Red Hat OpenShift Dev Spaces 3.4

ユーザーガイド

Red Hat OpenShift Dev Spaces 3.4 の使用

Last Updated: 2023-06-03

Red Hat OpenShift Dev Spaces 3.4 ユーザーガイド

Red Hat OpenShift Dev Spaces 3.4 の使用

Robert Kratky
rkratky@redhat.com

Fabrice Flore-Thébault
ffloreth@redhat.com

Jana Vrbkova
jvrbkova@redhat.com

Max Leonov
mleonov@redhat.com

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 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」
- 「初めて貢献する方のためのバッジ」
- 「プル要求およびマージ要求の確認」
- 「スタックサンプル」

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。

1.1.1. Microsoft Visual Studio Code - Open Source

Microsoft Visual Studio Code - オープンソースは、既定のブラウザーベースの IDE です。

OpenShift Dev Spaces は、次の機能を追加します。

VSX レジストリーを開く

IDE は、Open VSX レジストリー を使用して拡張機能を一覧表示およびダウンロードします。OpenShift Dev Spaces 管理者は、Open VSX レジストリー URL を設定 できます。

推奨拡張機能

IDE は、推奨される拡張機能 を自動的にインストールします。

OpenShift Dev Spaces は、次の拡張機能を追加します。

コマンド

Devfile コマンドを Microsoft Visual Studio Code - オープンソースタスクに変換します。

手順

- 使用可能なタスクのドロップダウンリストを表示するには、**F1** を押して **Tasks: Run Task** を選択し、**Enter** を押して **che**. と入力します。

アクティビティートラッカー

Microsoft Visual Studio Code - Open Source が提示するイベントを追跡し、アクティブではないワークスペースを特定して停止します。この拡張機能は、データを保存、収集、または保存しません。

API

DevWorkspace および OpenShift Dev Spaces と対話するためのヘルパーを提供します。

GitHub 認証

GitHub への認証をサポートします。他の拡張機能で利用できる **github** 認証プロバイダーを登録します。これにより、Settings Sync で使用される GitHub 認証も提供されます。

ポート

開いているポートを検出し、リダイレクト URI を提供します。プロセスがポートのリッスンを開始すると、OpenShift Dev Spaces は結果のリソースを開くためのリンクを含めて、通知を表示します。

手順

- エンドポイントリストを表示するには、**F1** を押して **Explorer: Focus on endpoints View** を選択し、**Enter** を押します。

リモート

リモート機関にコマンドを提供します。

リソースモニター

CPU や RAM などのリソースを監視します。

Telemetry

次のイベントを検出し、**http://localhost:${DEVWORKSPACE_TELEMETRY_BACKEND_PORT}** でリッスンしているバックエンドの Telemetry プラグインに送信します。

WORKSPACE_OPENED

Telemetry 拡張機能がアクティブ化されたときに送信されます。

EDITOR_USED

vscode.workspace.onDidChangeTextDocument イベントで送信されます。

ターミナル

Dev Workspace コンテナーへのターミナルを開きます。

1.2. スタックサンプル

リモート開発環境としての Red Hat OpenShift Dev Spaces の機能を実証するために、Red Hat OpenShift Dev Spaces には、さまざまなプログラミング言語を使用したスタックサンプルが含まれています。各サンプルには devfile が含まれており、新しいプロジェクトをブートストラップするための参照として使用できます。OpenShift Dev Spaces 管理者は、サンプルをカスタマイズできます。

表1.1 サポートされる言語

言語	ビルダー、ランタイム、およびデータベース	Maturity

言語	ビルダー、ランタイム、およびデータベース	Maturity
Apache Camel K	• Red Hat Fuse	GA
Java	• OpenJDK 11 • Maven 3.6 • Gradle 6.1 • Quarkus ツール • Lombok 1.18 • JBoss EAP 7.4 • JBoss EAP XP 3.0	GA
Node.js	• Node.js 16 • NPM 8 • Express • MongoDB 3.6	GA
Python	• Python 3.8 • Pip 22.2	GA
C/C++	• GCC • cmake • make	テクノロジープレビュー
C#	• AMD64 および Intel 64 (**x86_64**) 上の Dotnet 3.1 • AMD64 および Intel 64 (**x86_64**)、および IBM Z (**s390x**) 上の Dotnet 6.0	テクノロジープレビュー
Go	• golang	テクノロジープレビュー

言語	ビルダー、ランタイム、および データベース	Maturity
PHP	• CakePHP • Composer	テクノロジープレビュー

1.3. 初めて貢献する方のためのバッジ

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>**) を置き換えて、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-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;"/#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。

1.4. プル要求およびマージ要求の確認

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。
- 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。

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 インスタンスの完全修飾ドメイン名 (FQDN) URL を知っている (**"https://devspaces-<openshift_deployment_name>.<domain_name>"**)。
- オプション: Git サーバーへの認証 が設定されています。
- 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。これにより、Universal Developer Image and with Microsoft Visual Studio Code - Open Source をワークスペース IDE として使用する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で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、プル要求のブランチのクローンを使用して新しいワークスー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」
- 「ワークスペースストレージの URL パラメーター」
- 「追加のリモートの 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)

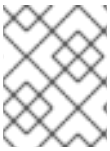
1. OpenShift Dev Spaces URL
2. 新しいワークスペースに複製される Git リポジトリーの URL。
3. 連結された任意の URL パラメーター。

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、ワークスペースはデフォルトのブラウザー内 IDE (Microsoft Visual Studio Code - Open Source) 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>**



注記

ワークスペース IDE は、リポジトリーの **che-editor.yaml** ファイル で、リモート Git リポジトリーに対してすでに設定されている場合があります。

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
Microsoft Visual Studio Code - Open Source	**che-incubator/che-code/insiders**	これは、URL パラメーターまたは **che-editor.yaml** が使用されていない場合に新しいワークスペースに読み込まれるデフォルトの IDE です。
JetBrains IntelliJ IDEA コミュニティー版	**che-incubator/che-idea/latest**	テクノロジープレビュー。
Eclipse Theia	**eclipse/che-theia/latest**	**非推奨であり、将来のリリースで削除される予定です。**

