Red Hat CodeReady Workspaces 2.3

エンドユーザーガイド

Red Hat CodeReady Workspaces 2.3 の使用

Last Updated: 2020-08-20

Red Hat CodeReady Workspaces 2.3 エンドユーザーガイド

Red Hat CodeReady Workspaces 2.3 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の情報

目次

第1章 DASHBOARD を使用した CODEREADY WORKSPACES の移動

Dashboard は、**http://<che-instance>.<IP-address> .mycluster.mycompany.com/dashboard/** などの URL からクラスター上でアクセスできます。このセクションでは、OpenShift でこの URL にアクセスする方法を説明します。

1.1. OAUTH を使用して OPENSHIFT で初めて CODEREADY WORKSPACES にログイン

このセクションでは、OAuth を使用して初めて 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連絡し、**Red Hat CodeReady Workspaces URL**を取得します。

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。**ユーザー** 名、**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

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

1.2. 新規ユーザーとして初めて登録するために、OPENSHIFT で CODEREADY WORKSPACES にログイン

このセクションでは、新しいユーザーとして初めて登録するために、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連絡し、**Red Hat CodeReady Workspaces URL**を取得します。

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. **新規のユーザー登録オプションを** 選択します。

3. アカウント情報を更新します。**ユーザー** 名、**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

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

1.3. OPENSHIFT 4 CLI を使用した CODEREADY WORKSPACES クラスター URL の検索

このセクションでは、OpenShift 4 CLI（コマンドラインインターフェース）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。
- ユーザーは CodeReady Workspaces インストールプロジェクトにあります。

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. 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/"//'
```

第2章 CHE-THEIA IDE の基本

このセクションでは、Red Hat CodeReady Workspaces のネイティブ統合開発環境である Che-Theia の基本ワークフローおよびコマンドを説明します。

- Che-Theia のカスタムコマンドの定義
- バージョン制御
- Troubleshooting

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 devfile でカスタムコマンドを定義できます。これは、ワークスペースで作業する場合に利用できます。

以下は、devfile の **commands** セクションの例です。

```
commands:
- name: theia:build
  actions:
  - type: exec
    component: che-dev
    command: >
             yarn
    workdir: /projects/theia
- name: run
  actions:
  - type: vscode-task
    referenceContent: |
           {
            "version": "2.0.0",
            "tasks":
            [
             {
              "label": "theia:watch",
               "type": "shell",
               "options": {"cwd": "/projects/theia"},
               "command": "yarn",
               "args": ["watch"]
             }
            ]
           }
- name: debug
  actions:
  - type: vscode-launch
    referenceContent: |
           {
            "version": "0.2.0",
            "configurations": [
             {
              "type": "node",
              "request": "attach",
              "name": "Attach by Process ID",
              "processId": "${command:PickProcess}"
```

```
        }
      ]
    }
```

CodeReady Workspaces コマンド

theia:build

- **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意味します。ユーザーは、コマンドが実行されるコンテナー内のコンポーネントを指定できます。
- **コマンド** フィールドには、実行の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。

Visual Studio Code(VS Code)タスク

run

- タイプは vscode **-task** です。
- このタイプのコマンドでは、referenceContent **フィールド** に VS Code 形式のタスク設定のコンテンツが含まれている必要があります。
- VS Code タスクの詳細は、「Visual Studio User Guide」ページ の Task セクションを参照してください。

vs コード起動の設定

debug

- タイプは vscode **-launch** です。
- VS Code 形式の起動設定が含まれます。
- VS コード起動設定の詳細は、Visual Studio のドキュメントページ の Debugging セクションを参照してください。

利用可能なタスクおよび起動設定の一覧は、以下を参照してください。

- **/projects/.theia ディレクトリーの tasks.json** ファイル
- **/home/theia/.theia ディレクトリー内の launch.json** ファイル
  theia **-ide** コンテナーから。

2.1.1. CHECH: Theia task type

devfile の **commands** セクションでは、CodeReady Workspaces ワークスペースを実行する際に、ユーザーは特定のアクションを自動化できます。

CodeReady Workspaces コマンドには、以下の 3 つのタイプがあります。

- EXEC:**exec タイプの各コマンドは、タイプ codeready の Che-Theia タスクに変換されます。codeready コマンドはシェルコマンドを実行しますが、通常の Che-Theia シェルタイプのタスクとは対照的に、ユーザーはコマンド実行用のワークスペースコンテナーを選択できま**

す。

- vscode-task: vscode-task は、特定のコマンドの referenceContentfield の内容を Che-Theia の ユーザーレベルのタスク設定に直接コピーする単一のコマンドタイプです。

- vscode-launch: vscode -task コマンドの関数として機能しますが、タスク設定にコピーされる代わりに、referenceContent フィールド の内容は launch.json 設定ファイルにコピーされます。

codeready タイプのタスク（ exec コマンドとも呼ばれる）は 、Terminal→Run Task メニューから、または My Workspace パネルでクリックして実行できます。その他のタスクは 、Terminal→Run Task でのみ利用できます。起動設定は Che-Theia デバッガーで利用できます。

たとえば、以下のようになります。

以下の例は、codeready タスクと vscode -task タスク サンプルです。

Che-Theia のカスタムコマンドの定義で、 theia ワークスペース に theia:watch および theia:build タスクを設定します。

- Theia:watch:watch: 監視 モード で theia プロジェクト を実行する。

- Theia:build: は、che-dev コンポーネントで theia プロジェクト をビルドします。

上記の例の theia :buildで使用されている codeready コマンドの説明

- 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意味します。

- ユーザーは、コマンドが実行されるコンテナー内のコンポーネントを指定できます。

- コマンド フィールドには、実行の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。

これらのタスクは 、Terminal→Run Task メニューからワークスペースで実行するか、My Workspace パネルでクリックします。

その他の例

- コード対応タイプの Theia task
- vscode-lunch task example'

2.1.2. 実行中およびデバッグ

che-Theia はデバッグアダプタープロトコルを 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方法の一般的な方法を定義します。これは、Che-Theia がすべての 実装 と連携することを意味します。

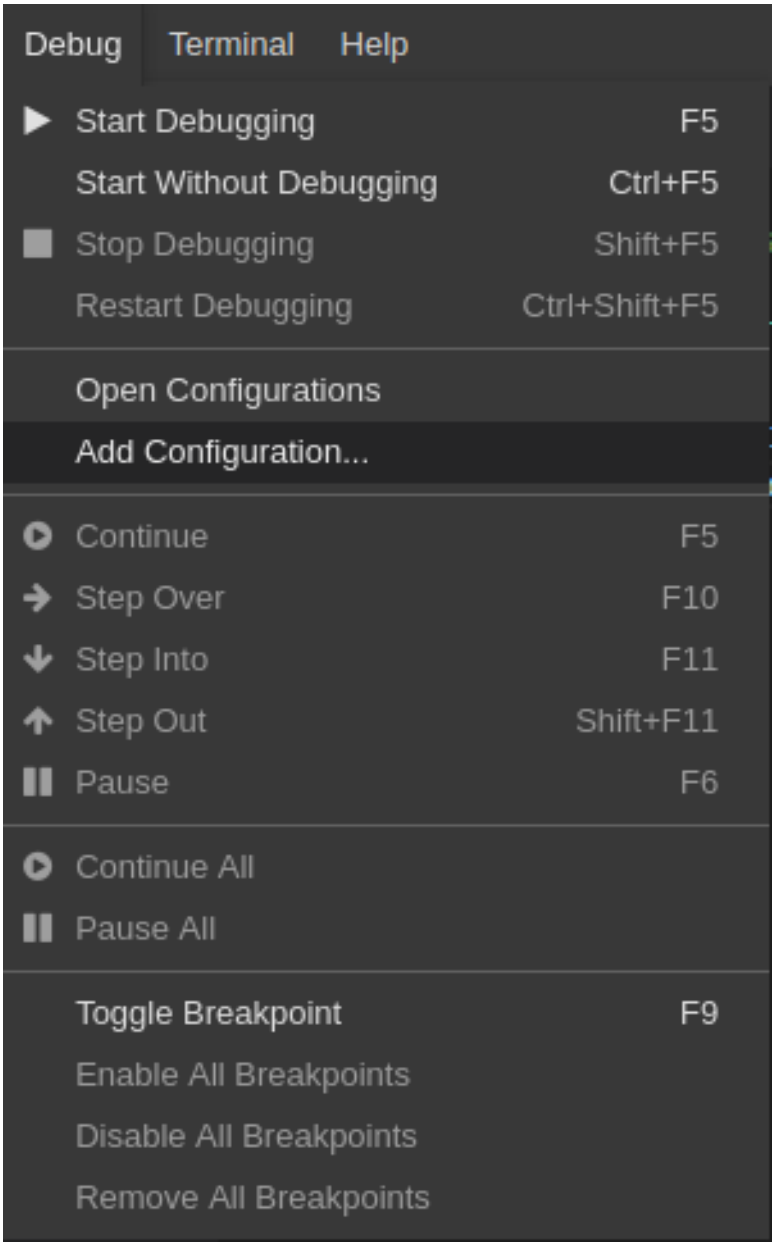
前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「 Installing CodeReady Workspaces on OpenShift Container Platform 」を参照してください。

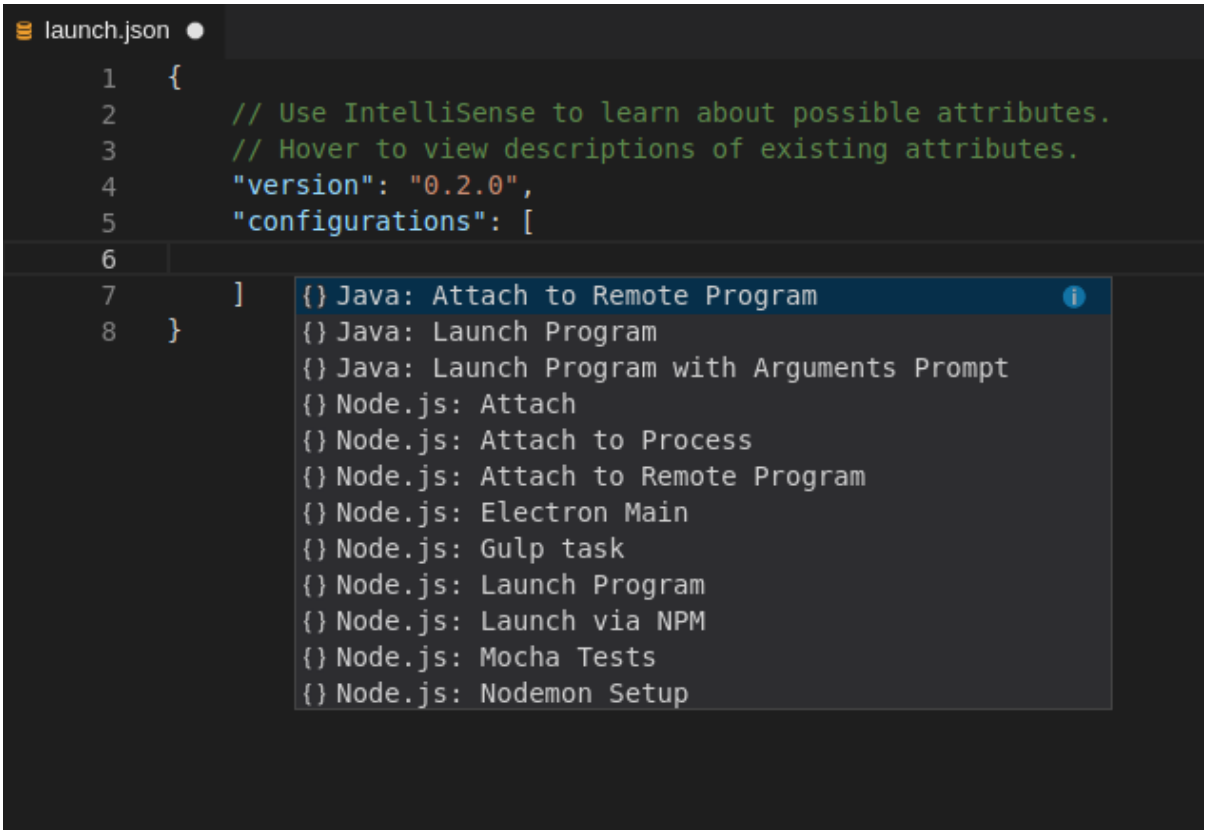
手順

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。

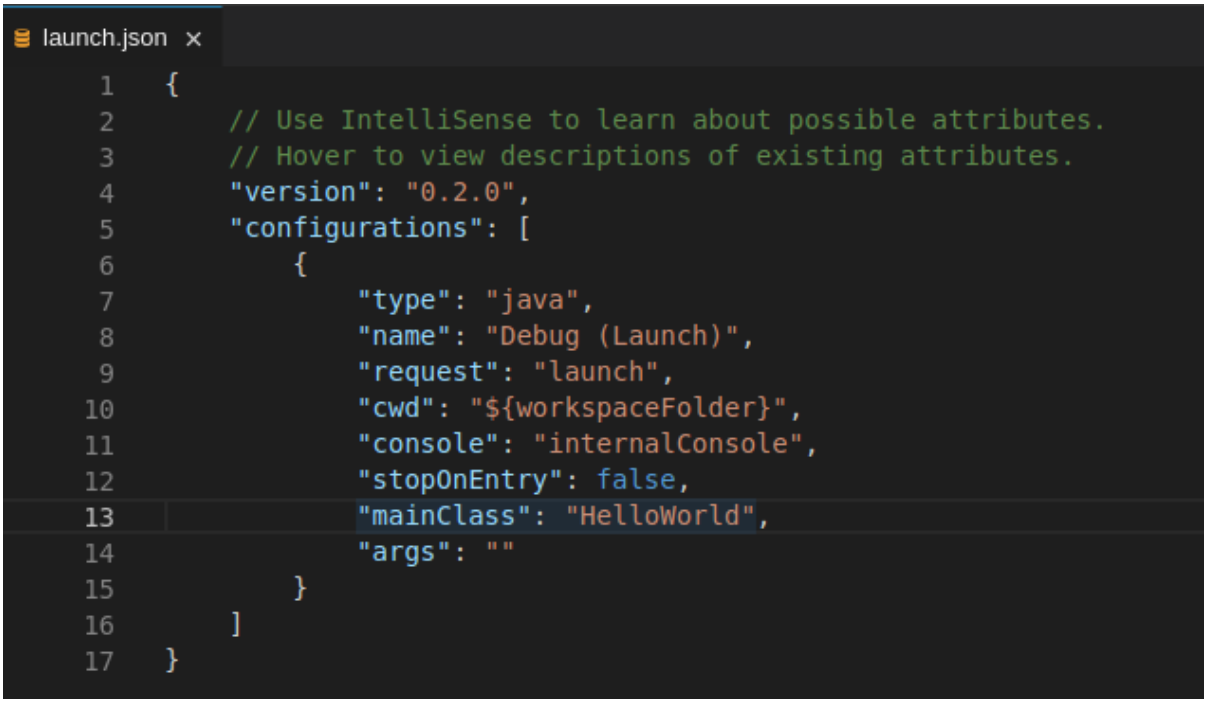
1. Debug → Add Configuration をクリックして、デバッグを追加するか、またはプロジェクトに設定を起動します。



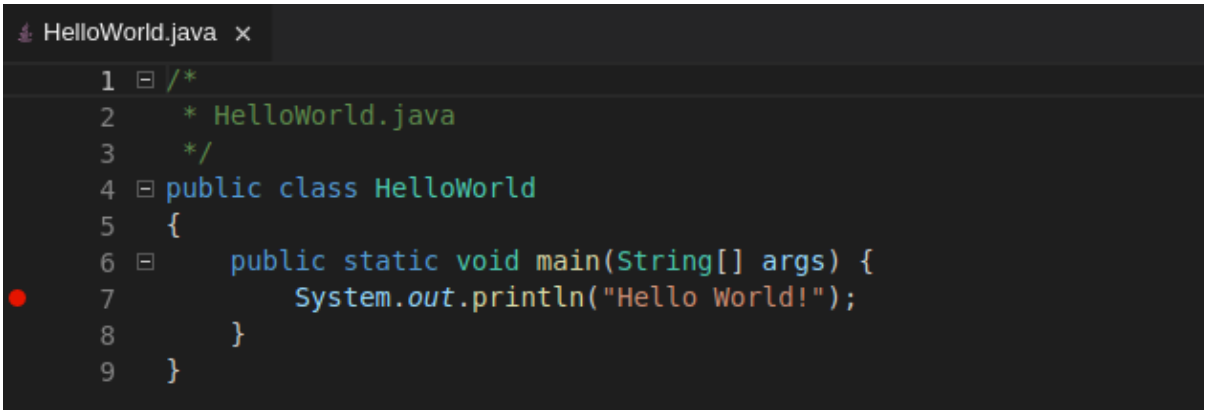
2. **ポップアップメニューから、デバッグするアプリケーションに適した設定を選択します。**



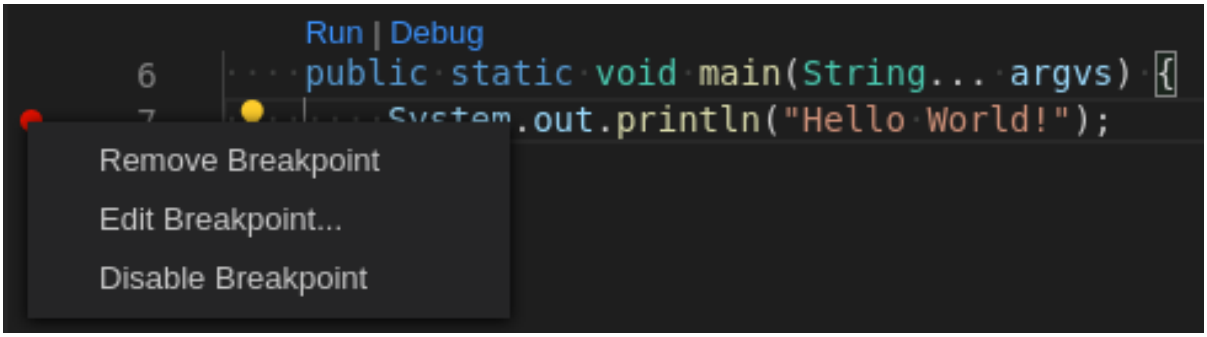
3. 属性を変更または追加して、設定を更新します。



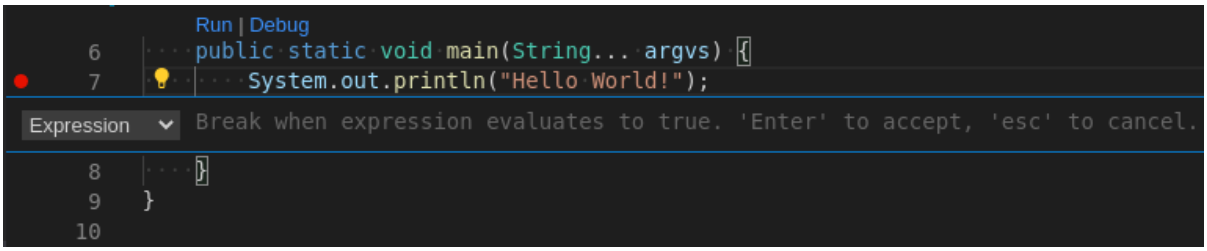
4. ブレークポイントは、エディターのマージンをクリックすると切り替えることができます。



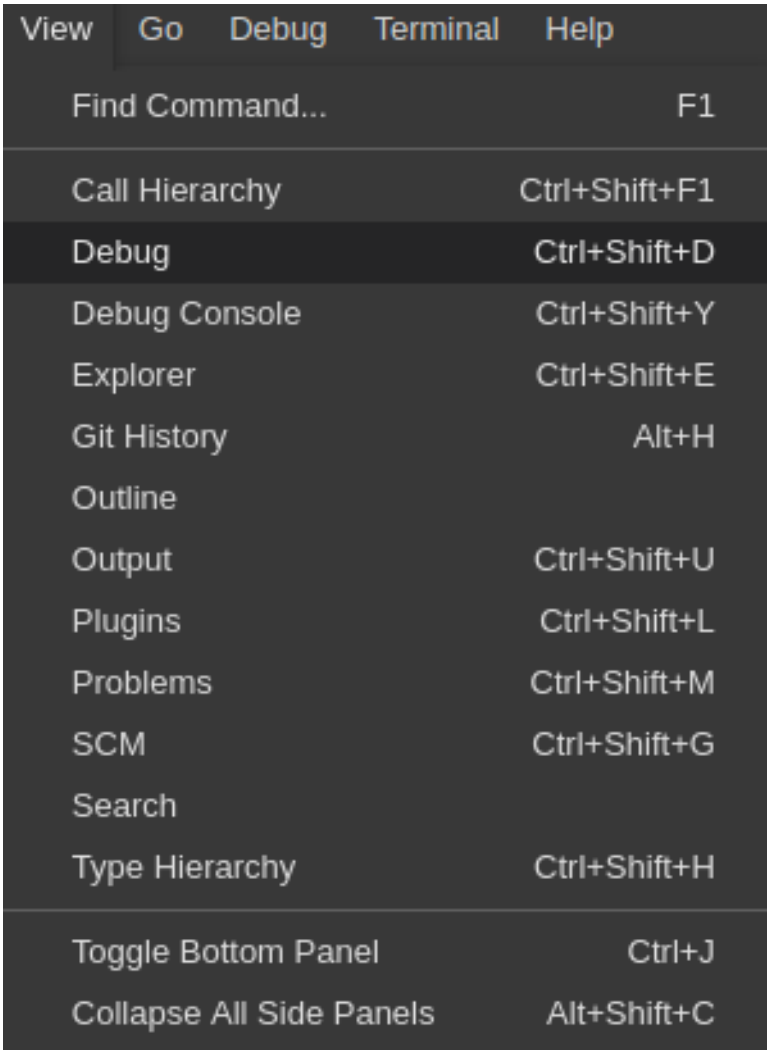
5. コンテキストメニューを開いた後に、**Edit Breakpoint** コマンド を使用して条件を追加します。



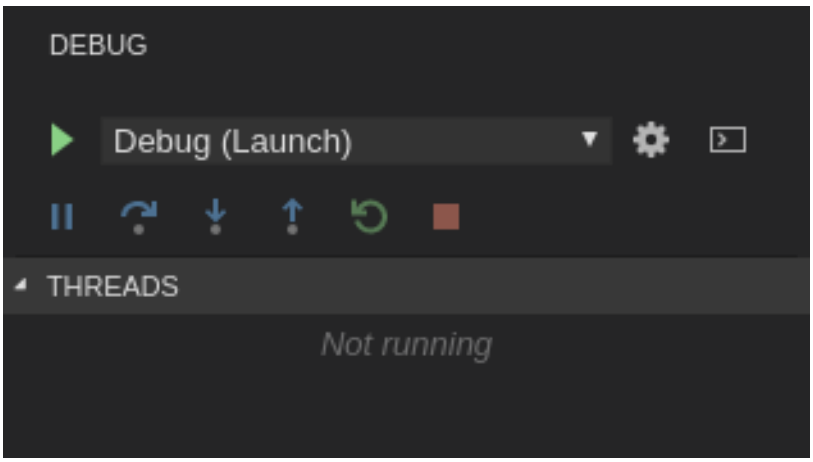
その後、**IDE** は **Expresion** 入力フィールド を 表示します。



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、デバッグなしでアプリケーションを起動します。**



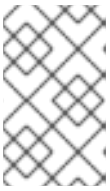
2.1.3. タスクの編集および設定の起動

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. 設定ファイルに新しい定義を追加するか、または既存の定義を変更します。



注記

この変更は設定ファイルに保存されます。

3. プラグインが提供するタスク設定をカスタマイズするには、Terminal → Configure Task メニュー オプションを選択し、設定するタスクを選択します。その後、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. バージョン制御

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ネイティブ VS Code Git 拡張 が含まれます。

2.2.1. Git 設定の管理： identity

Git を使用する前に最初に実行すべきことは、ユーザー名およびメールアドレスを設定することです。これは、すべての Git コミットでこの情報を使用するためです。

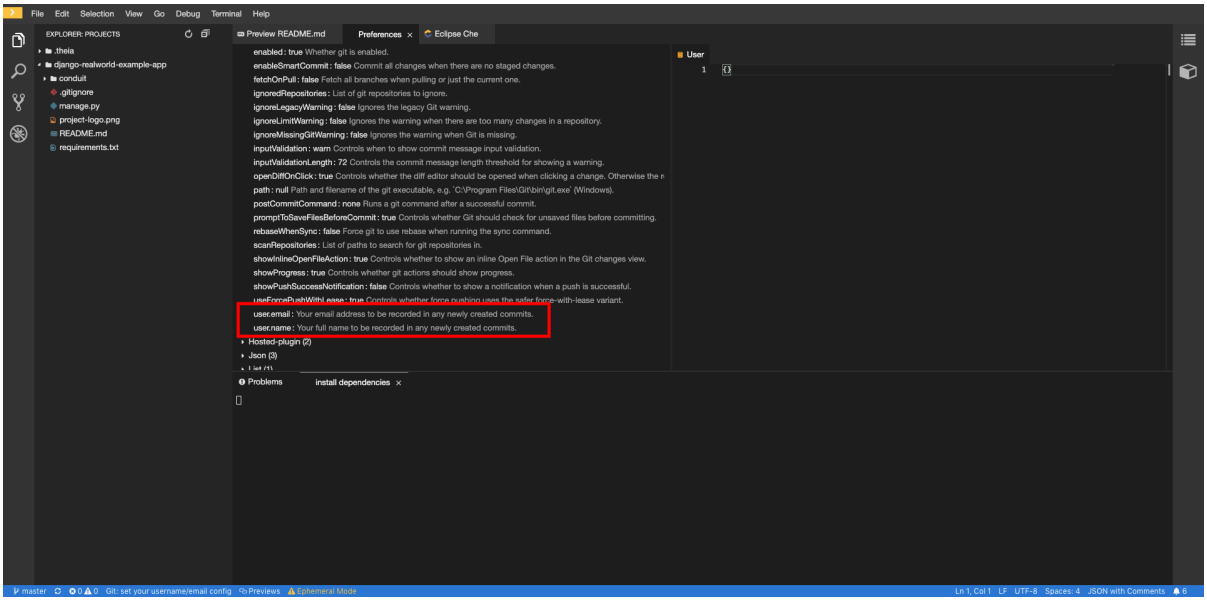
前提条件

- Visual Studio Code Git 拡張がインストールされている。

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、「 Preferences」 を参照してください。

1. File > Settings > Open Preferences:



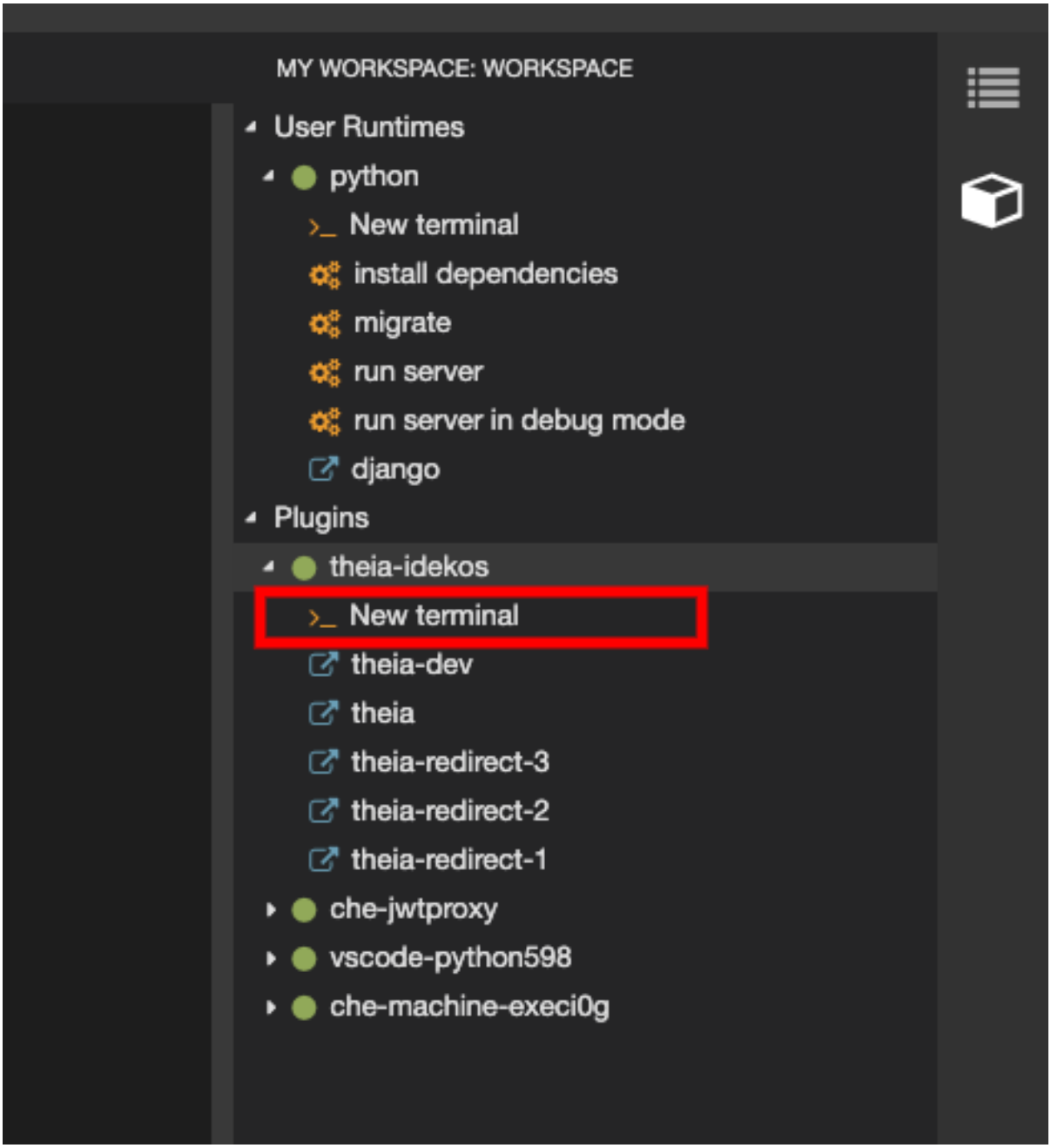
2. **開いているウィンドウで、Git セクションに移動し、以下を確認します。**

