Red Hat CodeReady Workspaces を使用するユーザー向けの情報

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、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 のナビゲーション

本章では、Red Hat CodeReady Workspaces のナビゲーションについて利用できる方法を説明します。

- 「Dashboard を使用した CodeReady Workspaces のナビゲーション」
- 「証明書のブラウザーへのインポート」
- 「OpenShift Developer パースペクティブからの CodeReady Workspaces へのアクセス」

1.1. DASHBOARD を使用した CODEREADY WORKSPACES のナビゲーション

Dashboard は、**http://<https://codeready-<openshift_deployment_name>.<domain_name>>/dashboard/** などの URL からクラスターからアクセスできます。本セクションでは、OpenShift でこの URL にアクセスする方法を説明します。

1.1.1. OAuth を使用した OpenShift の CodeReady Workspaces への初回ログイン

本セクションでは、OAuth を使用して初めて OpenShift で CodeReady Workspaces にログインする方法を説明します。

前提条件

- OpenShift インスタンスの管理者に問い合わせ、**Red Hat CodeReady Workspaces の URL**を取得してください。

手順

1. **Red Hat CodeReady Workspaces URL**に移動し、Red Hat CodeReady Workspaces ログインページを表示します。
2. **OpenShift OAuth** オプションを選択します。
3. **Authorize Access** ページが表示されます。
4. **Allow selected permissions** ボタンをクリックします。
5. アカウント情報を更新します。**Username**、**Email**、**First name**、および **Last name** フィールドを指定し、**Submit** ボタンをクリックします。

検証手順

- ブラウザーに Red Hat CodeReady Workspaces **Dashboard** が表示されます。

1.1.2. 新規ユーザーとして登録するための OpenShift の CodeReady Workspaces へのログイン

本セクションでは、初めて新しいユーザーとして登録するために OpenShift の CodeReady Workspaces にログインする方法を説明します。

前提条件

- OpenShift インスタンスの管理者に問い合わせ、**Red Hat CodeReady Workspaces の URL**を取得してください。

手順

1. **Red Hat CodeReady Workspaces URL**に移動し、Red Hat CodeReady Workspaces ログインページを表示します。
2. **Register as a new user**オプションを選択します。
3. アカウント情報を更新します。**Username**、**Email**、**First name**、および **Last name** フィールドを指定し、**Submit** ボタンをクリックします。

検証手順

- ブラウザーに Red Hat CodeReady Workspaces **Dashboard** が表示されます。

1.1.3. crwctl を使用した CodeReady Workspaces へのログイン

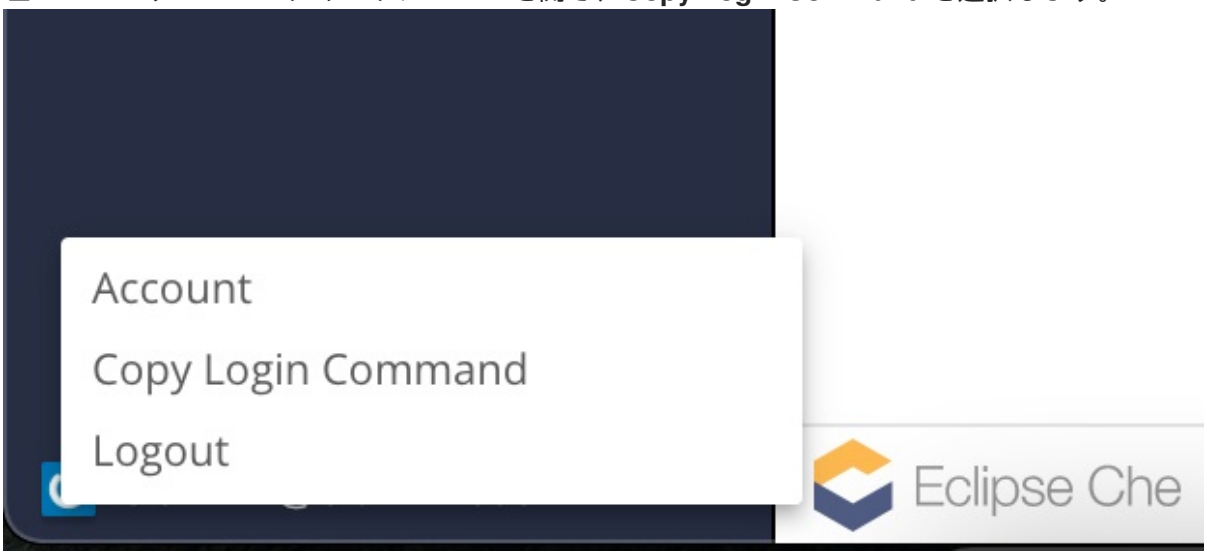
本セクションでは、Red Hat CodeReady Workspaces Dashboard から login コマンドをコピーして、crwctl ツールを使用して CodeReady Workspaces にログインする方法を説明します。

前提条件

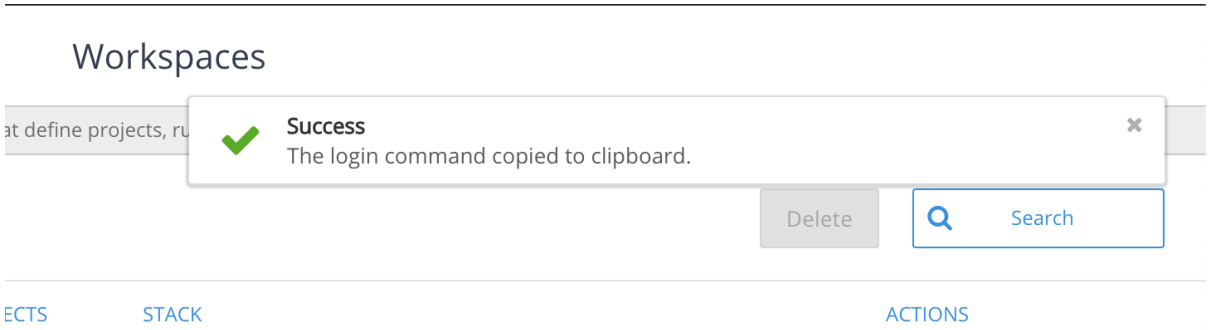
- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- CodeReady Workspaces CLI 管理ツール。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#using-the-crwctl-management-tool_crw を参照してください。
- Red Hat CodeReady Workspaces Dashboard をブラウザーで開いている。

手順

1. 左上のユーザーのコンテキストメニューを開き、**Copy Login Command** を選択します。



2. 通知メッセージを待機します。



3. login コマンドをターミナルに貼り付け、ログインが正常に実行されることを確認します。

```
$ chectl auth:login ...
Successfully logged into <server> as <user>
```

1.1.4. OpenShift 4 CLI を使用した CodeReady Workspaces クラスター URL の検索

本セクションでは、OpenShift 4 コマンドラインインターフェース (CLI) を使用して CodeReady Workspaces クラスター URL を取得する方法を説明します。URL は OpenShift ログまたは **checluster** カスタムリソースから取得できます。

前提条件

- OpenShift で実行している Red Hat CodeReady Workspaces のインスタンス。
- ユーザーの場所は、CodeReady Workspaces インストールプロジェクトになります。

手順

1. **checluster** CR (カスタムリソース) から CodeReady Workspaces クラスター URL を取得するには、以下を実行します。

```
$ oc get checluster --output jsonpath='{.items[0].status.cheURL}'
```

2. または、OpenShift ログから CodeReady Workspaces クラスター URL を取得するには、以下を実行します。

```
$ oc logs --tail=10 `(oc get pods -o name | grep operator)` | \
  grep "available at" | \
  awk -F'available at: ' '{print $2}' | sed 's/"//'
```

1.2. 証明書のブラウザーへのインポート

本セクションでは、ルート認証局を Web ブラウザーにインポートし、自己署名された TLS 証明書と共に CodeReady Workspaces を使用する方法を説明します。

TLS 証明書が信頼されていない場合には、以下のエラーメッセージが表示され、ログインプロセスがブロックされます。**"Your CodeReady Workspaces server may be using a self-signed certificate.To resolve the issue, import the server CA certificate in the browser."** これを防ぐには、CodeReady Workspaces のインストール後に、自己署名された CA 証明書の公開される部分をブラウザーに追加します。

1.2.1. 証明書の Linux または Windows の Google Chrome への追加

手順

1. CodeReady Workspaces がデプロイされている URL に移動します。
2. 証明書を保存します。
   a. アドレスバーの左側にある警告または開いたロックアイコンをクリックします。
   b. **Certificates** をクリックし、**Details** タブに移動します。
   c. ルート認証局である最上位の証明書を選択し、これをエクスポートします。
      - Linux の場合は、**Export** ボタンをクリックします。
      - Windows の場合は、**Save to file** ボタンをクリックします。
3. Google Chrome Settings、**Authorities** タブの順に移動します。
4. 左側のパネルで **Advanced** を選択し、**Privacy and security** に進みます。
5. 画面中央の **Manage certificates** をクリックし、 **Authorities** タブに移動します。
6. **Import** ボタンをクリックし、保存された証明書ファイルを開きます。
7. **Trust this certificate for identifying websites** を選択し、**OK** ボタンをクリックします。
8. CodeReady Workspaces 証明書をブラウザーに追加すると、アドレスバーの URL の横に閉じたロックアイコンが表示され、セキュアな接続であることが示唆されます。

1.2.2. 証明書の macOS の Google Chrome および Safari への追加

手順

1. CodeReady Workspaces がデプロイされている URL に移動します。
2. 証明書を保存します。
   a. アドレスバーの左側にあるロックアイコンをクリックします。
   b. **Certificates** をクリックします。
   c. 使用する証明書を選択し、表示される大きなアイコンをデスクトップにドラッグアンドドロップします。
3. **Keychain Access** アプリケーションを開きます。
4. **System** キーチェーンを選択し、保存された証明書ファイルをこれにドラッグアンドドロップします。
5. インポートした CA をダブルクリックして **Trust** に移動し、**When using this certificate**: **Always Trust** を選択します。
6. 追加した証明書を有効にするためにブラウザーを再起動します。

1.2.3. 証明書の Firefox への追加

手順

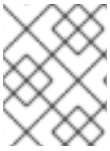
1. CodeReady Workspaces がデプロイされている URL に移動します。
2. 証明書を保存します。
   a. アドレスバーの左側にあるロックアイコンをクリックします。
   b. **Connection not secure** という警告の横にある **>** ボタンをクリックします。
   c. **More information** ボタンをクリックします。
   d. **Security** タブの **View Certificate** ボタンをクリックします。
   e. 2 番目の証明書タブを選択します。証明書の Common Name は **ingress-operator** で開始する必要があります。
   f. **PEM(cert)** リンクをクリックし、証明書を保存します。
3. about:preferences に移動し、**certificates** を検索して **View Certificates** をクリックします。
4. **Authorities** タブに移動し、**Import** ボタンをクリックしてから、保存した証明書ファイルを開きます。
5. **Trust this CA to identify websites** にチェックマークを付け、**OK** をクリックします。
6. 追加した証明書を有効にするために Firefox を再起動します。
7. CodeReady Workspaces 証明書をブラウザーに追加すると、アドレスバーの URL の横に閉じたロックアイコンが表示され、セキュアな接続であることが示唆されます。

1.3. OPENSHIFT DEVELOPER パースペクティブからの CODEREADY WORKSPACES へのアクセス

OpenShift Container Platform は、切り換えに使用するビュースイッチャーを提供します。

- **Administrator** - 従来の管理にフォーカスしたコンソール。
- **Developer** - OpenShift コンポーネントを高次元で抽象化し、開発者がアプリケーションの開発に集中できるようにします。

OpenShift Developer パースペクティブでは、OpenShift 4 Web コンソールで開発者中心のビューを提供します。



注記

OpenShift Developer パースペクティブは、バージョン 4.2 以降で利用可能な OpenShift Container Platform のデフォルトの一部を構成します。

1.3.1. OpenShift Developer パースペクティブの CodeReady Workspaces との統合

このセクションでは、CodeReady Workspaces の OpenShift Developer パースペクティブのサポートについて説明します。

CodeReady Workspaces Operator が OpenShift Container Platform 4.2 以降にデプロイされると、**ConsoleLink** カスタムリソース (CR) が作成されます。これにより、OpenShift Developer パース

ペクティブコンソールを使用して CodeReady Workspaces インストールにアクセスするための **Red Hat Applications** メニューへの対話的なリンクが追加されます。

Red Hat Application メニューにアクセスするには、OpenShift Web コンソールのメイン画面の 3 x 3 のマトリックスアイコンをクリックします。ドロップダウンメニューに表示される CodeReady Workspaces **Console Link** では、新規ワークスペースを作成するか、またはユーザーを既存のワークスペースにリダイレクトします。

注記

CodeReady Workspaces が HTTP リソースと共に使用される場合、OpenShift Container Platform コンソールリンクは作成されません。

From Git オプションを使用して CodeReady Workspaces をインストールする場合、OpenShift Developer パースペクティブのリンクは、CodeReady Workspaces が HTTPS を使用してデプロイされる場合にのみ作成されます。HTTP リソースが使用されている場合、コンソールリンクは作成されません。

1.3.2. CodeReady Workspaces を使用した OpenShift Container Platform で実行されるアプリケーションのコードの編集

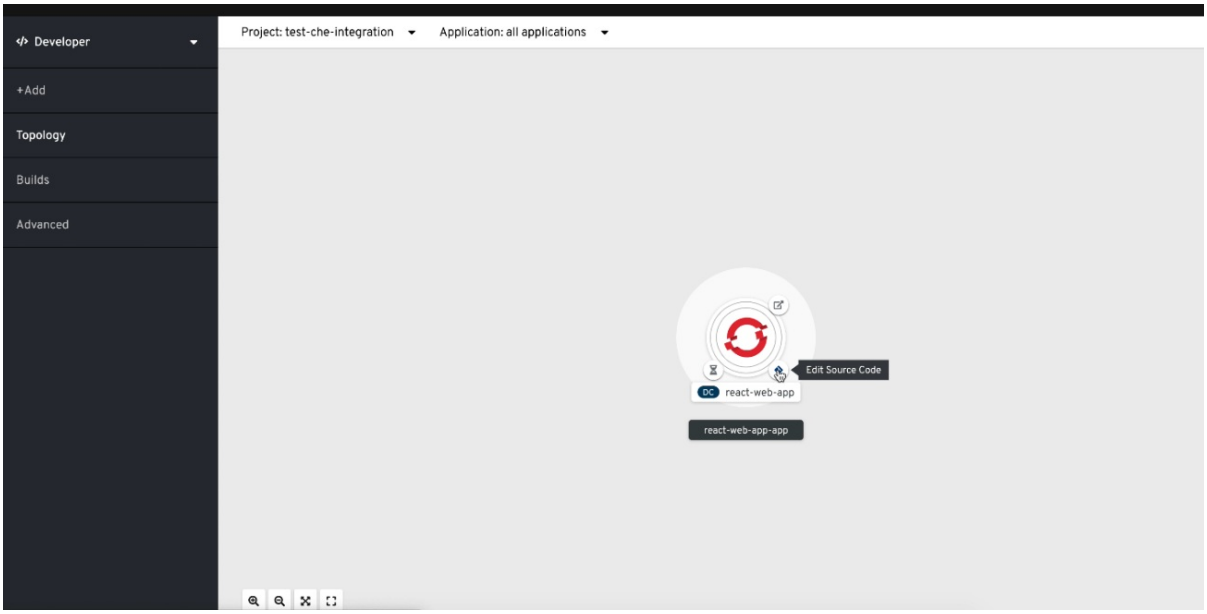
本セクションでは、CodeReady Workspaces を使用して OpenShift で実行しているアプリケーションのソースコードの編集を開始する方法を説明します。

前提条件

- CodeReady Workspaces は同じ OpenShift 4 クラスターにデプロイされている必要があります。

手順

1. **Topology** ビューを開き、すべてのプロジェクトを一覧表示します。
2. **Select an Application** 検索フィールドに **workspace** と入力してすべてのワークスペースを一覧表示します。
3. ワークスペースをクリックして編集します。
   デプロイメントは、円形のボタンで囲まれたグラフィカルな円で表示されます。これらのボタンの 1 つは **Edit Source Code** です。



4. CodeReady Workspaces を使用してアプリケーションのコードを編集するには、**Edit Source Code** ボタンをクリックします。これにより、アプリケーションコンポーネントのクローン作成されたソースコードのあるワークスペースにリダイレクトされます。

1.3.3. Red Hat Applications メニューからの CodeReady Workspaces へのアクセス

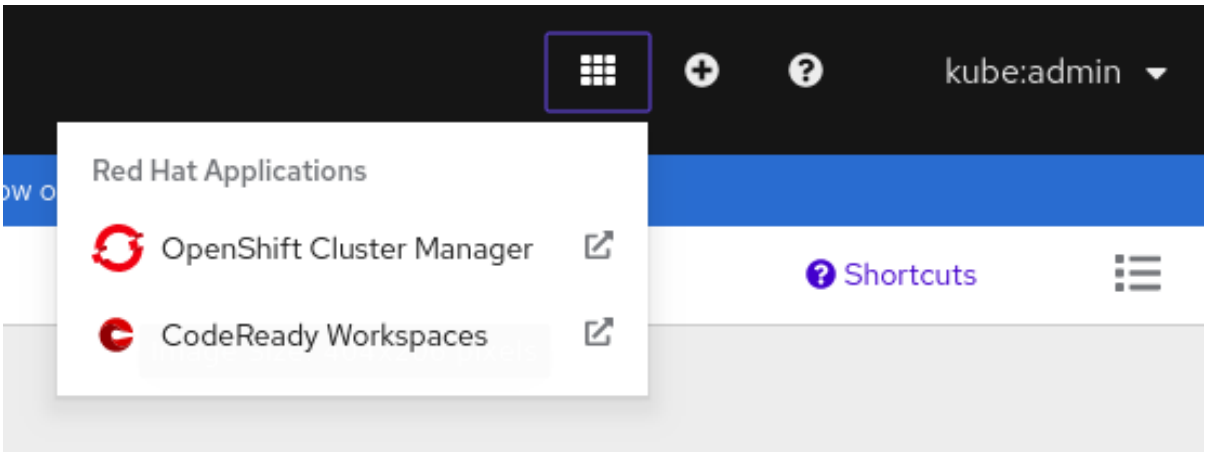
本セクションでは、OpenShift Container Platform の **Red Hat Applications** メニューから CodeReady Workspaces ワークスペースにアクセスする方法を説明します。

前提条件

- CodeReady Workspaces Operator は OpenShift 4 で利用可能である必要があります。

手順

1. メイン画面の右上にある 3 x 3 マトリックスアイコンをクリックして、**Red Hat Applications** メニューを開きます。
ドロップダウンメニューには、利用可能なアプリケーションが表示されます。



2. **CodeReady Workspaces** リンクをクリックして、CodeReady Workspaces Dashboard を開きます。

第2章 CHE-THEIA IDE の基本

本セクションでは、Che-Theia (Red Hat CodeReady Workspaces のネイティブ統合開発環境) の基本ワークフローとコマンドについて説明します。

- 「Che-Theia のカスタムコマンドの定義」
- 「バージョン管理」
- 「Che-Theia のトラブルシューティング」
- 「Che-Theia Webview の単一ホストモードで機能する方法のマルチホストモードとの比較」

2.1. CHE-THEIA のカスタムコマンドの定義

Che-Theia IDE を使用すると、ユーザーはワークスペースを使用する際に利用できる devfile でカスタムコマンドを定義できます。

これは、たとえば以下の場合に役立ちます。

- プロジェクトのビルド、実行、およびデバッグを単純化する。
- リード開発者がチームの要件に基づいてワークスペースをカスタマイズできるようにする。
- 新たなチームメンバーの研修に要する時間を短縮する。

「devfile を使用したワークスペースの設定」 も参照してください。

2.1.1. Che-Theia タスクのタイプ

以下は、devfile の **commands** セクションの例です。

```
commands:

  - name: Package Native App
    actions:
      - type: exec
        component: centos-quarkus-maven
        command: "mvn package -Dnative -Dmaven.test.skip"
        workdir: ${CHE_PROJECTS_ROOT}/quarkus-quickstarts/getting-started

  - name: Start Native App
    actions:
      - type: exec
        component: ubi-minimal
        command: ./getting-started-1.0-SNAPSHOT-runner
        workdir: ${CHE_PROJECTS_ROOT}/quarkus-quickstarts/getting-started/target

  - name: Attach remote debugger
    actions:
    - type: vscode-launch
      referenceContent: |
        {
          "version": "0.2.0",
          "configurations": [
```

```
        {
          "type": "java",
          "request": "attach",
          "name": "Attach to Remote Quarkus App",
          "hostName": "localhost",
          "port": 5005
        }
      ]
    }
```

CodeReady Workspaces コマンド

Package Native App および **Start Native App**
CodeReady Workspaces コマンドは、ワークスペースコンテナーで実行されるタスクを定義するために使用されます。

- **exec** タイプは、CodeReady Workspaces ランナーがコマンド実行に使用されることを示唆します。ユーザーは、コマンドが実行されるコンテナーでコンポーネントを指定できます。
- **command** フィールドには、実行するコマンドラインが含まれます。
- **workdir** は、コマンドを実行する作業ディレクトリーです。
- **component** フィールドは、コマンドを実行するコンテナーを参照します。このフィールドには、コンテナーが定義されているコンポーネント **alias** が含まれます。

VS Code の起動設定

Attach remote debugger
VS Code の起動設定は、通常デバッグ設定を定義するために使用されます。これらの設定をトリガーするには、**F5** を押すか、または **Debug** メニューから **Start Debugging** を選択します。この設定は、デバッグ用に接続するポートやデバッグするアプリケーションのタイプ (Node.js、Java など) などの情報をデバッガーに提供します。

- タイプは **vscode-launch** です。
- これには、VS Code 形式の起動設定が含まれます。
- VS Code の launch 設定の詳細は、Visual Studio のドキュメントページでデバッグのセクションを参照してください。

exec コマンドとしても知られるタイプ **che** のタスクは、**Terminal→Run Task** メニューから、または **My Workspace** パネルでそれらを選択して実行できます。他のタスクは、**Terminal→Run Task** からのみ選択できます。起動設定は、Che-Theia デバッガーで利用できます。

例

- theia タスクの例
- vscode-launch タスクの例

2.1.2. 実行およびデバッグ

Che-Theia は Debug Adapter Protocol をサポートします。このプロトコルは、開発ツールがデバッガーと通信する際の汎用的な方法を定義します。これは、Che-Theia がすべての実装で機能することを意味します。

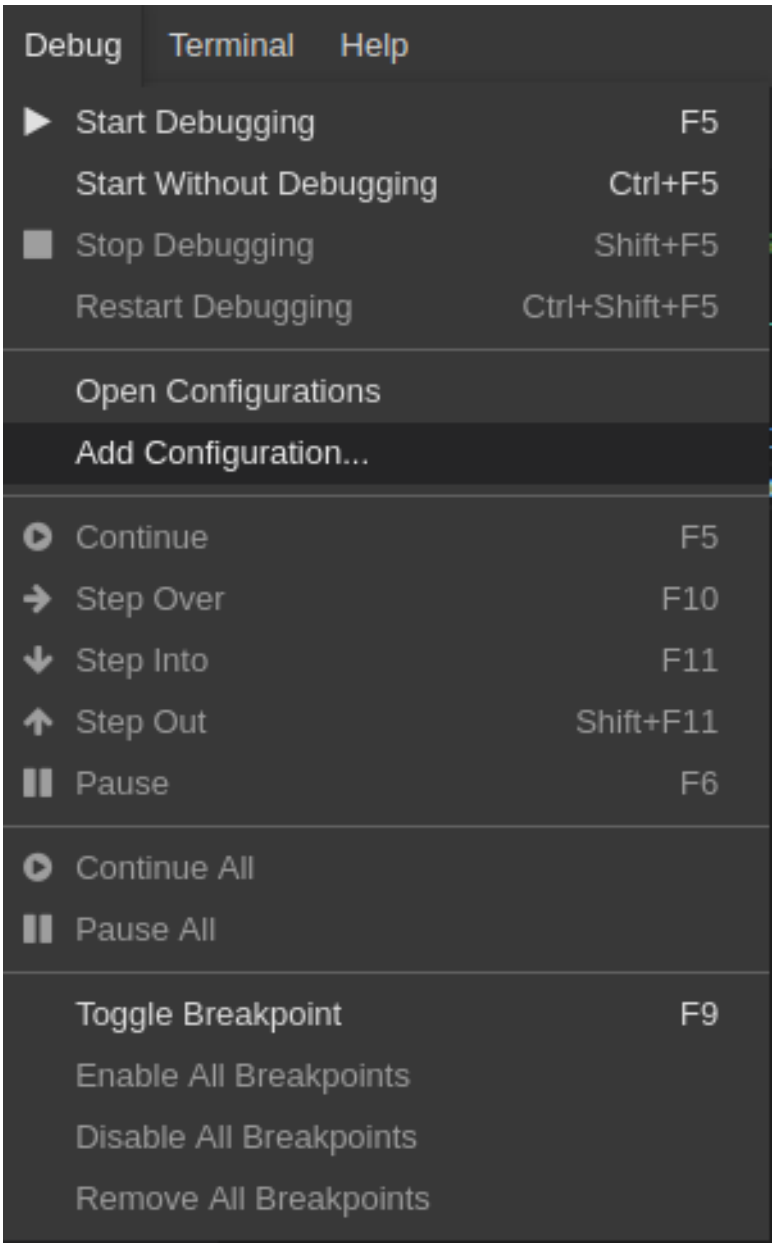
前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。

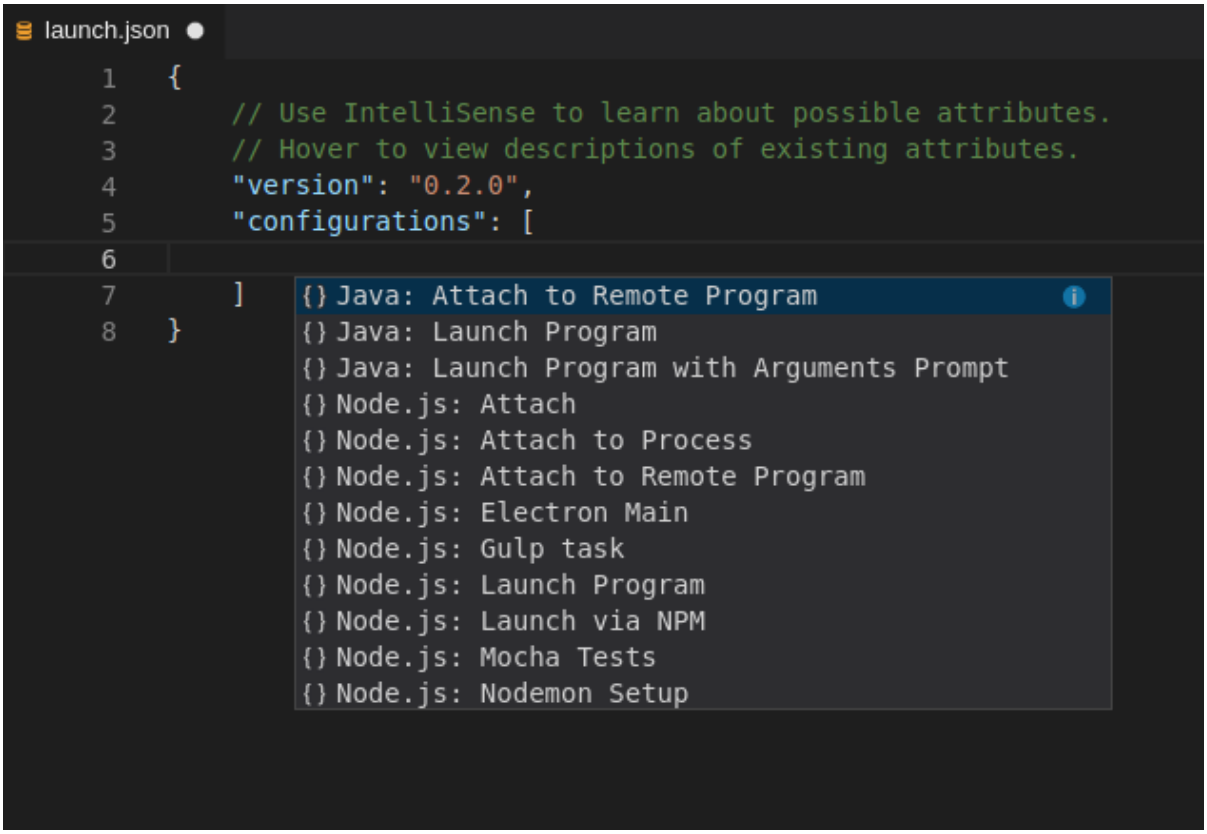
手順

アプリケーションをデバッグするには、以下を実行します。

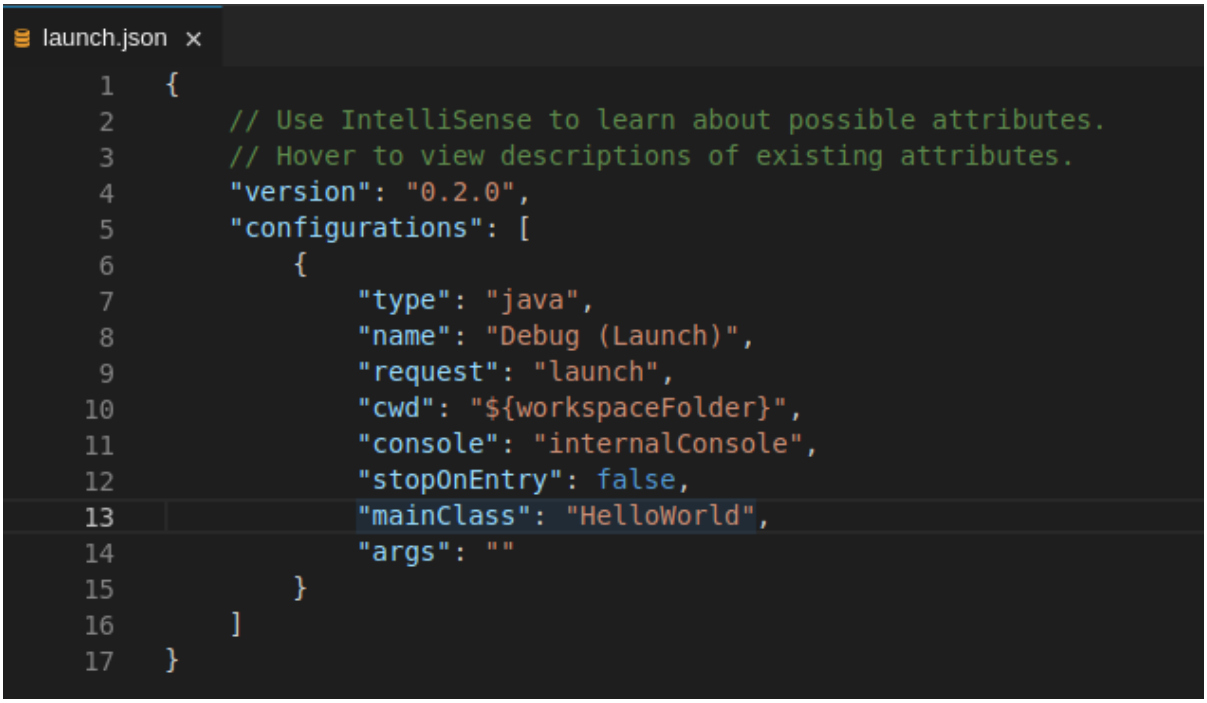
1. **Debug → Add Configuration** の順にクリックし、デバッグまたは起動設定をプロジェクトに追加します。



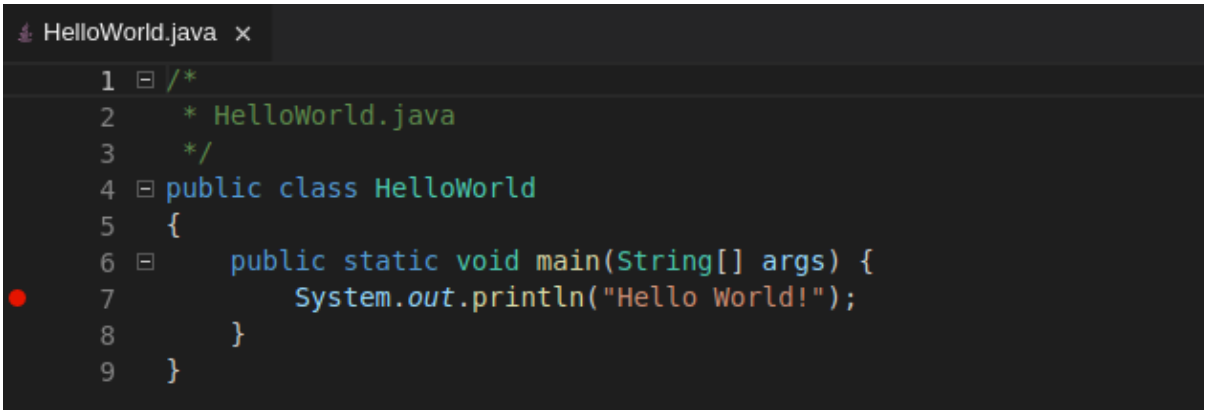
2. ポップアップメニューから、デバッグするアプリケーションの適切な設定を選択します。



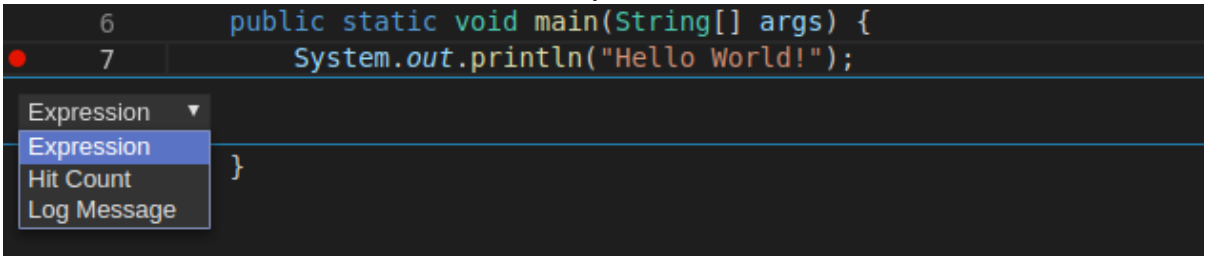
3. 属性を変更または追加して、設定を更新します。



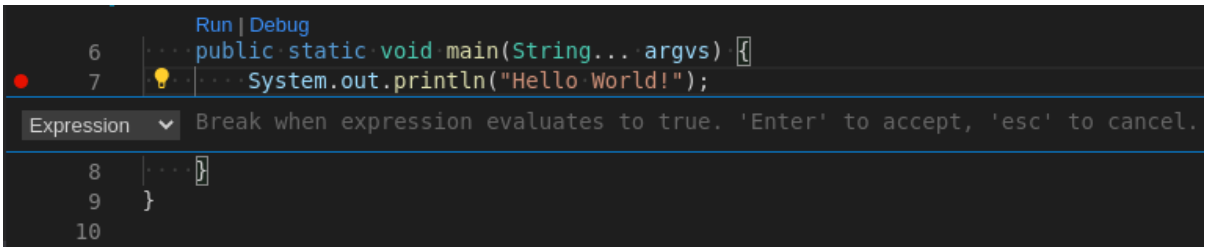
4. ブレークポイントは、エディターのマージンをクリックして切り替えることができます。



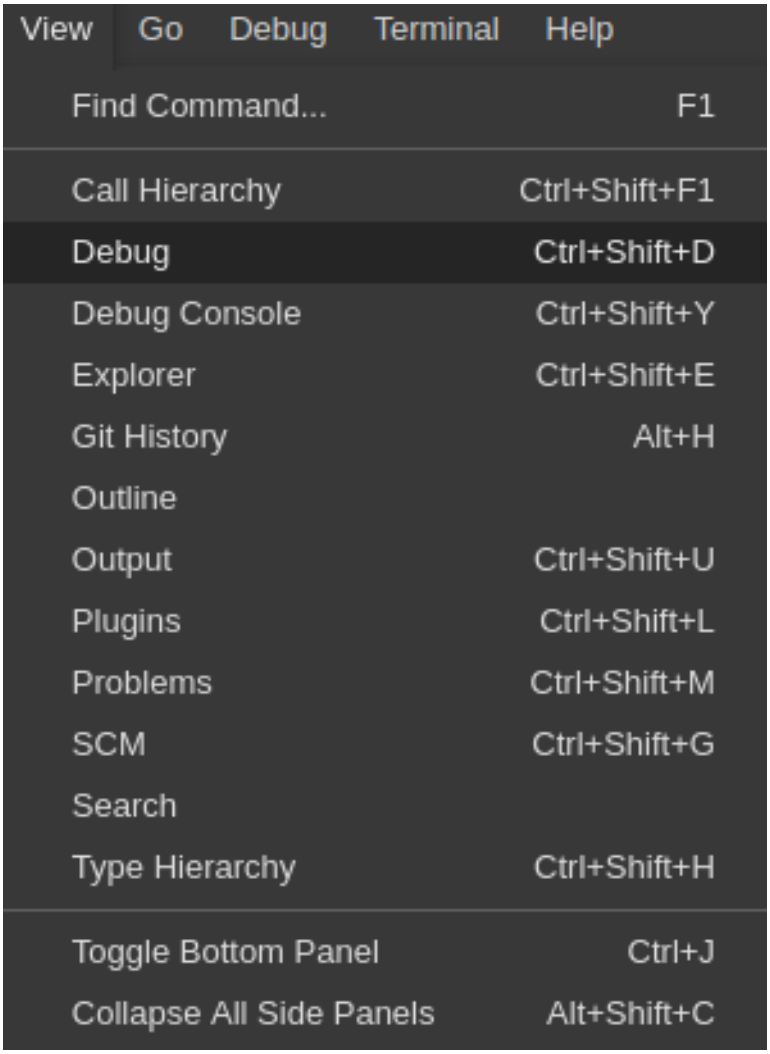
5. コンテキストメニューを開いたら、**Edit Breakpoint** コマンドを使用して条件を追加します。



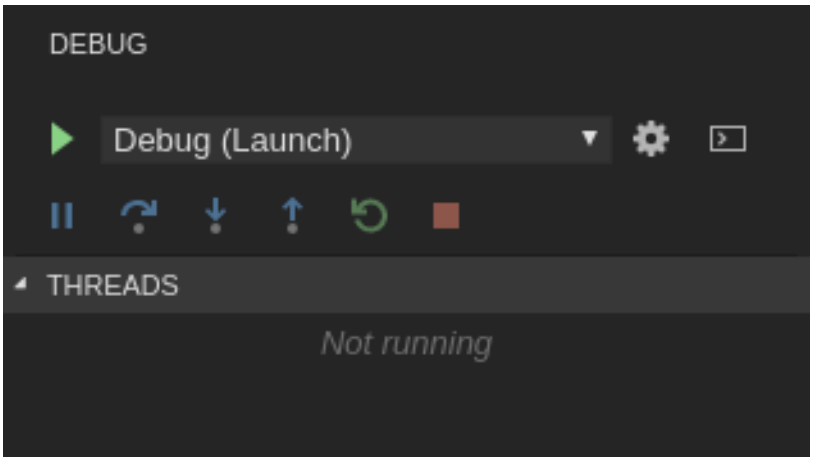
IDE に **Expresion** 入力フィールドが表示されます。



6. デバッグを開始するには、**View→Debug** をクリックします。



7. **Debug** ビューで設定を選択し、**F5** を押してアプリケーションをデバッグします。または、**Ctrl+F5** を押してデバッグせずにアプリケーションを起動します。



2.1.3. タスクおよび起動設定の編集

手順

設定ファイルをカスタマイズするには、以下を実行します。

1. **tasks.json** または **launch.json** 設定ファイルを編集します。
2. 設定ファイルに新規の定義を追加するか、既存の定義を変更します。

注記

変更内容は設定ファイルに保存されます。

3. プラグインで提供されるタスク設定をカスタマイズするには、**Terminal → Configure Tasks** メニューオプションを選択して、設定するタスクを選択します。設定は **tasks.json** ファイルにコピーされ、編集に利用できます。

2.2. バージョン管理

Red Hat CodeReady Workspaces は VS Code SCM モデル をネイティブにサポートします。デフォルトでは、Red Hat CodeReady Workspaces には、ソースコード管理 (SCM) プロバイダーとしてのネイティブのVS Code Git 拡張が含まれます。

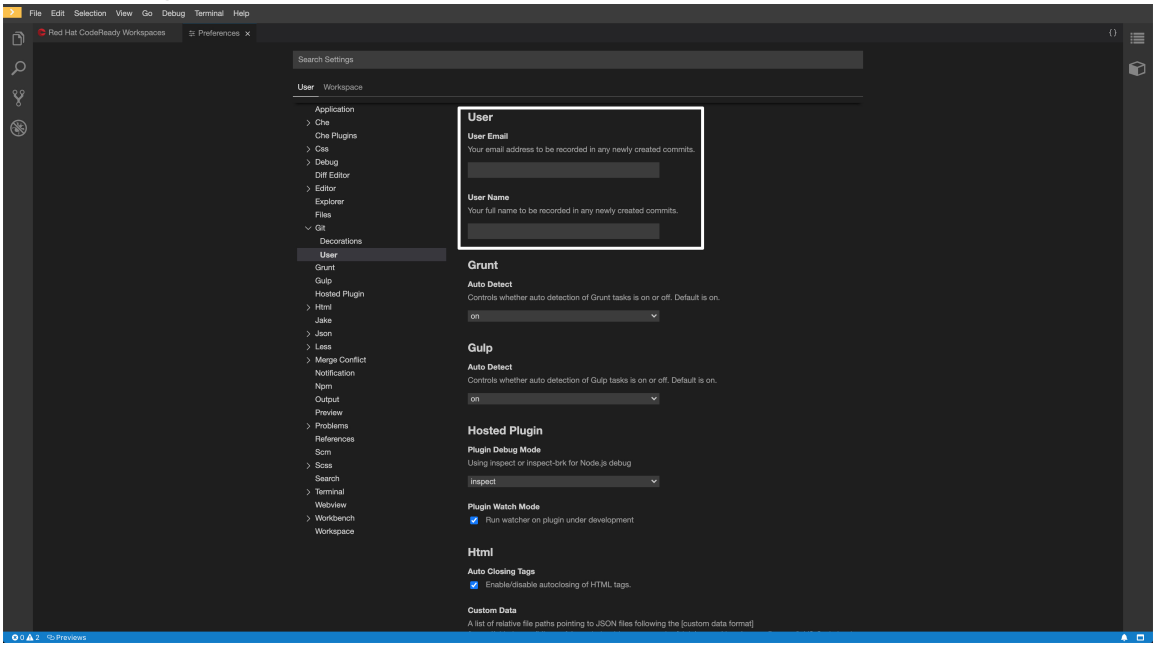
2.2.1. Git 設定の管理: アイデンティティー

Git の使用を開始する前に、まずユーザー名とメールアドレスを設定します。これは、毎回の Git コミットでこの情報を使用するために重要になります。

手順

- CodeReady Workspaces ユーザーインターフェースを使用して Git アイデンティティーを設定するには、以下を実行します。

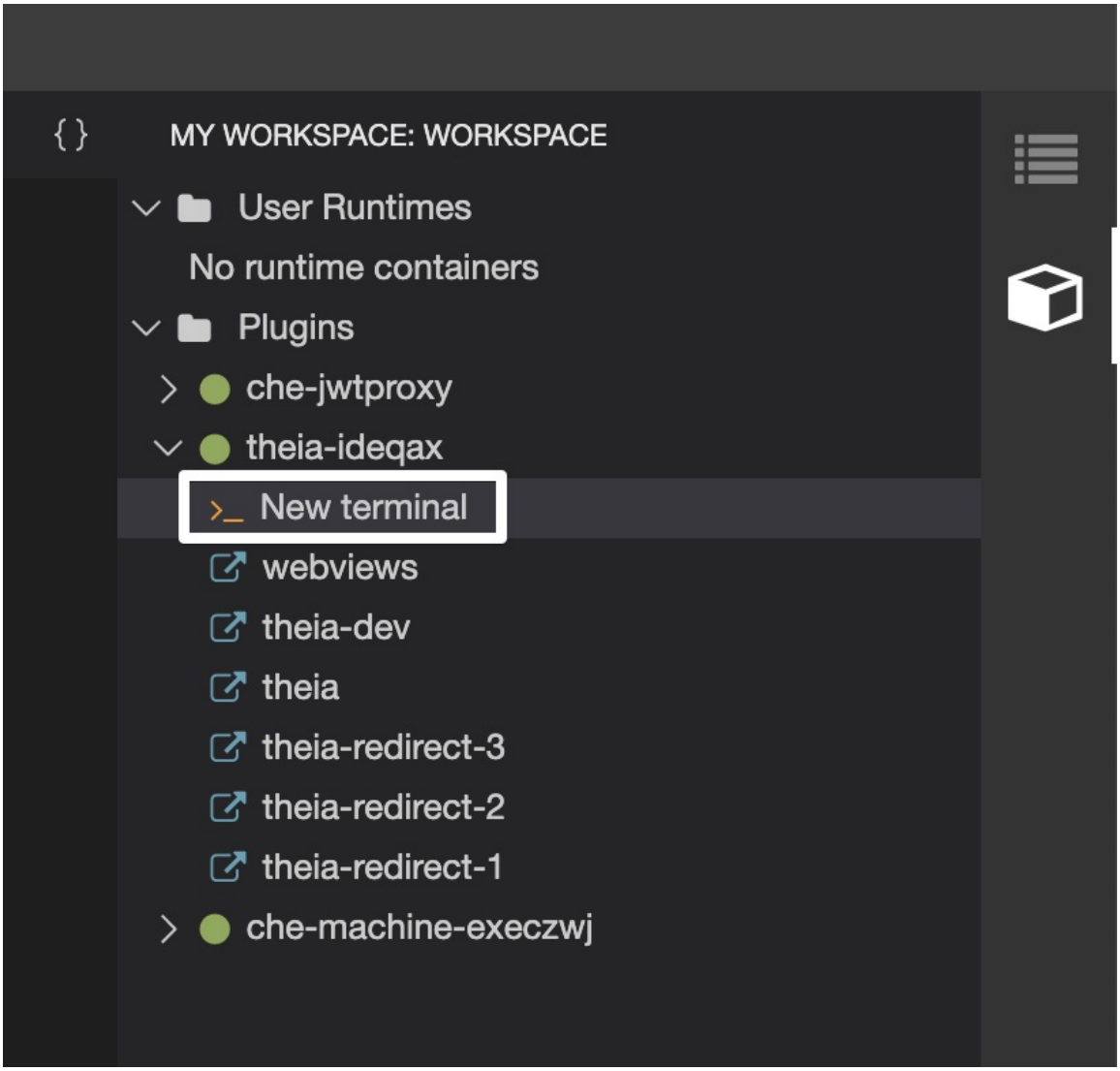
1. **File > Settings > Open Preferences**を開くか、または **Ctrl+,** を押します。



2. 開いたウィンドウで、**Git → User** サブセクションに移動し、User mail と User name の値を入力します。

- コマンドラインを使用して Git アイデンティティーを設定するには、Che-Theia コンテナーのターミナルを開きます。

1. **My Workspace** ビューに移動し、**Plugins > theia-ide... > New terminal**を開きます。



2. 以下のコマンドを実行します。

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Che-Theia は、現在のコンテナーにこの情報を永続的に保存し、ワークスペースの起動時に他のコンテナー用にこの情報を復元します。

2.2.2. HTTPS を使用した Git リポジトリーへのアクセス

手順

HTTPS を使用してリポジトリーのクローンを作成するには、以下を実行します。

1. Visual Studio Code **Git** 拡張で提供される clone コマンドを使用します。

または、ターミナルでネイティブの Git コマンドを使用して、プロジェクトのクローンを作成します。

1. **cd** コマンドを使用して、宛先フォルダーに移動します。
2. **git clone** を使用して、リポジトリーのクローンを作成します。

```
$ git clone <link>
```

Red Hat CodeReady Workspaces は Git の自己署名された TLS 証明書をサポートします。詳細は、、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#deploying-codeready-workspaces-with-support-for-git-repositories-with-self-signed-certificates_crw を参照してください。

2.2.3. 生成される SSH キーペアを使用した Git リポジトリーへのアクセス

2.2.3.1. CodeReady Workspaces コマンドパレットを使用した SSH キーの生成

以下のセクションでは、CodeReady Workspaces コマンドパレットを使用した SSH キーの生成と、これを Git プロバイダーの通信で使用する方法について説明します。この SSH キーは特定の Git プロバイダーのパーミッションを制限するため、ユーザーは使用する Git プロバイダーごとに一意の SSH キーを作成する必要があります。