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。

2.2.6. ワークスペースストレージの URL パラメーター

新規ワークスペースを起動する URL にストレージタイプを指定する URL パラメーターが含まれていない場合、新規ワークスペースは一時ストレージまたは永続ストレージに作成されます。これは、**CheCluster** カスタムリソースのデフォルトストレージタイプとして定義されます。

ワークスペースのストレージタイプを指定する URL パラメーターは、**storageType= <storage_type>** です。

> "https://devspaces-**<openshift_deployment_name>**.**<domain_name>**"**#<git_repository_url>**?storageType=**<storage_type>** 1

1 使用できる **<storage_type>** の値:

- **ephemeral**
- **per-user** (永続)
- **per-workspace** (永続)

ヒント

ephemeral または **per-workspace** ストレージタイプを使用すると、複数のワークスペースを同時に実行できますが、これはデフォルト **per-user** ストレージタイプでは不可能です。

関連情報

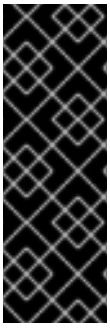
- *7章 ワークスペースの永続ストレージを要求する*

2.2.7. 追加のリモートの 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 にアクセスすると、OpenShift Dev Spaces は **origin** リモートを、組織の OpenShift Dev Spaces インスタンスの FQDN URL の後に **#** で指定した Git リポジトリーになるように設定します。

ワークスペースの追加のリモートを複製および設定するための URL パラメーターは、**remotes=** です。

> "https://devspaces-**<openshift_deployment_name>**.**<domain_name>**"**#<git_repository_url>**?remotes={{**<name_1>**,**<url_1>**},{**<name_2>**,**<url_2>**},{**<name_3>**,**<url_3>**},...}



重要

- 追加のリモートのいずれにも **origin** という名前を入力しない場合、<git_repository_url> からのリモートが複製され、デフォルトで **origin** という名前が付けられ、想定されるブランチが自動的にチェックアウトされます。
- 追加のリモートの 1 つに名前 **origin** を入力すると、そのデフォルトブランチは自動的にチェックアウトされますが、<git_repository_url> からのリモートはワークスペース用に複製されません。

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 **ワークスペース** 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

Action	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は、管理者によって設定されるか、場合によっては個々のユーザーによって設定されます。

- 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 を設定します。
- 回避策として、一部のユーザーは、個人の Git プロバイダーアクセストークン 用に独自の Kubernetes シークレットを作成して適用します。

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 プロバイダーアクセストークンの使用

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。
- devfile を使用します。
- 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 DEV SPACES の DEVFILE の概要

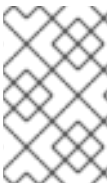
devfile は、開発環境のカスタマイズに使用される **yaml** テキストファイルです。それらを使用して、特定のニーズに合わせて devfile を設定し、カスタマイズされた devfile を複数のワークスペースで共有して、チーム全体で同一のユーザーエクスペリエンスと、ビルド、実行、およびデプロイの動作を保証します。

Devfile および Universal Developer Image

ワークスペースを開始するのに devfile は必要ありません。プロジェクトリポジトリーに devfile を含めない場合、Red Hat OpenShift Dev Spaces は Universal Developer Image (UDI) を含むデフォルトの devfile を自動的にロードします。

OpenShift Dev Spaces devfile レジストリー

OpenShift Dev Spaces devfile レジストリー には、さまざまな言語およびテクノロジー用のすぐに使用できる devfile が含まれています。



注記

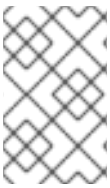
レジストリーに含まれる Devfile は Red Hat OpenShift Dev Spaces に固有のものであり、テンプレートではなくサンプルとして扱う必要があります。サンプルに含まれる他のバージョンのコンポーネントと連携するには、更新が必要になる場合があります。

関連情報

- devfile とは
- devfile の利点
- devfile カスタマイズの概要

第5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ブラウザー内 IDE は、Microsoft Visual Studio Code - Open Source です。



注記

Microsoft Visual Studio Code - Open Source の OpenShift Dev Spaces ビルドはカスタムブランド化をサポートしているため、組織はブランド化されたビルドを使用している可能性があります。

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。「1 つの新規ワークスペースのブラウザー内 IDE の選択」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。「同じ Git リポジトリーのクローンを作成するすべてのワークスペースのブラウザー内 IDE の選択」を参照してください。

表5.1 サポートされているブラウザー内 IDE

IDE	id	注記
Microsoft Visual Studio Code - Open Source	**che-incubator/che-code/insiders**	これは、URL パラメーターまたは **che-editor.yaml** が使用されていない場合に新しいワークスペースに読み込まれるデフォルトの IDE です。
JetBrains IntelliJ IDEA コミュニティー版	**che-incubator/che-idea/latest**	テクノロジープレビュー。
Eclipse Theia	**eclipse/che-theia/latest**	**非推奨であり、将来のリリースで削除される予定です。**

5.1. 1 つの新規ワークスペースのブラウザー内 IDE の選択

新しいワークスペースを開始するための URL を使用する場合には、任意のブラウザー内 IDE を選択できます。このようにして、OpenShift Dev Spaces を使用する各開発者は、同じプロジェクトリポジトリーのクローンとブラウザー内 IDE の個人的な選択でワークスペースを開始できます。

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」を追加します。
2. ブラウザーで URL にアクセスします。

検証

- 選択したブラウザー内 IDE が、起動したワークスペースのブラウザータブでロードされていることを確認します。

5.2. 同じ GIT リポジトリーのクローンを作成するすべてのワークスペースのブラウザー内 IDE の選択

5.2.1. che-editor.yaml の設定

プロジェクトの同じリモート Git リポジトリーのクローンを作成するすべてのワークスペースに同じブラウザー内 IDE を定義するには、**che-editor.yaml** ファイルを使用できます。

このようにして、チームに共通の既定のエディターを設定し、新しいコントリビューターに対して、プロジェクトに最適なエディターを提供できます。また、組織の OpenShift Dev Spaces インスタンスのデフォルトの IDE ではなく、特定のプロジェクトリポジトリーに別の IDE デフォルトを設定する必要がある場合は、**che-editor.yaml** ファイルを使用することもできます。

