



# Red Hat CodeReady Workspaces 2.12

## 管理ガイド

Red Hat CodeReady Workspaces 2.12 の管理



# Red Hat CodeReady Workspaces 2.12 管理ガイド

---

## Red Hat CodeReady Workspaces 2.12 の管理

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

## 法律上の通知

Copyright © 2021 | You need to change the HOLDER entity in the en-US/Administration\_Guide.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

管理者による Red Hat CodeReady Workspaces の運用に関する情報

## 目次

多様性を受け入れるオープンソースの強化 .....	5
<b>第1章 CODEREADY WORKSPACES アーキテクチャーの概要 .....</b>	<b>6</b>
1.1. CODEREADY WORKSPACES ワークスペースコントローラーについて	6
1.1.1. CodeReady Workspaces ワークスペースコントローラー	6
1.1.2. CodeReady Workspaces サーバー	7
1.1.3. CodeReady Workspaces ユーザーダッシュボード	7
1.1.4. CodeReady Workspaces Devfile レジストリー	7
1.1.5. CodeReady Workspaces プラグインレジストリー	8
1.1.6. CodeReady Workspaces および PostgreSQL	8
1.1.7. CodeReady Workspaces および RH-SSO	8
1.2. CODEREADY WORKSPACES ワークスペースアーキテクチャーについて	8
1.2.1. CodeReady Workspaces ワークスペースアーキテクチャー	9
1.2.2. CodeReady Workspaces ワークスペースコンポーネント	10
1.2.2.1. Che Editor プラグイン	10
1.2.2.2. CodeReady Workspaces ユーザーランタイム	11
1.2.2.3. CodeReady Workspaces ワークスペース JWT プロキシ	11
1.2.2.4. CodeReady Workspaces プラグインブローカー	11
1.2.3. CodeReady Workspaces ワークスペース作成フロー	12
<b>第2章 CODEREADY WORKSPACES リソース要件の計算 .....</b>	<b>14</b>
2.1. コントローラーの要件	14
2.2. ワークスペースの要件	14
2.3. ワークスペースの例	18
<b>第3章 レジストリーのカスタマイズ .....</b>	<b>20</b>
3.1. CODEREADY WORKSPACES レジストリーについて	20
3.2. カスタムレジストリーイメージのビルド	20
3.2.1. カスタム devfile レジストリーイメージのビルド	20
3.2.2. カスタムプラグインレジストリーイメージのビルド	22
3.3. カスタムレジストリーの実行	23
3.3.1. OpenShift でのレジストリーのデプロイ	23
3.3.2. 既存の CodeReady Workspaces ワークスペースでのカスタムプラグインレジストリーの追加	25
3.3.2.1. コマンドパレットを使用してカスタムプラグインレジストリーの追加	26
3.3.2.2. settings.json ファイルを使用したカスタムプラグインレジストリーの追加	26
<b>第4章 CODEREADY WORKSPACES ログの取得 .....</b>	<b>28</b>
4.1. サーバーロギングの設定	28
4.1.1. ログレベルの設定	28
4.1.2. ロガーの命名	28
4.1.3. HTTP トラフィックのロギング	28
4.2. OPENSIFT での OPENSIFT イベントへのアクセス	29
4.3. OPENSIFT 4 CLI ツールを使用した CODEREADY WORKSPACES クラスタデプロイメントの状態の表示	29
4.4. CODEREADY WORKSPACES サーバーログの表示	30
4.4.1. OpenShift CLI を使用した CodeReady Workspaces サーバーログの表示	30
4.5. 外部サービスログの表示	31
4.5.1. RH-SSO ログの表示	31
4.5.1.1. RH-SSO サーバーログの表示	31
4.5.1.2. Firefox での RH-SSO クライアントログの表示	31
4.5.1.3. Google Chrome での RH-SSO クライアントログの表示	32
4.5.2. CodeReady Workspaces データベースログの表示	32