前提条件

- CodeReady Workspaces の実行中のインスタンスがある。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義される既存のワークスペース。
- 個人の GitHub アカウント または他の Git プロバイダーのアカウントが作成されている必要があります。

手順

デフォルトで、すべての Git プロバイダーについて機能する一般的な SSH キーペアが表示されます。この使用を開始するには、パブリックキーを Git プロバイダーに追加します。

1. 特定の Git プロバイダーについてのみ機能する SSH キーペアを生成します。
   - 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。
   - 検索ボックスに **generate** を入力し、入力後に **Enter** を押して **SSH: generate key pair for particular host** の検索を行います。
   - SSH キーペアのホスト名を指定します (例: **github.com**)。
     SSH キーペアが生成されます。
2. 右下の **View** ボタンをクリックし、エディターからパブリックキーをコピーし、これを Git プロバイダーに追加します。
   コマンドパレットから別のコマンドを使用できるようになりました。SSH セキュア URL を指定して **git リポジトリーのクローン**を作成できるようになりました。

2.2.3.2. 関連付けられたパブリックキーを GitHub のリポジトリーまたはアカウントに追加する

関連付けられたパブリックキーを GitHub のリポジトリーまたはアカウントに追加するには、以下を実行します。

1. github.com に移動します。
2. ウィンドウの右上にあるユーザーアイコンの横のドロップダウン矢印をクリックします。
3. **Settings → SSH and GPG keys**をクリックしてから **New SSH key** ボタンをクリックします。
4. **Title** フィールドにキーのタイトルを入力し、**Key** フィールドに CodeReady Workspaces からコピーしたパブリックキーを貼り付けます。
5. **Add SSH key** ボタンをクリックします。

2.2.3.3. 関連付けられたパブリックキーを GitLab の Git リポジトリーまたは GitLab のアカウントに追加する

関連付けられたパブリックキーを GitLab の Git リポジトリーまたはアカウントに追加するには、以下を実行します。

1. gitlab.com に移動します。
2. ウィンドウの右上にあるユーザーアイコンをクリックします。
3. **Settings → SSH Keys** をクリックします。
4. **Title** フィールドにキーのタイトルを入力し、**Key** フィールドに CodeReady Workspaces からコピーしたパブリックキーを貼り付けます。
5. **Add key** ボタンをクリックします。

2.2.4. GitHub PR プラグインを使用したプルリクエストの管理

GitHub のプルリクエストを管理するには、ワークスペースのプラグインの一覧から選択可能な VS Code GitHub Pull Request プラグインを使用します。

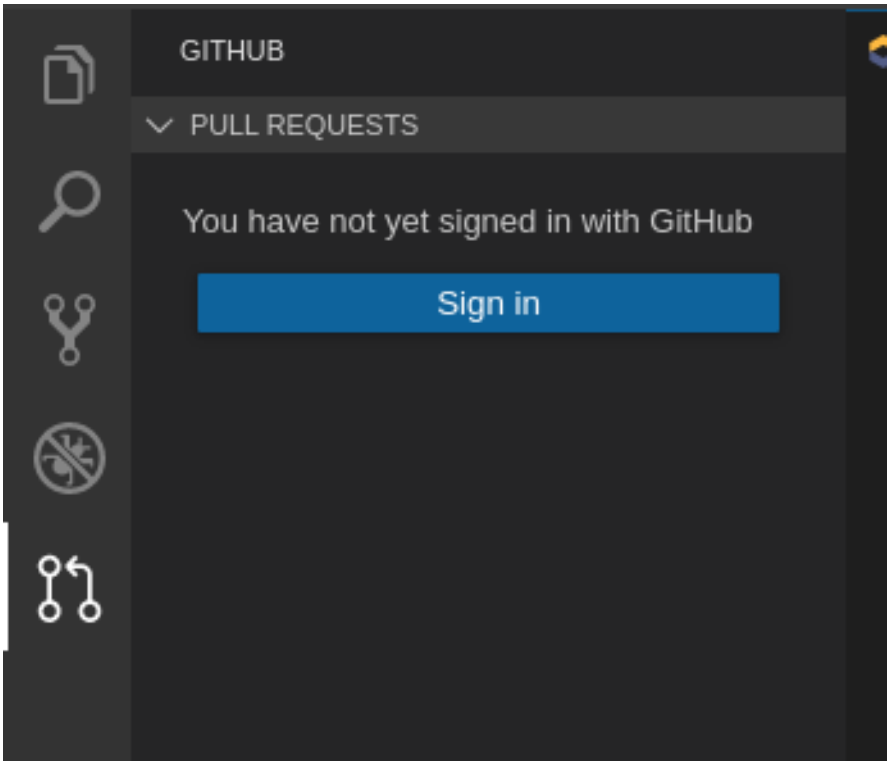
2.2.4.1. GitHub Pull Requests プラグインの使用

前提条件

- GitHub OAuth が設定されている必要があります。xref:[] を参照してください。

手順

- プラグインのビューで **Accounts** メニューまたは **Sign in** ボタンを使用して GitHub にサインインします。



GitHub からサインアウトするには、左下の **Accounts** メニュー、または **GitHub Pull Requests: Sign out from GitHub** コマンドを使用します。

関連資料

- VSCode GitHub Pull Requests プラグインドキュメント

2.3. CHE-THEIA のトラブルシューティング

本セクションでは、Che-Theia IDE で最も頻繁に発生する問題の一部を説明します。

Che-Theiaは、 Plugin runtime crashed unexpectedly, all plugins are not working, please reload the page. Probably there is not enough memory for the plugins.メッセージとともに通知を表示します。

つまり、Che-Theia IDE コンテナーで実行されている Che-Theia プラグインの 1 つに、コンテナーよりも多くのメモリーが必要です。この問題を修正するには、Che-Theia IDE コンテナーのメモリーの量を増やします。

1. CodeReady Workspaces Dashboard に移動します。
2. 問題が発生したワークスペースを選択します。
3. **Devfile** タブに切り替えます。
4. devfile の **components** セクションで、**cheEditor** タイプのコンポーネントを見つけます。
5. 新規のプロパティーを追加します (**memoryLimit: 1024M**)(またはすでにある場合は値を増やします)。
6. 変更を保存し、ワークスペースを再起動します。

2.4. CHE-THEIA WEBVIEW の単一ホストモードで機能する方法のマルチホストモードとの比較

使用される Che デプロイメントストラテジー、単一ホストかマルチホストに応じて、Che-Theia Webview API が機能する仕方に違いがあります。

2.4.1. Webview の概要

Webview プラグイン API を使用すると、Che-Theia 内でビューを作成し、任意の HTML コンテンツを表示できます。内部的には、これは iframe およびサービスワーカーを使用して実装されます。

2.4.2. マルチホストモードの Webview

Red Hat CodeReady Workspaces がマルチホストモードでデプロイされると、Webview コンテンツは別の Origin (オリジン) で提供されます。つまり、これはメインの Che-Theia コンテキストから分離されます。そのため、コントリビュートしたビューは以下にアクセスできません。

- 最上位の DOM
- ローカルストレージ、Cookie などの Che-Theia ステート

2.4.3. 単一ホストモードの Webview

Red Hat CodeReady Workspaces が単一ホストモードでデプロイされると、Webview コンテンツがメインの Che-Theia コンテキストと同じオリジンを使用して読み込まれます。これは、外部コンテンツがブラウザーでメイン Che-Theia にアクセスすることを妨げるものは何もないことを意味します。そのため、Webview をコントリビュートする異なるプラグインによって読み込まれる可能性のあるコンテンツに細心の注意を払う必要があります。

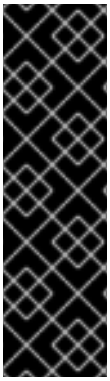
第3章 開発者ワークスペース

Red Hat CodeReady Workspaces は、開発者ワークスペースに、アプリケーションのコード、ビルド、テスト、実行、およびデバッグに必要なすべてのものを提供します。これを許可するために、開発者ワークスペースは 4 つの主なコンポーネントを提供します。

1. プロジェクトのソースコード。
2. Web ベースの統合開発環境 (IDE)
3. 開発者がプロジェクトで作業するために必要なツールの依存関係。
4. アプリケーションランタイム: アプリケーションの実稼働環境での実行に使用される環境のレプリカ。

Pod は CodeReady Workspaces ワークスペースの各コンポーネントを管理します。そのため、CodeReady Workspaces ワークスペースで実行されているすべてのものは、コンテナー内で実行されます。これにより、CodeReady Workspaces ワークスペースに高い移植性を持たせることができます。

組み込みブラウザーベースの IDE は、CodeReady Workspaces ワークスペースで実行しているすべてについてのアクセスポイントです。これにより、CodeReady Workspaces ワークスペースの共有が簡単になります。



重要

デフォルトでは、1 度に 1 つのワークスペースのみを実行できます。ユーザーが実行できる同時ワークスペースの数を増やすには、checluster にパッチを適用します。

```
$ oc patch checluster codeready-workspaces -n openshift-workspaces --type=merge \
-p '{ "spec": { "server": { "customCheProperties": {
"CHE_LIMITS_USER_WORKSPACES_RUN_COUNT": "-1" } } } }'
```

詳細は、ユーザーワークスペースの制限について参照してください。

表3.1 機能および利点

機能	従来の IDE ワークスペース	Red Hat CodeReady Workspaces ワークスペース
設定およびインストールが必要	Yes	No
組み込みツール	部分的。IDE プラグインには設定が必要です。依存関係にはインストールと設定が必要です。例: JDK、Maven、Node	Yesプラグインは、それらの依存関係を提供します。
アプリケーションランタイムが指定される	No。開発者はこれを個別に管理する必要があります。	Yesアプリケーションランタイムはワークスペースで複製されます。
共有可能	No。または容易ではない	Yes開発者ワークスペースは URL で共有可能です。

機能	従来の IDE ワークスペース	Red Hat CodeReady Workspaces ワークスペース
バージョン管理可能	No	Yesdevfile はプロジェクトのソースコードと共に存在します。
どこからでもアクセス可能	No。インストールが必要です。	Yesブラウザーのみが必要です。

CodeReady Workspaces ワークスペースを起動するには、以下のオプションを選択できます。

- 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」
- 「devfile を使用したワークスペースの設定」

Dashboard を使用して CodeReady Workspaces 2.6 を検出します。

- 「コードサンプルからのワークスペースの作成」
- 「プロジェクトのソースコードをインポートしてワークスペースを作成する」

CodeReady Workspaces 2.6 ワークスペースの起動に推奨される方法として devfile を使用します。

- 「devfile を使用してワークスペースを移植可能にする」
- 「OpenShift アプリケーションのワークスペースへのインポート」

CodeReady Workspaces 2.6 ワークスペースと対話に推奨される方法としてブラウザーベースの IDE を使用します。CodeReady Workspaces 2.6 ワークスペースと対話する別の方法については、「ワークスペースへのリモートアクセス」 を参照してください。

3.1. DEVFILE を使用したワークスペースの設定

CodeReady Workspaces ワークスペースを迅速かつ簡単に設定するには、devfile を使用します。devfile の概要とその使用方法については、本セクションの手順を参照してください。

3.1.1. devfile とは

devfile は、開発環境を記述し、定義するファイルです。

- ソースコード。
- ブラウザー IDE ツールやアプリケーションランタイムなどの開発コンポーネント。
- 事前定義コマンドの一覧。
- クローン作成するプロジェクト。

devfile は、CodeReady Workspaces が消費し、複数のコンテナーで構成されるクラウドワークスペースに変換する YAML ファイルです。devfile はリモートまたはローカルに保存できます。以下のような各種の方法で実行できます。

- git リポジトリーのルートフォルダー、または機能ブランチを使用。
- HTTP 経由でアクセス可能な一般にアクセスできる Web サーバーを使用。

- ローカルでファイルとして使用、または**crwctl**を使用してデプロイ。
- devfile のコレクション (devfile レジストリー として知られる) を使用。

ワークスペースの作成時に、CodeReady Workspaces はその定義を使用してすべてを開始し、必要なツールおよびアプリケーションランタイムのすべてのコンテナーを実行します。また、CodeReady Workspaces はファイルシステムボリュームをマウントして、ソースコードをワークスペースで利用できるようにします。

devfile は、プロジェクトのソースコードでバージョン管理できます。ワークスペースで古いメンテナンスブランチを修正する必要がある場合、プロジェクトの devfile は、ワークスペースの定義を古いブランチでの機能を開始するために必要な各種ツールと依存関係と共に提供します。これを使用してワークスペースをオンデマンドでインスタンス化します。

CodeReady Workspaces は、ワークスペースで使用するツールと共に devfile の最新の状態を維持します。

- パス、git の場所、ブランチなどのプロジェクトの要素。
- ビルド、実行、テスト、デバッグなどの日次タスクを実行するためのコマンド。
- アプリケーションの実行に必要なコンテナーイメージを含むランタイム環境。
- 開発者がワークスペースで使用するツール、IDE 機能、ヘルパーが含まれる Che-Theia プラグイン (例: Git、Java サポート、SonarLint、およびプルリクエスト)。

3.1.2. Git リポジトリーのデフォルトブランチでのワークスペースの作成

Git ソースリポジトリーに保存される devfile を参照して、CodeReady Workspaces ワークスペースを作成できます。次に、CodeReady Workspaces インスタンスは検出された devfile.yaml ファイルを使用して factory URL (**/f?url=**) API を使用してワークスペースをビルドします。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- **devfile.yaml** または **.devfile.yaml** ファイルは、HTTPS で利用可能な Git リポジトリーのルートフォルダーにあります。devfile の作成および使用についての詳細は、「devfile を使用してワークスペースを移植可能にする」 を参照してください。

手順

URL **https://codeready-<openshift_deployment_name>.<domain_name>/f?url=https://<GitRepository>** を開いてワークスペースを実行します。

例

```
https://che.openshift.io/f?url=https://github.com/eclipse/che
```

3.1.3. Git リポジトリーの機能ブランチでのワークスペースの作成

CodeReady Workspaces ワークスペースは、ユーザーの選択する機能ブランチの Git ソースリポジトリーに保存される devfile を参照して作成できます。次に CodeReady Workspaces インスタンスは検出された devfile を使用してワークスペースをビルドします。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- **devfile.yaml** または **.devfile.yaml** ファイルは、HTTPS 経由でアクセス可能なユーザーが選択する特定のブランチの、Git リポジトリーのルートフォルダーにあります。devfile の作成および使用についての詳細は、「devfile を使用してワークスペースを移植可能にする」 を参照してください。

手順

URL **https://codeready-<openshift_deployment_name>.<domain_name>/f?url=<GitHubBranch>** を開いてワークスペースを実行します。

例

以下の URL 形式を使用して、che.openshift.io でホストされる実験的な quarkus-quickstarts ブランチを開きます。

```
https://che.openshift.io/f?url=https://github.com/maxandersen/quarkus-quickstarts/tree/che
```

3.1.4. HTTP を使用した一般にアクセス可能なスタンドアロン devfile でのワークスペースの作成

ワークスペースは、devfile、devfile の未加工コンテンツを参照する URL を使用して作成できます。次に CodeReady Workspaces インスタンスは検出された devfile を使用してワークスペースをビルドします。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- 一般にアクセス可能なスタンドアロン **devfile.yaml** ファイル。devfile の作成および使用についての詳細は、「devfile を使用してワークスペースを移植可能にする」 を参照してください。

手順

1. URL **https://codeready-<openshift_deployment_name>.<domain_name>/f?url=https://<yourhosturl>/devfile.yaml** を開いてワークスペースを実行します。

例

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c20743cde712bbc822521463359a60d1f7a9/devfile.yaml
```

3.1.5. factory パラメーターを使用した devfile 値の上書き

リモート devfile の以下のセクションにある値は、特別に作成される追加の factory パラメーターを使用して上書きできます。

- **apiVersion**
- **metadata**
- **projects**
- **attributes**

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- 一般にアクセス可能なスタンドアロン **devfile.yaml** ファイル。devfile の作成および使用についての詳細は、「devfile を使用してワークスペースを移植可能にする」 を参照してください。

手順

1. URL **https://codeready-<openshift_deployment_name>.<domain_name>/f?url=https://<hostURL>/devfile.yaml&override.<parameter.path>=<value>** に移動してワークスペースを開きます。

generateName プロパティーを上書きする例

以下の初期 devfile について考えてみましょう。

```
---
apiVersion: 1.0.0
metadata:
  generateName: golang-
projects:
...
```

generateName 値を追加または上書きするには、以下の factory URL を使用します。

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c20743cde712bbc822521463359a60d1f7a9/devfile.yaml&override.metadata.generateName=myprefix
```

作成されるワークスペースには、以下の devfile モデルがあります。

```
---
apiVersion: 1.0.0
metadata:
  generateName: myprefix
projects:
...
```

プロジェクトソースブランチプロパティーの上書き例

以下の初期 devfile について考えてみましょう。

```
---
apiVersion: 1.0.0
metadata:
  generateName: java-mysql-
projects:
  - name: web-java-spring-petclinic
    source:
      type: git
      location: "https://github.com/spring-projects/spring-petclinic.git"
...
```

ソースの **branch** の値を追加または上書きするには、以下の factory URL を使用します。

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c2074
3cde712bbc822521463359a60d1f7a9/devfile.yaml&override.projects.web-java-spring-
petclinic.source.branch=1.0.x
```

作成されるワークスペースには、以下の devfile モデルがあります。

```
apiVersion: 1.0.0
metadata:
  generateName: java-mysql-
projects:
  - name: web-java-spring-petclinic
    source:
      type: git
      location: "https://github.com/spring-projects/spring-petclinic.git"
      branch: 1.0.x
...
```

属性値の上書きまたは作成例

以下の初期 devfile について考えてみましょう。

```
---
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
   persistVolumes: false
projects:
...
```

persistVolumes 属性値を追加または上書きするには、以下の factory URL を使用します。

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c2074
3cde712bbc822521463359a60d1f7a9/devfile.yaml&override.attributes.persistVolumes=true
```

作成されるワークスペースには、以下の devfile モデルがあります。

```
---
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
   persistVolumes: true
projects:
...
```

属性を上書きする場合、**attributes** キーワードをベースとするものはすべて属性名として解釈されるため、ユーザーはドットで区切られた名前を使用することができます。

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c20743cde712bbc822521463359a60d1f7a9/devfile.yaml&override.attributes.dot.name.format.attribute=true
```

作成されるワークスペースには、以下の devfile モデルがあります。

```
---
apiVersion: 1.0.0
metadata:
  generateName: golang-
attributes:
   dot.name.format.attribute: true
projects:
...
```

3.1.6. crwctl およびローカル devfile を使用したワークスペースの作成

crwctl ツールにローカルに保存された devfile を参照させて、CodeReady Workspaces ワークスペースを作成できます。次に CodeReady Workspaces インスタンスは検出された devfile を使用してワークスペースをビルドします。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- CodeReady Workspaces CLI 管理ツール。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#using-the-crwctl-management-tool_crw を参照してください。
- devfile は、現在の作業ディレクトリー内のローカルファイルシステムで利用できます。devfile の作成および使用についての詳細は、「devfile を使用してワークスペースを移植可能にする」を参照してください。
- Red Hat CodeReady Workspaces にログインしている必要があります。「crwctl を使用した CodeReady Workspace へのログイン」を参照してください。

  例

  devfile.yaml ファイルを GitHub リポジトリーから現行の作業ディレクトリーにダウンロードします。

手順

1. 以下のように、**crwctl** ツールに **workspace:create** パラメーターを指定して devfile からワークスペースを実行します。

```
$ crwctl workspace:create --name=<WORKSPACE_NAME> \ (1)
--devfile=devfile.yaml --start \
-n openshift-workspaces
```

(1) 作成するワークスペース名

注記

--devfile フラグを省略すると、crwctl は現行ディレクトリーで、ワークスペースの作成に使用する **devfile.yaml** または **devfile.yml** ファイルを検索します。

3.1.7. ユーザーによるワークスペースのデプロイメントラベルおよびアノテーションの定義を許可する

本セクションでは、factory パラメーターを使用してワークスペースのデプロイメントのラベルとアノテーションをカスタマイズする方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- 一般にアクセス可能なスタンドアロン **devfile.yaml** ファイル。devfile の作成および使用についての詳細は、「devfile を使用してワークスペースを移植可能にする」 を参照してください。

手順

1. URL **https://codeready-<openshift_deployment_name>.<domain_name>/f?url=https://<hostURL>/devfile.yaml&workspaceDeploymentLabels=<url_encoded_comma_separated_key_values>&workspaceDeploymentAnnotations=<url_encoded_comma_separated_key_values override>** に移動してワークスペースを開きます。

deployment labelsを上書きする例

追加する以下のラベルについて検討します。

```
ike.target=preference-v1
ike.session=test
```

ラベルを追加または上書きするには、以下のファクトリー URL を使用します。

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c20743cde712bbc822521463359a60d1f7a9/devfile.yaml&workspaceDeploymentLabels=ike.target%3Dpreference-v1%2Cike.session%3Dtest
```

作成されるワークスペースには、以下のデプロイメントラベルが含まれます。

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2020-10-27T14:03:26Z"
  generation: 1
  labels:
    che.component.name: che-docs-dev
    che.original_name: che-docs-dev
    che.workspace_id: workspacegln2g1shejjufpkd
    ike.session: test
    ike.target: preference-v1
  name: workspacegln2g1shejjufpkd.che-docs-dev
  namespace: opentlc-mgr-che
  resourceVersion: "107516"
spec:
...
```

deployment annotationsを上書きする例

追加する以下のアノテーションについて検討します。

```
ike.A1=preference-v1
ike.A=test
```

アノテーションを追加または上書きするには、以下のファクトリー URL を使用します。

```
https://che.openshift.io/f?
url=https://gist.githubusercontent.com/themr0c/ef8e59a162748a8be07e900b6401e6a8/raw/8802c2074
3cde712bbc822521463359a60d1f7a9/devfile.yaml&workspaceDeploymentAnnotations=ike.A1%3Dpref
erence-v1%2Cike.A%3Dtest
```

生成されるワークスペースには、以下のデプロイメントアノテーションが含まれます。

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
    ike.A: test
    ike.A1: preference-v1
  creationTimestamp: "2020-10-28T09:58:52Z"
  generation: 1
  labels:
    che.component.name: che-docs-dev
    che.original_name: che-docs-dev
    che.workspace_id: workspacexrtf710v64rl5ouz
  name: workspacexrtf710v64rl5ouz.che-docs-dev
```

```
  namespace: opentlc-mgr-che
  resourceVersion: "213191"
...
```

関連資料

- 「devfile を使用してワークスペースを移植可能にする」

3.2. DEVFILE を使用してワークスペースを移植可能にする

設定した CodeReady Workspaces ワークスペースを転送するには、ワークスペースの devfile を作成し、エクスポートして、別のホストで devfile を読み込み、ワークスペースの新規インスタンスを初期化します。この devfile を作成する方法についての詳細は、以下を参照してください。

3.2.1. 最小の devfile

以下は、devfile で必要なコンテンツの最小コンテンツです。

- apiVersion
- メタデータ名

```
apiVersion: 1.0.0
metadata:
  name: crw-in-crw-out
```

devfile の完全なサンプルについては、CodeReady Workspaces devfile.yaml の Red Hat CodeReady Workspaces を参照してください。

注記

パラメーター **generateName** または **name** を使用するかどうかは選択できますが、ユーザーはこれらのパラメーターのいずれかを選択し、定義する必要があります。両方の属性を指定すると、**generateName** は無視されます。「ワークスペース名の生成」 を参照してください。

```
metadata:
  generatedName:
```

または

```
metadata:
  name:
```

3.2.2. ワークスペース名の生成

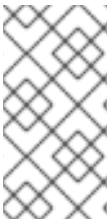
自動生成されるワークスペース名のプレフィックスを指定するには、devfile で **generateName** パラメーターを設定します。

```
apiVersion: 1.0.0
metadata:
  generateName: crw-
```

ワークスペース名は **<generateName>YYYYY** 形式（例: **che-2y7kp**）になります。**Y** はランダムな **[a-z0-9]** 文字です。

ワークスペースの作成時に、以下の命名規則が適用されます。

- **name** が定義される際に、これはワークスペース名 **<name>** として使用されます。
- **generateName** のみが定義されている場合、これは生成される名前 **<generateName>YYYYY** のベースとして使用されます。



注記

factory を使用して作成されたワークスペースの場合、**name** または **generateName** を定義すると同じ結果になります。定義された値は、名前のプレフィックス **<name>YYYYY** または **<generateName>YYYYY** として使用されます。**generateName** と **name** の両方が定義されている場合は、**generateName** が優先されます。

3.2.3. プロジェクトの devfile の作成

本セクションでは、プロジェクト用に最小限の devfile を作成し、devfile に複数のプロジェクトを追加する方法を説明します。

3.2.3.1. 最小 devfile の作成

ワークスペースを実行するのに十分な最小の devfile は、以下の部分素で構成されます。

- 仕様バージョン
- 名前

プロジェクトのない最小 devfile の例

```
apiVersion: 1.0.0
metadata:
  name: minimal-workspace
```

追加の設定がない場合、デフォルトのエディターが含まれるワークスペースが CodeReady Workspaces サーバーで設定されるデフォルトのプラグインと共に起動します。Che-Theia は、**CodeReady Workspaces Machine Exec** プラグインと共にデフォルトのエディターとして設定されます。factory を使用して Git リポジトリー内でワークスペースを起動すると、指定のリポジトリーとブランチからのプロジェクトがデフォルトで作成されます。プロジェクト名は、リポジトリー名と一致します。

ワークスペースの機能を追加するために、以下の部分を追加します。

- コンポーネントの一覧: 開発コンポーネントおよびユーザーランタイム
- プロジェクトの一覧: ソースコードリポジトリー
- コマンドの一覧: 開発環境の実行、ランタイム環境の起動など、ワークスペースコンポーネントを管理するためのアクション

プロジェクトを含む最小 devfile の例

```
apiVersion: 1.0.0
metadata:
```

```
  name: petclinic-dev-environment
projects:
  - name: petclinic
    source:
      type: git
      location: 'https://github.com/spring-projects/spring-petclinic.git'
components:
  - type: chePlugin
    id: redhat/java/latest
```

3.2.3.2. devfile の複数プロジェクトの指定

単一の devfile は、必要な宛先に対してクローン作成される複数のプロジェクトを定義できます。これらのプロジェクトは、ワークスペースの起動後にユーザーのワークスペース内に作成されます。

各プロジェクトについて、以下を指定します。

- ソースリポジトリーのタイプ: これには .git または .zip を指定できます。詳細は、Devfile リファレンスのセクションを参照してください。
- ソースリポジトリーの場所: Git リポジトリーまたは ZIP アーカイブへの URL。
- オプションで、プロジェクトのクローンが作成されるディレクトリー。指定がない場合は、デフォルトのディレクトリーが使用されます。これは、プロジェクト名またはプロジェクトの Git リポジトリーに一致するディレクトリーです。

2 つのプロジェクトを含む devfile の例

以下の例では、プロジェクトの **frontend** および **backend** はユーザーのプロジェクトのサンプルとして機能します。各プロジェクトは別個のリポジトリーに置かれます。

- **backend** プロジェクトには、CodeReady Workspaces ランタイムで暗黙的に定義される、ソースルート下の **src/github.com/<github-organization>/<backend>/** ディレクトリーにクローン作成する特定の要件があります。
- **frontend** プロジェクトは、ソースルート下の **<frontend/>** ディレクトリーにクローン作成されます。

```
apiVersion: 1.0.0
metadata:
  name: example-devfile
projects:
- name: <frontend>
  source:
    type: git
    location: https://github.com/<github-organization>/<frontend>.git
- name: <backend>
  clonePath: src/github.com/<github-organization>/<backend>
  source:
    type: git
    location: https://github.com/<github-organization>/<backend>.git
```

関連資料

すべての devfile コンポーネントの割り当てと使用可能な値についての詳細は、以下を参照してください。

- 仕様リポジトリー
- 詳細の json-schema ドキュメント

以下のサンプル devfile は、適切な参照例です。

- ユーザーインターフェースでデフォルトで使用される Red Hat CodeReady Workspaces ワークスペースのサンプル devfile。
- Red Hat Developer プログラムの Red Hat CodeReady Workspaces ワークスペースのサンプル devfile。

3.2.4. devfile リファレンス

本セクションでは、devfile の参照および devfile を構成するさまざまな要素を使用する方法について説明します。

3.2.4.1. devfile へのプロジェクトの追加

通常、devfile には 1 つ以上のプロジェクトが含まれます。これらのプロジェクトを開発するためのワークスペースが作成されます。プロジェクトは、devfile の **projects** セクションに追加されます。

単一 devfile の各プロジェクトには、以下が必要です。

- 一意な名前
- 指定されるソース

プロジェクトソースは、**type** および **location** の 2 つの必須の値で構成されます。

type
: プロジェクトソースプロバイダーの種類。

location
: プロジェクトソースの URL。

CodeReady Workspaces は以下のプロジェクトタイプをサポートします。

git
: Git のソースを含むプロジェクト。この場所はクローンのリンクを参照します。

github
: **git** と同じですが、GitHub でホストされるプロジェクト専用です。GitHub 固有の機能を使用しないプロジェクトには **git** を使用します。

zip
: ZIP アーカイブのソースを含むプロジェクト。場所は ZIP ファイルを参照します。

3.2.4.1.1. プロジェクトソースタイプ: git

```
source:
  type: git
  location: https://github.com/eclipse/che.git
  startPoint: master
  tag: 7.2.0
```

```
    commitId: 36fe587
    branch: 7.20.x
    sparseCheckoutDir: core  2
```

1 **startPoint**: **tag**、**commitId**、および **branch** の一般的な値。**startPoint**、**tag**、**commitId**、および **branch** パラメーターは相互に排他的です。複数の値を指定すると、**startPoint**、**tag**、**commitId**、**branch** の順に適用されます。

2 **sparseCheckoutDir**: sparse checkout Git 機能のテンプレート。これは、プロジェクトの一部 (通常は単一ディレクトリー) のみが必要な場合に役立ちます。

例3.1 sparseCheckoutDir パラメーター設定

- ルートの **my-module** ディレクトリー (およびそのコンテンツ) のみを作成するには **/my-module/** に設定されます。
- 先頭のスラッシュ (**my-module/**) を省略して、プロジェクトに存在するすべての **my-module** ディレクトリーを作成します。たとえば、**/addons/my-module/**を含みます。最後のスラッシュは、指定される名前のディレクトリー (それらのコンテンツを含む) のみが作成されることを示します。
- ワイルドカードを使用して、複数のディレクトリー名を指定します。たとえば、**module-*** を設定すると、**module-** で始まる指定のプロジェクトのすべてのディレクトリーがチェックアウトされます。

詳細は、Git ドキュメントで sparse checkout を参照してください。

3.2.4.1.2. プロジェクトソースタイプ: zip

```
source:
    type: zip
    location: http://host.net/path/project-src.zip
```

3.2.4.1.3. プロジェクトのクローンパスパラメーター: clonePath

clonePath パラメーターは、プロジェクトのクローンを作成するパスを指定します。パスは **/projects/** ディレクトリーに相対的であり、**/projects/** ディレクトリーから外すことはできません。デフォルト値はプロジェクト名です。

プロジェクトが含まれる devfile の例

```
apiVersion: 1.0.0
metadata:
  name: my-project-dev
projects:
  - name: my-project-resourse
    clonePath: resources/my-project
    source:
      type: zip
      location: http://host.net/path/project-res.zip
  - name: my-project
    source:
```

```
    type: git
    location: https://github.com/my-org/project.git
    branch: develop
```

3.2.4.2. devfile へのコンポーネントの追加

単一の devfile のコンポーネントにはそれぞれ一意の名前を指定する必要があります。

3.2.4.2.1. コンポーネントタイプ: cheEditor

id を定義してワークスペースで使用するエディターを記述します。devfile には、**cheEditor** タイプの 1 つのコンポーネントのみを含めることができます。

```
components:
  - alias: theia-editor
    type: cheEditor
    id: eclipse/che-theia/next
```

cheEditor がない場合、デフォルトのエディターがそのデフォルトのプラグインと共に提供されます。デフォルトのプラグインは、デフォルトと同じ **id** で明示的に定義されるエディターについて指定されます (バージョンが異なる場合でも同様です)。Che-Theia は、**CodeReady Workspaces Machine Exec** プラグインと共にデフォルトのエディターとして設定されます。

ワークスペースでエディターが必要にならないように指定するには、devfile 属性の **editorFree:true** 属性 を使用します。

3.2.4.2.2. コンポーネントタイプ: chePlugin

id を定義してワークスペース内のプラグインを記述します。複数の **chePlugin** コンポーネントを含めることができます。

```
components:
  - alias: exec-plugin
    type: chePlugin
    id: eclipse/che-machine-exec-plugin/0.0.1
```

上記の両方のタイプで、CodeReady Workspaces プラグインレジストリーからの、スラッシュで区切られたパブリッシャー、プラグインの名前およびバージョンである ID を使用します。

利用可能な Eclipse Che プラグインおよびレジストリーの詳細は、Eclipse Che プラグインレジストリーの GitHub リポジトリーを参照してください。

3.2.4.2.3. 代替コンポーネントレジストリーの指定

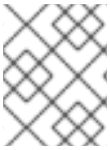
cheEditor および **chePlugin** コンポーネントタイプの代替レジストリーを指定するには、**registryUrl** パラメーターを使用します。

```
components:
  - alias: exec-plugin
    type: chePlugin
    registryUrl: https://my-customregistry.com
    id: eclipse/che-machine-exec-plugin/0.0.1
```

3.2.4.2.4. 記述子にリンクしてコンポーネントを指定する

エディターやプラグイン **id** (およびオプションとして代替レジストリー) を使用する代わりに、**cheEditor** または **chePlugin** を指定する代替方法では、**reference** フィールドを使用してコンポーネント記述子 (通常は **meta.yaml** という名前) への直接リンクを指定します。

```
components:
 - alias: exec-plugin
   type: chePlugin
   reference: https://raw.githubusercontent.com.../plugin/1.0.1/meta.yaml
```



注記

単一のコンポーネント定義で **id** および **reference** フィールドを組み合わせることはできません。それらは相互に排他的です。

3.2.4.2.5. chePlugin コンポーネント設定のチューニング

chePlugin コンポーネントを詳細に調整する必要がある場合があり、その場合はコンポーネントの設定を使用できます。この例は、プラグイン設定を使用して JVM を設定する方法を示しています。

```
id: redhat/java/0.38.0
type: chePlugin
preferences:
  java.jdt.ls.vmargs: '-noverify -Xmx1G -XX:+UseG1GC -XX:+UseStringDeduplication'
```

設定は配列として指定することもできます。

```
id: redhat/java/0.38.0
type: chePlugin
preferences:
  go.lintFlags: ["--enable-all", "--new"]
```

3.2.4.2.6. コンポーネントタイプ: kubernetes

OpenShift コンポーネントの一覧からの設定の適用を可能にする複雑なコンポーネントタイプ。

コンテンツは、コンポーネントのコンテンツを含むファイルを参照する **reference** 属性で指定できます。

```
components:
  - alias: mysql
    type: kubernetes
    reference: petclinic.yaml
    selector:
      app.kubernetes.io/name: mysql
      app.kubernetes.io/component: database
      app.kubernetes.io/part-of: petclinic
```

または、このコンポーネントを持つ devfile を REST API に追加するには、**referenceContent** フィールドを使用して OpenShift **List** オブジェクトのコンテンツを devfile に組み込むことができます。

```
components:
```

```
- alias: mysql
  type: kubernetes
  reference: petclinic.yaml
  referenceContent: |
      kind: List
      items:
       -
        apiVersion: v1
        kind: Pod
        metadata:
         name: ws
        spec:
         containers:
         ... etc
```

3.2.4.2.7. コンテナーのエントリーポイントの上書き

OpenShift で認識される場合と同様です。

一覧には複数のコンテナーが含まれる場合があります (デプロイメントの Pod または Pod テンプレートに含まれる場合があります)。エントリーポイントの変更を適用するコンテナーを選択するには、以下を実行します。

エントリーポイントは、以下のように定義できます。

```
components:
  - alias: appDeployment
    type: kubernetes
    reference: app-deployment.yaml
    entrypoints:
    - parentName: mysqlServer
      command: ['sleep']
      args: ['infinity']
    - parentSelector:
        app: prometheus
      args: ['-f', '/opt/app/prometheus-config.yaml']
```

entrypoints 一覧には、適用する **command** および **args** パラメーターと共にコンテナーを選択する場合の制約が含まれます。上記の例では、制約は **parentName: mysqlServer** であり、これにより **mysqlServer** という親オブジェクトで定義されるすべてのコンテナーにコマンドが適用されます。親オブジェクトは、参照ファイル (上記の例では **app-deployment.yaml**) で定義される一覧の最上位オブジェクトであることが想定されます。

その他のタイプの制約 (およびそれらの組み合わせ) も使用できます。

containerName
: コンテナーの名前

parentName
: 上書きするコンテナーが (間接的に) 含まれる親オブジェクトの名前

parentSelector
: 親オブジェクトに必要なラベルのセット

これらの制約の組み合わせは、参照される OpenShift **List** 内のコンテナーを正確に特定するために使用されます。

3.2.4.2.8. コンテナー環境変数の上書き

OpenShift コンポーネントのエントリーポイントをプロビジョニングまたは上書きするには、以下のように設定します。

```
components:
  - alias: appDeployment
    type: kubernetes
    reference: app-deployment.yaml
    env:
      - name: ENV_VAR
        value: value
```

これは、一時的なコンテンツの場合や、参照されるコンテンツを編集するためのアクセスがない場合に役立ちます。指定される環境変数は各 init コンテナーおよびすべての Pod および Deployment 内のコンテナーにプロビジョニングされます。

3.2.4.2.9. mount-source オプションの指定

プロジェクトのソースディレクトリーのマウントをコンテナーに指定するには、**mountSources** パラメーターを使用します。

```
components:
  - alias: appDeployment
    type: kubernetes
    reference: app-deployment.yaml
    mountSources: true
```

有効にされている場合、プロジェクトソースマウントは指定されるコンポーネントのすべてのコンテナーに適用されます。このパラメーターは、**chePlugin** タイプのコンポーネントにも適用できます。

3.2.4.2.10. コンポーネントタイプ: dockerimage

ワークスペースでコンテナーのコンテナーイメージベースの設定を定義することが可能なコンポーネントタイプ。**dockerimage** タイプのコンポーネントは、カスタムツールをワークスペースに組み込みます。コンポーネントは、そのイメージによって特定されます。

```
components:
  - alias: maven
    type: dockerimage
    image: quay.io/eclipse/che-java11-maven:nightly
    volumes:
      - name: mavenrepo
        containerPath: /root/.m2
    env:
      - name: ENV_VAR
        value: value
    endpoints:
      - name: maven-server
        port: 3101
        attributes:
          protocol: http
          secure: 'true'
          public: 'true'
          discoverable: 'false'
```

```
      memoryLimit: 1536M
      memoryRequest: 256M
      command: ['tail']
      args: ['-f', '/dev/null']
```

最小の dockerimage コンポーネントの例

```
apiVersion: 1.0.0
metadata:
   name: MyDevfile
components:
   - type: dockerimage
     image: golang
     memoryLimit: 512Mi
     command: ['sleep', 'infinity']
```

これはコンポーネントのタイプ **dockerimage** および、**image**属性名を指定し、通常の Docker の命名規則を使用してコンポーネントに使用されるイメージの名前を指定します。つまり、上記の **type** 属性は **docker.io/library/golang:latest** と等しくなります。

dockerimage コンポーネントには、Red Hat CodeReady Workspaces のイメージで提供される **ツール** の有効な統合に必要な追加のリソースおよび情報を使用してイメージを拡張するための数多くの機能が含まれています。

3.2.4.2.11. プロジェクトソースのマウント

dockerimage コンポーネントがプロジェクトソースにアクセスできるようにするには、**mountSources** 属性を **true** に設定する必要があります。

```
apiVersion: 1.0.0
metadata:
   name: MyDevfile
components:
   - type: dockerimage
     image: golang
     memoryLimit: 512Mi
     command: ['sleep', 'infinity']
```

ソースは、イメージの実行中のコンテナーで利用可能な **CHE_PROJECTS_ROOT** 環境変数に保存される場所にマウントされます。この場所はデフォルトで **/projects** に設定されます。

3.2.4.2.12. コンテナーエントリーポイント

dockerimage の **command** 属性は、他の引数と共に、イメージから作成されるコンテナーの **entrypoint** コマンドを変更するために使用されます。Red Hat CodeReady Workspaces では、コンテナーを無限に実行して、コンテナーに接続し、任意のコマンドをいつでも実行できるようにする必要があります。**sleep** コマンドの可用性と **infinity** 引数のサポートは特定のイメージで使用されるベースイメージによって異なるため、CodeReady Workspaces はこの動作を独自に自動的に挿入することはできません。ただし、この機能を活用すると、たとえば、変更した設定で必要なサーバーを起動することなどが可能になります。

3.2.4.2.13. 永続ストレージ

任意のタイプのコンポーネントは、イメージ内の特定の場所にマウントするカスタムボリュームを指定

任意のタイプのコンポーネントは、イメージ内の特定の場所にマウントするカスタムボリュームを指定することができます。ボリューム名はすべてのコンポーネントで共有されるため、このメカニズムを使用してコンポーネント間でファイルシステムを共有することもできることに留意してください。

dockerimage タイプのボリュームを指定する例:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: dockerimage
    image: golang
    memoryLimit: 512Mi
    mountSources: true
    command: ['sleep', 'infinity']
    volumes:
      - name: cache
        containerPath: /.cache
```

cheEditor/**chePlugin** タイプのボリュームを指定する例:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: cheEditor
    alias: theia-editor
    id: eclipse/che-theia/next
    env:
    - name: HOME
      value: $(CHE_PROJECTS_ROOT)
    volumes:
    - name: cache
      containerPath: /.cache
```

kubernetes/**openshift** タイプのボリュームを指定する例：

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: openshift
    alias: mongo
    reference: mongo-db.yaml
    volumes:
    - name: mongo-persistent-storage
      containerPath: /data/db
```

3.2.4.2.14. コンポーネントのコンテナーメモリー制限の指定

dockerimage、**chePlugin**、または **cheEditor** のコンテナーメモリー制限を指定するには、**memoryLimit** パラメーターを使用します。

```
  components:
   - alias: exec-plugin
```

```
    type: chePlugin
    id: eclipse/che-machine-exec-plugin/0.0.1
    memoryLimit: 1Gi
  - alias: maven
    type: dockerimage
    image: quay.io/eclipse/che-java11-maven:nightly
    memoryLimit: 512M
```

この制限は、指定されるコンポーネントのすべてのコンテナーに適用されます。

cheEditor および **chePlugin** コンポーネントタイプの場合、RAM 制限はプラグイン記述子ファイル (通常は **meta.yaml** という名前) で記述できます。

指定がない場合は、システム全体のデフォルトが適用されます (**CHE_WORKSPACE_SIDECAR_DEFAULT__MEMORY__LIMIT__MB** システムプロパティーの説明を参照)。

3.2.4.2.15. コンポーネントのコンテナーメモリー要求の指定

dockerimage、**chePlugin**、または **cheEditor** のコンテナーメモリー要求を指定するには、**memoryRequest** パラメーターを使用します。

```
components:
 - alias: exec-plugin
   type: chePlugin
   id: eclipse/che-machine-exec-plugin/0.0.1
   memoryLimit: 1Gi
   memoryRequest: 512M
 - alias: maven
   type: dockerimage
   image: quay.io/eclipse/che-java11-maven:nightly
   memoryLimit: 512M
   memoryRequest: 256M
```

この制限は、指定されるコンポーネントのすべてのコンテナーに適用されます。

cheEditor および **chePlugin** コンポーネントタイプの場合、RAM 要求はプラグイン記述子ファイル (通常は **meta.yaml** という名前) で記述できます。

指定がない場合は、システム全体のデフォルトが適用されます（**CHE_WORKSPACE_SIDECAR_DEFAULT__MEMORY__REQUEST__MB** の説明を参照）。

3.2.4.2.16. コンポーネントのコンテナー CPU 制限の指定

chePlugin、**cheEditor**、または **dockerimage** のコンテナー CPU 制限を指定するには、**cpuLimit** パラメーターを使用します。

```
components:
 - alias: exec-plugin
   type: chePlugin
   id: eclipse/che-machine-exec-plugin/0.0.1
   cpuLimit: 1.5
 - alias: maven
```

```
    type: dockerimage
    image: quay.io/eclipse/che-java11-maven:nightly
    cpuLimit: 750m
```

この制限は、指定されるコンポーネントのすべてのコンテナーに適用されます。

cheEditor および **chePlugin** コンポーネントタイプの場合、CPU 制限はプラグイン記述子ファイル (通常は **meta.yaml** という名前) で記述できます。

指定がない場合は、システム全体のデフォルトが適用されます（**CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__LIMIT__CORES** の説明を参照）。

3.2.4.2.17. コンポーネントのコンテナー CPU 要求の指定

chePlugin、**cheEditor**、または **dockerimage** のコンテナー CPU 要求を指定するには、**cpuRequest** パラメーターを使用します。

```
components:
 - alias: exec-plugin
   type: chePlugin
   id: eclipse/che-machine-exec-plugin/0.0.1
   cpuLimit: 1.5
   cpuRequest: 0.225
 - alias: maven
   type: dockerimage
   image: quay.io/eclipse/che-java11-maven:nightly
   cpuLimit: 750m
   cpuRequest: 450m
```

この制限は、指定されるコンポーネントのすべてのコンテナーに適用されます。

cheEditor および **chePlugin** コンポーネントタイプの場合、CPU 要求はプラグイン記述子ファイル (通常は **meta.yaml** という名前) で記述できます。

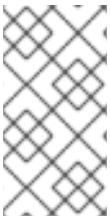
指定がない場合は、システム全体のデフォルトが適用されます（**CHE_WORKSPACE_SIDECAR_DEFAULT__CPU__REQUEST__CORES** の説明を参照）。

3.2.4.2.18. 環境変数

Red Hat CodeReady Workspaces では、コンポーネントの設定で利用可能な環境変数を変更して、Docker コンテナーを設定できます。環境変数は、コンポーネントタイプ**dockerimage**、**chePlugin**、**cheEditor**、**kubernetes**、**openshift** でサポートされます。コンポーネントに複数のコンテナーがある場合、環境変数は各コンテナーにプロビジョニングされます。

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: dockerimage
    image: golang
    memoryLimit: 512Mi
    mountSources: true
    command: ['sleep', 'infinity']
    env:
      - name: GOPATH
```

```
      value: $(CHE_PROJECTS_ROOT)/go
  - type: cheEditor
    alias: theia-editor
    id: eclipse/che-theia/next
    memoryLimit: 2Gi
    env:
    - name: HOME
      value: $(CHE_PROJECTS_ROOT)
```



注記

- 変数の拡張は環境変数間で機能し、この場合、変数の参照に Kubernetes の命名規則が使用されます。
- 事前定義された変数は、カスタム定義で使用できます。

以下の環境変数は、CodeReady Workspaces サーバーで事前に設定されます。

- **CHE_PROJECTS_ROOT**: プロジェクトディレクトリーの場所 (コンポーネントがソースをマウントしないと、プロジェクトにアクセスできない点に注意してください)。
- **CHE_WORKSPACE_LOGS_ROOT__DIR**: すべてのコンポーネントに共通するログの場所。コンポーネントでこのディレクトリーにログを配置する選択をする場合、ログファイルは他のすべてのコンポーネントからアクセスできます。
- **CHE_API_INTERNAL**: CodeReady Workspaces サーバーとの通信に使用される CodeReady Workspaces サーバー API エンドポイントへの URL。
- **CHE_WORKSPACE_ID**: 現行ワークスペースの ID。
- **CHE_WORKSPACE_NAME**: 現行ワークスペースの名前。
- **CHE_WORKSPACE_NAMESPACE**: 現行ワークスペースの CodeReady Workspaces プロジェクト。この環境変数は、ワークスペースが属するユーザーまたは組織の名前です。これは、ワークスペースがデプロイされる OpenShift プロジェクトとは異なることに注意してください。
- **CHE_MACHINE_TOKEN**: CodeReady Workspaces サーバーに対する要求の認証に使用されるトークン。
- **CHE_MACHINE_AUTH_SIGNATURE__PUBLIC__KEY**: CodeReady Workspaces サーバーとの通信のセキュリティーを保護するために使用するパブリックキー。
- **CHE_MACHINE_AUTH_SIGNATURE__ALGORITHM**: CodeReady Workspaces サーバーとのセキュリティーが保護された通信で使用される暗号化アルゴリズム。

devfile は、コンポーネントのコンテナーでクローン作成されたプロジェクトを見つけるために **CHE_PROJECTS_ROOT** 環境変数のみが必要になる場合があります。さらに高度な devfile は、**CHE_WORKSPACE_LOGS_ROOT__DIR** 環境変数を使用してログを読み取る場合があります (例: devfile コマンドの一部として実行します)。CodeReady Workspaces サーバーに安全にアクセスするために使用される環境変数は、ほとんどの場合は devfile の範囲外であり、通常 CodeReady Workspaces プラグインによって処理される高度なユースケースのみを対象としています。

