Red Hat OpenShift Dev Spaces 3.0

ユーザーガイド

Red Hat OpenShift Dev Spaces 3.0 の使用

Last Updated: 2023-04-16

Red Hat OpenShift Dev Spaces 3.0 ユーザーガイド

Red Hat OpenShift Dev Spaces 3.0 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 OpenShift Dev Spaces を使用するユーザー向けの情報

目次

第1章 OPENSHIFT DEV SPACES の採用

組織での OpenShift Dev Spaces の採用を開始するには、以下をお読みください。

- 「開発者ワークスペース」
- 「リンク付きのバッジを使用して、初めての貢献者がワークスペースを開始できるようにする」
- 「Red Hat OpenShift Dev Spaces でプルおよびマージリクエストを確認する利点」
- 「サポートされる言語」

1.1. 開発者ワークスペース

Red Hat OpenShift Dev Spaces は、アプリケーションのコーディング、ビルド、テスト、実行、およびデバッグに必要なすべてのものを開発者ワークスペースに提供します。

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

Pod は OpenShift Dev Spaces ワークスペースの各コンポーネントを管理します。したがって、OpenShift Dev Spaces ワークスペースで実行しているものはすべてコンテナー内で実行します。これにより、OpenShift Dev Spaces ワークスペースの移植性が高くなります。

組み込みのブラウザーベースの IDE は、OpenShift Dev Spaces ワークスペースで実行しているすべてのものへのアクセスポイントです。これにより、OpenShift Dev Spaces ワークスペースを簡単に共有できます。

1.2. リンク付きのバッジを使用して、初めての貢献者がワークスペースを開始できるようにする

初めての貢献者がプロジェクトでワークスペースを開始できるようにするには、OpenShift Dev Spaces インスタンスへのリンクを含むバッジを追加します。

図1.1 ファクトリーバッジ

手順

1. OpenShift Dev Spaces URL (**https://devspaces-<openshift_deployment_name>.<domain_name>**) とリポジトリー URL (**<your-repository-url>**) に置き換えて、プロジェクトの **README.md** ファイルにリポジトリーへのリンクを追加します。

```
[![Contribute](https://www.eclipse.org/che/contribute.svg)]
(https://devspaces-<openshift_deployment_name>.<domain_name>/#https://<your-repository-url>)
```

2. Git プロバイダーの Web インターフェイスの **README.md** ファイルには、 

ファクトリーバッジが表示されます。バッジをクリックして、OpenShift Dev Spaces インスタンスでプロジェクトを含むワークスペースを開きます。

1.3. RED HAT OPENSHIFT DEV SPACES でプルおよびマージリクエストを確認する利点

Red Hat OpenShift Dev Spaces ワークスペースには、プルリクエストとマージリクエストを最初から最後まで確認するために必要なすべてのツールが含まれています。OpenShift Dev Spaces リンクをクリックすると、Red Hat OpenShift DevSpaces でサポートされている Web IDE にアクセスでき、リンター、単体テスト、ビルドなどを実行できるすぐに使用できるワークスペースがあります。

前提条件

- Git プロバイダーによってホストされているリポジトリーにアクセスできる。
- Red Hat OpenShift Dev Spaces でサポートされているブラウザー (Google Chrome または Mozilla Firefox) を使用する。
- OpenShift Dev Spaces インスタンスにアクセスできる。

手順

1. 機能ブランチを開いて、OpenShift Dev Spaces で確認します。ブランチのクローンが、デバッグとテスト用のツールを備えたワークスペースで開きます。
2. プルまたはマージ要求の変更を確認してください。
3. 必要なデバッグおよびテストツールを実行します。
   - リンターを実行します。
   - ユニットテストを実行します。

- ビルドを実行します。
- アプリケーションを実行して問題を確認します。

4. Git プロバイダーの UI に移動してコメントを残し、割り当てられたリクエストをプルまたはマージします。

検証

- (オプション) リポジトリーのメインブランチを使用して 2 番目のワークスペースを開き、問題を再現します。

1.4. サポートされる言語

Java 11 と JBoss EAP 7.4

OpenJDK 11、Maven 3.6、および JBoss EAP 7.4 を使用する Java スタック

Java 11 と JBoss EAP XP 3.0 の起動可能な Jar

OpenJDK 11、Maven 3.6、および JBoss EAP XP 3.0 Bootable Jar を使用した Java スタック

JBoss EAP XP 3.0 マイクロプロファイルを使用した Java 11

OpenJDK 11、Maven 3.6、および JBoss EAP XP 3.0 を使用する Java スタック

Red Hat Fuse

OpenJDK 11 および Maven 3.6.3 を使用する Red Hat Fuse スタック

Tooling for Apache Camel K

Apache Camel K との統合プロジェクトを開発するためのツール

Java 11 と Gradle

Java スタックと OpenJDK 11、Maven 3.6.3、および Gradle 6.1

Java 11 と Lombok

OpenJDK 11、Maven 3.6.3、および Lombok 1.18.18 を使用した Java スタック

Java 11 と Quarkus

OpenJDK 11、Maven 3.6.3、Gradle 6.1、および Quarkus ツールを使用した Java スタック

Java 11 と Vert.x

OpenJDK 11、Maven 3.6.3、および Vert.x ブースターを使用した Java スタック

Java 11 と Maven

OpenJDK 11、Maven 3.6.3、および Vert.x デモを使用した Java スタック

Java 8 と Spring Boot

Java スタックと OpenJDK 8、Maven 3.6.3、および Spring Boot Petclinic デモアプリケーション

NodeJS ConfigMap Express

NPM 8、NodeJS 16、および ConfigMap Web アプリケーションを使用した NodeJS スタック

NodeJS MongoDB

NodeJS スタックと NPM 8、NodeJS 16、および MongoDB 3.6

NodeJS Express

NodeJS スタックと NPM 8、NodeJS 16、および Express Web アプリケーション

Python

Python 3.8 と pip 19.3 を使用した Python スタック

C/C++

GCC、cmake、および make を使用した C および C++ 開発者ツールスタック (テクノロジープレビュー)

.NET

.NET Core SDK 6 および 3.1、ランタイム、C# 言語サポート、およびデバッガー (テクノロジープレビュー) を備えた .NET スタック

Go

Stack with Go (テクノロジープレビュー)

PHP CakePHP

PHP スタック、PHP、Apache Web サーバー、Composer、および OpenShift 用のクイックスタート CakePHP アプリケーション (テクノロジープレビュー)

PHP-DI

PHP、Apache Web サーバー、および Composer を使用した PHP スタック (テクノロジープレビュー)

第2章 ユーザーのオンボーディング

組織ですでに OpenShift Dev Spaces インスタンスを実行している場合は、新しいワークスペースを開始し、ワークスペースを管理し、ワークスペースから Git サーバーに対して自分自身を認証する方法を学習することで、新しいユーザーとして開始できます。

1. 「Git リポジトリーのクローンを使用して新しいワークスペースを開始」
2. 「新しいワークスペースを開始するための URL の任意のパラメーター」
3. 「ワークスペースで実行できる基本的なアクション」
4. 「ワークスペースから Git サーバーへの認証」

2.1. GIT リポジトリーのクローンを使用して新しいワークスペースを開始

ブラウザーで OpenShift Dev Spaces を操作するには、複数の URL が必要です。

- 以下のすべての URL の一部として使用される組織の OpenShift Dev Spaces インスタンスの URL
- ワークスペースコントロールパネルを備えた OpenShift Dev Spaces ダッシュボードの **ワークスペース** ページの URL
- 新しいワークスペースを開始するための URL
- 使用中のワークスペースの URL

OpenShift Dev Spaces を使用すると、ブラウザーで URL にアクセスして、Git リポジトリーのクローンを含む新しいワークスペースを開始できます。このようにして、GitHub、GitLab インスタンス、または Bitbucket サーバーでホストされている Git リポジトリーのクローンを作成できます。

ヒント

OpenShift Dev Spaces ダッシュボードの **Create Workspace** ページにある **Git Repo URL** *フィールドを使用して、Git リポジトリーの URL を入力し、新しいワークスペースを開始することもできます。

前提条件

- 組織に、OpenShift Dev Spaces の実行中のインスタンスがある。
- 組織の OpenShift Dev Spaces インスタンスの FQDN URL が分かっている: **https://devspaces-<openshift_deployment_name>.<domain_name>**
- Git リポジトリーのメンテナーは、**devfile.yaml** または **.devfile.yaml** ファイルを Git リポジトリーのルートディレクトリーに保持します。(代替ファイル名とファイルパスについては、「新しいワークスペースを開始するための URL の任意のパラメーター」を参照してください。)

  #### ヒント

  devfile を含まない Git リポジトリーの URL を指定して、新しいワークスペースを開始することもできます。そうすることで、Che-Theia IDE と Universal Developer Image を備えたワークスペースが作成されます。

手順

Git リポジトリーのクローンを使用して新しいワークスペースを開始するには、以下を行います。

1. オプション: OpenShift Dev Spaces ダッシュボードページにアクセスして、組織の OpenShift Dev Spaces のインスタンスを認証します。
2. URL にアクセスして、基本的な構文を使用して新しいワークスペースを開始します。

> https://devspaces-**<openshift_deployment_name>.<domain_name>#<git_repository_url>**

ヒント

この URL は、任意のパラメーターを使用して拡張できます。

> https://devspaces-**<openshift_deployment_name>.<domain_name>#<git_repository_url>?<optional_parameters>** (1)

(1) 「新しいワークスペースを開始するための URL の任意のパラメーター」を参照してください。

> **例2.1 新しいワークスペースを開始するための URL**
>
> **https://devspaces-<openshift_deployment_name>.<domain_name>#https://github.com/che-samples/cpp-hello-world**

> **例2.2 GitHub でホストされているリポジトリーのクローンを使用して新しいワークスペースを開始するための URL 構文**
>
> GitHub と GitLab を使用すると、クローンを作成するリポジトリーの特定のブランチの URL を使用することもできます。
>
> - **https://devspaces-<openshift_deployment_name>.<domain_name>#https://github.com/<user_or_org>/<repository>** は、デフォルトのブランチのクローンを使用して新しいワークスペースを開始します。
> - **https://devspaces-<openshift_deployment_name>.<domain_name>#https://github.com/<user_or_org>/<repository>/tree/<branch_name>** は、指定されたブランチのクローンを使用して新しいワークスペースを開始します。
> - **https://devspaces-<openshift_deployment_name>.<domain_name>#https://github.com/<user_or_org>/<repository>/pull/<pull_request_id>** は、プルリクエストのブランチのクローンを使用して新しいワークスペースを開始します。

ブラウザータブで新しいワークスペースを開始するための URL を入力すると、ワークスペース開始ページが表示されます。

新しいワークスペースの準備ができると、ワークスペース IDE がブラウザータブにロードされます。

Git リポジトリーのクローンは、新しいワークスペースのファイルシステムに存在します。

ワークスペースには一意の URL があります:
https://devspaces-<openshift_deployment_name>.<domain_name>#workspace<unique_url>

ヒント

これはアドレスバーではできませんが、ブラウザーのブックマークマネージャーを使用して、新しいワークスペースをブックマークとして開始するための URL を追加できます。

- Mozilla Firefox で、☰ > **Bookmarks** > **Manage bookmarks Ctrl+Shift+O** > **Bookmarks Toolbar** > **Organize** > **Add bookmark** に移動します。
- Google Chrome で、⋮ > **Bookmarks** > **Bookmark manager** > **Bookmarks bar** > ⋮ > **Add new bookmark** に移動します。

関連情報

- 「新しいワークスペースを開始するための URL の任意のパラメーター」
- 「ワークスペースで実行できる基本的なアクション」

2.2. 新しいワークスペースを開始するための URL の任意のパラメーター

新しいワークスペースを開始すると、OpenShift Dev Spaces は devfile の指示に従ってワークスペースを設定します。URL を使用して新しいワークスペースを開始する場合は、ワークスペースをさらに設定する任意のパラメーターを URL に追加できます。これらのパラメーターを使用して、ワークスペース IDE を指定し、複製ワークスペースを開始し、devfile ファイル名またはパスを指定できます。

- 「URL パラメーターの連結」
- 「ワークスペース IDE の URL パラメーター」
- 「重複するワークスペースを開始するための URL パラメーター」
- 「devfile ファイル名の URL パラメーター」
- 「devfile ファイルパスの URL パラメーター」

2.2.1. URL パラメーターの連結

新しいワークスペースを開始するための URL は、次の URL 構文で **&** を使用することにより、複数の任意の URL パラメーターの連結をサポートします。

https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?<url_parameter_1>&<url_parameter_2>&<url_parameter_3>

> **例2.3 Git リポジトリーの URL と任意の URL パラメーターを使用して新しいワークスペースを開始するための URL**
>
> ブラウザーの完全な URL:
>
> **https://devspaces-<openshift_deployment_name>.<domain_name>#https://github.com/che-samples/cpp-hello-world?new&che-editor=che-incubator/intellij-community/latest&devfilePath=tests/testdevfile.yaml**
>
> URL の部分の説明:

```
https://devspaces-<openshift_deployment_name>.<domain_name> (1)
#https://github.com/che-samples/cpp-hello-world (2)
?new&che-editor=che-incubator/intellij-community/latest&devfilePath=tests/testdevfile.yaml (3)
```

1. OpenShift Dev Spaces URL
2. 新しいワークスペースに複製される Git リポジトリーの URL。
3. 連結された任意の URL パラメーター。

2.2.2. ワークスペース IDE の URL パラメーター

新しいワークスペースを開始するための URL に、統合開発環境 (IDE) を指定する URL パラメーターが含まれていない場合、ワークスペースは既定の IDE である Che Theia で読み込まれます。