手順

- プロジェクトのリモート Git リポジトリーで、次のセクションで説明されているように、関連するパラメーターを指定する行で **/.che/che-editor.yaml** ファイルを作成します。

検証

1. URL にアクセスして、新しいワークスペース を開始 します。
2. 選択したブラウザー内 IDE が、起動したワークスペースのブラウザータブでロードされていることを確認します。

5.2.2. che-editor.yaml のパラメーター

che-editor.yaml で IDE を最も簡単に選択する方法は、*5章 ワークスペース IDE の選択* のブラウザー IDE でサポートされる表で提供されている IDE の **id** を指定する方法です。

例5.1 id は、プラグインレジストリーから IDE を選択

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

id パラメーターを提供する代わりに、**che-editor.yaml** ファイルは別の **che-editor.yaml** ファイルの URL への **参照**、またはプラグインレジストリーの外部にある IDE の **inline** 定義をサポートします。

例5.2 参照 は、リモート che-editor.yaml ファイルを参照

```
reference: https://<hostname_and_path_to_a_remote_file>/che-editor.yaml
```

例5.3 inline は、プラグインレジストリーなしでカスタマイズされた IDE の完全な定義を指定

```
inline:
  schemaVersion: 2.1.0
  metadata:
    name: JetBrains IntelliJ IDEA Community IDE
```

```
components:
  - name: intellij
    container:
      image: 'quay.io/che-incubator/che-idea:next'
      volumeMounts:
        - name: projector-user
          path: /home/projector-user
      mountSources: true
      memoryLimit: 2048M
      memoryRequest: 32Mi
      cpuLimit: 1500m
      cpuRequest: 100m
      endpoints:
        - name: intellij
          attributes:
            type: main
            cookiesAuthEnabled: true
            urlRewriteSupported: true
            discoverable: false
            path: /?backgroundColor=434343&wss
          targetPort: 8887
          exposure: public
          secure: false
          protocol: https
      attributes: {}
    - name: projector-user
      volume: {}
```

より複雑なシナリオの場合に、**che-editor.yaml** ファイルは **registryUrl** および **override** パラメーターをサポートします。

例5.4 registryUrl は、デフォルトの OpenShift Dev Spaces プラグインレジストリーではなく、カスタムプラグインレジストリーを参照

```
id: <editor_id> 1
registryUrl: <url_of_custom_plugin_registry>
```

1 カスタムプラグインレジストリーの IDE の **ID**。

例5.5 IDE の 1 つ以上の定義済みプロパティーのデフォルト値の override

```
... 1
override:
  containers:
    - name: che-idea
      memoryLimit: 1280Mi
      cpuLimit: 1510m
      cpuRequest: 102m
    ...
```

1 **id:**、**registryUrl:**、または **reference:**

第6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 **Dev Workspace** 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。
- 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 **Dev Workspace** 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、**Dev Workspace** 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。

- ユーザー固有 の **settings.xml** ファイルなどの Maven 設定
- SSH キーペア
- Git プロバイダーアクセストークン
- AWS 認証トークン
- 設定ファイル
- 永続ストレージ

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

6.1. 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 **Dev Workspace** 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。CLI の使用方法 を参照。
- ユーザープロジェクトですべての **Dev Workspace** コンテナーにマウントする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ている。

手順

1. Secret のマウントに必要なラベルを Secret に追加します。

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

2. 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6.1 オプションのアノテーション

アノテーション	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 **Dev Workspace** 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6.1 シークレットをファイルとしてマウント

```
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 **Dev Workspace** 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
```

6.1.1. 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 **Dev Workspace** 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。

6.1.1.1. 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。CLI の使用方法を参照。

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
```

6.1.1.2. .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。CLI の使用方法を参照。
- **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
```

6.1.1.3. 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。CLI の使用方法を参照。
- **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
```

6.1.2. Git プロバイダーアクセストークンの使用

GitHub、GitLab、または Bitbucket の OAuth は、組織の OpenShift Dev Spaces インスタンスの 管理者が設定する 必要があります。管理者が OpenShift Dev Spaces ユーザー用に設定できなかった場合の回避策は、パーソナルアクセストークンを Kubernetes シークレットとして適用することです。

アクセストークンを Secret としてマウントすると、OpenShift Dev Spaces サーバーは、リポジトリーの **/.che** および **/.vscode** フォルダーへのアクセスを含め、ワークスペースの作成中に複製されたリモートリポジトリーにアクセスできます。

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シークレットを適用します。

シークレットを適用した後、プライベート GitHub、GitLab、または Bitbucket-server リポジトリーから新しいワークスペースを作成できます。

Git プロバイダーごとに複数のアクセストークンシークレットを作成して適用できます。これらの各 Secret をユーザープロジェクトに適用する必要があります。

前提条件

- 組織の OpenShift Dev Spaces インスタンスが実行されているクラスターのクラスター管理者権限を持っています。
- クラスターにログインしました。

  ヒント

  OpenShift では、**oc** コマンドラインツールを使用してクラスターにログインできます。

  $ oc login "https://devspaces-<openshift_deployment_name>.<domain_name>" --username=<my_user>

手順

1. Git プロバイダーの Web サイトでアクセストークンを生成します。
2. アクセストークンを Base64 にエンコードします。

   ヒント

   オペレーティングシステムに base64 コマンドラインツールがインストールされている場合は、次のコマンドラインを使用できます。

   $ echo -n '<your_access_token_string>' | base64

3. Web ブラウザーで

"https://devspaces-<openshift_deployment_name>.<domain_name>"/api/user アクセスし、レスポンスから **id** 値をコピーします。これは、OpenShift Dev Spaces ユーザー ID です。

4. Git プロバイダーの API ドキュメントに従って、Git プロバイダーのユーザー ID を取得します。
   - GitHub: ユーザーを取得します。レスポンスの **id** 値を参照してください。
   - GitLab: ユーザーの一覧表示: 通常のユーザー の場合は、**username** フィルターを使用します: **/users?username=:username**。レスポンスの **id** 値を参照してください。
   - Bitbucket Server: ユーザーを取得します。レスポンスの **id** 値を参照してください。
5. 新しい OpenShift シークレットを準備します。