3.2.4.2.19. エンドポイント

すべてのタイプのコンポーネントは、Docker イメージが公開するエンドポイントを指定できます。こ

れらのエンドポイントは、CodeReady Workspaces クラスターが Kubernetes Ingress または OpenShift ルートを使用して実行されており、これがワークスペース内の他のコンポーネントに対して実行されている場合にユーザーがアクセスすることができます。アプリケーションまたはデータベースサーバーがポートでリッスンし、これと直接対話できるようにするか、または他のコンポーネントがこれと対話できるようにする必要がある場合は、アプリケーションまたはデータベースのエンドポイントを作成できます。

エンドポイントには、以下の例のように複数のプロパティーがあります。

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
  - name: my-go-project
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
components:
  - type: dockerimage
    image: golang
    memoryLimit: 512Mi
    mountSources: true
    command: ['sleep', 'infinity']
    env:
      - name: GOPATH
        value: $(CHE_PROJECTS_ROOT)/go
      - name: GOCACHE
        value: /tmp/go-cache
    endpoints:
     - name: web
       port: 8080
       attributes:
         discoverable: false
         public: true
         protocol: http
  - type: dockerimage
    image: postgres
    memoryLimit: 512Mi
    env:
      - name: POSTGRES_USER
        value: user
      - name: POSTGRES_PASSWORD
        value: password
      - name: POSTGRES_DB
        value: database
    endpoints:
      - name: postgres
        port: 5432
        attributes:
          discoverable: true
          public: false
```

ここで、それぞれが単一のエンドポイントを定義する 2 つの Docker イメージがあります。エンドポイントは、ワークスペース内または (UI などを使用して) パブリックにアクセス可能なポートです。各エンドポイントには、名前とポート (コンテナー内で実行している特定のサーバーがリッスンしている

ポート) があります。以下は、エンドポイントに設定できるいくつかの属性です。

- **discoverable**: エンドポイントが検出可能である場合、そのエンドポイントをワークスペースコンテナー内のホスト名として使用してアクセスできます (OpenShift のコンテキストでは、そのサービスが指定された名前で作成されます)。55
- **public**: エンドポイントはワークスペース外からもアクセスできます (このエンドポイントは CodeReady Workspaces ユーザーインターフェースからアクセスできます)。このエンドポイントは、ポート **80** または **443** で常に公開されます (**tls** が CodeReady Workspaces で有効にされるかどうかによって異なります)。
- **protocol**: パブリックエンドポイントの場合、プロトコルはエンドポイントアクセスの URL の作成方法についての UI に対するヒントとなります。一般的な値は **http**、**https**、**ws**、**wss** です。
- **secure**: エンドポイントがアクセスの付与に JWT ワークスペーストークンを必要とする JWT プロキシーの背後に配置するかどうかを指定するブール値 (デフォルトは **false**)。JWT プロキシーはサーバーと同じ Pod にデプロイされ、サーバーは **127.0.0.1** などのローカルループバックインターフェースでのみリッスンすることを前提としています。

警告

ローカルループバック以外のインターフェースをリッスンすると、このサーバーは対応する IP アドレスのクラスターネットワーク内に JWT 認証がなくてもアクセスできるため、セキュリティー上のリスクが発生します。

- **path**: エンドポイントへの URL のパスの部分です。これはデフォルトで / に設定されます。つまり、エンドポイントはコンポーネントによって定義されるサーバーの Web ルートでアクセスできることを前提としています。
- **unsecuredPaths**: **secure** 属性が **true** に設定されていても、セキュアでない状態のままになるエンドポイントパスのコンマ区切りの一覧。
- **cookiesAuthEnabled**: **true** (デフォルトは **false**) に設定すると、JWT ワークスペーストークンは自動的にフェッチされ、ワークスペース固有の Cookie に組み込まれ、要求による JWT プロキシーのパススルーが許可されます。

警告

この設定は、POST 要求を使用するサーバーと併用される場合に CSRF 攻撃を生じさせる可能性があります。

コンポーネント内で新規サーバーを起動すると、CodeReady Workspaces はこれを自動検出し、UI はこのポートを **public** ポートとして自動的に公開します。これは、たとえば Web アプリケーションのデバッグに役立ちます。コンテナー (データベースサーバーなど) を使用して自動起動するサーバーの場

合、これを実行することはできません。このコンポーネントについては、エンドポイントを明示的に指定します。

kubernetes/**openshift** および **chePlugin**/**cheEditor** タイプのエンドポイントを指定する例:

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: cheEditor
    alias: theia-editor
    id: eclipse/che-theia/next
    endpoints:
    - name: 'theia-extra-endpoint'
      port: 8880
      attributes:
        discoverable: true
        public: true

  - type: chePlugin
    id: redhat/php/latest
    memoryLimit: 1Gi
    endpoints:
    - name: 'php-endpoint'
      port: 7777

  - type: chePlugin
    alias: theia-editor
    id: eclipse/che-theia/next
    endpoints:
    - name: 'theia-extra-endpoint'
      port: 8880
      attributes:
        discoverable: true
        public: true

  - type: openshift
    alias: webapp
    reference: webapp.yaml
    endpoints:
    - name: 'web'
      port: 8080
      attributes:
        discoverable: false
        public: true
        protocol: http

  - type: openshift
    alias: mongo
    reference: mongo-db.yaml
    endpoints:
    - name: 'mongo-db'
      port: 27017
      attributes:
        discoverable: true
        public: false
```

3.2.4.2.20. OpenShift リソース

複雑なデプロイメントを記述するには、devfile に OpenShift リソース一覧への参照を含めます。OpenShift リソース一覧はワークスペースの一部になります。

重要

- CodeReady Workspaces は、OpenShift リソース一覧のすべてのリソースを単一のデプロイメントにマージします。
- 名前の競合やその他の問題が生じないように、十分注意してこの一覧を準備してください。

表3.2 サポートされる OpenShift リソース

プラットフォーム	サポートされるリソース
OpenShift	**deployments**、**pods**、**services**、**persistent volume claims**、**secrets**、**ConfigMaps**、**Routes**

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
  - name: my-go-project
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
components:
  -  type: kubernetes
     reference: ../relative/path/postgres.yaml
```

上記のコンポーネントは、devfile 自体の場所と相対的なファイルを参照します。つまり、この devfile は、devfile の場所を指定する CodeReady Workspaces factory によってのみ読み込み可能であるため、参照される OpenShift リソース一覧の場所を特定することができます。

以下は **postgres.yaml** ファイルの例です。

```
apiVersion: v1
kind: List
items:
-
    apiVersion: v1
    kind: Deployment
    metadata:
        name: postgres
        labels:
            app: postgres
    spec:
        template:
        metadata:
```

```
          name: postgres
          app:
            name: postgres
        spec:
          containers:
          - image: postgres
            name: postgres
            ports:
            - name: postgres
              containerPort: 5432
              volumeMounts:
              - name: pg-storage
                mountPath: /var/lib/postgresql/data
          volumes:
          - name: pg-storage
            persistentVolumeClaim:
              claimName: pg-storage
-
    apiVersion: v1
    kind: Service
    metadata:
        name: postgres
        labels:
          app: postgres
          name: postgres
    spec:
        ports:
          - port: 5432
            targetPort: 5432
        selector:
          app: postgres
-
    apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
        name: pg-storage
      labels:
        app: postgres
    spec:
        accessModes:
         - ReadWriteOnce
        resources:
          requests:
            storage: 1Gi
```

関連付けられた OpenShift 一覧を含む devfile の基本的な例については、redhat-developer GitHub の web-nodejs-with-db-sample を参照してください。

リソースのサブセットのみを必要とする汎用または大規模なリソース一覧を使用する場合は、セレクター (通常は OpenShift セレクターとして一覧にあるリソースラベルで機能する) を使用して、一覧から特定のリソースを選択できます。

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
```

```
  - name: my-go-project
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
components:
  - type: kubernetes
    reference: ../relative/path/postgres.yaml
    selector:
      app: postgres
```

また、リソース一覧にあるコンテナーのエントリーポイント (コマンドおよび引数) を変更することもできます。高度なユースケースについての詳細は、 特定コンテナーイメージの定義について参照してください。

3.2.4.3. devfile へのコマンドの追加

devfile を使用すると、ワークスペースで実行できるコマンドを指定できます。すべてのコマンドにはアクションのサブセットが含まれますが、それらは実行されるコンテナーの特定のコンポーネントに関連するものです。

```
commands:
  - name: build
    actions:
      - type: exec
        component: mysql
        command: mvn clean
        workdir: /projects/spring-petclinic
```

コマンドを使用するとワークスペースを自動化できます。コードをビルドし、テストするか、またはデータベースのクリーニングを実行するためのコマンドを定義できます。

以下は、2 種類のコマンドです。

- CodeReady Workspaces 固有のコマンド: コマンドを実行するコンポーネントを完全に制御できます。
- エディター固有のコマンド: エディター固有のコマンド定義を使用できます (例: Che-Theia の **tasks.json** および **launch.json**。これは、VS Code でのこれらのファイルの動作と同等です)。

3.2.4.3.1. CodeReady Workspaces 固有のコマンド

CodeReady Workspaces 固有のコマンドの機能

- 実行するコマンドを指定する **actions** 属性。
- コマンドを実行するコンテナーを指定する **component** 属性。

コマンドは、コンテナーのデフォルトシェルを使用して実行します。

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
  - name: my-go-project
```

```
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
components:
  - type: dockerimage
    image: golang
    alias: go-cli
    memoryLimit: 512Mi
    mountSources: true
    command: ['sleep', 'infinity']
    env:
      - name: GOPATH
        value: $(CHE_PROJECTS_ROOT)/go
      - name: GOCACHE
        value: /tmp/go-cache
commands:
  - name: compile and run
    actions:
     - type: exec
       component: go-cli
       command: “go get -d && go run main.go”
       workdir: “${CHE_PROJECTS_ROOT}/src/github.com/acme/my-go-project”
```

注記

- コマンドで使用されるコンポーネントにはエイリアスが必要です。このエイリアスは、コマンド定義でコンポーネントを参照するために使用されます。例: コンポーネント定義の **alias: go-cli** およびコマンド定義の **component: go-cli**。これにより、Red Hat CodeReady Workspaces がコマンドを実行できる正しいコンテナーを検索できます。
- コマンドには 1 つのアクションのみを指定できます。

3.2.4.3.2. エディター固有のコマンド

ワークスペースのエディターがサポートする場合、devfile はエディター固有の形式で追加の設定を指定できます。これはワークスペースエディター自体にある統合コードに依存するため、汎用的なメカニズムではありません。ただし、Red Hat CodeReady Workspaces 内のデフォルトの Che-Theia エディターは、devfile で提供される **tasks.json** および **launch.json** ファイルを把握できるように機能します。

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
projects:
  - name: my-go-project
    clonePath: go/src/github.com/acme/my-go-project
    source:
      type: git
      location: https://github.com/acme/my-go-project.git
commands:
  - name: tasks
    actions:
      - type: vscode-task
```

```
referenceContent: >
  {
    "version": "2.0.0",
    "tasks": [
      {
        "label": "create test file",
        "type": "shell",
        "command": "touch ${workspaceFolder}/test.file"
      }
    ]
  }
```

この例では、devfile と **tasks.json** ファイルの関連付けを示しています。このコマンドをタスク定義として解釈するよう Che-Theia エディターに指示する **vscode-task** タイプと、ファイル自体のコンテンツを含む **referenceContent** 属性があることに留意してください。このファイルを devfile とは別に保存し、**reference** 属性を使用して相対パスまたは絶対 URL を指定することもできます。

vscode-task コマンドのほかにも、Che-Theia エディターは起動設定を指定して **vscode-launch** タイプを認識します。

3.2.4.3.3. コマンドプレビュー URL

Web UI を公開するコマンドのプレビュー URL を指定できます。この URL は、コマンドの実行時に開けるように提供されます。

```
commands:
  - name: tasks
    previewUrl:
      port: 8080 (1)
      path: /myweb (2)
    actions:
    - type: exec
      component: go-cli
      command: "go run webserver.go"
      workdir: ${CHE_PROJECTS_ROOT}/webserver
```

(1) アプリケーションがリッスンする TCP ポート。必須パラメーター。

(2) UI への URL のパスの部分。オプションのパラメーター。デフォルトは root (/) です。

上記の例では、**http://__<server-domain>__/myweb** が開きます。ここで、**<server-domain>** は動的に作成される OpenShift Route の URL です。

3.2.4.3.3.1. プレビュー URL を開くデフォルトの方法の設定

デフォルトで、URL を開く際の設定についてユーザーに尋ねる通知が表示されます。

サービス URL のプレビューに使用する推奨される方法を指定するには、以下を実行します。

1. **File → Settings → Open Preferences**で CodeReady Workspaces 設定を開き、 **CodeReady Workspaces** セクションで **che.task.preview.notifications** を見つけます。
2. 使用できる値の一覧から選択します。
   - **on** –URL を開く際の設定についてユーザーに尋ねる通知を有効にします。

- **alwaysPreview** – プレビュー URL は、タスクが実行されるとすぐに自動的に **Preview** パネルで開きます。
- **alwaysGoTo** –プレビュー URL は、タスクが実行されるとすぐに別のブラウザータブで自動的に開きます。
- **off** –プレビュー URL を開くを無効にします (自動的に無効にし、通知を出します)。

3.2.4.4. devfile 属性

devfile 属性を使用して、各種の機能を設定することもできます。

3.2.4.4.1. 属性: editorFree

エディターが devfile で指定されない場合、デフォルトが指定されます。エディターが必要ない場合は、**editorFree** 属性を使用します。デフォルト値の **false** は、devfile がデフォルトエディターのプロビジョニングを要求することを意味します。

エディターのない devfile の例

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
components:
  - alias: myApp
    type: kubernetes
    reference: my-app.yaml
attributes:
  editorFree: true
```

3.2.4.4.2. 属性: persistVolumes（一時モード）

デフォルトで、devfile で指定されるボリュームおよび PVC は、コンテナーの再起動後もデータを永続化させるためにホストフォルダーにバインドされます。ボリュームのバックエンドの速度が遅い場合など、データの永続性を無効にしてワークスペースをより高速にするには、devfile の **persistVolumes** 属性を変更します。デフォルト値は **true** です。設定されたボリュームおよび PVC に **emptyDir** を使用するには、**false** に設定します。

一時モードが有効にされた devfile の例

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
projects:
  - name: petclinic
    source:
      type: git
      location: 'https://github.com/che-samples/web-java-spring-petclinic.git'
attributes:
  persistVolumes: false
```

3.2.4.4.3. 属性: asyncPersist (非同期ストレージ)

persistVolumes が **false** に設定されている場合（上記を参照）は、追加の属性 **asyncPersist** を **true**

に設定して非同期ストレージを有効にすることができます。詳細は、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#configuring-storage-types_crw を参照してください。

非同期ストレージが有効にされている devfile の例

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
projects:
  - name: petclinic
    source:
      type: git
      location: 'https://github.com/che-samples/web-java-spring-petclinic.git'
attributes:
  persistVolumes: false
  asyncPersist: true
```

3.2.4.4.4. 属性: mergePlugins

このプロパティーは、プラグインをワークスペースに追加する方法を手動で制御できるようにするために設定できます。**mergePlugins** プロパティーが **true** に設定されている場合、Che はプラグインを組み合わせて同じコンテナーの複数のインスタンスの実行を回避するよう試行します。このプロパティーが devfile に含まれない場合のデフォルト値は Che 設定プロパティー **che.workspace.plugin_broker.default_merge_plugins** で制御されます。**mergePlugins: false** 属性を devfile に追加すると、そのワークスペースのプラグインのマージが無効になります。

プラグインのマージが無効にされている devfile の例

```
apiVersion: 1.0.0
metadata:
  name: petclinic-dev-environment
projects:
  - name: petclinic
    source:
      type: git
      location: 'https://github.com/che-samples/web-java-spring-petclinic.git'
attributes:
  mergePlugins: false
```

3.2.5. Red Hat CodeReady Workspaces 2.6 でサポートされるオブジェクト

以下の表は、Red Hat CodeReady Workspaces 2.6 で部分的にサポートされるオブジェクトの一覧です。

オブジェクト	API	Kubernetes インフラストラクチャー	OpenShift インフラストラクチャー	注
Pod	Kubernetes	Yes	Yes	-

オブジェクト	API	Kubernetes インフラストラクチャー	OpenShift インフラストラクチャー	注
デプロイメント	Kubernetes	Yes	Yes	-
ConfigMap	Kubernetes	Yes	Yes	-
PVC	Kubernetes	Yes	Yes	-
Secret	Kubernetes	Yes	Yes	-
サービス	Kubernetes	Yes	Yes	-
Ingress	Kubernetes	Yes	No	minishift を使用すると Ingress を作成でき、これはホストが指定されていると機能します (OpenShift はそのルートを作成します)。ただし、**loadBalancer** IP はプロビジョニングされません。OpenShift インフラストラクチャーノードの Ingress サポートを追加するには、提供される Ingress に基づいてルートを生成します。
Route	OpenShift	No	Yes	OpenShift recipe は Kubernetes インフラストラクチャーと互換性がなければならず、提供されるルートではなく Ingress が生成される必要があります。
Template	OpenShift	Yes	Yes	Kubernetes API はテンプレートをサポートしません。recipe 内のテンプレートを使用するワークスペースが正常に開始し、デフォルトのパラメーターが解決されます。

関連資料

- devfile の仕様

3.3. 新規 CODEREADY WORKSPACES 2.6 ワークスペースの作成および設定

3.3.1. Dashboard からの新規ワークスペースの作成

この手順では、**Dashboard** を使用して新しい CodeReady Workspaces devfile を作成し、編集する方法を説明します。

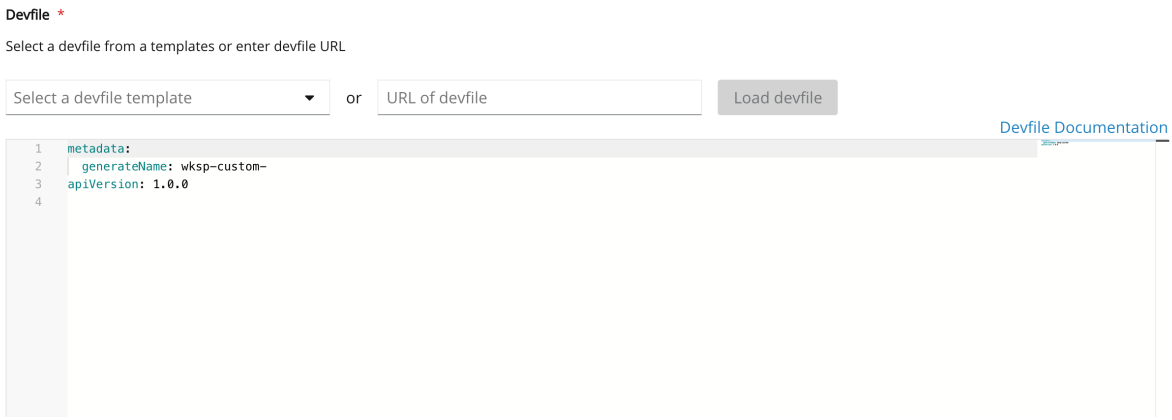
前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。

手順

devfile を編集するには、以下を実行します。

1. **Workspaces** ウィンドウで、**Add Workspace** ボタンをクリックします。**Custom Workspace** ページを開く必要があります。
2. **Devfile** セクションまでスクロールダウンし、 **Devfile editor** を使用して devfile を編集します。



以下の例を参照してください。

例: devfile を使用したワークスペースへの .git プロジェクトの追加

以下のインスタンスでは、プロジェクト **crw** はユーザーのプロジェクトのサンプルとして機能します。ユーザーは、devfile の **name** 属性を使用してこのプロジェクトを指定します。**location** 属性は、Git リポジトリーまたは ZIP アーカイブへの URL で表されるソースリポジトリーを定義します。

プロジェクトをワークスペースに追加するには、以下のセクションを追加または編集します。

```
projects:
  - name: <crw>
    source:
      type: git
      location: 'https://github.com/<github-organization>/<crw>.git'
```

詳細は、「devfile リファレンス」 のセクションを参照してください。

3.3.2. ワークスペースへのプロジェクトの追加

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペース。新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定について参照してください。

手順

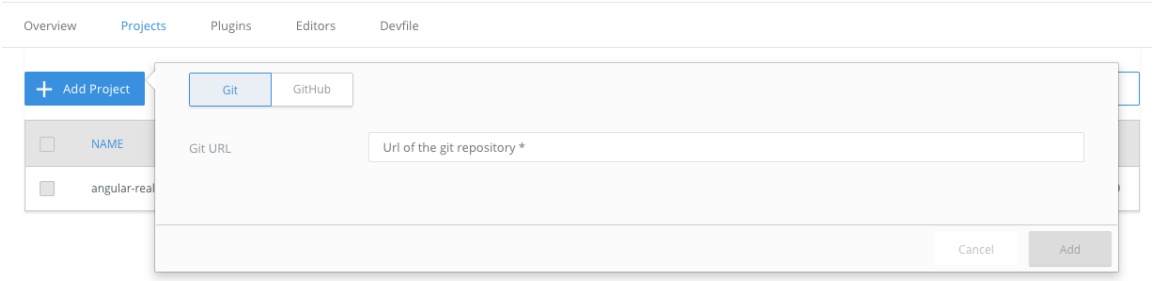
ワークスペースにプロジェクトを追加するには、以下を実行します。

1. **Workspaces** ページに移動し、更新するワークスペースをクリックします。
   ここでは、ワークスペースにプロジェクトを追加するための 2 つの方法を使用できます。

2. **Projects** タブを使用。

   a. **Projects** タブを開き、**Add Project** ボタンをクリックします。

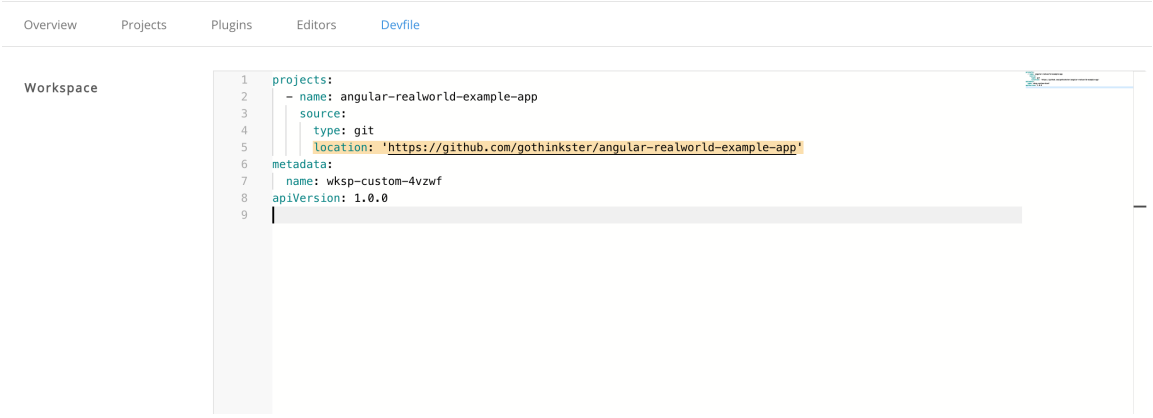
   b. Git URL または GitHub アカウントを使用してプロジェクトをインポートするかどうかを選択します。



3. **Devfile** タブを使用。

   a. **Devfile** タブを開きます。

   b. **Devfile editor** で、必要なプロジェクトを指定して **projects** セクションを追加します。



デモの例は、以下を参照してください。

例: devfile を使用したワークスペースへの .git プロジェクトの追加

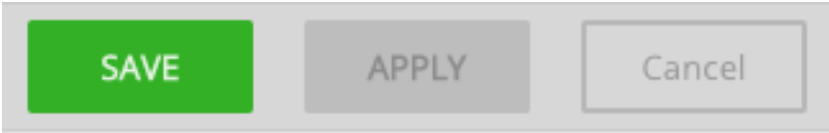
以下のインスタンスでは、プロジェクト **crw** はユーザーのプロジェクトのサンプルとして機能します。ユーザーは、devfile の **name** 属性を使用してこのプロジェクトを指定します。**location** 属性は、Git リポジトリーまたは ZIP アーカイブへの URL で表されるソースリポジトリーを定義します。

プロジェクトをワークスペースに追加するには、以下のセクションを追加または編集します。

```
projects:
  - name: <crw>
    source:
      type: git
      location: 'https://github.com/<github-organization>/<crw>.git'
```

詳細は、「devfile リファレンス」 のセクションを参照してください。

1. プロジェクトを追加したら、**Save** ボタンをクリックしてこのワークスペース設定を保存するか、**Apply** ボタンをクリックして実行中のワークスペースに変更を適用します。



3.3.3. ワークスペースの設定およびツールの追加

3.3.3.1. プラグインの追加

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペース。新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定について参照してください。

手順

ワークスペースにプラグインを追加するには、以下を実行します。

1. **Plugins** タブをクリックします。
2. 追加するプラグインを有効にし、**Save** ボタンをクリックします。

3.3.3.2. ワークスペースエディターの定義

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペース。新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定について参照してください。

手順

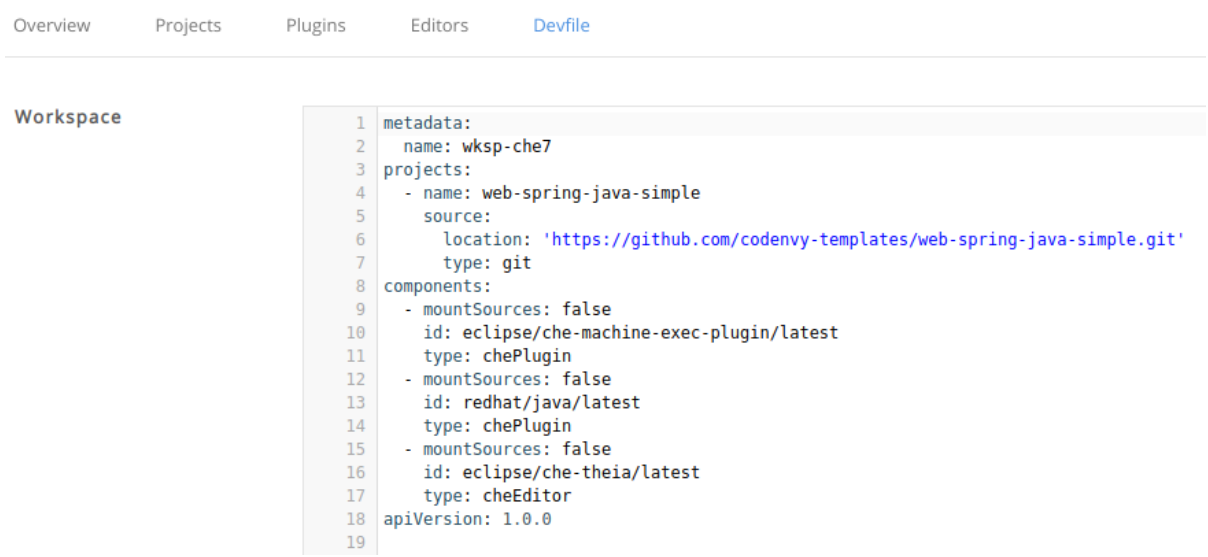
ワークスペースで使用するエディターを定義するには、以下を実行します。

1. **Editors** タブをクリックします。

注記

CodeReady Workspaces 2.6 に推奨されるエディターは Che-Theia です。

2. 追加するエディターを有効にして、**Save** ボタンをクリックします。
3. **Devfile** タブをクリックして変更を表示します。



3.3.3.3. 特定のコンテナーイメージの定義

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペース。新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定について参照してください。

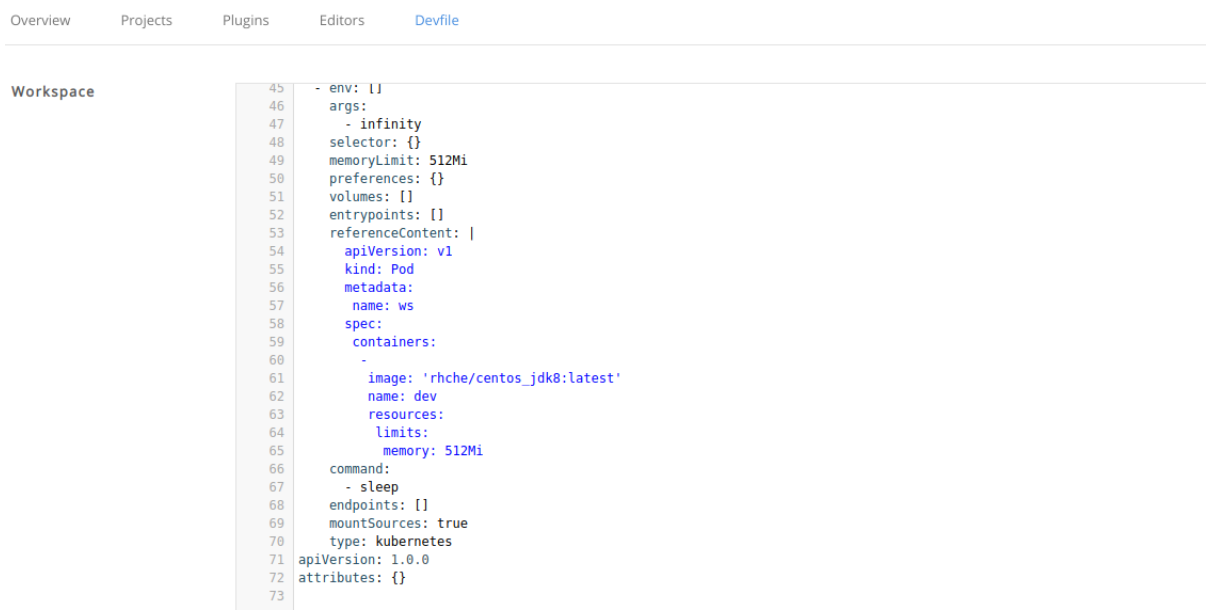
手順

新しいコンテナーイメージを追加するには、以下を実行します。

1. **Devfile** タブの **components** プロパティーの下に以下のセクションを追加します。

```
components:
  - mountSources: true
    command:
      - sleep
    args:
      - infinity
    memoryLimit: 1Gi
    alias: maven3-jdk11
    type: dockerimage
    endpoints:
      - name: 8080/tcp
        port: 8080
    volumes:
      - name: projects
        containerPath: /projects
    image: 'maven:3.6.0-jdk-11'
```

2. CodeReady Workspaces 2.5 recipe コンテンツを **referenceContent** として CodeReady Workspaces 2.6 devfile に追加します。



a. 元の CodeReady Workspaces 2.5 設定からタイプを設定します。以下は、作成されるファイルの例になります。

```
type: kubernetes
referenceContent: |
  apiVersion: v1
  kind: Pod
  metadata:
   name: ws
  spec:
   containers:
    -
     image: 'rhche/centos_jdk8:latest'
     name: dev
     resources:
     limits:
       memory: 512Mi
```

3. 古いワークスペースから **image**、**volumes**、**endpoints** などの必須フィールドをコピーします。以下を参照してください。

```
    endpoints:
      - name: 8080/tcp
        port: 8080
    volumes:
      - name: m2
        containerPath: /home/user/.m2
    image: 'maven:3.6.0-jdk-11'
```

4. 必要に応じて、**memoryLimit** および **alias** 変数を変更します。フィールド **alias** は、コンポーネントの名前を設定するために使用されます。手動で設定しないと、**image** 属性フィールドの値から生成されます。

```
image: 'maven:3.6.0-jdk-11'
alias: maven3-jdk11
```

5. **RAM** コンポーネントの要件を指定するには、**memoryLimit**または**memoryRequest**のいずれか、もしくはその両方を設定します。

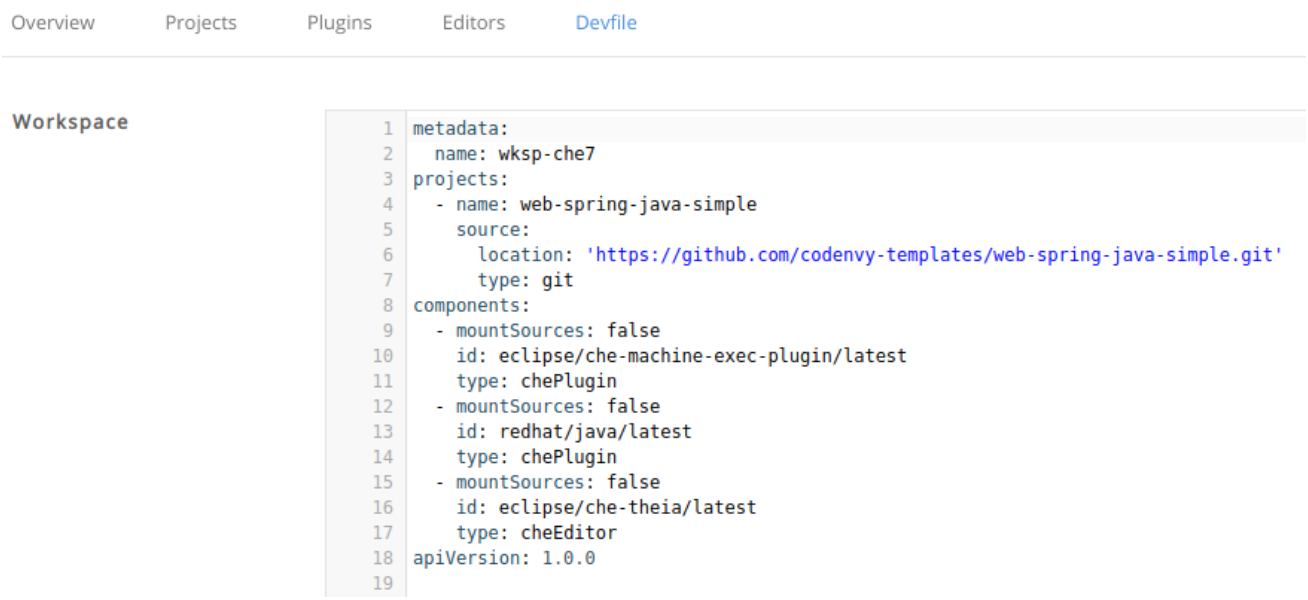
```
alias: maven3-jdk11
memoryLimit: 256M
memoryRequest: 128M
```

6. この手順を繰り返して、コンテナーイメージを追加します。

3.3.3.4. ワークスペースへのコマンドの追加

以下の図は、CodeReady Workspaces 2.6 でのワークスペース設定コマンドを示しています。

図3.1 CodeReady Workspaces 2.6 のワークスペース設定コマンドの例



前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペース。新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定について参照してください。

手順

ワークスペースにコマンドを定義するには、ワークスペースの devfile を編集します。

1. 最初のコマンドで **command** セクションを追加または編集します。元のワークスペース設定から **name** および **command** フィールドを変更します (前述の等価表を参照してください)。

```
commands:
  - name: build
    actions:
      - type: exec
        command: mvn clean install
```

2. 新規コマンドを追加するか、または他の devfile からコマンドを編集するには、以下の YAML コードを **command** セクションにコピーし、コマンドを定義します。

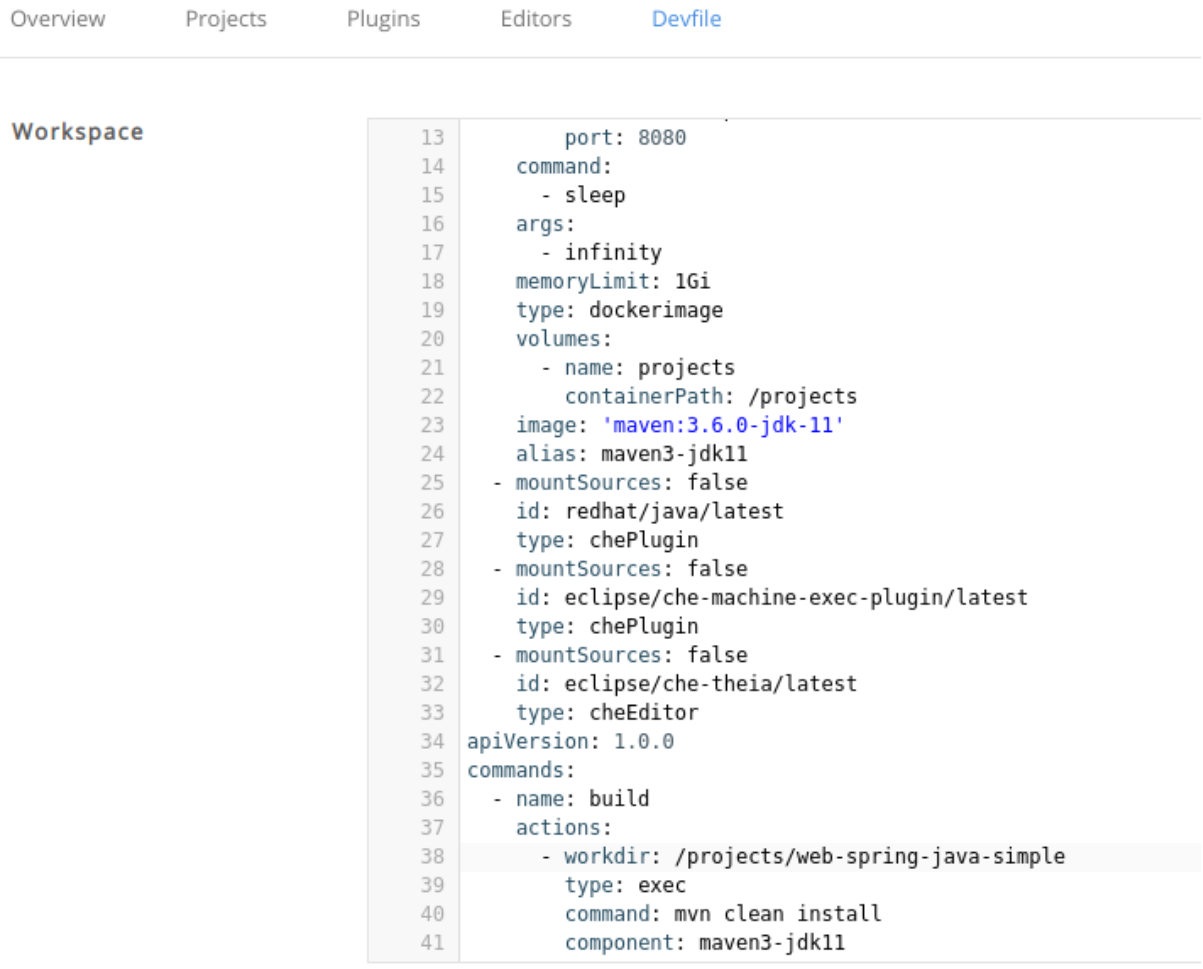
   a. 本章の概要のスクリーンショットに示されるように、元のワークスペース設定から **name** および **command** フィールドを変更します。

```
- name: build and run
  actions:
    - type: exec
      command: mvn clean install && java -jar
```

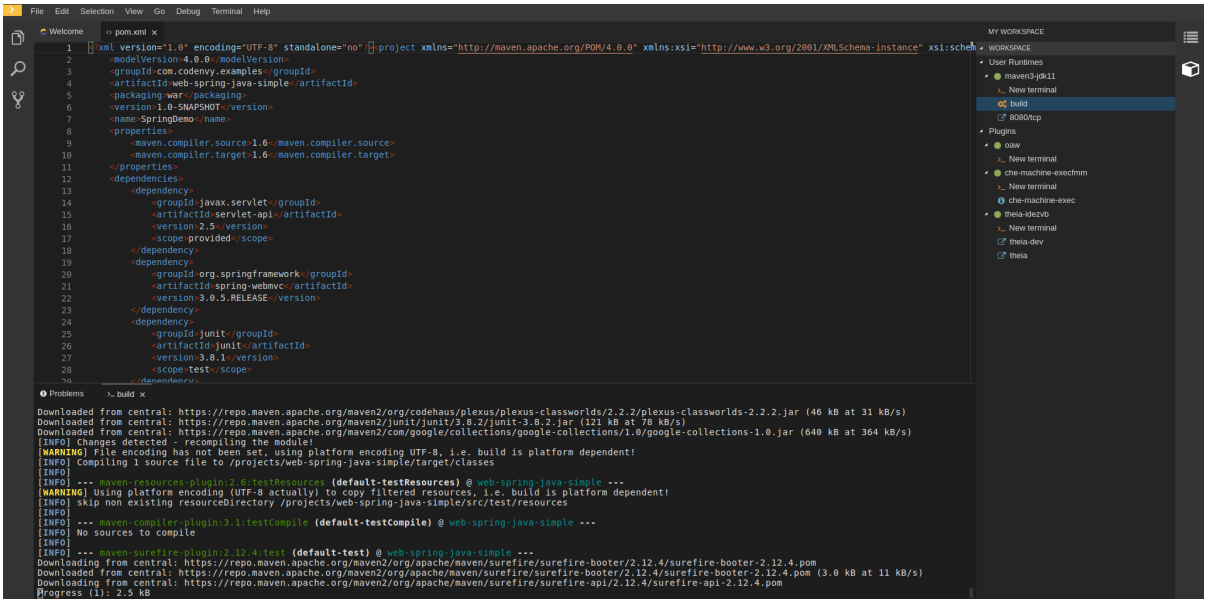
3. 必要に応じて、**component**フィールドを**actions**に追加します。これは、コマンドが実行されるコンポーネントのエイリアスを示します。

4. ステップ 2 を繰り返し、devfile にコマンドを追加します。

5. **Devfile** タブをクリックして変更を表示します。



6. 変更を保存し、新しい CodeReady Workspaces 2.6 ワークスペースを起動します。



3.4. OPENSHIFT アプリケーションのワークスペースへのインポート

本セクションでは、OpenShift アプリケーションを CodeReady Workspaces ワークスペースにインポートする方法を説明します。

デモの目的で、以下の 2 つの Pod を持つサンプル OpenShift アプリケーションを使用します。

- この nodejs-app.yaml で指定される Node.js アプリケーション
- この mongo-db.yaml で指定される MongoDB Pod

OpenShift クラスターでアプリケーションを実行するには、以下を実行します。

```
$ node=https://raw.githubusercontent.com/redhat-developer/devfile/master/samples/web-nodejs-with-db-sample/nodejs-app.yaml && \
mongo=https://raw.githubusercontent.com/redhat-developer/devfile/master/samples/web-nodejs-with-db-sample/mongo-db.yaml && \
oc  apply -f ${mongo} && \
oc  apply -f ${node}
```

CodeReady Workspaces ワークスペースでアプリケーションの新規インスタンスをデプロイするには、以下の 3つのシナリオの中から 1 つを使用します。

- ゼロからの開始: 新規 devfile の記述
- 既存ワークスペースの変更: Dashboard ユーザーインターフェースの使用
- 実行中のアプリケーションの使用: **crwctl** を使用した dev ファイルの生成

3.4.1. OpenShift アプリケーションのワークスペース devfile 定義への追加

この手順では、OpenShift アプリケーションを組み込むために CodeReady Workspaces ワークスペース devfile を定義する方法を説明します。

devfile 形式は、CodeReady Workspaces ワークスペースを定義するために使用されます。その形式は、「devfile を使用してワークスペースを移植可能にする」 セクションで説明されています。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンスでクラスターにログインしている。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- **crwctl** 管理ツールが利用可能である。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#using-the-crwctl-management-tool_crw を参照してください。

手順

1. 最も簡単な devfile を作成します。

```
apiVersion: 1.0.0
metadata:
 name: minimal-workspace (1)
```

(1) **minimal-workspace** という名前のみが指定されています。CodeReady Workspaces サーバーがこの devfile を処理すると、devfile はデフォルトのエディター (Che-Theia) とデフォルトのエディタープラグイン (ターミナルなど) のみを持つ最小の CodeReady Workspaces ワークスペースに変換されます。

2. OpenShift アプリケーションをワークスペースに追加するには、devfile を変更して **Kubernetes** コンポーネントタイプを追加します。
たとえば、NodeJS-Mongo アプリケーションを **minimal-workspace** に組み込むには、以下を実行します。

```
apiVersion: 1.0.0
metadata:
 name: minimal-workspace
components:
 - type: kubernetes
   reference: https://raw.githubusercontent.com/.../mongo-db.yaml
 - alias: nodejs-app
   type: kubernetes
   reference: https://raw.githubusercontent.com/.../nodejs-app.yaml
   entrypoints:
     - command: ['sleep'] (1)
       args: ['infinity']
```

(1) **sleep infinity** コマンドは、Node.js アプリケーションのエントリーポイントとして追加されます。このコマンドは、ワークスペースの開始フェーズでアプリケーションが起動しないようにします。この設定により、ユーザーはテストまたはデバッグ目的で必要に応じてアプリケーションを起動できます。

3. devfile にコマンドを追加して、開発者のアプリケーションのテストをより容易にします。

```
apiVersion: 1.0.0
metadata:
 name: minimal-workspace
```

```
components:
 - type: kubernetes
   reference: https://raw.githubusercontent.com/.../mongo-db.yaml
 - alias: nodejs-app
   type: kubernetes
   reference: https://raw.githubusercontent.com/.../nodejs-app.yaml
   entrypoints:
     - command: ['sleep']
       args: ['infinity']
commands:
 - name: run (1)
   actions:
     - type: exec
       component: nodejs-app
       command: cd ${CHE_PROJECTS_ROOT}/nodejs-mongo-app/EmployeeDB/ && npm
install && sed -i -- "s/localhost/mongo/g" app.js && node app.js
```

(1) devfile に追加された **run** コマンドは、コマンドパレットから Che-Theia のタスクとして利用できます。実行すると、コマンドは Node.js アプリケーションを起動します。

4. devfile を使用してワークスペースを作成し、起動します。