サポートされている別の IDE を指定するための URL パラメーターは **che-editor=<editor_key>** です。

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?che-editor=<editor_key>
```

表2.1 サポートされている IDE の URL パラメーター<editor_key> の値

IDE	<editor_key> 値	注記
Che-Theia	**eclipse/che-theia/latest**	これは、URL パラメーター使用せずに新しいワークスペースで読み込むデフォルトの IDE です。
IntelliJ IDEA	**che-incubator/che-idea/latest**	Community Edition - 安定版

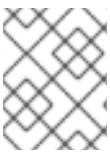
2.2.3. 重複するワークスペースを開始するための URL パラメーター

新しいワークスペースを開始するために URL にアクセスすると、devfile に従って、リンクされた Git リポジトリーのクローンを使用して新しいワークスペースが作成されます。

状況によっては、devfile とリンクされた Git リポジトリーに関して重複する複数のワークスペースが必要になる場合があります。これを行うには、同じ URL にアクセスして、URL パラメーターを使用して新しいワークスペースを開始します。

複製ワークスペースを開始するための URL パラメーターは **new** です。

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?new
```



注記

現在 URL の使用を開始したワークスペースがある場合、**new** URL パラメーターを指定せずに URL に再度アクセスすると、エラーメッセージが表示されます。

2.2.4. devfile ファイル名の URL パラメーター

新しいワークスペースを開始するために URL にアクセスすると、OpenShift Dev Spaces は、リンクされた Git リポジトリーで、ファイル名が **.devfile.yaml** または **devfile.yaml** の devfile を検索します。リンクされた Git リポジトリー内の devfile は、このファイル命名規則に従う必要があります。

状況によっては、devfile に別の型にはまらないファイル名を指定する必要がある場合があります。

devfile の型にはまらないファイル名を指定するための URL パラメーターは **df=<filename>.yaml** です。

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?df=<filename>.yaml (1)
```

(1) **<filename>.yaml** は、リンクされた Git リポジトリー内の devfile の型にはまらないファイル名です。

ヒント

df=<filename>.yaml パラメーターにも長いバージョン (**devfilePath=<filename>.yaml**) があります。

2.2.5. devfile ファイルパスの URL パラメーター

新しいワークスペースを開始するために URL にアクセスすると、OpenShift Dev Spaces は、リンクされた Git リポジトリーの root ディレクトリーで、ファイル名が **.devfile.yaml** または **devfile.yaml** の devfile を検索します。リンクされた Git リポジトリー内の devfile のファイルパスは、このパス規則に従う必要があります。

状況によっては、リンクされた Git リポジトリー内の devfile に別の型にはまらないファイルパスを指定する必要がある場合があります。

devfile の型にはまらないファイルパスを指定するための URL パラメーターは **devfilePath=<relative_file_path>** です。

```
https://devspaces-<openshift_deployment_name>.<domain_name>#<git_repository_url>?devfilePath=<relative_file_path> (1)
```

(1) **<relative_file_path>** は、リンクされた Git リポジトリー内の devfile の型にはまらないファイルパスです。

2.3. ワークスペースで実行できる基本的なアクション

ワークスペースを管理し、OpenShift Dev Spaces ダッシュボードの **Workspaces** ページ (**https://devspaces-<openshift_deployment_name>.<domain_name>/dashboard/#/workspaces**) で現在の状態を確認します。

新しいワークスペースを開始した後、**Workspaces** ページで次のアクションを実行できます。

表2.2 ワークスペースで実行できる基本的なアクション

アクション	ワークスペースページの GUI ステップ
実行中のワークスペースを再度開く	**開く** をクリックします。
実行中のワークスペースを再起動する	⋮ > **Restart Workspace** に移動します。
実行中のワークスペースを停止する	⋮ > **Stop Workspace** に移動します。
停止したワークスペースを開始する	**開く** をクリックします。
ワークスペースを削除する	⋮ > **Delete Workspace** に移動します。

2.4. ワークスペースから GIT サーバーへの認証

ワークスペースでは、リモートのプライベート Git リポジトリーのクローンを作成したり、リモートのパブリックまたはプライベート Git リポジトリーにプッシュしたりするなど、ユーザー認証を必要とする Git コマンドを実行できます。

ワークスペースから Git サーバーへのユーザー認証を設定するために、OpenShift Dev Spaces には 2 つのオプションがあります。

- 管理者は、組織の Red Hat OpenShift Dev Spaces インスタンス用に、 GitHub、GitLab、または Bitbucket で OAuth アプリケーション をセットアップします。
- Git クレデンシャルストア用に独自のユーザー Kubernetes シークレット を作成します。

関連情報

- 管理ガイド: GitHub、GitLab、または Bitbucket の OAuth
- ユーザーガイド: Git クレデンシャルストアの使用

第3章 ワークスペースコンポーネントのカスタマイズ

ワークスペースコンポーネントをカスタマイズするには:

- ワークスペースの Git リポジトリーを選択します。
- 最新の devfile2 仕様を満たす devfile を使用してください。Devfile ユーザーガイド を参照してください。
- ブラウザー内 IDE を選択してカスタマイズします。
- 一般的な devfile 仕様に加えて、OpenShift Dev Spaces 固有の属性を追加できます。

第4章 ワークスペース IDE の選択

新しいワークスペースのデフォルトのブラウザー内 IDE は Che Theia です。

次のいずれかの方法で、サポートされている別のブラウザー内 IDE を選択できます。

- URL にアクセスして新しいワークスペースを開始する場合、URL に **che-editor** パラメーターを追加することで、そのワークスペースの IDE を選択できます。「URL パラメーターを使用して新しいワークスペースのブラウザー内 IDE を選択する」を参照してください。
- Git リポジトリーの **.che/che-editor.yaml** ファイルで、そのリポジトリーのクローンを特徴とするすべての新しいワークスペースに IDE を指定できます。「**che-editor.yaml** を使用して Git リポジトリーのブラウザー内 IDE を指定する」を参照してください。

表4.1 サポートされているブラウザー内 IDE

IDE	id	注記
Che-Theia	**eclipse/che-theia/latest**	これは、URL パラメーター使用せずに新しいワークスペースで読み込むデフォルトの IDE です。
IntelliJ IDEA	**che-incubator/che-idea/latest**	Community Edition - 安定版

4.1. URL パラメーターを使用して新しいワークスペースのブラウザー内 IDE を選択する

新しいワークスペースを開始するときに、好みのブラウザー内 IDE を選択できます。これは最も簡単な方法であり、他のワークスペースや他のユーザーには影響しません。

手順

1. 新しいワークスペースを開始するための URL に、ワークスペース IDE の URL パラメーターを含めます。「ワークスペース IDE の URL パラメーター」 を参照してください。
2. ブラウザーで URL にアクセスします。「Git リポジトリーのクローンを使用して新しいワークスペースを開始」を参照してください。

4.2. CHE-EDITOR.YAML を使用して GIT リポジトリーのブラウザー内 IDE を指定する

4.2.1. OpenShift Dev Spaces エディターファイルを使用して IDE を選択する

che-editor.yaml ファイルを使用して、プロジェクトユーザーのデフォルト IDE を定義します。サポートされている ID のリストについては、新しいワークスペースを開始するための URL のオプションのパラメーター を参照してください。

手順

1. **che-editor.yaml** ファイルをプロジェクトのルートディレクトリーの **.che** フォルダーに配置します。
2. **che-editor.yaml** ファイルで、選択している IDE の ID を指定します。以下に例を示します。

```
id: che-incubator/che-idea/latest
```

関連情報

- ここで サンプルファイルのサンプルを確認してください。
- che-editors.yaml に示されている ID を使用して、デフォルトのプラグインレジストリーから実験的な新しい IDE をロードします。

4.2.2. che-editor.yaml ファイルを使用した IDE 選択のカスタマイズ

che-editor.yaml ファイルにさまざまなディレクティブを追加することで、プロジェクトの特定のニーズに合わせて IDE の選択をさらにカスタマイズできます。これらのカスタマイズオプションには、次のディレクティブが含まれます。

- カスタムプラグインレジストリー
- Web リファレンス
- 埋め込みエディターの定義

4.2.3. IDE 用のカスタムプラグインレジストリーを使用する

OpenShift Dev Spaces プラグインレジストリーにデフォルトリストとは異なる IDE を含めるには、オプションの **registryUrl** ディレクティブを使用します。

手順

- **che-editor.yaml** ファイルにオプションの **registryUrl** ディレクティブを設定します。以下に例を示します。

```
id: eclipse/che-theia/next            # mandatory
registryUrl: https://my-registry.com  # optional
override:                             # optional
  containers:
    - name: theia-ide
      memoryLimit: 1280Mi
```

4.2.4. IDE の Web 参照を使用する

reference ディレクティブを使用して YAML ファイルを指定することにより、IDE の Web 参照を使用します。

手順

- **che-editor.yaml** ファイルに **reference** ディレクティブを設定します。以下に例を示します。

```
reference: https://gist.github.com/.../che-editor.yaml # mandatory
override:                                               # optional
```

```
containers:
  - name: theia-ide
    memoryLimit: 1280Mi
```

4.2.5. IDE に埋め込まれたエディター定義を使用する

標準の IDE 動作では対応できないプロジェクトの特定の要件がある場合は、**inline** ディレクティブを使用してプロジェクト IDE をカスタマイズし、完全な IDE 定義を **che-editor.yaml** ファイルに配置できます。

手順

- **che-editor.yaml** ファイルに **inline** ディレクティブを設定します。以下に例を示します。

```
inline:
  endpoints:
    - name: "theia"
      public: true
      targetPort: 3100
      attributes:
        protocol: http
        type: ide
        secure: true
        cookiesAuthEnabled: true
        discoverable: false
    (...)
  containers:
    - name: theia-ide
      image: "quay.io/eclipse/che-theia:next"
      env:
        - name: THEIA_PLUGINS
          value: local-dir:///plugins
      volumeMounts:
        - name: plugins
          path: "/plugins"
        - name: theia-local
          path: "/home/theia/.theia"
      mountSources: true
      ports:
        - exposedPort: 3100
      memoryLimit: "512M"
      cpuLimit: 1000m
      cpuRequest: 100m
  initContainers:
    - name: remote-runtime-injector
      image: "quay.io/eclipse/che-theia-endpoint-runtime-binary:next"
      volumeMounts:
        - name: remote-endpoint
          path: "/remote-endpoint"
          ephemeral: true
      env:
        - name: PLUGIN_REMOTE_ENDPOINT_EXECUTABLE
          value: /remote-endpoint/plugin-remote-endpoint
        - name: REMOTE_ENDPOINT_VOLUME_NAME
          value: remote-endpoint
```

第5章 ワークスペースでのクレデンシャルと設定の使用

ワークスペースでクレデンシャルと設定を使用できます。

これを行うには、組織の OpenShift Dev Spaces インスタンスの OpenShift クラスター内の **DevWorkspace** コンテナーにクレデンシャルと設定をマウントします。

- クレデンシャルと機密性の高い設定を Kubernetes シークレット としてマウントします。1つの例は、Git クレデンシャルストア です。
- 機密性のない設定を Kubernetes ConfigMaps としてマウントします。

クラスター内の **DevWorkspace** Pod が認証を必要とするコンテナーレジストリーにアクセスできるようにする必要がある場合は、**DevWorkspace** Pod の イメージプルシークレット を作成します。

マウントプロセスでは、標準の Kubernetes マウントメカニズムを使用し、既存のリソースに追加のラベルとアノテーションを適用する必要があります。新しいワークスペースを開始するとき、または既存のワークスペースを再起動するときに、リソースがマウントされます。

さまざまなコンポーネントの永続的なマウントポイントを作成できます。

- **settings.xml** ファイルなどの Maven 設定
- SSH キーペア
- AWS 認証トークン
- 設定ファイル
- 永続ストレージ
- Git 認証情報ストアファイル

関連情報

- Kubernetes ドキュメント: シークレット
- Kubernetes ドキュメント: ConfigMaps

5.1. GIT クレデンシャルストアの使用

組織の OpenShift Dev Spaces インスタンスの管理者によって設定された GitHub、GitLab、または Bitbucket の OAuth の代わりに、Git クレデンシャルストアを Kubernetes シークレットとして適用できます。

組織の OpenShift Dev Spaces インスタンスの OpenShift クラスターのユーザープロジェクトに Kubernetes シークレットを適用します。

シークレットを適用すると、マウントされた Git クレデンシャルストアへのパスを含む Git 設定ファイルが自動的に設定され、クラスター内の **/etc/gitconfig** にある **DevWorkspace** コンテナーにマウントされます。これにより、Git クレデンシャルストアをワークスペースで利用できるようになります。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the CLI を参照。

- base64 コマンドラインツールは、使用しているオペレーティングシステムにインストールされている。

手順

1. すでにある場合は、ホームディレクトリーで **.git-credentials** ファイルを見つけて開きます。または、このファイルがない場合は、Git クレデンシャルストレージ形式 を使用して、新しい **.git-credentials** ファイルを保存します。各クレデンシャルは、ファイル内の独自の行に保存されます。

```
https://<username>:<token>@<git_server_hostname>
```

例5.1.git-credentials ファイルの行