4.6. プラグインブローカーログの表示	32
4.7. CRWCTL を使用したログの収集	33
<b>第5章 CODEREADY WORKSPACES の監視</b>	<b>34</b>
5.1. CODEREADY WORKSPACES メトリクスの有効化および公開	34
5.2. PROMETHEUS を使用した CODEREADY WORKSPACES メトリクスの収集	35
<b>第6章 CODEREADY WORKSPACES のトレース</b>	<b>38</b>
6.1. トレース API	38
6.2. バックエンドの追跡	38
6.3. JAEGER トレースツールのインストール	38
6.3.1. OpenShift 4 での OperatorHub を使用した Jaeger のインストール	38
6.3.2. OpenShift 4 での CLI を使用した Jaeger のインストール	39
6.4. メトリクス収集の有効化	40
6.5. JAEGER UI での CODEREADY WORKSPACES トレースの表示	42
6.6. CODEREADY WORKSPACES トレーシングコードベースの概要および拡張ガイド	43
6.6.1. タグ付け	43
<b>第7章 バックアップおよび障害復旧</b>	<b>44</b>
7.1. バックアップサーバーの設定	44
7.2. CRWCTL を使用したバックアップの管理	45
7.2.1. 新規バックアップの作成	45
7.2.2. バックアップからの復元	45
7.3. バックアップサーバーを使用するように CRWCTL を設定	46
7.4. カスタムリソースを使用したバックアップの管理	47
7.4.1. 新規バックアップの作成	47
7.4.2. バックアップからの復元	48
7.5. バックアップサーバーを使用するように CODEREADY WORKSPACES を設定	49
7.5.1. REST サーバーの設定	50
7.5.2. AWS S3 または API 互換サーバーの設定	50
7.5.3. SFTP サーバーの設定	51
7.6. 永続ボリュームのバックアップ	52
7.6.1. 推奨されるバックアップツール: Velero	52
7.7. 外部データベースの設定	52
7.7.1. 外部 PostgreSQL の設定	53
7.7.2. 外部 PostgreSQL と連携するように CodeReady Workspaces を設定する	54
<b>第8章 ワークスペースの起動を迅速化するイメージのキャッシュ</b>	<b>56</b>
8.1. プルするイメージの一覧の定義	57
8.2. IMAGE PULLER のメモリーパラメーターの定義	62
8.3. CODEREADY WORKSPACES OPERATOR を使用した IMAGE PULLER のインストール	63
8.4. OPERATORHUB を使用した OPENSIFT 4 での IMAGE PULLER のインストール	66
8.5. OPENSIFT テンプレートを使用した OPENSIFT への IMAGE PULLER のインストール	67
<b>第9章 ID および承認の管理</b>	<b>73</b>
9.1. ユーザーの認証	73
9.1.1. CodeReady Workspaces サーバーに対する認証	73
9.1.1.1. 他の認証の実装を使用した CodeReady Workspaces サーバーに対する認証	73
9.1.1.2. OAuth を使用した CodeReady Workspaces サーバーに対する認証	73
9.1.1.3. Swagger または REST クライアントを使用したクエリーの実行	74
9.1.2. CodeReady Workspaces ワークスペースでの認証	74
9.1.2.1. セキュアなサーバーの作成	75
9.1.2.2. ワークスペース JWT トークン	75
9.1.2.3. マシントークンの検証	76

---

9.2. ユーザーの認証	76
9.2.1. CodeReady Workspaces ワークスペースパーミッション	77
9.2.2. CodeReady Workspaces システムパーミッション	77
9.2.3. manageSystem パーミッション	78
9.2.4. monitorSystem パーミッション	79
9.2.5. CodeReady Workspaces パーミッションの一覧表示	79
9.2.6. CodeReady Workspaces パーミッションの割り当て	79
9.3. 認証の設定	80
9.3.1. 認証およびユーザー管理	80
9.3.2. RH-SSO と連携する CodeReady Workspaces の設定	80
9.3.3. RH-SSO トークンの設定	81
9.3.4. ユーザーフェデレーションの設定	81
9.3.5. ソーシャルアカウントおよびブローカーを使用した認証の有効化	81
9.3.5.1. GitHub OAuth の設定	81
9.3.5.2. 自己署名 TLS 証明書を使用する Bitbucket サーバーの設定	82
9.3.5.3. OAuth1 を使用するように Bitbucket および CodeReady Workspaces 統合を設定	84
9.3.5.4. GitLab サーバーの設定	87
9.3.5.5. Configuring GitLab OAuth2	87
9.3.6. プロトコルベースのプロバイダーの使用	89
9.3.7. RH-SSO を使用したユーザーの管理	89
9.3.8. 外部 RH-SSO インストールを使用するように CodeReady Workspaces を設定する	89
9.3.9. SMTP およびメール通知の設定	91
9.3.10. 自己登録の有効化	91
9.4. OPENSIFT OAUTH の設定	92
9.4.1. 初期ユーザーでの OpenShift OAuth の設定	92
9.4.2. OpenShift の初期 OAuth ユーザーをプロビジョニングせずに OpenShift OAuth を設定する	93
9.4.3. OpenShift 初期 OAuth ユーザーの削除	93
9.5. ユーザーデータの削除	94
9.5.1. GDPR に準拠したユーザーデータの削除	94





## 多様性を受け入れるオープンソースの強化

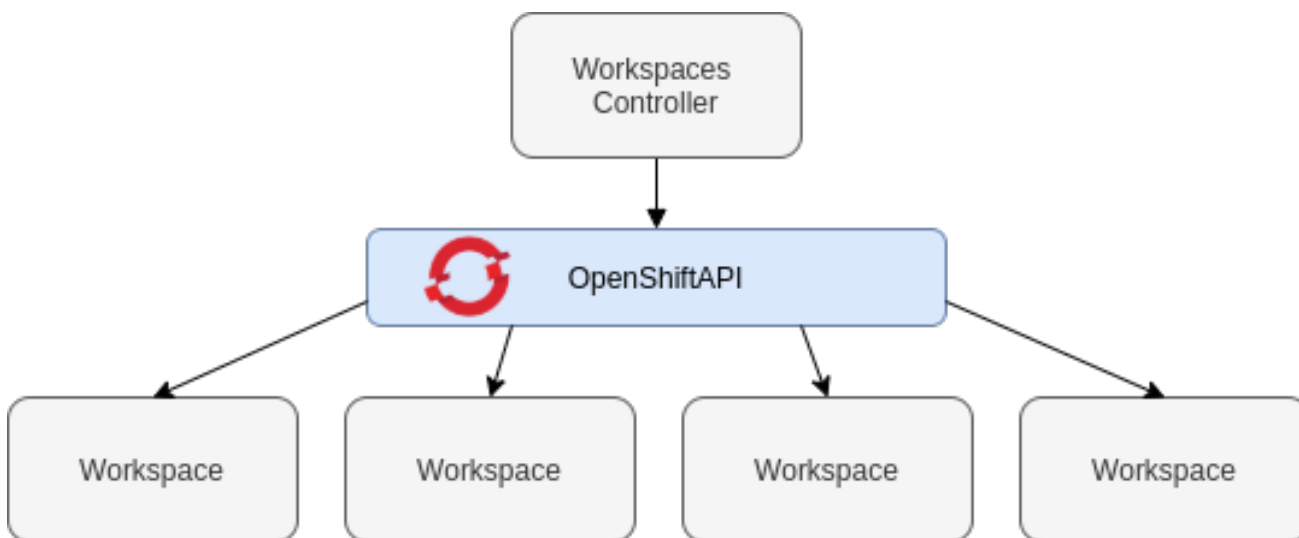
Red Hat では、コード、ドキュメント、Web プロパティにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、[弊社](#) の CTO、Chris Wright の [メッセージ](#) を参照してください。