```
$ crwctl workspace:start --devfile <devfile-path>
```

3.4.2. Dashboard を使用した OpenShift アプリケーションの既存ワークスペースへの追加

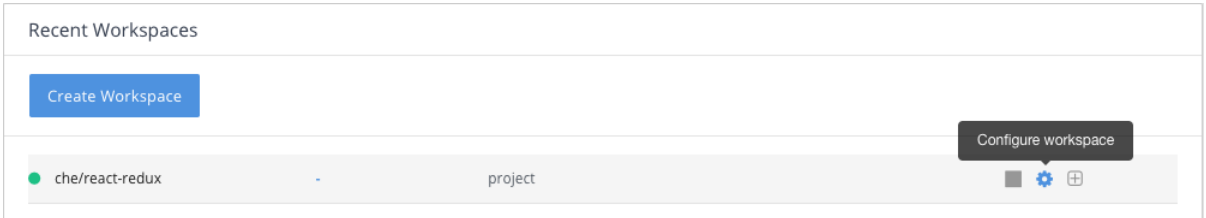
この手順では、既存のワークスペースを変更し、新たに作成された devfile を使用して OpenShift アプリケーションをインポートする方法を説明します。

前提条件

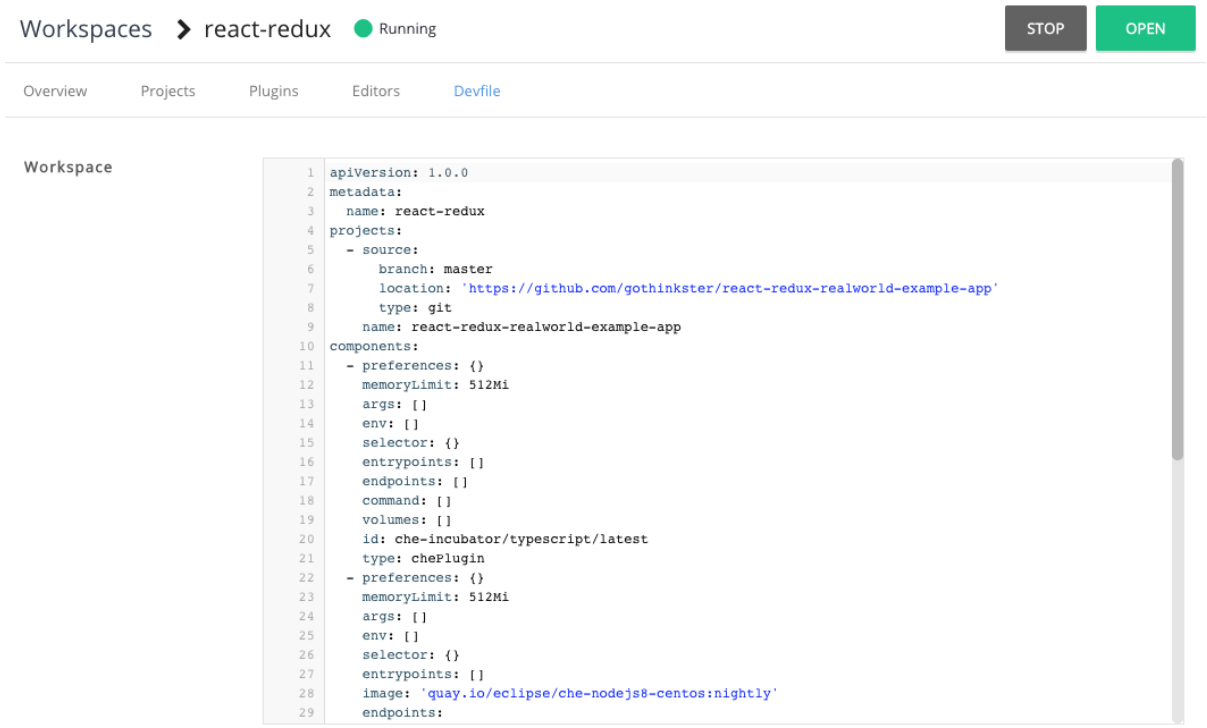
- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義される既存のワークスペース。

手順

1. ワークスペースの作成後に、**Workspace** メニューを使用してから **Configure workspace** アイコンを使用してワークスペースを管理します。



2. ワークスペースの詳細を変更するには、**Devfile** タブを使用します。ワークスペースの詳細は、このタブに devfile 形式で表示されます。



3. OpenShift コンポーネントを追加するには、ダッシュボードで **Devfile** エディターを使用します。

4. 変更を有効にするには、devfile を保存して、CodeReady Workspaces ワークスペースを再起動します。

3.4.3. 既存の OpenShift アプリケーションからの devfile の生成

この手順では、**crwctl** ツールを使用して、既存の OpenShift アプリケーションから devfile を生成する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- **crwctl** 管理ツールが利用可能である。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#using-the-crwctl-management-tool_crw を参照してください。

手順

1. devfile を生成するには、以下を使用します。

```
$ crwctl devfile:generate
```

- **crwctl devfile:generate** コマンドを使用して、**NodeJS** コンポーネントが含まれる **NodeJS-MongoDB** アプリケーションなどから devfile を生成することもできます。

  例:

```
$ crwctl devfile:generate --selector="app=nodejs"
```

```
apiVersion: 1.0.0
metadata:
  name: crwctl-generated
components:
  - type: kubernetes
    alias: app=nodejs
    referenceContent: |
      kind: List
      apiVersion: v1
      metadata:
        name: app=nodejs
      items:
        - apiVersion: apps/v1
          kind: Deployment
          metadata:
            labels:
              app: nodejs
            name: web
(...)
```

Node.js アプリケーションの YAML 定義は、**referenceContent** 属性を使用し、devfile のインラインで利用できます。

- 言語のサポートを追加するには、**--language** パラメーターを使用します。

```
$ crwctl devfile:generate --selector="app=nodejs" --language="typescript"
apiVersion: 1.0.0
metadata:
  name: crwctl-generated
components:
  - type: kubernetes
    alias: app=nodejs
    referenceContent: |
      kind: List
      apiVersion: v1
(...)
  - type: chePlugin
    alias: typescript-ls
    id: che-incubator/typescript/latest
```

2. 生成された devfile を使用して、**crwctl** で CodeReady Workspaces ワークスペースを起動します。

```
$ crwctl workspace:start --devfile=devfile.yaml
```

3.5. ワークスペースへのリモートアクセス

本セクションでは、ブラウザーの外部で CodeReady Workspaces ワークスペースにリモートでアクセスする方法を説明します。

CodeReady Workspaces ワークスペースはコンテナーとして存在し、デフォルトではブラウザーウィンドウから変更されます。さらに、CodeReady Workspaces ワークスペースと対話する方法として以下の方法を使用できます。

- OpenShift コマンドラインツール **oc** を使用して、ワークスペースコンテナーでコマンドラインを開きます。
- **oc** ツールを使用したファイルのアップロードおよびダウンロード。

3.5.1. oc を使用したワークスペースへのリモートアクセス

OpenShift コマンドラインツール (**oc**) を使用して CodeReady Workspaces ワークスペースにリモートでアクセスするには、本セクションの手順に従います。

前提条件

- **oc** バージョン 1.5.0 以降が利用できる。インストールされているバージョンの情報については、以下を実行します。

```
$ oc version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0"

...
```

手順

以下の例を参照してください。

- **workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4** は Pod の名前です。
- **crw** はプロジェクトです。

1. OpenShift プロジェクトの名前と、CodeReady Workspaces ワークスペースを実行する Pod を検索するには、以下を実行します。

```
$ oc  get pod -l che.workspace_id --all-namespaces
NAMESPACE   NAME                                              READY   STATUS    RESTARTS
AGE
crw         workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4   4/4     Running   0
6m4s
```

2. コンテナーの名前を見つけるには、以下を実行します。

```
$ NAMESPACE=crw
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
$ oc  get pod ${POD} -o custom-columns=CONTAINERS:.spec.containers[*].name
CONTAINERS
maven,che-machine-execpau,theia-ide6dj,vscode-javaw92
```

3. プロジェクト、Pod 名、およびコンテナーの名前がある場合は、'oc ' コマンドを使用してリモートシェルを開きます。

```
$ NAMESPACE=crw
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
$ CONTAINER=maven
$ oc exec -ti -n ${NAMESPACE} ${POD} -c ${CONTAINER} bash
user@workspace7b2wemdf3hx7s3ln $
```

4. コンテナーから **build** および **run** コマンドを実行します (CodeReady Workspaces ワークスペースのターミナルから実行する場合と同様)。

```
user@workspace7b2wemdf3hx7s3ln $ mvn clean install
[INFO] Scanning for projects...
(...)
```

関連資料

- **oc** の詳細は、Getting started with the CLIを参照してください。

3.5.2. コマンドラインインターフェースを使用したワークスペースへのファイルのダウンロードおよびアップロード

この手順では、「oc」ツールを使用してファイルをリモートで CodeReady Workspaces ワークスペースから/にダウンロードする方法を説明します。

前提条件

- CodeReady Workspaces の実行中のインスタンスがある。CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- 変更する予定の CodeReady Workspaces ワークスペースへのリモートアクセス手順は、「ワークスペースへのリモートアクセス」を参照してください。
- **oc** バージョン 1.5.0 以降が利用できる。インストールされているバージョンの情報については、以下を実行します。

```
$ oc version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0"

...
```

手順

以下の手順では、ユーザープロジェクトの例として **crw** を使用します。

- ワークスペースコンテナーからユーザーの現在のホームディレクトリーに **downloadme.txt** という名前のローカルファイルをダウンロードするには、CodeReady Workspaces リモートシェルで以下を実行します。

```
$ REMOTE_FILE_PATH=/projects/downloadme.txt
$ NAMESPACE=crw
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
$ CONTAINER=maven
$ oc cp ${NAMESPACE}/${POD}:${REMOTE_FILE_PATH} ~/downloadme.txt -c
${CONTAINER}
```

- **uploadme.txt** という名前のローカルファイルを **/projects** ディレクトリー内のワークスペースコンテナーにアップロードするには、以下を実行します。

```
$ LOCAL_FILE_PATH=./uploadme.txt
$ NAMESPACE=crw
```

```
$ POD=workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4
$ CONTAINER=maven
$ oc cp ${LOCAL_FILE_PATH} ${NAMESPACE}/${POD}:/projects -c ${CONTAINER}
```

前述の手順を使用すると、ユーザーはディレクトリーをダウンロードし、アップロードすることもできます。

関連資料

- **oc** の詳細は、Getting started with the CLIを参照してください。

3.6. コードサンプルからのワークスペースの作成

すべてのスタックには、スタックの devfile で定義されるサンプルコードベースが含まれています。本セクションでは、3 つの手順に従ってこのコードサンプルからワークスペースを作成する方法を説明します。

1. ユーザーダッシュボードからワークスペースを作成します。
   a. Get Started ビューの使用。
   b. カスタム Workspace ビューの使用。
2. ワークスペースの設定を変更してコードサンプルを追加します。
3. ユーザーダッシュボードからの既存ワークスペースの実行。

devfile についての詳細は、「devfile を使用したワークスペースの設定」 を参照してください。

3.6.1. ユーザーダッシュボードの「Get Started」(スタート) ビューからのワークスペースの作成

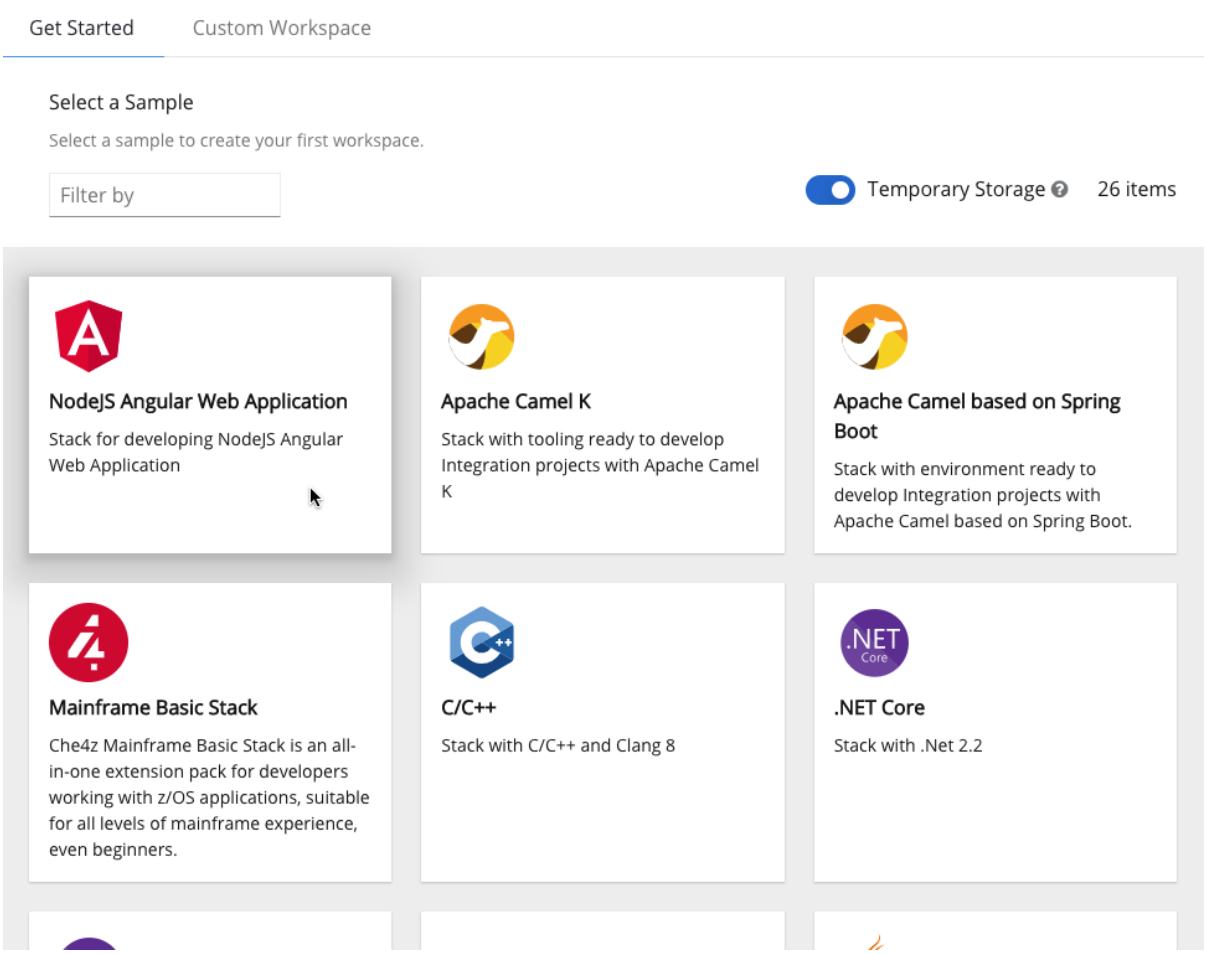
本セクションでは、User Dashboard からワークスペースを作成する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。

手順

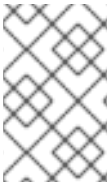
1. CodeReady Workspaces Dashboard に移動します。「Dashboard を使用した CodeReady Workspaces のナビゲーション」 を参照してください。
2. 左側のナビゲーションパネルで **Get Started** に移動します。
3. **Get Started** タブをクリックします。
4. ギャラリーには、プロジェクトのビルドおよび実行に使用できるサンプルの一覧があります。



リソース制限の変更

メモリー要件の変更は、devfile からのみ実行できます。「既存ワークスペースの設定変更」 を参照してください。

5. ワークスペースを起動します。選択したスタックカードをクリックします。



新規ワークスペース名

ワークスペース名は、スタックの基礎となる devfile に基づいて自動生成できます。生成される名前は、常にプレフィックスおよび 4 つのランダムな文字としての devfile **metadata.generateName** のプロパティーで構成されます。

3.6.2. User Dashboard のカスタム Workspace ビューからのワークスペースの作成

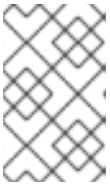
本セクションでは、User Dashboard からワークスペースを作成する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。

手順

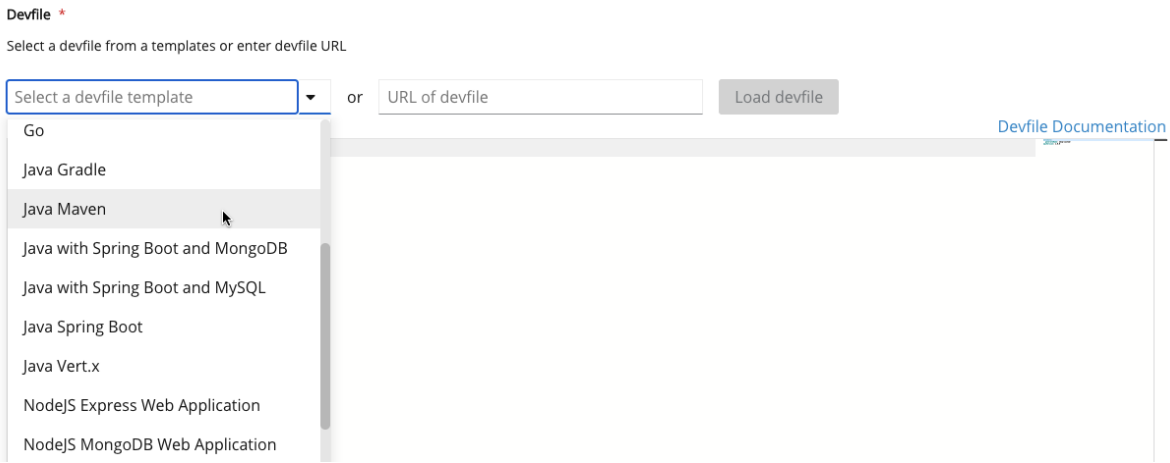
1. CodeReady Workspaces Dashboard に移動します。「Dashboard を使用した CodeReady Workspaces のナビゲーション」 を参照してください。
2. 左側のナビゲーションパネルで **Get Started** に移動します。
3. **Custom Workspace** タブをクリックします。
4. ワークスペースの **Name** を定義します。

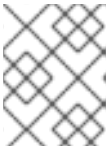


新規ワークスペース名

ワークスペース名は、スタックの基礎となる devfile に基づいて自動生成できます。生成される名前は、常にプレフィックスおよび 4 つのランダムな文字としての devfile **metadata.generateName** のプロパティーで構成されます。

5. **Devfile** セクションで、プロジェクトのビルドおよび実行に使用される devfile テンプレートを選択します。





リソース制限の変更

メモリー要件の変更は、devfile からのみ実行できます。「既存ワークスペースの設定変更」 を参照してください。

6. ワークスペースを起動します。フォームの下部にある **Create & Open** ボタンをクリックします。

3.6.3. 既存ワークスペースの設定変更

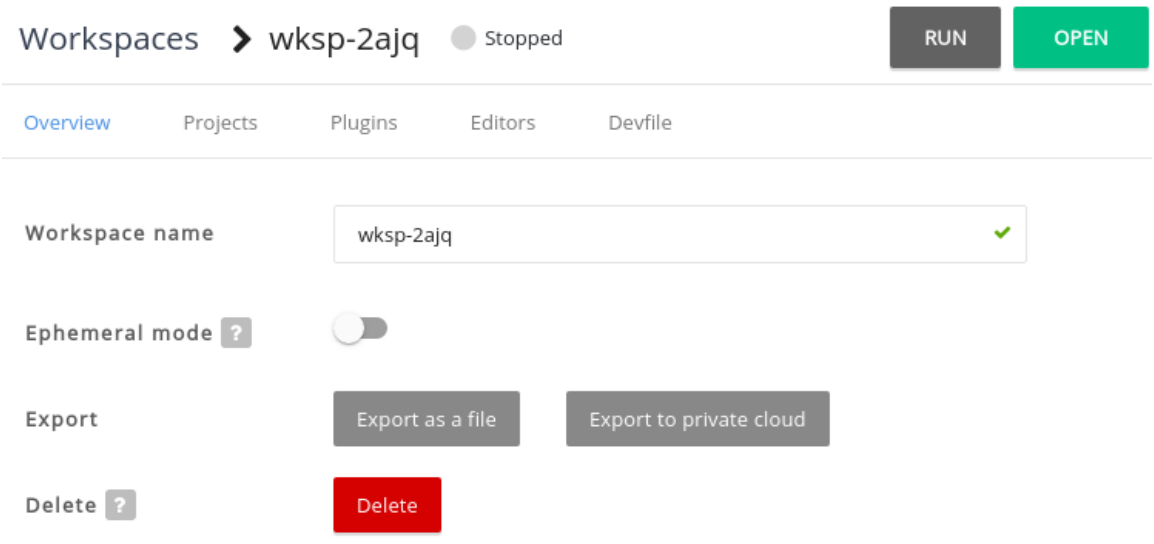
本セクションでは、User Dashboard から既存のワークスペースの設定を変更する方法を説明します。

前提条件

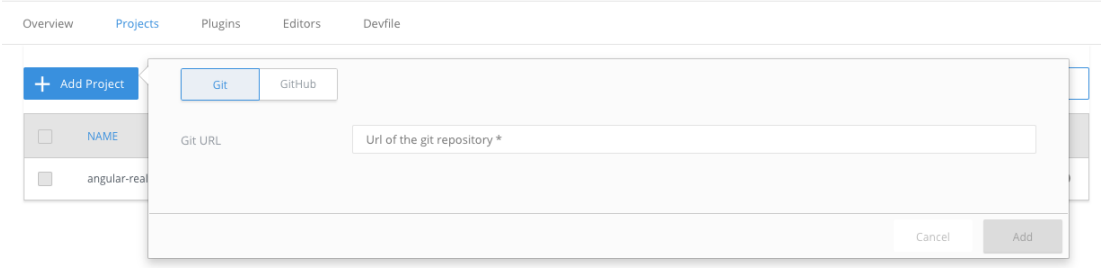
- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義される既存のワークスペース。

手順

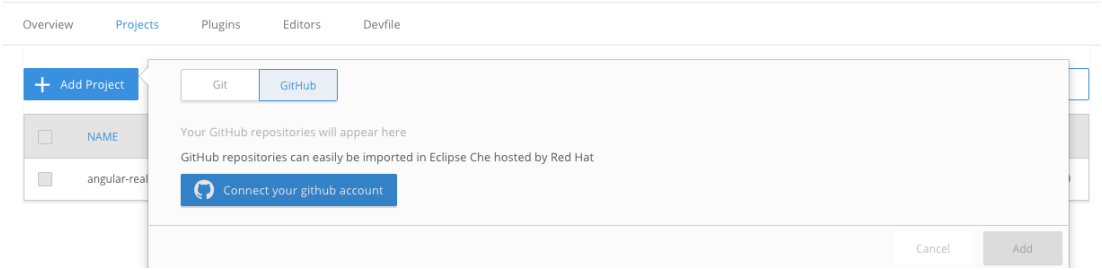
1. CodeReady Workspaces Dashboard に移動します。「Dashboard を使用した CodeReady Workspaces のナビゲーション」 を参照してください。
2. 左側のナビゲーションパネルで **Workspaces** に移動します。
3. ワークスペースの名前をクリックして、設定の概要ページに移動します。
4. **Overview** タブをクリックし、以下の操作を実行します。
   - **Workspace name** を変更します。
   - **Storage Type** を選択します。
   - ワークスペース設定を、ファイルまたはプライベートクラウドに **Export** (エクスポート) します。
   - ワークスペースを**Delete** (削除) します。



5. **Projects** セクションで、ワークスペースで統合するプロジェクトを選択します。
   a. **Add Project** ボタンをクリックして、以下のいずれかを実行します。
      i. プロジェクトの Git リポジトリー URL を入力し、ワークスペースに統合します。



      ii. GitHub アカウントに接続し、統合するプロジェクトを選択します。



   b. **追加** ボタンをクリックします。

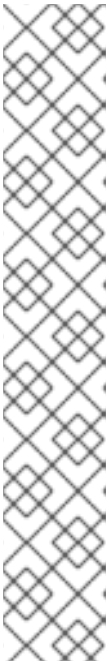
6. **Plugins** セクションで、ワークスペースに統合するプラグインを選択します。

例

汎用の Java ベースのスタックで開始し、Node.js または Python のサポートを追加します。

7. **Editors** セクションで、ワークスペースに統合するエディターを選択します。CodeReady Workspaces 2.6 エディターは Che-Theia をベースにしています。

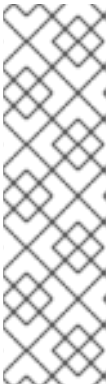
8. **Devfile** タブで、ワークスペースの YAML 設定を編集します。「devfile を使用してワークスペースを移植可能にする」 を参照してください。



例: コマンドの追加

Workspace

```
          -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true
          -Xms20m -Djava.security.egd=file:/dev/./urandom
        name: JAVA_TOOL_OPTIONS
      - value: '${echo ${0}}\$'
        name: PS1
      - value: /home/user
        name: HOME
apiVersion: 1.0.0
commands:
  - name: build the project
    actions:
      - type: exec
        command: 'cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn clean install'
        component: maven
  - name: run the services
    actions:
      - type: exec
        command: >-
          cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn spring-boot:run
          -DskipTests
        component: maven
  - name: run and debug the services
    actions:
      - type: exec
        command: >-
          cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn spring-boot:run
          -DskipTests -Drun.jvmArguments="-Xdebug
          -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005"
        component: maven
```



例: プロジェクトの追加

プロジェクトをワークスペースに追加するには、以下のセクションを追加または編集します。

```
projects:
  - name: che
    source:
      type: git
      location: 'https://github.com/eclipse/che.git'
```

3.6.4. User Dashboard からの既存ワークスペースの実行

本セクションでは、User Dashboard から既存のワークスペースを実行する方法を説明します。

3.6.4.1. Run ボタンを使用した User Dashboard からの既存ワークスペースの実行

本セクションでは、**Run** ボタンを使用して User Dashboard から既存のワークスペースを実行する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義される既存のワークスペース。

手順

1. CodeReady Workspaces Dashboard に移動します。「Dashboard を使用した CodeReady Workspaces のナビゲーション」 を参照してください。
2. 左側のナビゲーションパネルで **Workspaces** に移動します。

3. 実行していないワークスペースの名前をクリックし、概要ページに移動します。
4. ページ右上の **Run** ボタンをクリックします。
5. ワークスペースが起動します。
6. ブラウザーはワークスペースに移動 **しません**。

3.6.4.2. Open ボタンを使用した User Dashboard からの既存ワークスペースの実行

本セクションでは、**Open** ボタンを使用して、User Dashboard から既存のワークスペースを実行する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義される既存のワークスペース。

手順

1. CodeReady Workspaces Dashboard に移動します。「Dashboard を使用した CodeReady Workspaces のナビゲーション」 を参照してください。
2. 左側のナビゲーションパネルで **Workspaces** に移動します。
3. 実行していないワークスペースの名前をクリックし、概要ページに移動します。
4. ページ右上にある **Open** ボタンをクリックします。
5. ワークスペースが起動します。
6. ブラウザーはワークスペースに移動します。

3.6.4.3. Recent Workspaces を使用した User Dashboard からの既存ワークスペースの実行

本セクションでは、Recent Workspaces を使用して、User Dashboard から既存のワークスペースを実行する方法を説明します。

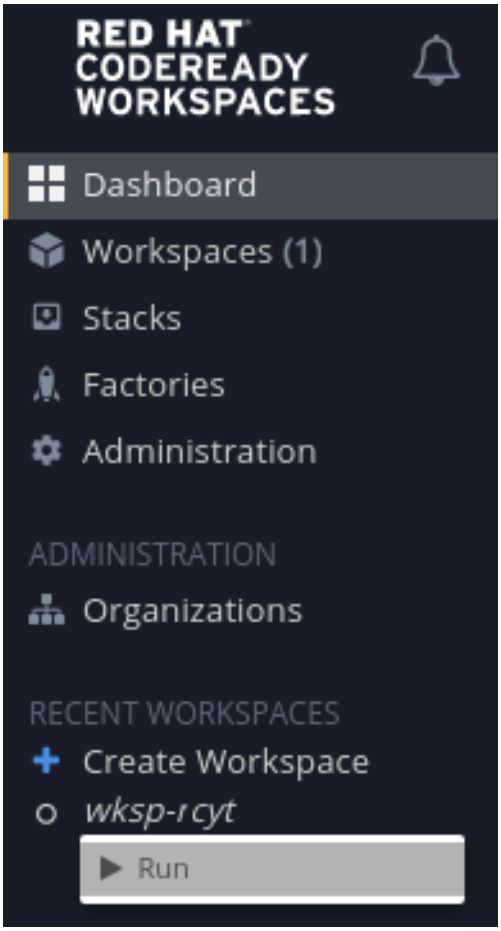
前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義される既存のワークスペース。

手順

1. CodeReady Workspaces Dashboard に移動します。「Dashboard を使用した CodeReady Workspaces のナビゲーション」 を参照してください。

2. 左側のナビゲーションパネルの **Recent Workspaces** セクションで、実行していないワークスペースの名前を右クリックし、コンテキストメニューで **Run** をクリックしてこれを起動します。



3.7. プロジェクトのソースコードをインポートしてワークスペースを作成する

本セクションでは、既存のコードベースを編集するために新規ワークスペースを作成する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義する開発環境に関連するプラグインが含まれる既存のワークスペース。

これは、ワークスペースを起動する **前** に 2 つの方法で実行できます。

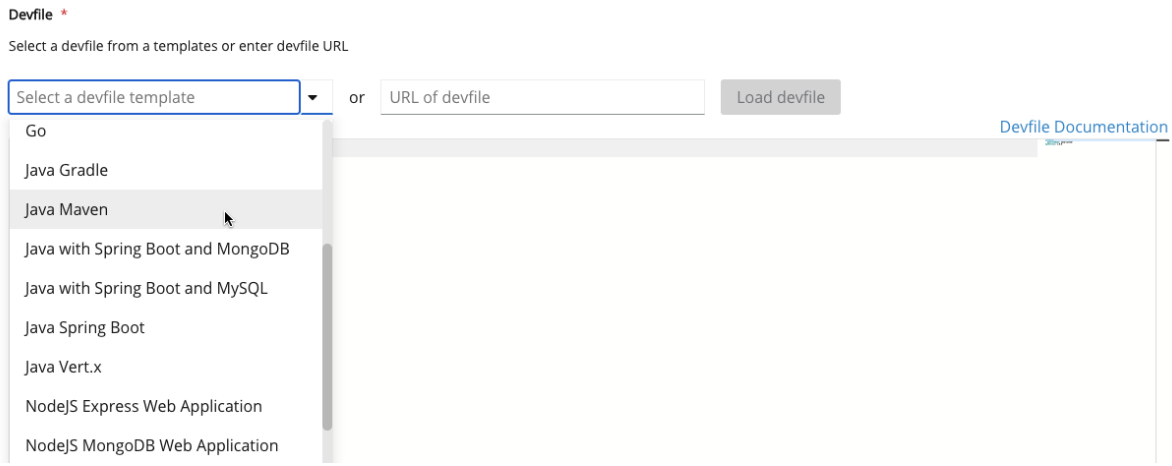
- Dashboard からサンプルを選択し、devfile をプロジェクトに含めるように変更します。
- 「devfile を使用したワークスペースの設定」

新規ワークスペースを作成して既存のコードベースを編集するには、ワークスペースの起動 **後** に以下の 3 つのメソッドのいずれかを使用します。

- Dashboard から既存のワークスペースへのインポート
- **git clone** コマンドを使用して実行中のワークスペースにインポート
- ターミナルでの **git clone** の使用による実行中のワークスペースへのインポート

3.7.1. Dashboard からサンプルを選択し、devfile をプロジェクトに含めるように変更します。

- 左側のナビゲーションパネルで **Get Started** に移動します。
- 選択されていない場合は、**Custom Workspace** タブをクリックします。
- **Devfile** セクションで、プロジェクトのビルドおよび実行に使用される devfile テンプレートを選択します。



- **Devfile エディター** で **projects** セクションを更新します。

例: プロジェクトの追加

プロジェクトをワークスペースに追加するには、以下のセクションを追加または編集します。

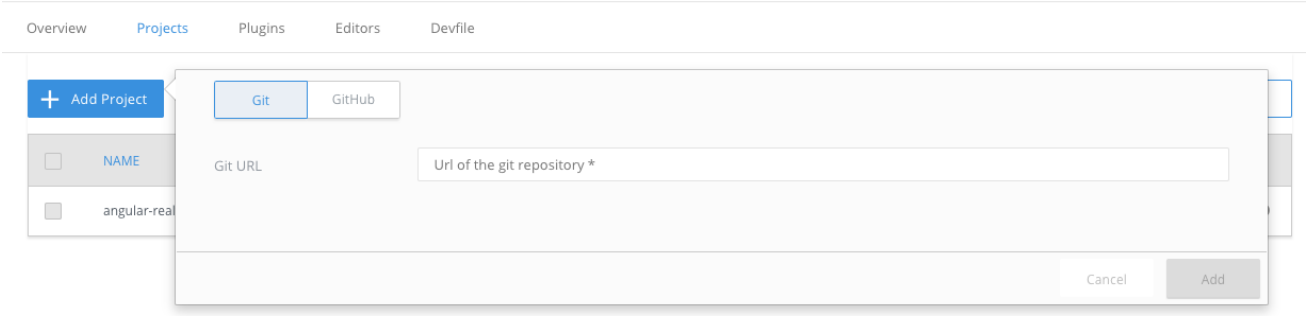
```
projects:
  - name: che
    source:
      type: git
      location: 'https://github.com/eclipse/che.git'
```

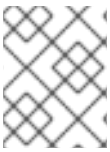
「devfile リファレンス」 を参照してください。

- ワークスペースを開くには、**Create & Open** ボタンをクリックします。

3.7.2. Dashboard から既存ワークスペースへのインポート

1. プロジェクトをインポートします。**Dashboard** を使用してプロジェクトをインポートする方法は 2 つ以上あります。
   - **Dashboard** で **Workspace** を選択し、その名前をクリックしてワークスペースを選択します。これにより、ワークスペースの **Overview** タブにリンクされます。
   - または、ギアアイコンを使用します。これは、独自の YAML 設定を入力できる **Devfile** タブにリンクします。
2. **Projects** タブをクリックします。
3. **Add Project** をクリックします。次に、リポジトリー Git URL または GitHub からプロジェクトをインポートできます。





注記

プロジェクトを実行中でないワークスペースに追加できますが、これを削除するにはワークスペースを起動する必要があります。

3.7.2.1. プロジェクトのインポート後のコマンドの編集

ワークスペースにプロジェクトを追加したら、これにコマンドを追加できます。プロジェクトにコマンドを追加すると、ブラウザーでアプリケーションを実行し、デバッグし、起動できます。

プロジェクトにコマンドを追加するには、以下を実行します。

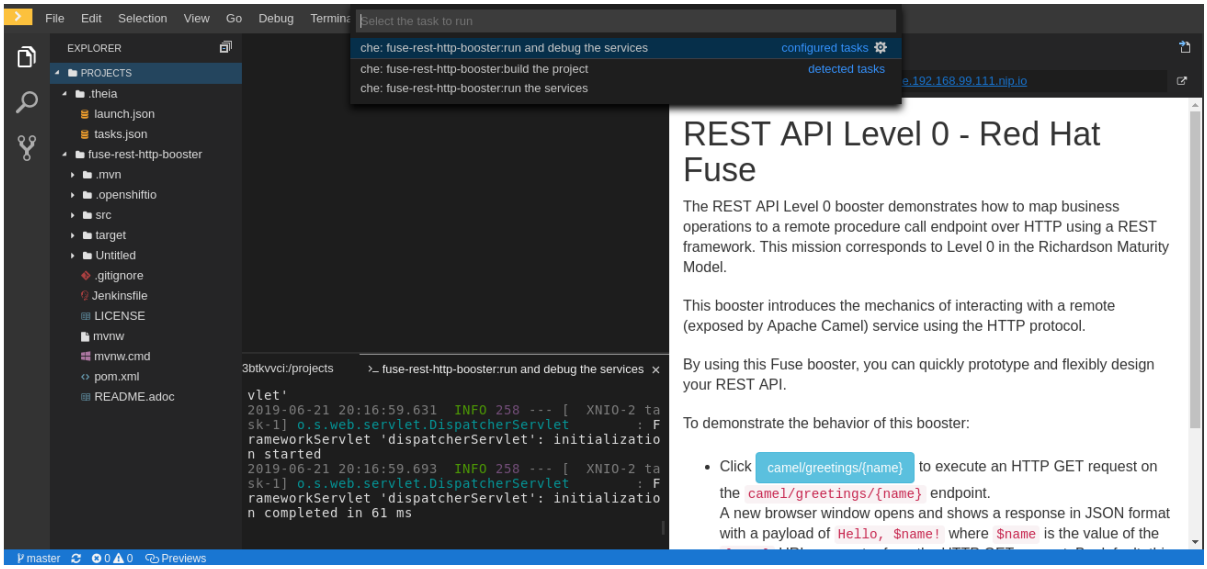
1. **Dashboard** でワークスペース設定を開き、**Devfile** タブを選択します。

Workspace

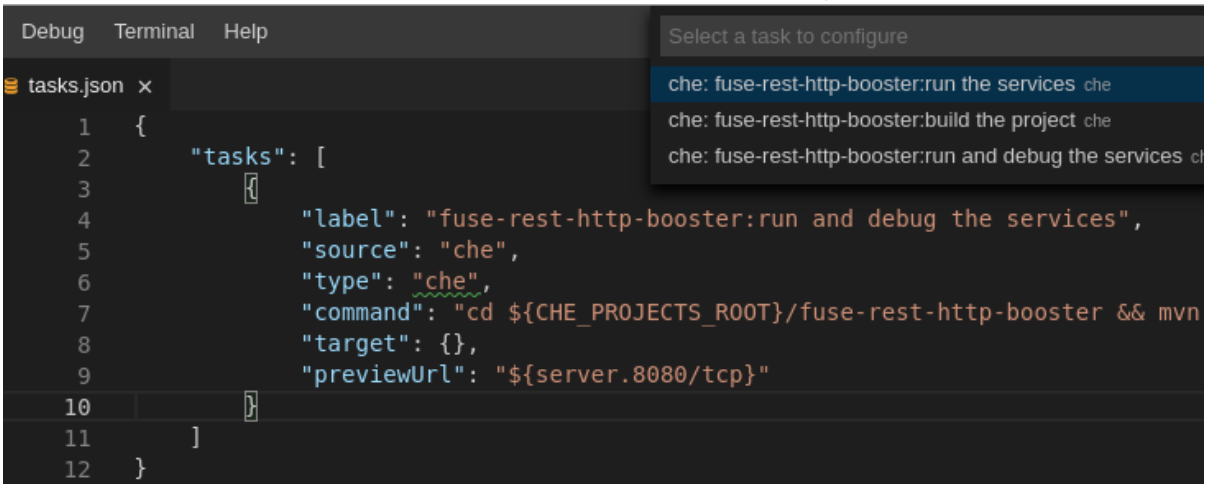
```
          -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true
          -Xms20m -Djava.security.egd=file:/dev/./urandom
        name: JAVA_TOOL_OPTIONS
      - value: '$(echo ${0})\$'
        name: PS1
      - value: /home/user
        name: HOME
apiVersion: 1.0.0
commands:
  - name: build the project
    actions:
      - type: exec
        command: 'cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn clean install'
        component: maven
  - name: run the services
    actions:
      - type: exec
        command: >-
          cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn spring-boot:run
          -DskipTests
        component: maven
  - name: run and debug the services
    actions:
      - type: exec
        command: >-
          cd ${CHE_PROJECTS_ROOT}/fuse-rest-http-booster && mvn spring-boot:run
          -DskipTests -Drun.jvmArguments="-Xdebug
          -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005"
        component: maven
```

2. ワークスペースを開きます。

3. コマンドを実行するには、メインメニューから **Terminal** > **Run Task** を選択します。



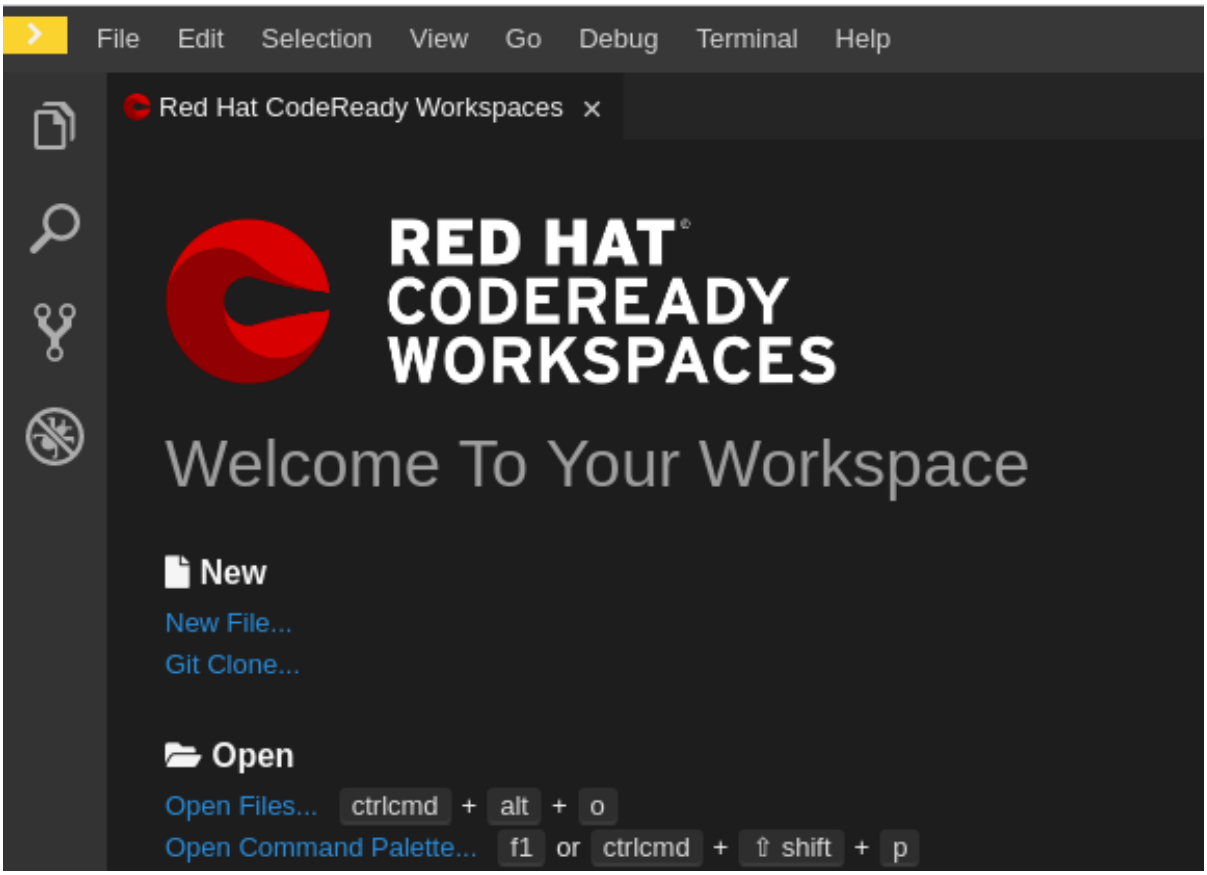
4. コマンドを設定するには、メインメニューから **Terminal** > **Configure Tasks** を選択します。



3.7.3. Git: Clone コマンドを使用した実行中のワークスペースへのインポート

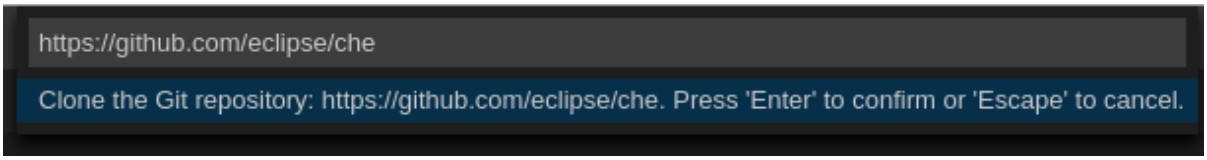
Git: Clone コマンドを使用して実行中のワークスペースにインポートするには、以下を実行します。

1. ワークスペースを起動してから、コマンドパレットまたは Welcome 画面で **Git: Clone** コマンドを使用して、プロジェクトを実行中のワークスペースにインポートします。



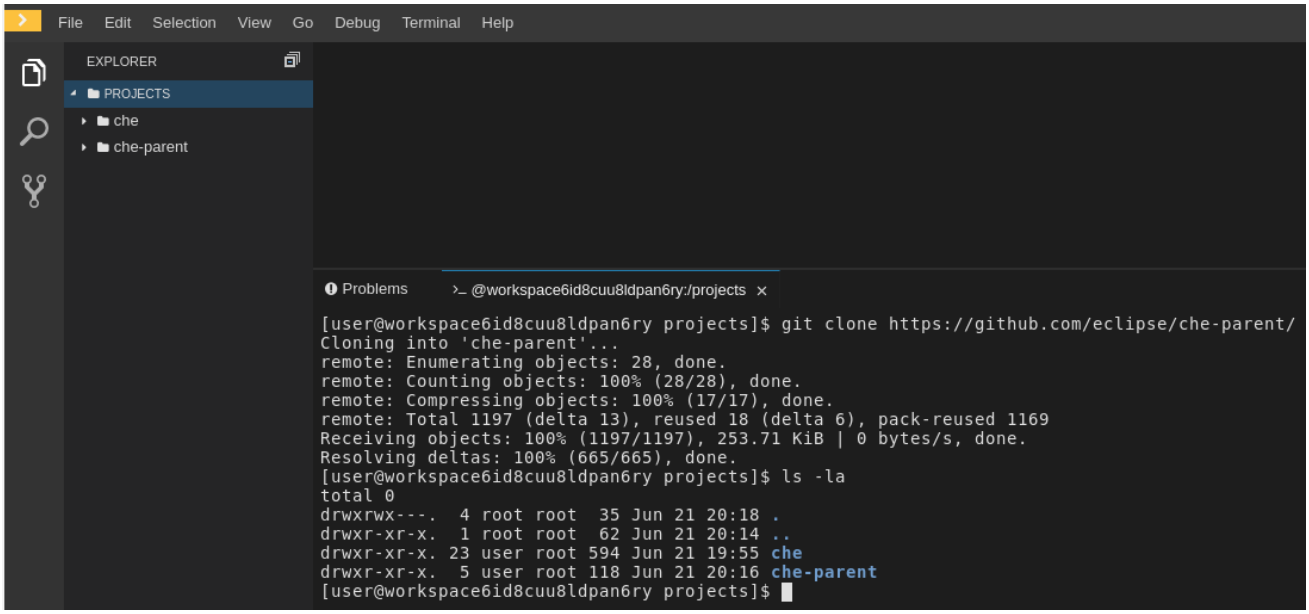
2. **F1** または **CTRL-SHIFT-P** を使用してコマンドパレットを開くか、または Welcome 画面のリンクからコマンドを開きます。

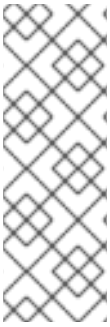
3. クローンを作成するプロジェクトのパスを入力します。



3.7.4. ターミナルで git clone を使用した実行中のワークスペースへのインポート

上記の方法に加え、ワークスペースを起動し、**ターミナル** を開き、**git clone** を入力してコードをプルすることもできます。





注記

ターミナルでワークスペースプロジェクトをインポートまたは削除してもワークスペース設定が更新されず、その変更は Dashboard の **Project** タブおよび **Devfile** タブに反映されません。

同様に、**Dashboard** を使用してプロジェクトを追加してから **rm -fr myproject** でプロジェクトを削除すると、プロジェクトは 、**Projects** または **Devfile** タブに依然として表示される可能性があります。

3.8. シークレットをファイルまたは環境変数としてワークスペースコンテナーにマウントする

シークレットは、ユーザー名、パスワード、認証トークン、設定などの機密データを暗号化された形式で保存する OpenShift オブジェクトです。

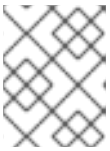
ユーザーは、機密データが含まれるシークレットをワークスペースコンテナーにマウントできます。これにより、新規に作成されるワークスペースごとに、シークレットから保存されたデータが自動的に再適用されます。その結果、ユーザーはこれらの認証情報と設定を手動で指定する必要はありません。

以下のセクションでは、OpenShift シークレットをワークスペースコンテナーに自動的にマウントし、以下のようなコンポーネントの永続的なマウントポイントを作成する方法を説明します。

- Maven 設定（**settings.xml** ファイル）
- SSH キーペア
- AWS 認証トークン
- Git 認証情報ストアファイル

OpenShift シークレットは以下のようにワークスペースコンテナーにマウントできます。

- ファイル: これにより、Maven 機能を持つすべての新規ワークスペースに適用される自動的にマウントされた Maven 設定が作成されます。
- 環境変数: これは、自動認証に SSH キーペアおよい AWS 認証トークンを使用します。



注記

SSH キーペアはファイルとしてマウントすることもできますが、この形式は主に Maven 設定を設定することを目的として使用されます。

マウントプロセスでは標準の OpenShift マウントメカニズムを使用しますが、シークレットを必要な CodeReady Workspaces ワークスペースコンテナーに適切にバインドするために追加のアノテーションとラベルが必要です。

3.8.1. シークレットをファイルとしてワークスペースコンテナーにマウントする

警告

OpenShift 3.11 では、ファイルとしてマウントされるシークレットは devfile に定義されたボリュームマウントを上書きします。

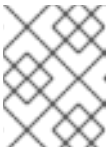
本セクションでは、CodeReady Workspaces の単一ワークスペースまたは複数ワークスペースコンテナーで、ユーザーのプロジェクトからシークレットをファイルとしてマウントする方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。

手順

1. CodeReady Workspaces ワークスペースが作成される OpenShift プロジェクトで新規の OpenShift シークレットを作成します。
   - 作成されるシークレットのラベルは、CodeReady Workspaces の **che.workspace.provision.secret.labels** プロパティーに設定されるラベルのセットと一致する必要があります。デフォルトのラベルは以下の通りです。
   - **app.kubernetes.io/part-of: che.eclipse.org**
   - **app.kubernetes.io/component: workspace-secret**:



注記

以下の例では、Red Hat CodeReady Workspaces のバージョン 2.1 と 2.2 での **target-container** アノテーションの使用法の違いについて説明します。

例:

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
```

```
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspace-secret
...
```

アノテーションは指定のシークレットがファイルとしてマウントされていることを示し、オプションで、シークレットがマウントされるコンテナーの名前を指定します。target-container アノテーションがない場合、シークレットは CodeReady Workspaces ワークスペースのすべてのユーザーコンテナーにマウントされますが、これは **CodeReady Workspaces バージョン 2.1 についてのみ** 適用されます。