```
kind: Secret
apiVersion: v1
metadata:
  name: personal-access-token-<your_choice_of_name_for_this_token>
  labels:
    app.kubernetes.io/component: scm-personal-access-token
    app.kubernetes.io/part-of: che.eclipse.org
  annotations:
    che.eclipse.org/che-userid: <devspaces_user_id> (1)
    che.eclipse.org/scm-personal-access-token-name: <git_provider_name> (2)
    che.eclipse.org/scm-url: <git_provider_endpoint> (3)
    che.eclipse.org/scm-userid: '<git_provider_user_id>' (4)
    che.eclipse.org/scm-username: <git_provider_username>
data:
  token: <Base64_encoded_access_token>
type: Opaque
```

(1) OpenShift Dev Spaces ユーザー ID。

(2) Git プロバイダー名: **github** または **gitlab** または **bitbucket-server**。

(3) Git プロバイダーの URL。

(4) Git プロバイダーのユーザー ID。

6. **"https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kubernetes/namespace** にアクセスして、OpenShift Dev Spaces ユーザーの namespace を **name** として取得します。
7. クラスター内の OpenShift Dev Spaces ユーザー namespace に切り替えます。

ヒント

OpenShift の場合:

- **oc** コマンドラインツールは、クラスター内で現在使用している namespace を返すことができます。これを使用して、現在の namespace を確認できます。
  $ oc project

- 必要に応じて、コマンドラインで OpenShift Dev Spaces ユーザー namespace に切り替えることができます。
  $ oc project <your_user_namespace>

8. シークレットを適用します。

ヒント

OpenShift では、**oc** コマンドラインツールを使用できます。

```
$ oc apply -f - <<EOF
<Secret_prepared_in_step_5>
EOF
```

検証

1. Git プロバイダーがホストする リモート Git リポジトリーの URL を使用して、新しいワークスペースを開始します。
2. 変更を加えて、ワークスペースからリモート Git リポジトリーにプッシュします。

6.2. 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 **Dev Workspace** 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。CLI の使用方法を参照。
- ユーザープロジェクトで、新規の ConfigMap を作成するか、またはすべての **Dev Workspace** コンテナーにマウントする既存の ConfigMap を決定している。

手順

1. ConfigMap のマウントに必要なラベルを ConfigMap に追加します。

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

-

2. 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6.2 オプションのアノテーション

アノテーション	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。
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 **Dev Workspace** 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6.2 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 **Dev Workspace** コンテナーで使用可能になります。

6.3. 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

6.3.1. Maven

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。
- ユーザー名前空間は **<username> -devspaces** であり、**<username>** は OpenShift Dev Spaces ユーザー名です。

手順

1. **<username> -devspaces** 名前空間で、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. **<username> -devspaces** 名前空間で、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. オプション: EAP ベースの devfile を使用する場合は、2 つ目 **の settings-xml** ConfigMap を **<username> -devspaces** 名前空間に、同じ内容、別の名前、および **/home/jboss/.m2** マウントパスで適用します。

4. **<username> -devspaces** 名前空間で、TrustStore 初期化スクリプトの ConfigMap を適用します。

   Java 8

```
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-java8-truststore.sh: |
    #!/usr/bin/env bash

    keytool -importcert -noprompt -file /home/user/certs/tls.cer -trustcacerts -keystore
~/.java/current/jre/lib/security/cacerts -storepass changeit
```

Java 11

```
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-java11-truststore.sh: |
    #!/usr/bin/env bash

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

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

6.3.2. Gradle

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。

6.3.3. npm

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/
```

6.3.4. Python

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
```

6.3.5. Go

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
```

6.3.6. NuGet

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>
```

第7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 **Dev Workspace** 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

ReadWriteOnce PV を使用して別の同時実行ワークスペースを開始すると、失敗する場合があります。

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

7.1. 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7.1 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
```

7.2. 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 **Dev Workspace** 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。CLI の使用方法を参照。
- すべての **Dev Workspace** 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7.1 オプションのアノテーション

アノテーション	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7.2 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8章 OPENSHIFT との統合

- 「自動 OpenShift トークン注入」
- 「OpenShift 開発者の観点からの Dev Spaces のナビゲート」
- 「Dev Spaces からの OpenShift Web コンソールのナビゲート」

8.1. OPENSHIFT API を使用したワークスペースの管理

組織の OpenShift クラスターでは、OpenShift Dev Spaces ワークスペースは同じ名前の **DevWorkspace** カスタムリソースとして表されます。その結果、OpenShift Dev Spaces ダッシュボードに **my-workspace** という名前のワークスペースがある場合、クラスター上のユーザーのプロジェクトに **my-workspace** という名前の対応する **DevWorkspace** カスタムリソースがあります。

クラスター上の各 **DevWorkspace** カスタムリソースは OpenShift Dev Spaces ワークスペースを表すため、コマンドライン **oc** などのクライアントで OpenShift API を使用して OpenShift Dev Spaces ワークスペースを管理できます。

各 **DevWorkspace** カスタムリソースには、ワークスペース用に複製された Git リポジトリーの devfile から派生した詳細が含まれています。たとえば、devfile は、devfile コマンドとワークスペースコンテナー設定を提供する場合があります。

8.1.1. すべてのワークスペースの一覧表示

ユーザーは、コマンドラインを使用してワークスペースを一覧表示できます。

前提条件

- クラスター上のプロジェクトで **DevWorkspace** リソースを **get** するための権限を持つアクティブな **oc** セッションです。CLI の使用方法 を参照。
- クラスター上の関連する OpenShift Dev Spaces ユーザー namespace を把握している。

  ヒント

  "https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kubernetes/namespace にアクセスして、OpenShift Dev Spaces ユーザーの namespace を **name** として取得できます。

- クラスターの OpenShift Dev Spaces ユーザー namespace にいる。

  ヒント

  OpenShift では、コマンドライン **oc** ツールを使用して、現在の namespace を表示したり、namespace に切り替えたり できます。

手順

- ワークスペースを一覧表示するには、コマンドラインで次のように入力します。