## 第1章 CODEREADY WORKSPACES アーキテクチャーの概要

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

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

図1.1 CodeReady Workspaces アーキテクチャーの概要



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

### 関連情報

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

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

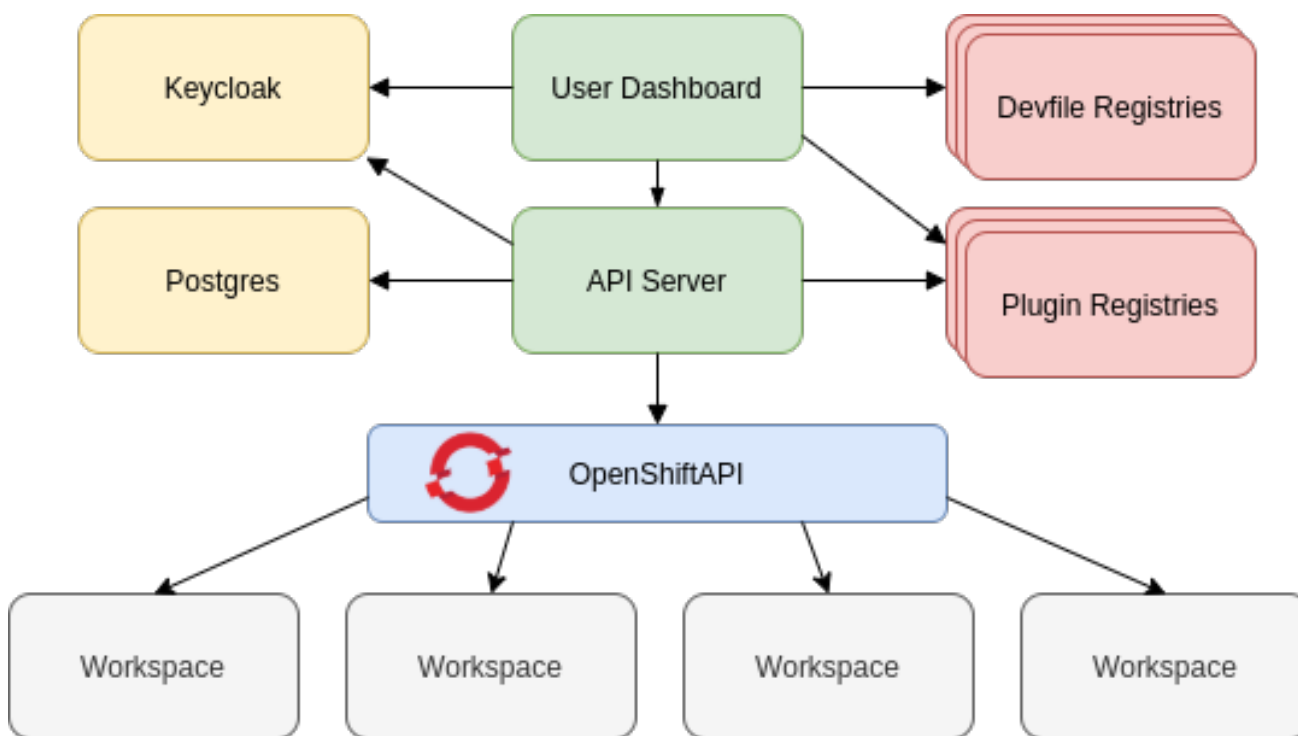
本章では、CodeReady Workspaces コントローラーとコントローラーの一部であるサービスを説明します。

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

ワークスペースコントローラーは、コンテナベースの開発環境 (CodeReady Workspaces ワークスペース) を管理します。認証で開発環境のセキュリティを保護するために、デプロイメントは常にマルチユーザーおよびマルチテナントになります。

以下の図は、CodeReady Workspaces ワークスペースコントローラーの一部である各種サービスを示しています。

図1.2 CodeReady Workspaces ワークスペースコントローラー



#### 関連情報

- [「ユーザーの認証」](#)

### 1.1.2. CodeReady Workspaces サーバー

CodeReady Workspaces サーバーは、ワークスペースコントローラーの中心となるサービスです。これは、HTTP REST API を公開し、CodeReady Workspaces ワークスペースおよび CodeReady Workspaces ユーザーを管理する Java Web サービスです。