```
user.name
user.email
```

アイデンティティーを設定します。

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... > 新しいターミナル:**



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、この情報を永続的に保存し、今後のワークスペースで再開します。

2.2.2. HTTPS を使用した Git リポジトリーへのアクセス

前提条件

- **git ツールが利用可能である。「スタートガイド： Git のインストール」を参照してくだ**

さい。

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 拡張が提供する clone コマンドを使用します。

以下の手順では、ターミナルのネイティブ 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 自己署名の TLS 証明書をサポートします。詳細は、「 link:https://access.redhat.com/documentation/en-us/red_hat_codeready_workspaces/2.3/html/installation_guide/」を参照してください。

2.2.3. 生成された 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 コマンドを使用した SSH キーの生成と、さらに Git プロバイダー通信での使用を説明します。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、ユーザーは使用中の 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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces 'quick-starts' を参照してください。

- 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が 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。

- 個人の GitHub アカウント または他の Git プロバイダーアカウントが作成されている。

手順

すべての Git プロバイダーと機能する一般的な SSH キーペアがデフォルトで存在します。これを使用するには、公開鍵を Git プロバイダーに追加します。

1. 特定の Git プロバイダーでのみ機能する SSH キーペアを生成します。

   - CodeReady Workspaces IDE で F1 を押して Command Pal を開くか、トップメニューの View → Find Command に移動します。

     また、この コマンド をアクティベートするには、Ctrl+Shift+p （または macOS の場合は Cmd+Shift+p ）を押します。

   - SSH の検索 - 検索ボックスに generate を入力し、Enter を満たして、特定のホストの鍵ペアを 生成 します。

   - github.com などの SSH キーペアのホスト名を指定します。

     SSH キーペアが生成されます。

2. View ボタンをクリックし、エディターから公開鍵をコピーし、それを Git プロバイダーに追加します。

   このアクションにより、SSH で保護された URL を提供して、コマンドで Clone git リポジトリー から別のコマンドを使用できるようになりました。

2.2.3.2. 関連付けられた公開鍵を GitHub のリポジトリーまたはアカウントに追加

関連付けられた公開鍵を 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. ウィンドウの右上隅にあるユーザーアイコンの横にあるドロップダウン矢印をクリックします。

3. Settings → SSH および GPG 鍵 の順にクリックし、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 からコピーした公開鍵を貼り付けます。

5. SSH キーの追加 ボタンをクリックします。

2.2.3.3. 関連付けられた公開鍵を Git リポジトリーまたは GitLab のアカウントに追加

関連付けられた公開鍵を Git リポジトリーまたは GitLab の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. ウィンドウの右上隅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 からコピーした公開鍵を貼り付けます。

5. キーの追加 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要求の管理

GitHub プル要求を管理するには、VS Code GitHub Pull Request プラグインがワークスペースのプラグインの一覧で利用できます。

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。「 GitHub OAuth の設定 」を参照してください。

手順

1. GitHub の認証コマンドを実行して認証 します。
2. CodeReady Workspaces を許可するために GitHub にリダイレクトされます。
3. CodeReady Workspaces が承認されたら、CodeReady Workspaces を実行しているブラウザーページを更新し、プラグインを GitHub トークンで更新します。

GitHub Pull Requests: Manually Provide Authentication Response コマンドを実行して、GitHub トークンを手動でフェッチ し、プラグインに貼り付けます。

2.2.4.2. 新規プル要求の作成

1. GitHub リポジトリーを開きます。リモート操作を実行できるようにするには、リポジトリーに SSH URL がある リモート が必要になります。
2. 新しいブランチをチェックアウトし、公開する変更を加えます。
3. GitHub Pull Requests: Create Pull Request コマンドを実行します。

2.3. CHE-THEIA TROUBLESHOOTING

このセクションでは、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 is not working, please reload the page」というメッセージが表示されます。おそらく、プラグイン用に十分なメモリーがない可能性があります。

つまり、Che-Theia IDE コンテナーで実行している Che-Theia プラグインの 1 つには、コンテナーよりも多くのメモリーが必要です。この問題を解決するには、Che-Theia IDE コンテナーのメモリー容量を増やします。

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. 新しいプロパティー memoryLimit : 1024M を追加します（すでに存在する場合はこの値を増やします）。
6. 変更を保存し、ワークスペースを再起動します。

その他のリソース

- コミュニティーに Red Hat CodeReady Workspaces 専用の チャンネル をご質問ください。
- バグの報告： Red Hat CodeReady Workspaces リポジトリーの問題

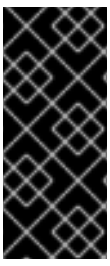
第3章 DEVELOPER WORKSPACES

Red Hat CodeReady Workspaces は、コード、ビルド、テスト、実行、デバッグのアプリケーションに必要なあらゆる内容を開発者ワークスペースに提供します。これを可能にするために、開発者ワークスペースは主に 4 つの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 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依存関係
4. アプリケーションランタイム： 実稼働環境でアプリケーションを実行する環境のレプリカ

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しているすべてはコンテナー内で実行されます。これにより、CodeReady Workspaces ワークスペースの移植性が非常に高くなります。

埋め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のアクセスのポイントとなります。これにより、CodeReady Workspaces ワークスペースを簡単に共有できます。



重要

デフォルトでは、一度に 1 つのワークスペースのみを実行できます。デフォルト値を変更するには、『CodeReady Workspaces 2.3 Installation Guide』を参照してください。

表3.1 機能および利点

Features	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に必要な	はい。	いいえ。

Features	従来の IDE ワークスペース	Red Hat CodeReady Workspaces ワークスペース
埋め込みツール	パーシャル。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および設定が必要です。例： JDK、Maven、Node	はい。プラグインはそれらの依存関係を提供します。
提供されるアプリケーションランタイム	いいえ、開発者がこれを別々に管理する必要があります。	はい。アプリケーションランタイムはワークスペースにレプリケートされます。
shareable	いいえ。簡単にはなりません。	はい。Developer workspaces と URL を共有できます。
Versionable	いいえ	はい。devfile は、プロジェクトのソースコードに存在します。
どこからでもアクセス可能	いいえ。インストールは必要ありません。	はい。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使用できます。

- **Dashboard を使用した新規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 2.3 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によるワークスペースの作成**

CodeReady Workspaces 2.3 ワークスペースを起動するのに、devfile を優先的に使用します。

- **devfile を使用したワークスペースの移植**

- 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 2.3 ワークスペースと対話する優先方法として、ブラウザーベースの IDE を使用します。CodeReady Workspaces 2.3 ワークスペースと対話する別の方法は、「リモートアクセスワークスペース」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 とそれらの使用手順の概要は、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
- 開発コンポーネント（ブラウザー IDE ツールおよびアプリケーションランタイム）
- 事前定義済みのコマンドの一覧
- 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消費し、変換する YAML ファイルです。devfile は Git リポジトリーのルートフォルダー、Git リポジトリーの機能ブランチ、一般にアクセス可能な宛先、またはローカルに保存されたアーティファクトとして保存できます。Git リポジトリーに保存される devfile は、devfile.yaml や .devfile.yaml などの複数の名前を使用でき ます。

ワークスペースの作成時に、CodeReady Workspaces はこ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に必要な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、ワークスペースでソースコードを使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付けできます。古いメンテナンスブランチを修正するためにワークスペースが必要な場合、プロジェクト devfile はワークスペースの定義と、古いブランチでの作業を開始するための正確な依存関係を提供します。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されるツールで devfile を最新の状態に維持します。

- ワークスペースのプロジェクト（パス、Git の場所、ブランチ）
- 毎日のタスクを実行するコマンド（build、run、test、debug）
- ランタイム環境（アプリケーションを実行するためのコンテナーイメージ）
- che-Theia プラグインには、開発者がワークスペースで使用するツール、IDE 機能、およびヘルパー関数が同梱されています（Git、Java サポート、PKarLint、Pull Request）。

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

CodeReady Workspaces ワークスペースは、Git ソースリポジトリーに保存されている devfile を参照して作成できます。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、/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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタート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関する詳細は、「devfile を使用したワークスペースポータブルの使用」を参照してください。

手順

https://codeready-<openshift_deployment_name>.<domain_name>/f?url=https://<GitRepository> の URL 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からのワークスペースの作成

CodeReady Workspaces ワークスペースを作成するには、ユーザーの任意の機能ブランチにある Git ソースリポジトリーに保存されている devfile を参照できます。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構築します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタート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選択の特定のブランチにある 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関する詳細は、「devfile を使用したワークスペースポータブルの使用」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 を使用した公開アクセス可能なスタンドアロンの devfile からのワークスペースの作成

ワークスペースは、devfile を使用して作成できます。これは、devfile の raw コンテンツを示す URL です。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

を構築します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『 CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタートを参照してください。

- パブリック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関する詳細は、「Devfile を使用したワークスペースポータブルの使用」を 参照してください。

手順

1. https://codeready-<openshift_deployment_name>.<domain_name>/f?url=https://<yourhosturl> /devfile.yamlの URL 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. ファクトリーパラメーターを使用した devfile 値のオーバーライド

リモート devfile の以下のセクションの値は、特別に構築された追加のファクトリー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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタート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関する詳細は、「Devfile を使用したワークスペースポータブルの使用」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>の URL に移動してワークスペースを開きます。

generateName プロパティーの オーバーライド 例

以下の初期 devfile を考慮してください。

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

generateName の値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&override.metadata.generateName=myprefix
```

その結果、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オーバーライド例

以下の初期 devfile を考慮してください。

```
---
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 ブランチ の値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&override.projects.web-java-spring-
petclinic.source.branch=1.0.x
```

その結果、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
...
```

属性値のオーバーライドまたは作成の例

以下の初期 devfile を考慮してください。

```
---
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&override.attributes.persistVolumes=true
```

その結果、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:
...
```

属性を上書きすると、attributes キーワードに続くすべての内容が属性名として解釈されるため、ドット区切り名を使用できます。

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

その結果、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

CodeReady Workspaces ワークスペースは、crwctl ツール をローカルに保存された devfile を参照して作成できます。CodeReady Workspaces インスタンスは、検出された devfile を使用してワー

クスペースを構築します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタートを参照してください。
- CodeReady Workspaces CLI 管理ツール。『CodeReady Workspaces 2.3 Installation Guide』を参照してください。
- devfile は、現在の作業ディレクトリー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関する詳細は、「Devfile を使用したワークスペースポータブルの使用」を 参照してください。

  例

  devfile.yaml ファイルを GitHub リポジトリー から現在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、workspace:start パラメーターで crwctl ツールを指定して、devfile からワークス ペース を実行します。

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

その他のリソース

- Devfile を使用したワークスペースの移植

3.2. DEVFILE を使用したワークスペースの移植

設定済みの 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 を作成してエクスポートし、別のホストに devfile を読み込み、ワークスペースの新しいインスタンスを初期化します。このような devfile の作成方法については、以下を参照してください。

3.2.1. 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
- 開発コンポーネント（ブラウザー IDE ツールおよびアプリケーションランタイム）
- 事前定義済みのコマンドの一覧
- 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消費し、変換する YAML ファイルです。devfile は Git リポジトリーのルートフォルダー、Git リポジトリーの機能ブランチ、一般にアクセス可能な宛先、またはローカルに保存されたアーティファクトとして保存できます。Git リポジトリーに保存される devfile は、devfile.yaml や .devfile.yaml などの複数の名前を使用でき ます。

ワークスペースの作成時に、CodeReady Workspaces はこ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に必要な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、ワークスペースでソースコードを使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付けできます。古いメンテナンスブランチを修正するためにワークスペースが必要な場合、プロジェクト devfile はワークスペースの定義と、古いブランチでの作業を開始するための正確な依存関係を提供します。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されるツールで devfile を最新の状態に維持します。

- ワークスペースのプロジェクト（パス、Git の場所、ブランチ）
- 毎日のタスクを実行するコマンド（build、run、test、debug）

- ランタイム環境（アプリケーションを実行するためのコンテナーイメージ）

- che-Theia プラグインには、開発者がワークスペースで使用するツール、IDE 機能、およびヘルパー関数が同梱されています（Git、Java サポート、PKarLint、Pull Request）。

3.2.2. 最小の devfile

以下は、devfile に必要な最低限のコンテンツです。

- apiVersion

- metadata name

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

完全な devfile の例は、CodeReady Workspaces devfile.yaml の Red Hat CodeReady Workspaces を参照してください。

NAME または GENERATENAME を定義 する 必要があります。

name と generateName は どちらもオプションのパラメーターですが、最低でも 1 つのパラメーターを定義する必要があります。を参照してください 「ワークスペース名の生成」。

3.2.3. ワークスペース名の生成

自動生成されたワークスペース名の接頭辞を指定するには、devfile に generateName パラメーターを設定します。

```
apiVersion: 1.0.0
metadata:
  generateName: che-
```

ワークスペース名は <generateName>YYYYY 形式 （例： che-2y7kp）になります。y は [a-z0-9] 文字です。

ワークスペースの作成時に、以下の命名ルールが適用されます。

- 名前 を定義すると、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、これは生成された名前のベースとして使用されます(<generateName>YYYYY) 。

注記

ファクトリーを使用して作成されたワークスペースの場合、名前 の定義 または generateName は同じ効果を持ちます。定義された値は名前プレフィックス <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、generateName が 優先されます。

3.2.4. プロジェクトの devfile の作成

本セクションでは、プロジェクトの最小 devfile の作成方法と、devfile に複数のプロジェクトを追加する方法を説明します。

3.2.4.1. 最小限の devfile の準備

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

- 仕様バージョン
- name

プロジェクトのない最小限の devfile の例

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

追加設定がないと、デフォルトのエディターを持つワークスペースが、CodeReady Workspaces サーバーに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。ファクトリーを使用して 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および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。その後、プロジェクト名はリポジトリー名と一致します。

より機能的なワークスペース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： 開発ツールの実行、ランタイム環境の起動などのワークスペースコンポーネントを管理するアクション

プロジェクトを含む最小の 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
```

3.2.4.2. devfile での複数プロジェクトの指定

1 つの devfile は複数のプロジェクトを指定できます。各プロジェクトに対して、ソースリポジトリーのタイプ、その場所を指定します。任意で、プロジェクトのクローン先のディレクトリーを指定し

ます。

2 つのプロジェクトを含む devfile の例

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

上記の例では、frontend と backend の 2 つのプロジェクトが定義されています。各プロジェクトは独自のリポジトリーに置かれます。バックエンドプロジェクトには、ソースルート下の（CodeReady Workspaces ランタイムで簡単に定義）の src/github.com/acmecorp/backend/ ディレクトリーにクローンする 固有 の要件があります。一方、フロント エンドプロジェクトはソースルートの下にある frontend/ ディレクトリーにクローンされます。

その他のリソース

devfile コンポーネントの割り当ておよび可能な値に関する詳細な説明は、以下を参照してください。

- specification repository
- json-schema の詳細なドキュメント

以下のサンプル devfile は、インスパイレーションの適切なソースです。

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

- Red HatDeveloper program の Red Hat CodeReady Workspaces ワークスペースの devfile のサンプル

3.2.5. devfile 参照

本セクションでは、devfile の参照と、devfile が構成するさまざまな要素の使用方法を説明します。

3.2.5.1. 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にワークスペースが作成されます。プロジェクトは、devfile の projects セクションで追加されます。

1 つの devfile の各プロジェクトには以下が必要です。

- 一意な名前
- ソースが指定されている

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

type

project-source 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 のソースを持つプロジェクト。場所はクローンリンクを参照します。

github

git と同じですが、GitHub でのみホストされるプロジェクト向けです。GitHub 固有の機能を

使用しないプロジェクトに git を使用します。

zip

ZIP アーカイブのソースがあるプロジェクト。location は ZIP ファイルを参照します。

3.2.5.1.1. project-source type: git

```
source:
  type: git
  location: https://github.com/eclipse/che.git
  startPoint: master (1)
  tag: 7.2.0
  commitId: 36fe587
  branch: master
  sparseCheckoutDir: wsmaster (2)
```

(1)

startPoint は、タグ、commitId 、 および ブランチ の一般値です。startPoint、tag、commitId 、 および branch パラメーターは相互排他的です。複数のものを指定すると、startPoint、tag、commitId、branch の順序が使用 さ れます。

(2)

sparseCheckout Git 機能のテンプレート。これは、プロジェクトの一部のみ（通常は単一のディレクトリーのみ）が必要な場合に便利です。

例3.1 sparseCheckoutDir parameter settings

- /my-module/ に設定して、ルート my-module ディレクトリー（およびそのコンテンツ）のみを作成します。
- スラッシュ(my-module/)を省略して、プロジェクトに存在する my-module ディレクトリーをすべて作成します。たとえば、/addons/my-module/ を追加します。

  スラッシュは、指定された名前を持つディレクトリー（コンテンツを含む）のみが作成されることを示しています。
- 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- で始まる指定プロジェクトのディレクトリーがすべてチェックアウトされます。

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

3.2.5.1.2. project-source type: zip

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

3.2.5.1.3. Project clone-path 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/ ディレクトリーと相対的である必要があり、/projects/ ディレクトリーはそのままにすることはできません 。デフォルト値はプロジェクト名です。

プロジェクトを含む 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
```

3.2.5.2. devfile へのコンポーネントの追加

1 つの devfile の各コンポーネントには一意な名前を付ける必要があります。

3.2.5.2.1. コンポーネントのタイプ： cheEditor

id を定義してワークスペースで使用されるエディターについて説明します。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は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、デフォルトの ID と同じ id で明示的に定義されたエディターにも提供されます（異なるバージョンであっても）。che-Theia は、CodeReady Workspaces Machine Exec プラグインと共にデフォルト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を必要としないように指定するには、devfile 属性で editorFree :true 属性 を使用します。

3.2.5.2.2. コンポーネントのタイプ： chePlugin

id を定義してワークスペースのプラグインについて説明します。これは複数の chePlugin コンポーネントを持つこと が できます。

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

上記のタイプとも ID を使用します。これは、CodeReady Workspaces プラグインレジストリーからのプラグイン名およびバージョンです。

利用可能な CodeReady Workspaces プラグインの一覧と、レジストリーの詳細は、CodeReady Workspaces 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。

3.2.5.2.3. 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
```

3.2.5.2.4. 記述子にリンクしてコンポーネントの指定

エディターまたはプラグイン ID （ または オプションで別のレジストリー）を使用する代わりに 、cheEditor または chePlugin を指定する別の方法は、参照 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 という名前の meta.yaml）への直接リンクを提供することです。

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

注記

単一コンポーネント定義で id および 参照 フィールドを混在させることはできません。これらは相互に排他的です。

3.2.5.2.5. chePlugin コンポーネント設定のチューニング

chePlugin コンポーネントは正確なチューニングが必要になる場合があります。この場合は、コンポーネント設定を使用できます。この例では、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'
```

設定はアレイ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"]
```

3.2.5.2.6. コンポーネントタイプ： kubernetes

OpenShift コンポーネントの一覧から設定を適用できるようにする複雑なコンポーネントタイプ。

コンテンツは、参照 属性から提供でき、コンポーネントコンテンツが含まれるファイルを参照します。

```
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
```

このようなコンポーネントを持つ devfile を REST API に投稿するには、referenceContent フィールド を使用して OpenShift リストの内容を devfile に埋め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
```

3.2.5.2.7. コンテナーエントリーポイントの上書き

dockerimage コンポーネント と同様に、（OpenShift で使用される） コマンド および args プロパティーを使用して、OpenShift リストに含まれるコンテナーのエントリーポイント を 上書きできます。

一覧には、さらに多くのコンテナーが存在する可能性があります（Pod またはデプロイメントの Pod テンプレートに含まれる）。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、コンテナーを選択するための制約と、それらに適用する コマンド および args パラメーターが含まれます。上記の例では、制約は parentName : mysqlServer で、 コマンドが mysqlServer という任意の親オブジェクトに定義されたすべてのコンテナーに適用され ます。parent オブジェクトは、参照されるファイルで定義される一覧の最上位オブジェクト（上記の例では app-deployment.yaml ）であると想定されます。

その他の制約のタイプ（およびその組み合わせ）が可能です。

containerName

コンテナーの名前

parentName

（間接的に）オーバーライドするコンテナーが含まれる親オブジェクトの名前

parentSelector

親オブジェクトが必要とするラベルのセット。

これらの制約の組み合わせを使用して、参照される OpenShift 一覧内でコンテナーを正確に特定できます。

3.2.5.2.8. コンテナーの環境変数の上書き

OpenShift または OpensShift コンポーネントで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や、参照されるコンテンツを編集できない場合に便利です。指定された環境変数は、すべての Pod およびデプロイメント内の各 init コンテナーおよびコンテナーにプロビジョニングされます。

3.2.5.2.9. mount-source オプションの指定

コンテナーにマウントされるプロジェクトソースディレクトリーを指定するには、mountSources パラメーターを使用 し ます。

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

有効にすると、プロジェクトソースマウントが指定の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 適用されます。

3.2.5.2.10. コンポーネントのタイプ： dockerimage

ワークスペース内の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できるコンポーネントタイプ。devfile には dockerimage タイプ のコンポーネントが 1 つだけ含まれます。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提供します。コンポーネントはそのイメージによって識別されます。

```
components:
 - alias: maven
   type: dockerimage
   image: eclipe/maven-jdk8:latest
   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
   command: ['tail']
   args: ['-f', '/dev/null']
```

最小限の 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、dockerimage、および image 属性のタイプを指定します。これは、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 と同等です。

dockerimage コンポーネント に は、Red Hat CodeReady Workspaces でイメージが提供するツール の有意な統合に必要な追加のリソースや情報でイメージを拡張できるようにする多くの機能があります。

3.2.5.2.10.1. 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
mountSources: true
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ている場所にマウントされます。この場所のデフォルトは /projects です。

3.2.5.2.10.2. 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 使用され、イメージから作成されたコンテナーの entrypoint コマンドを変更します。Red Hat CodeReady Workspaces では、コンテナーに接続し、いつでも任意のコマンドを実行できるように、コンテナーを無期限に実行する必要があります。sleep コマンドの可用性と、その引数の infinity 引数のサポートは異なり、特定のイメージで使用されるベースイメージに依存するため、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、この機能を利用して、変更した設定で必要なサーバーなどを起動できます。

3.2.5.2.11. 永続ストレージ

いずれのタイプのコンポーネントでも、イメージ内の特定の場所にマウントされるカスタムボリュームを指定できます。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ます。

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
```

3.2.5.2.12. コンポーネントのコンテナーメモリー制限の指定

dockerimage、chePlugin、cheEditor のコンテナーのメモリー制限を 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
 - type: dockerimage
   image: eclipe/maven-jdk8:latest
   memoryLimit: 512M
```

この制限は、指定の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 では、RAM 制限をプラグイン記述子ファイル（通常は meta.yaml ）に記述できます。

いずれも指定されていない場合、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）。

3.2.5.2.13. コンポーネントのコンテナーメモリー要求の指定

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
 - type: dockerimage
   image: eclipe/maven-jdk8:latest
   memoryLimit: 512M
   memoryRequest: 256M
```

この制限は、指定の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 については、RAM リクエストをプラグイン記

述子ファイル（通常は meta.yaml ）に記述できます。

いずれも指定されていない場合は、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 システムプロパティーの説明を参照）。

3.2.5.2.14. 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
 - type: dockerimage
   image: eclipe/maven-jdk8:latest
   cpuLimit: 750m
```

この制限は、指定の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 では、CPU 制限をプラグイン記述子ファイル（通常は meta.yaml ）に記述できます。

いずれも指定されていない場合は、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 システムプロパティーの説明を参照）。