```
$ oc get devworkspaces
```

例8.1 出力

```
NAMESPACE   NAME                DEVWORKSPACE ID            PHASE     INFO
user1-dev   spring-petclinic    workspace6d99e9ffb9784491  Running   https://url-to-
workspace.com
user1-dev   golang-example      workspacedf64e4a492cd4701  Stopped   Stopped
user1-dev   python-hello-world  workspace69c26884bbc141f2  Failed    Container tooling
has state CrashLoopBackOff
```

ヒント

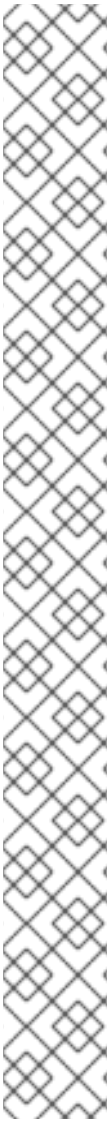
このコマンドに **--watch** フラグを追加すると、**PHASE** の変更をライブで表示できます。

注記

クラスターの管理権限を持つユーザーは、**--all-namespaces** フラグを含めることで、すべての OpenShift Dev Spaces ユーザーからのすべてのワークスペースを一覧表示できます。

8.1.2. ワークスペースの作成

ユースケースで OpenShift Dev Spaces ダッシュボードの使用が許可されていない場合は、カスタムリソースをクラスターに適用することで、OpenShift API を使用してワークスペースを作成できます。



注記

OpenShift Dev Spaces ダッシュボードを使用してワークスペースを作成すると、コマンドラインを使用する場合と比較して、ユーザーエクスペリエンスと設定の利点が向上します。

- ユーザーとして、クラスターに自動的にログインします。
- OpenShift クライアントは自動的に動作します。
- OpenShift Dev Spaces とそのコンポーネントは、ターゲット Git リポジトリーの devfile をクラスター上の **DevWorkspace** および **DevWorkspaceTemplate** カスタムリソースに自動的に変換します。
- ワークスペースへのアクセスは、デフォルトで、ワークスペースの **DevWorkspace** にある **routingClass: che** で保護されています。
- **DevWorkspaceOperatorConfig** 設定の認識は、OpenShift Dev Spaces によって管理されます。
- 以下を含む、**CheCluster** カスタムリソースで指定された **spec.devEnvironments** の設定の認識:
  - 永続的なストレージ戦略は **devEnvironments.storage** で指定されます。
  - デフォルトの IDE は **devEnvironments.defaultEditor** で指定されます。
  - デフォルトのプラグインは **devEnvironments.defaultPlugins** で指定されます。
  - コンテナーのビルド設定は **devEnvironments.containerBuildConfiguration** で指定されます。

前提条件

- クラスター上のプロジェクトに **DevWorkspace** リソースを作成する権限を持つアクティブな **oc** セッションです。CLI の使用方法 を参照。
- クラスター上の関連する OpenShift Dev Spaces ユーザー namespace を把握している。

  ヒント

  "https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kubernetes/namespace にアクセスして、OpenShift Dev Spaces ユーザーの namespace を **name** として取得できます。

- クラスターの OpenShift Dev Spaces ユーザー namespace にいる。

  ヒント

  OpenShift では、コマンドライン **oc** ツールを使用して、現在の namespace を表示したり、namespace に切り替えたり できます。

注記

他のユーザーのためにワークスペースを作成する予定の OpenShift Dev Spaces 管理者は、OpenShift Dev Spaces または管理者によってプロビジョニングされたユーザー namespace に **DevWorkspace** カスタムリソースを作成する必要があります。https://access.redhat.com/documentation/ja-jp/red_hat_openshift_dev_spaces/3.4/html-single/administration_guide/index#administration-guide:configuring-namespace-provisioning を参照してください。

手順

1. **DevWorkspace** カスタムリソースを準備するには、ターゲット Git リポジトリーの devfile の内容をコピーします。

例8.2 schemaVersion: 2.2.0 の devfile コンテンツをコピー

```
components:
  - name: tooling-container
    container:
      image: quay.io/devfile/universal-developer-image:ubi8-latest
```

ヒント

詳細については、devfile v2 のドキュメントを 参照してください。

2. 前のステップの devfile の内容を **spec.template** フィールドの下に貼り付けて、**DevWorkspace** カスタムリソースを作成します。

例8.3 DevWorkspace カスタムリソース

```
kind: DevWorkspace
apiVersion: workspace.devfile.io/v1alpha2
metadata:
  name: my-devworkspace 1
  namespace: user1-dev 2
spec:
  routingClass: che
  started: true 3
  contributions: 4
    - name: ide
      uri:
"https://devspaces-&lt;openshift_deployment_name&gt;.&lt;domain_name&gt;"/plugin
-registry/v3/plugins/che-incubator/che-code/insiders/devfile.yaml
  template:
    projects: 5
      - name: my-project-name
        git:
          remotes:
            origin: https://github.com/eclipse-che/che-docs
    components: 6
      - name: tooling-container
        container:
          image: quay.io/devfile/universal-developer-image:ubi8-latest
```

1. **DevWorkspace** カスタムリソースの名前です。これが新しいワークスペースの名前になります。
2. 新しいワークスペースのターゲットプロジェクトであるユーザー namespace です。
3. **DevWorkspace** カスタムリソースの作成時にワークスペースを開始する必要があるかどうかを決定します。
4. プラグインレジストリーからの Microsoft Visual Studio Code - オープンソース IDE devfile への URL 参照 です。
5. 起動時にワークスペースに複製する Git リポジトリーの詳細です。
6. ワークスペースコンテナーやボリュームコンポーネントなどのコンポーネントの一覧です。

3. **DevWorkspace** カスタムリソースをクラスターに適用します。

検証

1. **DevWorkspace** の **PHASE** ステータスをチェックして、ワークスペースが起動していることを確認します。