```
https://trailblazer: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@github.com
```

2. シークレットの **.git-credentials** ファイルからクレデンシャルを選択します。次の手順のために、選択したクレデンシャルを Base64 にエンコードします。

ヒント

- ファイル内のすべての行をエンコードするには:
  $ cat .git-credentials | base64 | tr -d '\n'
- 選択した行をエンコードするには:
  $ echo -n '<copied_and_pasted_line_from_.git-credentials>' | base64

3. ユーザープロジェクトに新しい OpenShift シークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: git-credentials-secret
  labels:
    controller.devfile.io/git-credential: 'true' (1)
    controller.devfile.io/watch-secret: 'true'
  annotations:
    controller.devfile.io/mount-path: /etc/secret (2)
data:
  credentials: <Base64_content_of_.git-credentials> (3)
```

(1) **controller.devfile.io/git-credential** ラベルは、シークレットに Git クレデンシャルが含まれていることを示します。

(2) **DevWorkspace** コンテナーのカスタム絶対パス。シークレットは、このパスに **credentials** ファイルとしてマウントされます。デフォルトのパスは / です。

(3) 前の手順で Base64 にエンコードした **.git-credentials** から選択したコンテンツ。

ヒント

ユーザープロジェクトで複数の Git クレデンシャルシークレットを作成して適用できます。それらはすべて、**DevWorkspace** コンテナーにマウントされる 1 つのシークレットにコピーされます。たとえば、マウントパスを **/etc/secret** に設定すると、すべての Git クレデンシャルを持つ 1 つのシークレットが **/etc/secret/credentials** にマウントされます。ユーザープロジェクトのすべての Git クレデンシャルシークレットを同じマウントパスに設定する必要があります。マウントパスは **/etc/gitconfig** で設定された Git 設定ファイルで自動的に設定されるため、マウントパスを任意のパスに設定できます。

4. シークレットを適用します。

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

5.2. 制限された環境でのアーティファクトリーポジトリーの有効化

テクノロジースタックを設定することで、自己署名証明書を使用して、インハウスリポジトリーからアーティファクトを扱うことができます。

- Maven
- Gradle
- npm
- Python
- Go
- NuGet

5.2.1. Maven アーティファクトリーポジトリの有効化

制限された環境で実行される Maven ワークスペースで Maven アーティファクトリーポジトリを有効にできます。

前提条件

- Maven ワークスペースを実行していない。

手順

1. TLS 証明書のシークレットを適用します。

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
```

```
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> (1)
```

(1) 行の折り返しが無効になっている Base64 エンコーディング。

2. ConfigMap を適用して **settings.xml** ファイルを作成します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: settings-xml
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /home/user/.m2
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  settings.xml: |
    <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
https://maven.apache.org/xsd/settings-1.0.0.xsd">
      <localRepository/>
      <interactiveMode/>
      <offline/>
      <pluginGroups/>
      <servers/>
      <mirrors>
        <mirror>
          <id>redhat-ga-mirror</id>
          <name>Red Hat GA</name>
          <url>https://<maven_artifact_repository_route>/repository/redhat-ga/</url>
          <mirrorOf>redhat-ga</mirrorOf>
        </mirror>
        <mirror>
          <id>maven-central-mirror</id>
          <name>Maven Central</name>
          <url>https://<maven_artifact_repository_route>/repository/maven-central/</url>
          <mirrorOf>maven-central</mirrorOf>
        </mirror>
        <mirror>
          <id>jboss-public-repository-mirror</id>
          <name>JBoss Public Maven Repository</name>
          <url>https://<maven_artifact_repository_route>/repository/jboss-public/</url>
          <mirrorOf>jboss-public-repository</mirrorOf>
        </mirror>
      </mirrors>
      <proxies/>
      <profiles/>
      <activeProfiles/>
    </settings>
```

3. TrustStore 初期化スクリプトに ConfigMap を適用します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: init-truststore
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /home/user/
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  init-truststore.sh: |
    #!/usr/bin/env bash

    keytool -importcert -noprompt -file /home/user/certs/tls.cer -cacerts -storepass changeit
```

4. Maven ワークスペースを開始します。

5. **tools** コンテナーで新しいターミナルを開きます。

6. ~/**init-truststore.sh** を実行します。

5.2.2. Gradle アーティファクトリーポジトリの有効化

制限された環境で実行される Gradle ワークスペースで Gradle アーティファクトリーポジトリを有効にできます。

前提条件

- Gradle ワークスペースを実行していない。

手順

1. TLS 証明書のシークレットを適用します。

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> (1)
```

(1) 行の折り返しが無効になっている Base64 エンコーディング。

2. TrustStore 初期化スクリプトに ConfigMap を適用します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: init-truststore
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /home/user/
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  init-truststore.sh: |
    #!/usr/bin/env bash

    keytool -importcert -noprompt -file /home/user/certs/tls.cer -cacerts -storepass changeit
```

3. Gradleinit スクリプトに ConfigMap を適用します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: init-gradle
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /home/user/.gradle
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  init.gradle: |
    allprojects {
      repositories {
        mavenLocal ()
        maven {
          url "https://<gradle_artifact_repository_route>/repository/maven-public/"
          credentials {
            username "admin"
            password "passwd"
          }
        }
      }
    }
```

4. Gradle ワークスペースを開始します。
5. **tools** コンテナーで新しいターミナルを開きます。
6. **~/init-truststore.sh** を実行します。

5.2.3. npm アーティファクトリーポジトリの有効化

制限された環境で実行される npm ワークスペースで npm アーティファクトリーポジトリを有効にできます。

前提条件

- npm ワークスペースを実行していない。

警告

環境変数を設定する ConfigMap を適用すると、ワークスペースのブートループが発生する可能性があります。

この動作が発生した場合は、**ConfigMap** を削除し、devfile を直接編集してください。

手順

1. TLS 証明書のシークレットを適用します。

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> (1)
```

(1) 行の折り返しが無効になっている Base64 エンコーディング。

2. ConfigMap を適用して、**tools** コンテナーに次の環境変数を設定します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  NODE_EXTRA_CA_CERTS: /home/user/certs/tls.cer
  NPM_CONFIG_REGISTRY: >-
    https://<npm_artifact_repository_route>/repository/npm-all/
```

5.2.4. Python アーティファクトリーポジトリの有効化

制限された環境で実行される Python ワークスペースで Python アーティファクトリーポジトリを有効にできます。

前提条件

- Python ワークスペースを実行していない。

警告

環境変数を設定する ConfigMap を適用すると、ワークスペースのブートループが発生する可能性があります。

この動作が発生した場合は、**ConfigMap** を削除し、devfile を直接編集してください。

手順

1. TLS 証明書のシークレットを適用します。

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> (1)
```

(1) 行の折り返しが無効になっている Base64 エンコーディング。

2. ConfigMap を適用して、**tools** コンテナーに次の環境変数を設定します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
```

```
PIP_INDEX_URL: >-
  https://<python_artifact_repository_route>/repository/pypi-all/
PIP_CERT: /home/user/certs/tls.cer
```

5.2.5. Go アーティファクトリーポジトリの有効化

制限された環境で実行される Go ワークスペースで Go アーティファクトリーポジトリを有効にできます。

前提条件

- Go ワークスペースを実行していない。

警告

環境変数を設定する ConfigMap を適用すると、ワークスペースのブートループが発生する可能性があります。

この動作が発生した場合は、**ConfigMap** を削除し、devfile を直接編集してください。

手順

1. TLS 証明書のシークレットを適用します。

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> (1)
```

(1) 行の折り返しが無効になっている Base64 エンコーディング。

2. ConfigMap を適用して、**tools** コンテナーに次の環境変数を設定します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
```

```
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  GOPROXY: >-
    http://<athens_proxy_route>
  SSL_CERT_FILE: /home/user/certs/tls.cer
```

5.2.6. NuGet アーティファクトリーポジトリの有効化

制限された環境で実行される NuGet ワークスペースで NuGet アーティファクトリーポジトリを有効にできます。

前提条件

- NuGet ワークスペースを実行していない。

警告

環境変数を設定する ConfigMap を適用すると、ワークスペースのブートループが発生する可能性があります。

この動作が発生した場合は、**ConfigMap** を削除し、devfile を直接編集してください。

手順

1. TLS 証明書のシークレットを適用します。

```
kind: Secret
apiVersion: v1
metadata:
  name: tls-cer
  annotations:
    controller.devfile.io/mount-path: /home/user/certs
    controller.devfile.io/mount-as: file
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  tls.cer: >-
    <Base64_encoded_content_of_public_cert> (1)
```

(1) 行の折り返しが無効になっている Base64 エンコーディング。

2. ConfigMap を適用して、**tools** コンテナー内の TLS 証明書ファイルのパスの環境変数を設定します。

```
kind: ConfigMap
```

```
apiVersion: v1
metadata:
  name: disconnected-env
  annotations:
    controller.devfile.io/mount-as: env
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  SSL_CERT_FILE: /home/user/certs/tls.cer
```

3. ConfigMap を適用して、**nuget.config** ファイルを作成します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: init-nuget
  annotations:
    controller.devfile.io/mount-as: subpath
    controller.devfile.io/mount-path: /projects
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
data:
  nuget.config: |
    <?xml version="1.0" encoding="UTF-8"?>
    <configuration>
      <packageSources>
        <add key="nexus2"
value="https://<nuget_artifact_repository_route>/repository/nuget-group/"/>
      </packageSources>
      <packageSourceCredentials>
        <nexus2>
          <add key="Username" value="admin" />
          <add key="Password" value="passwd" />
        </nexus2>
      </packageSourceCredentials>
    </configuration>
```

5.3. イメージプルシークレットの作成

組織の OpenShift Dev Spaces インスタンスの OpenShift クラスター内の **DevWorkspace** Pod が、認証を必要とするコンテナーレジストリーにアクセスできるようにするには、イメージプルシークレットを作成します。

oc、**.dockercfg** ファイル、または **config.json** ファイルを使用して、イメージプルシークレットを作成できます。

5.3.1. oc でシークレットをプルするイメージを作成する

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the CLI を参照。

手順

1. ユーザープロジェクトで、プライベートコンテナーレジストリーの詳細とクレデンシャルを使用してイメージプルシークレットを作成します。

```
$ oc create secret docker-registry <Secret_name> \
    --docker-server=<registry_server> \
    --docker-username=<username> \
    --docker-password=<password> \
    --docker-email=<email_address>
```

2. 次のラベルをイメージプルシークレットに追加します。

```
$ oc label secret <Secret_name> controller.devfile.io/devworkspace_pullsecret=true
controller.devfile.io/watch-secret=true
```

5.3.2. .dockercfg ファイルからイメージプルシークレットを作成する

プライベートコンテナーレジストリーのクレデンシャルを **.dockercfg** ファイルにすでに保存している場合は、そのファイルを使用してイメージプルシークレットを作成できます。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the CLI を参照。
- **base64** コマンドラインツールは、使用しているオペレーティングシステムにインストールされている。

手順

1. **.dockercfg** ファイルを Base64 にエンコードします。

```
$ cat .dockercfg | base64 | tr -d '\n'
```

2. ユーザープロジェクトに新しい OpenShift シークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <Secret_name>
  labels:
    controller.devfile.io/devworkspace_pullsecret: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  .dockercfg: <Base64_content_of_.dockercfg>
type: kubernetes.io/dockercfg
```

3. シークレットを適用します。

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

5.3.3. config.json ファイルからイメージプルシークレットを作成する

プライベートコンテナーレジストリーのクレデンシャルを **$HOME/.docker/config.json** ファイルに既に保存している場合は、そのファイルを使用してイメージプルシークレットを作成できます。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the CLI を参照。
- **base64** コマンドラインツールは、使用しているオペレーティングシステムにインストールされている。

手順

1. **$HOME/.docker/config.json** ファイルを Base64 にエンコードします。

```
$ cat config.json | base64 | tr -d '\n'
```

2. ユーザープロジェクトに新しい OpenShift シークレットを作成します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <Secret_name>
  labels:
    controller.devfile.io/devworkspace_pullsecret: 'true'
    controller.devfile.io/watch-secret: 'true'
data:
  .dockerconfigjson: <Base64_content_of_config.json>
type: kubernetes.io/dockerconfigjson
```

3. シークレットを適用します。

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

5.4. シークレットのマウント

機密データをワークスペースにマウントするには、Kubernetes シークレットを使用します。

Kubernetes Secrets を使用すると、ユーザー名、パスワード、SSH キーペア、認証トークン (AWS など)、および機密性の高い設定をマウントできます。

組織の OpenShift Dev Spaces インスタンスの OpenShift クラスター内の **DevWorkspace** コンテナーに Kubernetes シークレットをマウントします。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the CLI を参照。

- すべての **DevWorkspace** コンテナーにマウントするために、ユーザープロジェクトに新しいシークレットを作成するか、既存のシークレットを決定した。

手順

1. すべてのワークスペースコンテナーにマウントするユーザープロジェクトの既存の ConfigMap またはシークレットを決定します。
2. 取り付けに必要なラベルを設定します。

```
$ oc label secret <Secret_name> \
      controller.devfile.io/mount-to-devworkspace=true \
      controller.devfile.io/watch-secret=true
```

3. オプション: アノテーションを使用して、シークレットのマウント方法を設定します。

表5.1 オプションのアノテーション

Annotation	説明
controller.devfile.io/mount-path:	マウントパスを指定します。 デフォルトは **/etc/secret/<Secret_name>** です。
controller.devfile.io/mount-as:	リソースのマウント方法を指定します: **file**、**subpath**、または **env**。 デフォルトは **file** です。 **mount-as: file** は、キーと値をマウントパス内のファイルとしてマウントします。 **mount-as: subpath** は、サブパスボリュームマウントを使用して、マウントパス内のキーと値をマウントします。 **mount-as: env** は、すべての **DevWorkspace** コンテナーに環境変数としてキーと値をマウントします。

例5.2 シークレットをファイルとしてマウントする