3.2.5.2.15. 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
 - type: dockerimage
```

```
    image: eclipe/maven-jdk8:latest
    cpuLimit: 750m
    cpuRequest: 450m
```

この制限は、指定の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 については、通常 meta.yaml という名前のプラグイン記述子ファイルで CPU 要求を記述できます。

いずれも指定されていない場合は、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 システムプロパティーの説明を参照）。

3.2.5.2.16. 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 dockerimage、chePlugin、cheEditor、kubernetes 、 openshift のコンポーネントタイプ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、変数の参照に OpenShift の慣例を使用します。
- 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 サーバーにより事前設定されています。

- CHECH_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ことに注意してください）。
- CHECH_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。コンポーネントがこのディレクトリーにログインすることを選択する場合、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 API エンドポイントへの URL。CodeReady Workspaces サーバーとの通信に使用されます。
- CHECH_WORKSPACE_ID: 現在のワークスペースの ID。
- CHECH_WORKSPACE_NAME: 現在のワークスペースの名前。
- CHECH_WORKSPACE_NAMESPACE: 現在のワークスペースの 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 プロジェクトまたは OpenShift プロジェクトとは異なることに注意してください。
- CHECH_MACHINE_TOKEN: CodeReady Workspaces サーバーに対するリクエストの認証に使用されるトークン。
- CHECH_MACHINE_AUTH_SIGNATUREPUBLICKEY: 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される公開鍵。

- che_MACHINE_AUTH_SIGNATURE__ALGORITHM: CodeReady Workspaces サーバーとの通信でセキュアな通信で使用される暗号化アルゴリズム。

devfile は、CHE _PROJECTS_ROOT 環境変数のみが必要な場合があり、コンポーネントのコンテナーでクローンされたプロジェクトを見つける必要があります。より高度な devfile は、CHE _WORKSPACE_LOGS_ROOT__DIR 環境変数を使用してログを読み取る可能性があります（例：devfile コマンドの一部として）。CodeReady Workspaces サーバーへのアクセスに使用される環境変数は、大半は devfile のスコープ外であり、通常は 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でのみ存在します。

3.2.5.2.17. 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。CodeReady Workspaces クラスターが OpenShift Ingress または OpenShift ルートを使用し、ワークスペース内の他のコンポーネントを使用して実行している場合は、これらのエンドポイントにアクセスできます。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をリッスンし、直接対話できるようにする場合や、他のコンポーネントで対話できるようにする場合は、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
```

ここでは、2 つの Docker イメージがあり、1 つのエンドポイントを定義します。エンドポイントは、ワークスペース内や、一般に UI からアクセス可能な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があります。これは、コンテナー内で実行中の特定のサーバーがリッスンするポートです。エンドポイントに設定できる属性を以下に示します。

- discoverable: エンドポイントが検出可能な場合、その名前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ることを意味します（OpenShift パリスでは、指定された名前でサービスが作成されます）。55

- 公開: エンドポイントはワークスペースの外部でもアクセスできます（このようなエンドポイントは CodeReady Workspaces ユーザーインターフェースからアクセスできます）。このようなエンドポイントは、常にポート 80 または 443 で公開されます（CodeReady Workspaces で tls が有効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 を構築する方法の UI へのヒントになります。通常の値は http、https 、 wss です。

- Secure: アクセスを許可するために 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ブール値（ falseにデフォルト設定）。JWT プロキシーはサーバーと同じ Pod にデプロイされ、サーバーが 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、対応する IP アドレスのクラスターネットワーク内で JWT 認証がなくてもアクセスできるため、セキュリティーリスクが発生します。

- path: エンドポイントの URL。
- unsecuredPaths: secure 属性が true に設定されている場合でもセキュア でないままにするエンドポイントパスのコンマ区切りリスト。
- CookieAuthEnabled: true に設定すると（デフォルトは false）、JWT ワークスペーストークンは自動的に取得され、リクエストが JWT プロキシーを通過できるようにワークスペース固有のクッキーに含まれます。

警告

この設定では、POST リクエストを使用するサーバーとともに使用されると、CSRF 攻撃が可能になる可能性があります。

コンポーネント内で新しいサーバーを起動すると、CodeReady Workspaces がこれを自動検出し、UI はこのポートを パブリック 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。コンテナーで自動起動するサーバー（データベースサーバーなど）では、これは実行できません。このような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
```

3.2.5.2.18. OpenShift リソース

複合デプロイメントは、devfile で参照できる OpenShift リソース一覧を使用して記述できます。これにより、ワークスペースの一部になります。

重要

- CodeReady Workspaces ワークスペースは内部的に単一のデプロイメントとして表されるため、OpenShift 一覧のすべてのリソースはその単一デプロイメントにマージされます。
- このようなリストを設計する場合は注意してください。これは名前競合やその他の問題が発生する可能性があるためです。
- OpenShift オブジェクトのサブセットのみがサポートされます。デプロイメント、Pod、サービス、永続ボリューム要求、シーク レット、および ConfigMapKubernetes Ingress は無視されますが、OpenShift ルートがサポートされます。他のオブジェクトタイプを使用して devfile から作成されたワークスペースは起動できません。
- OpenShift クラスターで CodeReady Workspaces を実行する場合、OpenShift リストのみがサポートされます。OpenShift クラスターで CodeReady Workspaces を実行する場合、両方の OpenShift リストがサポートされます。

```
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
```

上記のコンポーネントは、devfile 自体の場所に関連する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 ファクトリーによってのみロードできるため、参照される OpenShift リソース一覧の場所を判別できます。

postgres.yaml ファイルの例を以下に示します。

```
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 リストを含む devfile の基本例は、「 web-nodejs-with-db-sample on

redhat- developer GitHub」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きなリソース一覧を使用している場合、セレクターを使用して特定のリソースを選択できます（通常の OpenShift セレクターと同様に、一覧内のリソースのラベルで機能します）。

```
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
components:
  - type: kubernetes
    reference: ../relative/path/postgres.yaml
    selector:
      app: postgres
```

さらに、リソース一覧に存在す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の詳細は、参考(TODO: link)を参照してください。

3.2.5.3. devfile へのコマンドの追加

devfile では、ワークスペース内で実行できるようにコマンドを指定できます。すべてのコマンドには、コンテナーが実行される特定のコンポーネントに関連するアクションのサブセットを含めることができます。

```
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。コードを構築およびテストするためのコマンドを定義するか、またはデータベースの消去を行うことができます。

以下は、2 種類のコマンドになります。

-

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定義： エディター固有のコマンド定義（例： tasks.json および launch.json ）を Che-Theia で使用することができます。これは、VS コードでファイルがどのように機能するかと同じです。

3.2.5.3.1. CodeReady Workspaces 固有のコマンド

各 CodeReady Workspaces 固有のコマンド機能

- 実行するコマンドである action 属性。
- コマンドを実行するコンテナーを指定する コンポーネント 属性です。

> The commands are run using the default shell in the container.

```
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エイリアスが必要です。このエイリアスは、コマンド定義内のコンポーネントを参照するために使用されます。例： alias: go-cli in the component 定義および component: go-cli in the command 定義。これにより、Red Hat CodeReady Workspaces がコマンドを実行する正しいコンテナーを見つけることができます。

- コマンドには 1 つのアクションのみを使用できます。

3.2.5.3.2. 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これに対応している場合は、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存在するインテグレーションコードに依存するため、汎用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 ファイルを理解するために追加されています。

```
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、tasks.json ファイルの devfile との関連付けを示しています。このコマンドをタスク

定義およびファイル自体が含まれる referenceContent 属性 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 に注意してください。また、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 URL または絶対 URL を指定することもできます。

vscode -task コマンドの他に、Che-Theia エディターは、起動設定を指定することのできる vscode -launch タイプを認識します。

3.2.5.3.3. 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くために提供されます。

```
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。必須のパラメーター。

(2) UI への URL のパスの部分です。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 を 開きます。ここで、<server-domain> は動的に作成された OpenShift Ingress または OpenShift Route の URL です。

3.2.5.3.3.1. プレビュー URL を開くデフォルトの方法の設定

デフォルトでは、開始する URL についてユーザーに質問する通知が表示されます。

サービス URL をプレビュー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 preferences を開

き、CodeReady Workspaces セクションの che.task.preview.notifications を検索します。

2. 可能な値の一覧から選択します。

- on - URL を開く設定についてユーザーに質問する通知を有効にします。
- alwaysview - タスクの実行直後に プレビュー パネルで自動的にプレビュー URL が開きます。
- alwaysGoTo - タスクの実行直後に、別のブラウザータブでプレビュー URL が自動的に開きます。
- off - プレビュー URL（自動および通知）を開くことを無効にします。

3.2.5.4. devfile 属性

devfile 属性は、さまざまな機能の設定に使用できます。

3.2.5.4.1. attribute: editorFree

エディターが devfile に指定されていない場合、デフォルトが提供されます。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
    local: my-app.yaml
attributes:
  editorFree: true
```

3.2.5.4.2. attribute: persistVolumes（一時モード）

デフォルトで、devfile で指定されたボリュームおよび PVC はホストフォルダーにバインドされ、コンテナーの再起動後もデータを永続化します。ボリュームのバックエンドの速度が遅いなど、データの永続性を無効にしてワークスペースを高速にするには、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。

一時モードが有効な 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
```

3.2.6. Red Hat CodeReady Workspaces 2.3 でサポートされているオブジェクト

以下の表は、Red Hat CodeReady Workspaces 2.3 で一部サポートされているオブジェクトの一覧です。

オブジェクト	API	OpenShift Infra	OpenShift Infra	注記
Pod	OpenShift	○	○	-
deployment	OpenShift	○	○	-
ConfigMap	OpenShift	○	○	-
PVC	OpenShift	○	○	-
Secret	OpenShift	○	○	-

オブジェクト	API	OpenShift Infra	OpenShift Infra	注記
service	OpenShift	◯	◯	-
Ingress	OpenShift	◯	いいえ	minishift を使用すると、Ingress の作成が可能になり、ホストが指定されている時に機能します（OpenShift がこれ用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ていません。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	いいえ	◯	OpenShift レシピは OpenShift インフラストラクチャーと互換性があり、指定されるルートの代わりに Ingress を生成する必要があります。
template	OpenShift	◯	◯	OpenShift API はテンプレートをサポートしません。レシピのテンプレートを持つワークスペースが正常に起動し、デフォルトのパラメーターは解決されます。

その他のリソース

- devfile 仕様

3.3. 新しい CODEREADY WORKSPACES 2.3 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

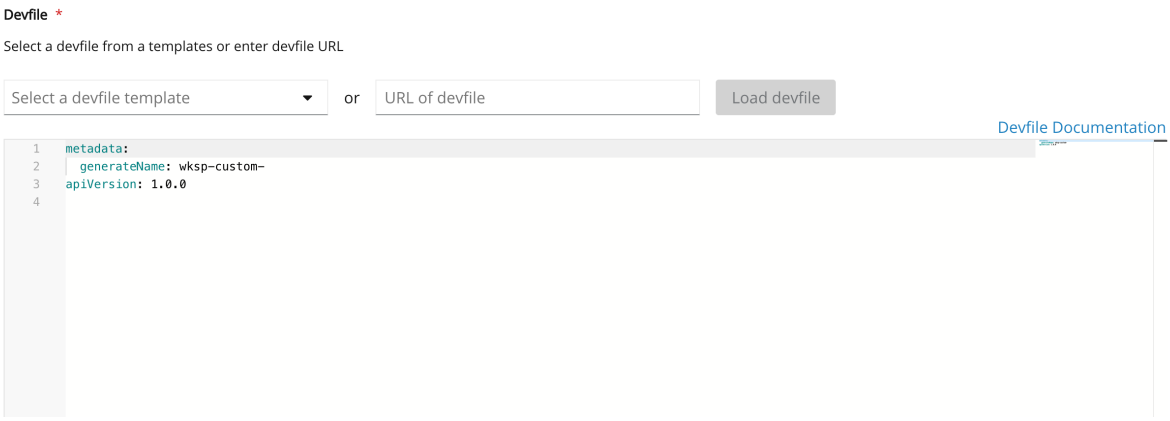
- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」を参照してください。

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。カスタムの 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します。Devfile エディター で必要な変更を追加します。



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 reference 」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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「 Installing CodeReady Workspaces on OpenShift Container Platform 」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が 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。

手順

プロジェクトをワークスペースに追加するには、以下を実行します。

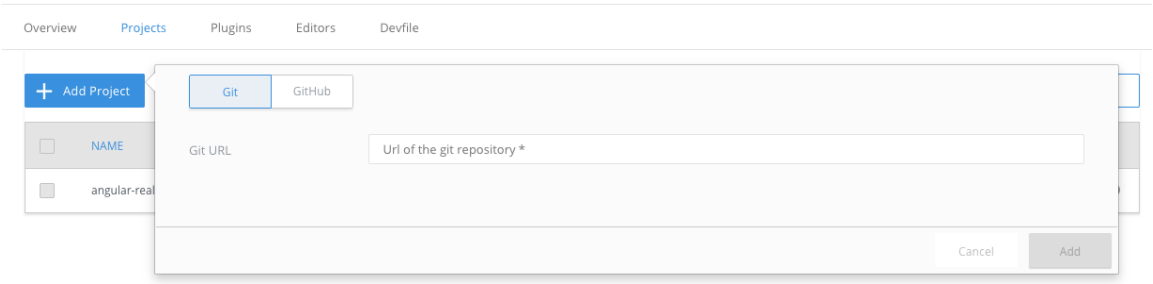
1. Workspaces ページに移動し、更新するワークスペースをクリックします。

   ここでは、プロジェクトをワークスペースに追加する 2 つの方法があります。

2. Projects タブから。

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

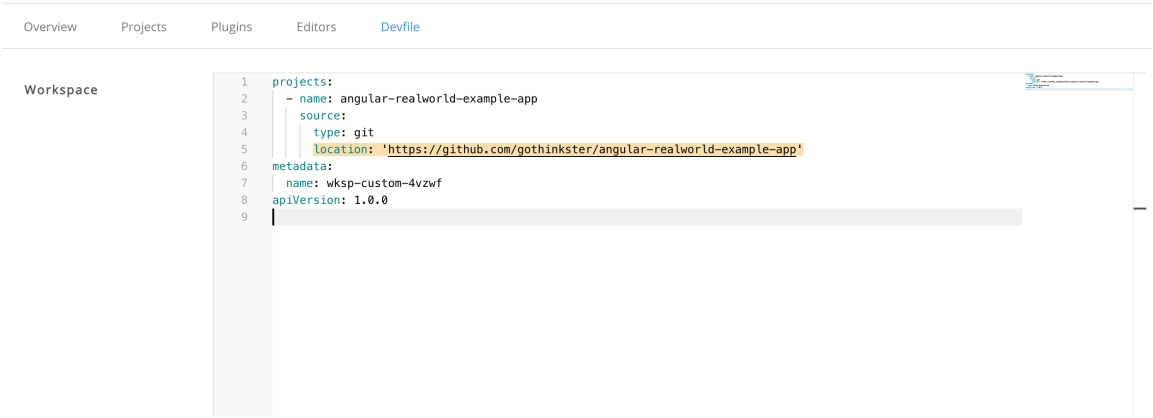
   b. Git URL または GitHub アカウントでプロジェクトをインポートするかを選択します。



3. **Devfile タブから。**

a. **Devfile タブを開きます。**

b. **Devfile エディター で、必要なプロジェクトで projects セクションを追加します。**



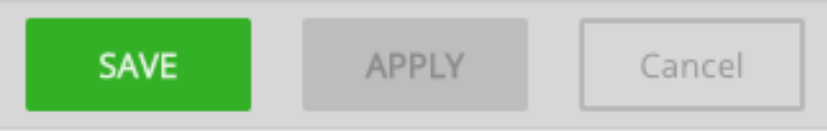
例： プロジェクトの追加

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 reference 」を参照してください。

4. **プロジェクトを追加したら、保存 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、適用 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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」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が 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。

手順

ワークスペースにプラグインを追加するには、以下を行います。

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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」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が 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。

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。

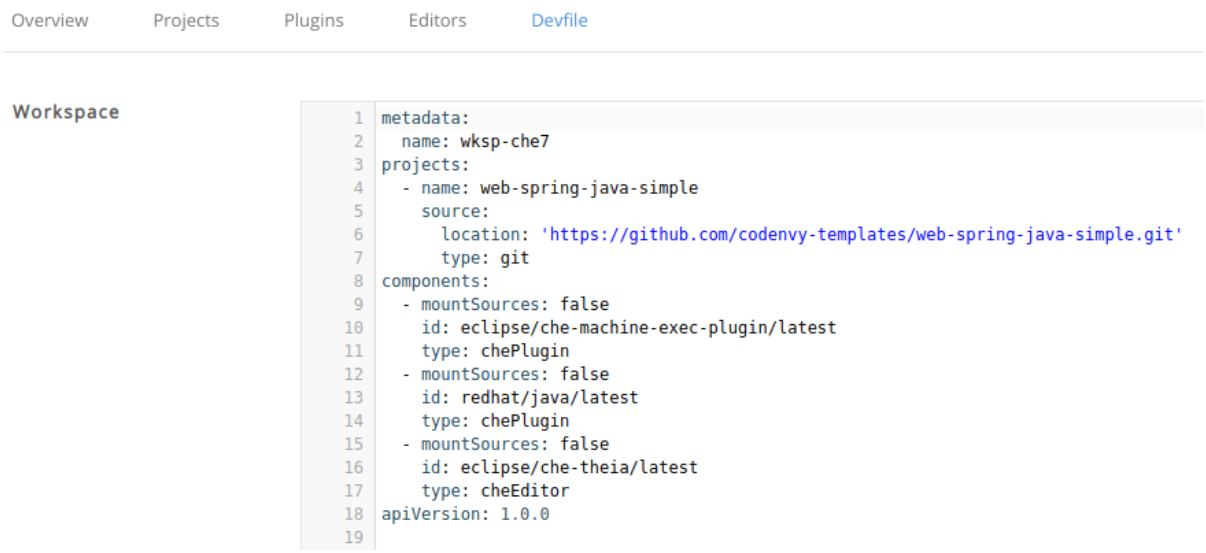
1. Editors タブをクリックします。

注記

CodeReady Workspaces 2.3 に推奨されるエディターは Che-Theia です。

2. エディターを有効にして追加し、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

手順

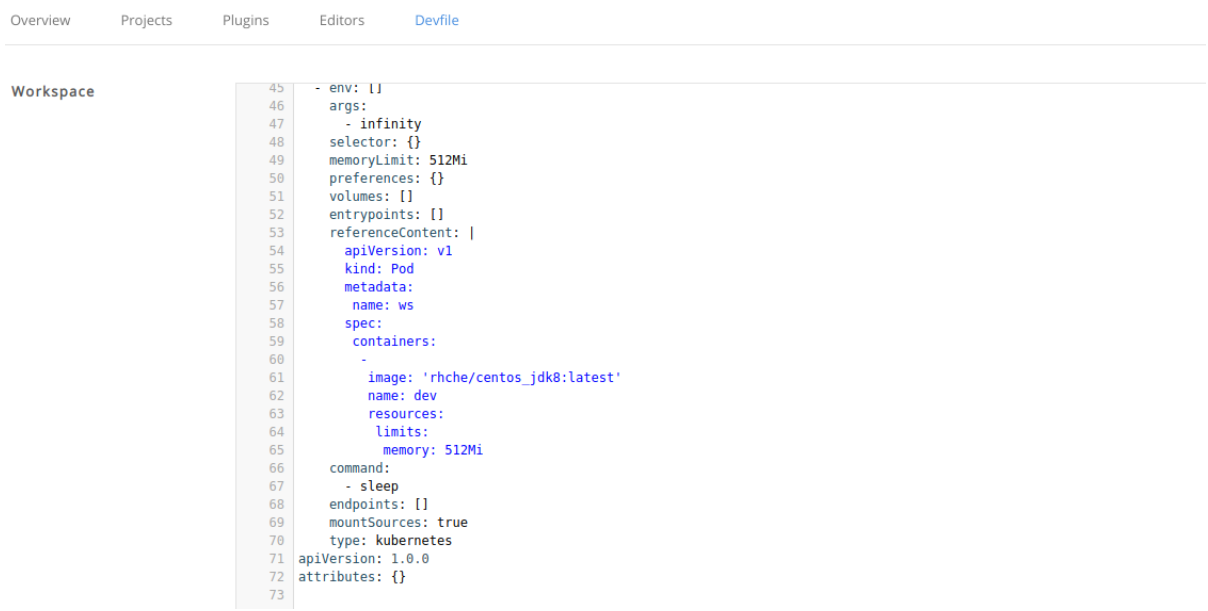
新しいコンテナーイメージを追加するには、以下を行います。

1. Devfile タブで、components プロパティー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 2.2 レシピコンテンツを referenceContent として CodeReady Workspaces 2.3 devfile に追加し ます。



a. 元の CodeReady Workspaces 2.2 設定からタイプを設定します。作成されたファイルの例を以下に示します。

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

3.

必要なフィールドをワークスペース（イメージ、ボリューム、エンドポイント）からコピーします。以下に例を示します。

```
17      endpoints:
18        - name: 8080/tcp
19          port: 8080
20      volumes:
21        - name: m2
22          containerPath: /home/user/.m2
23      image: 'maven:3.6.0-jdk-11'
```

4. memoryLimit および alias 変数（必要な場合）を変更します。ここでは、フィールドエイリアス を 使用してコンポーネントの名前を設定します。設定されていない場合は、イメージフィールドから自動的に生成されます。

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

5. memoryLimit、memoryRequest 、 または両方のフィールドを変更して、コンポーネントに必要な RAM を指定します。

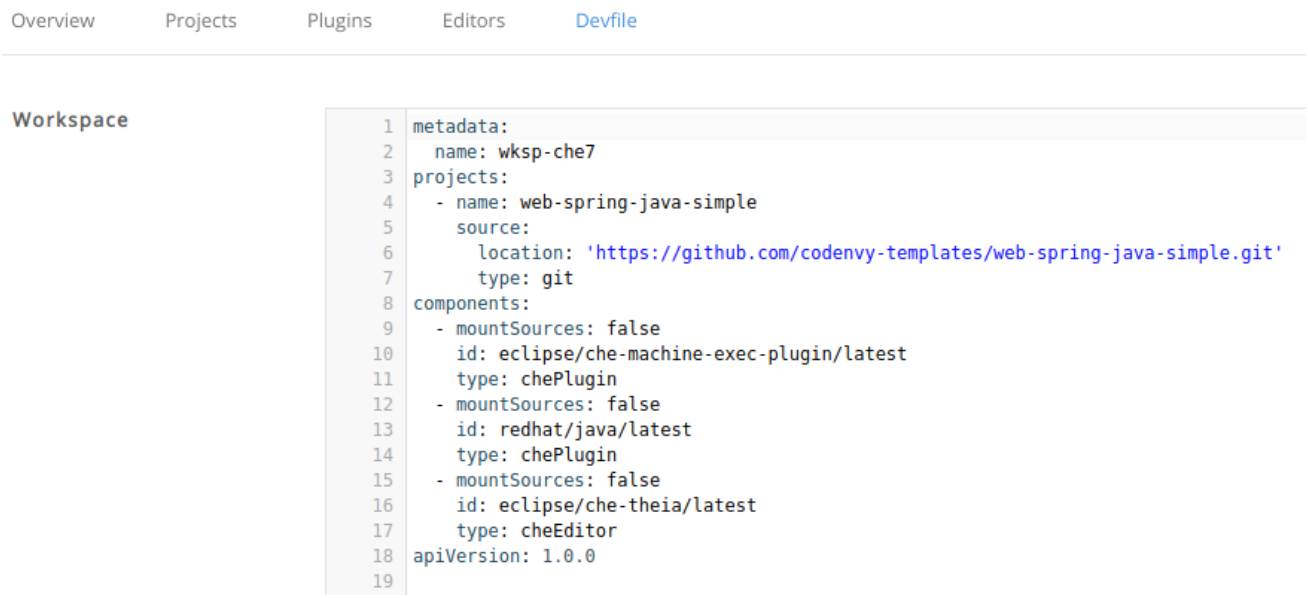
```
alias: maven3-jdk11
memoryLimit: 256M
memoryRequest: 128M
```

6. 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

以下は、CodeReady Workspaces 2.2(｜ 1)と CodeReady Workspaces 2.3(2)のワークスペース設定コマンドの比較です。

図3.1 CodeReady Workspaces 2.3 の Workspace 設定コマンドの例



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 devfile を編集します。

1. **commands セクションを最初のコマンドに追加（または置き換え）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（前述の等価テーブルを参照）。**

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

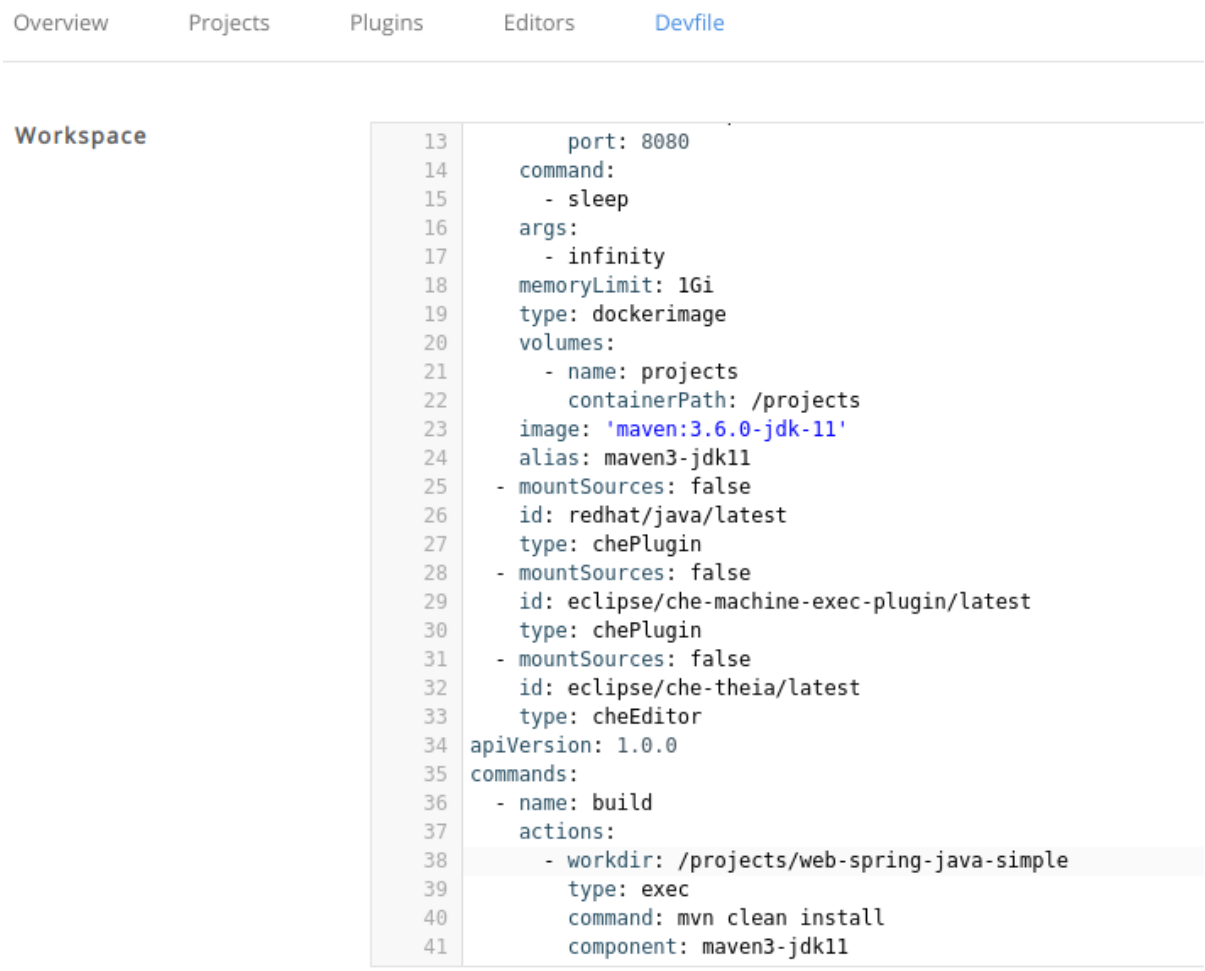
2. **以下の YAML コードを commands セクションにコピーし、新規コマンドを追加します。元のワークスペース設定から name および command フィールドを変更します（前述の等価テーブルを参照）。**