コンテナイメージ	<code>eclipse/che-server</code>
----------	---------------------------------

#### 関連情報

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

### 1.1.3. CodeReady Workspaces ユーザーダッシュボード

ユーザーダッシュボードは、Red Hat CodeReady Workspaces のランディングページです。これは Angular フロントエンドアプリケーションです。CodeReady Workspaces ユーザーは、ユーザーダッシュボードでブラウザーから CodeReady Workspaces ワークスペースを作成し、起動し、管理します。

コンテナイメージ	<code>eclipse/che-server</code>
----------	---------------------------------

### 1.1.4. CodeReady Workspaces Devfile レジストリー

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

は、**Dashboard** → **Create Workspace** ウィンドウで使用されます。devfile レジストリーはコンテナで実行され、ユーザーダッシュボードが接続できる任意の場所にデプロイできます。

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

コンテナイメージ	<b>registry.redhat.io/codeready-workspaces/devfileregistry-rhel8:2.12</b>
----------	---

### 1.1.5. CodeReady Workspaces プラグインレジストリー

CodeReady Workspaces プラグインレジストリーは、CodeReady Workspaces ワークスペースのプラグインおよびエディターの一覧を提供するサービスです。devfile は、CodeReady Workspaces プラグインレジストリーに公開されるプラグインのみを参照します。これはコンテナで実行され、CodeReady Workspaces サーバーが接続するすべての場所にデプロイできます。

コンテナイメージ	<b>registry.redhat.io/codeready-workspaces/pluginregistry-rhel8:2.12</b>
----------	--

### 1.1.6. CodeReady Workspaces および PostgreSQL

PostgreSQL データベースは、CodeReady Workspaces を設定するための前提条件です。CodeReady Workspaces 管理者は、CodeReady Workspaces を既存の PostgreSQL インスタンスに接続するか、または CodeReady Workspaces デプロイメントで新規の専用 PostgreSQL インスタンスを起動することを選択できます。

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

コンテナイメージ	<b>registry.redhat.io/rhel8/postgresql-96:1</b>
----------	---

### 1.1.7. CodeReady Workspaces および RH-SSO

RH-SSO は、CodeReady Workspaces を設定するための前提条件です。CodeReady Workspaces 管理者は、CodeReady Workspaces を既存の RH-SSO インスタンスに接続するか、または CodeReady Workspaces デプロイメントで新規の専用 RH-SSO インスタンスを起動することを選択できます。

CodeReady Workspaces サーバーは、OpenID Connect (OIDC) プロバイダーとして RH-SSO を使用して CodeReady Workspaces ユーザーの認証を行い、CodeReady Workspaces リソースへのアクセスのセキュリティを保護します。

コンテナイメージ	<b>registry.redhat.io/rh-ss0-7/sso74-openshift-rhel8:7.4</b>
----------	--

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

本章では、CodeReady Workspaces のアーキテクチャーおよびコンポーネントを説明します。

### 1.2.1. CodeReady Workspaces ワークスペースアーキテクチャー

クラスターの CodeReady Workspaces デプロイメントは、CodeReady Workspaces サーバーコンポーネント、ユーザープロファイルおよび設定を保存するデータベース、およびワークスペースをホストするいくつかの追加のデプロイメントで構成されます。CodeReady Workspaces サーバーは、ワークスペースコンテナと有効にされたプラグイン、以下のような関連するコンポーネントを含むデプロイメントで構成されるワークスペースの作成をオーケストレーションします。