```
apiVersion: v1
kind: Secret
metadata:
  name: mvn-settings-secret
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-secret: 'true'
  annotations:
    controller.devfile.io/mount-path: '/home/user/.m2'
data:
  settings.xml: <Base64_encoded_content>
```

ワークスペースを開始すると、**/home/user/.m2/settings.xml** ファイルが **DevWorkspace** コンテナーで使用可能になります。

Maven を使用すると、**settings.xml** ファイルのカスタムパスを設定できます。以下に例を示します。

```
$ mvn --settings /home/user/.m2/settings.xml clean install
```

5.5. CONFIGMAP のマウント

機密でない設定データをワークスペースにマウントするには、Kubernetes ConfigMaps を使用します。

Kubernetes ConfigMaps を使用すると、アプリケーションの設定値などの機密性の低いデータをマウントできます。

Kubernetes ConfigMaps を組織の OpenShift Dev Spaces インスタンスの OpenShift クラスター内の **DevWorkspace** コンテナーにマウントします。

前提条件

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the CLI を参照。
- 新しい ConfigMap を作成するか、ユーザープロジェクトに既存の ConfigMap を決定して、すべての **Dev Workspace** コンテナーにマウントしました。

手順

1. すべてのワークスペースコンテナーにマウントするユーザープロジェクトの既存の ConfigMap を決定します。
2. 取り付けに必要なラベルを設定します。

```
$ oc label configmap <ConfigMap_name> \
      controller.devfile.io/mount-to-devworkspace=true \
      controller.devfile.io/watch-configmap=true
```

3. オプション: アノテーションを使用して、ConfigMap のマウント方法を設定します。

表5.2 オプションのアノテーション

Annotation	説明
controller.devfile.io/mount-path:	マウントパスを指定します。 デフォルトは **/etc/config/<ConfigMap_name>** です。

Annotation	説明
controller.devfile.io/mount-as:	リソースのマウント方法を指定します: **file**、**subpath**、または **env**。 デフォルトは **file** です。 **mount-as:file** は、キーと値をマウントパス内のファイルとしてマウントします。 **mount-as:subpath** は、サブパスボリュームマウントを使用して、マウントパス内のキーと値をマウントします。 **mount-as:env** は、すべての **DevWorkspace** コンテナーに環境変数としてキーと値をマウントします。

例5.3 ConfigMap を環境変数としてマウントする

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: my-settings
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
    controller.devfile.io/watch-configmap: 'true'
  annotations:
    controller.devfile.io/mount-as: env
data:
  <env_var_1>: <value_1>
  <env_var_2>: <value_2>
```

ワークスペースを開始すると、**<env_var_1>** および **<env_var_2>** 環境変数が **DevWorkspace** コンテナーで使用可能になります。

第6章 ワークスペースの永続ストレージを要求する

OpenShift Dev Spaces ワークスペースとワークスペースデータは一時的なものであり、ワークスペースが停止すると失われます。

ワークスペースが停止している間、ワークスペースの状態を永続ストレージに保持するには、組織の OpenShift Dev Spaces インスタンスの OpenShift クラスター内の **DevWorkspace** コンテナーに対して Kubernetes 永続ボリューム (PV) をリクエストします。

devfile または Kubernetes PersistentVolumeClaim (PVC) を使用して PV をリクエストできます。

PV の例は、ワークスペースの **/projects/** ディレクトリーです。これは、non-ephemeral ワークスペース用にデフォルトでマウントされます。

永続ボリュームにはコストがかかります。永続ボリュームを接続すると、ワークスペースの起動が遅くなります。

警告

別のワークスペースを **ReadWriteOnce** PV で同時に実行すると、失敗する可能性があります。

関連情報

- Red Hat OpenShift ドキュメント: 永続ストレージを理解する
- Kubernetes ドキュメント: 永続ボリューム

6.1. DEVFILE での永続ストレージのリクエスト

ワークスペースに独自の永続ストレージが必要な場合は、devfile で PersistentVolume (PV) をリクエストすると、OpenShift Dev Spaces が必要な PersistentVolumeClaims を自動的に管理します。

前提条件

- ワークスペースを開始していない。

手順

1. devfile に **volume** コンポーネントを追加します。

```
...
components:
  ...
  - name: <chosen_volume_name>
    volume:
      size: <requested_volume_size>G
  ...
```

2. devfile に該当する **container** の **volumeMount** を追加する。

```
...
components:
  - name: ...
    container:
      ...
      volumeMounts:
        - name: <chosen_volume_name_from_previous_step>
          path: <path_where_to_mount_the_PV>
      ...
```

例6.1 ワークスペースの PV をコンテナーにプロビジョニングする devfile

ワークスペースが次の devfile で開始されると、**cache** PV は **./cache** コンテナーパスの **golang** コンテナーにプロビジョニングされます。

```
schemaVersion: 2.1.0
metadata:
  name: mydevfile
components:
  - name: golang
    container:
      image: golang
      memoryLimit: 512Mi
      mountSources: true
      command: ['sleep', 'infinity']
      volumeMounts:
        - name: cache
          path: /.cache
  - name: cache
    volume:
      size: 2Gi
```

6.2. PVC での永続ストレージの要求

次の場合は、PersistentVolumeClaim (PVC) を適用して、ワークスペースの PersistentVolume (PV) を要求することができます。

- プロジェクトのすべての開発者が PV を必要とするわけではありません。
- PV のライフサイクルは、単一のワークスペースのライフサイクルを超えています。
- PV に含まれるデータは、ワークスペース間で共有されます。

ヒント

ワークスペースがエフェメラルであり、その devfile に **controller.devfile.io/storage-type: ephemeral** 属性が含まれている場合でも、PVC を **DevWorkspace** コンテナーに適用できます。

前提条件

- ワークスペースを開始していない。

- 宛先 OpenShift クラスターへの管理権限を持つアクティブな **oc** セッション。Getting started with the CLI を参照。
- すべての **DevWorkspace** コンテナーにマウントするために、ユーザープロジェクトに PVC が作成されます。

手順

1. **controller.devfile.io/mount-to-devworkspace: true** ラベルを PVC に追加します。

```
$ oc label persistentvolumeclaim <PVC_name> \ controller.devfile.io/mount-to-devworkspace=true
```

2. オプション: アノテーションを使用して、PVC のマウント方法を設定します。

表6.1 オプションのアノテーション

Annotation	説明
controller.devfile.io/mount-path:	PVC のマウントパス。 デフォルトは **/tmp/<PVC_name>** です。
controller.devfile.io/read-only:	**'true'** または **'false'** に設定して、PVC を読み取り専用としてマウントするかどうかを指定します。 デフォルトは **'false'** で、PVC は読み取り/書き込みとしてマウントされます。