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

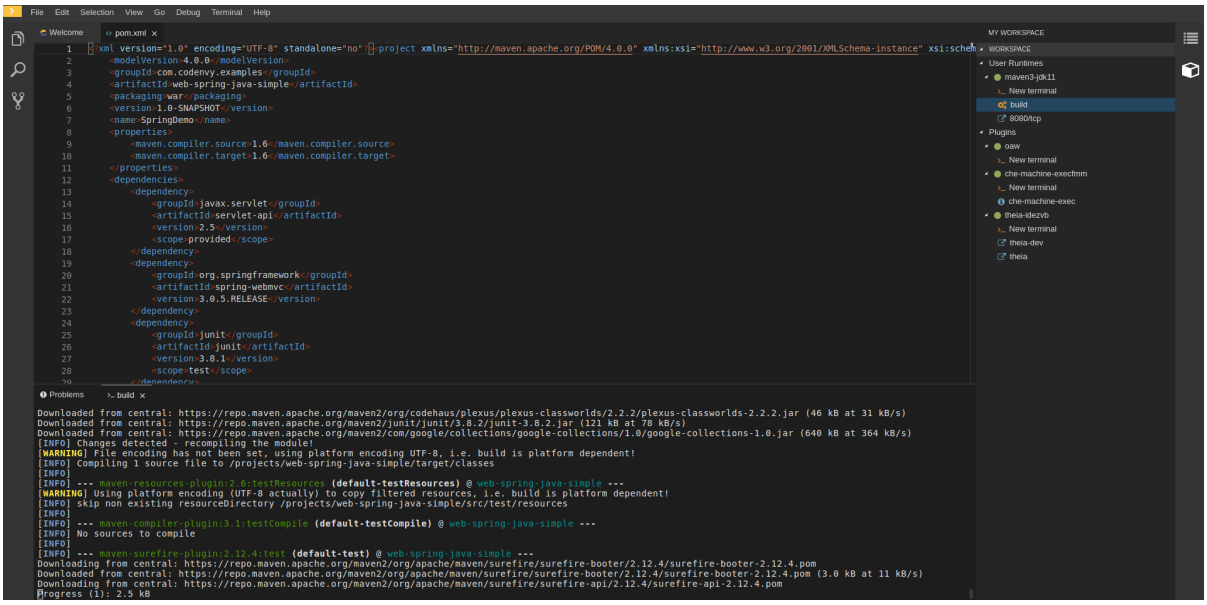
3. **必要に応じて、component フィールドを actions に追加します。これは、コマンドが実行されるコンポーネントのエイリアスを示します。**

4. **ステップ 2 を繰り返して、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 2.3** 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

このセクションでは、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、セクションは以下の 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によって指定される 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 ワークスペースでこの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 つのシナリオのいずれかを使用します。

- Starting from scratch: 新しい devfile の書き込み
- 既存の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から： devfile を crwctl で 生成

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

この手順では、OpenShift アプリケーションで CodeReady Workspaces 2.3 ワークスペース devfile を定義する方法を説明します。

前提条件

-

Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「 Installing CodeReady Workspaces on OpenShift Container Platform 」を参照してください。

- crwctl 管理ツールが利用できます。『 CodeReady Workspaces 2.3 Installation Guide』を参照してください。

devfile 形式は CodeReady Workspaces ワークスペースの定義に使用され、その形式は「 devfile を使用したワークスペースポータブルの作成」セクションを 参照してください。以下は、最も単純な devfile の例です。

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

名前(minimal-workspace)のみが指定されます。CodeReady Workspaces サーバーがこの devfile を処理すると、デフォルトのエディター(Che-Theia)およびデフォルトのエディタープラグイン（ターミナル）のみを持つ最小限の CodeReady Workspaces ワークスペースに devfile が変換されます。

devfile の OpenShift タイプのコンポーネントを使用して、OpenShift アプリケーションをワークスペースに追加します。

たとえば、ユーザーは、コンポーネント セクションを追加することで、このパラグラフで定義された minimal-workspace に NodeJS-Mongo アプリケーションを埋め込むことができます。

```
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']
```

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ることに注意してください。これにより、アプリケーションがワークスペースの開始フェーズで起動できなくなります。これにより、ユーザーは、テストまたはデバッグの目的で必要なときに起動できます。

開発者がアプリケーションのテストを容易にするには、devfile にコマンドを追加します。

```
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
   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
```

この devfile を使用して、crwctl コマンドでワークスペースを作成および 起動 します。

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

devfile に追加された run コマンドは、Che-Theia のコマンドのタスクとして利用できます。実行中、コマンドは Node.JS アプリケーションを起動します。

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

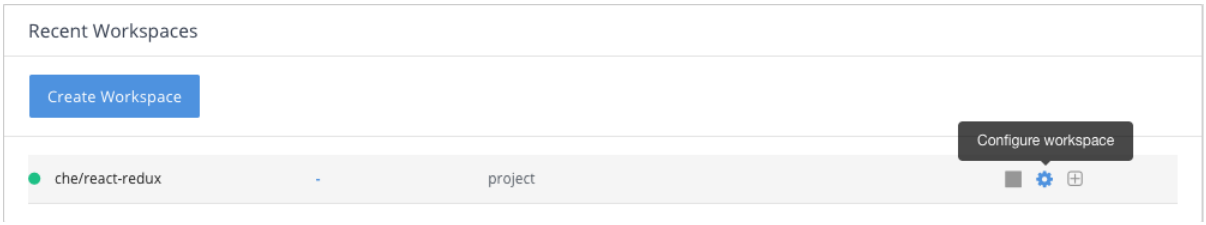
この手順では、既存のワークスペースを変更し、新たに作成された devfile を使用して OpenShift アプリケーションをインポートする方法を説明します。

前提条件

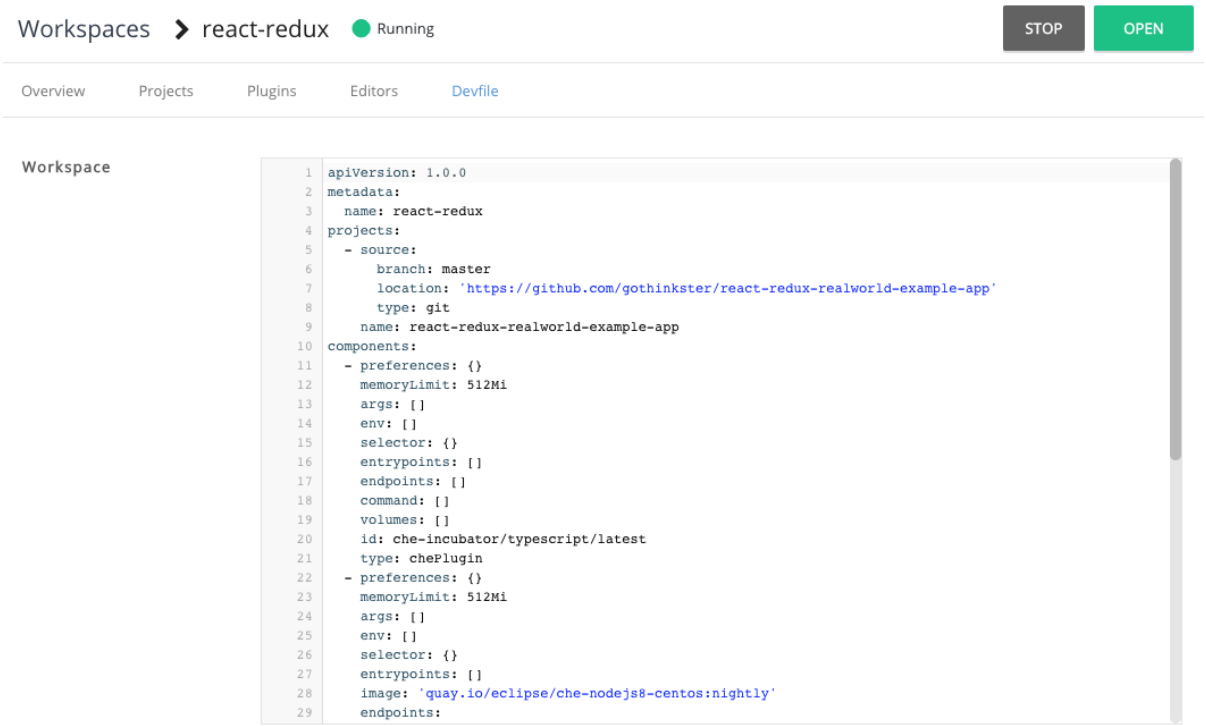
- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」を参照してください。
- Red Hat CodeReady Workspaces のこのインスタンスで定義された既存のワークスペースが ユーザーダッシュボードからワークスペースを作成 します。

手順

1. ワークスペースを作成したら、ワークスペースメニューを 使用 して ワークスペースを設定 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、このタブが 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、Dashboard 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、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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「 Installing CodeReady Workspaces on OpenShift Container Platform 」を参照してください。

- crwctl 管理 ツール が利用可能である。『 CodeReady Workspaces 2.3 Installation Guide』を参照してください。

手順

1. crwctl devfile:generate コマンドを使用して devfile を生成します。

```
$ crwctl devfile:generate
```

   - また、ユーザーは crwctl devfile:generate コマンドを使用して、たとえば NodeJS-MongoDB アプリケーションから devfile を生成できます。

     以下の例では、NodeJS コンポーネントが含まれる devfile を生成します。

```
$ 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 を使用して、devfile でインラインで利用できます。

- 言語のサポートを含めるには、**--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。

3.5. リモートでワークスペースにアクセス

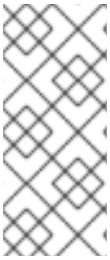
本セクションでは、ブラウザー外の **CodeReady Workspaces** ワークスペースにリモートでアクセスする方法を説明します。

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で変更されています。これに加えて、**CodeReady Workspaces** ワークスペースと対話する以下の方法があります。

- **OpenShift** コマンドラインツール **kubectl**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く
- **kubectl** 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. OpenShift コマンドラインツールを使用したワークスペースのリモートアクセス

OpenShift コマンドラインツール(**kubectl**)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。



注記

このセクションでは、**kubectl** ツールを使用してシェルを開き、**CodeReady Workspaces** ワークスペースでファイルを管理します。**oc OpenShift** コマンドラインツールを使用できます。

前提条件

- **kubectl** ツールが利用可能である。**OpenShift の Web サイトを参照して** ください。
- **oc version** コマンドを使用して **kubectl** のインストールを確認します。

```
$ oc version
Client Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0",
GitCommit:"e8462b5b5dc2584fdcd18e6bcfe9f1e4d970a529", GitTreeState:"clean",
BuildDate:"2019-06-19T16:40:16Z", GoVersion:"go1.12.5", Compiler:"gc",
Platform:"darwin/amd64"}
Server Version: version.Info{Major:"1", Minor:"15", GitVersion:"v1.15.0",
GitCommit:"e8462b5b5dc2584fdcd18e6bcfe9f1e4d970a529", GitTreeState:"clean",
BuildDate:"2019-06-19T16:32:14Z", GoVersion:"go1.12.5", Compiler:"gc",
Platform:"linux/amd64"}
```

バージョン 1.5.0 以降については、本セクションの手順に進みます。

手順

1. **exec** コマンドを使用してリモートシェルを開きます。
2. **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
che         workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4   4/4     Running   0
6m4s
```

上記の例では、**Pod** 名は **workspace7b2wemdf3hx7s3ln.maven-74885cf4d5-kf2q4** で、プロジェクトは **codeready** です。

1. コンテナーの名前を確認するには、以下のコマンドを実行します。

```
$ NAMESPACE=che
$ 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
```

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

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

3. コンテナーから、CodeReady Workspaces ワークスペースターミナルから、（CodeReady Workspaces ワークスペースターミナルから） build および run コマンドを実行します。

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

その他のリソース

- kubectl の詳細は、OpenShift ドキュメント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、kubectl ツールを使用して、Red Hat CodeReady Workspaces ワークスペースからファイルをリモートでダウンロードまたはアップロードする方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」を参照してください。

変更予定の CodeReady Workspaces ワークスペースへのリモートアクセス。手順はを参照してください「OpenShift コマンドラインツールを使用したワークスペースのリモートアクセス」。

- kubectl ツールが利用可能である。「 kubectlのインストール 」を参照してください。
- oc version コマンドを使用して kubectl のインストールを確認します。

手順

- 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使用します。

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

上記の手順を使用して、ユーザーはディレクトリーをダウンロードおよびアップロードすることもできます。

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、サンプルのコードベースが含まれています。これは、スタックの devfile で定義されます。本セクションでは、一連の手順で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. カスタムワークスペースビュー 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既存のワークスペースを実行し ます。

devfile の詳細は、「 Configuring a CodeReady Workspaces workspace using a devfile 」を参照してください。

3.6.1. ユーザーダッシュボードの取得開始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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタートを参照してください。

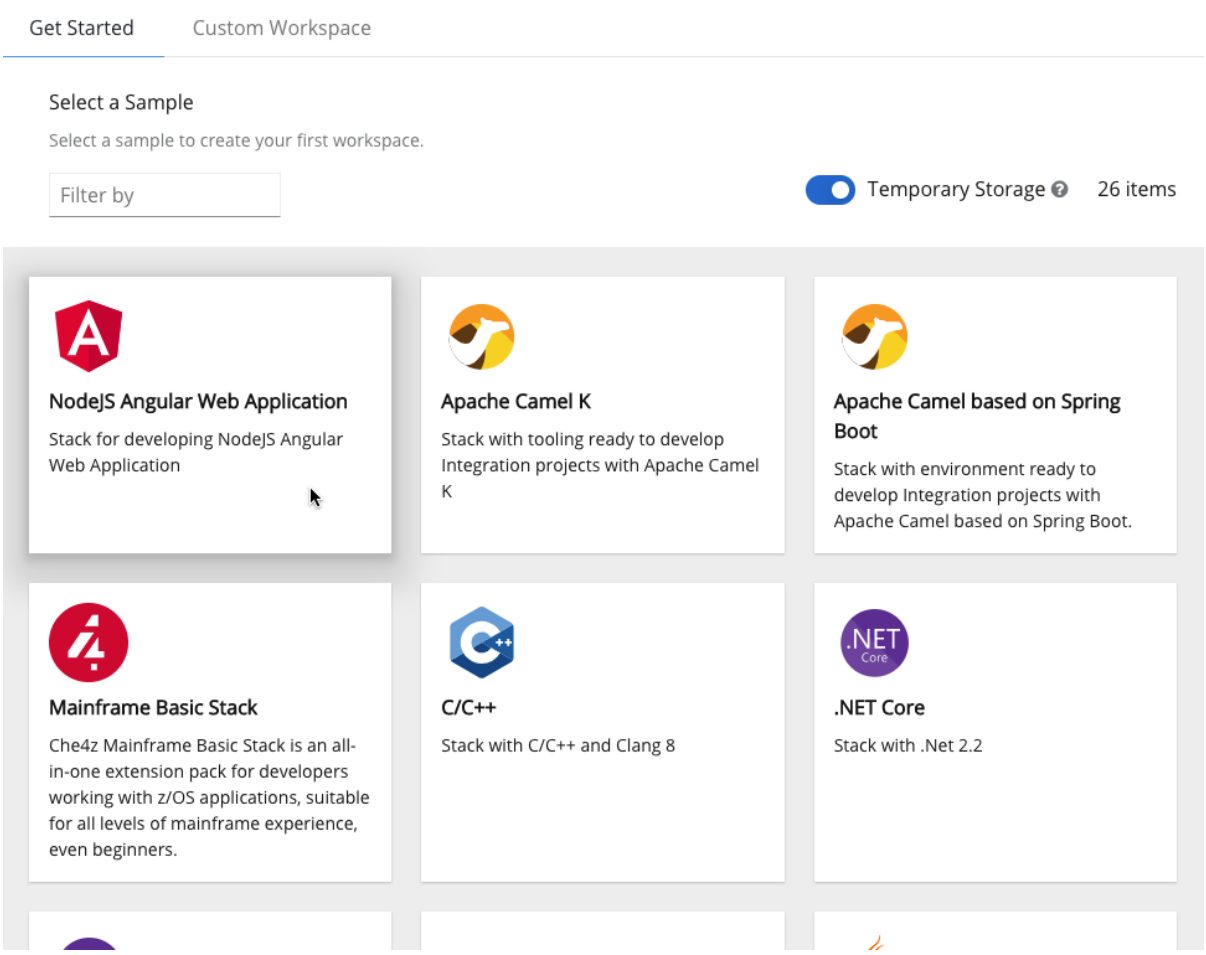
手順

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. スタートガイドタブ をクリックし ます。

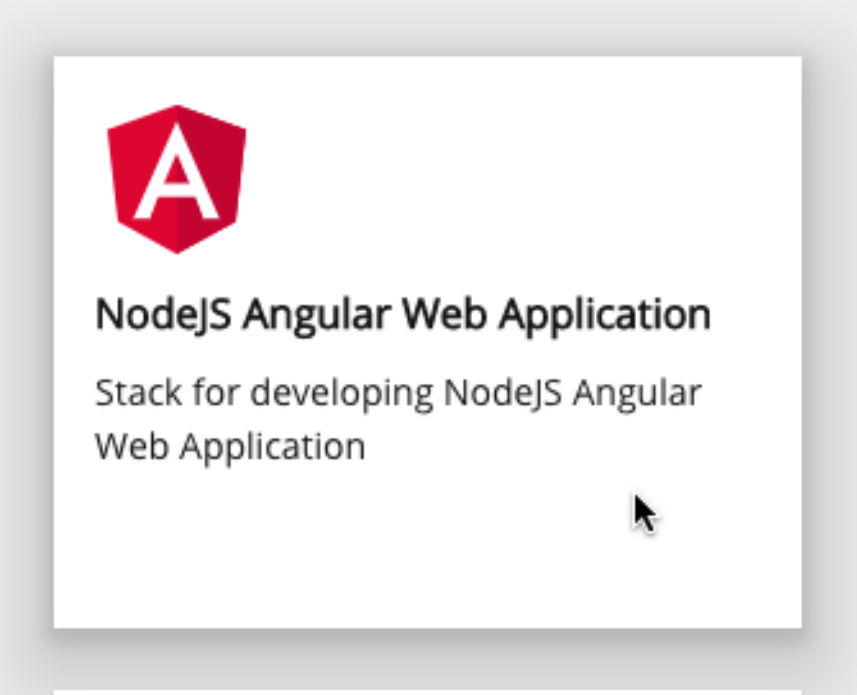
4. この場合、プロジェクトのビルドおよび実行に使用できるサンプルの一覧があります。



リソース制限の変更

メモリー要件の変更は、devfile からしか実行 できません。

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。**



新しいワークスペース名

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。生成された名前は、常に devfile metadata.generateName プロパティー を接頭辞と 4 つのランダムな文字として構成します。

3.6.2. ユーザーダッシュボードのカスタムワークスペース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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタート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。

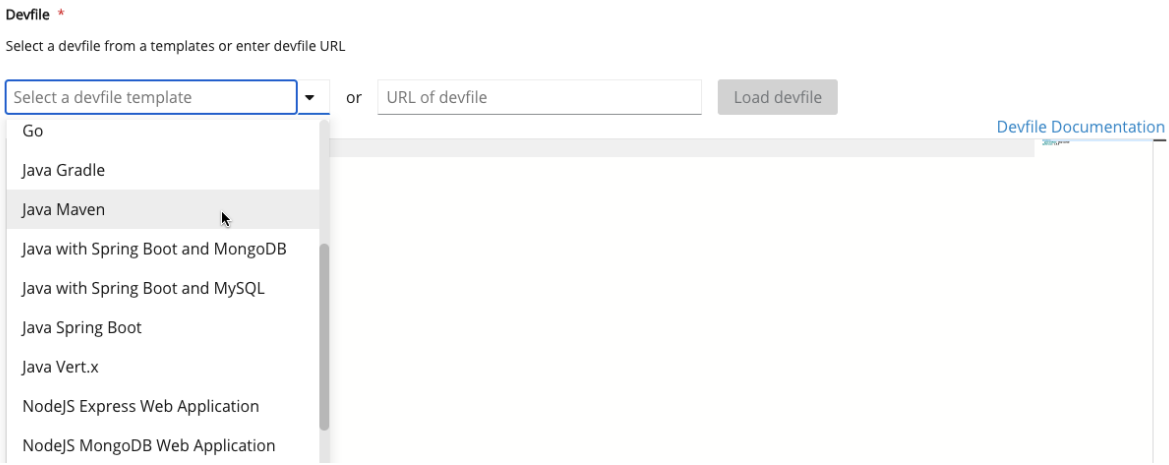
3. カスタムワークスペースタブを クリックし ます。

4. ワークスペースの 名前 を定義します。

新しいワークスペース名

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。生成された名前は、常に devfile metadata.generateName プロパティー を接頭辞と 4 つのランダムな文字として構成します。

5. Devfile セクションで、プロジェクトをビルドおよび実行するために使用される 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 からしか実行 できません。

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. 既存のワークスペースの設定変更

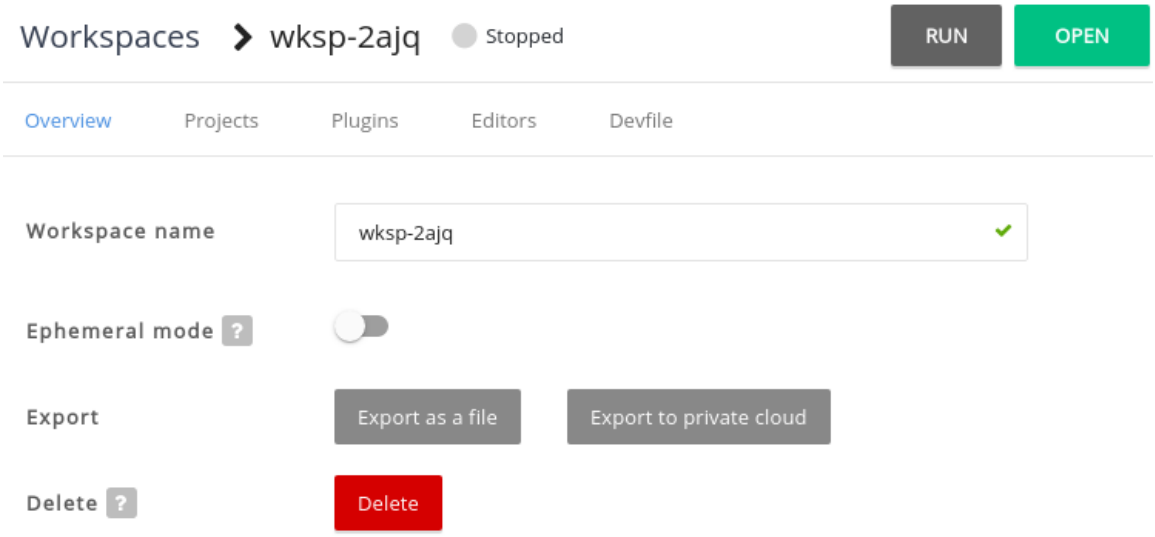
本セクションでは、既存のワークスペースの設定を変更する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」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が 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。

手順

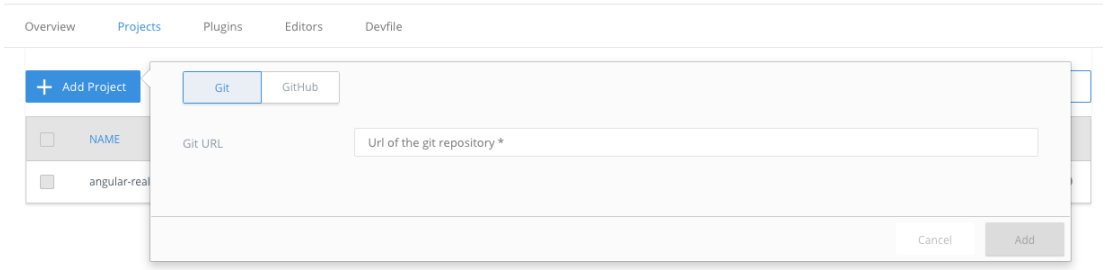
1. CodeReady Workspaces Dashboard に移動します。「ダッシュボードを使用した CodeReady Workspaces のナビゲーション」を 参照してください。
2. 左側のナビゲーションパネルで Workspaces に移動します。
3. ワークスペースの名前をクリックして、設定の概要ページに移動します。
4. Overview タブをクリックし、以下のアクションを実行します。
   - ワークスペース名 を変更します。
   - Storage Type を選択し ます。
   - ワークスペース設定をファイルまたはプライベートクラウドにエクスポート します。
   - ワークスペースを削除 します。



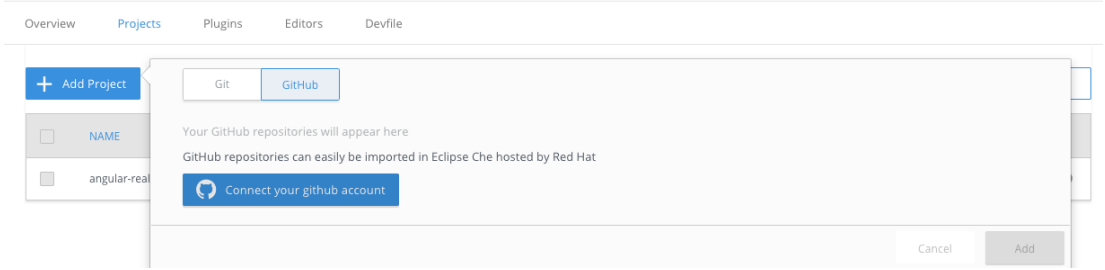
5. **Projects セクションで、ワークスペースに統合するプロジェクトを選択します。**

   a. **Add Project ボタンをクリックし、以下のいずれかを行います。**

      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 アカウントを接続し、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