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  annotations:
    che.eclipse.org/target-container: maven
    che.eclipse.org/mount-path: {prod-home}/.m2/
    che.eclipse.org/mount-as: file
  labels:
...
```

CodeReady Workspaces バージョン 2.2 以降、target-container アノテーションは非推奨となり、ブール値が含まれる **automount-workspace-secret** アノテーションが導入されました。これは、devfile で上書きされる機能を使用して、デフォルトのシークレットのマウント動作を定義することを目的としています。**true** の値により、すべてのワークスペースコンテナーへの自動マウントが有効になります。これとは対照的に、**false** の値は、**automountWorkspaceSecrets:true** プロパティーを使用して devfile コンポーネントで明示的に要求されるまでマウントプロセスを無効にします。

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  annotations:
    che.eclipse.org/automount-workspace-secret: true
    che.eclipse.org/mount-path: {prod-home}/.m2/
    che.eclipse.org/mount-as: file
  labels:
...
```

OpenShift シークレットのデータには複数の項目が含まれる可能性があり、その名前はコンテナーにマウントされる必要なファイル名と一致する必要があります。

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspace-secret
  annotations:
    che.eclipse.org/automount-workspace-secret: true
    che.eclipse.org/mount-path: {prod-home}/.m2/
```

```
    che.eclipse.org/mount-as: file
data:
  settings.xml: <base64 encoded data content here>
```

これにより、**settings.xml** という名前のファイルが、すべてのワークスペースコンテナーの **/home/jboss/.m2/** パスにマウントされます。

secret-s マウントパスは、devfile を使用するワークスペースの特定のコンポーネントに対して上書きできます。マウントパスを変更するには、devfile のコンポーネントで、上書きされたシークレット名と必要なマウントパスと一致する名前を使用して追加のボリュームを宣言する必要があります。

```
apiVersion: 1.0.0
metadata:
  ...
components:
 - type: dockerimage
   alias: maven
   image: maven:3.11
   volumes:
     - name: <secret-name>
       containerPath: /my/new/path
   ...
```

このような上書きの場合、コンポーネントはそれらに属するコンテナーを区別するためにエイリアスを宣言し、上書きパスをこれらのコンテナーについてのみに適用する必要があります。

3.8.2. シークレットを環境変数としてワークスペースコンテナーにマウントする

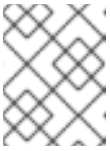
以下のセクションでは、OpenShift シークレットを環境変数として、ユーザーのプロジェクトから CodeReady Workspaces の単一ワークスペースまたは複数ワークスペースコンテナーにマウントする方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。

手順

1. CodeReady Workspaces ワークスペースが作成される k8s プロジェクトで新規の OpenShift シークレットを作成します。
   - 作成されるシークレットのラベルは、CodeReady Workspaces の **che.workspace.provision.secret.labels** プロパティーに設定されるラベルのセットと一致する必要があります。デフォルトでは、これは 2 つのラベルのセットになります。
   - **app.kubernetes.io/part-of: che.eclipse.org**
   - **app.kubernetes.io/component: workspace-secret**:



注記

以下の例では、Red Hat CodeReady Workspaces のバージョン 2.1 と 2.2 での **target-container** アノテーションの使用法の違いについて説明します。

例:

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

アノテーションは、指定のシークレットが環境変数としてマウントされていることを示す必要があり、オプションでこのマウントが適用されるコンテナー名を指定します。target-container アノテーションが定義されていない場合、シークレットは CodeReady Workspaces ワークスペースのすべてのユーザーコンテナーにマウントされますが、これは **CodeReady Workspaces バージョン 2.1 についてのみ** 適用されます。

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

これにより、**FOO_ENV** という名前の環境変数と値 **myvalue** が、**maven** という名前のコンテナーにプロビジョニングされます。

CodeReady Workspaces バージョン 2.2 以降、target-container アノテーションは非推奨となり、ブール値が含まれる **automount-workspace-secret** アノテーションが導入されました。これは、devfile で上書きされる機能を使用して、デフォルトのシークレットのマウント動作を定義することを目的としています。**true** の値により、すべてのワークスペースコンテナーへの自動マウントが有効になります。これとは対照的に、**false** の値は、**automountWorkspaceSecrets:true** プロパティーを使用して devfile コンポーネントで明示的に要求されるまでマウントプロセスを無効にします。

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  annotations:
    che.eclipse.org/automount-workspace-secret: true
    che.eclipse.org/env-name: FOO_ENV
    che.eclipse.org/mount-as: env
  labels:
```

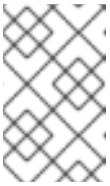
```
  ...
data:
  mykey: myvalue
```

これにより、**FOO_ENV** という名前の環境変数、および値 **myvalue** がすべてのワークスペースコンテナーにプロビジョニングされます。

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

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  annotations:
    che.eclipse.org/automount-workspace-secret: true
    che.eclipse.org/mount-as: env
    che.eclipse.org/mykey_env-name: FOO_ENV
    che.eclipse.org/otherkey_env-name: OTHER_ENV
  labels:
    ...
data:
  mykey: myvalue
  otherkey: othervalue
```

これにより、**FOO_ENV**、**OTHER_ENV** という名前の、**myvalue** および **othervalue** の値を持つ 2 つの環境変数がワークスペースコンテナーにプロビジョニングされます。



注記

OpenShift シークレットのアノテーション名の最大長さは 63 文字です。ここで、9 文字は、/ で終わるプレフィックス用に予約されます。これは、シークレットに使用できるキーの最大長さの制限として機能します。

3.8.3. git 認証情報のワークスペースコンテナーへのマウント

本セクションでは、git 認証情報ストアをユーザーのプロジェクトから、CodeReady Workspaces の単一ワークスペースまたは複数ワークスペースコンテナーのファイルにシークレットとしてマウントする方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。

手順

1. git 認証情報ファイルを、https://git-scm.com/docs/git-credential-store#_storage_format の形式で作成します。
2. ファイルのコンテンツを base64 形式にエンコードします。
3. CodeReady Workspaces ワークスペースが作成される OpenShift プロジェクトで新規の OpenShift シークレットを作成します。

- 作成されるシークレットのラベルは、CodeReady Workspaces の **che.workspace.provision.secret.labels** プロパティーに設定されるラベルのセットと一致する必要があります。デフォルトのラベルは以下の通りです。

- **app.kubernetes.io/part-of: che.eclipse.org**

- **app.kubernetes.io/component: workspace-secret**:

3.8.4. シークレットをワークスペースコンテナーにマウントするプロセスでのアノテーションの使用

Kubernetes アノテーションとラベルは、任意の非識別メタデータを OpenShift ネイティブオブジェクトに割り当てるために、ライブラリー、ツール、およびその他のクライアントで使用されるツールです。

ラベルはオブジェクトを選択し、それらを特定の条件を満たすコレクションに接続します。ここで、アノテーションは OpenShift オブジェクトによって内部で使用されない非識別情報に使用されます。

本セクションでは、CodeReady Workspaces ワークスペースでの OpenShift シークレットのマウントプロセスで使用される OpenShift アノテーションの値について説明します。

アノテーションには、適切なマウント設定を特定するのに役立つアイテムが含まれている必要があります。これらのアイテムは以下の通りです。

- **che.eclipse.org/target-container**: **バージョン 2.1 まで有効**です。マウントするコンテナーの名前。名前が定義されていない場合、シークレットは CodeReady Workspaces ワークスペースのすべてのユーザーのコンテナーにマウントされます。

- **che.eclipse.org/automount-workspace-secret** ：**バージョン 2.2. で導入** されました。メインマウントセレクター。**true** に設定すると、シークレットは CodeReady Workspaces ワークスペースのすべてのユーザーのコンテナーにマウントされます。**false** に設定すると、シークレットはデフォルトでコンテナーにマウントされません。この属性の値は、ワークスペースのオーナーにより多くの柔軟性を提供する **automountWorkspaceSecrets** ブール値プロパティーを使用して devfile コンポーネントで上書きできます。このプロパティーを使用するには、これを使用するコンポーネントについて **alias** を定義する必要があります。

- **che.eclipse.org/env-name**: シークレットのマウントに使用される環境変数の名前。

- **che.eclipse.org/mount-as**: このアイテムは、シークレットが環境変数またはファイルとしてマウントされるかどうかを記述します。オプション: **env** または **file**。

- **che.eclipse.org/<mykeyName>-env-name: FOO_ENV**: データに複数のアイテムが含まれる場合に使用される環境変数の名前。**mykeyName** は例として使用されます。

第4章 開発者環境のカスタマイズ

Red Hat CodeReady Workspaces は、拡張可能かつカスタマイズ可能な開発者のワークスペースプラットフォームです。

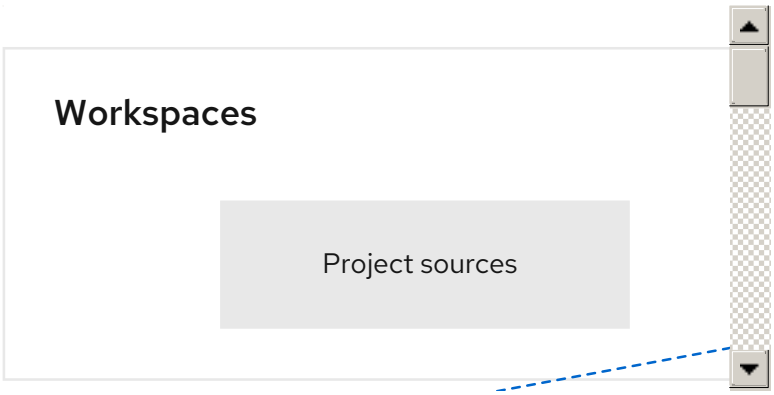
Red Hat CodeReady Workspaces は、以下の 3 つの方法で拡張できます。

- **代替 IDE** は、Red Hat CodeReady Workspaces に特化したツールを提供します。たとえば、データ分析用の Jupyter ノートブックなどです。代替 IDE は Eclipse Theia またはその他の IDE（Web またはデスクトップベース) をベースに使用できます。Red Hat CodeReady Workspaces のデフォルト IDE は Che-Theia です。

- **Che-Theia プラグイン**は各種機能を Che-Theia IDE に追加します。これらは、Visual Studio Code と互換性のあるプラグイン API に依存します。プラグインは IDE 自体から分離されます。それらはファイルとしてパッケージ化されるか、またはコンテナーとしてパッケージ化でき、独自の依存関係を提供できます。

- **Stacks** は、異なる開発者の担当者に対応する、専用のツールセットを含む事前に設定された CodeReady Workspaces ワークスペースです。たとえば、該当する目的に必要なツールのみを含むテスター用のワークベンチを事前に設定できます。

図4.1 CodeReady Workspaces の拡張性



ユーザーは、デフォルトで CodeReady Workspaces が提供する **self-hosted** モードで CodeReady Workspaces を拡張できます。

- 「Che-Theia プラグインについて」
- 「CodeReady Workspaces での代替 IDE の使用」
- 「VS Code 拡張機能のワークスペースへの追加」

4.1. CHE-THEIA プラグインについて

Che-Theia プラグインは、IDE から分離した開発環境の拡張です。プラグインは、ファイルまたはコンテナーとしてパッケージ化され、独自の依存関係を提供できます。

プラグインを使用して Che-Theia を拡張すると、以下の機能を有効にすることができます。

- **言語サポート:** Language Server Protocol に依存してサポートされる言語を拡張します。
- **デバッガー:** Debug Adapter Protocol を使用してデバッグ機能を拡張します。
- **開発ツール:** 優先するリンターを、テストおよびパフォーマンスツールとして統合します。

- **メニュー、パネルおよびコマンド**: 独自のアイテムを IDEコンポーネントに追加します。
- **テーマ**: カスタムテーマの構築、UI の拡張、またはアイコンテーマのカスタマイズを行います。
- **スニペット、フォーマッター、および構文のハイライト**: サポートされるプログラミング言語での使いやすさを強化します。
- **キーバインディング**: 新規のキーマップと一般的なキーバインディングを追加して、より自然な環境にします。

4.1.1. Che-Theia プラグインの機能と利点

機能	詳細	利点
高速ロード	プラグインはランタイム時に読み込まれ、すでにコンパイルされた状態になります。IDE はプラグインコードを読み込みます。	コンパイル時間は使用しないでください。インストール後の手順は使用しないでください。
セキュアなロード	プラグインは IDE とは別に読み込まれます。IDE は常に使用可能な状態のままになります。	バグがある場合、プラグインは IDE 全体を中断しません。ネットワークの問題を処理します。
ツールの依存関係	プラグインの依存関係は、独自のコンテナーのプラグインと共にパッケージ化されます。	ツールのインストールは不要です。コンテナーで実行される依存関係。
コードの分離	プラグインが、ファイルを開いたり、入力したりするなどの IDE の主な機能をブロックしないことを保証します。	プラグインは個別のスレッドで実行されます。依存関係の不一致を回避します。
VS Code 拡張機能の互換性	既存の VS Code 拡張機能で IDE の機能を拡張します。	複数のプラットフォームをターゲットにします。必要なインストールでの Visual Studio Code 拡張機能を簡単に検出できます。

4.1.2. Che-Theia プラグインの概念の詳細

Red Hat CodeReady Workspaces はワークスペースのデフォルト Web IDE (Che-Theia) を提供します。Eclipse Theia をベースにしています。これは、Red Hat CodeReady Workspaces ワークスペースの性質に基づいて追加された機能があるため、単純な Eclipse Theia とは若干異なります。CodeReady Workspaces のこのバージョンの Eclipse Theia は **Che-Theia** と呼ばれています。

Che-Theia プラグイン を構築することで、Red Hat CodeReady Workspaces で提供される IDE を拡張できます。Che-Theia プラグインは、その他の Eclipse Theia ベースの IDE と互換性があります。

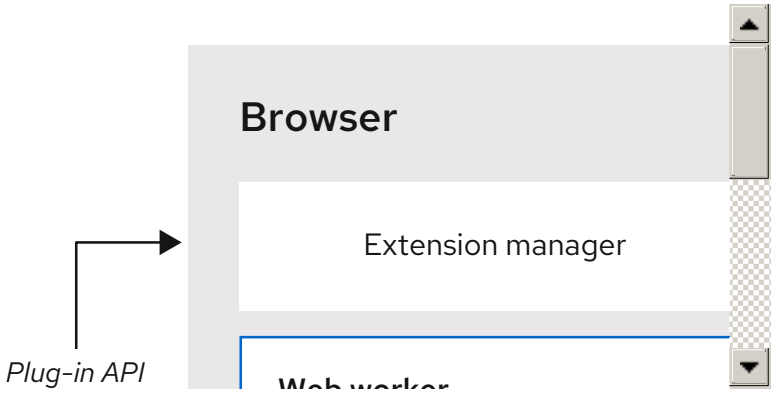
4.1.2.1. クライアントサイドおよびサーバーサイドの Che-Theia プラグイン

Che-Theia エディタープラグインを使用すると、開発ワークフローをサポートするために言語、デバッガー、およびツールをインストールに追加できます。エディターの読み込みが完了するとプラグインが

実行されます。Che-Theia プラグインが失敗すると、メインの Che-Theia エディターは機能し続けます。

Che-Theia プラグインはクライアントサイドまたはサーバーサイドのいずれかで実行されます。これは、クライアントおよびサーバーサイドのプラグインの概念のスキームです。

図4.2 クライアントおよびサーバー側の Che-Theia プラグイン



同じ Che-Theia プラグイン API がクライアントサイド (Web ワーカー) またはサーバーサイド (Node.js) で実行されるプラグインに公開されます。

4.1.2.2. Che-Theia プラグイン API

Red Hat CodeReady Workspaces でツールの分離と拡張のしやすさを実現する目的で、Che-Theia IDE にはプラグイン API のセットがあります。API は Visual Studio Code 拡張 API と互換性があります。通常、Che-Theia は、VS Code 拡張機能を独自のプラグインとして実行することができます。

CodeReady Workspaces ワークスペースのコンポーネント (コンテナー、設定、factory) に依存するか、またはこれと対話するプラグインを開発する場合は、Che-Theia に組み込まれた CodeReady Workspaces API を使用します。

4.1.2.3. Che-Theia プラグイン機能

Che-Theia プラグインには以下の機能があります。

プラグイン	詳細	リポジトリー
CodeReady Workspaces の拡張タスク	CodeReady Workspaces コマンドを処理し、ワークスペースの特定のコンテナーでそれらを起動する機能を提供します。	タスクプラグイン
CodeReady Workspaces 拡張ターミナル	ワークスペースのコンテナーのいずれかにターミナルを提供できるようにします。	拡張ターミナルの拡張機能
CodeReady Workspaces Factory	Red Hat CodeReady Workspaces Factory を処理します。	ワークスペースプラグイン

プラグイン	詳細	リポジトリー
CodeReady Workspaces コンテナー	ワークスペースで実行されているすべてのコンテナーを表示し、それらとの対話を可能にするコンテナービューを提供します。	コンテナープラグイン
ダッシュボード	IDE と **Dashboard** を統合し、ナビゲーションを容易にします。	Che-Theia Dashbord 拡張機能
CodeReady Workspaces API	IDE API を拡張し、CodeReady Workspaces 固有のコンポーネント (ワークスペース、設定) との対話を可能にします。	Che-Theia API 拡張

4.1.2.4. VS Code 拡張機能および Eclipse Theia プラグイン

Che-Theia プラグインは、VS Code 拡張機能または Eclipse Theia プラグインをベースとすることができます。

Visual Studio Code 拡張機能

VS Code 拡張機能を独自の依存関係セットを含む Che-Theia プラグインとして再パッケージ化するには、依存関係をコンテナーにパッケージ化します。これにより、エクステンションの使用時に Red Hat CodeReady Workspaces ユーザーが拡張機能の使用時に依存関係をインストールする必要がなくなります。「VS Code 拡張機能のワークスペースへの追加」を参照してください。

Eclipse Theia プラグイン

Eclipse Theia プラグインを実装し、Red Hat CodeReady Workspaces にパッケージ化することで、Che-Theia プラグインを構築できます。

関連資料

- 「埋め込み、およびリモートの Che-Theia プラグイン」

4.1.3. Che-Theia プラグインのメタデータ

Che-Theia プラグインメタデータは、プラグインレジストリーの個々のプラグインについての情報です。

それぞれの特定のプラグインの Che-Theia プラグインメタデータは、**meta.yaml** ファイルに定義されます。

以下は、プラグインメタ YAML ファイルで利用可能なすべてのフィールドの概要です。本書では、プラグインのメタ YAML 構造のバージョン 3 について示します。

che-plugin-registry リポジトリー には以下が含まれます。

表4.1 meta.yml

apiVersion	バージョン 2 以降（バージョン 1 は後方互換性のためにサポートされます）

category	利用可能: カテゴリ は**Editor**、**Debugger**、**Formatter**、**Language**、**Linter**、**Snippet**、**Theme**、**Other**のいずれかに設定する必要があります。
description	プラグインの目的についての簡単な説明
displayName	ユーザーダッシュボードに表示される名前
deprecate	オプション: プラグインを他を優先するために非推奨にするためのセクション * autoMigrate - ブール値 * migrateTo - 新しい org/plugin-id/version（例: **redhat/vscode-apache-camel/latest**）
firstPublicationDate	YAML に存在する必要はありませんが、存在しない場合、これは Plugin Registry dockerimage のビルド時に生成されます。
latestUpdateDate	YAML に存在する必要はありませんが、存在しない場合、これは Plugin Registry dockerimage のビルド時に生成されます。
icon	SVG または PNG アイコンの URL
name	名前 (スペースは使用できません) は [-a-z0-9] と一致する必要があります。
publisher	パブリッシャーの名前は [-a-z0-9] に一致する必要があります
repository	プラグインリポジトリーの URL (例: GitHub)
title	プラグインのタイトル (long)
type	**Che Plugin**、**VS Code extension**
バージョン	バージョン情報 (例: 7.5.1、[-.a-z0-9]
spec	仕様 (以下を参照)

表4.2 spec 属性

endpoint	オプション: プラグインエンドポイント。「エンドポイント」を参照してください。

containers	オプション: プラグインのサイドカーコンテナー。Che プラグインおよび VS Code 拡張機能は 1 つのコンテナーのみをサポートします。
initContainers	オプション: プラグイン用のサイドカーの init コンテナー
workspaceEnv	オプション: ワークスペースの環境変数
extensions	オプション: .vsix や .theia ファイルなど、プラグインのアーティファクトに対して VS Code および Che-Theia プラグインで必要な属性。

表4.3 spec.containers.注: spec.initContainers には同じコンテナー定義が含まれます。

name	サイドカーコンテナー名
image	絶対または相対コンテナーイメージ URL
memoryLimit	OpenShift メモリー制限の文字列 (例: **512Mi**)
memoryRequest	OpenShift メモリー要求文字列（例: **512Mi**)
cpuLimit	OpenShift CPU 制限の文字列（例: **2500m**)
cpuRequest	OpenShift CPU 要求の文字列 (例: **125m**)
env	サイドカーに設定される環境変数の一覧
command	コンテナー内の root プロセスコマンドの文字列配列の定義
args	コンテナー内の root プロセスコマンドの文字列配列の引数
volumes	プラグインで必要なボリューム
ポート	プラグインによって公開されるポート (コンテナー上)
commands	プラグインコンテナーで利用可能な開発コマンド
mountSources	ソースコード **/projects** のあるボリュームをプラグインコンテナーにバインドするブール値フラグ
initContainers	オプション: サイドカープラグイン用の init コンテナー

ライフサイクル	コンテナーライフサイクルフック。ライフサイクルの説明を参照してください。

表4.4 spec.containers.env および spec.initContainers.env 属性注: workspaceEnv は同じ属性を持ちます。

name	環境変数名
value	環境変数の値

表4.5 spec.containers.volumes および spec.initContainers.volumes 属性

mountPath	コンテナー内のボリュームへのパス
name	ボリューム名
ephemeral	true の場合、ボリュームは一時的になります。そうでない場合、ボリュームは永続化されます。

表4.6 spec.containers.ports および spec.initContainers.ports 属性

exposedPort	公開されるポート

表4.7 spec.containers.commands および spec.initContainers.commands 属性

name	コマンド名
workingDir	コマンドの作業ディレクトリー
command	開発コマンドを定義する文字列配列

表4.8 spec.endpoints 属性

name	名前 (スペースは使用できません) は [-a-z0-9] と一致する必要があります。
public	**true**, **false**
targetPort	ターゲットポート
attributes	エンドポイント属性

表4.9 spec.endpoints.attributes 属性

protocol	プロトコル (例: **ws**)

type	**ide**, **ide-dev**
discoverable	**true**, **false**
secure	**true**、**false**.**true** の場合、エンドポイントは **127.0.0.1** でのみリッスンすることが想定され、JWT プロキシーを使用して公開されます。
cookiesAuthEnabled	**true**, **false**
requireSubdomain	**true**、**false**.**true** の場合、エンドポイントは単一ホストモードでサブドメインで公開されます。

表4.10 spec.containers.lifecycle および spec.initContainers.lifecycle 属性

postStart	コンテナーの起動直後に実行される **postStart** イベント。postStart および preStop ハンドラーを参照してください。 * **exec**: 特定のコマンドを実行します。コマンドが使用するリソースはコンテナーに対してカウントされます。 * **command**: ["/bin/sh", "-c", "/bin/post-start.sh"]
preStop	コンテナーが終了する前に実行される **preStop** イベント。postStart および preStop ハンドラーを参照してください。 * **exec**: 特定のコマンドを実行します。コマンドが使用するリソースはコンテナーに対してカウントされます。 * **command**: ["/bin/sh", "-c", "/bin/post-start.sh"]

Che-Theia プラグインの meta.yaml の例: CodeReady Workspaces machine-exec サービス

```
  apiVersion: v2
  publisher: eclipse
  name: che-machine-exec-plugin
  version: 7.9.2
  type: Che Plugin
  displayName: CodeReady Workspaces machine-exec Service
  title: Che machine-exec Service Plugin
  description: CodeReady Workspaces Plug-in with che-machine-exec service to provide creation
terminal
    or tasks for Eclipse CHE workspace containers.
  icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
  repository: https://github.com/eclipse/che-machine-exec/
  firstPublicationDate: "2020-03-18"
  category: Other
  spec:
```

```
  endpoints:
   -  name: "che-machine-exec"
      public: true
      targetPort: 4444
      attributes:
        protocol: ws
        type: terminal
        discoverable: false
        secure: true
        cookiesAuthEnabled: true
  containers:
   - name: che-machine-exec
     image: "quay.io/eclipse/che-machine-exec:7.9.2"
     ports:
       - exposedPort: 4444
     command: ['/go/bin/che-machine-exec', '--static', '/cloud-shell', '--url', '127.0.0.1:4444']
```

VisualStudio コード拡張機能の meta.yaml の例: AsciiDoc サポート拡張機能

```
apiVersion: v2
category: Language
description: This extension provides a live preview, syntax highlighting and snippets for the AsciiDoc
format using Asciidoctor flavor
displayName: AsciiDoc support
firstPublicationDate: "2019-12-02"
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
name: vscode-asciidoctor
publisher: joaompinto
repository: https://github.com/asciidoctor/asciidoctor-vscode
title: AsciiDoctor Plug-in
type: VS Code extension
version: 2.7.7
spec:
  extensions:
  - https://github.com/asciidoctor/asciidoctor-vscode/releases/download/v2.7.7/asciidoctor-vscode-
2.7.7.vsix
```

4.1.4. Che-Theia プラグインのライフサイクル

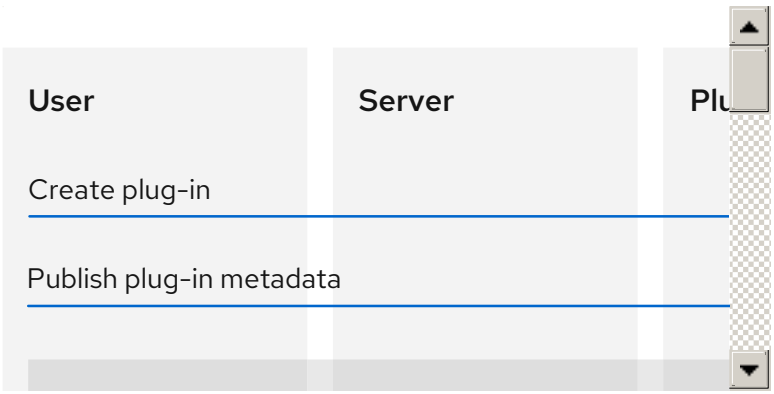
ユーザーが Che ワークスペースを起動するたびに、Che-Theia プラグインライフサイクルプロセスが開始します。このプロセスの手順は以下のとおりです。

1. CodeReady Workspaces サーバーは、プラグインがワークスペース定義から起動するかどうかを確認します。
2. CodeReady Workspaces サーバーはプラグインメタデータを取得し、各プラグインタイプを認識し、それらをメモリーに保存します。
3. CodeReady Workspaces サーバーは、プラグインタイプに応じてブローカーを選択します。
4. ブローカーはプラグインのインストールおよびデプロイメントを処理します。プラグインのインストールプロセスは、特定のブローカーごとに異なります。

注記

プラグインはさまざまなタイプで使用できます。ブローカーは、すべてのインストール要件を満たすことで、プラグインのデプロイメントを正常に実行します。

図4.3 Che-Theia プラグインのライフサイクル



CodeReady Workspaces ワークスペースを起動する前に、CodeReady Workspaces サーバーがワークスペースコンテナーを起動します。

1. Che-Theia プラグインブローカーは、特定のプラグインが **.theia** ファイルから必要とするサイドカーコンテナーについての情報を抽出します。
2. ブローカーは、適切なコンテナー情報を CodeReady Workspaces サーバーに送信します。
3. ブローカーは Che-Theia プラグインをボリュームにコピーし、これを Che-Theia エディターコンテナーで利用できるようにします。
4. 次に、CodeReady Workspaces サーバーはワークスペースのすべてのコンテナーを起動します。
5. Che-Theia はそのコンテナーで起動し、プラグインをロードするための適切なフォルダーをチェックします。

Che-Theia プラグインのライフサイクルにおけるユーザーエクスペリエンス

1. ユーザーが Che-Theia のブラウザータブを開くと、Che-Theia は、以下で新しいプラグインセッションを開始します。
   - Web Worker (フロントエンド)
   - Node.js (バックエンド)
2. Che-Theia は、トリガーされたそれぞれのプラグインの **start()** 関数を呼び出して、すべての Che-Theia プラグインに対して新規セッションの開始を通知します。
3. Che-Theia プラグインセッションが実行され、Che-Theia バックエンドおよびフロントエンドと対話します。
4. ユーザーが Che-Theia ブラウザータブを閉じるか、またはセッションがタイムアウトの制限で終了すると、Che-Theia はすべてのプラグインに対して、トリガーされたそれぞれのプラグインの **stop()** 関数で通知します。

4.1.5. 埋め込み、およびリモートの Che-Theia プラグイン

Red Hat CodeReady Workspaces の開発者ワークスペースは、プロジェクトで作業するために必要なすべての依存関係を提供します。アプリケーションには、使用されるすべてのツールおよびプラグインに必要な依存関係が含まれます。

必要な依存関係に基づいて、Che-Theia プラグインは以下のように実行できます。

- 埋め込み (ローカル)
- リモート

4.1.5.1. 埋め込み (ローカル) プラグイン

埋め込みプラグインは、Che-Theia IDE に挿入される特定の依存関係のないプラグインです。これらのプラグインは IDE コンテナーで実行される Node.js ランタイムを使用します。

例:

- コードリンティング
- コマンドの新規セット
- 新規 UI コンポーネント

Che-Theia プラグインまたは VS Code 拡張を含めるには、**meta.yaml** ファイルでプラグインの **.theia** アーカイブバイナリーへの URL を定義します。「VS Code 拡張機能のワークスペースへの追加」を参照してください。

ワークスペースを起動すると、CodeReady Workspaces はプラグインバイナリーをダウンロードして展開し、それらを Che-Theia エディターコンテナーに追加します。Che-Theia エディターは、起動時にプラグインを初期化します。

4.1.5.2. リモートプラグイン

プラグインは依存関係に依存するか、またはバックエンドがあります。これは独自のサイドカーコンテナーで実行され、すべての依存関係はコンテナーにパッケージ化されます。

リモート Che-Theia プラグインは、以下の 2 つの部分で構成されます。

- Che-Theia プラグインまたは VS Code 拡張機能バイナリー。**meta.yaml** ファイルの定義は埋め込みプラグインの場合と同じです。
- コンテナーイメージの定義（例: **eclipse/che-theia-dev:nightly**）。このイメージから、CodeReady Workspaces はワークスペース内に別のコンテナーを作成します。

例:

- Java Language Server
- Python Language Server

ワークスペースを起動すると、CodeReady Workspaces はプラグインイメージからコンテナーを作成し、プラグインバイナリーをダウンロードして展開し、それらを作成されたコンテナーに追加します。Che-Theia エディターは起動時にリモートプラグインに接続します。

4.1.5.3. 比較マトリックス

- 埋め込みプラグインは、コンテナー内で追加の依存関係を必要としない Che-Theia プラグインまたは VS Code 拡張機能です。
- リモートプラグインは、必要なすべての依存関係を持つプラグインが含まれるコンテナーです。

表4.11 Che-Theia プラグイン比較マトリックス: 組み込み vs. リモート

	プラグインごとに RAM を設定	環境の依存関係	分離されたコンテナーの作成
リモート	TRUE	プラグインは、リモートコンテナーで定義された依存関係を使用します。	TRUE
組み込み	FALSE (ユーザーはエディターコンテナー全体に対して RAM を設定できますが、プラグインごとに設定できません)	プラグインはエディターコンテナーから依存関係を使用します。コンテナーにこれらの依存関係が含まれない場合は、プラグインは失敗するか、または予想通りに機能しません。	FALSE

ユースケースおよびプラグインが提供する機能に応じて、上記の実行モードのいずれかを選択します。

4.1.6. リモートプラグインエンドポイント

Red Hat CodeReady Workspaces には、別のコンテナーで VS Code 拡張機能および Che-Theia プラグインを起動するためのリモートプラグインのエンドポイントサービスがあります。Red Hat CodeReady Workspaces は、リモートプラグインのエンドポイントのバイナリーを各リモートプラグインコンテナーに挿入します。このサービスは、プラグインの **meta.yaml** ファイルで定義されるリモート拡張機能とプラグインを起動し、それらを Che-Theia エディターコンテナーに接続します。

リモートプラグインのエンドポイントは、リモートプラグインコンテナーと Che-Theia エディターコンテナーとの間でプラグインの API プロキシーを作成します。リモートプラグインのエンドポイントは、一部のプラグインの API の部分のインターセプターにもなります。これは、エディターコンテナーではなくリモートサイドカーコンテナー内で起動します。例: ターミナル API、デバッグ API

リモートプラグインのエンドポイントの実行可能コマンドは、リモートプラグインコンテナーの環境変数 **PLUGIN_REMOTE_ENDPOINT_EXECUTABLE** に保存されます。

Red Hat CodeReady Workspaces は、サイドカーイメージを使用してリモートプラグインのエンドポイントを起動する 2 つの方法を提供します。

- Dockerfile を使用した起動リモートプラグインのエンドポイントの定義。この方法を使用するには、元のイメージにパッチを適用して再度ビルドします。
- プラグイン **meta.yaml** ファイルで、launch リモートプラグインのエンドポイントを定義します。元のイメージへのパッチの適用を回避するには、この方法を使用します。

4.1.6.1. Dockerfile を使用した起動リモートプラグインのエンドポイントの定義

リモートプラグインエンドポイントを起動するには、Dockerfile で **PLUGIN_REMOTE_ENDPOINT_EXECUTABLE** 環境変数を設定します。

手順

- Dockerfile で **CMD** コマンドを使用して、リモートプラグインのエンドポイントを起動します。

  Dockerfile の例

```
FROM fedora:30

RUN dnf update -y && dnf install -y nodejs htop && node -v

RUN mkdir /home/jboss

ENV HOME=/home/jboss

RUN mkdir /projects \
    && chmod -R g+rwX /projects \
    && chmod -R g+rwX "${HOME}"

CMD ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
```

- Dockerfile で **ENTRYPOINT** コマンドを使用して、リモートプラグインのエンドポイントを起動します。

  Dockerfile の例

```
FROM fedora:30

RUN dnf update -y && dnf install -y nodejs htop && node -v

RUN mkdir /home/jboss

ENV HOME=/home/jboss

RUN mkdir /projects \
    && chmod -R g+rwX /projects \
    && chmod -R g+rwX "${HOME}"

ENTRYPOINT ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
```

4.1.6.1.1. ラッパースクリプトの使用

一部のイメージは、ラッパースクリプトを使用してコンテナー内のパーミッションを設定します。Dockertfile **ENTRYPOINT** コマンドは、Dockerfile の **CMD** コマンドで定義される主なプロセスを実行するこのスクリプトを定義します。

CodeReady Workspaces はラッパースクリプトと共にイメージを使用し、高度なセキュリティーで保護される複数の異なるインフラストラクチャーに対するパーミッション設定を行います。OpenShift Container Platform はこのようなインフラストラクチャーの一例です。

- ラッパースクリプトの例:

```
#!/bin/sh
```

```
set -e

export USER_ID=$(id -u)
export GROUP_ID=$(id -g)

if ! whoami >/dev/null 2>&1; then
    echo "${USER_NAME:-user}:x:${USER_ID}:0:${USER_NAME:-user}
user:${HOME}:/bin/sh" >> /etc/passwd
fi

# Grant access to projects volume in case of non root user with sudo rights
if [ "${USER_ID}" -ne 0 ] && command -v sudo >/dev/null 2>&1 && sudo -n true > /dev/null
2>&1; then
    sudo chown "${USER_ID}:${GROUP_ID}" /projects
fi

exec "$@"
```

- ラッパースクリプトを含む Dockerfile の例:

Dockerfile の例

```
FROM alpine:3.10.2

ENV HOME=/home/theia

RUN mkdir /projects ${HOME} && \
    # Change permissions to let any arbitrary user
    for f in "${HOME}" "/etc/passwd" "/projects"; do \
      echo "Changing permissions on ${f}" && chgrp -R 0 ${f} && \
      chmod -R g+rwX ${f}; \
    done

ADD entrypoint.sh /entrypoint.sh

ENTRYPOINT [ "/entrypoint.sh" ]
CMD ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
```

説明:

- コンテナーは、Dockerfile の **ENTRYPOINT** コマンドで定義される **/entrypoint.sh** スクリプトを起動します。
- このスクリプトはパーミッションを設定し、**exec $@** を使用してコマンドを実行します。
- **CMD** は **ENTRYPOINT** の引数で、**exec $@** コマンドは **${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}** を呼び出します。
- 次に、リモートプラグインのエンドポイントは、パーミッションの設定後にコンテナーで起動します。

4.1.6.2. meta.yaml ファイルで、launch リモートプラグインのエンドポイントを定義します。

この方法を使用して、変更なしにリモートプラグインのエンドポイントを起動するためにイメージを再利用します。

手順

プラグインの **meta.yaml** ファイルプロパティー **command** および **args** を変更します。

- **Command** - CodeReady Workspaces は、 **command** プロパティーを使用して **Dockerfile#ENTRYPOINT** の値を上書きします。
- **args** - CodeReady Workspaces は **args** プロパティーを使用して **Dockerfile#CMD** 値を上書きします。
- **command** および **args** プロパティーが変更された YAML ファイルの例:

```
apiVersion: v2
category: Language
description: "Typescript language features"
displayName: Typescript
firstPublicationDate: "2019-10-28"
icon: "https://www.eclipse.org/che/images/logo-eclipseche.svg"
name: typescript
publisher: che-incubator
repository: "https://github.com/Microsoft/vscode"
title: "Typescript language features"
type: "VS Code extension"
version: remote-bin-with-override-entrypoint
spec:
  containers:
    - image: "example/fedora-for-ts-remote-plugin-without-endpoint:latest"
      memoryLimit: 512Mi
      name: vscode-typescript
      command:
        - sh
        - -c
      args:
        - ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
  extensions:
    - "https://github.com/che-incubator/ms-code.typescript/releases/download/v1.35.1/che-typescript-language-1.35.1.vsix"
```

- ラッパースクリプトのパターンでイメージを使用し、**entrypoint.sh** スクリプトの呼び出しを保持するには、**command** ではなく **args** を変更します。

```
apiVersion: v2
category: Language
description: "Typescript language features"
displayName: Typescript
firstPublicationDate: "2019-10-28"
icon: "https://www.eclipse.org/che/images/logo-eclipseche.svg"
name: typescript
publisher: che-incubator
repository: "https://github.com/Microsoft/vscode"
title: "Typescript language features"
type: "VS Code extension"
version: remote-bin-with-override-entrypoint
```

```
spec:
  containers:
    - image: "example/fedora-for-ts-remote-plugin-without-endpoint:latest"
      memoryLimit: 512Mi
      name: vscode-typescript
      args:
        - sh
        - -c
        - ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
  extensions:
    - "https://github.com/che-incubator/ms-code.typescript/releases/download/v1.35.1/che-
typescript-language-1.35.1.vsix"
```

Red Hat CodeReady Workspaces は、Dockerfile の **ENTRYPOINT** コマンドで定義された **entrypoint.sh** ラッパースクリプトを呼び出します。このスクリプトは、**exec "$@"** コマンドを使用して **['sh', '-c", ' ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}']** を実行します。

注記

meta.yaml ファイルの command および args プロパティーを変更することにより、ユーザーは以下を実行できます。

- コンテナー起動時のサービスの実行
- リモートプラグインのエンドポイントの開始

これらのアクションを同時に実行するには、以下を実行します。

1. サービスを起動します。
2. プロセスの割り当てを解除します。
3. リモートプラグインエンドポイントを起動します。

4.2. VS CODE 拡張機能のワークスペースへの追加

本セクションでは、**CodeReady Workspaces Plugins** パネルまたはワークスペース設定を使用して、VS Code 拡張をワークスペースに追加する方法を説明します。

前提条件

- VS Code 拡張機能は CodeReady Workspaces プラグインレジストリーで利用でき、VS Code 拡張機能のメタデータも利用できます。「VS Code 拡張のメタデータの公開」を参照してください。

4.2.1. CodeReady Workspaces プラグインパネルを使用した VS Code 拡張機能の追加

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。

- VS Code 拡張機能は CodeReady Workspaces プラグインレジストリーで利用でき、VS Code 拡張機能のメタデータも利用できます。「VS Code 拡張のメタデータの公開」を参照してください。

手順

CodeReady Workspaces Plugins パネルを使用して VS Code 拡張機能を追加するには、以下を実行します。

1. **CTRL+SHIFT+J** を押して **CodeReady Workspaces プラグイン** パネルを開くか、または **View/Plugins** に移動します。
2. 現行のレジストリーを、VS Code 拡張機能が追加されているレジストリーに変更します。
3. 検索バーで **Menu** ボタンをクリックしてから **Change Registry** をクリックして、一覧からレジストリーを選択します。必要なレジストリーが一覧にない場合は、**Add Registry** メニューオプションを使用して追加します。レジストリーのリンクは、レジストリーの **plugins** セグメントを参照します（例: **https://my-registry.com/v3/plugins/index.json**）。
4. 新規のレジストリーリンクを追加した後にプラグインの一覧を更新するには、検索バーメニューから **Refresh** コマンドを使用します。
5. フィルターを使用して必要なプラグインを検索し、**Install** ボタンをクリックします。
6. 変更を有効にするには、ワークスペースを再起動します。

4.2.2. ワークスペース設定を使用した VS Code 拡張の追加

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」 のこのインスタンスで定義される既存のワークスペース。
- VS Code 拡張機能は CodeReady Workspaces プラグインレジストリーで利用でき、VS Code 拡張機能のメタデータも利用できます。「VS Code 拡張のメタデータの公開」を参照してください。

手順

ワークスペース設定を使用して VS Code 拡張機能を追加するには、以下を実行します。

1. **Dashboard** の **Workspaces** タブをクリックして、プラグインを追加するワークスペースを選択します。**Workspace <workspace-name>** ウィンドウが開き、ワークスペースの詳細が表示されます。
2. **devfile** タブをクリックします。
3. **components** セクションを見つけ、以下の構造で新規エントリーを追加します。

```
- type: chePlugin
  id:                (1)
```

(1) ID フォーマット: **<publisher>/<plug-inName>/<plug-inVersion>**

CodeReady Workspaces は、他のフィールドを新規コンポーネントに自動的に追加します。

または、専用の参照フィールドを使用して、GitHub でホストされる **meta.yaml** ファイルにリンクできます。

```
- type: chePlugin
  reference:
```
(1)

(1) **https://raw.githubusercontent.com/<username>/<registryRepository>/v3/plugins/<publisher>/<plug-inName>/<plug-inVersion>/meta.yaml**

4. 変更を有効にするには、ワークスペースを再起動します。

4.3. VS CODE 拡張のメタデータの公開

CodeReady Workspaces ワークスペースで VS Code 拡張機能を使用するには、CodeReady Workspaces が拡張機能を記述するメタデータを使用する必要があります。CodeReady Workspaces プラグインレジストリーは、一般的な VS Code 拡張機能についてのメタデータを公開する静的 Web サイトです。

拡張機能設定を使用してCodeReadyWorkspacesプラグインレジストリで利用できない、追加の拡張機能のメタデータを公開する方法。**meta.yaml** ファイル

前提条件

- VS Code 拡張機能で必要な場合には、必要とされる関連付けられたコンテナーイメージが利用可能になります。

手順

1. **meta.yaml** ファイルを作成します。
2. **meta.yaml** ファイルを編集し、必要な情報を指定します。ファイルには、以下の構造が必要です。

```
apiVersion: v2                                                  (1)
publisher: myorg                                                (2)
name: my-vscode-ext                                             (3)
version: 1.7.2                                                  (4)
type: value                                                     (5)
displayName:                                                    (6)
title:                                                          (7)
description:                                                    (8)
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg    (9)
repository:                                                     (10)
category:                                                       (11)
spec:
  containers:                                                   (12)
    - image:                                                    (13)
      memoryLimit:                                              (14)
      memoryRequest:                                            (15)
      cpuLimit:                                                 (16)
```

```
    cpuRequest:                                    17
  extensions:                                      18
        - https://github.com/redhat-developer/vscode-
yaml/releases/download/0.4.0/redhat.vscode-yaml-0.4.0.vsix
        - https://github.com/SonarSource/sonarlint-vscode/releases/download/1.16.0/sonarlint-
vscode-1.16.0.vsix
```

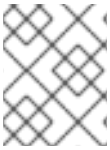
1. ファイル構造のバージョン。
2. プラグインパブリッシャーの名前。パスのパブリッシャーと同じである必要があります。
3. プラグインの名前。パスで使用されているものと同じである必要があります。
4. プラグインのバージョン。パスで使用されているものと同じである必要があります。
5. プラグインのタイプ。設定可能な値: **Che Plugin**、**Che Editor**、**Theia plugin**、**VS Code extension**。
6. プラグインの省略名。
7. プラグインのタイトル。
8. プラグインとその機能についての簡単な説明。
9. プラグインロゴへのリンク。
10. オプション。プラグインのソースコードリポジトリーへのリンク。
11. このプラグインが属するカテゴリーを定義します。**Editor**、**Debugger**、**Formatter**、**Language**、**Linter**、**Snippet**、**Theme**、または **Other** のいずれかでなければなりません。
12. このセクションを省略すると、VS Code 拡張機能が Che-Theia IDE コンテナーに追加されます。
13. サイドカーコンテナーが起動する Docker イメージ。例: **theia-endpoint-image**.
14. サイドカーコンテナーで利用可能な最大 RAM。例: "512Mi"この値は、コンポーネント設定でユーザーによって上書きされる可能性があります。
15. デフォルトでサイドカーコンテナーに指定される RAM。例: "256Mi"この値は、コンポーネント設定でユーザーによって上書きされる可能性があります。
16. サイドカーコンテナーで利用可能なコアまたはミリコア単位 (末尾に m が付く) の CPU の最大量。例: "500m"、"2"この値は、コンポーネント設定でユーザーによって上書きされる可能性があります。
17. デフォルトでサイドカーコンテナーに指定されるコアまたはミリコア単位 (末尾に m が付く) の CPU 量。例: "125m"この値は、コンポーネント設定でユーザーによって上書きされる可能性があります。
18. このサイドカーコンテナーで実行される VS Code 拡張機能の一覧。

3. GitHub または GitLab に公開されたファイルコンテンツで gist を作成し、**meta.yaml** ファイルを HTTP リソースとして公開します。

4.4. CODEREADY WORKSPACES での VISUAL STUDIO CODE 拡張機能のテスト

Visual Studio Code (VS Code) 拡張機能はワークスペースで機能します。VS Code 拡張機能は、Che-Theia エディターコンテナー、またはその前提条件を満たした独自の分離され、事前に設定されたコンテナーで実行されます。

本セクションでは、ワークスペースを使用して CodeReady Workspaces で VS Code 拡張をテストする方法と、VS Code 拡張の互換性を確認し、特定の API が利用可能かどうかを確認する方法を説明します。



注記

extension-hosting サイドカーコンテナーおよび devfile での拡張機能の使用は任意です。

4.4.1. GitHub gist を使用した VS Code 拡張機能のテスト

各ワークスペースには独自のプラグインセットを使用できます。プラグインの一覧、およびクローンを作成するプロジェクトの一覧は、**devfile.yaml** ファイルで定義されます。

たとえば、Red Hat CodeReady Workspaces ダッシュボードから AsciiDoc プラグインを有効にするには、以下のスニペットを devfile に追加します。

```
components:
 - id: joaopinto/vscode-asciidoctor/latest
   type: chePlugin
```

デフォルトのプラグインレジストリーにないプラグインを追加するには、カスタムプラグインレジストリーをビルドします。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/administration_guide/index#customizing-the-registries_crw を参照するか、または GitHub および gist サービスを使用してます。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- GitHub アカウント。

手順

1. gist Webページに移動し、**README.md** ファイルを作成します。これには、説明 **Try Bracket Pair Colorizer extension in Red Hat CodeReady Workspaces** およびコンテンツ **Example VS Code extension** を含みます。(Bracket Pair Colorizer はよく使用される VS Code 拡張機能です。)
2. **Create secret gist** ボタンをクリックします。
3. ブラウザーのナビゲーションバーの URL を使用して、gist リポジトリーのクローンを作成します。

```
$ git clone https://gist.github.com/<your-github-username>/<gist-id>
```

git clone コマンドの出力例

```
git clone https://gist.github.com/benoitf/85c60c8c439177ac50141d527729b9d9 (1)
Cloning into '85c60c8c439177ac50141d527729b9d9'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