例6.2 読み取り専用 PVC のマウント

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: <pvc_name>
  labels:
    controller.devfile.io/mount-to-devworkspace: 'true'
  annotations:
    controller.devfile.io/mount-path: </example/directory> (1)
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi (2)
  volumeName: <pv_name>
  storageClassName: manual
  volumeMode: Filesystem
```

(1) マウントされた PV は、ワークスペースの **</example/directory>** にあります。

(2) 要求されたストレージのサイズ値の例。

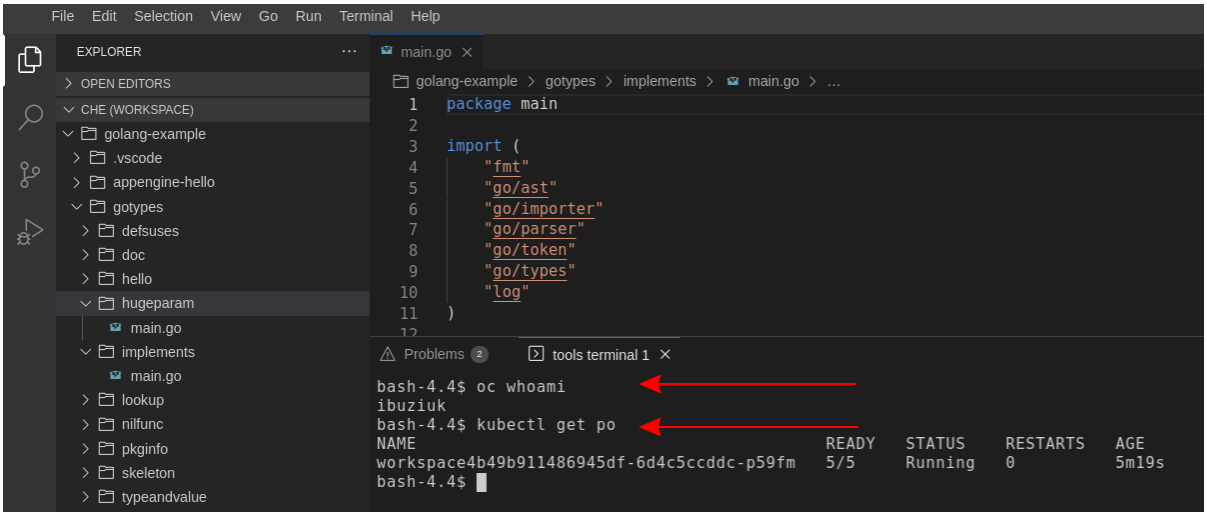
第7章 OPENSHIFT との統合

7.1. 自動 OPENSHIFT トークン注入

このセクションでは、OpenShift クラスターに対して OpenShift Dev Spaces CLI コマンドを実行できるようにするワークスペースコンテナーに自動的に挿入される OpenShift ユーザートークンの使用方法を説明します。

手順

1. OpenShift Dev Spaces ダッシュボードを開き、ワークスペースを開始します。
2. ワークスペースが開始されたら、OpenShift Dev Spaces CLI を含むコンテナーでターミナルを開きます。
3. OpenShift クラスターに対してコマンドを実行できる OpenShift Dev Spaces CLI コマンドを実行します。CLI は、アプリケーションのデプロイ、クラスターリソースの検査および管理、ならびにログの表示に使用できます。OpenShift ユーザートークンは、コマンドの実行中に使用されます。



警告

自動トークン注入は現在、OpenShift インフラストラクチャーでのみ機能します。

7.2. OPENSHIFT 開発者の観点からの OPENSHIFT DEV SPACES のナビゲート

OpenShift Container Platform Web コンソールは、**Administrator** パースペクティブと **Developer** パースペクティブという 2 つのパースペクティブを提供します。

Developer パースペクティブは、以下を実行する機能などの、開発者のユースケースに固有のワークフローを提供します。

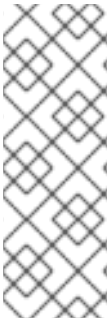
- 既存のコードベース、イメージ、および Dockerfile をインポートして、OpenShift Container Platform でアプリケーションを作成し、デプロイします。
- アプリケーション、コンポーネント、およびプロジェクト内のこれらに関連付けられたサービスと視覚的に対話し、それらのデプロイメントとビルドステータスを監視します。
- アプリケーション内のコンポーネントをグループ化し、アプリケーション内およびアプリケーション間でコンポーネントを接続します。
- サーバーレス機能 (テクノロジープレビュー) を統合します。
- OpenShift Dev Spaces を使用して、アプリケーションコードを編集するためのワークスペースを作成します。

7.2.1. OpenShift Developer Perspective と OpenShift Dev Spaces の統合

このセクションでは、OpenShift Dev Spaces の OpenShift Developer Perspective サポートに関する情報を提供します。

OpenShift Dev Spaces Operator が OpenShift Container Platform 4.2 以降にデプロイされると、**ConsoleLink** カスタムリソース (CR) が作成されます。これにより、OpenShift Developer パースペクティブコンソールを使用して OpenShift Dev Spaces インストールにアクセスするための **Red Hat Applications** メニューへの対話的なリンクが追加されます。

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



注記

OpenShift Dev Spaces が HTTP リソースで使用されている場合、OpenShift Container Platform コンソールリンクは作成されません

FromGit オプションを使用して OpenShift Dev Spaces をインストールすると、OpenShift Developer Perspective コンソールリンクは、OpenShift Dev Spaces が HTTPS でデプロイされている場合にのみ作成されます。HTTP リソースが使用されている場合、コンソールリンクは作成されません。

7.2.2. OpenShift Dev Spaces を使用して OpenShift Container Platform で実行しているアプリケーションコードの編集

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

前提条件

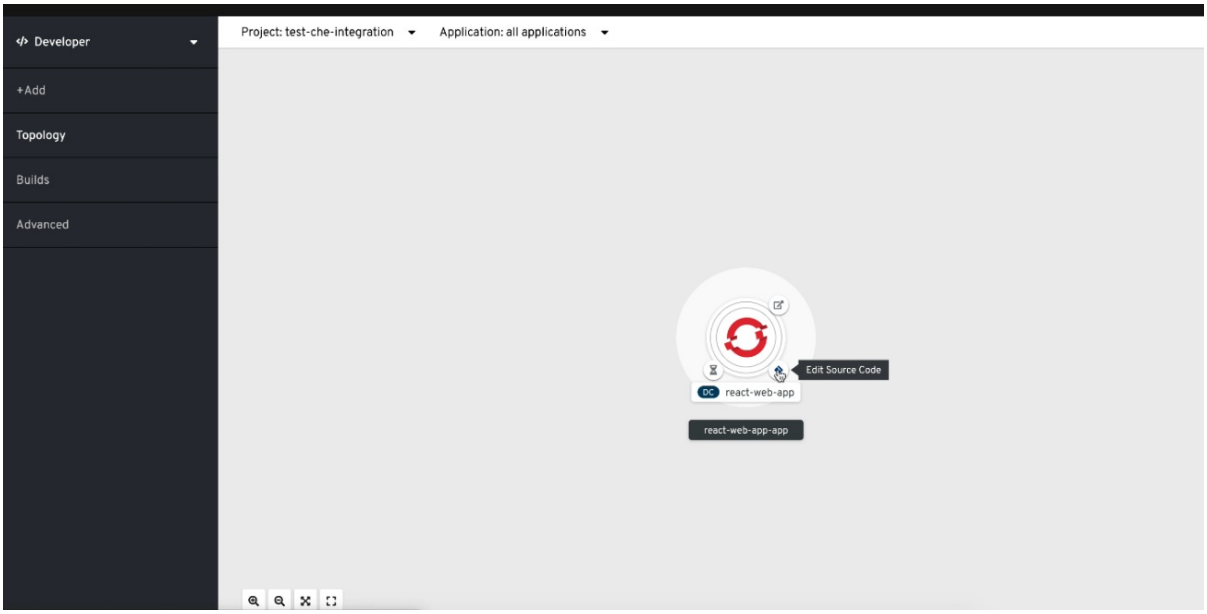
- OpenShift Dev Spaces は、同じ OpenShift4 クラスターにデプロイされる。

手順

1. **Topology** ビューを開き、すべてのプロジェクトを一覧表示します。

2. **Select an Application** 検索フィールドに **workspace** と入力してすべてのワークスペースを一覧表示します。

3. ワークスペースをクリックして編集します。
デプロイメントは、円形のボタンで囲まれたグラフィカルな円で表示されます。これらのボタンの 1 つは **Edit Source Code** です。



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

7.2.3. Red Hat アプリケーションメニューから OpenShift Dev Spaces にアクセス

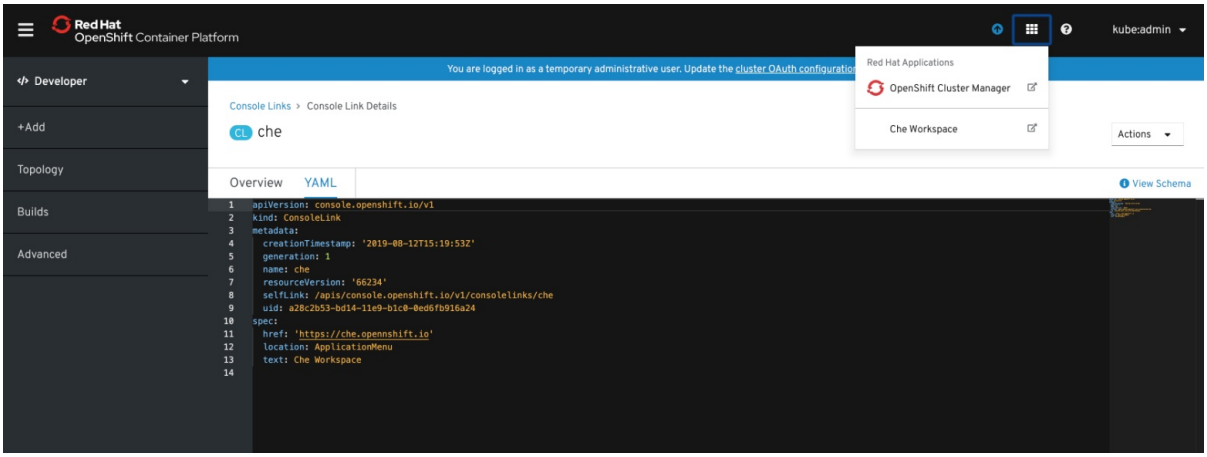
このセクションでは、OpenShift Container Platform の **Red Hat Applications** メニューから OpenShift Dev Spaces ワークスペースにアクセスする方法を説明します。

前提条件

- OpenShift Dev Spaces Operator は OpenShift4 で使用できる。

手順

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



2. **OpenShift Dev Spaces** リンクをクリックして、Dev Spaces ダッシュボードを開きます。

7.3. OPENSHIFT DEV SPACES からの OPENSHIFT WEB コンソールのナビゲート

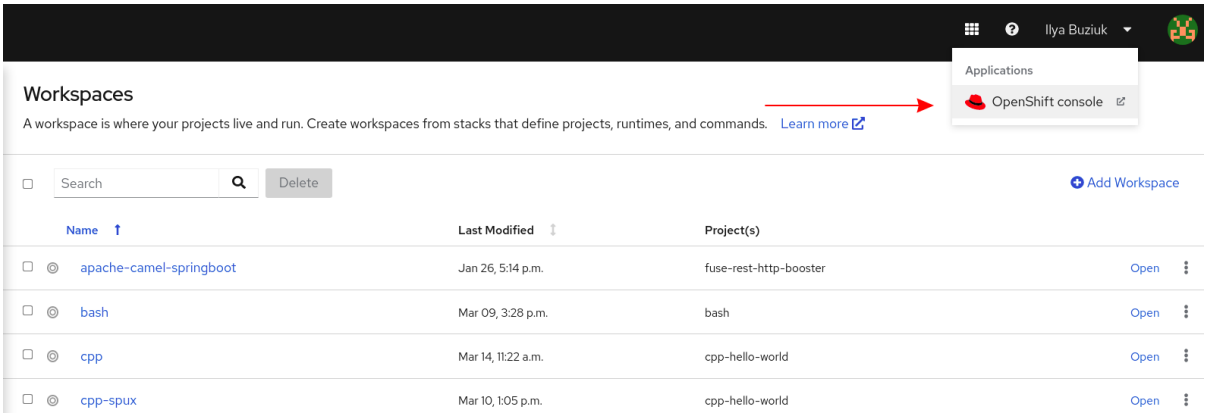
このセクションでは、OpenShift Dev Spaces から OpenShift Web コンソールにアクセスする方法を説明します。

前提条件

- OpenShift Dev Spaces Operator は OpenShift4 で使用できる。

手順

1. OpenShift Dev Spaces ダッシュボードを開き、メイン画面の右上隅にある 3 行 3 列のマトリックスアイコンをクリックします。
ドロップダウンメニューには、利用可能なアプリケーションが表示されます。



2. **OpenShift コンソール** のリンクをクリックして、OpenShift Web コンソールを開きます。

第8章 OPENSHIFT DEV SPACES のトラブルシューティング

本セクションでは、ユーザーが競合する可能性のある最も頻繁に発生する問題についてのトラブルシューティング手順を説明します。

関連情報

- 「OpenShift Dev Spaces ワークスペースログの表示」
- 「Verbose モードを使用したワークスペースの開始時の障害の調査」
- 「速度の遅いワークスペースのトラブルシューティング」
- 「ネットワーク問題のトラブルシューティング」

8.1. OPENSHIFT DEV SPACES ワークスペースログの表示

このセクションでは、OpenShift Dev Spaces ワークスペースのログを表示する方法を説明します。

8.1.1. 言語サーバーおよびデバッグアダプターのログの表示

8.1.1.1. 重要なログの確認

本セクションでは、重要なログを確認する方法を説明します。

手順

1. OpenShift web コンソールで、**Applications → Pods** をクリックし、すべてのアクティブなワークスペースの一覧を表示します。
2. ワークスペースが実行されている実行中の Pod の名前をクリックします。Pod 画面には、追加情報が含まれるすべてのコンテナーの一覧が含まれます。
3. コンテナーを選択し、コンテナー名をクリックします。

注記

最も重要なログは、**theia-ide** コンテナーとプラグインコンテナーログです。

4. コンテナー画面で、**Logs** セクションに移動します。

8.1.1.2. メモリー問題の検出

本セクションでは、メモリー不足のプラグインに関連するメモリーの問題を検出する方法を説明します。以下は、メモリー不足のプラグインに関連する 2 つの最もよく見られる問題になります。

プラグインコンテナーがメモリー不足になる

これは、コンテナーにイメージのエントリーポイントを実行するのに十分な RAM がない場合に、プラグインの初期化時に発生する可能性があります。ユーザーはプラグインコンテナーのログでこれを検知できます。この場合、ログには **OOMKilled** が含まれます。これは、コンテナーのプロセスがコンテナーで利用可能な量よりも多くのメモリーを要求することを示唆します。

コンテナー内のプロセスがコンテナーの通知なしにメモリー不足になる

たとえば、Java 言語サーバー (**vscode-java** 拡張によって起動する Eclipse JDT Language Server) は **OutOfMemoryException** を出力します。これは、プラグインが言語サーバーを起動する際、または処理する必要があるプロジェクトのサイズによりプロセスがメモリー不足になる場合などに、コンテナーの初期化後いつでも発生する可能性があります。
この問題を検出するには、コンテナーで実行しているプライマリープロセスのログを確認します。たとえば、Eclipse JDT Language Server のログファイルで詳細を確認するには、関連するプラグイン固有のセクションを参照してください。

8.1.1.3. デバッグアダプター用のクライアントサーバーのトラフィックのロギング

本セクションでは、Che-Theia とデバッグアダプター間のやり取りのログを **Output** ビューに記録する方法を説明します。

前提条件

- **Debug アダプター** のオプションを一覧に表示するには、デバッグセッションを開始する必要がある。

手順

1. **File → Settings**、**Preferences** をクリックします。
2. **Preferences** ビューの **Debug** セクションを展開します。
3. **trace** 設定の値を **true** に設定します (デフォルトは **false** です)。
すべての通信イベントがログに記録されます。
4. これらのイベントを確認するには、**View → Output** をクリックし、**Output** ビューの右上にあるドロップダウンリストから **Debug adapters** を選択します。

8.1.1.4. Python のログの表示

本セクションでは、Python 言語サーバーのログを表示する方法を説明します。

手順

- **Output** view ビューに移動し、ドロップダウンリストで **Python** を選択します。

8.1.1.5. Go のログの表示

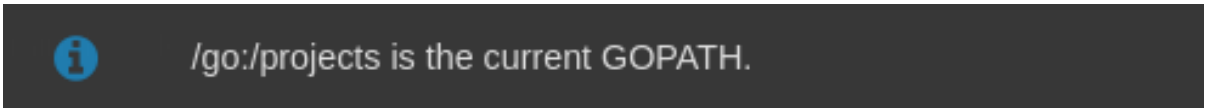
本セクションでは、Go 言語サーバーのログを表示する方法を説明します。

8.1.1.5.1. Go パスの検索

本セクションでは、**GOPATH** 変数が参照する場所を見つける方法を説明します。

手順

- **Go: Current GOPATH** コマンドを実行します。



8.1.1.5.2. Go の Debug Console ログの表示

本セクションでは、Go デバッガーからのログ出力を表示する方法を説明します。

手順

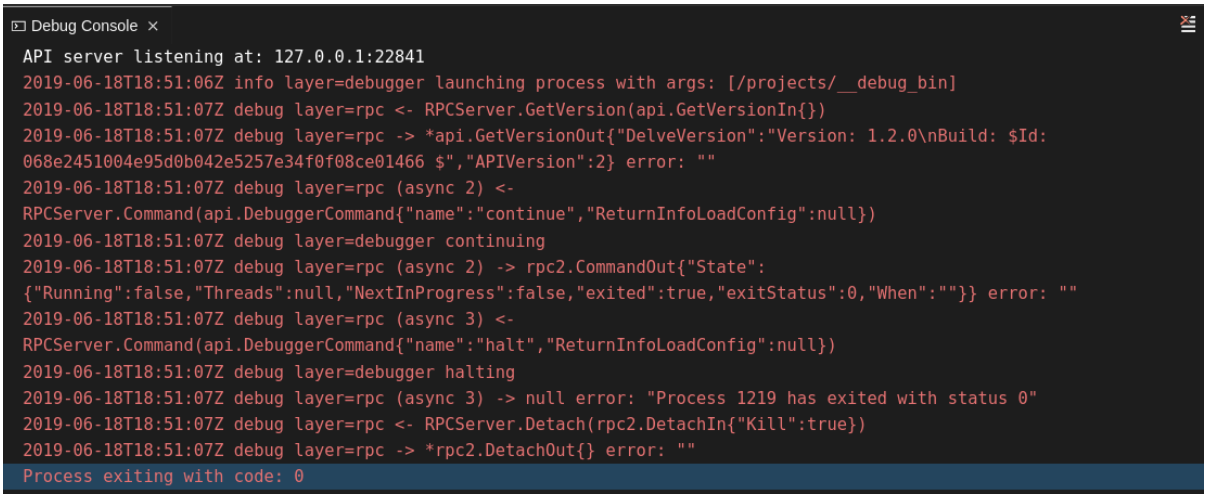
1. デバッグ設定で **showLog** 属性を **true** に設定します。

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
      "showLog": true
      ....
    }
  ]
}
```