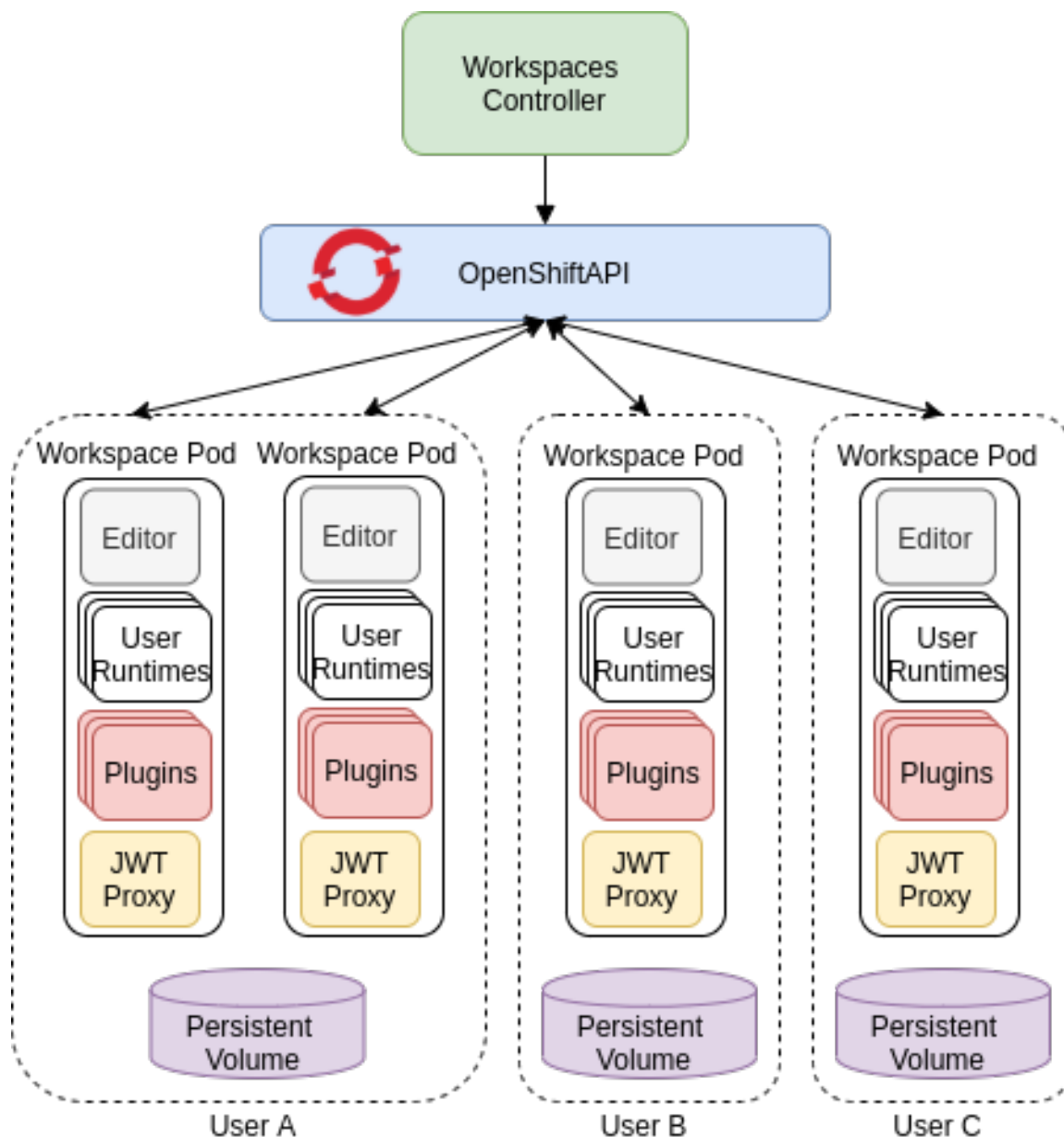
- ConfigMap
- service
- endpoint
- ingress または ルート
- secrets
- 永続ボリューム (PV)

CodeReady Workspaces ワークスペースは Web アプリケーションです。これは、エディター、言語の自動補完、デバッグツールなどの最新の IDE のすべてのサービスを提供するコンテナで実行されるマイクロサービスで構成されます。IDE サービスは、OpenShift リソースとして定義されるコンテナおよびユーザーランタイムアプリケーションにパッケージ化された開発ツールでデプロイされます。

CodeReady Workspaces ワークスペースのプロジェクトのソースコードは、OpenShift **PersistentVolume** で永続化されます。マイクロサービスは、ソースコード (IDE サービス、開発ツール) への読み取り/書き込みアクセスを持つコンテナ内で実行され、ランタイムアプリケーションはこの共有ディレクトリーへの読み取り/書き込みアクセスがあります。

以下の図は、CodeReady Workspaces ワークスペースの詳細なコンポーネントを示しています。

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



この図では、実行中のワークスペースが4つあります。2つは **User A** に属し、1つは **User B** に属し、もう1つは **User C** に属します。

devfile 形式を使用して、CodeReady Workspaces ワークスペースのツールおよびランタイムアプリケーションを指定します。

## 1.2.2. CodeReady Workspaces ワークスペースコンポーネント

本セクションでは、CodeReady Workspaces ワークスペースのコンポーネントについて説明します。

### 1.2.2.1. Che Editor プラグイン

**Che Editor** プラグインは、CodeReady Workspaces ワークスペースプラグインです。これは、ワークスペースでエディターとして使用される Web アプリケーションを定義します。デフォルトの CodeReady Workspaces ワークスペースエディターは [Che-Theia](#) です。これは、[Visual Studio Code](#) (VS Code) と同様の Web ベースのソースコードエディターです。これには、VS Code 拡張機能をサポートするプラグインシステムがあります。

ソースコード	<a href="#">Che-Theia</a>
コンテナイメージ	<b>eclipse/che-theia</b>
エンドポイント	<b>theia、webviews、theia-dev、theia-redirect-1、theia-redirect-2、theia-redirect-3</b>

### 関連情報

- [Che-Theia](#)
- [Eclipse Theia オープンソースプロジェクト](#)
- [Visual Studio Code](#)

#### 1.2.2.2. CodeReady Workspaces ユーザーランタイム

ユーザーランタイムとして終了しないユーザーコンテナを使用します。コンテナイメージまたは OpenShift リソースのセットとして定義できるアプリケーションは、CodeReady Workspaces ワークスペースに追加できます。これにより、CodeReady Workspaces ワークスペースでのアプリケーションのテストが容易になります。

CodeReady Workspaces ワークスペースでアプリケーションをテストするには、ワークスペース仕様にステージまたは実稼働環境で使用するアプリケーションの YAML 定義を含めます。これは、12 要素のアプリケーション開発/実稼働パリティです。

ユーザーランタイムの例は Node.js、SpringBoot または MongoDB、および MySQL です。

#### 1.2.2.3. CodeReady Workspaces ワークスペース JWT プロキシ

JWT プロキシは、CodeReady Workspaces ワークスペースサービスの通信のセキュリティを保護します。

HTTP プロキシは、ワークスペースサービスから CodeReady Workspaces サーバーへの送信要求に署名し、ブラウザで実行されている IDE クライアントからの受信要求を認証するために使用されます。