(1) 各 gist には固有の ID があります。

4. ディレクトリーを変更します。

```
$ cd <gist-directory-name> (1)
```

(1) gist ID に一致するディレクトリー名。

5. VS Code marketplace または GitHub ページ からプラグインをダウンロードし、プラグインファイルをクローン作成されたディレクトリーに保存します。

6. クローン作成されたディレクトリーに **plugin.yaml** ファイルを作成し、このプラグインの定義を追加します。

.vsix バイナリーファイルの拡張を参照する plugin.yaml ファイルの例

```
apiVersion: v2
publisher: CoenraadS
name: bracket-pair-colorizer
version: 1.0.61
type: VS Code extension
displayName: Bracket Pair Colorizer
title: Bracket Pair Colorizer
description: Bracket Pair Colorizer
icon: https://raw.githubusercontent.com/redhat-developer/codeready-
workspaces/master/dependencies/che-plugin-registry/resources/images/default.svg?
sanitize=true
repository: https://github.com/CoenraadS/BracketPair
category: Language
firstPublicationDate: '2020-07-30'
spec:                                          (1)
  extensions:
  - "{{REPOSITORY}}/CoenraadS.bracket-pair-colorizer-1.0.61.vsix" (2)
latestUpdateDate: "2020-07-30"
```

(1) この拡張には基本的な Node.js ランタイムが必要であるため、**plugin.yaml** にカスタムのランタイムイメージを追加する必要はありません。

(2) **{{REPOSITORY}}** は pre-commit フックのマクロです。

7. メモリー制限およびボリュームを定義します。

```
spec:
  containers:
    - image: "quay.io/eclipse/che-sidecar-java:8-0cfbacb"
      name: vscode-java
      memoryLimit: "1500Mi"
      volumes:
      - mountPath: "/home/theia/.m2"
        name: m2
```

8. **plugin.yaml** ファイルを参照する **devfile.yaml** を作成します。

```
apiVersion: 1.0.0
metadata:
  generateName: java-maven-
projects:
  -
    name: console-java-simple
    source:
      type: git
      location: "https://github.com/che-samples/console-java-simple.git"
      branch: java1.11
components:
  -
    type: chePlugin
    id: redhat/java11/latest
  -
    type: chePlugin 1
    reference: "{{REPOSITORY}}/plugin.yaml"
  -
    type: dockerimage
    alias: maven
    image: quay.io/eclipse/che-java11-maven:nightly
    memoryLimit: 512Mi
    mountSources: true
    volumes:
      - name: m2
        containerPath: /home/user/.m2
commands:
  -
    name: maven build
    actions:
      -
        type: exec
        component: maven
        command: "mvn clean install"
        workdir: ${CHE_PROJECTS_ROOT}/console-java-simple
  -
    name: maven build and run
    actions:
      -
        type: exec
        component: maven
        command: "mvn clean install && java -jar ./target/*.jar"
        workdir: ${CHE_PROJECTS_ROOT}/console-java-simple
```

1 その他の devfile 定義も使用できます。この devfile の重要な情報は、この外部コンポーネントを定義する行になります。これは、外部参照が (デフォルトプラグインレジストリー

9. 現在の Git ディレクトリーに 4 つのファイルがあることを確認します。

```
$ ls -la
.git
CoenraadS.bracket-pair-colorizer-1.0.61.vsix
README.md
devfile.yaml
plugin.yaml
```

10. ファイルをコミットする前に、pre-commit フックを追加して、**{{REPOSITORY}}** 変数を公開されている外部の未加工 gist リンクに対して更新します。

    a. 以下の内容で **.git/hooks/pre-commit** ファイルを作成します。

```
#!/bin/sh

# get modified files
FILES=$(git diff --cached --name-only --diff-filter=ACMR "*.yaml" | sed 's| |\\ |g')

# exit fast if no files found
[ -z "$FILES" ] && exit 0

# grab remote origin
origin=$(git config --get remote.origin.url)
url="${origin}/raw"

# iterate on files and add the good prefix pattern
for FILE in ${FILES}; do
 sed -e "s#{{REPOSITORY}}#${url}#g" "${FILE}" > "${FILE}.back"
 mv "${FILE}.back" "${FILE}"
done

# Add back to staging
echo "$FILES" | xargs git add

exit 0
```

    フックは **{{REPOSITORY}}** マクロを置き換え、外部の未加工リンクを gist に追加します。

    b. スクリプトを実行可能にします。

```
$ chmod u+x .git/hooks/pre-commit
```

11. ファイルをコミットし、プッシュします。

```
# Add files
$ git add *

# Commit
$ git commit -m "Initial Commit for the test of our extension"
[master 98dd370] Initial Commit for the test of our extension
```

```
 3 files changed, 61 insertions(+)
 create mode 100644 CoenraadS.bracket-pair-colorizer-1.0.61.vsix
 create mode 100644 devfile.yaml
 create mode 100644 plugin.yaml

# and push the files to the main branch
$ git push origin
```

12. gist の Web サイトにアクセスし、すべてのリンクに正しい公開 URL があり、**{{REPOSITORY}}** 変数が含まれていないことを確認します。devfile に到達するには、以下を実行します。

```
$ echo "$(git config --get remote.origin.url)/raw/devfile.yaml"
```

または

```
$ echo "https://<che-server>/f?url=$(git config --get remote.origin.url)/raw/devfile.yaml"
```

4.4.2. VS Code 拡張機能 API の互換性レベルの確認

Che-Theia は、VS Code 拡張機能 API を完全にサポートしません。vscode-theia-comparator は、Che-Theia プラグイン API と VS Code 拡張機能 API 間の互換性を分析するために使用されます。このツールは夜間に実行され、結果は vscode-theia-comparator GitHub ページで公開されます。

前提条件

- 個人用の GitHub アクセストークン。コマンドラインの個人アクセストークンの作成を参照してください。IP アドレスの GitHub ダウンロード制限を増やすには、GitHub アクセストークンが必要です。

手順

vscode-theia comparator を手動で実行するには、以下を行います。

1. vscode-theia-comparator リポジトリーをクローン作成し、**yarn** コマンドを使用してこれをビルドします。
2. **GITHUB_TOKEN** 環境変数をトークンに設定します。
3. **yarn run generate** コマンドを実行してレポートを生成します。
4. **out/status.html** ファイルを開き、レポートを表示します。

4.5. CODEREADY WORKSPACES での代替 IDE の使用

異なる IDE (統合開発環境) を使用して Red Hat CodeReady Workspaces 開発者ワークスペースを拡張すると、以下が可能になります。

- 異なるユースケース用に環境を使用する。
- 特定ツールに専用のカスタム IDE を提供する。
- ユーザーの個別ユーザーまたはグループにそれぞれ異なるパースペクティブを提供する。

Red Hat CodeReady Workspaces は、開発者ワークスペースで使用するデフォルトの Web IDE を提供します。この IDE は完全に切り離されています。Red Hat CodeReady Workspaces 用に独自のカスタム IDE を組み込むことができます。

- Web IDE を構築するフレームワークである **Eclipse Theia からビルド** されます。例: Web 上の Sirius
- **完全に異なる Web IDE** になります (Jupyter、Eclipse Dirigible など)。例: Red Hat CodeReady Workspaces ワークスペースの Jupyter。

Eclipse Theia からビルドされたカスタム IDE の追加

- Eclipse Theia をベースにした独自のカスタム IDE の作成。
- CodeReady Workspaces 固有のツールのカスタム IDE への追加。
- カスタム IDE を CodeReady Workspaces の利用可能なエディターにパッケージ化する。

完全に異なる Web IDE の CodeReady Workspaces への追加

- カスタム IDE を CodeReady Workspaces の利用可能なエディターにパッケージ化する。

4.6. JETBRAINS IDE のサポート

本セクションでは、Red Hat CodeReady Workspaces ワークスペースで使用できるサポート対象の JetBrains IDE について説明します。

Red Hat CodeReady Workspaces は、以下の JetBrains IDE の一覧に基づくワークスペースの実行をサポートします。

- IntelliJ Idea Community Edition
- IntelliJ Idea Ultimate Edition
- WebStorm

サポートされる予定の JetBrains IDE の一覧。

- GoLand
- PhpStorm
- PyCharm Professional Edition
- PyCharm Community Edition

注記

サポートされる JetBrains 製品のバージョンは 2018.1 以降である必要があります。

以下のセクションでは、ビルドされたイメージに基づいて、特定の IDE およびワークスペースでイメージを作成する方法を説明します。

- 「IntelliJ Idea Community Edition の使用」

- 「IntelliJ Idea Ultimate Edition の使用」
- 「WebStorm の使用」

4.6.1. IntelliJ Idea Community Edition の使用

手順

1. che-editor-intellij-community リポジトリーのクローンを作成します。これは、**che-incubator** 組織下にある IntelliJ Idea Community Edition をビルドするために必要です。
2. リポジトリーのフォルダー内で以下のコマンドを呼び出して、IntelliJ Idea Community Edition をビルドします。

```
$ podman build -t idea-ic --build-arg PRODUCT_NAME=ideaIC .
```

このコマンドは、デフォルトで **2020.2.3** バージョンでイメージをビルドします。

3. ビルドしたイメージをタグ付けし、ユーザーリポジトリーにプッシュします。

```
$ podman tag idea-ic:latest <username>/idea-ic:latest
$ podman push <username>/idea-ic:latest
```

4. このイメージを CodeReady Workspaces エディターとして使用します。これを実行するには、2 つの YAML 設定ファイルを作成します。
   - **workspace.yaml** – ワークスペースの設定。**meta.yaml** ファイルには必ず正しい URL を指定してください。

```
metadata:
  name: che-ideaic
components:
  - type: cheEditor
    reference: '<URL to the meta.yaml>'
    alias: ideaic-editor
apiVersion: 1.0.0
```

   - **meta.yaml** – CodeReady Workspaces エディターの設定。**<username>** は、必ずイメージがプッシュされるリポジトリーのユーザー名に置き換えてください。

```
apiVersion: v2
publisher: <username>
name: ideaic-NOVNC
version: 2020.2.3
type: Che Editor
displayName:  IntelliJ IDEA Community Edition
title:  IntelliJ IDEA Community Edition (in browser using noVNC) as editor for Red Hat CodeReady Workspaces
description:  IntelliJ IDEA Community Edition running on the Web with noVNC
icon: https://resources.jetbrains.com/storage/products/intellij-idea/img/meta/intellij-idea_logo_300x300.png
category: Editor
repository: https://github.com/che-incubator/che-editor-intellij-community
firstPublicationDate: "2020-10-27"
```

```
spec:
  endpoints:
   - name: "intellij"
     public: true
     targetPort: 8080
     attributes:
       protocol: http
       type: ide
       path: /vnc.html?resize=remote&autoconnect=true&reconnect=true
  containers:
   - name: ideaic-novnc
     image: "<username>/idea-ic:latest"
     mountSources: true
     volumes:
      - mountPath: "/JetBrains/ideaIC"
        name: ideaic-configuration
     ports:
      - exposedPort: 8080
     memoryLimit: "2048M"
```

4.6.2. IntelliJ Idea Ultimate Edition の使用

手順

1. che-editor-intellij-community リポジトリーのクローンを作成します。これは、**che-incubator** 組織下にある IntelliJ Idea Community Edition をビルドするために必要です。
2. リポジトリーのフォルダー内で以下のコマンドを呼び出して、IntelliJ Idea Ultimate Edition をビルドします。

```
$ podman build -t idea-iu --build-arg PRODUCT_NAME=ideaIU .
```

このコマンドは、デフォルトで **2020.2.3** バージョンでイメージをビルドします。

3. ビルドしたイメージをタグ付けし、ユーザーリポジトリーにプッシュします。

```
$ podman tag idea-iu:latest <username>/idea-iu:latest
$ podman push <username>/idea-iu:latest
```

4. オフラインで使用するアクティベーションコードをプロビジョニングし、登録済みのライセンスで WebStorm を使用できるようにします。「オフラインで使用するための JetBrain アクティベーションコードのプロビジョニング」 セクションを参照してください。
5. 以下の **workspace.yaml** ファイルおよび **meta.yaml** ファイルを使用してワークスペースを作成します。
   - **workspace.yaml** – ワークスペースの設定。**meta.yaml** ファイルには必ず正しい URL を指定してください。

```
metadata:
  name: che-ideaiu
components:
  - type: cheEditor
    reference: '<URL to the meta.yaml>'
```

```
    alias: ideaiu-editor
    automountWorkspaceSecrets: true
apiVersion: 1.0.0
```

注記

現在のワークスペース定義には、新規のプロパティー **automountWorkspaceSecrets: true** があります。このプロパティーは、シークレットを特定のコンポーネントにプロビジョニングするように Red Hat CodeReady Workspaces に指示します。この場合、これは IntelliJ Idea Ultimate Edition に基づいて CodeReady Workspaces エディターにプロビジョニングします。このパラメーターは、オフラインで使用する目的で IDE をアクティベーションコードで正常に登録するために **必須** です。

- **meta.yaml** – CodeReady Workspaces エディターの設定。**<username>** は、必ずイメージがプッシュされるリポジトリーのユーザー名に置き換えてください。

```
apiVersion: v2
publisher: <username>
name: ideaIU-NOVNC
version: 2020.2.3
type: Che Editor
displayName:  IntelliJ IDEA Ultimate Edition
title:  IntelliJ IDEA Ultimate Edition (in browser using noVNC) as editor for Red Hat
CodeReady Workspaces
description:  IntelliJ IDEA Ultimate Edition running on the Web with noVNC
icon: https://resources.jetbrains.com/storage/products/intellij-idea/img/meta/intellij-
idea_logo_300x300.png
category: Editor
repository: https://github.com/che-incubator/che-editor-intellij-community
firstPublicationDate: "2020-10-27"
spec:
  endpoints:
  -  name: "intellij"
     public: true
     targetPort: 8080
     attributes:
       protocol: http
       type: ide
       path: /vnc.html?resize=remote&autoconnect=true&reconnect=true
  containers:
  - name: ideaiu-novnc
    image: "<username>/idea-iu:latest"
    mountSources: true
    volumes:
       - mountPath: "/JetBrains/ideaIU"
         name: ideaiu-configuration
    ports:
       - exposedPort: 8080
    memoryLimit: "2048M"
```

4.6.3. WebStorm の使用

手順

1. che-editor-intellij-community リポジトリーのクローンを作成します。これは、**che-incubator** 組織下にある IntelliJ Idea Community Edition をビルドするために必要です。

2. イメージをビルドします。

```
$ podman build -t webstorm --build-arg PRODUCT_NAME=WebStorm .
```

   このコマンドは、デフォルトで **2020.2.3** バージョンでイメージをビルドします。

3. ビルドしたイメージをタグ付けし、ユーザーリポジトリーにプッシュします。

```
$ podman tag webstorm:latest <username>/webstorm:latest
$ podman push <username>/webstorm:latest
```

4. オフラインで使用するアクティベーションコードをプロビジョニングし、登録済みのライセンスで WebStorm を使用できるようにします。「オフラインで使用するための JetBrain アクティベーションコードのプロビジョニング」 セクションを参照してください。

5. 以下の **workspace.yaml** ファイルおよび **meta.yaml** ファイルを使用してワークスペースを作成します。

   - **workspace.yaml** – ワークスペースの設定。**meta.yaml** ファイルには必ず正しい URL を指定してください。

```
metadata:
  name: che-webstorm
components:
  - type: cheEditor
    reference: '<URL to meta.yaml>'
    alias: webstorm-editor
    automountWorkspaceSecrets: true
apiVersion: 1.0.0
```

注記

現在のワークスペース定義には、新規のプロパティー **automountWorkspaceSecrets: true** があります。このプロパティーは、シークレットを特定のコンポーネントにプロビジョニングするように Red Hat CodeReady Workspaces に指示します。この場合、これは IntelliJ Idea Ultimate Edition に基づいて CodeReady Workspaces エディターにプロビジョニングします。このパラメーターは、オフラインで使用する目的で IDE をアクティベーションコードで正常に登録するために **必須** です。

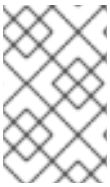
   - **meta.yaml** – CodeReady Workspaces エディターの設定。**<username>** は、必ずイメージがプッシュされるリポジトリーのユーザー名に置き換えてください。

```
apiVersion: v2
publisher: <username>
name: webstorm-NOVNC
version: 2020.2.3
type: Che Editor
displayName:  WebStorm
title:  WebStorm (in browser using noVNC) as editor for Red Hat CodeReady Workspaces
```

```
description:  WebStorm running on the Web with noVNC
icon:
https://resources.jetbrains.com/storage/products/webstorm/img/meta/webstorm_logo_300x
300.png
category: Editor
repository: https://github.com/che-incubator/che-editor-intellij-community
firstPublicationDate: "2020-10-27"
spec:
  endpoints:
   - name: "intellij"
     public: true
     targetPort: 8080
     attributes:
       protocol: http
       type: ide
       path: /vnc.html?resize=remote&autoconnect=true&reconnect=true
  containers:
   - name: webstorm-novnc
     image: "<username>/webstorm:latest"
     mountSources: true
     volumes:
      - mountPath: "/JetBrains/WebStorm"
        name: webstorm-configuration
     ports:
      - exposedPort: 8080
     memoryLimit: "2048M"
```

4.6.4. オフラインで使用するための JetBrain アクティベーションコードのプロビジョニング

オフラインで使用するためのアクティベーションコードは、割り当てられたライセンスについて JetBrains アカウントのライセンス管理セクションから取得できるライセンスコードを含むファイルです。個人のサブスクリプションを購入したり、組織によって商用サブスクリプションに割り当てられる場合、ライセンスに接続される JetBrains アカウントの作成を求めるメールを受信します。



注記

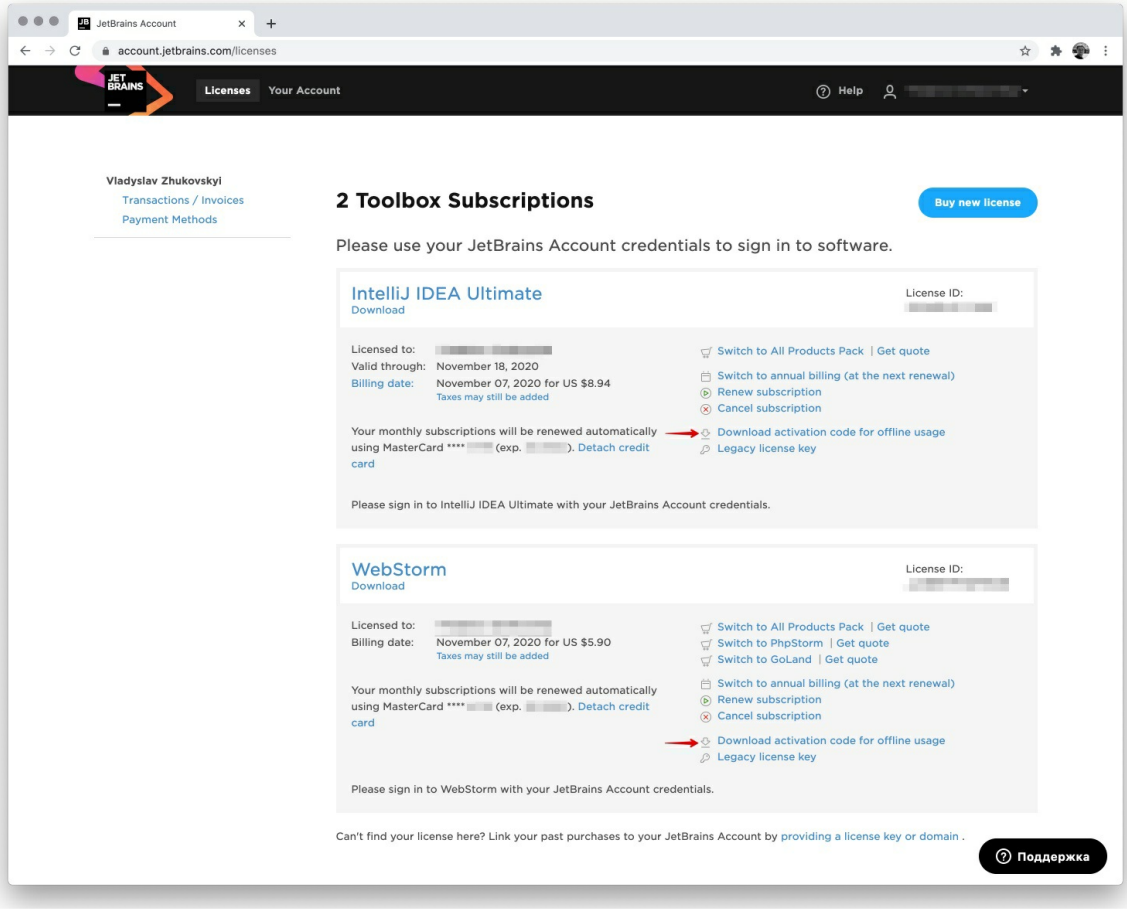
アクティベーションキーを使用して製品をアクティブにする場合は、新規アクティベーションコードを生成し、これをサブスクリプションが更新されるたびに製品に適用する必要があります。

前提条件

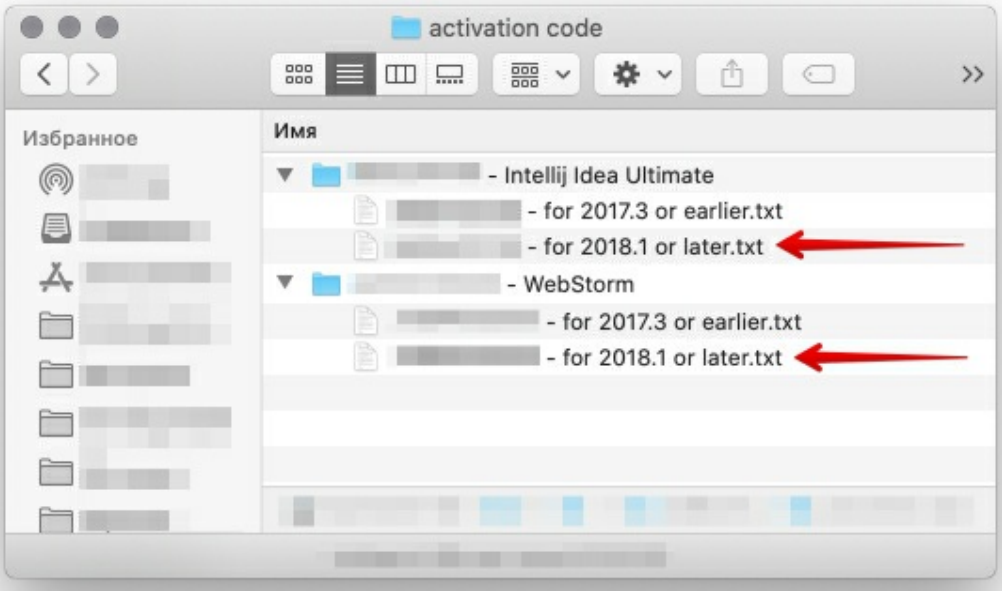
- JetBrains アカウント
- 個人または組織のサブスクリプション

手順

1. JetBrains アカウントからアクティベーションコードを取得します。



JetBrains は、2 種類のアクティベーションコードと共に ZIP アーカイブを提供します。**<License ID> - for 2018.1 or later.txt**ファイルを使用します。



2. Che でオフラインで使用するためのアクティベーションコードをプロビジョニングします。この手順は、OpenShift のシークレットを使用して実行します。

3. OpenShift シークレットを作成し、CodeReady Workspaces に対してアクティベーションコードを JetBrains 固有の製品に基づいてコンテナーにマウントするよう指示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret name> (1)
  labels:
    app.kubernetes.io/component: workspace-secret
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/automount-workspace-secret: 'false'
    che.eclipse.org/mount-path: /tmp/
    che.eclipse.org/mount-as: file
data:
  <product name (idealU or WebStorm)>.key: <base64-encoded data content> (2)
```

(1) **<secret name>** – シークレット名を指定するセクション。この名前が異なる可能性があります（例: **ideaiu-offline-activation-code**）。シークレット名を小文字で指定します。

(2) 製品名とアクティベーションコード:

- **<product name (idealU or WebStorm)>** – JetBrains 製品名に置き換えます。「JetBrains の製品と名前のマッピング」 セクションを参照してください。
- **<base64-encoded data content>** – base64 でエンコードされたアクティベーションコードの内容。

false に設定された **automount-workspace-secret** オプションを使用して、**automountWorkspaceSecrets:true** プロパティーを使用して devfile コンポーネントで明示的に要求されるまでマウントプロセスを無効にします。上記の **workspace.yaml** サンプルファイルを参照してください。これは、連携が必要な特定のコンテナーがある場合を除き、すべてのコンテナーにアクティベーションコードをマウントしないようにするデフォルトの動作です。

その結果、Che Editor では、オフラインで使用するためのアクティベーションコードを持つファイルが、**/tmp/idealU.key** または **/tmp/WebStorm.key** パス (またはビルドのタイプに基づくこれと同様のパス) にマウントされます。

IntelliJ Idea Community Edition には、この手順は必要ありません。これは、登録が必要な JetBrains 製品に対して実行する必要があります。

4.6.4.1. JetBrains の製品と名前のマッピング

本セクションでは、JettyBrains 製品とイメージビルド時の製品名間の内部で使用されるマッピングを提供します。

JetBrains 製品	PRODUCT_NAME
IntelliJ Idea Community Edition	**idealC**
IntelliJ Idea Ultimate Edition	**idealU**

JetBrains 製品	PRODUCT_NAME
WebStorm	**WebStorm**

4.7. ワークスペースの作成後にツールを CODEREADY WORKSPACES に追加する

ワークスペースにインストールすると、CodeReady Workspaces プラグインは新機能を CodeReady Workspaces に追加します。プラグインは、Che-Theia プラグイン、メタデータ、およびホストコンテナーで構成されます。これらのプラグインは以下の機能を提供する場合があります。

- OpenShift を含む他のシステムとの統合
- 一部の開発者タスク (フォーマット、リファクタリング、自動化されたテストの実行など) の自動化。
- IDE から直接行う複数データベースとの通信。
- コードナビゲーション、自動補完、およびエラーの強調表示の強化。

本章では、ワークスペースでの CodeReady Workspaces プラグインのインストール、有効化、および使用に関する基本的な情報を提供します。

- 「CodeReady Workspaces ワークスペースの追加のツール」
- 「CodeReady Workspaces ワークスペースへの言語サポートプラグインの追加」

4.7.1. CodeReady Workspaces ワークスペースの追加のツール

CodeReady Workspaces プラグインは、コンテナーイメージと共にバンドルされる Che-Theia IDE の拡張機能です。これらのイメージには、それぞれの拡張機能のネイティブの前提条件が含まれます。たとえば、oc コマンドラインツールは、インストール用のコマンドと共にバンドルされます。これにより、OpenShift Connector プラグインの適切な機能がすべて専用のイメージで利用可能になります。

プラグインには、説明、分類タグ、およびアイコンを定義するメタデータを含めることもできます。CodeReady Workspaces は、ユーザーのワークスペースへのインストールに使用できるプラグインのレジストリーを提供します。

Che-Theia IDE は通常、VS Code 拡張機能 API と互換性があり、VS Code 拡張機能は Che-Theia と自動的に互換性を持ちます。これらの拡張機能は、依存関係と組み合わすことで CodeReady Workspaces プラグインとしてパッケージ化できます。デフォルトで、CodeReady Workspaces には共通のプラグインが含まれるプラグインレジストリーが含まれます。

プラグインの追加

- Dashboard の使用
  - **Workspace details** ページの **Plugins** タブを使用して、プラグインレジストリーからプラグインを追加します。
  - **Devfile** タブを使用して、プラグインを devfile に直接追加します。
    devfile は、メモリーや CPU の使用の定義など、プラグインを追加で設定することもできます。

- Che-Theia IDE の使用:
  - **Ctrl**+**Shift**+**J** を押すか、または **View** → **Plugins** に移動します。

関連資料

- 「devfile へのコンポーネントの追加」

4.7.2. CodeReady Workspaces ワークスペースへの言語サポートプラグインの追加

この手順では、Dashboard から専用のプラグインを有効にして、ツールを既存のワークスペースに追加する方法を説明します。

プラグインとして使用できるツールを CodeReady Workspaces ワークスペースに追加するには、以下のいずれかの方法を使用します。

- Dashboard **Plugins** タブからプラグインを有効にします。
- Dashboard の **Devfile** タブからワークスペースの devfile を編集します。

この手順では、例として Language Support for Java プラグインを使用します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペース。以下を参照してください。
  - 「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」
  - 「User Dashboard のカスタム Workspace ビューからのワークスペースの作成」
- ワークスペースは **停止** 状態でなければなりません。ワークスペースを停止するには、以下を実行します。
  a. CodeReady Workspaces Dashboard に移動します。「Dashboard を使用した CodeReady Workspaces のナビゲーション」 を参照してください。
  b. **Dashboard** で **Workspaces** メニューをクリックし、ワークスペース一覧を開き、ワークスペースを見つけます。
  c. 画面右側の表示されるワークスペースと同じ行で、四角の **Stop** ボタンをクリックしてワークスペースを停止します。
  d. ワークスペースが停止するまで数秒間待機し (一覧のワークスペースのアイコンがグレー表示になります)、クリックしてワークスペースを設定します。

手順

プラグインをプラグインレジストリーから既存の CodeReady Workspaces ワークスペースに追加するには、以下のいずれかの方法を使用します。

- **Plugins** タブからのプラグインのインストール。

1. **Plugins** タブに移動します。利用可能なプラグインの一覧が表示されます。
2. **Enable** スライドトグルを使用して、必要なプラグイン（Language Support for Java 11 など) を有効にします。これにより、プラグインの ID がワークスペースの devfile に追加され、プラグインが有効になります。
3. 画面右下の **Save** ボタンをクリックして変更を保存します。変更が保存されたら、ワークスペースを再起動でき、これには新しいプラグインが追加されます。

- devfile にコンテンツを追加して、プラグインをインストールする。

1. **Devfile** タブに移動します。devfile YAML が表示されます。
2. devfile の **components** セクションを見つけ、以下の行を追加して、Java 8 を使用する Java 言語プラグインをワークスペースに追加します。

```
- id: redhat/java8/latest
  type: chePlugin
```

最終的な結果の例:

```
components:
 - id: redhat/php/latest
   memoryLimit: 1Gi
   type: chePlugin
 - id: redhat/php-debugger/latest
   memoryLimit: 256Mi
   type: chePlugin
 - mountSources: true
   endpoints:
     - name: 8080/tcp
       port: 8080
   memoryLimit: 512Mi
   type: dockerimage
   volumes:
     - name: composer
       containerPath: {prod-home}/.composer
     - name: symfony
       containerPath: {prod-home}/.symfony
   alias: php
   image: 'quay.io/eclipse/che-php-7:nightly'
 - id: redhat/java8/latest
   type: chePlugin
```

3. 画面右下の **Save** ボタンをクリックして変更を保存します。変更が保存されたら、ワークスペースを再起動でき、これには新しいプラグインが追加されます。

関連資料

- devfile の仕様

4.8. ランタイム時に DEVFILE およびプラグインの編集

カスタムレジストリーイメージをビルドする代わりに、以下を実行します。

1. レジストリーの開始
2. ランタイム時に内容を変更します。

このアプローチはより簡単で高速です。ただし、コンテナーが削除されるとすぐに変更が失われます。

4.8.1. 実行時のプラグインの追加

手順

プラグインを追加するには、以下を実行します。

1. プラグインレジストリーソースをチェックアウトします。

```
$ git clone https://github.com/redhat-developer/codeready-workspaces; \
  cd codeready-workspaces/dependencies/che-plugin-registry
```

2. 一部のローカルディレクトリーに **meta.yaml** を作成します。これは、ゼロから実行することも、既存のプラグインの **meta.yaml** ファイルからコピーして実行することもできます。

```
$ PLUGIN="v3/plugins/new-org/new-plugin/0.0.1"; \
  mkdir -p ${PLUGIN}; cp v3/plugins/che-incubator/cpptools/0.1/* ${PLUGIN}/
  echo "${PLUGIN##*/}" > ${PLUGIN}/../latest.txt
```

3. 既存のプラグインからコピーする場合には、必要に応じて **meta.yaml** ファイルに変更を加えます。新規プラグインに一意の **title**、**displayName**、および **description** があることを確認します。**firstPublicationDate** を今日の日付に更新します。

4. **meta.yaml** のこれらのフィールドは、上記の **PLUGIN** で定義されたパスと一致する必要があります。

```
publisher: new-org
name: new-plugin
version: 0.0.1
```

5. プラグインレジストリーコンテナーをホストする Pod の名前を取得します。これを実行するには、**component=plugin-registry** ラベルをフィルターします。

```
$ PLUGIN_REG_POD=$(oc get -o custom-columns=NAME:.metadata.name \
  --no-headers pod -l component=plugin-registry)
```

6. レジストリーの **index.json** ファイルを再生成し、新規プラグインを追加します。

```
$ cd codeready-workspaces/dependencies/che-plugin-registry; \
    "$(pwd)/build/scripts/generate_latest_metas.sh" v3 && \
    "$(pwd)/build/scripts/check_plugins_location.sh" v3 && \
    "$(pwd)/build/scripts/set_plugin_dates.sh" v3 && \
    "$(pwd)/build/scripts/check_plugins_viewer_mandatory_fields.sh" v3 && \
    "$(pwd)/build/scripts/index.sh" v3 > v3/plugins/index.json
```

7. 新しい **index.json** および **meta.yaml** ファイルを新しいローカルプラグインフォルダーからコンテナーにコピーします。

```
$ cd codeready-workspaces/dependencies/che-plugin-registry; \
```

```
LOCAL_FILES="$(pwd)/${PLUGIN}/meta.yaml $(pwd)/v3/plugins/index.json"; \
oc exec ${PLUGIN_REG_POD} -i -t -- mkdir -p /var/www/html/$/{PLUGIN}; \
for f in $LOCAL_FILES; do e=${f/$(pwd)\/}; echo "Upload ${f} -> /var/www/html/${e}"; \
  oc cp "${f}" ${PLUGIN_REG_POD}:/var/www/html/${e}; done
```

8. 新規プラグインは、プラグインレジストリーの既存の CodeReady Workspaces インスタンスから使用できるようになりました。これを検出するには、CodeReady Workspaces ダッシュボードに移動してから、**Workspace**リンクをクリックします。そこから、ギアアイコンをクリックしてワークスペースのいずれかを設定します。**Plugins** タブを選択して、利用可能なプラグインの更新された一覧を表示します。

4.8.2. 実行時の devfile の追加

手順

devfile を追加するには、以下を実行します。

1. devfile レジストリーソースを確認します。

```
$ git clone https://github.com/redhat-developer/codeready-workspaces; \
  cd codeready-workspaces/dependencies/che-devfile-registry
```

2. **devfile.yaml** および **meta.yaml** を一部のローカルフォルダーに作成します。これは、ゼロから実行することも、既存の devfile からコピーして実行することもできます。

```
$ STACK="new-stack"; \
  mkdir -p devfiles/${STACK}; cp devfiles/03_web-nodejs-simple/* devfiles/${STACK}/
```

3. 既存の devfile からコピーする場合は、ニーズに合わせて devfile に変更を加えます。新規 devfile に一意の **displayName** および **description** があることを確認します。

4. devfile レジストリーコンテナーをホストする Pod の名前を取得します。これを実行するには、**component=devfile-registry** ラベルをフィルターします。

```
$ DEVFILE_REG_POD=$(oc get -o custom-columns=NAME:.metadata.name \
  --no-headers pod -l component=devfile-registry)
```

5. レジストリーの **index.json** ファイルを再生成し、新規 devfile を追加します。

```
$ cd codeready-workspaces/dependencies/che-devfile-registry; \
  "$(pwd)/build/scripts/check_mandatory_fields.sh" devfiles; \
  "$(pwd)/build/scripts/index.sh" > index.json
```

6. 新規ローカルの devfile フォルダーから、新しい **index.json**、**devfile.yaml**、および **meta.yaml** ファイルをコンテナーにコピーします。

```
$ cd che-devfile-registry; \
  oc exec ${DEVFILE_REG_POD} -i -t -- mkdir -p /var/www/html/devfiles/${STACK}; \
  oc cp $(pwd)/devfiles/${STACK}/meta.yaml
${DEVFILE_REG_POD}:/var/www/html/devfiles/${STACK}/meta.yaml; \
  oc cp $(pwd)/devfiles/${STACK}/devfile.yaml
${DEVFILE_REG_POD}:/var/www/html/devfiles/${STACK}/devfile.yaml; \
  oc cp $(pwd)/index.json ${DEVFILE_REG_POD}:/var/www/html/devfiles/index.json
```

7. 新しい devfile は、devfile レジストリーの既存の CodeReady Workspaces インスタンスから使用できるようになりました。これを検出するには、CodeReady Workspaces ダッシュボードに移動してから、**Workspace**リンクをクリックします。そこから、**Add Workspace** をクリックし、利用可能な devfile の更新された一覧を表示します。

第5章 OAUTH 認証の設定

本セクションでは、Red Hat CodeReady Workspaces を OAuth アプリケーションとしてサポートされる OAuth プロバイダーに接続する方法を説明します。

- xref:[]
- 「OpenShift OAuth の設定」

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。

手順

- GitHub OAuth クライアント を設定します。**Authorization コールバック URL** は、次の手順で入力されます。
  1. RH-SSO 管理コンソールに移動し、**Identity Providers** タブを選択します。
  2. ドロップダウンリストで **GitHub** アイデンティティープロバイダーを選択します。
  3. **リダイレクト URL** を GitHub OAuth アプリケーションの **認証コールバック URL** に貼り付けます。
  4. GitHub oauth アプリケーションから **Client ID** および **Client Secret** を入力します。
  5. **repo,user,write:public_key** を Default Scopes フィールドに貼り付けます。
  6. **Store Tokens** を有効にします。
  7. Github アイデンティティープロバイダーの変更を保存し、GitHub oauth app ページで **Register application** をクリックします。

5.2. 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。

8章OpenShift Connector の概要 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ようにする方法になります。

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。

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/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#using-the-crwctl-management-tool_crw を参照してください。

- OpenShift アイデンティティープロバイダーがクラスターで設定されます。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。

手順

- OpenShift OAuth はデフォルトで有効になり、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** 仕様の章を参照してください。

第6章 制限された環境でのアーティファクトリポジトリーの使用

本セクションでは、自己署名された証明書を使用して、社内のリポジトリーのアーティファクトと連携するように各種のテクノロジースタックを手動で設定する方法を説明します。

- 「Maven アーティファクトリポジトリーの使用」
- 「Gradle アーティファクトリポジトリーの使用」
- 「Python アーティファクトリポジトリーの使用」
- 「Go アーティファクトリポジトリーの使用」
- 「NuGet アーティファクトリポジトリーの使用」
- 「npm アーティファクトリポジトリーの使用」

6.1. MAVEN アーティファクトリポジトリーの使用

Maven は、2 つの場所で定義されているアーティファクトをダウンロードします。

- プロジェクトの **pom.xml** ファイルで定義されるアーティファクトリポジトリー。**pom.xml** でのリポジトリーの設定は、Red Hat CodeReady Workspaces に固有のものではありません。詳細は、POM に関する Maven ドキュメント を参照してください。
- **settings.xml** ファイルで定義されるアーティファクトリポジトリー。デフォルトでは、**settings.xml** は `~/**.m2/settings.xml** にあります。

6.1.1. settings.xmlでのリポジトリーの定義

example.server.org で独自のアーティファクトリポジトリーを指定するには、**settings.xml** ファイルを使用します。これを実行するには、**settings.xml** がとくに Maven コンテナーおよび Java プラグインコンテナーで、Maven ツールを使用するすべてのコンテナーにあることを確認します。

デフォルトで、**settings.xml** は、Maven および Java プラグインコンテナーの永続ボリューム上にある **<home dir>/.m2** ディレクトリーにあり、一時モードではない場合はワークスペースを再起動するたびにファイルを再作成する必要はありません。

Maven ツールを使用する別のコンテナーがあり、このコンテナーで **<home dir>/.m2** フォルダーを共有する場合は、devfile でこの特定のコンポーネントのカスタムボリュームを指定する必要があります。

```
apiVersion: 1.0.0
metadata:
  name: MyDevfile
components:
  - type: chePlugin
    alias: maven-tool
    id: plugin/id
    volumes:
    - name: m2
      containerPath: <home dir>/.m2
```

手順

1. **example.server.org** でアーティファクトリポジトリーを指定するには、**settings.xml** ファイルを使用します。

```
<settings>
  <profiles>
    <profile>
      <id>my-nexus</id>
      <pluginRepositories>
        <pluginRepository>
          <id>my-nexus-snapshots</id>
          <releases>
            <enabled>false</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-snapshots/</url>
        </pluginRepository>
        <pluginRepository>
          <id>my-nexus-releases</id>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-releases/</url>
        </pluginRepository>
      </pluginRepositories>
      <repositories>
        <repository>
          <id>my-nexus-snapshots</id>
          <releases>
            <enabled>false</enabled>
          </releases>
          <snapshots>
            <enabled>true</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-snapshots/</url>
        </repository>
        <repository>
          <id>my-nexus-releases</id>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
          <url>http://example.server.org/repository/maven-releases/</url>
        </repository>
      </repositories>
    </profile>
  </profiles>
  <activeProfiles>
```

```
    <activeProfile>my-nexus</activeProfile>
  </activeProfiles>
</settings>
```

6.1.2. ワークスペース全体での Maven settings.xml ファイルの定義

すべてのワークスペースで独自の **settings.xml** ファイルを使用するには、ワークスペースと同じプロジェクトに (任意の名前の) Secret オブジェクトを作成します。Secret のデータセクションに、必要な **settings.xml** の内容を追加します (同じディレクトリーに存在するはずの他のファイルも含まれる可能性があります)。「シークレットをファイルとしてワークスペースコンテナーにマウントする」 に従ってこの Secret にラベルおよびアノテーションを付けます。これにより、Secret の内容はワークスペース Pod にマウントされます。この Secret を使用するには、それ以前に実行されているワークスペースをすべて再起動する必要があります。

前提条件

これは、プライベート認証情報を Maven リポジトリーに設定するために必要です。詳細は、Maven ドキュメントの Settings.xml#Servers を参照してください。

この **settings.xml** をマウントするには、以下を実行します。

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                              https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>repository-id</id>
      <username>username</username>
      <password>password123</password>
    </server>
  </servers>
</settings>
```

手順

1. **settings.xml** を base64 に変換します。

```
$ cat settings.xml | base64
```

2. 出力を新規ファイル **secret.yaml** にコピーし、必要なアノテーションおよびラベルも定義します。

```
apiVersion: v1
kind: Secret
metadata:
  name: maven-settings-secret
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: workspace-secret
  annotations:
    che.eclipse.org/automount-workspace-secret: true
    che.eclipse.org/mount-path: /home/jboss/.m2
    che.eclipse.org/mount-as: file
```

```
type: Opaque
data:
  settings.xml:
PHNldHRpbmdzIHhtbG5zPSJodHRwOi8vbWF2ZW4uYXBhY2hlLm9yZy9TRVRUSU5HUy8xLj
AuMCIKICAgICAgICAgIHhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2
hlbWEtaW5zdGFuY2UiCiAgICAgICAgICB4c2k6c2NoZW1hTG9jYXRpb249Imh0dHA6Ly9tYXZl
bi5hcGFjaGUub3JnL1NFVFRJTkdTLzEuMC4wCiAgICAgICAgICAgICAgICAgICAgICAgICAgI
CAgIGh0dHBzOi8vbWF2ZW4uYXBhY2hlLm9yZy94c2Qvc2V0dGluZ3MtMS4wLjAueHNkIj4KIC
A8c2VydmVycz4KICAgIDxzZXJ2ZXI+CiAgICAgIDxpZD5zZXBvc2l0b3J5LWlkPC9pZD4KI
CAgPHVzZXJuYW1lPnVzZXJuYW1lPC91c2VybmFtZT4KICAgICAgPHBhc3N3b3JkPnBhc3N3
b3JkMTIzPC9wYXNzd29yZD4KICAgIDwvc2VydmVyPgogIDwvc2VydmVycz4KPC9zZXR0aW5
ncz4K
```

3. このシークレットをクラスターに作成します。

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

4. 新規ワークスペースを起動します。**/home/jboss/.m2/settings.xml** と **maven** コンテナーの元のコンテンツが表示されます。

6.1.2.1. OpenShift 3.11 および OpenShift <1.13

OpenShift 3.11 では、同じパスに複数のボリュームマウントを含めることができないため、ボリュームのある devfile を **/home/jboss/.m2** におき、シークレットを **/home/jboss/.m2/settings.xml** に置くことで競合を解決できます。これらのクラスターでは、**/home/jboss/.m2/repository** が devfile の maven リポジトリーのボリュームとして使用されます。

```
apiVersion: 1.0.0
metadata:
  ...
components:
 - type: dockerimage
   alias: maven
   image: maven:3.11
   volumes:
     - name: m2
       containerPath: /home/jboss/.m2/repository
   ...
```

6.1.3. Maven プロジェクトでの自己署名証明書の使用

内部アーティファクトリポジトリーには、多くの場合、Java でデフォルトで信頼される認証局によって署名された証明書がありません。通常は、企業内の認証局によって署名されるか、または自己署名されます。これらの証明書を Java トラストストアに追加して、これらを受け入れるようにツールを設定します。

手順

1. リポジトリーサーバーからサーバー証明書ファイルを取得します。通常、管理者は内部アーティファクトサーバーの証明書を OpenShift シークレットとして提供する必要があります (https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#importing-untrusted-tls-certificates_crw を参照してください)。関連するサーバー証明書は、ワークスペース内のすべてのコンテナーの **/public-certs** にマウントされます。

a. 元の Java トラストストアファイルをコピーします。

```
$ mkdir /projects/maven
$ cp $JAVA_HOME/lib/security/cacerts /projects/maven/truststore.jks
$ chmod +w /projects/maven/truststore.jks
```

b. 証明書の Java トラストストアファイルへのインポート

```
$ keytool -import -noprompt -file /public-certs/nexus.cer -alias nexus -keystore
/projects/maven/truststore.jks -storepass changeit
Certificate was added to keystore
```

2. トラストストアファイルを追加します。

- Maven コンテナーで以下を実行します。

a. **javax.net.ssl** システムプロパティーを **MAVEN_OPTS** 環境変数に追加します。

```
 - mountSources: true
   alias: maven
   type: dockerimage
   ...
   env:
     -name: MAVEN_OPTS
      value: >-
        -Duser.home=/projects/maven -
Djavax.net.ssl.trustStore=/projects/maven/truststore.jks -
Djavax.net.ssl.trustStorePassword=changeit
```

b. ワークスペースを再起動します。

- Java プラグインコンテナーで以下を実行します。
  devfile で、Java 言語サーバーの **javax.net.ssl** システムプロパティーを追加します。

```
components:
  - id: redhat/java11/latest
    type: chePlugin
    preferences:
      java.jdt.ls.vmargs: >-
        -noverify -Xmx1G -XX:+UseG1GC -XX:+UseStringDeduplication
        -Duser.home=/projects/maven
        -Djavax.net.ssl.trustStore=/projects/maven/truststore.jks
        -Djavax.net.ssl.trustStorePassword=changeit
[...]
```

6.2. GRADLE アーティファクトリポジトリーの使用

6.2.1. 各種バージョンの Gradle のダウンロード

任意のバージョンの Gradle をダウンロードする方法として、Gradle Wrapper スクリプトを使用することが推奨されます。プロジェクトに **gradle/wrapper** ディレクトリーがない場合は、**$ gradle wrapper** を実行して Wrapper を設定します。

前提条件

- Gradle Wrapper がプロジェクトに存在すること。

手順

標準以外の場所から Gradle バージョンをダウンロードするには、**/projects/<your_project>/gradle/wrapper/gradle-wrapper.properties** で Wrapper 設定を変更します。

- **distributionUrl** プロパティーを、Gradle ディストリビューション ZIP ファイルの URL を参照するように変更します。

```
properties
distributionUrl=http://<url_to_gradle>/gradle-6.1-bin.zip
```

または、Gradle ディストリビューションの zip ファイルをワークスペースの **/project/gradle** にローカルに配置できます。

- **distributionUrl** プロパティーを、Gradle ディストリビューション zip ファイルの URL を参照するように変更します。

```
properties
distributionUrl=file\:/projects/gradle/gradle-6.1-bin.zip
```