2. コンポーネントのデバッグ出力を有効にするには、パッケージを **logOutput** 属性のコンマ区切りの一覧に追加します。

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "go",
      "showLog": true,
      "logOutput": "debugger,rpc,gdbwire,lldbout,debuglineerr"
      ....
    }
  ]
}
```

3. デバッグコンソールは、デバッグコンソールに追加情報を出力します。



8.1.1.5.3. Output パネルでの Go ログ出力の表示

本セクションでは、**Output** パネルで Go ログ出力を表示する方法を説明します。

手順

1. **Output** ビューに移動します。
2. ドロップダウンリストで **Go** を選択します。

8.1.1.6. NodeDebug NodeDebug2 アダプターのログの表示

注記

一般的な診断以外の特定の診断はありません。

8.1.1.7. Typescript のログの表示

8.1.1.7.1. Label Switched Protocol (LSP) トレースの有効化

手順

1. Typescript (TS) サーバーに送信されるメッセージのトレースを有効にするには、**Preferences** ビューで **typescript.tsserver.trace** 属性を **verbose** に設定します。これを使用して、TS サーバーの問題を診断します。
2. TS サーバーのファイルへのロギングを有効にするには、**typescript.tsserver.log** 属性を **verbose** に設定します。このログを使用して、TS サーバーの問題を診断します。ログにはファイルパスが含まれます。

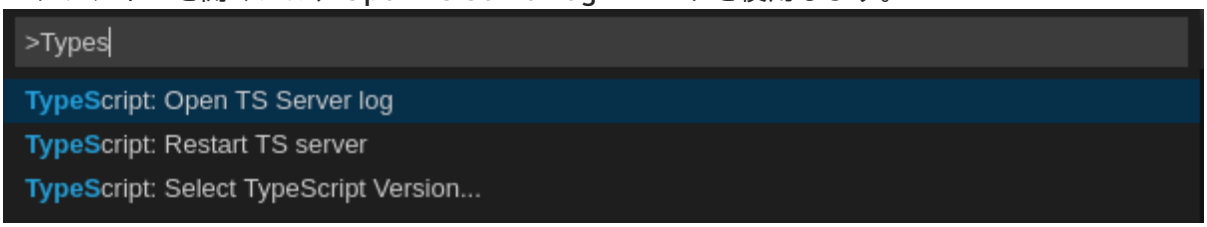
8.1.1.7.2. Typescript 言語サーバーログの表示

本セクションでは、Typescript 言語サーバーのログを表示する方法を説明します。

手順

1. ログファイルへのパスを取得するには、Typescript **Output** コンソールを参照してください。

2. ログファイルを開くには、**Open TS Server log**コマンドを使用します。

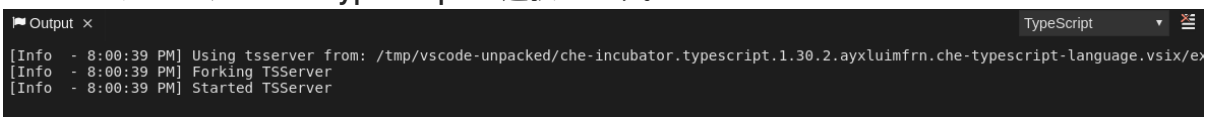


8.1.1.7.3. Output パネルでの Typescript ログ出力の表示

本セクションでは、**Output** パネルで Typescript ログの出力を表示する方法を説明します。

手順

1. **Output** ビューに移動します。
2. ドロップダウンリストで **TypeScript** を選択します。



8.1.1.8. Java のログの表示

一般的な診断以外に、ユーザーは Language Support for Java(Eclipse JDT Language Server) プラグインの各種アクションを実行できます。

8.1.1.8.1. Eclipse JDT Language Server の状態の確認

手順

Eclipse JDT Language Server プラグインを実行しているコンテナーが Eclipse JDT Language Server のメインプロセスを実行しているかどうかを確認します。

1. Eclipse JDT Language Server プラグインを実行しているコンテナーでターミナルを開きます (コンテナーのサンプル名: **vscode-javaxxx**)。
2. ターミナル内で **ps aux | grep jdt** コマンドを実行して、Eclipse JDT Language Server プロセスがコンテナーで実行されているかどうかを確認します。プロセスが実行されている場合、出力は以下のようになります。

```
usr/lib/jvm/default-jvm/bin/java --add-modules=ALL-SYSTEM --add-opens java.base/java.util
```

   このメッセージは、使用される Visual Studio Code Java 拡張機能も表示します。言語サーバーが実行されていない場合には、これはコンテナー内で起動していません。

3. 「OpenShift Dev Spaces ワークスペースログの表示」 で説明されているすべてのログを確認してください。

8.1.1.8.2. Eclipse JDT Language Server 機能の確認

手順

Eclipse JDT Language Server プロセスを実行している場合は、言語サーバーの機能が機能しているかどうかを確認します。

1. Java ファイルを開き、ホバーや自動補完機能を使用します。誤りのあるファイルの場合、ユーザーはこの **Outline** ビューまたは **Problems** ビューで Java を確認できます。

8.1.1.8.3. Java 言語サーバーログの表示

手順

Eclipse JDT Language Server には、エラー、実行されたコマンド、およびイベントについてのログを記録する独自のワークスペースがあります。

1. このログファイルを開くには、Eclipse JDT Language Server プラグインを実行しているコンテナーでターミナルを開きます。**Java: Open Java Language Server log file**コマンドを実行してログファイルを表示することもできます。

2. **cat <PATH_TO_LOG_FILE>** を実行します。**PATH_TO_LOG_FILE** は **/home/theia/.theia/workspace-storage/<workspace_name>/redhat.java/jdt_ws/.metadata/.log** になります。

8.1.1.8.4. Java Language Server Protocol (LSP) メッセージのロギング

手順

LSP メッセージのログを Visual Studio Code **Output** ビューに記録するには、**java.trace.server** 属性を **verbose** に設定してトレースを有効にします。

関連情報

トラブルシューティングの手順については、Visual Studio Code Java GitHub リポジトリー を参照してください。

8.1.1.9. Intelephense のログの表示

8.1.1.9.1. Intelephense クライアントサーバー通信のロギング

手順

PHP Intelephense 言語のサポートを、クライアントサーバー接続のログを **Output** ビューに記録するように設定するには、以下を実行します。

1. **File → Settings** をクリックします。

2. **Preferences** ビューを開きます。

3. **Intelephense** セクションを拡張し、**trace.server.verbose** 設定値を **verbose** に設定してすべての通信イベントを表示します (デフォルト値は **off** です)。

8.1.1.9.2. Output パネルでの Intelephense イベントの表示

この手順では、**Output** パネルで Intelephense イベントを表示する方法を説明します。

手順

1. **View → Output** をクリックします。
2. **Output** ビューのドロップダウンリストで **Intelephense** を選択します。

8.1.1.10. PHP-Debug のログの表示

この手順では、PHP Debug プラグインの診断メッセージのログを **Debug Console** ビューに記録するように PHP Debug プラグインを設定する方法を説明します。デバッグセッションの開始前にこれを設定します。

手順

1. **launch.json** ファイルで、**"log": true** 属性を **php** 設定に追加します。
2. デバッグセッションを開始します。
3. 診断メッセージは、アプリケーションの出力と共に **Debug Console** ビューに出力されます。

8.1.1.11. XML のログの表示

一般的な診断以外に、ユーザーが実行できる XML プラグイン固有のアクションがあります。

8.1.1.11.1. XML 言語サーバーの状態の確認

手順

1. **vscode-xml-<xxx>** という名前のコンテナーでターミナルを開きます。
2. **ps aux | grep java** を実行して、XML 言語サーバーが起動していることを確認します。プロセスが実行されている場合、出力は以下のようになります。

   ```
   java ***/org.eclipse.ls4xml-uber.jar`
   ```

   そうでない場合は、「OpenShift Dev Spaces ワークスペースログの表示」 の章を参照してください。

8.1.1.11.2. XML 言語サーバーの機能フラグの確認

手順

1. 機能が有効にされているかどうかを確認します。XML プラグインは、機能を有効かつ無効にできる複数の設定を提供します。
   - **xml.format.enabled**: フォーマッターを有効にします。
   - **xml.validation.enabled**: 検証を有効にします。
   - **xml.documentSymbols.enabled**: ドキュメントシンボルを有効にします。
2. XML 言語サーバーが機能しているかどうかを確認するには、**<hello></hello>** などの単純な XML 要素を作成し、右側の **Outline** パネルに表示されることを確認します。
3. ドキュメントのシンボルが表示されない場合は、**xml.documentSymbols.enabled** 属性が **true**

に設定されていることを確認します。**true** の場合でシンボルがない場合は、言語サーバーはエディターにフックされない可能性があります。ドキュメントのシンボルがある場合は、言語サーバーがエディターに接続されます。

4. ユーザーが必要とする機能が、設定の **true** に設定されていることを確認します (デフォルトでは **true** に設定されます)。機能のいずれかが機能していないか、または予想通りに機能しない場合は、言語サーバー に対する問題を報告します。

8.1.1.11.3. XML Language Server Protocol (LSP) トレースの有効化

手順

LSP メッセージのログを Visual Studio Code **Output** ビューに記録するには、**xml.trace.server** 属性を **verbose** に設定してトレースを有効にします。

8.1.1.11.4. XML 言語サーバーログの表示

手順

言語サーバーのログは、**/home/theia/.theia/workspace-storage/<workspace_name>/redhat.vscode-xml/lsp4xml.log** のプラグインサイドカーコンテナーにあります。

8.1.1.12. YAML のログの表示

本セクションでは、一般的な診断のほかに、ユーザーが実行できる YAML プラグインに固有のアクションについて説明します。

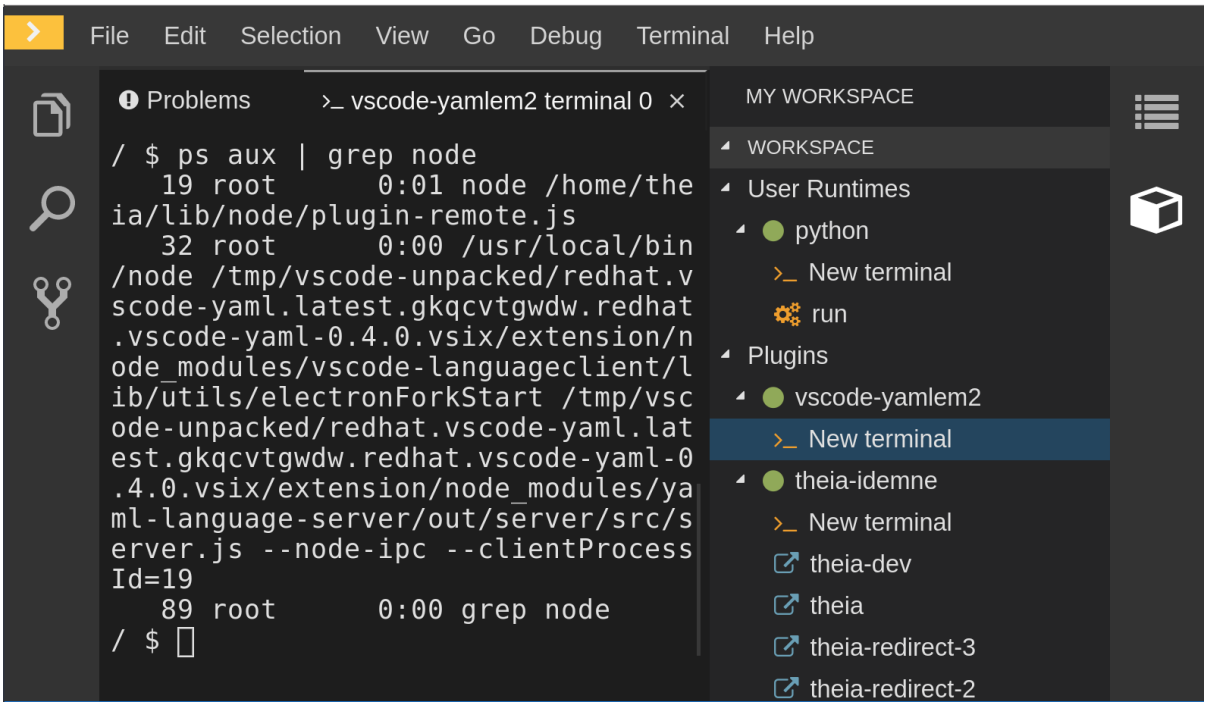
8.1.1.12.1. YAML 言語サーバーの状態の確認

本セクションでは、YAML 言語サーバーの状態を確認する方法を説明します。

手順

YAML プラグインを実行するコンテナーが YAML 言語サーバーを実行しているかどうかを確認します。

1. エディターを使用し、YAML プラグインを実行しているコンテナーでターミナルを開きます (コンテナーの名前の例: **vscode-yaml-<xxx>**)。
2. ターミナルで **ps aux | grep node** コマンドを実行します。このコマンドは、現在のコンテナーで実行されているすべてのノードプロセスを検索します。
3. コマンド **node **/server.js が実行していることを確認します。**



コンテナーで実行している node **/server.js は、言語サーバーが実行していることを示します。これが実行されていない場合、言語サーバーはコンテナー内で起動していません。この場合は、「OpenShift Dev Spaces ワークスペースログの表示」 を参照してください。

8.1.1.12.2. YAML 言語サーバーの機能フラグの確認

手順

機能フラグを確認するには、以下を実行します。

1. 機能が有効にされているかどうかを確認します。YAML プラグインは、以下のような機能を有効および無効にできる複数の設定を提供します。
   - **xml.format.enabled**: フォーマッターを有効にします。
   - **yaml.validate**: 検証を有効にします。
   - **yaml.hover**: ホバー機能を有効にします。
   - **yaml.completion**: 補完 (completion) 機能を有効にします。
2. プラグインが機能しているかどうかを確認するには、**hello: world** などの最も簡単な YAML を入力してから、エディターの右側にある Outline パネルを開きます。
3. ドキュメントのシンボルがあるかどうかを確認します。yes の場合、言語サーバーはエディターに接続されます。
4. いずれの機能も機能していない場合は、上記の設定が **true** に設定されていることを確認してください (デフォルトでは **true** に設定されます)。この機能が機能していない場合は、言語サーバー に対する問題を報告します。

8.1.1.12.3. YAML Language Server Protocol (LSP) トレースの有効化

手順

LSP メッセージのログを Visual Studio Code Output ビューに記録するには、**yaml.trace.server** 属性を **verbose** に設定してトレースを有効にします。

8.1.1.13. OmniSharp-Theia プラグインを使用した .NET のログの表示

8.1.1.13.1. Omnisharp-Theia プラグイン

OpenShift Dev Spaces は、OmniSharp-Theia プラグインをリモートプラグインとして使用します。これは github.com/redhat-developer/omnisharp-theia-plugin にあります。問題が生じた場合は、これを報告し、修正をリポジトリーにコントリビュートします。

このプラグインは omnisharp-roslyn を言語サーバーとして登録し、C# アプリケーションについてのプロジェクトの依存関係および言語構文を提供します。

言語サーバーは、.NET SDK 2.2.105 で実行されます。

8.1.1.13.2. OmniSharp-Theia プラグイン言語サーバーの状態の確認

手順

Omnisharp-Theia プラグインを実行するコンテナーが OmniSharp を実行しているかどうかを確認するには、**ps aux | grep OmniSharp.exe** コマンドを実行します。プロセスが実行されている場合、以下が出力例になります。