```
$ oc get devworkspaces -n <user_project> --watch
```

例8.4 出力

```
NAMESPACE        NAME                DEVWORKSPACE ID            PHASE      INFO
user1-dev        my-devworkspace     workspacedf64e4a492cd4701  Starting   Waiting
for workspace deployment
```

2. ワークスペースが正常に開始されると、**oc get devworkspaces** コマンドの出力でその **PHASE** ステータスが **Running** に変わります。

例8.5 出力

```
NAMESPACE        NAME                DEVWORKSPACE ID            PHASE      INFO
user1-dev        my-devworkspace     workspacedf64e4a492cd4701  Running
https://url-to-workspace.com
```

次に、次のいずれかのオプションを使用してワークスペースを開くことができます。

- **oc get devworkspaces** コマンドの出力の **INFO** セクションにある URL にアクセスします。
- OpenShift Dev Spaces ダッシュボードからワークスペースを開きます。

8.1.3. ワークスペースの停止

Devworkspace カスタムリソースの **spec.started** フィールドを **false** に設定することで、ワークスペースを停止できます。

前提条件

- クラスター上のアクティブな **oc** セッション。CLI の使用方法 を参照。
- ワークスペース名は把握している。

  ヒント

  $oc get devworkspaces の出力で、関連するワークスペース名を見つけることができます。

- クラスター上の関連する OpenShift Dev Spaces ユーザー namespace を把握している。

  ヒント

  "https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kubernetes/namespace にアクセスして、OpenShift Dev Spaces ユーザーの namespace を **name** として取得できます。

- クラスターの OpenShift Dev Spaces ユーザー namespace にいる。

  ヒント

  OpenShift では、コマンドライン **oc** ツールを使用して、現在の namespace を表示したり、namespace に切り替えたり できます。

手順

- 次のコマンドを実行して、ワークスペースを停止します。

```
$ oc patch devworkspace <workspace_name> \
-p '{"spec":{"started":false}}' \
--type=merge -n <user_namespace> && \
oc wait --for=jsonpath='{.status.phase}'=Stopped \
dw/<workspace_name> -n <user_namespace>
```

8.1.4. 停止したワークスペースの開始

Devworkspace カスタムリソースの **spec.started** フィールドを **true** に設定することで、停止したワークスペースを開始できます。

前提条件

- クラスター上のアクティブな **oc** セッション。CLI の使用方法 を参照。
- ワークスペース名は把握している。

  ヒント

  $oc get devworkspaces の出力で、関連するワークスペース名を見つけることができます。

- クラスター上の関連する OpenShift Dev Spaces ユーザー namespace を把握している。

ヒント

"https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kubernetes/namespace にアクセスして、OpenShift Dev Spaces ユーザーの namespace を **name** として取得できます。

- クラスターの OpenShift Dev Spaces ユーザー namespace にいる。

ヒント

OpenShift では、コマンドライン **oc** ツールを使用して、現在の namespace を表示したり、namespace に切り替えたり できます。

手順

- 次のコマンドを実行して、停止したワークスペースを開始します。