ソースコード	<a href="#">JWT プロキシ</a>
コンテナイメージ	<b>eclipse/che-jwtproxy</b>

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

プラグインブローカーは、プラグイン **meta.yaml** ファイルに関連した特別なサービスです。以下を実行します。

- CodeReady Workspaces サーバーが認識するプラグイン定義を提供するためにすべての情報を収集します。
- ワークスペースプロジェクトで準備の各種アクションを実行する (ダウンロード、ファイルの展開、プロセス設定)。

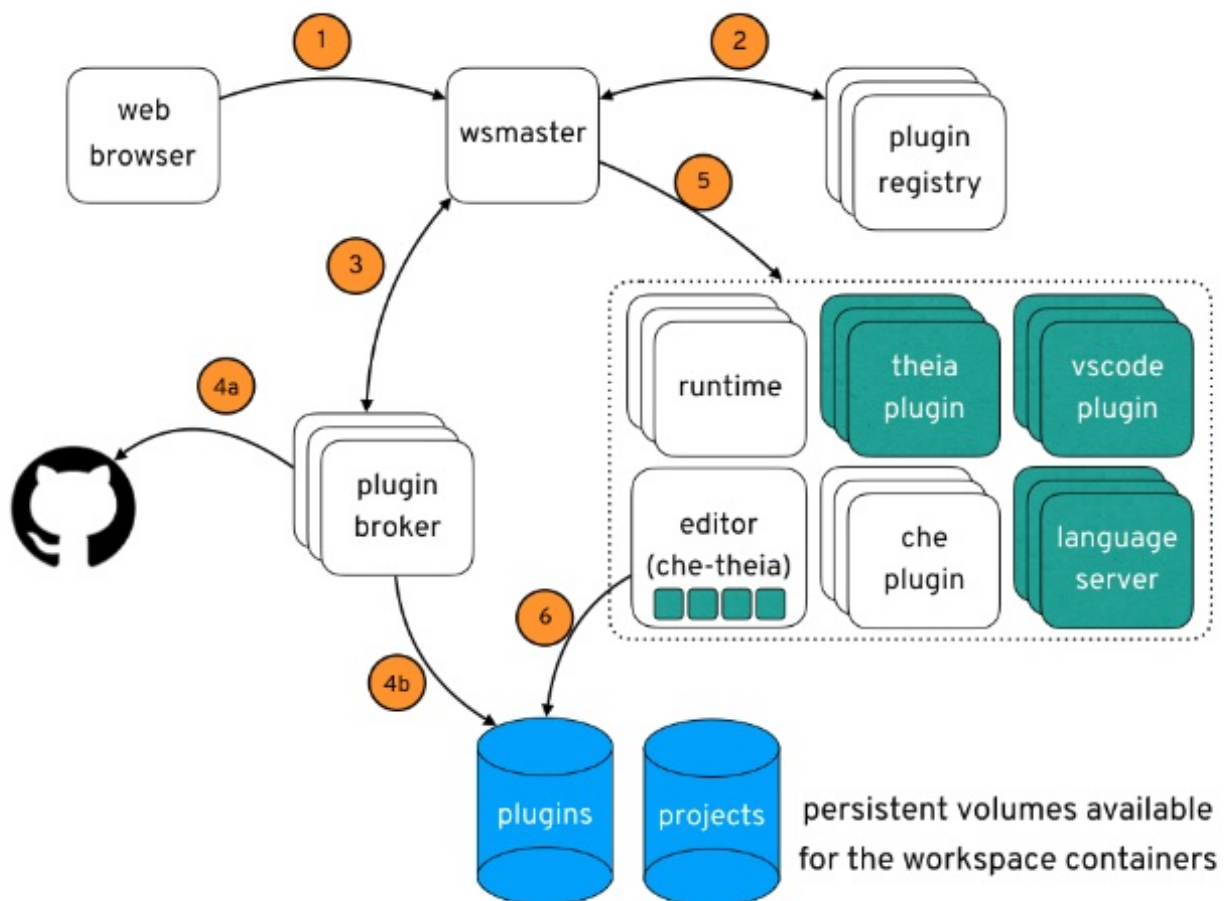
プラグインブローカーの主な目的は、CodeReady Workspaces がサポートできる実際のプラグインから CodeReady Workspaces プラグイン定義を切り離すことにあります。ブローカーでは、CodeReady Workspaces は CodeReady Workspaces サーバーを更新せずに異なるプラグインをサポートできます。

CodeReady Workspaces サーバーはプラグインブローカーを起動します。プラグインブローカーは、ワークスペースと同じ OpenShift プロジェクトで実行されます。これには、プラグインおよびプロジェクトの永続ボリュームへのアクセスがあります。

プラグインブローカーはコンテナイメージとして定義されます (例: **eclipse/che-plugin-broker**)。プラグインタイプは、起動しているブローカーのタイプを判別します。**Che Plugin** および **Che Editor** の 2 種類のプラグインがサポートされます。

ソースコード	CodeReady Workspaces プラグインブローカー
コンテナイメージ	<b>quay.io/eclipse/che-plugin-artifacts-broker</b> <b>eclipse/che-plugin-metadata-broker</b>