```
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/bin/mono
/tmp/theia-unpacked/redhat-developer.che-omnisharp-
plugin.0.0.1.zcpaqpczwb.omnisharp_theia_plugin.theia/server/omnisharp/OmniSharp.exe
```

出力が異なる場合、言語サーバーはコンテナー内で起動していません。「OpenShift Dev Spaces ワークスペースログの表示」 で説明されているログを確認してください。

8.1.1.13.3. OmniSharp Che-Theia プラグインの言語サーバー機能の確認

手順

- OmniSharp.exe プロセスが実行されている場合、**.cs** ファイルを開き、ホバーまたは補完機能を試行するか、または Problems または Outline ビューを開いて、言語サーバーの機能が機能しているかどうかを確認します。

8.1.1.13.4. OmniSharp-Theia プラグインログの Output パネルでの表示

手順

OmniSharp.exe が実行されている場合、Output パネルのすべての情報のログが記録されます。ログを表示するには、Output ビューを開き、ドロップダウンリストから C# を選択します。

8.1.1.14. NetcoredebugOutput プラグインを使用した .NET のログの表示

8.1.1.14.1. NetcoredebugOutput プラグイン

NetcoredebugOutput プラグインは、**netcoredbg** ツールを提供します。このツールは、Visual Studio Code Debug Adapter プロトコルを実装し、ユーザーが .NET Core ランタイムで .NET アプリケーションをデバッグできるようにします。

NetcoredebugOutput プラグインが実行されているコンテナーには .NET SDK v.2.2.105 が含まれます。

8.1.1.14.2. NetcoredebugOutput プラグインの状態の確認

手順

1. launch.json ファイルに netcoredbg デバッグ設定があるかどうかを確認します。

   例8.1 デバッグ設定のサンプル

   ```
   {
    "type": "netcoredbg",
    "request": "launch",
    "program": "${workspaceFolder}/bin/Debug/<target-framework>/<project-name.dll>",
    "args": [],
    "name": ".NET Core Launch (console)",
    "stopAtEntry": false,
    "console": "internalConsole"
   }
   ```

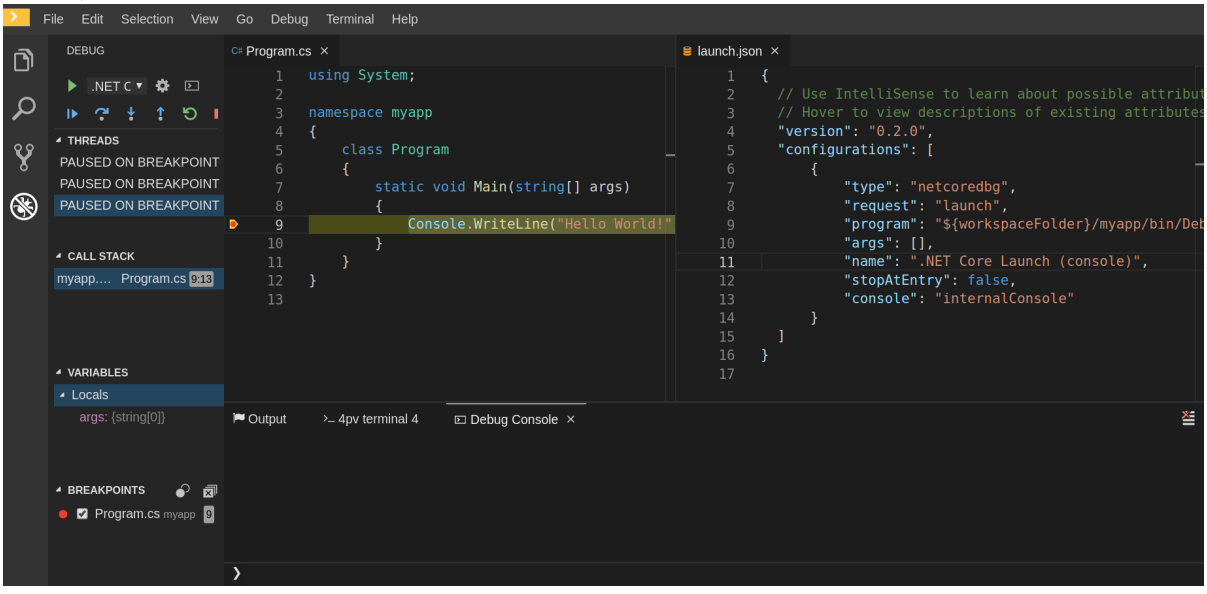
2. launch.json ファイルの configuration セクションの中括弧内で自動補完機能をテストします。netcoredbg が見つかる場合、Che-Theia プラグインは正しく初期化されています。そうでない場合は、「OpenShift Dev Spaces ワークスペースログの表示」を参照してください。

8.1.1.14.3. NetcoredebugOutput プラグインログの Output パネルでの表示

本セクションでは、Output パネルで NetcoredebugOutput プラグインログを表示する方法を説明します。

手順

- Debug コンソールを開きます。



8.1.1.15. Camel のログの表示

8.1.1.15.1. Camel 言語サーバーの状態の確認

手順

ユーザーは、vscode-apache-camel<xxx> Camel コンテナーに保存される Camel 言語ツールを使用して、サイドカーコンテナーのログ出力を検査できます。

言語サーバーの状態を確認するには、以下のコマンドを実行します。

1. **vscode-apache-camel<xxx> コンテナー内でターミナルを開きます。**
2. **ps aux | grep java コマンドを実行します。以下は、言語サーバープロセスの例です。**

```
java -jar /tmp/vscode-unpacked/camel-tooling.vscode-apache-
camel.latest.euqhbmepxd.camel-tooling.vscode-apache-camel-
0.0.14.vsix/extension/jars/language-server.jar
```

3. **見つからない場合は、「OpenShift Dev Spaces ワークスペースログの表示」を参照してください。**

8.1.1.15.2. Camel ログの Output パネルでの表示

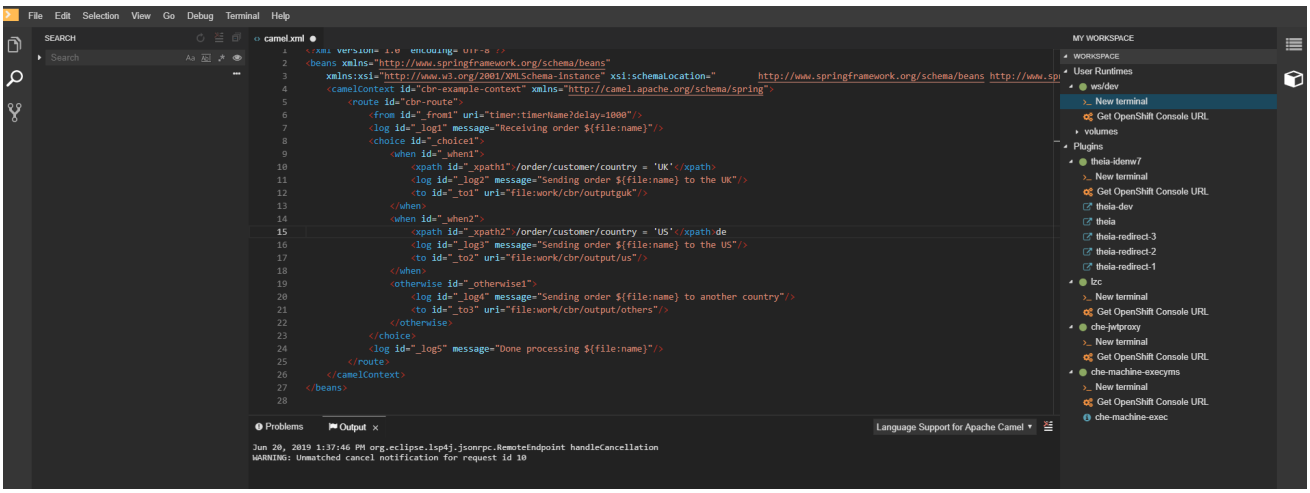
Camel 言語サーバーは、そのログを $\{java.io.tmpdir}/log-camel-lsp.out ファイルに書き込む SpringBoot アプリケーションです。$\{java.io.tmpdir} は /tmp ディレクトリーを参照するため、ファイル名は /tmp/log-camel-lsp.out になります。

手順

Camel 言語サーバーのログは、Language Support for Apache Camelという名前の Output チャネルに出力されます。

注記

出力チャネルは、クライアントサイドで最初にログエントリーが作成される際にのみ作成されます。これは、問題がない場合には存在しない場合があります。



8.1.2. Che-Theia IDE ログの表示

本セクションでは、Che-Theia IDE ログを表示する方法を説明します。

8.1.2.1. OpenShift CLI を使用した Che-Theia エディターログの表示

Che-Theia エディターログの確認は、エディターによって読み込まれるプラグインについてよりよく理解し、知見を得るのに役立ちます。本セクションでは、OpenShift CLI (コマンドコマンドラインインターフェイス) を使用して Che-Theia エディターログにアクセスする方法を説明します。

前提条件

- **OpenShift Dev Spaces は OpenShift クラスターにデプロイされます。**
- **ワークスペースが作成されています。**
- **ユーザーは OpenShift Dev Spaces インストールプロジェクトにいます。**

手順

1. **利用可能な Pod の一覧を取得します。**

```
$ oc get pods
```

例

```
$ oc get pods
NAME                                                READY  STATUS   RESTARTS  AGE
devspaces-9-xz6g8                                   1/1    Running  1         15h
workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2  4/4    Running  0         1h
```

2. **特定の Pod で利用可能なコンテナーの一覧を取得します。**

```
$ oc get pods <name-of-pod> --output jsonpath='\{.spec.containers[*].name}'
```

以下に例を示します。

```
$ oc get pods workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -o
jsonpath='\{.spec.containers[*].name}'
> go-cli che-machine-exechr7 theia-idexzb vscode-gox3r
```

3. **theia/ide コンテナーからログを取得します。**

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

以下に例を示します。

```
$ oc logs --follow workspace0zqb2ew3py4srthh.go-cli-549cdcf69-9n4w2 -container
theia-idexzb
>root INFO unzipping the plug-in 'task_plugin.theia' to directory: /tmp/theia-
unpacked/task_plugin.theia
root INFO unzipping the plug-in 'theia_yeoman_plugin.theia' to directory: /tmp/theia-
unpacked/theia_yeoman_plugin.theia
root WARN A handler with prefix term  is already registered.
root INFO [nsfw-watcher: 75] Started watching: /home/theia/.theia
root WARN e.onStart is slow, took: 367.4600000013015 ms
```