6.2.2. グローバル Gradle リポジトリーの設定

初期化スクリプトを使用して、ワークスペースのグローバルリポジトリーを設定します。Gradle は、プロジェクトが評価される前に追加の設定を実行し、この設定はワークスペースから各 Gradle プロジェクトで使用されます。

手順

ワークスペース内の各 Gradle プロジェクトで使用できる Gradle のグローバルリポジトリーを設定するには、**~/.gradle/** ディレクトリーに **init.gradle** スクリプトを作成します。

```
allprojects {
  repositories {
    mavenLocal ()
    maven {
      url "http://repo.mycompany.com/maven"
      credentials {
        username "admin"
        password "my_password"
      }
    }
  }
}
```

このファイルは、Gradle を、指定の認証情報でローカルの Maven リポジトリーを使用するように設定します。

注記

~/**.gradle** ディレクトリーは現在の Java プラグインバージョンで維持されないため、Java プラグインサイドカーコンテナーで毎回のワークスペースの起動時に **init.gradle** スクリプトを作成する必要があります。

6.2.3. Gradle プロジェクトでの自己署名証明書の使用

内部アーティファクトリポジトリーには、多くの場合、Java でデフォルトで信頼される認証局によって署名された証明書がありません。通常は、企業内の認証局によって署名されるか、または自己署名されます。これらの証明書を Java トラストストアに追加して、これらを受け入れるようにツールを設定します。

手順

1. リポジトリーサーバーからサーバー証明書ファイルを取得します。通常、管理者は内部アーティファクトサーバーの証明書を OpenShift シークレットとして提供する必要があります (https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#importing-untrusted-tls-certificates_crw を参照してください)。関連するサーバー証明書は、ワークスペース内のすべてのコンテナーの **/public-certs** にマウントされます。

   a. 元の Java トラストストアファイルをコピーします。

   ```
   $ mkdir /projects/maven
   $ cp $JAVA_HOME/lib/security/cacerts /projects/maven/truststore.jks
   $ chmod +w /projects/maven/truststore.jks
   ```

   b. 証明書の Java トラストストアファイルへのインポート

   ```
   $ keytool -import -noprompt -file /public-certs/nexus.cer -alias nexus -keystore
   /projects/maven/truststore.jks -storepass changeit
   Certificate was added to keystore
   ```

   c. トラストストアファイルを **/projects/gradle/truststore.jks** にアップロードし、これをすべてのコンテナーで利用可能にします。

2. Gradle コンテナーにトラストストアファイルを追加します。

   a. **javax.net.ssl** システムプロパティーを **JAVA_OPTS** 環境変数に追加します。

   ```
   - mountSources: true
     alias: maven
     type: dockerimage
     ...
     env:
       -name: JAVA_OPTS
        value: >-
          -Duser.home=/projects/gradle
          -Djavax.net.ssl.trustStore=/projects/maven/truststore.jks
          -Djavax.net.ssl.trustStorePassword=changeit
   ```

関連資料

- 初期化スクリプトについての Gradle ドキュメント

- Gradle Wrapper ドキュメント

6.3. PYTHON アーティファクトリポジトリーの使用

6.3.1. 標準以外のレジストリーを使用するように Python を設定する

Python pip ツールで使用する標準以外のリポジトリーを指定するには、**PIP_INDEX_URL** 環境変数を設定します。

手順

- devfile で、言語サポートおよびコンテナーのコンポーネント用に **PIP_INDEX_URL** 環境変数を設定します。

```
- id: ms-python/python/latest
  memoryLimit: 512Mi
  type: chePlugin
  env:
    - name: 'PIP_INDEX_URL'
      value: 'https://<username>:<password>@pypi.company.com/simple'
- mountSources: true
  memoryLimit: 512Mi
  type: dockerimage
  alias: python
  image: 'registry.redhat.io/codeready-workspaces/plugin-java8-rhel8:2.5'
  env:
    - name: 'PIP_INDEX_URL'
      value: 'https://<username>:<password>@pypi.company.com/simple'
```

6.3.2. Python プロジェクトでの自己署名証明書の使用

内部アーティファクトリポジトリーには、デフォルトで信頼される認証局によって署名された自己署名 TLS 証明書がありません。通常は、企業内の認証局によって署名されるか、または自己署名されます。これらの証明書を受け入れるようにツールを設定します。

Python は、**PIP_CERT** 環境変数で定義されるファイルから証明書を使用します。

手順

1. pip サーバーで使用する証明書を Privacy-Enhanced Mail (PEM) 形式で取得します。通常、管理者は内部アーティファクトサーバーの証明書を OpenShift シークレットとして提供する必要があります (https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#importing-untrusted-tls-certificates_crw を参照してください)。関連するサーバー証明書は、ワークスペース内のすべてのコンテナーの **/public-certs** にマウントされます。

注記

pip は、Privacy-Enhanced Mail (PEM) 形式の証明書のみを受け入れます。必要に応じて、OpenSSL を使用して証明書を PEM 形式に変換します。

2. devfile を設定します。

```
 - id: ms-python/python/latest
   memoryLimit: 512Mi
   type: chePlugin
   env:
     - name: 'PIP_INDEX_URL'
       value: 'https://<username>:<password>@pypi.company.com/simple'
     - value: '/projects/tls/rootCA.pem'
       name: 'PIP_CERT'
 - mountSources: true
   memoryLimit: 512Mi
   type: dockerimage
   alias: python
   image: 'registry.redhat.io/codeready-workspaces/plugin-java8-rhel8:2.5'
   env:
     - name: 'PIP_INDEX_URL'
       value: 'https://<username>:<password>@pypi.company.com/simple'
     - value: '/projects/tls/rootCA.pem'
       name: 'PIP_CERT'
```

6.4. GO アーティファクトリポジトリーの使用

制限された環境で Go を設定するには、**GOPROXY** 環境変数と Athens モジュールデータストアおよびプロキシーを使用します。

6.4.1. 標準以外のレジストリーを使用するように Go を設定する

Athens は Go モジュールデータストアおよびプロキシーであり、多くの設定オプションがあります。これは、モジュールデータストアとしてのみ動作するように設定でき、プロキシーとしては機能しません。管理者は Go モジュールを Athens データストアにアップロードし、それらを Go プロジェクト全体で利用可能にできます。ノードが Athens データストアにない Go モジュールにアクセスしようとすると、Go ビルドに失敗します。

- Athens と連携するには、CLI コンテナーの devfile で **GOPROXY** 環境変数を設定します。

```
components:
- mountSources: true
  type: dockerimage
  alias: go-cli
  image: 'quay.io/eclipse/che-golang-1.12:7.7.0'
  ...
  - value: /tmp/.cache
    name: GOCACHE
  - value: 'http://your.athens.host'
    name: GOPROXY
```

6.4.2. Go プロジェクトでの自己署名証明書の使用

内部アーティファクトリポジトリーには、デフォルトで信頼される認証局によって署名された自己署名 TLS 証明書がありません。通常は、企業内の認証局によって署名されるか、または自己署名されます。これらの証明書を受け入れるようにツールを設定します。

Go は、**SSL_CERT_FILE** 環境変数で定義されるファイルの証明書を使用します。

手順

1. Privacy-Enhanced Mail (PEM) 形式の Athens サーバーで使用される証明書を取得します。通常、管理者は内部アーティファクトサーバーの証明書を OpenShift シークレットとして提供する必要があります (https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#importing-untrusted-tls-certificates_crw を参照してください)。関連するサーバー証明書は、ワークスペース内のすべてのコンテナーの **/public-certs** にマウントされます。

2. 適切な環境変数を devfile に追加します。

```
components:
- mountSources: true
  type: dockerimage
  alias: go-cli
  image: 'registry.redhat.io/codeready-workspaces/stacks-golang-rhel8:2.5'
  ...
  - value: /tmp/.cache
    name: GOCACHE
  - value: 'http://your.athens.host'
    name: GOPROXY
  - value: '/projects/tls/rootCA.crt'
    name: SSL_CERT_FILE
```

関連資料

- GitHub - gomods/athens: Go モジュールデータストアおよびプロキシー

6.5. NUGET アーティファクトリポジトリーの使用

制限された環境で NuGet を設定するには、**nuget.config** ファイルを変更し、devfile で **SSL_CERT_FILE** 環境変数を使用して自己署名証明書を追加します。

6.5.1. 標準以外のアーティファクトリポジトリーを使用するように NuGet を設定する

NuGet は、ソリューションディレクトリーとドライバーのルートディレクトリー間で設定ファイルを検索します。**nuget.config** ファイルを **/projects** ディレクトリーに置くと、**nuget.config** ファイルは **/projects** 内のすべてのプロジェクトの NuGet の動作を定義します。

手順

- **nuget.config** ファイルを作成し、**/projects** ディレクトリーに置きます。

  nexus.example.org でホストされる Nexus リポジトリーを含む nuget.config の例:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <packageSources>
    <add key="nexus2" value="https://nexus.example.org/repository/nuget-hosted/"/>
  </packageSources>
  <packageSourceCredentials>
    <nexus2>
        <add key="Username" value="user" />
        <add key="Password" value="..." />
```

```
    </nexus2>
  </packageSourceCredentials>
</configuration>
```

6.5.2. NuGet プロジェクトでの自己署名証明書の使用

内部アーティファクトリポジトリーには、デフォルトで信頼される認証局によって署名された自己署名 TLS 証明書がありません。通常は、企業内の認証局によって署名されるか、または自己署名されます。これらの証明書を受け入れるようにツールを設定します。

手順

1. Privacy-Enhanced Mail (PEM) 形式で、.NET サーバーが使用する証明書を取得します。通常、管理者は内部アーティファクトサーバーの証明書を OpenShift シークレットとして提供する必要があります (https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#importing-untrusted-tls-certificates_crw を参照してください)。関連するサーバー証明書は、ワークスペース内のすべてのコンテナーの **/public-certs** にマウントされます。

2. OmniSharp プラグインおよび .NET について、devfile の **SSL_CERT_FILE** 環境変数で証明書ファイルの場所を指定します。

   devfile の例:

```
components:
  - id: redhat-developer/che-omnisharp-plugin/latest
    memoryLimit: 1024Mi
    type: chePlugin
    alias: omnisharp
    env:
      - value: /public-certs/nuget.cer
        name: SSL_CERT_FILE
  - mountSources: true
    endpoints:
      - name: 5000/tcp
        port: 5000
    memoryLimit: 512Mi
    type: dockerimage
    volumes:
      - name: dotnet
        containerPath: /home/jboss
    alias: dotnet
    image: 'quay.io/eclipse/che-dotnet-2.2:7.7.1'
    env:
      - value: /projects/tls/rootCA.crt
        name: SSL_CERT_FILE
```

6.6. NPM アーティファクトリポジトリーの使用

通常、npm は **npm config** コマンドを使用して設定され、**.npmrc** ファイルに値を書き込みます。ただし、設定値は、**NPM_CONFIG_** で始まる環境変数を使用して設定することもできます。

Red Hat CodeReady Workspaces で使用される Javascript/Typescript プラグインはアーティファクトをダウンロードしません。dev-machine コンポーネントで npm を設定するだけで十分です。

設定には、以下の環境変数を使用します。

- アーティファクトリポジトリーの URL は **NPM_CONFIG_REGISTRY** です。
- ファイルから証明書を使用する場合は、**NODE_EXTRA_CA_CERTS** を使用します。

リポジトリーサーバーからサーバー証明書ファイルを取得します。通常、管理者は内部アーティファクトサーバーの証明書を OpenShift シークレットとして提供する必要があります (https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#importing-untrusted-tls-certificates_crw を参照してください)。関連するサーバー証明書は、ワークスペース内のすべてのコンテナーの **/public-certs** にマウントされます。

1. 自己署名証明書で内部リポジトリーを使用する設定の例:

```
- mountSources: true
  endpoints:
    - name: nodejs
      port: 3000
  memoryLimit: '512Mi'
  type: 'dockerimage'
  alias: 'nodejs'
  image: 'quay.io/eclipse/che-nodejs10-ubi:nightly'
  env:
    -name: NODE_EXTRA_CA_CERTS
     value: '/public-certs/nexus.cer
   - name: NPM_CONFIG_REGISTRY
     value: 'https://snexus-airgap.apps.acme.com/repository/npm-proxy/'
```

第7章 CODEREADY WORKSPACES のトラブルシューティング

本セクションでは、ユーザーが競合する可能性のある最も頻繁に発生する問題についてのトラブルシューティング手順を説明します。

関連資料

- 「CodeReady Workspaces ワークスペースログの表示」
- 「速度の遅いワークスペースのトラブルシューティング」
- 「ネットワーク問題のトラブルシューティング」
- 「デバッグモードでの CodeReady Workspaces ワークスペースの起動」
- 「起動の失敗後のデバッグモードでの CodeReady Workspaces ワークスペースの再起動」

7.1. CODEREADY WORKSPACES ワークスペースログの表示

本セクションでは、CodeReady Workspaces ワークスペースログを表示する方法を説明します。

7.1.1. 言語サーバーおよびデバッグアダプターのログの表示

7.1.1.1. 重要なログの確認

本セクションでは、重要なログを確認する方法を説明します。

手順

1. OpenShift web コンソールで、**Applications → Pods** をクリックし、すべてのアクティブなワークスペースの一覧を表示します。
2. ワークスペースが実行されている実行中の Pod の名前をクリックします。Pod 画面には、追加情報が含まれるすべてのコンテナーの一覧が含まれます。
3. コンテナーを選択し、コンテナー名をクリックします。

注記

最も重要なログは、**theia-ide** コンテナーとプラグインコンテナーログです。

4. コンテナー画面で、**Logs** セクションに移動します。

7.1.1.2. メモリー問題の検出

本セクションでは、メモリー不足のプラグインに関連するメモリーの問題を検出する方法を説明します。以下は、メモリー不足のプラグインに関連する 2 つの最もよく見られる問題になります。

プラグインコンテナーがメモリー不足になる

これは、コンテナーにイメージのエントリーポイントを実行するのに十分な RAM がない場合に、プラグインの初期化時に発生する可能性があります。ユーザーはプラグインコンテナーのログでこれを検知できます。この場合、ログには **OOMKilled** が含まれます。これは、コンテナーのプロセスがコンテナーで利用可能な量よりも多くのメモリーを要求することを示唆します。

コンテナー内のプロセスがコンテナーの通知なしにメモリー不足になる

たとえば、Java 言語サーバー (vscode-java 拡張によって起動する Eclipse JDT Language Server) は **OutOfMemoryException** をスローします。これは、プラグインが言語サーバーを起動する際、または処理する必要があるプロジェクトのサイズによりプロセスがメモリー不足になる場合などに、コンテナーの初期化後いつでも発生する可能性があります。
この問題を検出するには、コンテナーで実行しているプライマリープロセスのログを確認します。たとえば、Eclipse JDT Language Server のログファイルで詳細を確認するには、関連するプラグイン固有のセクションを参照してください。

7.1.1.3. デバッグアダプター用のクライアントサーバーのトラフィックのロギング

本セクションでは、Che-Theia とデバッグアダプター間のやり取りのログを **Output** ビューに記録する方法を説明します。

前提条件

- **Debug アダプター** のオプションを一覧に表示するには、デバッグセッションを開始する必要があります。

手順

1. **File → Settings、Preferences** をクリックします。
2. **Preferences** ビューの **Debug** セクションを展開します。
3. **trace** 設定の値を **true** に設定します (デフォルトは**false**です)。
4. これで、すべての通信イベントがログに記録されます。
5. これらのイベントを確認するには、**View → Output** をクリックし、**Output** ビューの右上にあるドロップダウンリストから **Debug adapters** を選択します。

7.1.1.4. Python のログの表示

本セクションでは、Python 言語サーバーのログを表示する方法を説明します。

手順

- **Output** viewビューに移動し、ドロップダウンリストで **Python** を選択します。

7.1.1.5. Go のログの表示

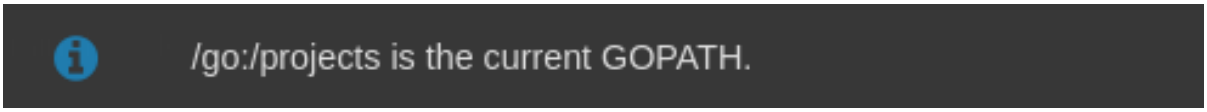
本セクションでは、Go 言語サーバーのログを表示する方法を説明します。

7.1.1.5.1. gopath の検索

本セクションでは、**GOPATH** 変数が参照する場所を見つける方法を説明します。

手順

- **Go: Current GOPATH**コマンドを実行します。



7.1.1.5.2. Go の Debug Console ログの表示

本セクションでは、Go デバッガーからのログ出力を表示する方法を説明します。

手順

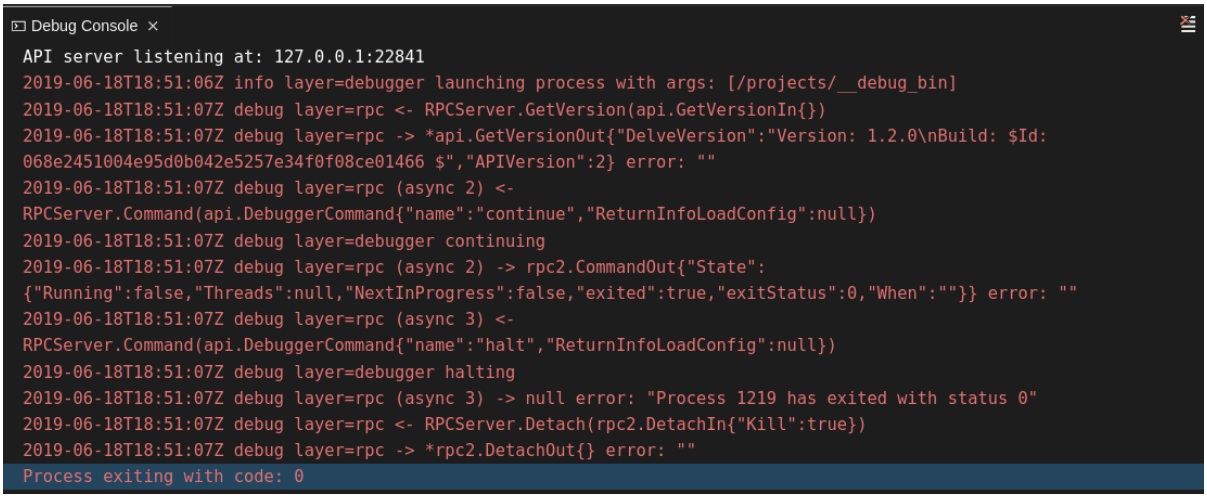
1. デバッグ設定で **showLog** 属性を **true** に設定します。

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
      "showLog": true
      ....
    }
  ]
}
```

2. コンポーネントのデバッグ出力を有効にするには、パッケージを **logOutput** 属性のコンマ区切りの一覧に追加します。

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
      "showLog": true,
      "logOutput": "debugger,rpc,gdbwire,lldbout,debuglineerr"
      ....
    }
  ]
}
```

3. デバッグコンソールは、デバッグコンソールに追加情報を出力します。



7.1.1.5.3. Output パネルでの Go ログ出力の表示

本セクションでは、**Output** パネルで Go ログ出力を表示する方法を説明します。

手順

1. **Output** ビューに移動します。
2. ドロップダウンリストで **Go** を選択します。

7.1.1.6. NodeDebug NodeDebug2 アダプターのログの表示

注記

一般的な診断以外の特定の診断はありません。

7.1.1.7. Typescript のログの表示

7.1.1.7.1. Label Switched Protocol (LSP) トレースの有効化

手順

1. Typescript (TS) サーバーに送信されるメッセージのトレースを有効にするには、**Preferences** ビューで **typescript.tsserver.trace** 属性を **verbose** に設定します。これを使用して、TS サーバーの問題を診断します。
2. TS サーバーのファイルへのロギングを有効にするには、**typescript.tsserver.log** 属性を **verbose** に設定します。このログを使用して、TS サーバーの問題を診断します。ログにはファイルパスが含まれます。

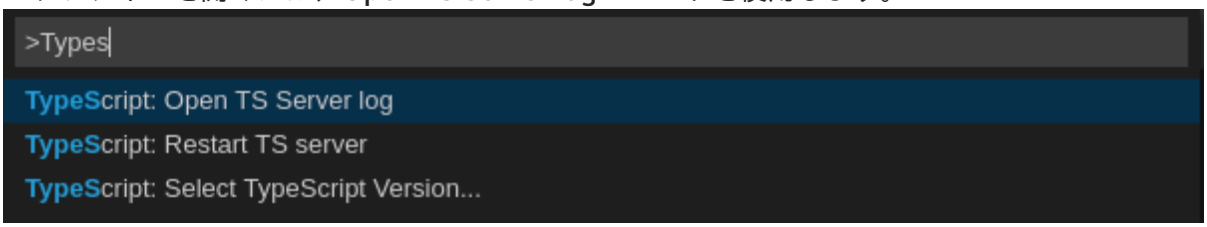
7.1.1.7.2. Typescript 言語サーバーログの表示

本セクションでは、Typescript 言語サーバーのログを表示する方法を説明します。

手順

1. ログファイルへのパスを取得するには、Typescript **Output** コンソールを参照してください。

2. ログファイルを開くには、**Open TS Server log**コマンドを使用します。

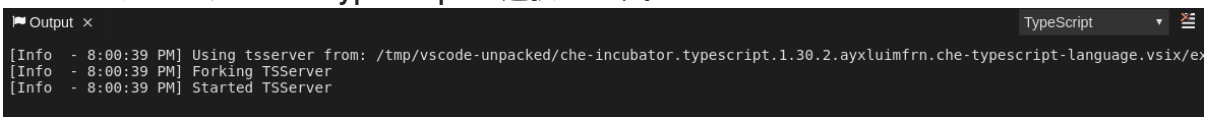


7.1.1.7.3. Output パネルでの Typescript ログ出力の表示

本セクションでは、**Output** パネルで Typescript ログの出力を表示する方法を説明します。

手順

1. **Output** ビューに移動します。
2. ドロップダウンリストで **TypeScript** を選択します。



7.1.1.8. Java のログの表示

一般的な診断以外に、ユーザーは Language Support for Java(Eclipse JDT Language Server) プラグインの各種アクションを実行できます。

7.1.1.8.1. Eclipse JDT Language Server の状態の確認

手順

Eclipse JDT Language Server プラグインを実行しているコンテナーが Eclipse JDT Language Server のメインプロセスを実行しているかどうかを確認します。

1. Eclipse JDT Language Server プラグインを実行しているコンテナーでターミナルを開きます (コンテナーのサンプル名: **vscode-javaxxx**)。
2. ターミナル内で **ps aux | grep jdt** コマンドを実行して、Eclipse JDT Language Server プロセスがコンテナーで実行されているかどうかを確認します。プロセスが実行されている場合、出力は以下のようになります。

```
usr/lib/jvm/default-jvm/bin/java --add-modules=ALL-SYSTEM --add-opens java.base/java.util
```

   このメッセージは、使用される VSCode Java 拡張機能も表示します。言語サーバーが実行されていない場合には、これはコンテナー内で起動していません。

3. 「重要なログの確認」で説明されているすべてのログを確認します。

7.1.1.8.2. Eclipse JDT Language Server 機能の確認

手順

Eclipse JDT Language Server プロセスを実行している場合は、言語サーバーの機能が機能しているかどうかを確認します。

1. Java ファイルを開き、ホバーや自動補完機能を使用します。誤りのあるファイルの場合、ユーザーはこの **Outline** ビューまたは **Problems** ビューで Java を確認できます。

7.1.1.8.3. Java 言語サーバーログの表示

手順

Eclipse JDT Language Server には、エラー、実行されたコマンド、およびイベントについてのログを記録する独自のワークスペースがあります。

1. このログファイルを開くには、Eclipse JDT Language Server プラグインを実行しているコンテナーでターミナルを開きます。**Java: Open Java Language Server log file**コマンドを実行してログファイルを表示することもできます。

2. **cat <PATH_TO_LOG_FILE>** を実行します。**PATH_TO_LOG_FILE** は **/home/theia/.theia/workspace-storage/<workspace_name>/redhat.java/jdt_ws/.metadata/.log** になります。

7.1.1.8.4. Java Language Server Protocol (LSP) メッセージのロギング

手順

LSP メッセージのログを VS Code **Output** ビューに記録するには、**java.trace.server** 属性を **verbose** に設定してトレースを有効にします。

関連資料

トラブルシューティングの手順については、 VS Code Java GitHub リポジトリー を参照してください。

7.1.1.9. Intelephense のログの表示

7.1.1.9.1. Intelephense クライアントサーバー通信のロギング

手順

PHP Intelephense 言語のサポートを、クライアントサイドのやり取りのログを **Output** ビューに記録するように設定するには、以下を実行します。

1. **File → Settings** をクリックします。

2. **Preferences** ビューを開きます。

3. **Intelephense** セクションを拡張し、**trace.server.verbose** 設定値を **verbose** に設定してすべての通信イベントを表示します (デフォルト値は **off** です)。

7.1.1.9.2. Output パネルでの Intelephense イベントの表示

この手順では、**Output** パネルで Intelephense イベントを表示する方法を説明します。

手順

1. **View → Output** をクリックします。
2. **Output** ビューのドロップダウンリストで **Intelephense** を選択します。

7.1.1.10. PHP-Debug のログの表示

この手順では、PHP Debug プラグインの診断メッセージのログを **Debug Console** ビューに記録するように PHP Debug プラグインを設定する方法を説明します。デバッグセッションの開始前にこれを設定します。

手順

1. **launch.json** ファイルで、**"log": true** 属性を選択した起動設定に追加します。
2. デバッグセッションを開始します。
3. 診断メッセージは、アプリケーションの出力と共に **Debug Console** ビューに出力されます。

7.1.1.11. XML のログの表示

一般的な診断以外に、ユーザーが実行できる XML プラグイン固有のアクションがあります。

7.1.1.11.1. XML 言語サーバーの状態の確認

手順

1. **vscode-xml-<xxx>** という名前のコンテナーでターミナルを開きます。
2. **ps aux | grep java** を実行して、XML 言語サーバーが起動していることを確認します。プロセスが実行されている場合、出力は以下のようになります。

   ```
   java ***/org.eclipse.ls4xml-uber.jar`
   ```

   そうでない場合は、「重要なログの確認」の章を参照してください。

7.1.1.11.2. XML 言語サーバーの機能フラグの確認

手順

1. 機能が有効にされているかどうかを確認します。XML プラグインは、機能を有効かつ無効にできる複数の設定を提供します。
   - **xml.format.enabled**: フォーマッターを有効にします。
   - **xml.validation.enabled**: 検証を有効にします。
   - **xml.documentSymbols.enabled**: ドキュメントシンボルを有効にします。
2. XML 言語サーバーが機能しているかどうかを確認するには、**<hello></hello>** などの単純な XML 要素を作成し、右側の **Outline** パネルに表示されることを確認します。
3. ドキュメントのシンボルが表示されない場合は、**xml.documentSymbols.enabled** 属性が **true** に設定されていることを確認します。**true** の場合でシンボルがない場合は、言語サーバーはエディターにフックされない可能性があります。ドキュメントのシンボルがある場合は、言語サーバーがエディターに接続されます。

4. ユーザーが必要とする機能が、設定の **true** に設定されていることを確認します (デフォルトでは **true** に設定されます)。機能のいずれかが機能していないか、または予想通りに機能しない場合は、言語サーバー に対する問題を報告します。

7.1.1.11.3. XML Language Server Protocol (LSP) トレースの有効化

手順

LSP メッセージのログを VS Code **Output** ビューに記録するには、**xml.trace.server** 属性を **verbose** に設定してトレースを有効にします。

7.1.1.11.4. XML 言語サーバーログの表示

手順

言語サーバーのログは、**/home/theia/.theia/workspace-storage/<workspace_name>/redhat.vscode-xml/lsp4xml.log** のプラグインサイドカーコンテナーにあります。

7.1.1.12. YAML のログの表示

本セクションでは、一般的な診断のほかに、ユーザーが実行できる YAML プラグインに固有のアクションについて説明します。

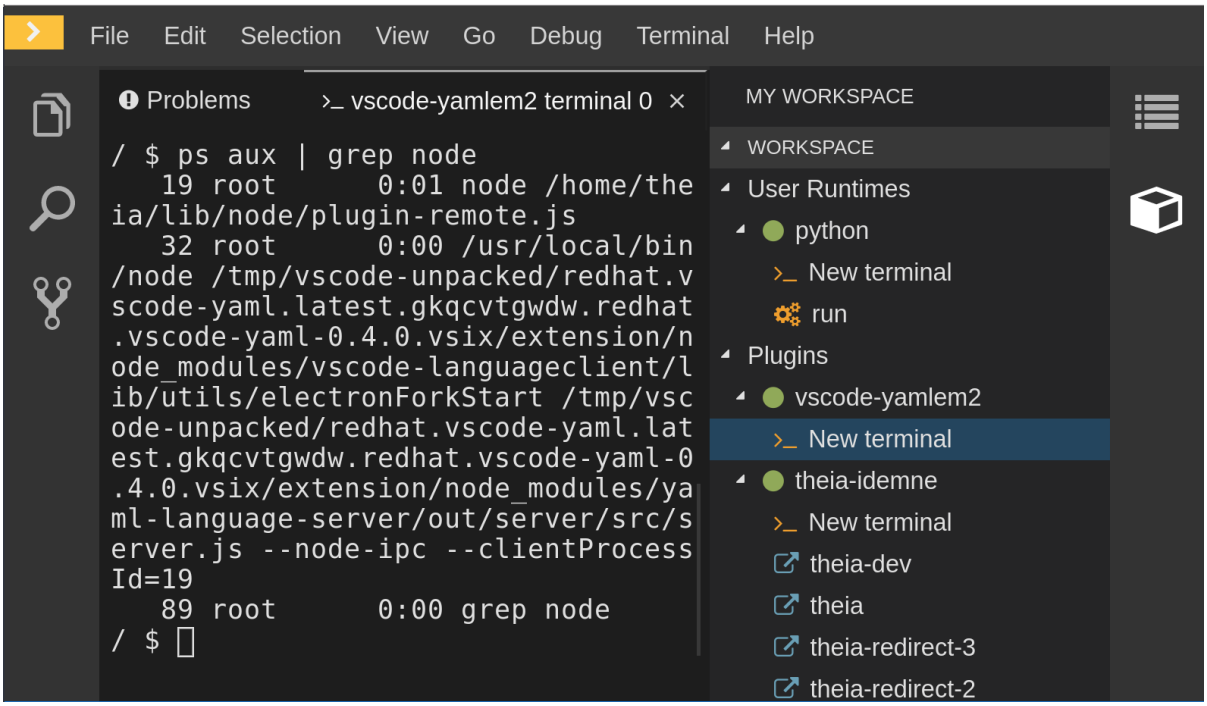
7.1.1.12.1. YAML 言語サーバーの状態の確認

本セクションでは、YAML 言語サーバーの状態を確認する方法を説明します。

手順

YAML プラグインを実行するコンテナーが YAML 言語サーバーを実行しているかどうかを確認します。

1. エディターを使用し、YAML プラグインを実行しているコンテナーでターミナルを開きます (コンテナーの名前の例: **vscode-yaml-<xxx>**)。
2. ターミナルで **ps aux | grep node** コマンドを実行します。このコマンドは、現在のコンテナーで実行されているすべてのノードプロセスを検索します。
3. コマンド **node **/server.js が実行していることを確認します。**



コンテナーで実行している node **/server.js は、言語サーバーが実行していることを示します。これが実行されていない場合、言語サーバーはコンテナー内で起動していません。この場合は、重要なログの確認を参照してください。

7.1.1.12.2. YAML 言語サーバーの機能フラグの確認

手順

機能フラグを確認するには、以下を実行します。

1. 機能が有効にされているかどうかを確認します。YAML プラグインは、以下のような機能を有効および無効にできる複数の設定を提供します。
   - **xml.format.enabled**: フォーマッターを有効にします。
   - **yaml.validate**: 検証を有効にします。
   - **yaml.hover**: ホバー機能を有効にします。
   - **yaml.completion**: 補完 (completion) 機能を有効にします。
2. プラグインが機能しているかどうかを確認するには、**hello: world** などの最も簡単な YAML を入力してから、エディターの右側にある Outline パネルを開きます。
3. ドキュメントのシンボルがあるかどうかを確認します。yes の場合、言語サーバーはエディターに接続されます。
4. いずれの機能も機能していない場合は、上記の設定が **true** に設定されていることを確認してください (デフォルトでは **true** に設定されます)。この機能が機能していない場合は、言語サーバーに対する問題を報告します。

7.1.1.12.3. YAML Language Server Protocol (LSP) トレースの有効化

手順

LSP メッセージのログを VS Code Output ビューに記録するには、**yaml.trace.server** 属性を **verbose** に設定してトレースを有効にします。

7.1.1.13. Omnisharp-Theia プラグインを使用した Dotnet のログの表示

7.1.1.13.1. Omnisharp-Theia プラグイン

CodeReady Workspaces は Omnisharp-Theia プラグインをリモートプラグインとして使用します。これは github.com/redhat-developer/omnisharp-theia-plugin にあります。問題が生じた場合は、これを報告し、修正をリポジトリーにコントリビュートします。

このプラグインは omnisharp-roslyn を言語サーバーとして登録し、C# アプリケーションについてのプロジェクトの依存関係および言語構文を提供します。

言語サーバーは、.NET SDK 2.2.105 で実行されます。

7.1.1.13.2. Omnisharp-Theia プラグイン言語サーバーの状態の確認

手順

Omnisharp-Theia プラグインを実行するコンテナーが OmniSharp を実行しているかどうかを確認するには、**ps aux | grep OmniSharp.exe** コマンドを実行します。プロセスが実行されている場合、以下が出力例になります。

```
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/bin/mono
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/omnisharp/OmniSharp.exe
```

出力が異なる場合、言語サーバーはコンテナー内で起動していません。「重要なログの確認」で説明されているログを確認します。

7.1.1.13.3. Omnisharp Che-Theia プラグインの言語サーバー機能の確認

手順

- OmniSharp.exe プロセスが実行されている場合、**.cs** ファイルを開き、ホバーまたは補完機能を試行するか、または Problems または Outline ビューを開いて、言語サーバーの機能が機能しているかどうかを確認します。

7.1.1.13.4. Omnisharp-Theia プラグインログの Output パネルでの表示

手順

OmniSharp.exe が実行されている場合、Output パネルのすべての情報のログが記録されます。ログを表示するには、Output ビューを開き、ドロップダウンリストから C# を選択します。

7.1.1.14. NetcoredebugOutput プラグインを使用した Dotnet のログの表示

7.1.1.14.1. NetcoredebugOutput プラグイン

NetcoredebugOutput プラグインは、**netcoredbg** ツールを提供します。このツールは、VS Code Debug Adapter プロトコルを実装し、ユーザーが .NET Core ランタイムで .NET アプリケーションをデバッグできるようにします。

NetcoredebugOutput プラグインが実行されているコンテナーには Dotnet SDK v.2.2.105 が含まれます。

7.1.1.14.2. NetcoredebugOutput プラグインの状態の確認

手順

プラグインの初期化をテストするには、以下を実行します。

1. **launch.json** ファイルに netcoredbg デバッグ設定があるかどうかを確認します。デバッグ設定の例を以下に示します。

```
{
 "type": "netcoredbg",
 "request": "launch",
 "program": "$\{workspaceFolder}/bin/Debug/<target-framework>/<project-name.dll>",
 "args": [],
 "name": ".NET Core Launch (console)",
 "stopAtEntry": false,
 "console": "internalConsole"
}
```

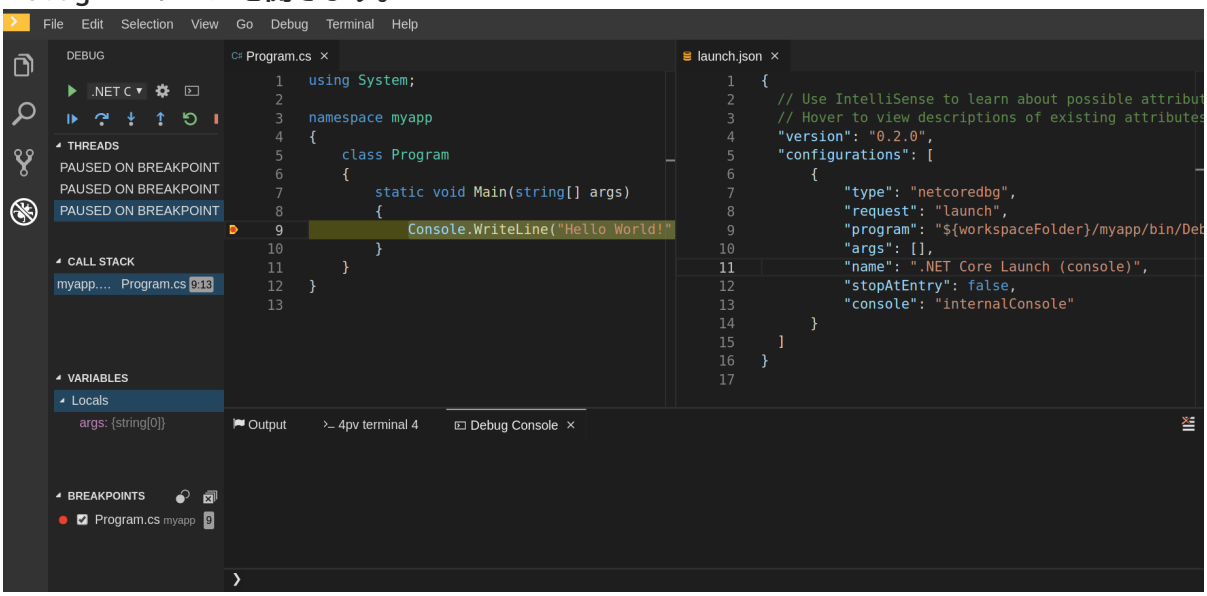
2. 存在するかどうかをテストするには、**launch.json** ファイルの **configuration** セクションの中括弧内で自動補完機能をテストします。**netcoredbg** が見つかる場合、Che-Theia プラグインは正しく初期化されています。そうでない場合は、重要なログの確認を参照してください。

7.1.1.14.3. NetcoredebugOutput プラグインログの Output パネルでの表示

本セクションでは、Output パネルで NetcoredebugOutput プラグインログを表示する方法を説明します。

手順

- Debug コンソールを開きます。



7.1.1.15. Camel のログの表示

7.1.1.15.1. Camel 言語サーバーの状態の確認

手順

ユーザーは、**vscode-apache-camel<xxx>** Camel コンテナーに保存される Camel 言語ツールを使用して、サイドカーコンテナーのログ出力を検査できます。

言語サーバーの状態を確認するには、以下のコマンドを実行します。

1. **vscode-apache-camel<xxx>** コンテナー内でターミナルを開きます。
2. **ps aux | grep java** コマンドを実行します。以下は、言語サーバープロセスの例です。

```
java -jar /tmp/vscode-unpacked/camel-tooling.vscode-apache-camel.latest.euqhbmepxd.camel-tooling.vscode-apache-camel-0.0.14.vsix/extension/jars/language-server.jar
```

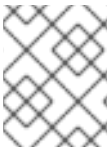
3. これが見つからない場合は、「重要なログの確認」を参照してください。

7.1.1.15.2. Camel ログの Output パネルでの表示

Camel 言語サーバーは、そのログを **$\{java.io.tmpdir}/log-camel-lsp.out** ファイルに書き込む SpringBoot アプリケーションです。**$\{java.io.tmpdir}** は **/tmp** ディレクトリーを参照するため、ファイル名は**/tmp/log-camel-lsp.out** になります。

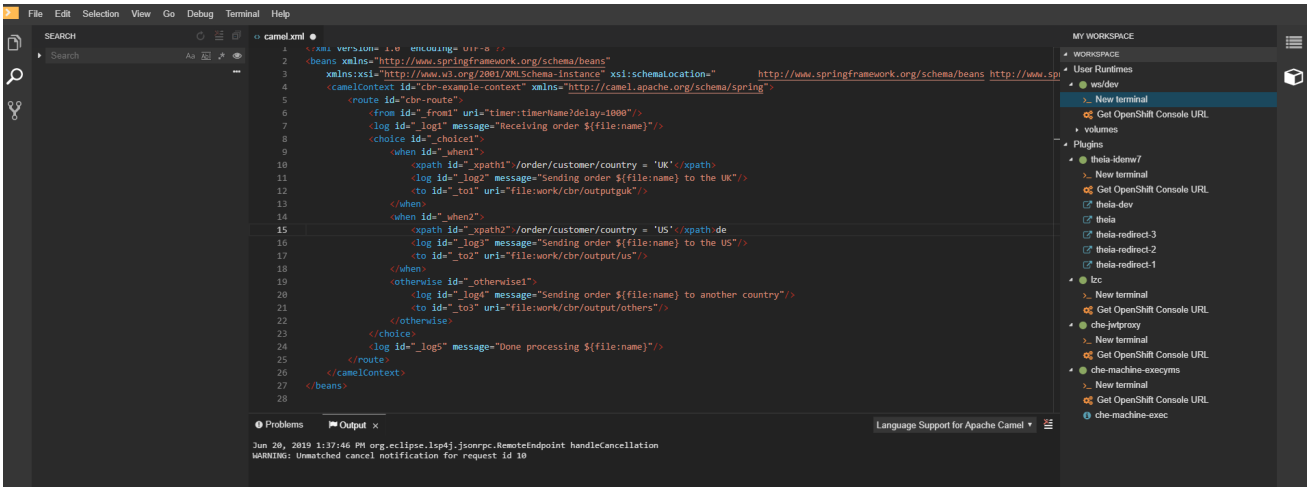
手順

Camel 言語サーバーのログは、Language Support for Apache Camelという名前の Output チャネルに出力されます。



注記

出力チャネルは、クライアントサイドで最初にログエントリーが作成される際にのみ作成されます。これは、問題がない場合には存在しない場合があります。



7.1.2. Che-Theia IDE ログの表示

本セクションでは、Che-Theia IDE ログを表示する方法を説明します。

7.1.2.1. OpenShift CLI を使用した Che-Theia エディターログの表示

Che-Theia エディターログの確認は、エディターによって読み込まれるプラグインについてよりよく理解し、知見を得るのに役立ちます。本セクションでは、OpenShift CLI (コマンドコマンドラインインターフェース) を使用して Che-Theia エディターログにアクセスする方法を説明します。

前提条件

- CodeReady Workspaces が OpenShift クラスターにデプロイされています。
- ワークスペースが作成されています。
- ユーザーの場所は、CodeReady Workspaces インストールプロジェクトになります。

手順

1. 利用可能な Pod の一覧を取得します。

```
$ oc get pods
```

例

```
$ oc get pods
NAME                                          READY  STATUS   RESTARTS  AGE
codeready-9-xz6g8                               1/1    Running  1         15h
workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2  4/4    Running  0         1h
```

2. 特定の Pod で利用可能なコンテナーの一覧を取得します。

```
$ oc get pods <name-of-pod> --output jsonpath='\{.spec.containers[*].name}'
```

例:

```
$ oc get pods workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -o
jsonpath='\{.spec.containers[*].name}'
> go-cli che-machine-exechr7 theia-idexzb vscode-gox3r
```

3. theia/ide コンテナーからログを取得します。

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

例:

```
$ oc logs --follow workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -container
theia-idexzb
>root INFO unzipping the plug-in 'task_plugin.theia' to directory: /tmp/theia-
unpacked/task_plugin.theia
root INFO unzipping the plug-in 'theia_yeoman_plugin.theia' to directory: /tmp/theia-
unpacked/theia_yeoman_plugin.theia
root WARN A handler with prefix term  is already registered.
root INFO [nsfw-watcher: 75] Started watching: /home/theia/.theia
root WARN e.onStart is slow, took: 367.4600000013015 ms
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [nsfw-watcher: 75] Started watching: /projects/.theia/tasks.json
```

```
root INFO [4f9590c5-e1c5-40d1-b9f8-ec31ec3bdac5] Sync of 9 plugins took:
62.26000000242493 ms
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [hosted-plugin: 88] PLUGIN_HOST(88) starting instance
```

7.2. 速度の遅いワークスペースのトラブルシューティング

ワークスペースの起動には時間がかかる場合があります。チューニングにより、この起動時間を短縮できる場合があります。オプションによっては、管理者またはユーザーはチューニングを行うことができます。

本セクションでは、ワークスペースをより迅速に起動したり、ワークスペースのランタイムパフォーマンスを改善したりするためのチューニングオプションが複数含まれています。

7.2.1. ワークスペースの起動時間の改善

Image Puller を使用したイメージのキャッシュ

ロール: 管理者
ワークスペースを起動すると、OpenShift はイメージをレジストリーからプルします。ワークスペースには、数多くのコンテナーを含めることができます。つまり OpenShift は、(コンテナーごとに 1 つの) Pod のイメージをプルするため、複数の Pod のイメージをプルすることを意味します。イメージのサイズと帯域幅によっては、これには時間がかかる場合があります。

Image Puller は、各 OpenShift ノードでイメージをキャッシュできるツールです。このため、プル前のイメージにより、起動時間が短縮されます。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/administration_guide/index#caching-images-for-faster-workspace-start_crw を参照してください。

より適切なストレージタイプの選択

ロール: 管理者およびユーザー
すべてのワークスペースには共有ボリュームが割り当てられています。このボリュームはプロジェクトファイルを保存するため、ワークスペースを再起動する際に変更が引き続き利用できるようになります。ストレージによっては、割り当てに数分かかる可能性があり、I/O が遅くなる可能性があります。

この問題を回避するには、一時ストレージまたは非同期ストレージを使用します。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#configuring-storage-types_crw を参照してください。

オフラインインストール

ロール: 管理者
CodeReady Workspaces のコンポーネントは OCI イメージです。オフラインモードで Red Hat CodeReady Workspaces を設定 (エアギャップシナリオ) することで、ランタイム時に追加のダウンロードを削減できます。この場合、開始時にすべてが準備できている必要があるためです。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces-in-a-restricted-environment_crw を参照してください。

ワークスペースプラグインの最適化

ロール: ユーザー
各種のプラグインを選択する場合、各プラグインでは OCI イメージである独自のサイドカーコンテナーを使用できます。OpenShift はこれらのサイドカーコンテナーのイメージをプルします。

プラグインの数を減らすか、またはそれらを無効にして起動時間が短縮されるかどうかを確認します。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/administration_guide/index#caching-images-for-faster-workspace-start_crw も参照してください。

パブリックエンドポイントの数の縮小

ロール: 管理者
それぞれのエンドポイントについて、OpenShift は OpenShift Route オブジェクトを作成します。基礎となる設定によっては、作成に時間がかかる場合があります。

この問題を回避するには、公開される部分を縮小します。たとえば、コンテナー内でリッスンする新規ポートを自動的に検出し、ローカル IP アドレス(127.0.0.1) を使用してプロセスのトラフィックをリダイレクトする場合、Che-Theia IDE プラグインには 3 つのオプションのルートがあります。

エンドポイントの数を減らし、すべてのプラグインのエンドポイントをチェックすることで、ワークスペースの起動が速くなります。

CDN 設定

IDE エディターは CDN (コンテンツ配信ネットワーク) を使用してコンテンツを提供します。コンテンツがクライアント (またはオフライン設定のローカルルート) に対して CDN を使用することを確認します。
これを確認するには、ブラウザーで Developer Tools を開き、Network タブに **vendors** があることを確認します。**vendor.<random-id>.js** または **editor.main.*** は CDN URL から取得する必要があります。

7.2.2. ワークスペースのランタイムパフォーマンスの改善

十分な CPU リソースを提供する

プラグインは CPU リソースを消費します。たとえば、プラグインが IntelliSense 機能を提供する場合、CPU リソースを増やすと、パフォーマンスが向上する可能性があります。
devfile 定義 **devfile.yaml** の CPU 設定が正しいことを確認します。

```
apiVersion: 1.0.0

components:
 -
   type: chePlugin
   id: id/of/plug-in
   cpuLimit: 1360Mi (1)
   cpuRequest: 100m (2)
```

(1) プラグインの CPU 制限を指定します。

(2) プラグインの CPU 要求を指定します。

十分なメモリーを提供する

プラグインは CPU およびメモリーリソースを消費します。たとえば、プラグインが IntelliSense 機能を提供する場合、データを収集すると、コンテナーに割り当てられるすべてのメモリーを消費する可能性があります。
プラグインにより多くのメモリーを提供することで、パフォーマンスを改善できます。以下のメモリー設定が正しいことを確認します。

- プラグイン定義: meta.yaml ファイル
- devfile 定義: devfile.yaml ファイル