```
$ oc patch devworkspace <workspace_name> \
-p '{"spec":{"started":true}}' \
--type=merge -n <user_namespace> && \
oc wait --for=jsonpath='{.status.phase}'=Running \
dw/<workspace_name> -n <user_namespace>
```

8.1.5. ワークスペースの削除

DevWorkspace カスタムリソースを削除するだけで、ワークスペースを削除できます。

警告

OpenShift Dev Spaces によって作成された場合、**DevWorkspace** カスタムリソースを削除すると、他のワークスペースリソースも削除されます (たとえば、参照された **DevWorkspaceTemplate** およびワークスペースごとの **PersistentVolumeClaims**)。

ヒント

可能な限り、OpenShift Dev Spaces ダッシュボードを使用してワークスペースを削除します。

前提条件

- クラスター上のアクティブな **oc** セッション。CLI の使用方法 を参照。
- ワークスペース名は把握している。

ヒント

$oc get devworkspaces の出力で、関連するワークスペース名を見つけることができます。

- クラスター上の関連する OpenShift Dev Spaces ユーザー namespace を把握している。

ヒント

"https://devspaces-<openshift_deployment_name>.<domain_name>"/api/kubernetes/namespace にアクセスして、OpenShift Dev Spaces ユーザーの namespace を **name** として取得できます。

- クラスターの OpenShift Dev Spaces ユーザー namespace にいる。

ヒント

OpenShift では、コマンドライン **oc** ツールを使用して、現在の namespace を表示したり、namespace に切り替えたり できます。

手順

- 次のコマンドを実行して、ワークスペースを削除します。

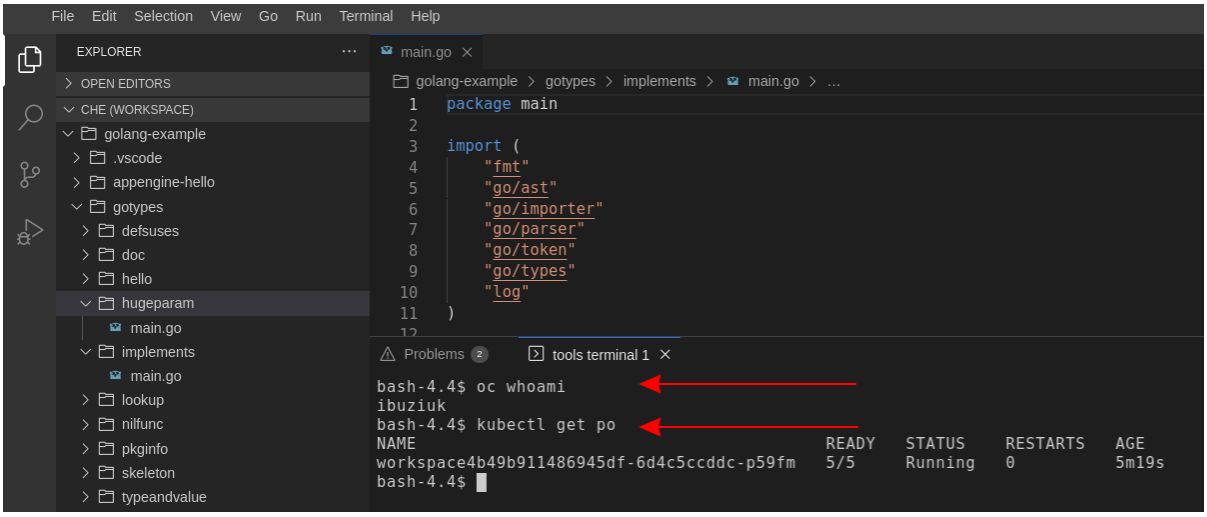
```
$ oc delete devworkspace <workspace_name> -n <user_namespace>
```

8.2. 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。

8.3. OPENSHIFT 開発者の観点からの 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。

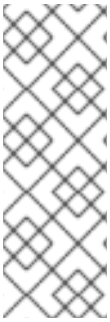
- 既存のコードベース、イメージ、および Dockerfile をインポートして、OpenShift Container Platform でアプリケーションを作成し、デプロイします。
- アプリケーション、コンポーネント、およびプロジェクト内のこれらに関連付けられたサービスと視覚的に対話し、それらのデプロイメントとビルドステータスを監視します。
- アプリケーション内のコンポーネントをグループ化し、アプリケーション内およびアプリケーション間でコンポーネントを接続します。
- サーバーレス機能 (テクノロジープレビュー) を統合します。
- OpenShift Dev Spaces を使用して、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

8.3.1. 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。

8.3.2. 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

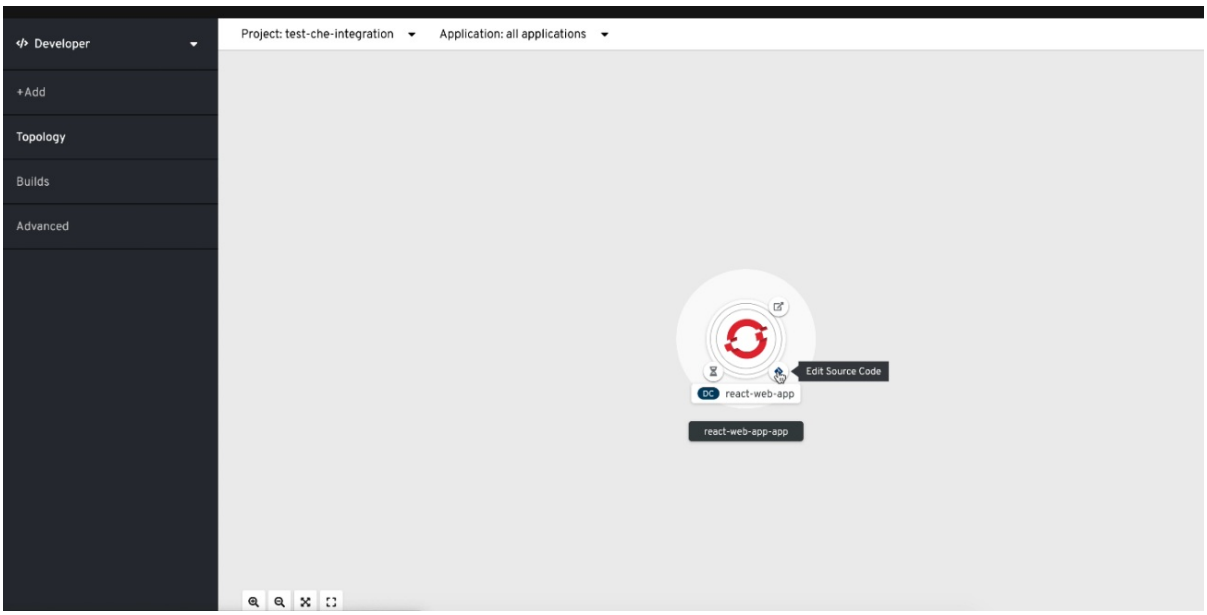
このセクションでは、OpenShift Dev Spaces を使用して OpenShift で実行しているアプリケーションのソースコードの編集を開始する方法を説明します。

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。

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

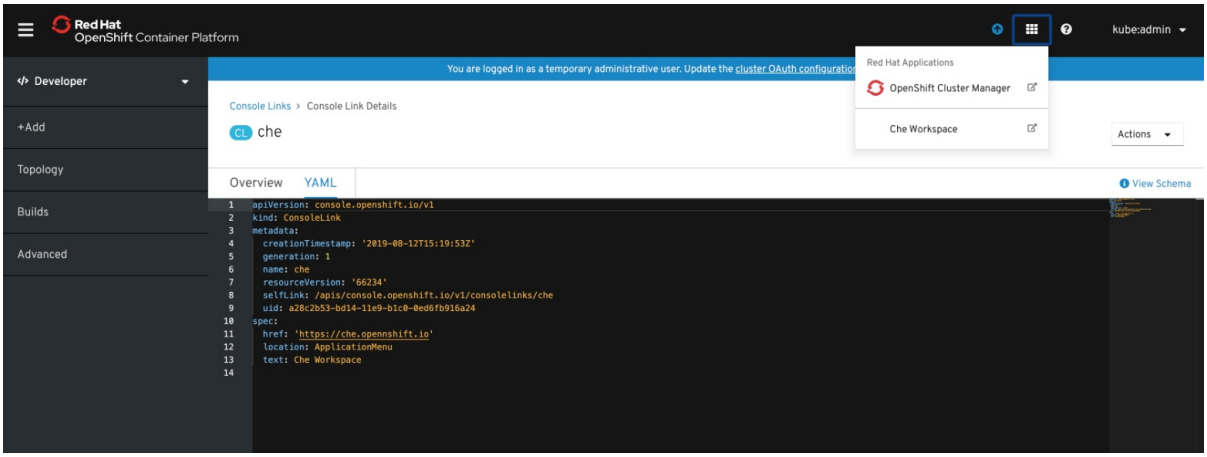
このセクションでは、OpenShift Container Platform の **Red Hat Applications** メニューから OpenShift Dev Spaces ワークスペースにアクセスする方法を説明します。

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。

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

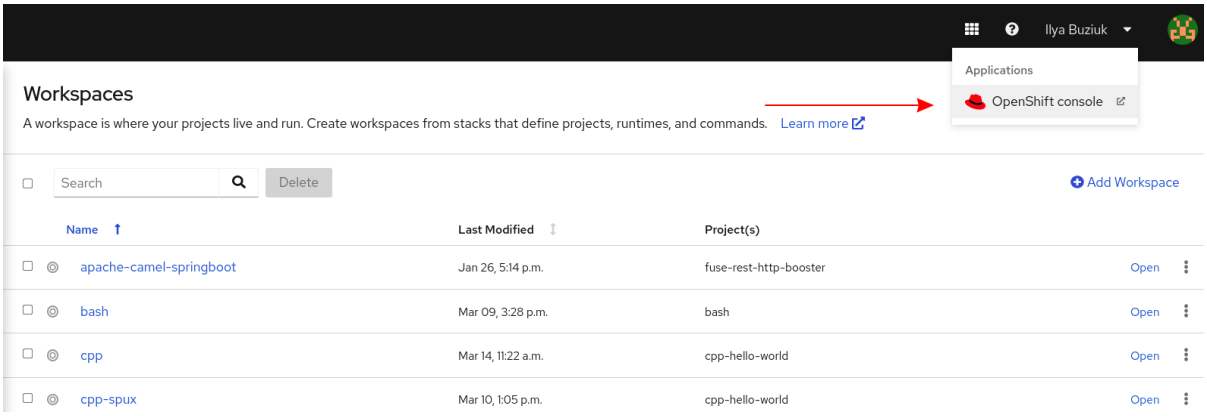
このセクションでは、OpenShift Dev Spaces から OpenShift Web コンソールにアクセスする方法を説明します。

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9章 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

- 「Dev Spaces ワークスペースログの表示」
- 「ワークスペースの開始エラーのトラブルシューティング」
- 「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」

9.1. DEV SPACES ワークスペースログの表示

OpenShift Dev Spaces ログを表示して、問題が発生した場合にバックグラウンドプロセスをよりよく理解し、デバッグすることができます。

IDE 拡張機能が正しく動作しないか、デバッグが必要です

ログには、エディターによって読み込まれたプラグインが一覧表示されます。

コンテナーのメモリーが不足している

ログには、**OOMKilled** エラーメッセージが含まれています。コンテナーで実行されているプロセスが、コンテナーで使用できるように設定されているよりも多くのメモリーを要求しようとしました。

プロセスがメモリー不足になる

ログには、**OutOfMemoryException** などのエラーメッセージが含まれています。コンテナー内のプロセスが、コンテナーが気付かれずにメモリー不足になりました。

関連情報

- 「CLI のワークスペースログ」
- 「OpenShift コンソールの Workspace ログ」
- 「エディターの言語サーバーとデバッグアダプターのログ」

9.1.1. CLI のワークスペースログ

OpenShift CLI を使用して、OpenShift Dev Spaces ワークスペースのログを観察できます。

前提条件

- OpenShift Dev Spaces ワークスペース **<workspace_name>** が実行されています。
- OpenShift CLI セッションは、このワークスペースを含む OpenShift プロジェクト **<namespace_name>** にアクセスできます。

手順

- **<namespace_name>** プロジェクトで **<workspace_name>** ワークスペースを実行している Pod からログを取得します。