### 1.2.3. CodeReady Workspaces ワークスペース作成フロー



以下は、CodeReady Workspaces ワークスペースの作成フローです。

1. ユーザーは、以下によって定義された CodeReady Workspaces ワークスペースを起動します。
  - エディター (デフォルトは Che-Theia)
  - プラグインの一覧 (Java や OpenShift ツールなど)



- ランタイムアプリケーションの一覧
2. CodeReady Workspaces サーバーは、プラグインレジストリーからエディターおよびプラグインメタデータを取得します。
  3. すべてのプラグインタイプに対して、CodeReady Workspaces サーバーは特定のプラグインブローカーを起動します。
  4. CodeReady Workspaces プラグインブローカーは、プラグインのメタデータを Che Plugin 定義に変換します。これは以下の手順を実行します。
    - a. プラグインをダウンロードし、そのコンテンツを展開します。
    - b. プラグインの **meta.yaml** ファイルを処理し、これを Che Plugin の形式で CodeReady Workspaces サーバーに戻します。
  5. CodeReady Workspaces サーバーはエディターとプラグインサイドカーを起動します。
  6. エディターは、プラグインの永続ボリュームからプラグインを読み込みます。

## 第2章 CODEREADY WORKSPACES リソース要件の計算

本セクションでは、Red Hat CodeReady Workspaces の実行に必要なメモリーや CPU などのリソースを計算する方法を説明します。

CodeReady Workspaces 中央コントローラーとユーザーワークスペースはいずれもコンテナのセットで構成されます。これらのコンテナは、CPU および RAM の制限および要求の点でリソース消費に貢献します。

### 2.1. コントローラーの要件

Workspace コントローラーは、5 つの異なるコンテナで実行される 5 つのサービスのセットで構成されます。以下の表は、これらの各サービスのデフォルトのリソース要件を示しています。

表2.1 ControllerServices

Pod	コンテナ名	デフォルトのメモリー制限	デフォルトのメモリー要求
CodeReady Workspaces サーバー およびダッシュボード	che	1 GiB	512 MiB
PostgreSQL	<b>postgres</b>	1 GiB	512 MiB
RH-SSO	<b>keycloak</b>	2 GiB	512 MiB
devfile レジストリー	<b>che-devfile-registry</b>	256 MiB	16 MiB
プラグインレジストリー	<b>che-plugin-registry</b>	256 MiB	16 MiB

これらのデフォルト値は、CodeReady Workspaces Workspace Controller が小規模な CodeReady Workspaces ワークスペースを管理する場合に問題なく使用できます。大規模なデプロイメントの場合は、メモリー制限を増やします。デフォルトの要求および制限を上書きする方法については、[CodeReady Workspaces サーバーコンポーネントの高度な設定オプション](#) を参照してください。たとえば、<https://workspaces.openshift.com> で実行される Red Hat がホストする Eclipse Che は 1GB のメモリーを使用します。

#### 関連情報

- [「CodeReady Workspaces ワークスペースコントローラーについて」](#)

### 2.2. ワークスペースの要件

本セクションでは、ワークスペースに必要なリソースを計算する方法を説明します。これは、このワークスペースの各コンポーネントに必要なリソースの合計です。

以下の例は、適切な計算の必要性について示しています。

- アクティブな 10 のプラグインがあるワークスペースには、これより少ないプラグインを持つ同じワークスペースよりも多くのリソースが必要になります。

- 標準の Java ワークスペースでは、ビルド、テスト、およびアプリケーションのデバッグにより多くのリソースが必要になるため、標準の Node.js ワークスペースよりも多くのリソースが必要になります。

## 手順

1. [Authoring a devfile 2.0.0](#) の **components** セクションで明示的に指定されるワークスペースコンポーネントを特定します。
2. 暗黙的なワークスペースコンポーネントを特定します。
  - a. CodeReady Workspaces はデフォルトの **cheEditor: che-theia** と、コマンドの実行を許可する **chePlugin (che-machine-exec-plugin)** を暗黙的に読み込みます。CodeReady Workspaces がマルチユーザーモードで実行されている場合は、**cheEditor** コンポーネントを読み込みます。
  - b. JWT プロキシコンポーネントは、ワークスペースコンポーネントの外部通信の認証および認可を行います。
3. 各コンポーネントの要件を計算します。
  - a. デフォルト値:  
以下の表は、すべてのワークスペースコンポーネントのデフォルト要件と、対応する CodeReady Workspaces サーバープロパティを示しています。CodeReady Workspaces サーバープロパティを使用して、デフォルトのクラスター全体を変更します。

表2.2 タイプ別のワークスペースコンポーネントのデフォルト要件

コンポーネントのタイプ	CodeReady Workspaces サーバープロパティ	デフォルトのメモ リ制限	デフォルトのメモ リ要求
<b>chePlugin</b>	<b>che.workspace.sidecar.default_memory_limit_mb</b>	128 MiB	64 MiB
<b>cheEditor</b>	<b>che.workspace.sidecar.default_memory_limit_mb</b>	128 MiB	64 MiB
<b>kubernetes、openshift、dockerimage</b>	<b>che.workspace.default_memory_limit_mb、che.workspace.default_memory_request_mb</b>	1 Gi	200 MiB
<b>JWT プロキシ</b>	<b>che.server.secure_exposer.jwtproxy.memory_limit、che.server.secure_exposer.jwtproxy.memory_request</b>	128 MiB	15 MiB