```
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [nsfw-watcher: 75] Started watching: /projects/.theia/tasks.json
root INFO [4f9590c5-e1c5-40d1-b9f8-ec31ec3bdac5] Sync of 9 plugins took:
62.26000000242493 ms
root INFO [nsfw-watcher: 75] Started watching: /projects
root INFO [hosted-plugin: 88] PLUGIN_HOST(88) starting instance
```

8.2. VERBOSE モードを使用したワークスペースの開始時の障害の調査

詳細モードでは、ユーザーは拡大したログ出力に到達し、ワークスペースの起動時に障害を調査できます。

通常のログエントリーの他に、Verbose モードには各ワークスペースのコンテナーログも表示されます。

8.2.1. 開始に失敗した後、OpenShift Dev Spaces ワークスペースを Verbose モードで再起動

本セクションでは、ワークスペースの起動時に障害後に Verbose モードで OpenShift Dev Spaces ワークスペースを再起動する方法を説明します。ダッシュボードは、ワークスペースの起動時にワークスペースが失敗すると、Verbose モードでワークスペースの再起動を提案します。

前提条件

- **OpenShift Dev Spaces の実行中のインスタンス。**
- **起動に失敗する既存のワークスペース。**

手順

1. **Dashboard を使用してワークスペースを起動しようとします。**
2. **起動に失敗する場合は、表示される Open in Verbose modeリンクをクリックします。**
3. **Logs タブを確認して、ワークスペースの失敗の理由を見つけます。**

8.2.2. Verbose モードでの OpenShift Dev Spaces ワークスペースの開始

このセクションでは、Red Hat OpenShift Dev Spaces ワークスペースを Verbose モードで開始する方法を説明します。

前提条件

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

手順

1. **Workspace タブを開きます。**
2. **ワークスペース専用の行の左側で、3 つの水平点として表示されるドロップダウンメニューにアクセスし、Open in Verbose modeオプションを選択します。または、このオプションは Actions ドロップダウンメニューでワークスペースの詳細でも利用できます。**

3. Logs タブを確認して、ワークスペースの失敗の理由を見つけます。

8.3. 速度の遅いワークスペースのトラブルシューティング

ワークスペースの起動には時間がかかる場合があります。チューニングにより、この起動時間を短縮できる場合があります。オプションによっては、管理者またはユーザーはチューニングを行うことができます。

本セクションでは、ワークスペースをより迅速に起動したり、ワークスペースのランタイムパフォーマンスを改善したりするためのチューニングオプションが複数含まれています。

8.3.1. ワークスペースの起動時間の改善

Image Puller を使用したイメージのキャッシュ

ロール: 管理者
ワークスペースを起動すると、OpenShift はイメージをレジストリーからプルします。ワークスペースには、数多くのコンテナーを含めることができます。つまり OpenShift は、(コンテナーごとに 1 つの) Pod のイメージをプルするため、複数の Pod のイメージをプルすることを意味します。イメージのサイズと帯域幅によっては、これには時間がかかる場合があります。

Image Puller は、各 OpenShift ノードでイメージをキャッシュできるツールです。このため、プル前のイメージにより、起動時間が短縮されます。https://access.redhat.com/documentation/ja-jp/red_hat_openshift_dev_spaces/3.0/html-single/administration_guide/index#caching-images-for-faster-workspace-start を参照してください。

より適切なストレージタイプの選択

ロール: 管理者およびユーザー
すべてのワークスペースには共有ボリュームが割り当てられています。このボリュームはプロジェクトファイルを保存するため、ワークスペースを再起動する際に変更が引き続き利用できるようになります。ストレージによっては、割り当てに数分かかる可能性があり、I/O が遅くなる可能性があります。

オフラインインストール

ロール: 管理者
OpenShift Dev Spaces のコンポーネントは OCI イメージです。オフラインモードで Red Hat OpenShift Dev Spaces をセットアップします (エアギャップシナリオ) は、すべてが最初から利用可能になる必要があるために追加のダウンロードを削減します。https://access.redhat.com/documentation/ja-jp/red_hat_openshift_dev_spaces/3.0/html-single/administration_guide/index#installing-devspaces-in-a-restricted-environment-on-openshift を参照してください。

ワークスペースプラグインの最適化

ロール: ユーザー
各種のプラグインを選択する場合、各プラグインでは OCI イメージである独自のサイドカーコンテナーを使用できます。OpenShift はこれらのサイドカーコンテナーのイメージをプルします。

プラグインの数を減らすか、またはそれらを無効にして起動時間が短縮されるかどうかを確認します。https://access.redhat.com/documentation/ja-jp/red_hat_openshift_dev_spaces/3.0/html-single/administration_guide/index#caching-images-for-faster-workspace-start も参照してください。

パブリックエンドポイントの数の縮小

ロール: 管理者
それぞれのエンドポイントについて、OpenShift は OpenShift Route オブジェクトを作成します。基礎となる設定によっては、作成に時間がかかる場合があります。

この問題を回避するには、公開される部分を縮小します。たとえば、コンテナー内でリッスンする新規ポートを自動的に検出し、ローカル IP アドレス (127.0.0.1) を使用してプロセスのトラフィックをリダイレクトする場合、Che-Theia IDE プラグインには 3 つのオプションのルートがあります。

エンドポイントの数を減らし、すべてのプラグインのエンドポイントをチェックすることで、ワークスペースの起動が速くなります。

CDN 設定

IDE エディターは CDN (コンテンツ配信ネットワーク) を使用してコンテンツを提供します。コンテンツがクライアント (またはオフライン設定のローカルルート) に対して CDN を使用することを確認します。
これを確認するには、ブラウザーで Developer Tools を開き、Network タブに **vendors** があることを確認します。**vendor.<random-id>.js** または **editor.main.*** は CDN URL から取得する必要があります。

8.3.2. ワークスペースのランタイムパフォーマンスの改善

十分な CPU リソースを提供する

プラグインは CPU リソースを消費します。たとえば、プラグインが IntelliSense 機能を提供する場合、CPU リソースを増やすと、パフォーマンスが向上する可能性があります。
devfile 定義 **devfile.yaml** の CPU 設定が正しいことを確認します。

```
apiVersion: 1.0.0

components:
 -
   type: chePlugin
   id: id/of/plug-in
   cpuLimit: 1360Mi (1)
   cpuRequest: 100m (2)
```

1. プラグインの CPU 制限を指定します。
2. プラグインの CPU 要求を指定します。

十分なメモリーを提供する

プラグインは CPU およびメモリーリソースを消費します。たとえば、プラグインが IntelliSense 機能を提供する場合、データを収集すると、コンテナーに割り当てられるすべてのメモリーを消費する可能性があります。
プラグインにより多くのメモリーを提供することで、パフォーマンスを改善できます。以下のメモリー設定が正しいことを確認します。

- プラグイン定義: **meta.yaml** ファイル
- devfile 定義: **devfile.yaml** ファイル

```
apiVersion: v2
```

```
spec:
  containers:
    - image: "quay.io/my-image"
      name: "vscode-plugin"
      memoryLimit: "512Mi" (1)
  extensions:
    - https://link.to/vsix
```

1. プラグインのメモリー制限を指定します。

devfile 定義 (devfile.yaml)

```
apiVersion: 1.0.0

components:
  -
    type: chePlugin
    id: id/of/plug-in
    memoryLimit: 1048M (1)
    memoryRequest: 256M
```

1. このプラグインのメモリー制限を指定します。

8.4. ネットワーク問題のトラブルシューティング

本セクションでは、ネットワークポリシーに関連する問題を回避したり、解決したりする方法を説明します。OpenShift Dev Spaces には、WebSocket Secure (WSS) 接続の可用性が必要です。セキュアな WebSocket 接続では、不正なプロキシーによる干渉リスクが軽減されるため、機密性および信頼性が強化されます。

前提条件

- ポート 443 の WebSocket Secure (WSS) 接続がネットワークで利用可能である必要があります。ファイアウォールおよびプロキシーに追加の設定が必要になる場合があります。
- サポートされる Web ブラウザーを使用します。
  - Google Chrome
  - Mozilla Firefox

手順

1. ブラウザーが WebSocket プロトコルをサポートすることを確認します。参照: Searching a websocket test
2. ファイアウォール設定の確認: ポート 443 で WebSocket Secure (WSS) 接続が利用可能である必要があります。
3. プロキシーサーバー設定の確認: プロキシーがポート 443 で WebSocket Secure (WSS) 接続を送信し、インターセプトします。

第9章 VISUAL STUDIO CODE 拡張機能のワークスペースへの追加

以前は、devfiles v1 形式では、devfile を使用して IDE 固有のプラグインと Visual Studio Code 拡張機能を指定していました。現在、devfiles v2 では、プラグインと拡張機能を指定するために、devfile ではなく特定のメタフォルダーを使用しています。

9.1. OPENSHIFT DEV SPACES プラグインレジストリーの概要

すべての OpenShift Dev Spaces インスタンスには、デフォルトのプラグインおよび拡張機能のレジストリーがあります。Che-Theia IDE は、これらのプラグインおよび拡張機能に関する情報をレジストリーから取得してインストールします。

デフォルトのプラグイン、拡張機能、およびソースコードの概要については、この OpenShift Dev Spaces レジストリープロジェクト を確認してください。メインブランチにコミットするたびに更新されるオンラインインスタンスは、ここ にあります。エアギャップ環境で作業していない場合は、Che-Theia に別のプラグインまたは拡張レジストリーを使用できます。デフォルトのレジストリーのみが使用可能です。

Che-Code Visual Studio Code エディターのプラグインおよび拡張機能の概要は、OpenVSX インスタンス にあります。このエディターでは、エアギャップはサポートされていません。

9.2. .VSCODE/EXTENSIONS.JSON への拡張子の追加

Visual Studio Code 拡張機能をワークスペースに追加する最も簡単な方法は、それを **.vscode/extensions.json** ファイルに追加することです。このメソッドの主な利点は、サポートされているすべての OpenShift Dev Spaces IDE で機能することです。

Che-Theia IDE を使用する場合、拡張機能は自動的にインストールおよび設定されます。Che-Code Visual Studio Code フォークでサポートされている別の IDE を使用する場合、IDE は拡張機能のインストールを推奨するポップアップを表示します。

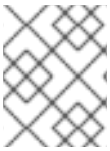
前提条件

1. GitHub リポジトリーのルートに **.vscode/extensions.json** ファイルがあります。

手順

1. 拡張機能 ID を **.vscode/extensions.json** ファイルに追加します。**.** 記号を使用して、発行元と拡張機能を分離するために署名します。次の例では、Red Hat Visual Studio Code Java 拡張機能の ID を使用しています。

```
{
  "recommendations": [
    "redhat.java"
  ]
}
```



注記

指定された拡張 ID のセットが OpenShift Dev Spaces レジストリーで使用できない場合、ワークスペースは拡張なしで開始されます。

9.3. プラグインパラメーターを .CHE/CHE-THEIA-PLUGINS.YAML に追加

.che/che-theia-plugins.yaml ファイルを変更することで、プラグインにパラメーターを追加できます。これらの変更には次のものが含まれます。

- ワークスペースインストール用のプラグインの定義
- デフォルトのメモリー制限を変更します。
- デフォルト設定を上書きします。

9.3.1. ワークスペースインストール用のプラグインの定義

ワークスペースにインストールするプラグインを定義します。

前提条件

1. GitHub リポジトリーのルートに .che/che-theia-plugins.yaml ファイルがある。

手順

1. プラグインの ID を .che/che-theia-plugins.yaml ファイルに追加します。/ 記号を使用して、発行元とプラグインの名前を区切ります。次の例では、Red Hat Visual Studio Code Java 拡張機能の ID を使用しています。

```
- id: redhat/java/latest
```

9.3.2. デフォルトのメモリー制限の変更

メモリー制限などのコンテナー設定を上書きします。

前提条件

1. GitHub リポジトリーのルートに .che/che-theia-plugins.yaml ファイルがある。

手順

1. プラグインの id の下にある .che/che-theia-plugins.yaml ファイルに override セクションを追加します。
2. 拡張機能のメモリー制限を指定します。次の例では、OpenShift Dev Spaces は Red Hat Visual Studio Code Java 拡張機能を OpenShift Dev Spaces ワークスペースに自動的にインストールし、ワークスペースのメモリーを 2 ギガバイト増やします。

```
- id: redhat/java/latest
  override:
    sidecar:
      memoryLimit: 2Gi
```

9.3.3. デフォルト設定のオーバーライド

ワークスペースの Visual Studio Code 拡張機能の既定の設定を上書きします。

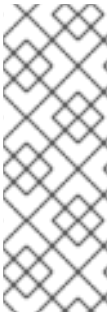
前提条件

1. GitHub リポジトリーのルートに .che/che-theia-plugins.yaml ファイルがある。

手順

1. 拡張子の id の下にある .che/che-theia-plugins.yaml ファイルに override セクションを追加します。
2. Preferences セクションで設定を指定します。次の例では、OpenShift Dev Spaces は Red Hat Visual Studio Code Java 拡張機能を Red Hat ワークスペースに自動的にインストールし、java.server.launchMode プリファレンスを LightWeight に設定します。

```
- id: redhat/java/latest
  override:
    preferences:
      java.server.launchMode: LightWeight
```



注記

IDE の UI で設定を変更するか、.vscode/settings.json ファイルに追加することで、.vscode/settings.json ファイルで設定を定義することもできます。

```
{
"my.preferences": "my-value"
}
```

9.4. DEVFILE での VISUAL STUDIO CODE 拡張属性の定義

GitHub リポジトリーにファイルを追加できない場合は、プラグインまたは拡張属性の一部を devfile にインライン化することで定義できます。この手順は、.vscode/extensions.json および .che/che-theia-plugins.yaml ファイルの内容で使用できます。

9.4.1. .vscode/extensions.json ファイルのインライン化

.vscode/extensions.json ファイルの内容を使用して、devfile の拡張属性をインライン化します。

手順

1. devfile.yaml ファイルに attributes セクションを追加します。
2. .vscode/extensions.json を atributes セクションに追加します。コロン区切り文字の後に | 記号を追加します。
3. | 記号の後に .vscode/extensions.json ファイルの内容を貼り付けます。次の例では、Red Hat Visual Studio Code Java 拡張属性を使用しています。

```
schemaVersion: 2.2.0
metadata:
  name: my-example
attributes:
  .vscode/extensions.json: |
    {
      "recommendations": [
```

```
      "redhat.java"
    ]
  }
```

9.4.2. .che/che-theia-plugins.yaml ファイルのインライン化

.che/che-theia-plugins.yaml ファイルの内容を使用して、devfile のプラグイン属性をインライン化します。

手順

1. **devfile.yaml ファイルに attributes セクションを追加します。**
2. **.vscode/extensions.json を atributes セクションに追加します。コロン区切り文字の後に | 記号を追加します。**
3. **| 記号の後に .che/che-theia-plugins.yaml ファイルの内容を貼り付けます。次の例では、Red Hat Visual Studio Code Java 拡張属性を使用しています。**

```
schemaVersion: 2.2.0
metadata:
  name: my-example
attributes:
  .che/che-theia-plugins.yaml: |
    - id: redhat/java/latest
```