```
apiVersion: v2

spec:
  containers:
    - image: "quay.io/my-image"
      name: "vscode-plugin"
      memoryLimit: "512Mi" (1)
  extensions:
    - https://link.to/vsix
```

(1) プラグインのメモリー制限を指定します。

devfile 定義 (devfile.yaml)

```
apiVersion: 1.0.0

components:
  -
    type: chePlugin
    id: id/of/plug-in
    memoryLimit: 1048M (1)
    memoryRequest: 256M
```

(1) このプラグインのメモリー制限を指定します。

より適切なストレージタイプの選択

I/O の速度を上げるために、一時ストレージまたは非同期ストレージを使用します。https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#configuring-storage-types_crw を参照してください。

7.3. ネットワーク問題のトラブルシューティング

ほとんどの場合、接続の問題は、ファイアウォール、プロキシサーバー、企業ネットワーク、またはその他のネットワークがCodeReady Workspacesをブロックするように構成されているために発生します。

本セクションでは、企業のネットワークポリシーに関連する問題を回避したり、解決したりする方法を説明します。ネットワーク管理者は、CodeReady Workspaces が適切に機能するために必要なポートまたは WebSocket プロトコルを有効にするのに必要になる場合があります。

一般的なシナリオ

- 特定の Web サイトの追加ポートを開きます。
- プロキシーサーバーで WebSocket を有効にします。

7.3.1. WebSocket プロトコルの有効化

CodeReady Workspaces IDE の適切な機能のために WebSocket プロトコルの有効化が重要です。

WebSocket プロトコル自体はプロキシーサーバーとファイアウォールを認識しませんが、HTTP サーバーはデフォルトの HTTP および HTTPS ポートを WebSocket サーバーと共有できます。

- HTTP: port 80
- HTTPS: port 433

一部のプロキシーサーバーはデフォルトで WebSocket と連携しますが、WebSocket が正しく機能しないようにするため、接続が失敗します。

プロキシーサーバーには追加の設定が必要で、特定のプロキシーサーバーにはアップグレードが必要になる場合があります。これにより、WebSocket のサポートが許可されるアップグレードが必要になります。

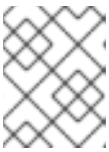
7.3.2. WebSocket セキュアな接続のトラブルシューティング

セキュアな WebSocket 接続では、不正なプロキシーによる干渉リスクが軽減されるため、機密性および信頼性が強化されます。CodeReady Workspaces はデフォルトで WebSocket セキュアな接続で操作するため、通常はアクションは必要ありません。顧客のセキュリティーポリシーにより、適切な CodeReady Workspaces 機能に関する問題を生じさせる WebSocket プロトコルの一部がブロックされます。ただし、これらの問題は CodeReady Workspaces サポートの対象外であり、ネットワーク管理者が解決する必要があります。

失敗した WebSocket Secure (WSS)接続のトラブルシューティングを行うには、本セクションの手順を使用します。

前提条件

- サポートされる Web ブラウザーの使用
  - Chrome
  - Firefox



注記

サポートされていない Web ブラウザーを使用すると接続が中断され、その後に警告メッセージが表示されます。

手順

1. ブラウザーのサポートを確認します。
   a. WebSocket プロトコルが、サポートされているブラウザーのいずれかで リアルタイム Web テスト を使用して有効になっていることを確認します。
   問題が解決されていない場合は、次の手順に従います。
2. プロキシーサーバーおよびファイアウォールの設定を確認します。
   a. ポート 443 で WebSocket Secure (WSS)接続をブロックするプロキシーサーバーまたはファイアウォールがあることを確認するようシステム管理者に依頼します。
   考えられる必要なアクションは以下のとおりです。
      - ファイアウォールに例外を追加します。

- プロキシにWebSocket接続をインターセプトさせます。

検証

WebSocket プロトコルが、サポートされているブラウザーのいずれかで リアルタイム Web テスト を使用して有効になっていることを確認します。

7.4. デバッグモードでの CODEREADY WORKSPACES ワークスペースの起動

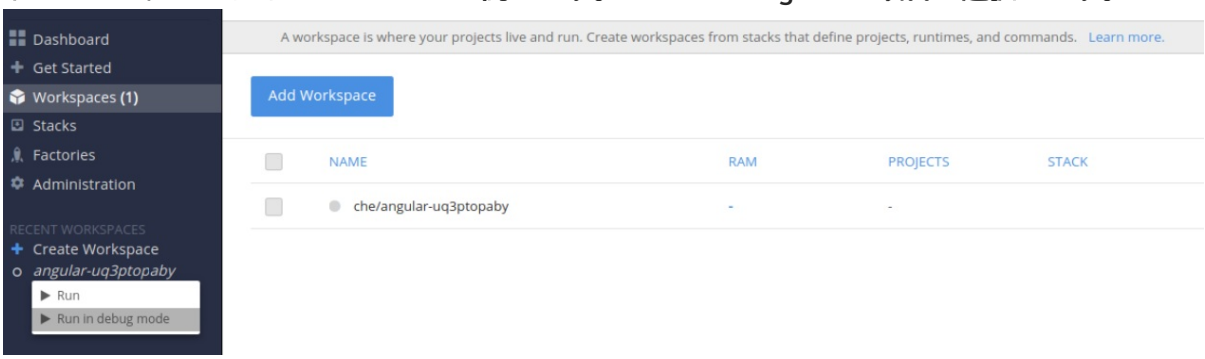
本セクションでは、デバッグモードで Red Hat CodeReady Workspaces ワークスペースを起動する方法を説明します。

前提条件

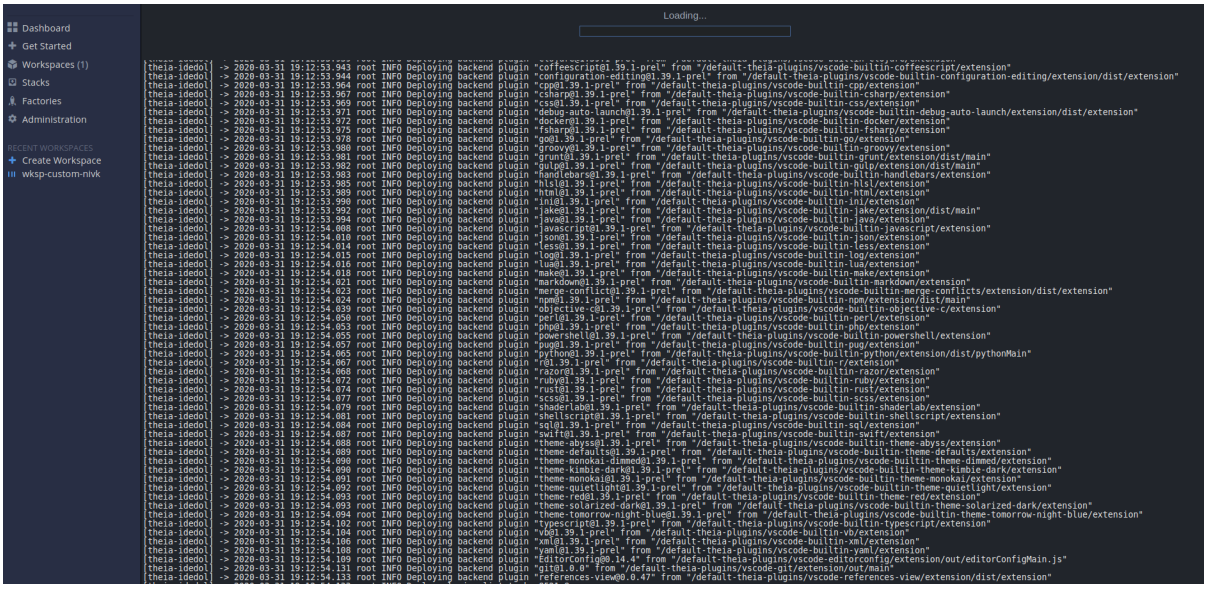
- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペース。「新規 CodeReady Workspaces 2.6 ワークスペースの作成および設定」を参照してください。

手順

1. 直近のワークスペースからターゲットワークスペースを見つけます。ワークスペース名を右クリックして、コンテキストメニューを開きます。Run in debug mode項目を選択します。



2. ターゲットワークスペースをクリックしてログを表示します。
3. ワークスペースログが表示されます。



7.5. 起動の失敗後のデバッグモードでの CODEREADY WORKSPACES ワークスペースの再起動

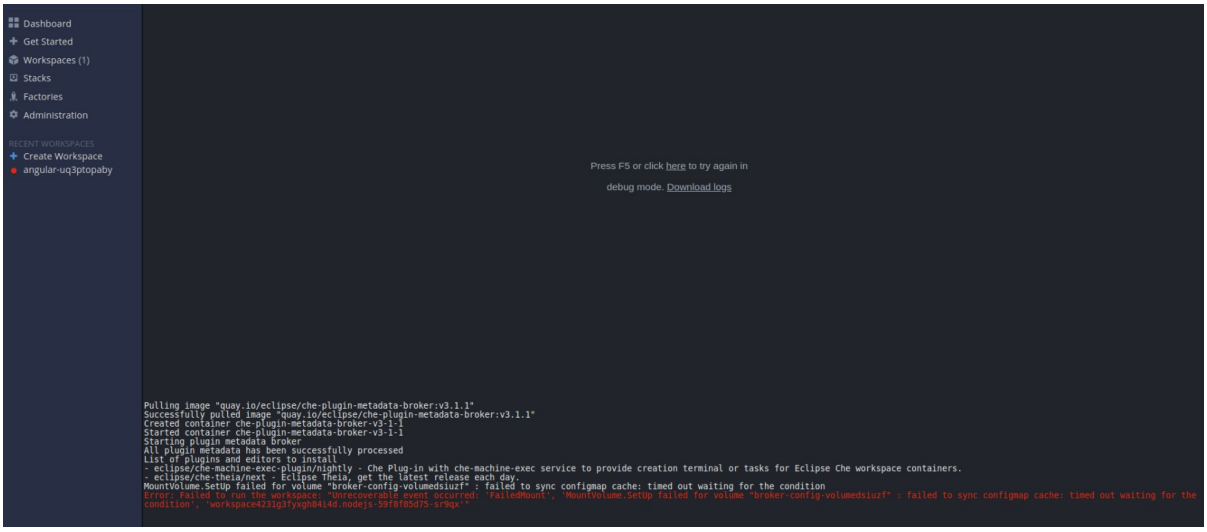
本セクションでは、ワークスペースの起動時に失敗した後にデバッグモードで Red Hat CodeReady Workspaces ワークスペースを再起動する方法を説明します。

前提条件

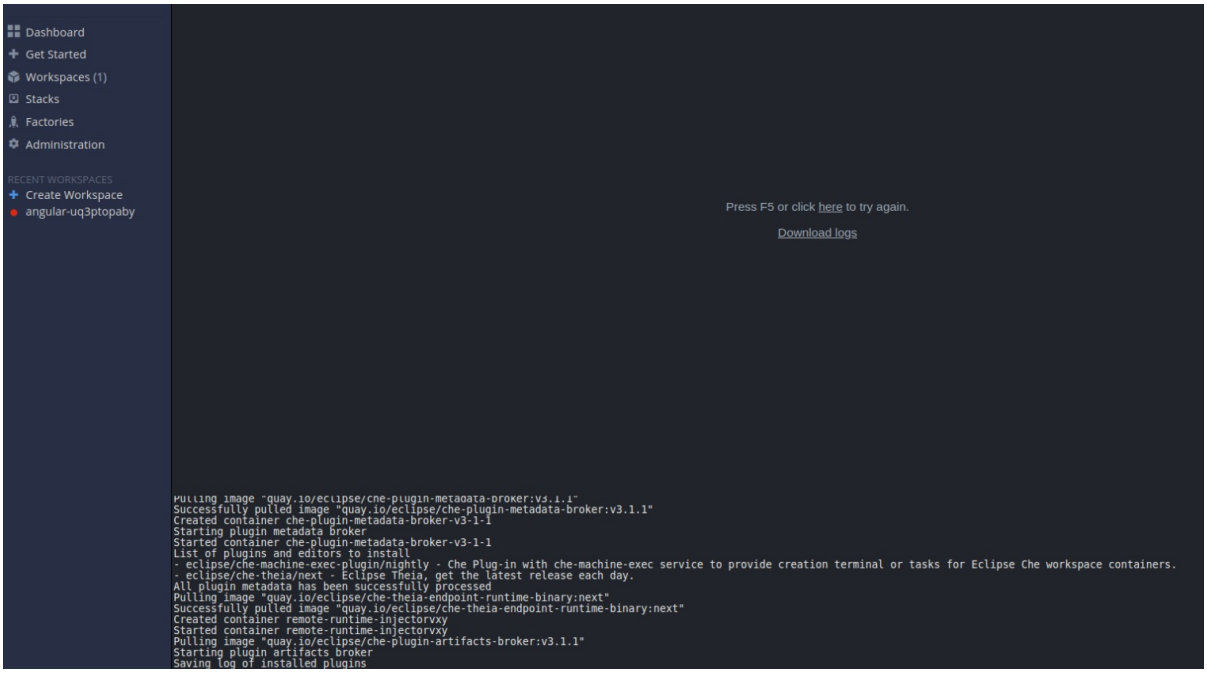
- **Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces のインストールについて参照してください。**
- **起動に失敗した既存のワークスペース。**

手順

1. **直近のワークスペースからターゲットワークスペースを見つけます。ターゲットワークスペースをクリックし、ログを表示します。**



2. **デバッグモードで再起動するリンクをクリックします。**
3. **起動後に Download logs リンクを使用して詳細なログをダウンロードします。**


Dashboard
Get Started
Workspaces (1)
Stacks
Factories
Administration
RECENT WORKSPACES
Create Workspace
angular-uq3ptopaby
Press F5 or click here to try again.
Download logs
Pulling image "quay.io/eclipse/che-plugin-metadata-broker:v3.1.1"
Successfully pulled image "quay.io/eclipse/che-plugin-metadata-broker:v3.1.1"
Created container che-plugin-metadata-broker-v3-1-1
Starting plugin metadata broker
Started container che-plugin-metadata-broker-v3-1-1
List of plugins and editors to install
- eclipse/che-machine-exec-plugin/nightly - Che Plug-in with che-machine-exec service to provide creation terminal or tasks for Eclipse Che workspace containers.
- eclipse/che-theia/next - Eclipse Theia, get the latest release each day.
All plugin metadata has been successfully processed
Pulling image "quay.io/eclipse/che-theia-endpoint-runtime-binary:next"
Successfully pulled image "quay.io/eclipse/che-theia-endpoint-runtime-binary:next"
Created container remote-runtime-injectorvxy
Started container remote-runtime-injectorvxy
Pulling image "quay.io/eclipse/che-plugin-artifacts-broker:v3.1.1"
Starting plugin artifacts broker
Saving log of installed plugins

第8章 OPENSHIFT CONNECTOR の概要

OpenShift Connector (Visual Studio Code OpenShift Connector for Red Hat OpenShift とも呼ばれる) は、Red Hat OpenShift 3 または 4 クラスターと対話する方法を提供する CodeReady Workspaces のプラグインです。

OpenShift Connector を使用すると、CodeReady Workspaces IDE でアプリケーションを作成し、ビルドし、デバッグし、アプリケーションを実行中の OpenShift クラスターに直接デプロイできます。

OpenShift Connector は、OpenShift Do (**odo**) ユーティリティーの GUI であり、これは OpenShift CLI (**oc**) コマンドを小型の単位に集約します。そのため、OpenShift Connector は OpenShift についての背景知識を持たない新規開発者がアプリケーションを作成し、それらをクラウドで実行するのに役立ちます。ユーザーは、複数の **oc** コマンドを使用する代わりに、プロジェクト、アプリケーション、またはサービスなどの事前に設定されたテンプレートを選択して新規コンポーネントまたはサービスを作成し、これを OpenShift コンポーネントとしてクラスターにデプロイします。

本セクションでは、OpenShift Connector プラグインのインストール、有効化、および基本的な使用について説明します。

- 「OpenShift Connector の機能」
- 「OpenShift Connector の CodeReady Workspaces へのインストール」
- 「CodeReady Workspaces からの OpenShift Connector を使用した認証」
- 「CodeReady Workspaces での OpenShift Connector を使用したコンポーネントの作成」
- 「OpenShift Connector を使用したソースコードの GitHub から OpenShift コンポーネントへの接続」

8.1. OPENSHIFT CONNECTOR の機能

OpenShift Connector プラグインを使用するユーザーは、GUI で OpenShift コンポーネントを作成し、これを OpenShift クラスターにデプロイし、プッシュできます。

CodeReady Workspaces で使用する場合、OpenShift Connector GUI はユーザーに以下の利点を提供します。

クラスター管理

- 以下を使用したクラスターへのログイン
  - 認証トークン
  - ユーザー名およびパスワード
  - CodeReady Workspaces が OpenShift OAuth サービスで認証される場合の自動ログイン機能
- 拡張ビューから直接、複数の異なる **.kube/config** エントリー間でコンテキストを切り替える。
- Explorer ビューから、OpenShift リソースをビルドおよびデプロイメント設定として表示し、管理する。

開発

- CodeReady Workspaces から直接、ローカルまたはホストされる OpenShift クラスターに接続する。
- 変更内容によるクラスターの迅速な更新。
- 接続されたクラスターを使用したコンポーネント、サービス、およびルートの作成。
- 拡張機能から直接ストレージをコンポーネントに追加する。

デプロイメント

- CodeReady Workspaces からの直接シングルクリックによる OpenShift クラスターへデプロイ
- デプロイされたアプリケーションにアクセスするために作成された複数のルートに移動する。
- 複数の相互にリンクされたコンポーネントおよびサービスをクラスターに直接デプロイする。
- CodeReady Workspaces IDE からコンポーネントの変更をプッシュし、監視する。
- CodeReady Workspaces の統合ターミナルビューでログを直接ストリーミングする。

モニタリング

- CodeReady Workspaces IDE から直接 OpenShift リソースを操作する。
- ビルドおよびデプロイメント設定の開始および再開
- デプロイメント、Pod、およびコンテナーのログの表示およびフォロー。

8.2. OPENSHIFT CONNECTOR の CODEREADY WORKSPACES へのインストール

OpenShift Connector は、CodeReady Workspaces をエディターとして使用して基本的な OpenShift コンポーネントを作成し、コンポーネントを OpenShift クラスターにデプロイするために設計されたプラグインです。インスタンスがプラグインが利用可能であることを視覚的に確認するには、OpenShift アイコンが CodeReady Workspaces の左側のメニューに表示されるかどうかを確認します。

CodeReady Workspaces インスタンスで OpenShift Connector をインストールし、有効にするには、本セクションの手順を使用します。

前提条件

- Red Hat CodeReady Workspaces の実行中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。

手順

OpenShift Connector を CodeReady Workspaces Plugins パネルの拡張機能として追加し、OpenShift Connector を CodeReady Workspaces にインストールします。

1. **Ctrl+Shift+J** を押すか、または View → Plugins に移動して、 CodeReady Workspaces Plugins パネルを開きます。

2. vscode-openshift-connector を検索し、**Install** ボタンをクリックします。
3. 変更を有効にするには、ワークスペースを再起動します。
4. 専用の OpenShift Application Explorer アイコンが左側のパネルに追加されます。

8.3. CODEREADY WORKSPACES からの OPENSHIFT CONNECTOR を使用した認証

以下のセクションは、OpenShift OAuth サービスが CodeReady Workspaces インスタンスを認証していない場合のみ関連します。それ以外の場合は、OpenShift Connector プラグインは CodeReady Workspaces が実行する Openshift インスタンスで認証を自動的に確立します。

ユーザーが CodeReady Workspaces からコンポーネントを開発およびプッシュする前に、OpenShift クラスターで認証する必要があります。

OpenShift Connector は、CodeReady Workspaces インスタンスから OpenShift クラスターにログインするための以下の方法を提供します。

- CodeReady Workspaces がデプロイされる OpenShift クラスターにログインするよう要求する通知を使用します。
- **Log in to the cluster** ボタンの使用。
- コマンドパレットの使用

注記

CodeReady Workspaces 2.6 では、OpenShift Connector プラグインではターゲットクラスターへの手動接続が必要です。

デフォルトで、Openshift Connector プラグインは、プロジェクト管理パーミッションがない可能性がある **inClusterUser** としてクラスターにログインします。これにより、Openshift Application Explorer を使用して新規プロジェクトが作成されるとエラーメッセージが表示されます。

> Failed to create Project with error 'Error: Command failed: "/tmp/vscode-unpacked/redhat.vscode-openshift -connector.latest.qvkozqtkba.openshift-connector-0.1.4-523.vsix/extension/out/tools/linux/odo" project create test-project ✗ projectrequests.project.openshift.io is forbidden

この一時的な問題を回避するには、ローカルクラスターからログアウトし、OpenShift ユーザーの認証情報を使用して OpenShift クラスターに再度ログインします。

OpenShift のローカルインスタンス（CodeReady Containers、Minishift など）を使用する場合、ユーザーの認証情報はワークスペース **~/.kube/config** ファイルに保存され、後続のログインで自動認証に使用できます。CodeReady Workspaces のコンテキストでは、**~/.kube/config** はプラグインサイドカーコンテナーの一部として保存されます。

前提条件

- CodeReady Workspaces の実行中のインスタンスがある。CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-

codeready-workspaces_crw を参照してください。

- CodeReady Workspaces ワークスペースが作成されている。
- OpenShift Connector プラグインが利用可能である。
- OpenShift OAuth プロバイダーが設定されている（CodeReady Workspaces がデプロイされている OpenShift クラスターへの自動ログインのみ）。「OpenShift OAuth の設定」 を参照してください。

手順

1. 左側のパネルで OpenShift Application Explorer アイコンを選択します。
   OpenShift Connector パネルが表示されます。
2. OpenShift Application Explorer を使用してログインします。以下の方法のいずれかを使用します。
   - ペインの左上にある **Log in to cluster** ボタンをクリックします。
   - **F1** 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。
     OpenShift: Log in to clusterを検索し、**Enter** を押します。
3. You are already logged in a cluster.メッセージが表示されたら、Yes をクリックします。
   Credentials または Token を使用してログインするかどうかの選択が画面上部に表示されます。
4. クラスターにログインする方法を選択し、ログイン手順に従います。

注記

トークンで認証するには、メインの OpenShift Container Platform 画面の右上の <User name> → Copy Login Commandにある必要なトークン情報を使用します。

8.4. CODEREADY WORKSPACES での OPENSHIFT CONNECTOR を使用したコンポーネントの作成

OpenShift のコンテキストでは、Component (コンポーネント) および Service (サービス) は、Application (アプリケーション) に保存する必要のある基本的な構造です。これは、読みやすさの向上のために、デプロイ可能な内容を仮想フォルダーに編成する OpenShift プロジェクトの一部です。

本章では、OpenShift Connector プラグインを使用して OpenShift コンポーネントを CodeReady Workspaces で作成し、それらを OpenShift クラスターにプッシュする方法を説明します。

前提条件

- CodeReady Workspaces の実行中のインスタンスがある。CodeReady Workspaces のインスタンスをインストールするには、https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.6/html-single/installation_guide/index#installing-codeready-workspaces_crw を参照してください。
- ユーザーは、OpenShift Connector プラグインを使用して OpenShift クラスターにログインしている。

手順

1. OpenShift Connector パネルで、赤い OpenShift アイコンがある行を右クリックし、New Project を選択します。
2. プロジェクトの名前を入力します。
3. 作成したプロジェクトを右クリックし、New Component を選択します。
4. プロンプトが表示されたら、コンポーネントを保存できる新規 OpenShift アプリケーションの名前を入力します。
   コンポーネントのソースの以下のオプションが表示されます。
   a. Git リポジトリー
      これにより、Git リポジトリー URL を指定し、ランタイムの意図されるリビジョンを選択することを求めるプロンプトが出されます。
   b. バイナリーファイル
      これにより、ファイルエクスプローラーからファイルを選択することを求めるプロンプトが出されます。
   c. ワークスペースディレクトリー
      これにより、ファイルエクスプローラーからフォルダーを選択することを求めるプロンプトが出されます。
5. コンポーネントの名前を入力します。
6. コンポーネントタイプを選択します。
7. コンポーネントタイプのバージョンを選択します。
8. コンポーネントが作成されます。コンポーネントを右クリックし、New URL を選択して、選択したコンポーネントの名前を入力します。
9. コンポーネントは OpenShift クラスターにプッシュできる状態になります。これを実行するには、コンポーネントを右クリックして Push を選択します。
   これで、コンポーネントはクラスターにデプロイされます。右クリックして、デバッグやブラウザーで開くなどの追加のアクションを選択します (これにはポート 8080 が公開されている必要があります)。

8.5. OPENSHIFT CONNECTOR を使用したソースコードの GITHUB から OPENSHIFT コンポーネントへの接続

ユーザーが追加の開発で必要となる Git に保存されたソースコードを持つ場合、Git リポジトリーから OpenShift Connector コンポーネントに直接デプロイするのがより効率的な方法になります。

本章では、Git リポジトリーからコンテンツを取得し、これを CodeReady Workspaces で開発された OpenShift コンポーネントに接続する方法を説明します。

前提条件

- CodeReady Workspaces ワークスペースが実行中である。
- OpenShift Connector を使用して OpenShift クラスターにログインしている。

手順

GitHub コンポーネントを変更するには、リポジトリーのクローンを CodeReady Workspaces に作成し、このソースコードを取得します。

1. CodeReady Workspaces のメイン画面で、 **F1** を押して Command Palette を開きます。
2. **Git Clone** コマンドを Command Palette に入力し、**Enter** を押します。
3. GitHub URL を指定し、デプロイメントの宛先を選択します。
4. **Add to workspace** ボタンをクリックしてソースコードファイルをプロジェクトに追加します。

Git リポジトリーのクローン作成についての詳細は、「HTTPS を使用した Git リポジトリーへのアクセス」を参照してください。

第9章 TELEMETRY の概要

Telemetry は、使用データのコレクションであり、これは透過的かつ倫理的です。デフォルトで、Telemetry は Red Hat CodeReady Workspaces では利用できませんが、プラグインのメカニズムを使用して Telemetry を有効にする抽象 API を使用できます。この方法は、Telemetry がすべてのワークスペースで有効にされている ホストされている Che サービスで使用されます。

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

9.1. ユースケース

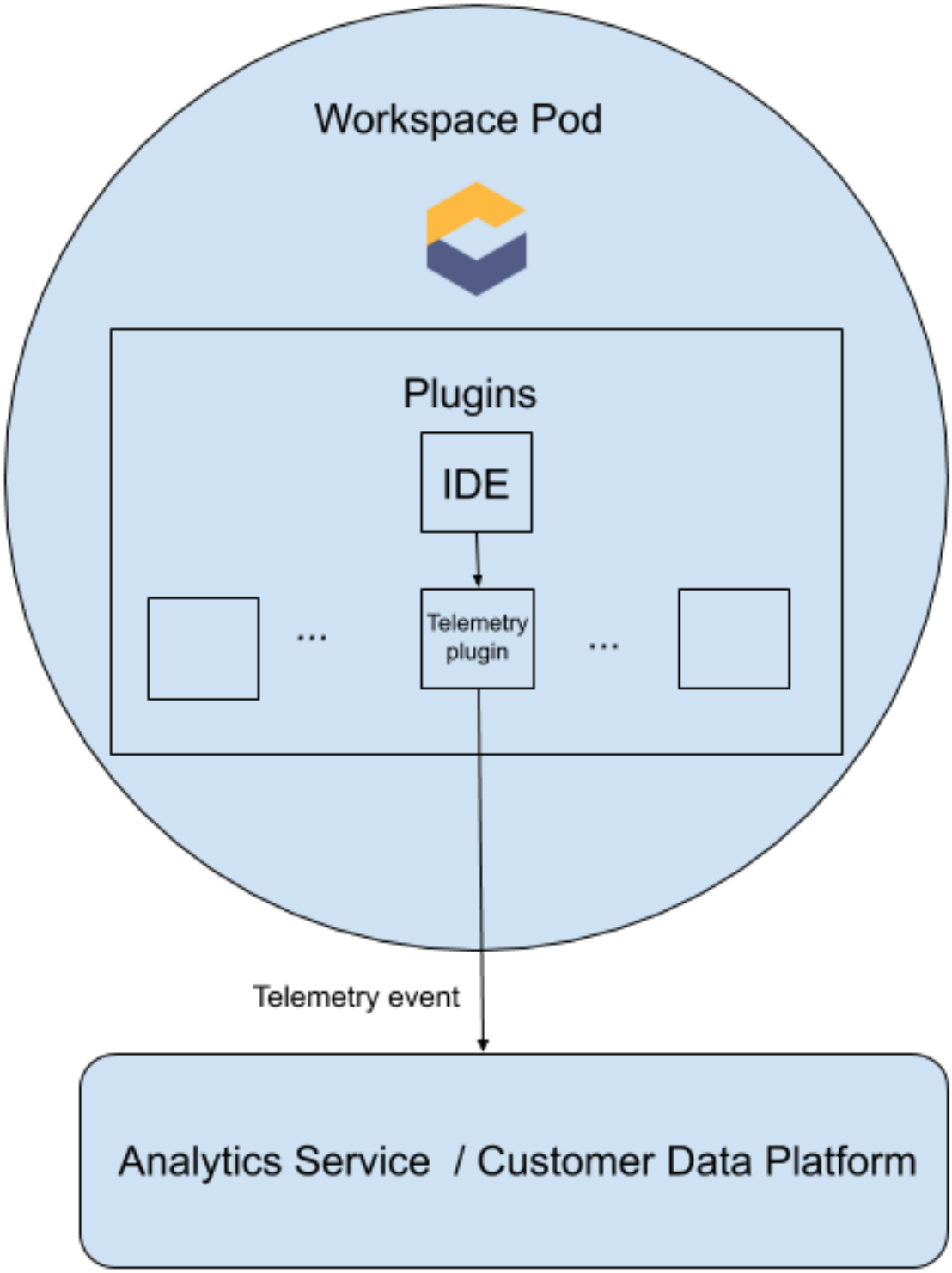
Red Hat CodeReady Workspaces Telemetry API では、以下の追跡が可能です。

- ワークスペースの使用時間
- ファイルの編集、コミット、およびリモートリポジトリーへのプッシュなどのユーザー駆動型アクション
- ワークスペースで有効なプラグインの一覧
- ワークスペースで使用されるプログラミング言語および devfile

9.2. 仕組み

CodeReady Workspaces ワークスペースが起動すると、**che-theia** コンテナーは Telemetry プラグインを起動します。これは Telemetry イベントをバックエンドに送信します。**$CHE_WORKSPACE_TELEMETRY_BACKEND_PORT** 環境変数がワークスペース Pod に設定されている場合、Telemetry プラグインはそのポートでリッスンしているバックエンドにイベントを送信します。

CodeReady Workspaces ワークスペースで Telemetry バックエンドコンテナーが実行されており、**$CHE_WORKSPACE_TELEMETRY_BACKEND_PORT** でリッスンしている場合は、Telemetry プラグインから送信されるイベントを取り、それらをバックエンド固有の表現に変換して、設定された解析バックエンド (Segment や Woopra など) に送信します。



9.3. TELEMETRY プラグインの作成

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

- **isEnabled()** - Telemetry バックエンドが正常に機能しているかどうかを判別します。これは、常に true を返すか、または接続プロパティーがない場合に false を返すなどしてさらに複雑なチェックがあることを意味します。
- **destroy()** - Telemetry バックエンドをシャットダウンする前に実行されるクリーンアップ方法。このメソッドは、WORKSPACE_STOPPED イベントを送信します。

- **onActivity() - 特定のユーザーについて一部のアクティビティーが依然として実行されていることを通知します。これは主にWORKSPACE_INACTIVEイベントを送信するために使用されます。**
- **onEvent() - Telemetry イベントをWORKSPACE_USED または WORKSPACE_STARTED などの Telemetry サーバーに送信します。**
- **increaseDuration() - 短時間に複数のイベントを送信するのではなく、現在のイベントの期間を長くします。**

次のセクションでは、以下について説明します。

- echo でイベントを標準出力に送信するための Telemetry サーバーの作成。
- CodeReady Workspaces Telemetry クライアントの拡張、およびユーザーのカスタムバックエンドの実装。
- ユーザーのカスタムバックエンドの CodeReady Workspaces ワークスペースプラグインを表す **meta.yaml** ファイルの作成。
- **CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS** 環境変数を設定して、カスタムプラグインの場所を CodeReady Workspaces に指定。

9.3.1. はじめに

以下では、CodeReady Workspaces Telemetry システムを拡張してカスタムのバックエンドに接続するのに必要な手順を説明します。

1. イベントを受信するサーバープロセスの作成
2. CodeReady Workspaces ライブラリーを拡張して、イベントをサーバーに送信するバックエンドを作成する
3. コンテナーでの Telemetry バックエンドのパッケージ化およびイメージレジストリーへのデプロイ
4. バックエンドのプラグインを追加し、CodeReady Workspaces に対してワークスペースにプラグインを読み込むように指示する

オプション: イベントを受信するサーバーの作成

この例は、CodeReady Workspaces からイベントを受信し、それらを標準出力に書き込むサーバーを作成する方法を示しています。

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

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

main.go

```
package main

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

```
 "go.uber.org/zap"
)

var logger *zap.SugaredLogger

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

func activity(w http.ResponseWriter, req *http.Request) {
 switch req.Method {
 case "GET":
  logger.Info("GET /activity, doing nothing")
 case "POST":
  logger.Info("POST /activity")
  body, err := req.GetBody()
  if err != nil {
   logger.With("error", err).Info("error getting body")
   return
  }
  responseBody, err := ioutil.ReadAll(body)
  if err != nil {
   logger.With("error", err).Info("error reading response body")
   return
  }
  logger.With("body", string(responseBody)).Info("got activity")
 }
}

func main() {

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

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

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

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

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

manifest.yaml で、プッシュしたイメージと OpenShift クラスターのパブリックホスト名に一致するように **image** および **host** フィールドを置き換えます。次に、以下を実行します。

```
$ oc apply -f manifest.yaml -n {prod-namespace}
```

9.3.2. 新しい Maven プロジェクトの作成

注記

開発時に迅速なフィードバックを可能にするためには、CodeReady Workspaces ワークスペース内で開発を行うことが推奨されます。これにより、クラスターでアプリケーションを実行し、ワークスペースのフロントエンド Telemetry プラグインに接続し、イベントをカスタムバックエンドに送信できます。

1. 新規 Maven Quarkus プロジェクトのスキャフォールディングを作成します。

```
$ mvn io.quarkus:quarkus-maven-plugin:1.2.1.Final:create \
  -DprojectGroupId=mygroup -DprojectArtifactId=telemetryback-end \
  -DprojectVersion=my-version -DclassName="org.my.group.MyResource"
```

2. **pom.xml**の**org.eclipse.che.incubator.workspace-telemetry.back-end-base**に依存関係を追加します。

   pom.xml

```
<dependency>
    <groupId>org.eclipse.che.incubator.workspace-telemetry</groupId>
    <artifactId>backend-base</artifactId>
    <version>0.0.11</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.12</version>
</dependency>
```

3. Apache HTTP コンポーネントライブラリーを追加して、HTTP リクエストを送信します。

4. **back-end-base の最新バージョンおよび Maven コーディネートについては、GitHub パッケージを参照してください。GitHub パッケージには、CodeReady Workspaces Telemetry ライブラリーをダウンロードするために read:packages パーミッションを持つ個人アクセストークンが必要です。個人アクセストークンを作成し、トークンの値をコピーします。**

5. **リポジトリールートに settings.xml ファイルを作成し、コーディネートとトークンを che-incubator パッケージに追加します。**

   settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>che-incubator</id>
      <username>${env.GITHUB_USERNAME}</username>
      <password>${env.GITHUB_TOKEN}</password>
    </server>
  </servers>

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

   このファイルは、コンテナーでアプリケーションをパッケージ化する際に使用されます。ローカルで実行している場合は、情報を個人の settings.xml ファイルに追加します。

9.3.3. アプリケーションの実行

アプリケーションを実行し、アプリケーションが CodeReady Workspaces ワークスペースにあるかどうかをテストします。

```
$ mvn quarkus:dev -Dquarkus.http.port=${CHE_WORKSPACE_TELEMETRY_BACKEND_PORT}
```

自己署名証明書を使用して CodeReady Workspaces のセキュリティーを保護する場合は、証明書をトラストストアに追加し、ワークスペースにマウントします。また、Java システムプロパティー **-Djavax.net.ssl.trustStore=/path/to/trustStore** を **mvn** コマンドに追加します。たとえば、トラストストアが **$JAVA_HOME/jre/lib/security/cacerts** にあることを前提とします。

```
$ keytool -import -alias self-signed-certificate \
  -file <path/to/self-signed-certificate> -keystore $JAVA_HOME/jre/lib/security/cacerts
```

次は以下のようになります。

```
$ mvn quarkus:dev -Dquarkus.http.port=${CHE_WORKSPACE_TELEMETRY_BACKEND_PORT} \
  -Djavax.net.ssl.trustStore=$JAVA_HOME/jre/lib/security/cacerts
```

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

プロジェクトに新しいファイルを 2 つ作成します。

- **AnalyticsManager.java** - Telemetry システムに固有のロジックが含まれます。
- **MainConfiguration.java** - AnalyticsManager のインスタンスを作成し、イベントのリッスンを開始する主なエントリーポイントです。

AnalyticsManager.java

```
package org.my.group;

import java.util.Map;

import org.eclipse.che.api.core.rest.HttpJsonRequestFactory;
import org.eclipse.che.incubator.workspace.telemetry.base.AbstractAnalyticsManager;
import org.eclipse.che.incubator.workspace.telemetry.base.AnalyticsEvent;

public class AnalyticsManager extends AbstractAnalyticsManager {

    public AnalyticsManager(String apiEndpoint, String workspaceId, String machineToken,
            HttpJsonRequestFactory requestFactory) {
        super(apiEndpoint, workspaceId, machineToken, requestFactory);
    }

    @Override
    public boolean isEnabled() {
        // TODO Auto-generated method stub
        return true;
    }

    @Override
    public void destroy() {
        // TODO Auto-generated method stub
    }

    @Override
    public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent,
String resolution,
            Map<String, Object> properties) {
        // TODO Auto-generated method stub
```

```
    }

    @Override
    public void increaseDuration(AnalyticsEvent event, Map<String, Object> properties) {
        // TODO Auto-generated method stub
    }

    @Override
    public void onActivity() {
        // TODO Auto-generated method stub
    }
}
```

MainConfiguration.java

```
package org.my.group;

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

import org.eclipse.che.incubator.workspace.telemetry.base.AbstractAnalyticsManager;
import org.eclipse.che.incubator.workspace.telemetry.base.BaseConfiguration;

@Dependent
public class MainConfiguration extends BaseConfiguration {
    @Produces
    public AbstractAnalyticsManager analyticsManager() {
      return new AnalyticsManager(apiEndpoint, workspaceId, machineToken,
requestFactory());

    }
}
```

9.3.5. isEnabled() の実装

この例では、このメソッドは呼び出される際に true を返します。サーバーが実行されている場合は常に、これは有効にされ、機能します。

AnalyticsManager.java

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

より複雑なログインを isEnabled() に設定することができます。たとえば、サービスは特定のケースでは機能すると見なすことはできません。ホストされる CodeReady Workspaces woopra バックエンドは、バックエンドが有効にされているかどうかを判定する前に設定プロパティーが存在することを確認します。

9.3.6. Implementing onEvent()

onEvent() は、バックエンドに渡されたイベントを Telemetry システムに送信します。サンプルアプリケーションでは、HTTP POST ペイロードをサーバーに送信します。サンプル Telemetry サーバーアプ

リケーションは、http://little-telemetry-back-end-che.apps-crc.testing の URL で OpenShift にデプロイされます。

AnalyticsManager.java

```
@Override
public void onEvent(AnalyticsEvent event, String ownerId, String ip, String userAgent, String
resolution, Map<String, Object> properties) {
    HttpClient httpClient = HttpClients.createDefault();
    HttpPost httpPost = new HttpPost("http://little-telemetry-backend-che.apps-
crc.testing/event");
    HashMap<String, Object> eventPayload = new HashMap<String, Object>(properties);
    eventPayload.put("event", event);
    StringEntity requestEntity = new StringEntity(new JsonObject(eventPayload).toString(),
          ContentType.APPLICATION_JSON);
    httpPost.setEntity(requestEntity);
    try {
       HttpResponse response = httpClient.execute(httpPost);
    } catch (IOException e) {
       e.printStackTrace();
    }
}
```

これにより、HTTP リクエストが Telemetry サーバーに送信され、短期間で同じイベントが自動的にデバウンスされます。この場合、デフォルト値は 1500 ミリ秒です。サブクラスを設定してこの期間を変更できます。

9.3.7. increaseDuration() の実装

多くの Telemetry システムはイベント期間を認識します。AbstractAnalyticsManager は、同じ期間内に発生する同様のイベントを 1 つのイベントにマージし、ユーザーが短期間にサーバーに送信される複数の同じイベントを取得することがないようにします。increaseDuration() のこの実装は no-op です。この方法では、ユーザーの Telemetry プロバイダーの API を使用してイベントまたはイベントプロパティーを変更してイベントの長くなった期間を反映します。

AnalyticsManager.java

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

9.3.8. onActivity() の実装

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

AnalyticsManager.java

```
public class AnalyticsManager extends AbstractAnalyticsManager {

   ...

   private long inactiveTimeLimt = 60000 * 3;

   ...
```

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

9.3.9. destroy()の実装

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

AnalyticsManager.java

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

「アプリケーションの実行」 で説明されるように mvn quarkus:dev を実行すると、Quarkus アプリケーションの強制終了時に WORKSPACE_STOPPED イベントがサーバーに送信されるはずです。

9.3.10. Quarkus アプリケーションのパッケージ化

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

9.3.11. プラグイン用の meta.yaml の作成

ワークスペース Pod でカスタムバックエンドを実行する CodeReady Workspaces プラグインを表す meta.yaml 定義を作成します。meta.yaml の詳細は、「Che-Theia プラグインについて」 を参照してください。

meta.yaml

```
apiVersion: v2
publisher: demo-publisher
name: little-telemetry-backend
version: 0.0.1
type: Che Plugin
displayName: Little Telemetry Backend
description: A Demo telemetry backend
title: Little Telemetry Backend
category: Other
spec:
  workspaceEnv:
    - name: CHE_WORKSPACE_TELEMETRY_BACKEND_PORT
      value: '4167'
  containers:
  - name: YOUR BACKEND NAME
```

```
    image: YOUR IMAGE NAME
    env:
      - name: CHE_API
        value: $(CHE_API_INTERNAL)
```

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

新規 meta.yaml ファイルを参照する ConfigMap を作成します。

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

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

manifests.yaml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: apache
  namespace: <openshift-workspaces>
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    spec:
      volumes:
        - name: plugin-meta-yaml
          configMap:
            name: telemetry-plugin-meta
            defaultMode: 420
      containers:
        - name: apache
          image: 'registry.redhat.io/rhscl/httpd-24-rhel7:latest'
          ports:
            - containerPort: 8080
              protocol: TCP
          resources: {}
          volumeMounts:
            - name: plugin-meta-yaml
              mountPath: /var/www/html
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 25%
      maxSurge: 25%
  revisionHistoryLimit: 10
  progressDeadlineSeconds: 600
---
```

```
kind: Service
apiVersion: v1
metadata:
  name: apache
  namespace: <openshift-workspaces>
spec:
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  selector:
    app: apache
  type: ClusterIP
---
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: apache
  namespace: <openshift-workspaces>
spec:
  host: apache-che.apps-crc.testing
  to:
    kind: Service
    name: apache
    weight: 100
  port:
    targetPort: 8080
  wildcardPolicy: None
```

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

イメージがプルし、デプロイメントが起動するまで数分待機してから、meta.yaml が Web サーバーで利用可能であることを確認します。

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

このコマンドにより、meta.yaml ファイルが返されるはずです。

9.3.12. Telemetry プラグインを参照するよう CodeReady Workspaces を更新する

CheCluster カスタムリソースを更新し、CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS 環境変数を spec.server.customCheProperties に追加します。環境変数の値は、Web サーバー上の meta.yaml ファイルの場所の URL である必要があります。これは、oc edit checluster -n openshift-workspaces を実行してターミナルで変更を入力するか、または OpenShift コンソール (Installed Operators → Red Hat CodeReady Workspaces → Red Hat CodeReady Workspaces Cluster → codeready-workspaces → YAML) で CR を編集して実行できます。

```
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  creationTimestamp: '2020-05-14T13:21:51Z'
  finalizers:
    - oauthclients.finalizers.che.eclipse.org
```

```
  generation: 18
  name: codeready-workspaces
  namespace: <openshift-workspaces>
  resourceVersion: '5108404'
  selfLink: /apis/org.eclipse.che/v1/namespaces/che/checlusters/eclipse-che
  uid: bae08db2-104d-4e44-a001-c9affc07528d
spec:
  auth:
    identityProviderURL: 'https://keycloak-che.apps-crc.testing'
    identityProviderRealm: che
    updateAdminPassword: false
    oAuthSecret: ZMmNPRbgOJJQ
    oAuthClientName: eclipse-che-openshift-identity-provider-yrlcxs
    identityProviderClientId: che-public
    identityProviderPostgresSecret: che-identity-postgres-secret
    externalIdentityProvider: false
    identityProviderSecret: che-identity-secret
    openShiftoAuth: true
  database:
    chePostgresDb: dbche
    chePostgresHostName: postgres
    chePostgresPort: '5432'
    chePostgresSecret: che-postgres-secret
    externalDb: false
  k8s: {}
  metrics:
    enable: false
  server:
    cheLogLevel: INFO
    customCheProperties:
      CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS: 'http://apache-che.apps-
crc.testing/meta.yaml'
    externalDevfileRegistry: false
    cheHost: che-che.apps-crc.testing
    selfSignedCert: true
    cheDebug: 'false'
    tlsSupport: true
    allowUserDefinedWorkspaceNamespaces: false
    externalPluginRegistry: false
    gitSelfSignedCert: false
    cheFlavor: che
  storage:
    preCreateSubPaths: true
    pvcClaimSize: 1Gi
    pvcStrategy: per-workspace
status:
  devfileRegistryURL: 'https://devfile-registry-che.apps-crc.testing'
  keycloakProvisioned: true
  cheClusterRunning: Available
  cheURL: 'https://che-che.apps-crc.testing'
  openShiftoAuthProvisioned: true
  dbProvisioned: true
  cheVersion: 7.13.1
  keycloakURL: 'https://keycloak-che.apps-crc.testing'
  pluginRegistryURL: 'https://plugin-registry-che.apps-crc.testing/v3'
```

CodeReady Workspaces サーバーが再起動するのを待機し、新規ワークスペースを作成します。プラグインがワークスペースにインストールされていることを示す新たなメッセージを参照してください。

```
Pulling image "quay.io/eclipse/che-plugin-metadata-broker:v3.2.0"
Successfully pulled image "quay.io/eclipse/che-plugin-metadata-broker:v3.2.0"
Created container che-plugin-metadata-broker-v3-2-0
Started container che-plugin-metadata-broker-v3-2-0
Starting plugin metadata broker
List of plugins and editors to install
- ms-vscode/go/latest - This extension adds rich language support for the Go language
- demo-publisher/little-telemetry-backend/0.0.1 - A Demo telemetry backend
- eclipse/che-theia/7.13.1 - Eclipse Theia
```

起動したワークスペースで操作を実行し、それらのイベントを Telemetry サーバーログのサンプルで確認します。

9.4. WOOPRA TELEMETRY プラグイン

Woopra Telemetry プラグインは、Telemetry を Red Hat CodeReady Workspaces インストールから Segment および Woopra に送信するためにビルドされたプラグインです。このプラグインは ホストされた Che で使用されますが、Red Hat CodeReady Workspaces デプロイメントはこのプラグインを使用できます。有効な Woopra ドメインおよびセグメント書き込みキー以外の依存関係はありません。リンク (https://raw.githubusercontent.com/che-incubator/che-workspace-telemetry-woopra-plugin/master/meta.yaml [plugin's meta.yaml]) ファイルには、プラグインに渡すことのできる 5 つの環境変数が含まれます。

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

Red Hat CodeReady Workspaces インストールで Woopra プラグインを有効にするには、環境変数が正しく設定された HTTP サーバーに **meta.yaml** ファイルをデプロイします。次に、**CheCluster** カスタムリソースを編集
し、**spec.server.customCheProperties.CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS** フィールドを設定します。

```
spec:
  server:
    customCheProperties:
      CHE_WORKSPACE_DEVFILE_DEFAULT__EDITOR_PLUGINS: 'eclipse/che-machine-exec-plugin/7.20.0,https://your-web-server/meta.yaml'
```