まず汎用 Java ベースのスタックから始め、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 2.3 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 reference」を参照してください。

例： ADD コマンド

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. 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

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 ボタンを使用して、ユーザーダッシュボードから既存の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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」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が 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。

手順

1. CodeReady Workspaces Dashboard に移動します。「ダッシュボードを使用した 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. 実行中の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. ページの右上隅にある Run ボタンをクリックします。
5. ワークスペースが起動している。
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

本セクションでは 、Open ボタンを使用して、ユーザーダッシュボード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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」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が 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。

手順

1. CodeReady Workspaces Dashboard に移動します。「ダッシュボードを使用した 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. 実行中の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. ページの右上隅にある Open ボタンを クリックします。
5. ワークスペースが起動している。
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

本セクションでは、Recent Workspaces を使用して、ユーザーダッシュボード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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady

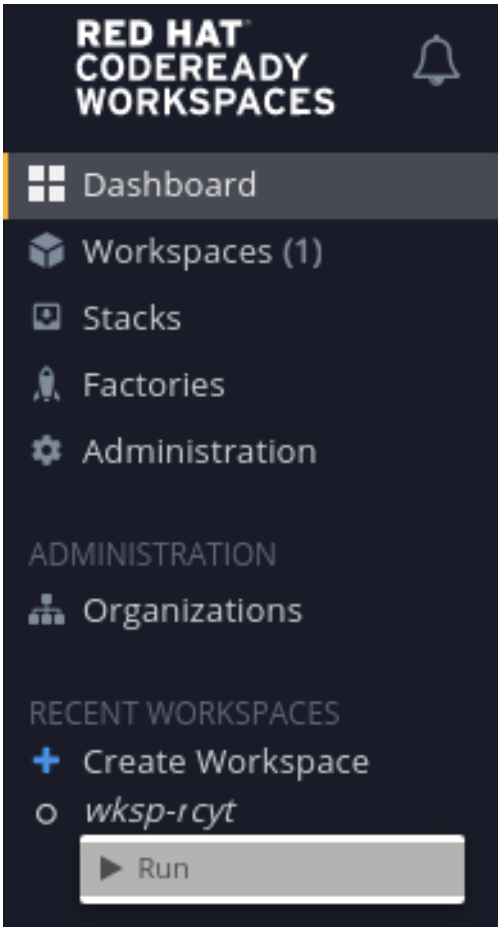
Workspaces on OpenShift Container Platform 」を参照してください。

- 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が 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。

手順

1. CodeReady Workspaces Dashboard に移動します。「 ダッシュボードを使用した 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 Recent Workspaces セクションで、稼働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の Run をクリックして起動します。



3.7. プロジェクトのソースコードをインポートによるワークスペースの作成

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しい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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」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。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。

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 を変更してプロジェクトが含まれるようにします。

- devfile を git リポジトリーに追加し、crwctl またはファクトリーを使用してワークスペースを起動します。

新しい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、ワークスペースを開始した後 に以下の 3 つの方法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

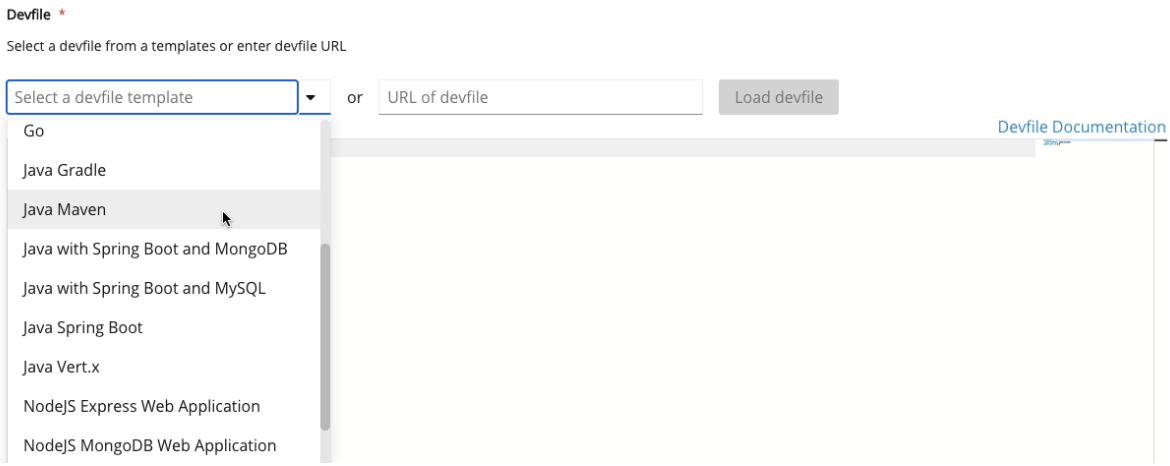
- 端末で git clone を使用して実行中のワークスペースにインポート

3.7.1. Dashboard からサンプルを選択し、devfile を変更してプロジェクトが含まれるようにします。

- 左側のナビゲーションパネルで Get Started に移動します。

- まだ選択されていない 場合 は、カスタムワークスペースタブをクリックします。

- Devfile セクションで、プロジェクトをビルドおよび実行するために使用される 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 reference 」を参照してください。

- **ワークスペースを開くには、作成および開く ボタンをクリックします。**

3.7.2. Dashboard から既存のワークスペースへのインポート

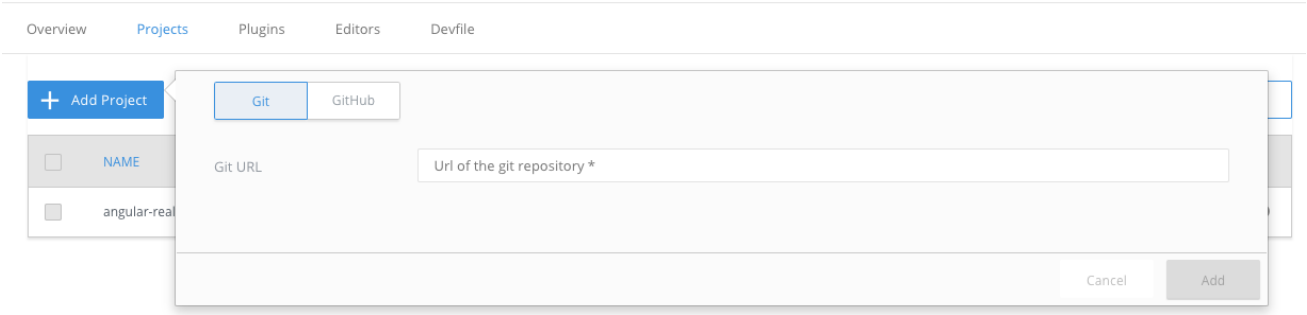
1. プロジェクトをインポートします。Dashboard を使用してプロジェクトをインポートする方法は 2 つ以上あります。

   - Dashboard から Workspaces を選択し、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 概要 タブにリンクされます。

   - または、歯車アイコンを使用します。これにより、独自の YAML 設定を入力できる Devfile タブにリンクされます。

2. Projects タブをクリックします。

3. プロジェクトの 追加をクリックし ます。その後、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

プロジェクトを稼働していないワークスペースに追加できますが、削除するために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取得したら、そのワークスペース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、デバッグ、または起動できます。

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。

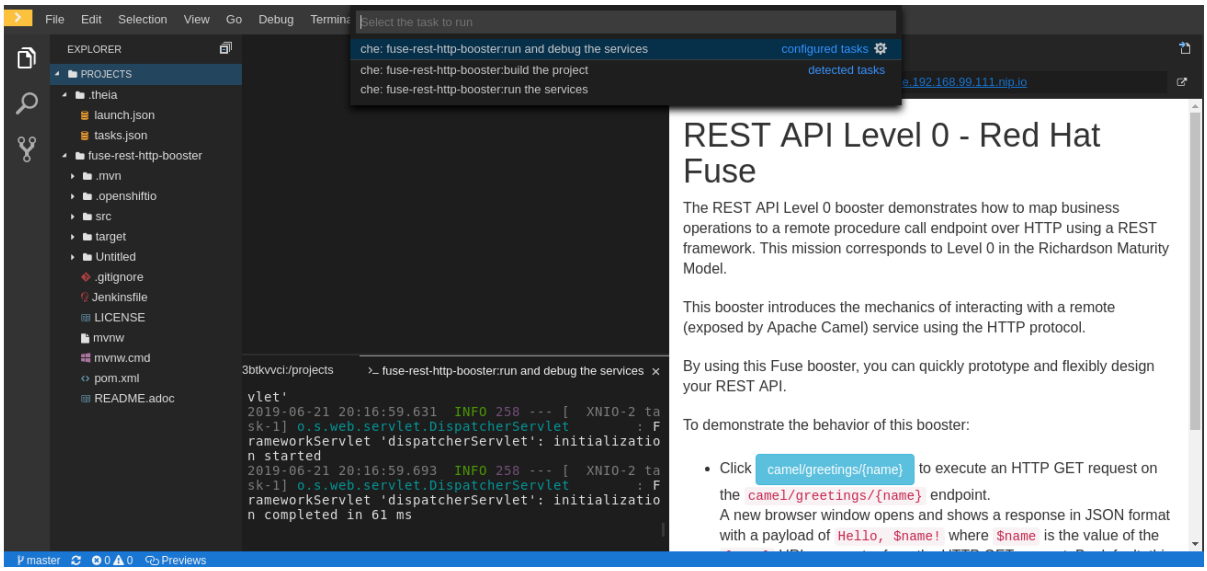
1. **Dashboard でワークスペース設定を開き、Devfile タブを選択します。**

Workspace

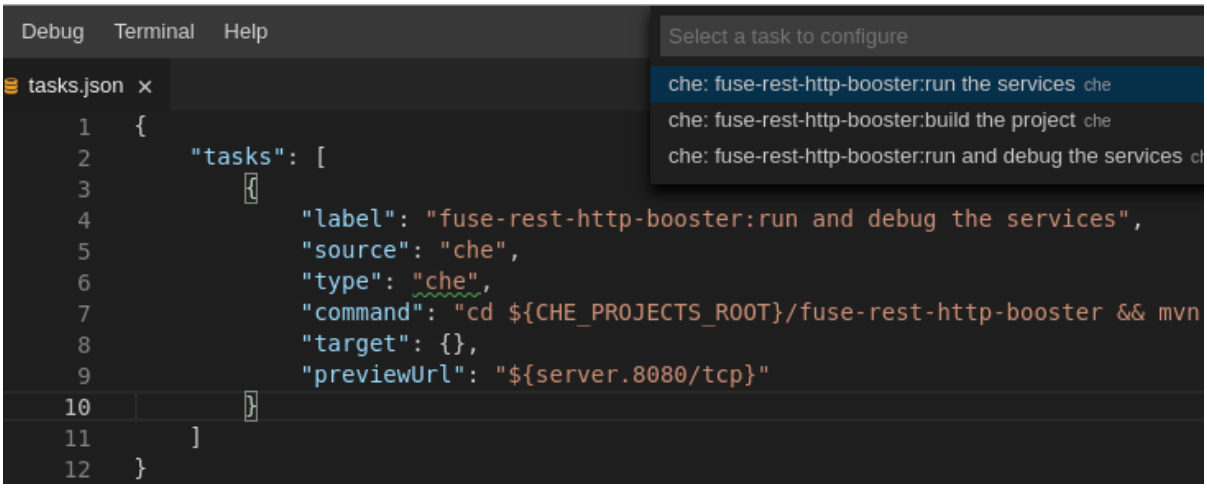
```
          -XX:AdaptiveSizePolicyWeight=90 -Dsun.zip.disableMemoryMapping=true
          -Xms20m -Djava.security.egd=file:/dev/./urandom
        name: JAVA_TOOL_OPTIONS
      - value: '$(echo ${0})\$'
        name: PS1
      - value: /home/user
        name: HOME
apiVersion: 1.0.0
commands:
  - name: 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。**



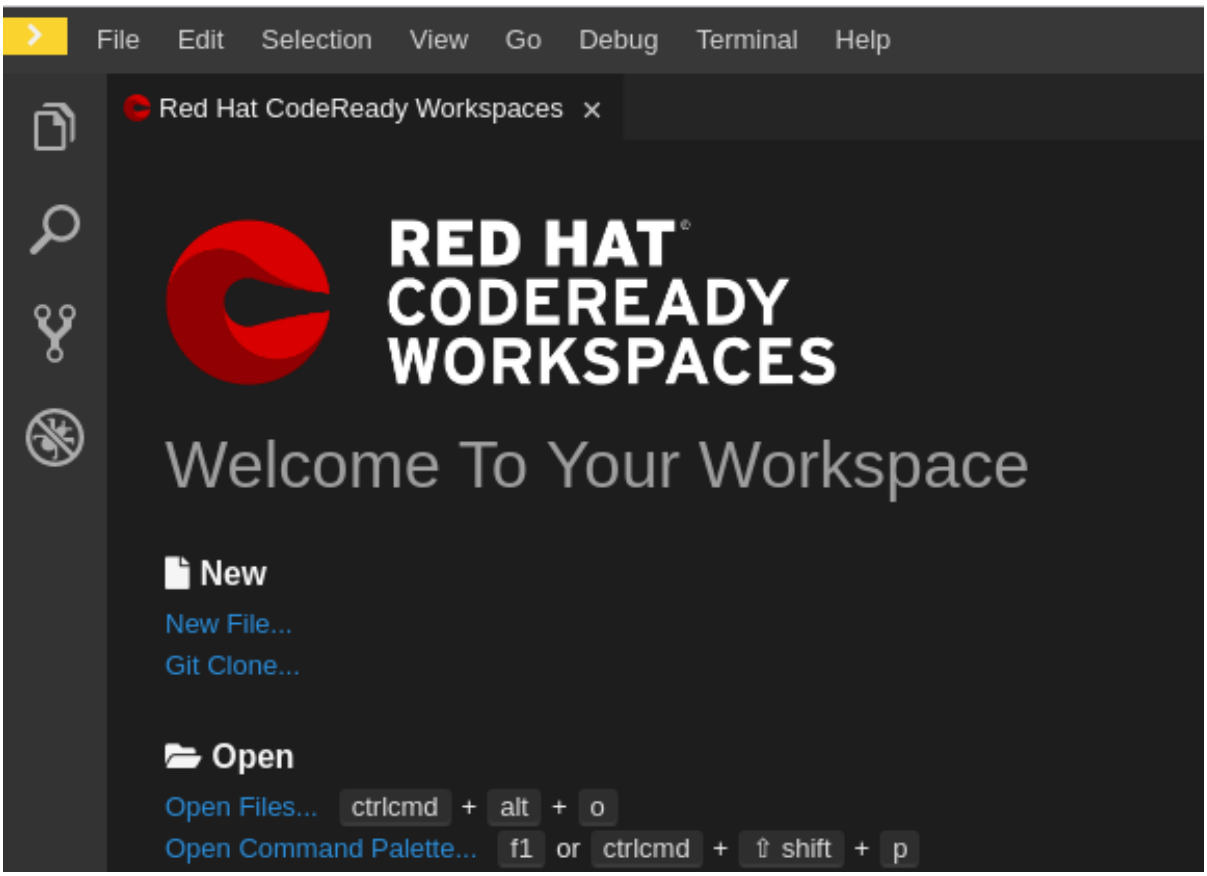
4. **コマンドを設定するには、メインメニューから Terminal > Configure Task を選択します。**



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

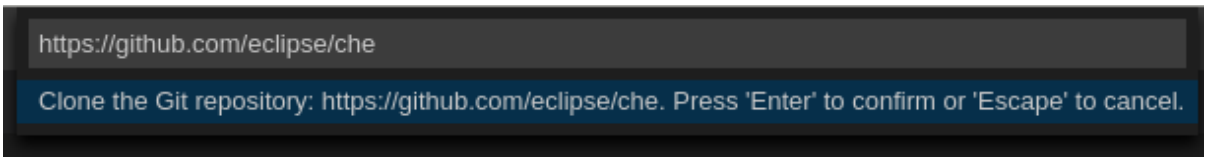
Git を使用して実行中のワークスペースにインポートするには、Clone コマンドを使用します。

1. ワークスペースを起動し、コマンドで Git: Clone コマンドを使用するか、Welcome 画面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。



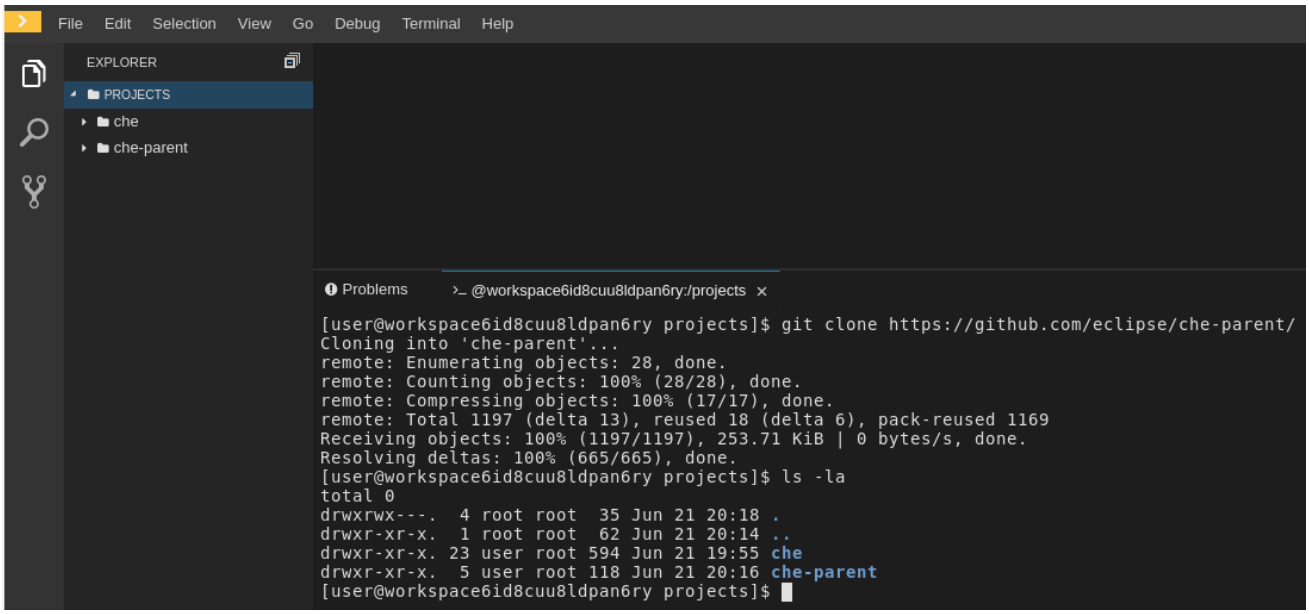
2. F1 または CTRL-SHIFT-P、または Welcome 画面でリンクからコマンドを開きます。

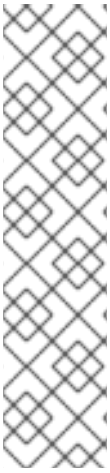
3. クローンを作成するプロジェクトのパスを入力します。



3.7.4. 端末に git clone で実行中のワークスペースにインポート

上記の方法に加えて、ワークスペースを起動し、ターミナル を開き、git clone と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のインポートまたは削除を実行してもワークスペース設定が更新されず、この変更は Dashboard の Project および Devfile タブには反映されません。

同様に、Dashboard を使用してプロジェクトを追加する場合は、rm -fr myproject でプロジェクトを削除すると、Projects または Devfile タブに依然として表示されます。

3.8. ワークスペースの公開ストラテジーの設定

CodeReady Workspaces サーバーのワークスペース公開ストラテジーを設定し、内部で実行されているアプリケーションが外部攻撃に対して脆弱にならないようにする方法を説明します。

ワークスペースの公開ストラテジーは、che.infra.kubernetes.server_strategy 設定プロパティーまたは CHE_ INFRA_KUBERNETES_ SERVER__STRATEGY 環境変数を使用して CodeReady Workspaces サーバーごとに設定されます。

che.infra.kubernetes.server_strategy でサポートされる値は以下のとおりです。

- multi-host

マルチホストストラテジーの場合は、che.infra.kubernetes.ingress.domain（または CHE _INFRA_KUBERNETES_INGRESS_DOMAIN 環境変数）設定プロパティーをワークスペースコンポーネントのサブドメインをホストするドメイン名に設定します。

3.8.1. ワークスペースの公開ストラテジー

ワークスペースの特定コンポーネントは、OpenShift クラスター外でアクセスできるようにする必要があります。これは通常、ワークスペースの IDE のユーザーインターフェースですが、開発中のアプリケーションの Web UI にすることもできます。これにより、開発者は開発プロセス中にアプリケーションと対話できます。

CodeReady Workspaces は、ユーザーがワークスペースコンポーネントを利用できるようにするための 3 つの方法（ ストラテジー とも呼ばれます）をサポートします。

- マルチホストストラテジー

ストラテジーは、ワークスペースのコンポーネント用に新規サブドメインを作成するかどうかと、これらのコンポーネントが利用可能であるホストを定義します。

3.8.1.1. マルチホストストラテジー

このストラテジーでは、各ワークスペースコンポーネントに CodeReady Workspaces サーバーに設定されたメインドメインの新しいサブドメインが割り当てられます。OpenShift ではこれが唯一のストラテジーであり、ワークスペースの公開ストラテジーの手動設定は常に無視されます。

コンポーネントへの URL に存在するパスはコンポーネントによって受信されるため、このストラテジーはコンポーネントのデプロイメントの観点から理解するのが最も簡単な方法です。

Transport Layer Security(TLS)プロトコルを使用してセキュア化された CodeReady Workspaces サーバーで、各ワークスペースの各コンポーネント用に新しいサブドメインを作成するには、CodeReady Workspaces デプロイメントのすべてのサブドメインでワイルドカード証明書が利用可能である必要があります。

3.8.2. セキュリティーに関する考慮事項

本セクションでは、異なる CodeReady Workspaces ワークスペースの公開ストラテジーを使用するセキュリティーへの影響について説明します。

本セクションのセキュリティー関連の考慮事項はすべて、マルチユーザーモードでの CodeReady Workspaces にのみ適用されます。シングルユーザーモードでは、セキュリティー制限は行われません。

3.8.2.1. JSON Web token(JWT)プロキシー

CodeReady Workspaces プラグイン、エディター、およびコンポーネントは、すべての CodeReady Workspaces プラグイン、エディター、およびコンポーネントでユーザーにアクセスするための認証を必要とすることができます。この認証は、設定に基づいて対応するコンポーネントのリバースプロキシーとして機能する JSON Web トークン(JWT)プロキシーを使用して実行され、コンポーネントの代わりに認証を実行します。

認証は、CodeReady Workspaces サーバーの特別なページへのリダイレクトを使用して、ワークスペースおよびユーザー固有の認証トークン（ワークスペースアクセストークン）を元の要求されたページに戻します。

JWT プロキシーは、受信リクエストの以下の場所からのワークスペースアクセストークンを以下の順序で受け入れます。

1. トークンクエリーパラメーター
2. bearer-token 形式の Authorization ヘッダー

3. access_token クッキー

3.8.2.2. セキュリティー保護されたプラグインおよびエディター

CodeReady Workspaces ユーザーは、ワークスペースプラグインおよびワークスペースエディター（Che-Theia など）のセキュリティーを保護する必要はありません。これは、JWT プロキシー認証がユーザーに透過的であり、その meta.yaml 記述子のプラグインまたはエディターの定義によって管理されるためです。

3.8.2.3. セキュリティー保護されたコンテナーイメージコンポーネント

コンテナーのイメージコンポーネントは、必要に応じて、devfile の作成者が CodeReady Workspaces が提供する認証を必要とするカスタムエンドポイントを定義できます。この認証は、エンドポイントの 2 つのオプション属性を使用して設定されます。

- secure: CodeReady Workspaces サーバーに JWT プロキシーをエンドポイントの前に配置するよう指示するブール値属性。このようなエンドポイントは、で説明されている複数の方法のいずれかでワークスペースアクセストークンで提供する必要があり 「JSON Web token(JWT)プロキシー」 ます。属性のデフォルト値は false です。

- cookieAuthEnabled- で説明されているように、CodeReady Workspaces サーバーに、現在のユーザー認証の認証されていないリクエストを自動的にリダイレクトするよう CodeReady Workspaces サーバーに指示するブール値属性 「JSON Web token(JWT)プロキシー」。この属性を true に設定すると、クロスサイトリクエスト偽装(CSRF)攻撃が可能になるため、セキュリティーの結果が得られます。属性のデフォルト値は false です。

3.8.2.4. クロスサイト要求偽装攻撃

クッキーベースの認証は、JWT プロキシーによりアプリケーションをセキュアにすることができます。これは、CSRF(Cross-site Request forgery)攻撃に対処します。アプリケーションが脆弱ではないようにするに は、クロスサイト要求 forgery— ページおよびその他のリソースを参照してください。

3.8.2.5. フィッシング攻撃

JWT プロキシーの背後にあるサービスとホストを共有するワークスペースを使用してクラスター内に Ingress またはルートを作成できる攻撃者は、サービスおよび特別な偽の Ingress オブジェクトを作成できる可能性があります。このようなサービスまたは Ingress が、以前にワークスペースで認証された正当なユーザーによってアクセスされると、正当ユーザーのブラウザーによって偽装 URL に送信されるクッキーからワークスペースアクセストークンを盗む可能性があります。この攻撃ベクトルを省略するには、Ingress のホストの設定を拒否するように OpenShift を設定します。

3.9. 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

シークレットとは、ユーザー名、パスワード、認証トークン、および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。

ユーザーはワークスペースコンテナーに機密データが含まれるシークレットをマウントできます。これにより、新規作成されたすべてのワークスペースにシークレットから保存されたデータが自動的に適用されます。そのため、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 設定の settings.xml ファイル
- SSH 鍵のペア
- AWS 認証トークン
- Git クレデンシャルストアファイル

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 機能を持つ新しいワークスペースすべてに適用される Maven 設定が自動的にマウントされます。
- 環境変数： 自動認証に SSH キーペアと 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 設定の設定を目的としています。

マウントプロセスは標準の OpenShift マウントメカニズムを使用しますが、必要な CodeReady

Workspaces ワークスペースコンテナーでシークレットを適切にバインドするために追加のアノテーションおよびラベル付けが必要になります。

3.9.1. 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

警告

ファイルとしてマウントされたシークレットが v1.13 より古い OpenShift では、devfile で定義されたボリュームマウントが上書きされます。

本セクションでは、CodeReady Workspaces のシングル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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」を参照してください。

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、新しい OpenShift シークレットを作成します。

   - 作成するシークレットのラベルは、CodeReady Workspaces の che.workspace.provision.secret.labels プロパティーで設定された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とおりです。

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

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

注記

以下の例は、Red Hat CodeReady Workspaces バージョン 2.1 および 2.2 での target-container アノテーションの使用における違いを説明します。

たとえば、以下のようになります。

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

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、マウントパスを提供することを示し、必要に応じて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する必要があります。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: /home/user/.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、ブール値で automount-workspace-secret アノテーションが導入されました。この目的は、devfile でオーバーライドする機能を使用して、デフォルトのシークレットマウント動作を定義することです。true を指定すると、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。一方、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: /home/user/.m2/
    che.eclipse.org/mount-as: file
  labels:
...
```