```
$ oc logs --follow --namespace='<workspace_namespace>' \
  --selector='controller.devfile.io/devworkspace_name=<workspace_name>'
```

9.1.2. OpenShift コンソールの Workspace ログ

OpenShift コンソールを使用して、OpenShift Dev Spaces ワークスペースのログを観察できます。

手順

1. OpenShift Dev Spaces ダッシュボードで、**Workspaces** に移動します。
2. ワークスペース名をクリックして、ワークスペースの概要ページを表示します。このページには、OpenShift プロジェクト名 **<project_name>** が表示されます。
3. 右上の **Applications** メニューをクリックし、OpenShift コンソールリンクをクリックします。
4. **Administrator** パースペクティブで、OpenShift コンソールで次の手順を実行します。
5. **Workloads** > **Pods** をクリックして、アクティブなすべてのワークスペースのリストを表示します。
6. **Project** ドロップダウンメニューで、**<project_name>** プロジェクトを選択して検索を絞り込みます。
7. ワークスペースを実行する実行中の Pod の名前をクリックします。**Details** タブには、追加情報を含むすべてのコンテナーのリストが含まれています。
8. **Logs** タブに移動します。

9.1.3. エディターの言語サーバーとデバッグアダプターのログ

ワークスペースで実行している Visual Studio Code エディターで、インストールされている言語サーバーとデバッグアダプターの拡張機能を設定して、それらのログを表示できます。

手順

1. 拡張機能を設定します。**File** > **Preferences** > **Settings** をクリックし、**Extensions** セクションを展開して拡張機能を検索し、**trace.server** または同様の設定を **verbose** に設定します (そのような設定が存在する場合)。詳細な設定については、拡張機能のドキュメントを参照してください。
2. **View → Output** をクリックし、出力ビューのドロップダウンリストで言語サーバーを選択して、言語サーバーログを表示します。

関連情報

- VSX レジストリーを開く

9.2. ワークスペースの開始エラーのトラブルシューティング

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。

9.2.1. 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。

9.2.2. 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。

9.3. 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。

9.3.1. 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/3.4/html-single/administration_guide/index#administration-guide: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。Red Hat OpenShift Dev Spaces をオフラインモードでセットアップして、実行時の余分なダウンロードを減らします。これは、最初からすべてを利用できる必要があるためです。https://access.redhat.com/documentation/ja-jp/red_hat_openshift_dev_spaces/3.4/html-single/administration_guide/index#administration-guide:installing-che-in-a-restricted-environment 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/3.4/html-single/administration_guide/index#administration-guide:caching-images-for-faster-workspace-start も併せて参照してください。

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。**vendors.<random_id>.js** または **editor.main.*** は、CDN URL から取得する必要があります。

9.3.2. 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 リソースを追加するとパフォーマンスが向上します。
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: __<plugin_id>__
    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: __<plugin_id>__
    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。

9.4. 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。

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。