Kubernetes シークレットのデータには、コンテナーにマウントされる必要なファイル名に一致する複数の項目が含まれる場合があります。

```
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: /home/user/.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/user/.m2/ パスにマウントされます。

secret-s マウントパスは、devfile を使用してワークスペースの特定コンポーネントで上書きできます。マウントパスを変更するには、追加のボリュームを devfile のコンポーネントで宣言し、名前がオーバーライドされたシークレット名と必要なマウントパスで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
   ...
```

このようなオーバーライドでは、コンポーネントはエイリアスを宣言して、それらのコンテナーに属するコンテナーを区別し、それらのコンテナー専用にオーバーライドパスを適用することができることに注意してください。

3.9.2. 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

以下のセクションでは、ユーザーのプロジェクトから環境変数または変数として 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 OpenShift シークレットをマウントする方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「 Installing CodeReady Workspaces on OpenShift Container Platform 」を参照してください。

手順

1. CodeReady Workspaces ワークスペースが作成される k8s プロジェクトで新しい OpenShift シークレットを作成します。

   - 作成するシークレットのラベルは、CodeReady Workspaces の che.workspace.provision.secret.labels プロパティーで設定された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です。

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

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

   注記

   以下の例は、Red Hat CodeReady Workspaces バージョン 2.1 および 2.2 での target-container アノテーションの使用における違いを説明します。

   たとえば、以下のようになります。

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

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、変数名を提供することを示し、任意でこのマウントが適用されるコンテナー名を指定する必要があります。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、ブール値で automount-workspace-secret アノテーションが導入されました。この目的は、devfile でオーバーライドする機能を使用して、デフォルトのシークレットマウント動作を定義することです。true を指定すると、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。一方、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と、すべてのワークスペースコンテナーに myvalue が プロビジョニングされる値が作成されます。

シークレットが複数のデータアイテムを提供する場合は、以下のように各データキーに環境変数名を指定する必要があります。

```
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 、および値 myvalue およびその他の値を持つ 2 つの環境変数が 、 すべてのワークプシーコンテナーにプロビジョニングされます。

注記

Kubernetes シークレットのアノテーション名の最大長は 63 文字です。ここでは、9 文字は / で終わる接頭辞に予約されます。これは、シークレットに使用できるキーの最大長の制限として機能します。

3.9.3. git credetials ストア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 credetials ストアをユーザーのプロジェクトから、CodeReady Workspaces のシングル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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」を参照してください。

手順

1. https://git-scm.com/docs/git-credential-store#_storage_format 形式の git 認証情報ファイルを準備 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、新しい OpenShift シークレットを作成します。

   - 作成するシークレットのラベルは、CodeReady Workspaces の che.workspace.provision.secret.labels プロパティーで設定された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とおりです。

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

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

3.9.4. 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

OpenShift のアノテーションおよびラベルは、ライブラリー、ツール、およびその他のクライアントによって使用されるツールで、任意の識別されていないメタデータを OpenShift ネイティブオブジェクトに割り当てます。

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。ここでは、アノテーションは OpenShift オブジェクトによって内部的に使用されない非識別情報に使用されます。

このセクションでは、CodeReady Workspaces ワークスペースの OpenShift シークレットマウントプロセスで使用される OpenShift アノテーション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識別するのに役立つ項目が含まれている必要があります。これらの項目は以下のとおりです。

- che.eclipse.org/target-container: バージョン 2.1 を確認し ます。マウントコンテナーの名前。名前が定義されていない場合は、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シークレットがマウントされます。

- che.eclipse.org/automount-workspace-secret: バージョン 2.2 で導入された。メインの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、ワークスペース所有者の柔軟性を向上させる automountWorkspaceSecrets ブール 値を使用して devfile コンポーネントで上書きできます。このプロパティーには、それを使用するコンポーネントの エイリアス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: この項目では、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説明します。オプション： env または file

- che.eclipse.org/ <mykeyName>-env-name: FOO_ENV: データに複数の項目が含まれる場合に使用される環境変数の名前。mykeyName はサンプルとして 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、拡張およびカスタマイズ可能な開発者ワークスペースプラットフォームです。

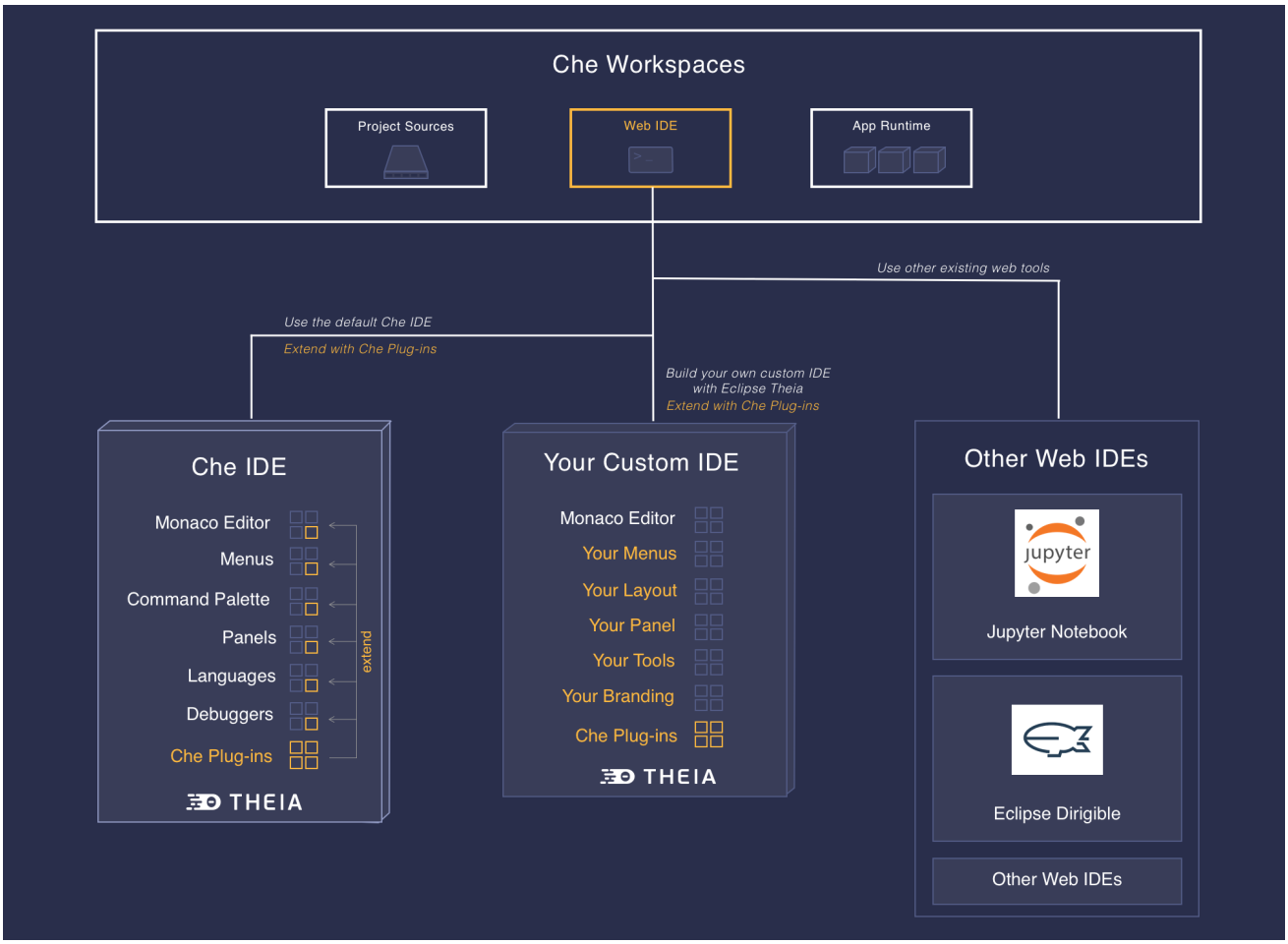
Red Hat CodeReady Workspaces は、以下の 3 つの方法で拡張できます。

- 代替の IDE は、Red Hat CodeReady Workspaces の特殊なツールを提供します。たとえば、データ分析用の Jupyter ノートブックです。代替 IDE は、Eclipse Theia またはその他の Web IDE をベースにすることができます。Red Hat CodeReady Workspaces のデフォルト IDE は Che-Theia です。

- Che-Theia プラグインは、Che- Theia IDE に機能を追加します。これは、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。これは、独自の依存関係を提供するために、ファイルまたはコンテナーとしてパッケージ化できます。

- スタック は、異なる開発者の人格に対応する専用のツールセットを備えた事前設定された CodeReady Workspaces ワークスペースです。たとえば、テスト用に必要なツールのみを持つテスト用ワークベンチを事前設定することが可能です。

図4.1 CodeReady Workspaces の拡張性



ユーザーは 、 デフォルトで CodeReady Workspaces が提供するセルフホストモードを使用して CodeReady Workspaces を拡張することができます。

- Che-Theia プラグインとは
- CodeReady Workspaces での代替 IDE の使用
- CodeReady Workspaces での Visual Studio コードエクステンションの使用

4.1. CHE-THEIA プラグインとは

Che-Theia プラグインは、IDE から分離され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し、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 に依存することで、サポートされる言語 を拡張します。
- デバッガー： Debug Adapter Protocol でデバッグ機能を拡張します。
- 開発ツール： お気に入りの Linter と、テストおよびパフォーマンスツールとして統合。
- メニュー、パネル、およびコマンド： IDE コンポーネントに独自の項目を追加します。
- themes: カスタムのテーマを作成し、UI を拡張するか、またはアイコンのテーマをカスタマイズします。
- スニペット、フォーマッター、および構文の強調表示： サポートされるプログラミング言語での使用の強化。
- KeyBindings: 環境が不安定であるように、新しいキーマップと一般的なキーバインディングを追加します。

4.1.1. Che-Theia プラグインの機能と利点

Features	description	利点
高速ロード	プラグインは実行時にロードされ、すでにコンパイルされています。IDE がプラグインコードを読み込んでいる。	コンパイルの時間を回避します。インストール後の手順は回避します。
セキュアなローディング	プラグインは IDE とは別にロードされます。IDE は、常に使用可能な状態のままになります。	バグがある場合には、プラグインは IDE 全体を破損しません。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でパッケージ化されます。	ツールのインストールなし。コンテナー内で実行されている依存関係。
コード分離	ファイルを開くか、入力など、IDE の主な機能をプラグインがブロックでき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ています。依存関係の不一致を回避します。

Features	description	利点
vs コード拡張の互換性	既存の VS Code Extensions で IDE の機能を拡張します。	ターゲットの複数のプラットフォーム。必要なインストールで、Visual Studio Code Extension 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 Che-Theia のデフォルト Web IDE 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、プレーン Eclipse Theia とは若干異なるバージョンです。このバージョンの Eclipse Theia for CodeReady Workspaces は Che-Theia と呼ばれています。

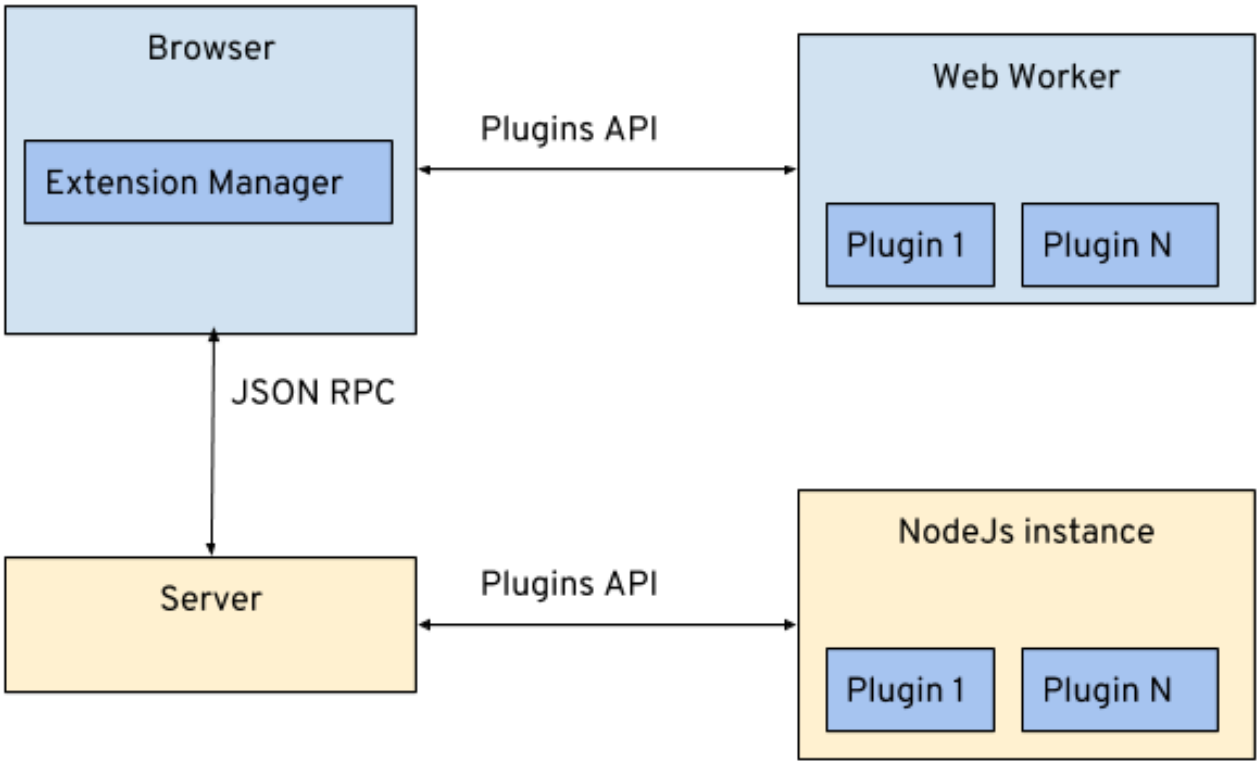
Che-Theia プラグインを構築すると、Red Hat CodeReady Workspaces で 提供される IDE を拡張することができます。Che-Theia プラグインは、他の Eclipse Theia ベースの IDE と互換性があります。

4.1.2.1. クライアント側およびサーバー側の Che-Theia プラグイン

Che-Theia エディタープラグインでは、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。エディターが読み込みを完了すると、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 エディターが機能し続けます。

che-Theia プラグインは、クライアント側またはサーバー側で実行します。以下は、クライアントおよびサーバー側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 は、クライアント側（Web ワーカー）またはサーバー側(Node.js)上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 でツールの分離と容易な拡張性を提供するために、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 extension API と互換性があります。通常、Che-Theia は VS Code 拡張を独自のプラグインとして実行できます。

CodeReady Workspaces ワークスペース（コンテナー、設定、ファクトリー）のコンポーネントに依存するプラグインを開発する場合、Che-Theia に組み込まれた CodeReady Workspaces API を使用します。

4.1.2.3. che-Theia プラグインの機能

che-Theia プラグインには以下の機能があります。

プラグイン	description	リポジトリー

プラグイン	description	リポジトリー
CodeReady Workspaces 拡張タスク	CodeReady Workspaces コマンドを処理し、それらをワークスペースの特定コンテナーを起動する機能を提供します。	
CodeReady Workspaces 拡張ターミナル	ワークスペースのコンテナーにターミナルを提供できます。	
CodeReady Workspaces Factory	Red Hat CodeReady Workspaces ファクトリーを処理します。	
CodeReady Workspaces Container	ワークスペース内で実行されているすべてのコンテナーを表示し、コンテナーと対話できるようにするコンテナービューを提供します。	コンテナーのプラグイン
Dashboard	IDE と **Dashboard** を統合し、ナビゲーションを容易にします。	
CodeReady Workspaces APIs	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）と対話できるようにします。	

4.1.2.4. vs コードエクステンションおよび Eclipse Theia プラグイン

Che-Theia プラグインは、VS Code 拡張または Eclipse Theia プラグインに基づいています。

Visual Studio Code 拡張

VS コード拡張を、独自の依存関係セットで 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、Red Hat CodeReady Workspaces ユーザーはエクステンション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。CodeReady Workspaces の「 Using a Visual Studio Code extension 」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。

その他のリソース

- 「Embedded および remote 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 プラグインのメタデータは、プラグインレジストリーの個々のプラグインに関する情報です。

Che-Theia プラグインのメタデータは、各プラグインの meta.yaml ファイルで定義されます。

以下は、プラグインメタ YAML ファイルで利用できるすべてのフィールドの概要です。本書では、Plugin meta YAML 構造バージョン 3 を示しています。

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（バージョンは後方互換性のためにサポート対象）
category	Available: category 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
非推奨	オプション。他のプラグインを非推奨にするセクション * autoMigrate - boolean * migrateTo - new 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、これはプラグインレジストリーの dockerimage 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、これはプラグインレジストリーの dockerimage 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（スペースは許可されていません）[-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。[-a-z0-9] と一致する必要があります。
リポジトリー	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 拡張**
バージョン	バージョン情報（例： 7.5.1、[-.a-z0-9]）
spec	仕様（下記参照）

表4.2 spec 属性

endpoints	オプション; プラグインエンドポイント。エンドポイントの説明を参照してください。
コンテナー	オプション： プラグインのサイドカーコンテナー。Che Plugin および VS コードエクステンションがサポートするのは 1 つのコンテナーだけです。
initContainers	オプション。プラグイン用のサイドカー 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 ファイルなどの、プラグインアーティファクトへの URL 一覧で VS コードおよび Che-Theia プラグインに必要な属性

表4.3 spec.containers。注記： spec.initContainers には、全く同じコンテナー定義があります。

name	サイドカーコンテナー名
image	絶対または相対的なコンテナーイメージの URL

memoryLimit	OpenShift メモリー制限文字列（例： **512Mi**）
memoryRequest	OpenShift メモリー要求文字列（例： **512Mi**）
cpuLimit	OpenShift CPU 制限文字列（例： **2500m**）
cpuRequest	OpenShift CPU 要求文字列（例： **125m**）
env	サイドカーコンテナーに設定する環境変数の一覧
command	コンテナーにおける root process コマンドの文字列配列定義
args	コンテナーの root process コマンドの文字列配列引数
ボリューム	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** で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。注記： workspaceEnv に は全く同じ属性があります。

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なものになります。そうでない場合は、ボリュームは永続化されます。

表4.6 spec.containers.ports and spec.initContainers.ports attributes

exposedPort	公開ポート

表4.7 spec.containers.commands and spec.initContainers.commands attributes

name	コマンド名
workingDir	コマンド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 attributes

name	名前（スペースは許可されていません）[-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 attributes

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** でのみリッスンし、JWT プロキシーを使用して公開されます。
cookiesAuthEnabled	**true**、**false**

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 ハンドラー 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、コマンドによって消費されるリソースが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 ハンドラー 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、コマンドによって消費されるリソースが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 Service

```
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 コードエクステンションの meta.yaml 例： AsciiDoc サポートエクステンション

```
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

ユーザーがワークスペースを起動すると、以下の手順に従います。

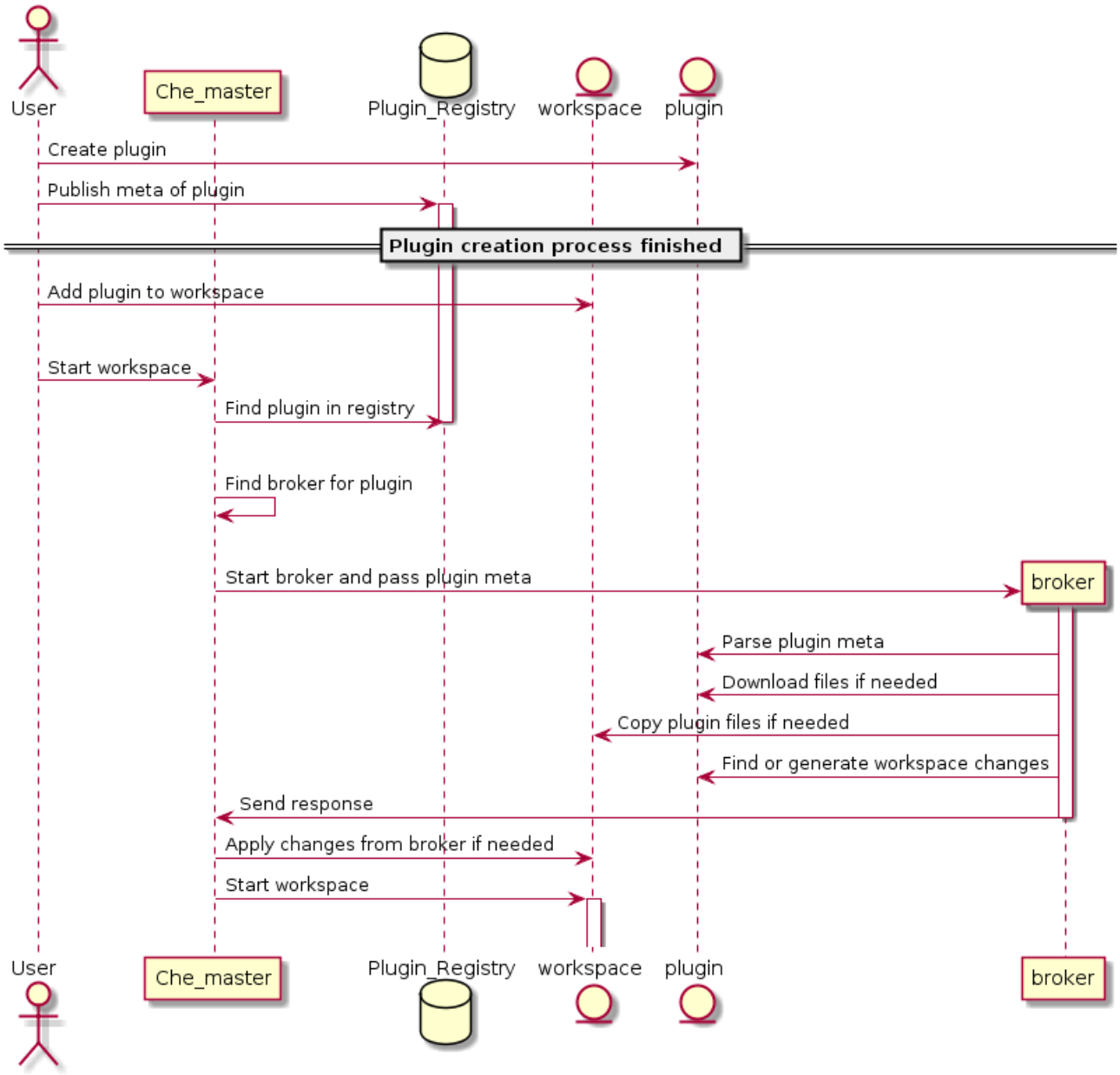
1. CodeReady Workspaces マスターは、ワークスペース定義から起動するプラグインをチェックします。
2. プラグインメタデータが取得され、各プラグインのタイプが認識されます。
3. Broker はプラグインタイプに従って選択されます。
4. ブローカーはプラグインのインストールおよびデプロイメントを処理します（インストー

ルプロセスはブローカーごとに異なります）。

注記

さまざまな種類のプラグインが存在する。ブローカーは、プラグインが正常にデプロイされるまでのインストール要件をすべて満たします。

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 マスターはワークスペースの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（ .theia ファイルから）プラグインを抽出し、プラグインに必要なサイドカーコンテナーを取得します。

2. ブローカーは適切なコンテナー情報を CodeReady Workspaces マスターに送信します。

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、Che-Theia エディターコンテナーで使用できるようにします。

4. 次に、CodeReady Workspaces ワークスペースマスターが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 は独自のコンテナーで起動され、プラグインを読み込むために正しい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：

1. ユーザーが Che-Theia でブラウザータブまたはウィンドウを開くと、Che-Theia は新しいプラグインセッションを開始します（フロントエンドの場合は Web ワーカー、バックエンドの場合は Node.js）。Che-Theia プラグインごとに、新しいセッションが開始されていることが通知されます（プラグインの start（） 関数はトリガーされます）。

2. 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ています。

3. ユーザーがブラウザータブを閉じるかタイムアウトがある場合、すべてのプラグインが通知されます（トリガーされたプラグインの stop（） 関数）。

4.1.5. Embedded および remote 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。

che-Theia プラグインは、必要な依存関係（ 組み込み （またはローカル）と リモート ）に基づいて、2 つの方法で実行できます。

4.1.5.1. Embedded（またはローカル）プラグイン

プラグインには特定の依存関係がなく、Node.js ランタイムのみを使用し、IDE と同じコンテナーで実行されます。プラグインが IDE に挿入されます。

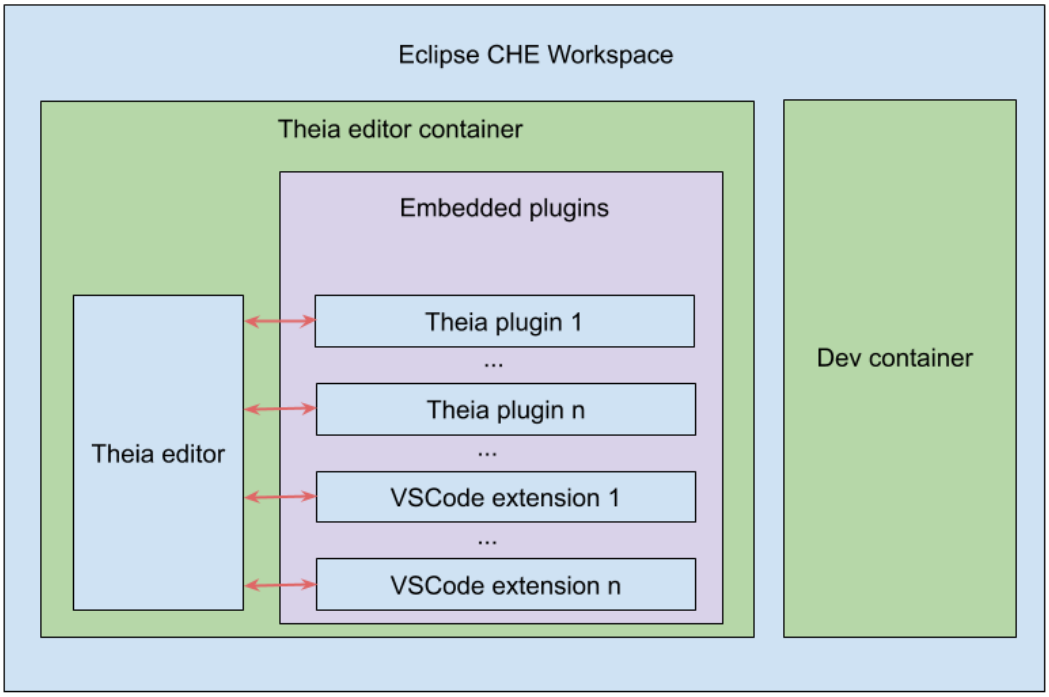
例：

- コードの繰り返し
- 新しいコマンドセット
- 新規 UI コンポーネント

Che-Theia プラグインを組み込みとして追加するには、meta.yaml ファイルにプラグインバイナリーファイル（ .theia アーカイブ） への URL を定義します。VS Code エクステンションの場合は、Visual Studio Code 市場からのエクステンション ID を提供します（CodeReady Workspaces の「Using a Visual Studio Code extension in CodeReady Workspaces」を参照してください）。

ワークスペースの起動時に、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、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。

図4.4 ローカル Che-Theia プラグイン



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、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、すべての依存関係が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 コード拡張バイナリーmeta.yaml ファイルの定義は、組み込みプラグインと同じです。
- コンテナーイメージの定義（例： eclipse/che-theia-dev:yely）。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。

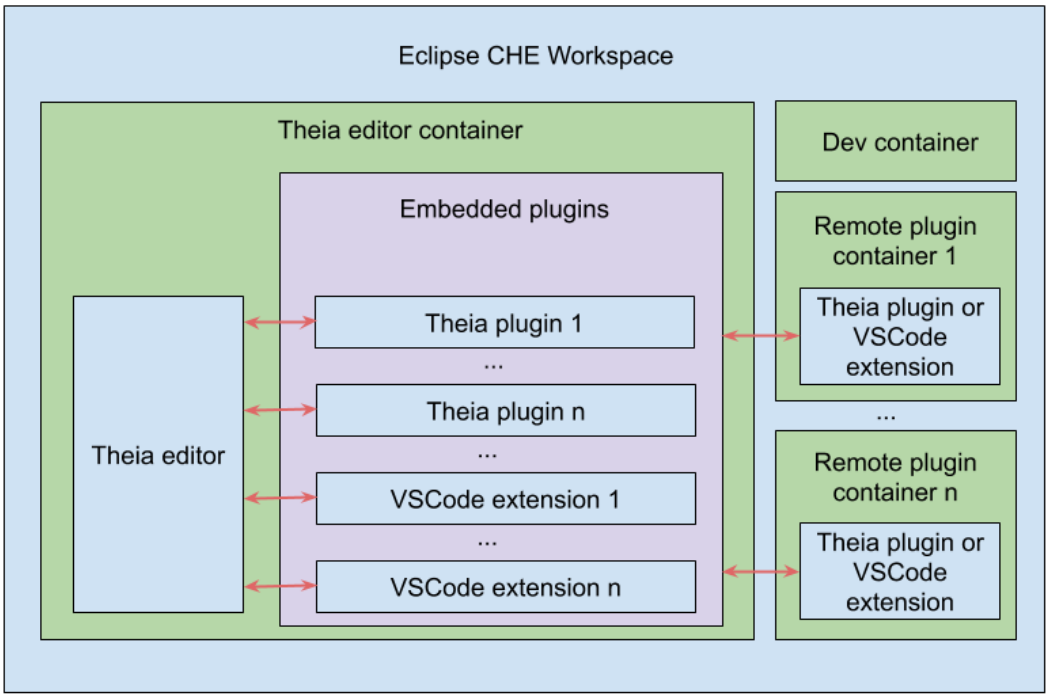
例：

- Java Language Server

- Python Language Server

ワークスペースの起動時に、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および展開し、作成し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。

図4.5 Remote Che-Theia プラグイン



4.1.5.3. 比較マトリックス

Che-Theia プラグイン（または VS コード拡張）にコンテナー内で追加の依存関係が必要ない場合は、埋め込みプラグインになります。プラグインを含む追加の依存関係を持つコンテナーはリモートプラグインです。

表4.11 che-Theia プラグイン比較マトリックス： Embedded vs remote

	プラグインごとの RAM の設定	環境の依存関係	分離されたコンテナーを作成する
remote	TRUE	プラグインは、リモートコンテナーに定義された依存関係を使用します。	TRUE
embedded	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 RAM を設定できません）	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ていない場合、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

ユースケースとプラグインが提供する機能に応じて、記述された実行モードの 1 つを選択します。

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、別個のコンテナーで VS Code Extensions 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エンドポイント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 meta.yaml ファイルで定義されたリモート拡張とプラグインを開始し、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にプラグイン API プロキシーを作成します。リモートプラグインエンドポイントは、一部のプラグイン API の部分のインターセプターで、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： 端末 API、デバッグ API

リモートプラグインエンドポイント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 では、サイドカーイメージでリモートプラグインエンドポイントを起動する方法が 2 つあります。

- Dockerfile を使用した起動リモートプラグインエンドポイントの定義。この方法を使用するには、元のイメージをパッチを適用して再ビルドします。
- プラグインの meta.yaml ファイルで起動リモートプラグインエンドポイントを定義する。

この方法を使用して、元のイメージのパッチ適用を回避します。

4.1.6.1. Dockerfile を使用した起動リモートプラグイン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 変数を使用します。

手順

- Dockerfile の CMD コマンドを使用して、リモートプラグイン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/user

ENV HOME=/home/user

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

CMD ${PLUGIN_REMOTE_ENDPOINT_EXECUTABLE}
```

- Dockerfile の ENTRYPOINT コマンドを使用して、リモートプラグイン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/user
```

```
ENV HOME=/home/user

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

イメージによっては、ラッパースクリプトを使用してパーミッションを設定します。スクリプトは、コンテナー内のパーミッションを設定する Dockerfile の ENTRYPOINT コマンドで定義され、Dockerfile の CMD コマンドで定義されたメインプロセスを実行します。

Red Hat CodeReady Workspaces は、このようなイメージをラッパースクリプトで使用し、OpenShift などの高度なセキュリティーで異なるインフラストラクチャーでパーミッション設定を提供します。

- 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}
```

この例では、コンテナーは **Dockerfile** の **ENTRYPOINT** コマンドで定義された **/entrypoint.sh** スクリプトを起動します。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}** を実行し ます。リモートプラグインエンドポイントは、パーミッション設定後にコンテナーで起動します。

4.1.6.2. meta.yaml ファイルで起動しているリモートプラグインエンドポイントの定義

この方法を使用して、イメージを再利用して、変更せずにリモートプラグインエンドポイントを起動します。

手順

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

- **Command - Red Hat CodeReady Workspaces** を使用して **Dockerfile#ENTRYPOINT** を上書きします。
- **args: Red Hat CodeReady Workspaces** は **Dockerfile#CMD** を上書きするために使用されます。

- コマンド および args プロパティーが変更された YAML ファイルの例：

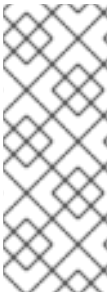
```
apiVersion: v2
category: Language
description: "Typescript language features"
displayName: Typescript
firstPublicationDate: "2019-10-28"
icon: "https://www.eclipse.org/che/images/logo-eclipseche.svg"
name: typescript
publisher: che-incubator
repository: "https://github.com/Microsoft/vscode"
title: "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"
```

- コマンド の代わりに args を変更し、ラッパースクリプト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維持します。

```
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。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

コンテナーの起動時にサービスを実行し、リモートプラグインエンドポイントも起動するには、変更した コマンド および args プロパティーで meta.yaml を使用します。サービスを起動し、プロセスをデタッチし、リモートプラグインエンドポイントを開始し、それらのエンドポイントが並行して機能します。

4.2. 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で環境を再消去します。
- 特定のツールに専用のカスタム IDE を提供します。
- 個々のユーザーまたはユーザーのグループに異なるパースペクティブを提供します。

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ます。Red Hat CodeReady Workspaces に独自のカスタム IDE を追加できます。

- Web IDE を構築するためのフレームワークである Eclipse Theia から 構築される。例 : Web 上
- Jupyter、Eclipse Diggible などのWeb IDE は完全に異なり ます。例： Red Hat CodeReady Workspaces ワークスペースの Jupyter

Eclipse Theia からカスタム IDE をビルド

- Eclipse Theia をベースにした独自のカスタム IDE を作成します。
- CodeReady Workspaces 固有のツールをカスタム IDE に追加します。
- カスタム IDE を CodeReady Workspaces で利用可能なエディターにパッケージ化します。

完全に異なる Web IDE を CodeReady Workspaces に導入

- カスタム IDE を CodeReady Workspaces で利用可能なエディターにパッケージ化します。

4.3. ワークスペース作成後の CODEREADY WORKSPACES への追加

CodeReady Workspaces 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と、CodeReady Workspaces に新しい機能が追加されました。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します。

- OpenShift を含む他のシステムとの統合。
- 一部の開発者タスクを自動化（フォーマット、リファクタリング、自動化テストの実行）。
- IDE から直接複数のデータベースと通信します。
- コードナビゲーション、自動補完、エラーの強調表示が強化されました。

本章では、CodeReady Workspaces ワークスペースでの CodeReady Workspaces 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」

4.3.1. CodeReady Workspaces ワークスペースの追加ツール

CodeReady Workspaces プラグインは、コンテナーイメージにバンドルされた Che-Theia IDE の拡張機能です。たとえば、OpenShift Connector プラグインは oc コマンドがインストールされている必要があります。CodeReady Workspaces プラグインは、プラグインの実行に必要な Linux コンテナーとともに Che-Theia プラグインの一覧です。また、説明、分類タグ、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にインストールできるプラグインのレジストリーを提供します。

多くの VS コード拡張は Che-Theia IDE 内で実行できるため、ランタイムまたはサイドカーコンテナーを組み合わせて、CodeReady Workspaces プラグインとしてパッケージ化できます。ユーザーは、追加設定なしで提供されている多くのプラグインから選択できます。

Dashboard から、レジストリーのプラグインをプラグインタブから追加したり、devfile に追加したり することができます。devfile は、メモリーや CPU の使用量を定義するなど、プラグインのその他の設定にも使用できます。CodeReady Workspaces からプラグインをインストールする場合は、Ctrl+Shift+J を押すか、View → Plugins と選択します。

その他のリソース

- devfile へのコンポーネントの追加

4.3.2. 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、既存のワークスペースにツールを追加する方法を説明します。

CodeReady Workspaces ワークスペースにプラグインとして利用可能なツールを追加するには、以下のいずれかの方法を使用します。

- Dashboard Plugin タブからプラグインを有効にします。
- Dashboard Devfile タブからワークスペース devfile を編集します。

この手順 では、例として Java 言語 サポートプラグインを使用します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」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ている既存のワークスペースは、以下を参照してください。

  - 新しい CodeReady Workspaces ワークスペースの作成および設定

  - ユーザーダッシュボードからのワークスペースの作成

- ワークスペースの状態が 停止 している。これを行うには、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. ダッシュボード で Workspaces メニューをクリックし、ワークスペース一覧を開き、ワークスペースを特定します。

  c. 表示されたワークスペースと同じ行で、画面の右側で 停止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待ってから、をクリックしてワークスペースを設定します。

手順

プラグインレジストリーから既存の CodeReady Workspaces ワークスペースにプラグインを追加するには、以下のいずれかの方法を使用します。

- プラグインタブからプラグインをインストール し ます。

  1. Plugin タブに移動します。

インストールまたはインストール可能なプラグインの一覧が表示されます。

2. Enable sl-toggle を使用して、Java 11 の言語サポートなどの必要なプラグインを 有効 にします。

プラグインソースコードがワークスペース devfile に追加され、プラグインが有効になりました。

3. 画面の右下にある Save ボタンをクリックして変更を保存します。変更が保存されると、ワークスペースが再起動します。

- devfile にコンテンツを追加してプラグインをインストールします。

1. Devfile タブに移動します。

devfile 構造が表示されます。

2. devfile の component セクションを見つけ、以下の行を追加して、ワークスペースに Java 言語 v8 を追加します。

```
- id: redhat/java8/latest
  type: chePlugin
```

最終的な結果の例を参照してください。

```
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: /home/user/.composer
    - name: symfony
      containerPath: /home/user/.symfony
  alias: php
  image: 'quay.io/eclipse/che-php-7:nightly'
- id: redhat/java8/latest
  type: chePlugin
```

その他のリソース

- devfile 仕様

第5章 OAUTH 承認の設定

このセクションでは、Red Hat CodeReady Workspaces を OAuth アプリケーションとしてサポートする OAuth プロバイダーに接続する方法を説明します。

- GitHub OAuth の設定
- OpenShift OAuth の設定

5.1. GITHUB OAUTH の設定

GitHub の OAuth では、GitHub への自動 SSH キー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 には、次の手順が記入されます。

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. GitHub OAuth アプリケーションの Authorization コールバック URL に Redirect URI を貼り付けます。
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. ストアトークンを有効 にし ます。
6. 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 アプリケーションの登録 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、ユーザーが取得した OAuth アクセストークンで API を介してクラスターに対して証明するプロセスです。

CodeReady Workspaces ユーザーが OpenShift クラスターで認証できるようにするには、OpenShift コネクタープラグイン による認証が可能です。

以下のセクションでは、OpenShift OAuth 設定オプションとその CodeReady Workspaces の使用について説明します。

前提条件

- oc ツールが利用可能である。

手順

OpenShift OAuth を自動的に有効にするには、--os-oauth オプションを指定して crwctl を使用してデプロイした CodeReady Workspaces。crwctl server:start 仕様 の章を参照してください。

- シングルユーザーモードでデプロイされた CodeReady Workspaces の場合：

1. OpenShift で CodeReady Workspaces OAuth クライアントを登録します。OpenShift の「 OAuth クライアントの登録」を参照し てください。

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
 name: che
secret: "<random set of symbols>"
redirectURIs:
 - "<CodeReady Workspaces api url>/oauth/callback"
grantMethod: prompt
')
```

2. OpenShift TLS 証明書を CodeReady Workspaces Java トラストストアに追加します。

   - 「 CodeReady Workspaces への自己署名 SSL 証明書の追加」を 参照してくだ

さい。

3. OpenShift デプロイメント設定を更新します。

```
CHE_OAUTH_OPENSHIFT_CLIENTID: <client-ID>
CHE_OAUTH_OPENSHIFT_CLIENTSECRET: <openshift-secret>
CHE_OAUTH_OPENSHIFT_OAUTH__ENDPOINT: <oauth-endpoint>
CHE_OAUTH_OPENSHIFT_VERIFY__TOKEN__URL: <verify-token-url>
```

- <client-ID> は OpenShift OAuthClient に指定された名前です。
- <openshift-secret> は OpenShift OAuthClient に指定されたシークレットです。
- <oauth-endpoint>: OpenShift OAuth サービスの URL:
  - OpenShift 3 では、OpenShift マスター URL を指定します。
  - OpenShift 4 の場合は、oauth-openshift ルートを指定します。
- トークンの検証に使用される <verify-token-url> 要求 URL。<OpenShift master url> /api は OpenShift 3 および 4 に使用できます。
- 「 CodeReady Workspaces configMaps and their behavior 」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、自己署名証明書を使用して、インハウスリポジトリーのアーティファクトと連携するようにさまざまな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

6.1. MAVEN アーティファクトリポジトリーの使用

Maven は 2 つの場所に定義された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 の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。これには、settings.xml が Maven ツールを使用するすべてのコンテナー（特に Maven コンテナーおよび Java プラグインコンテナー）にあることを確認します。

デフォルトでは、settings.xml は Maven 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、一時モードに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、このコンテナーと <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使用するように settings. xml ファイルを設定 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にまたがる 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 Secret オブジェクト（希望の名前）を作成します。必要な settings.xml の内容を Secret の data セクションに配置します（同じディレクトリーに存在する必要のある他のファイルとともに該当する可能性があります）。ファイルまたは環境変数をワークスペースコンテナーにマウントすることで、シークレットのラベル付けおよびこのシークレットにアノテーションを付けます。これにより、Secret の内容がワークス ペース Pod にマウントされるようにします。この Secret を使用するには、以前に実行したワークスペースを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 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. 必要なアノテーションおよびラベルも定義する新規ファイル secret.yaml に出力をコピーします。

```
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/user/.m2
    che.eclipse.org/mount-as: file
type: Opaque
data:
  settings.xml:
PHNldHRpbmdzIHhtbG5zPSJodHRwOi8vbWF2ZW4uYXBhY2hlLm9yZy9TRVRUSU5HU
y8xLjAuMCIKICAgICAgICAgIHhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS
9YTUxTY2hlbWEtaW5zdGFuY2UiCiAgICAgICAgICB4c2k6c2NoZW1hTG9jYXRpb249Im
h0dHA6Ly9tYXZlbi5hcGFjaGUub3JnL1NFVFRJTkdTLzEuMC4wCiAgICAgICAgICAgICA
gICAgICAgICAgICAgICAgIGh0dHBzOi8vbWF2ZW4uYXBhY2hlLm9yZy94c2Qvc2V0dGl
uZ3MtMS4wLjAueHNkIj4KICA8c2VydmVycz4KICAgIDxzZXJ2ZXI+CiAgICAgIDxpZD5yZ
XBvc2l0b3J5LWlkPC9pZD4KICAgICAgPHVzZXJuYW1lPnVzZXJuYW1lPC91c2VybmFtZ
T4KICAgICAgPHBhc3N3b3JkPnBhc3N3b3JkMTIzPC9wYXNzd29yZD4KICAgIDwvc2Vyd
mVyPgogIDwvc2VydmVycz4KPC9zZXR0aW5ncz4K
```

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. 新しいワークスペースを開始します。Maven コンテナーには、元のコンテンツと共に /home/user/.m2/settings.xml が表示さ れ ます。

6.1.2.1. OpenShift 3.11 および OpenShift <1.13

1.13 よりも古いバージョンの OpenShift では、複数の VolumeMounts が同じパスに配置することができません。したがって、devfile のボリューム /home/user/.m2 および /home/user/.m2/settings.xml でシークレットを設定すると競合に解決されます。これらのクラスターでは、devfile の Maven リポジトリーのボリュームとして /home/user/.m2/repository を使用します。

```
apiVersion: 1.0.0
metadata:
  ...
components:
 - type: dockerimage
   alias: maven
   image: maven:3.11
   volumes:
    - name: m2
      containerPath: /home/user/.m2/repository
   ...
```

6.1.3. Java 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、Java では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、内部企業の認証局によって署名されるか、自己署名されます。Java トラストストアに追加して、これらの証明書を受け入れる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。多くの場合、これは tls.crt という名前のファイルです。

    a. Java トラストストアファイルを作成します。

```
$ keytool -import -file tls.crt -alias nexus -keystore truststore.jks -storepass changeit

Trust this certificate? [no]:  yes
Certificate was added to keystore
Owner: CN=example.com
Issuer: CN=example.com
Serial number: 80ca0f6980c6019a
Valid from: Thu Feb 06 11:00:29 CET 2020 until: Fri Feb 05 11:00:29 CET 2021
Certificate fingerprints:
     MD5:  88:3C:EC:E1:BE:57:DD:9D:46:36:8E:DD:BF:14:04:22
     SHA1: 08:D8:79:D3:F8:6B:5C:3D:71:AA:23:CA:72:01:47:BD:9D:91:0A:AD
     SHA256:
```

```
5C:BB:66:81:44:D2:50:EE:EB:CE:D6:15:7E:63:E1:9A:71:EA:58:3F:14:01:15:4E:68:5D:71:
0A:A0:31:33:29
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  DNSName: *.apps.example.com
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

b. **/projects/maven/truststore.jks にトラストストアファイルをアップロードして、すべてのコンテナーで利用できるようにします。**

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行います。**

  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/truststore.jks
```

  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行います。**

  devfile に、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/truststore.jks
[...]
```

6.2. GRADLE アーティファクトリポジトリーの使用

6.2.1. 異なるバージョンの 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 ラッパー を実行して Wrapper を設定します。

前提条件

- プロジェクトに Gradle Wrapper がある。

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 のラッパー設定を変更します。

- distributionUrl プロパティー を 変更して、Gradle distribution ZIP ファイルの URL を参照します。

```
properties
distributionUrl=http://<url_to_gradle>/gradle-6.1-bin.zip
```

ワークスペースの /project/gradle に Gradle ディストリビューションの zip ファイルをローカルに配置することができます。

- distributionUrl プロパティー を 変更して、Gradle distribution zip ファイルのローカルアドレスを参照します。

```
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、プロジェクトを評価する前に追加の設定を実行します。この設定はワークスペースの各 Gradle プロジェクトで使用されます。

手順

ワークスペースの各 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、指定した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 Gradle を設定します。

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は維持されないため、Java プラグインサイドカーコンテナーで起動する各ワークスペースで init.gradle スクリプトを作成する必要があります。

6.2.3. Java 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、Java では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、内部企業の認証局によって署名されるか、自己署名されます。Java トラストストアに追加して、これらの証明書を受け入れる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。多くの場合、これは tls.crt という名前のファイルです。

a.

Java トラストストアファイルを作成します。

```
$ keytool -import -file tls.crt -alias nexus -keystore truststore.jks -storepass changeit

Trust this certificate? [no]:  yes
Certificate was added to keystore
Owner: CN=example.com
Issuer: CN=example.com
Serial number: 80ca0f6980c6019a
Valid from: Thu Feb 06 11:00:29 CET 2020 until: Fri Feb 05 11:00:29 CET 2021
Certificate fingerprints:
     MD5:  88:3C:EC:E1:BE:57:DD:9D:46:36:8E:DD:BF:14:04:22
     SHA1: 08:D8:79:D3:F8:6B:5C:3D:71:AA:23:CA:72:01:47:BD:9D:91:0A:AD
     SHA256:
5C:BB:66:81:44:D2:50:EE:EB:CE:D6:15:7E:63:E1:9A:71:EA:58:3F:14:01:15:4E:68:5D:71:
0A:A0:31:33:29
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  DNSName: *.apps.example.com
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

b.

/projects/gradle/truststore.jks にトラストストアファイルをアップロードして、すべてのコンテナーで利用できるようにします。

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/truststore.jks
```

その他のリソース

- 初期化スクリプトに関する 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. 非標準レジストリーを使用する Python の設定

Python pip ツールで使用する非標準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開発コンテナー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: 'quay.io/eclipse/che-python-3.7:nightly'
  env:
    - name: 'PIP_INDEX_URL'
      value: 'https://<username>:<password>@pypi.company.com/simple'
```

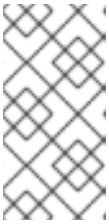
6.3.2. Python プロジェクトでの自己署名証明書の使用

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の TLS 証明書がありません。通常は、内部企業の認証局によって署名されるか、自己署名されます。これらの証明書を受け入れるツールを設定します。

Python は、PIP_CERT 環境変数で定義されているファイルからの証明書を使用します。

手順

1. 非標準リポジトリーから証明書を取得し、証明書ファイルを /projects/tls/rootCA.pem ファイルに配置して、すべてのコンテナーからアクセスできるようにします。



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: 'quay.io/eclipse/che-python-3.7:nightly'
  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 環境変数および 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. 非標準レジストリーを使用するように Go の設定

Athens は、多くの設定オプションを持つ Go モジュールデータストアおよびプロキシーです。プロキシーとしてではなく、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ます。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、Go プロジェクト全体で利用できるようにすることができます。プロジェクトが 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 ビルドは失敗します。

- Athens を使用するには、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の TLS 証明書がありません。通常は、内部企業の認証局によって署名されるか、自己署名されます。これらの証明書を受け入れるツールを設定します。

Go は、SSL_CERT_FILE 環境変数で定義されたファイルから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、これを /projects/tls/rootCA.crt ファイルに配置して、すべてのコンテナーからアクセスできるようにします。

2. プロジェクト選択を右クリックし、Upload files を選択して rootCA .crt 証明書ファイルを Red Hat CodeReady Workspaces ワークスペースにアップロードします。

3. 適切な環境変数を devfile に追加します。

```
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
  - value: 'on'
```

```
    name: GO111MODULE
  - value: '/projects/tls/rootCA.crt'
    name: SSL_CERT_FILE
```

その他のリソース

- GitHub - gomods/athens: A 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. NuGet が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設定

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の設定ファイルを検索します。nuget .config ファイルを /projects ディレクトリーに配置した場合、nuget .config ファイルは /projects のすべてのプロジェクトの NuGet 動作を定義します。

手順

- nuget .config ファイルを作成して、/projects ディレクトリーに配置します。

nexus .example.org でホストされる Nexus リポジトリーを持つ 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の TLS 証明書がありません。通常は、内部企業の認証局によって署名されるか、自己署名されます。これらの証明書を受け入れるツールを設定します。

手順

1. 非標準リポジトリーの証明書ファイルを取得して /projects/tls/rootCA.crt ファイルに配置して、すべてのコンテナーからアクセスできるようにします。

2. OmniSharp プラグインおよび .NET コンテナーの devfile の SSL_CERT_FILE 環境変数に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: /projects/tls/rootCA.crt
      name: SSL_CERT_FILE
 - mountSources: true
   endpoints:
    - name: 5000/tcp
      port: 5000
   memoryLimit: 512Mi
   type: dockerimage
   volumes:
    - name: dotnet
      containerPath: /home/user
   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

NPM は、通常 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む。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: NPM_CONFIG_REGISTRY
- ファイルの証明書を使用する場合： NODE_EXTRA_CA_CERTS

devfile で証明書を参照できるようにするには、npm リポジトリーサーバーの証明書のコピーを取得して、/project フォルダー内に配置します。

1. 自己署名証明書と共に内部リポジトリーを使用する設定例：

```
- 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: '/projects/config/tls.crt'
   - 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

7.1. 起動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

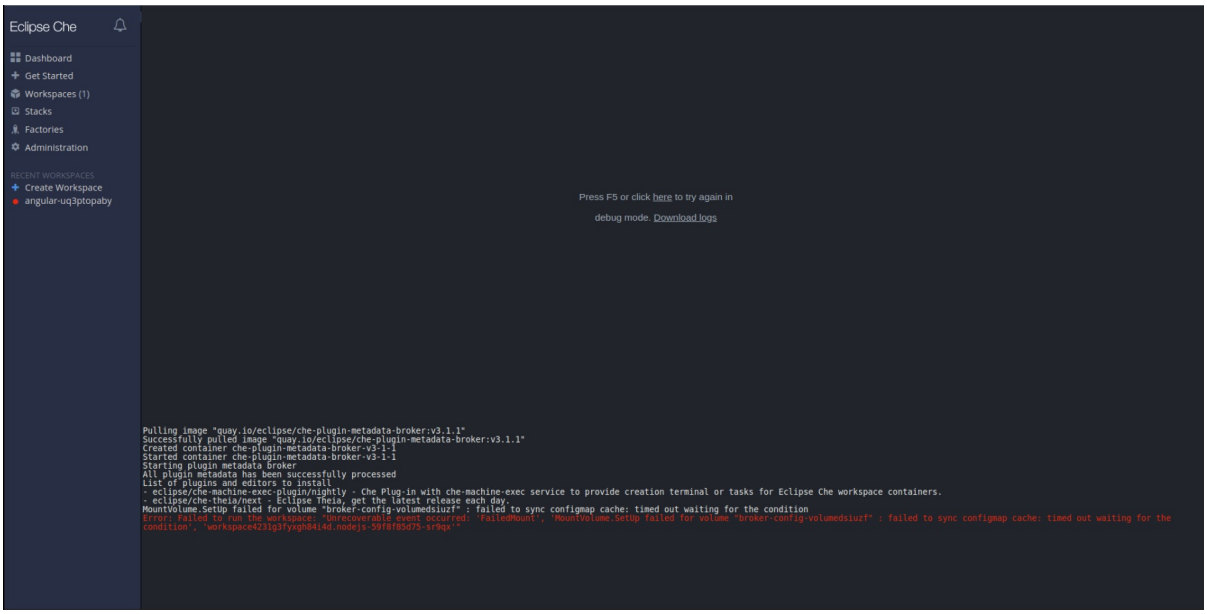
本セクションでは、ワークスペースの開始後にデバッグモードで Red Hat CodeReady Workspaces ワークスペースを再起動する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、「Installing CodeReady Workspaces on OpenShift Container Platform」を参照してください。
- 起動に失敗した既存のワークスペース。

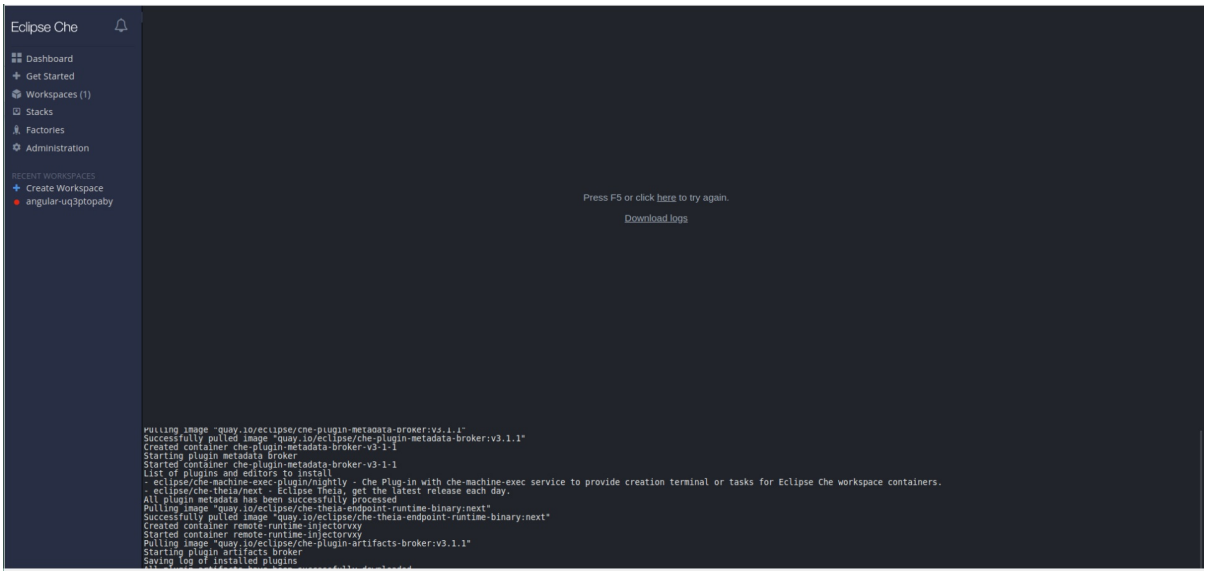
手順

1. 最新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. 開始後にログをすべてダウンロードすると、Download logs リンクが表示されます。



7.2. デバッグモードでの CODEREADY WORKSPACES ワークスペースの開始

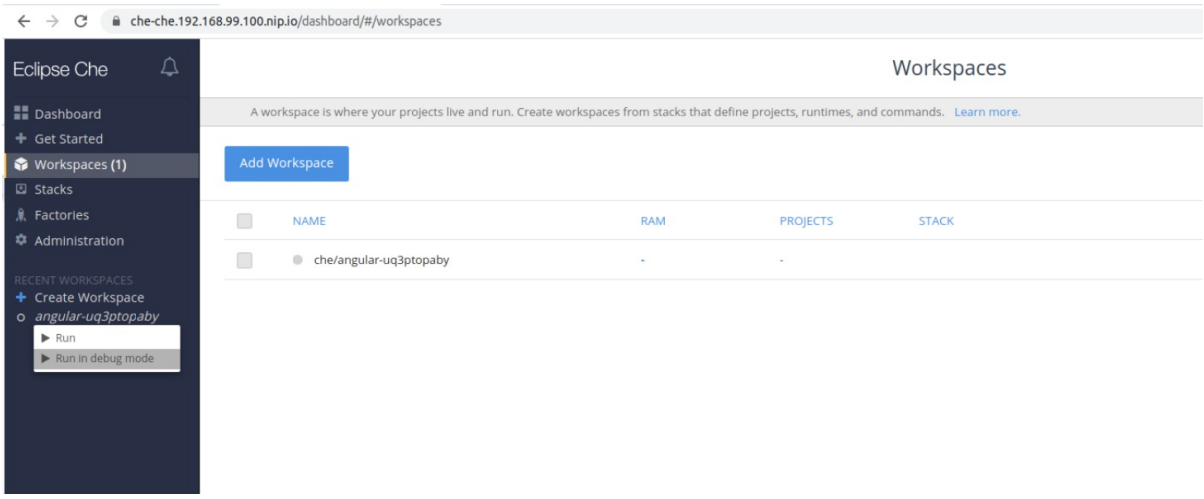
本セクションでは、デバッグモードで Red Hat CodeReady Workspaces ワークスペースを起動する方法を説明します。

前提条件

- Red Hat CodeReady Workspaces の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタート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。「新しいワークスペースの作成」を参照してください。

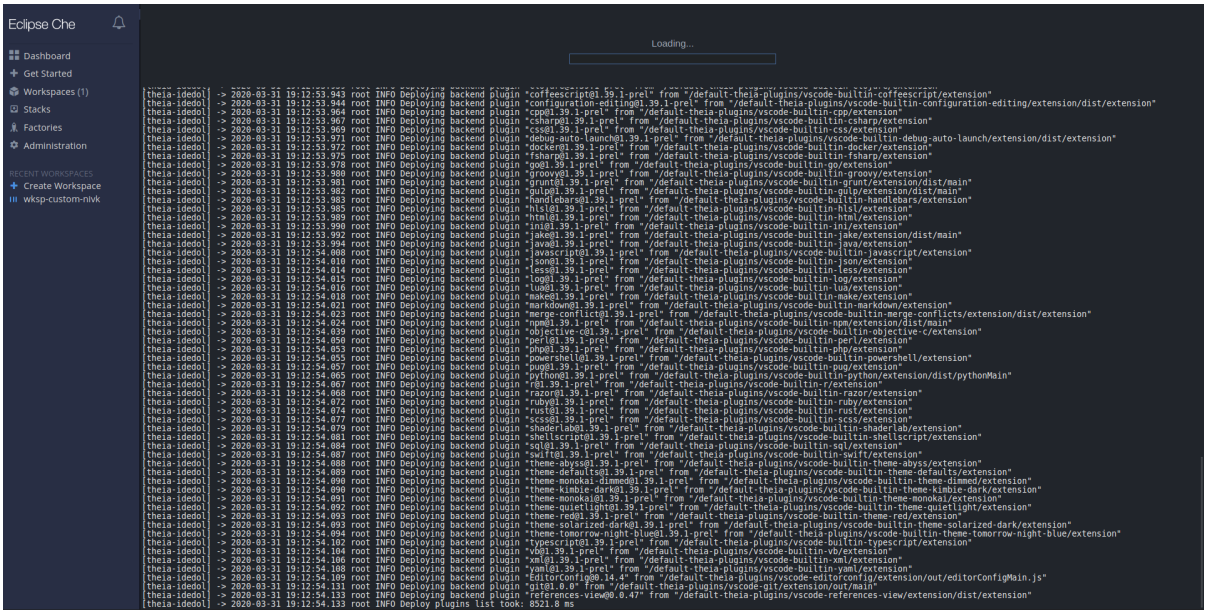
手順

1. 最新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、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、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。



7.3. 異なるタイプのストレージの使用

7.17.0 Red Hat CodeReady Workspaces では、永続ストレージ、一時および非同期の 3 つのタイプのストレージがサポートされます。

永続ストレージ

このタイプのストレージにより、変更をマウントされた永続ボリュームに直接保存できます。この場合、変更は完全に安全になります。OpenShift インフラストラクチャーが処理します（特にストレージバックエンド）。この価格は低速な I/O です（特に小規模なファイルの場合）。たとえば、NodeJS プロジェクトには多くの依存関係があり、node_modules には数千もの小さなファイルが埋められます。

注記

I/O の速度は、環境に設定さ れたストレージクラス によって異なります。

現在、これは新しいワークスペースのデフォルトモードですが、追加の属性は必要ありません。ワークスペース設定でこの設定を明示的に表示する場合は、以下を追加します。

```
attributes:
 persistVolumes: 'true'
```

一時ストレージ

このようなストレージを使用すると、ファイルは emptyDir ボリュームにマウントされます。名前で示すように、最初は空です。Pod が何らかの理由でノードから削除されると、emptyDir のデータは永久に削除されます。つまり、何らかの理由でワークスペースが停止または再起動すると、すべての変更が失われます。

警告

変更を保存するには、一時ワークスペースを停止する前に、リモートにコミットおよびプッシュします。それ以外の場合は、すべての変更が失われます。

したがって、ワークスペースを停止する前に、変更をリモートにコミットおよびプッシュする必要があります。同時に、Ephemeral モードを使用すると I/O が高速になります。そのため、どのような種類のプロジェクトでも迅速に動作します。このストレージタイプを有効にするには、以下をワークスペース設定に追加します。

```
attributes:
 persistVolumes: 'false'
```

7.3.1. Ephemeral(emptyDir)vs Persitent(AWS EBS)の比較テーブル

コマンド	ephemeral	Persitent
Red Hat CodeReady Workspaces のクローン	0m 19s	1m 26s

コマンド	ephemeral	Persitent
無作為なファイルを生成する 1000	1m 12s	44m 53s

非同期ストレージ（実験）

これは上記の 2 つの組み合わせです。最初のワークスペースコンテナーは emptyDir をマウントしますが、ワークスペースの起動時に変更が復元され、ワークスペースの停止にバックアップが実行されます。このストレージタイプは高速 I/O（一時モードと同様）を提供し、ワークスペースプロジェクトの変更は永続化されます。

同期は、SSH プロトコルを介して rsync により動作 します。ワークスペースを非同期ストレージタイプで設定すると、workspace-data-sync プラグインがワークスペース設定に自動的に追加されます。プラグインは、ワークスペースで rsync コマンドを実行し、変更が存在する場合に復元を開始します。ワークスペースが停止したら、変更を永続ストレージに送信します。rsync プロセスには多少時間がかかります。比較的小規模なプロジェクトでは、復元手順は速く完了し、CodeReady Workspaces Theia の初期化直後にプロジェクトソースファイルおよびフォルダーを確認できます。rsync の時間が長い場合は、CodeReady Workspaces Theia UI ステータスバーエリアに同期プロセスが表示されます。（CodeReady Workspaces Theia リポジトリーの拡張機能）。

注記

現在、このモードには、common' PVC ストラテジーでのみサポートする、'common' PVC ストラテジーでのみサポートされる制限があります。ユーザーは非同期ストレージを持つ 1 つのワークスペースを同時に実行できます。

ワークスペースの非同期ストレージを設定するには、以下の設定をワークスペースに追加します。

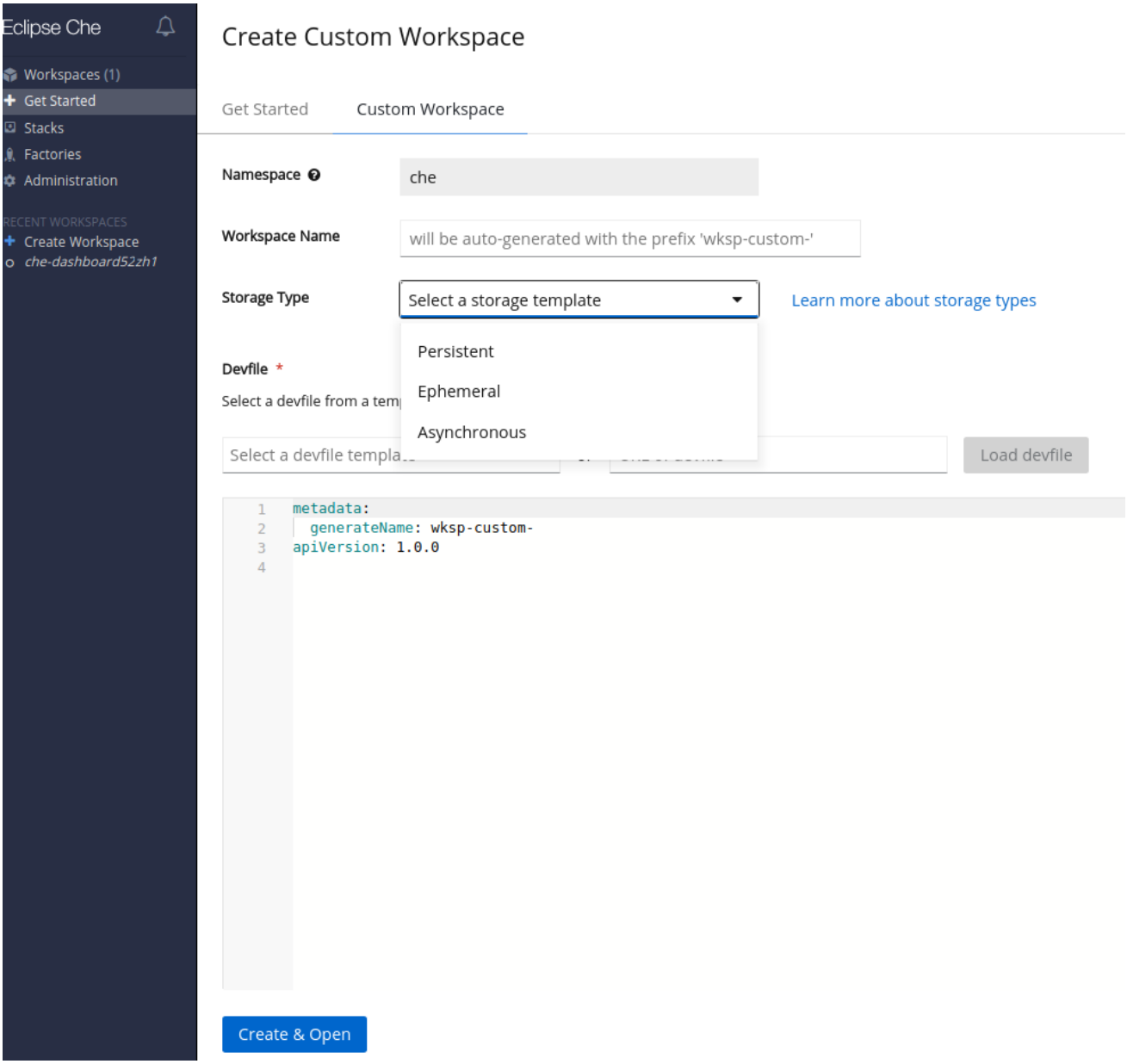
```
attributes:
 asyncPersist: 'true'
 persistVolumes: 'false'
```

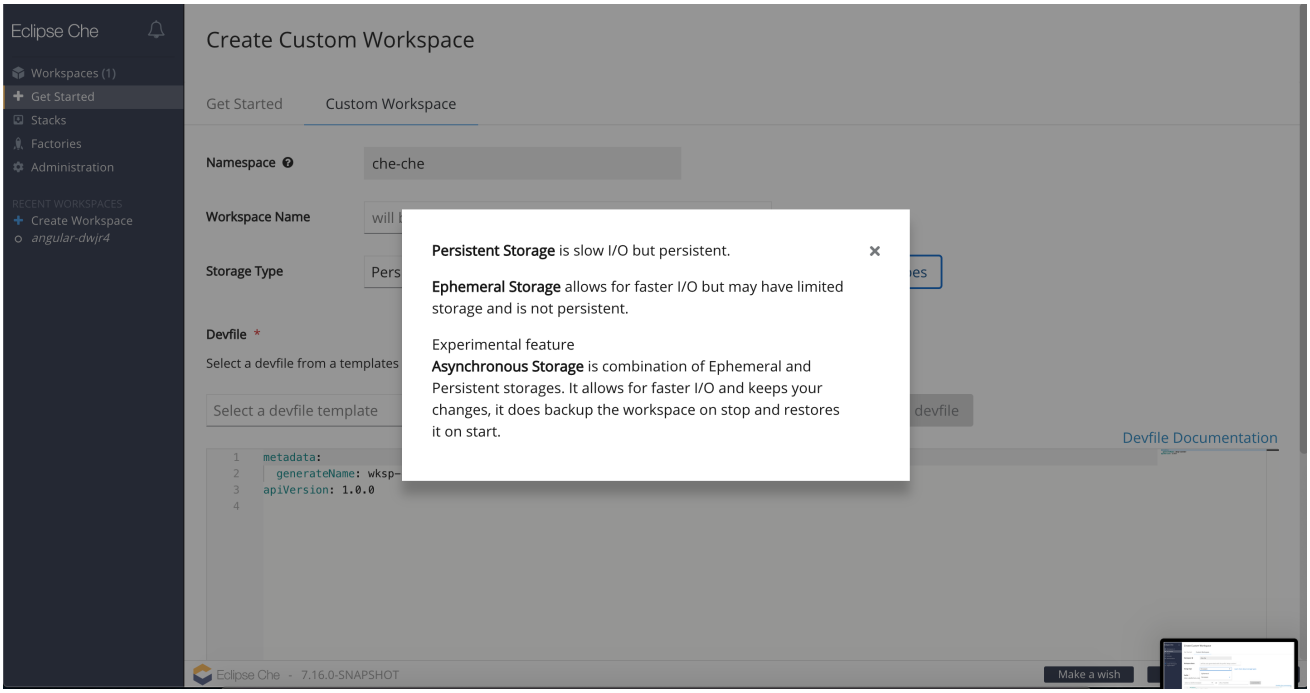
CodeReady Workspaces UserDashboard のようにクライアントの動作を設定するために使用できる che.properties に 2 つの新しいプロパティーが追加されました。

che.workspace.storage.available_types: この設定プロパティーは、ワークスペースの作成/更新時に Dashboard のようなクライアントがユーザーに提案するストレージタイプの利用可能な値を定義します。利用可能な値は 'persistent','ephemeral', 'async' です。複数のタイプをコンマで区切って使用できます（例： che.workspace.storage.available_types=persistent,ephemeral,async）。

che.workspace.storage.preferred_type: この設定プロパティーは、Dashboard のようなクライアントがワークスペースの作成/更新時にユーザーに提案する必要のあるストレージタイプのデフォルト値を定義します。実験的なものであるため、「async」値はデフォルトのタイプとして推奨されません（例： che.workspace.storage.preferred_type=persistent）。

ストレージタイプスイッチャーは、ユーザーダッシュボードの カスタムワークスペースの作成 ページで利用できます。


Eclipse Che
Workspaces (1)
Get Started
Stacks
Factories
Administration
RECENT WORKSPACES
Create Workspace
che-dashboard52zh1
Create Custom Workspace
Get Started
Custom Workspace
Namespace
che
Workspace Name
will be auto-generated with the prefix 'wksp-custom-'
Storage Type
Select a storage template
Learn more about storage types
Persistent
Ephemeral
Asynchronous
Devfile *
Select a devfile from a tem
Load devfile
metadata:
generateName: wksp-custom-
apiVersion: 1.0.0
Create & Open


Eclipse Che
Create Custom Workspace
Get Started
Custom Workspace
Namespace
che-che
Workspace Name
Storage Type
Devfile *
Select a devfile from a templates
Select a devfile template
Persistent Storage is slow I/O but persistent.
Ephemeral Storage allows for faster I/O but may have limited storage and is not persistent.
Experimental feature
Asynchronous Storage is combination of Ephemeral and Persistent storages. It allows for faster I/O and keeps your changes, it does backup the workspace on stop and restores it on start.
Devfile Documentation
Eclipse Che - 7.16.0-SNAPSHOT
Make a wish

第8章 OPENSHIFT CONNECTOR の概要

OpenShift Connector は、Red Hat OpenShift の Visual Studio Code OpenShift Connector とも呼ばれ、Red Hat OpenShift 3 または 4 クラスターと対話するための 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、ビルド、およびデバッグでき、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、OpenShift CLI(oc)コマンドをコンパクトな単位に集約します。そのため、OpenShift Connector は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ことで OpenShift 背景を持たない新規開発者を支援します。複数の oc コマンドを使用する代わりに、ユーザーはプロジェクト、アプリケーション、サービスなどの事前設定されたテンプレートを選択し、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て新規コンポーネントまたはサービスを作成します。

このセクションでは、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 コネクターの機能
- Red Hat CodeReady Workspaces への OpenShift コネクターのインストール
- Red Hat CodeReady Workspaces からの OpenShift コネクターでの認証
- Red Hat CodeReady Workspaces での OpenShift コネクターでのコンポーネントの作成
- OpenShift コネクターを使用した GitHub から OpenShift コンポーネントへのソースコードの接続

8.1. OPENSHIFT コネクターの機能

OpenShift Connector プラグインを使用すると、ユーザーは GUI で OpenShift コンポーネントを作成、デプロイ、およびプッシュできます。

CodeReady Workspaces で使用されると、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

- トークンとユーザー名およびパスワードの組み合わせを使用してクラスターにログインします。
- コンテキストをエクステンションビューから直接異なる .kube/config エントリー間で切り替えます。
- Explorer ビューで、ビルドおよびデプロイメントとして OpenShift リソースを表示し、管理します。

Development

- CodeReady Workspaces から直接ローカルまたはホストされた OpenShift クラスターに接続する。
- 変更内容でクラスターを迅速に更新します。
- 接続されたクラスターでコンポーネント、サービス、ルートを作成する。
- ストレージをエクステンション自体からコンポーネントに直接追加。

deployment

- CodeReady Workspaces から直接クリックする 1 つのクリックで OpenShift クラスターにデプロイします。
- 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 Routes に移動します。
- 複数の相互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します。

- CodeReady Workspaces IDE のコンポーネント変更をプッシュおよび監視します。
- 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します。

モニタリング

- CodeReady Workspaces IDE から直接 OpenShift リソースを使用する
- 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を表示し、以下のログに続きます。

8.2. CODEREADY WORKSPACES への OPENSHIFT コネクターのインストール

OpenShift Connector は、CodeReady Workspaces をエディターとして使用し、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。インスタンスで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、CodeReady Workspaces の左側のメニューに OpenShift アイコンが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 コネクターをインストールし、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 の稼働中のインスタンス。Red Hat CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイックスタートを参照してください。

手順

CodeReady Workspaces パネルでエクステンションとして OpenShift Connector を追加し、CodeReady Workspaces に OpenShift Connector をインストール し ます。

1. Ctrl+Shift+J を押すか、View → Plugins に移動し、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. 変更を有効にするためにワークスペースを再起動します。

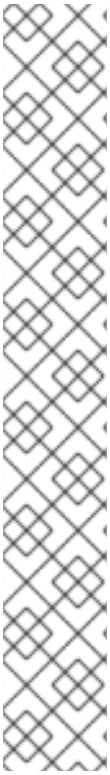
4. 専用の OpenShift Application Explorer アイコンが左側のパネルに追加されます。

8.3. CODEREADY WORKSPACES からの OPENSHIFT コネクターでの認証

ユーザーが CodeReady Workspaces からコンポーネントを開発し、プッシュできるようにするには、OpenShift クラスターで認証する必要があります。

OpenShift コネクターは、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。
- クラスターへのログイン ボタンの使用
- コマンドペアの使用



注記

CodeReady Workspaces 2.3 では、Openshift Connector プラグインでターゲットクラスターへの手動接続が必要になります。

デフォルトでは、Openshift Connector プラグインは、ClusterUser のようにクラスターにログインし 、 プロジェクトのパーミッションを持たない可能性があります。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、OpenShift ユーザーの認証情報を使用してローカルクラスターからログアウトし、OpenShift クラスターにログインします。

OpenShift のローカルインスタンスを使用する場合（CodeReady Containers や Minishift など）、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されます。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 の稼働中のインスタンス。CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイック起動を参照し てください。
- CodeReady Workspaces ワークスペースが作成されました。
- OpenShift Connector プラグインが利用できます。
- OpenShift OAuth プロバイダーは（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）設定されます。「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。以下のいずれかの方法を使用します。

   - 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 Command Pal✓ を開くか、トップメニューの 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. というメッセージが表示される場合は、Yes をクリックします。

   画面上部に Credentials または Token を使用してログインするかどうかを選択できます。

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、必要なトークン情報は、主な OpenShift Container Platform 画面の右上隅の <User name> → Copy Login Command の下にあります。

8.4. CODEREADY WORKSPACES での OPENSHIFT コネクターでのコンポーネントの作成

OpenShift のコンテキストでは、コンポーネントおよびサービスはアプリケーションに格納する必要のある基本的な構造です。これは、deployable を仮想フォルダーに整理し、読みやすさを向上させるために 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 CodeReady Workspaces で OpenShift コンポーネントを作成し、それらを OpenShift クラスターにプッシュする方法について説明します。

前提条件

- CodeReady Workspaces の稼働中のインスタンス。CodeReady Workspaces のインスタンスをインストールするには、『CodeReady Workspaces 2.3 Installation GuideCodeReady Workspaces』のクイック起動を参照し てください。

- ユーザーが 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 アイコンで行を右クリックして 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. 作成され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しい 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、ランタイムのリビジョンを選択するように求められます。

   b. バイナリーファイル

      これにより、ファイルからファイルを選択するように求められます。

   c. workspace Directory

これにより、ファイルからフォルダーを選択するように求められます。

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て New URL を選択し、選択した名前を入力します。

9. コンポーネントは OpenShift クラスターにプッシュされる準備ができています。これを行うには、コンポーネントを右クリックし、Push を選択します。

コンポーネントはクラスターにデプロイされます。デバッグやブラウザーで開くなど、追加のアクションを右クリックします（公開されるポート 8080 が必要です）。

8.5. OPENSHIFT コネクターを使用した GITHUB から OPENSHIFT コンポーネントへのソースコードの接続

ユーザーにさらなる開発に必要な Git ストアドソースコードがある場合は、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方が効率的です。

本章では、Git リポジトリーからコンテンツを取得し、CodeReady Workspaces で開発した 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されている。
- 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、リポジトリーを CodeReady Workspaces にクローンし、このソースコードを取得します。

1. CodeReady Workspaces メイン画面で、F1 を押し て Command Pal— を開きます。
2. コマンドペアに Git Clone コマンドを入力 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。