- b. **chePlugins** および **cheEditors** コンポーネントのカスタム要件 :

## i. カスタムメモリー制限および要求:

**meta.yaml** ファイルの **containers** セクションの **memoryLimit** および **memoryRequest** 属性を定義して、**chePlugins** または **cheEditors** コンポーネントのメモリー制限を設定します。CodeReady Workspaces は、明示的に指定されていない場合に、メモリー制限に一致するようにメモリー要求を自動的に設定します。

## 例2.1 chePlugin che-incubator/typescript/latest

## meta.yaml 仕様セクション:

```
spec:
  containers:
    - image: docker.io/eclipse/che-remote-plugin-node:next
      name: vscode-typescript
      memoryLimit: 512Mi
      memoryRequest: 256Mi
```

これにより、コンテナには以下のメモリー制限および要求が設定されます。

メモリー制限	512 MiB
メモリー要求	256 MiB

## 注記

IBM Power Systems (ppc64le) の場合は、一部のプラグインのメモリー制限が最大 1.5G まで増え、Pod が十分な RAM を実行できるようになりました。たとえば、IBM Power Systems (ppc64le) では、Theia エディター Pod には 2G が必要で、OpenShift コネクター Pod には 2.5G が必要です。AMD64 および Intel 64(x86\_64)、および IBM Z(s390x)の場合、メモリー要件は 512M と 1500M とそれぞれ低くなっています。ただし、一部の devfile は、AMD64 および Intel 64 (x86\_64)、ならびに IBM Z (s390x) に有効である低い制限を設定するように設定されます。これを回避するため、デフォルトの memoryLimit を 1-1.5 GB 以上を増やすために、devfile を編集します。

## 注記

## chePluginの meta.yaml ファイルの検索方法

コミュニティープラグインは、**v3/plugins/\${organization}/\${name}/\${version}/** フォルダの [CodeReady Workspaces plug-ins registry repository](#) で利用できます。

コミュニティー以外またはカスタマイズされたプラグインの場合、**meta.yaml** ファイルは **`\${pluginRegistryEndpoint}/v3/plugins/\${organization}/\${name}/\${version}/meta.yaml** のローカルの OpenShift クラスタで利用できます。

## ii. カスタム CPU の制限および要求:

CodeReady Workspaces は、デフォルトで CPU の制限および要求を設定しません。た

CodeReady Workspaces は、デフォルトとして CPU 制限および要求を既定値に設定しています。ただし、**meta.yaml** ファイルまたは devfile で **chePlugin** および **cheEditor** タイプの CPU 制限を、メモリー制限と同じ様に設定できます。

### 例2.2 chePlugin che-incubator/typescript/latest

#### meta.yaml 仕様セクション:

```
spec:
  containers:
    - image: docker.io/eclipse/che-remote-plugin-node:next
      name: vscode-typescript
      cpuLimit: 2000m
      cpuRequest: 500m
```

これにより、コンテナに以下の CPU 制限および要求が設定されます。

CPU 制限	2 コア
CPU 要求	0.5 コア

CPU 制限および要求をグローバルに設定するには、以下の専用の環境変数を使用します。

CPU Limit	CHE_WORKSPACE_SIDECAR_DEFAULT_CPU_LIMIT_CORES
CPU Request	CHE_WORKSPACE_SIDECAR_DEFAULT_CPU_REQUEST_CORES

CodeReady Workspaces サーバーコンポーネントの詳細な設定オプションを参照してください。

OpenShift プロジェクトの **LimitRange** オブジェクトは、クラスター管理者によって設定される CPU 制限および要求のデフォルト値を指定できることに注意してください。リソースのオーバーランによる開始エラーを防ぐために、アプリケーションやワークスペースレベルでの制限がこれらの設定に準拠している必要があります。

a. **dockerimage** コンポーネントのカスタム要件:

devfile の **memoryLimit** 属性と **memoryRequest** 属性を定義して、**dockerimage** コンテナのメモリー制限を設定します。CodeReady Workspaces は、明示的に指定されていない場合に、メモリー制限に一致するようにメモリー要求を自動的に設定します。

```
- alias: maven
  type: dockerimage
  image: eclipse/maven-jdk8:latest
  memoryLimit: 1536M
```

b. **kubernetes** または **openshift** コンポーネントのカスタム要件:

参照されるマニフェストは、メモリー要件および制限を定義できます。

1. 以前に計算された要件をすべて追加します。

## 関連情報

- [「CodeReady Workspaces ワークスペースアーキテクチャーについて」](#)

## 2.3. ワークスペースの例

本セクションでは、CodeReady Workspaces ワークスペースの例を説明します。

以下の devfile は、CodeReady Workspaces ワークスペースを定義します。

```
apiVersion: 1.0.0
metadata:
  generateName: guestbook-nodejs-sample-
projects:
  - name: guestbook-nodejs-sample
    source:
      type: git
      location: "https://github.com/l0rd/nodejs-sample"
components:
  - type: chePlugin
    id: che-incubator/typescript/latest
  - type: kubernetes
    alias: guestbook-frontend
    reference: https://raw.githubusercontent.com/l0rd/nodejs-sample/master/kubernetes-manifests/guestbook-frontend.deployment.yaml
    mountSources: true
  entrypoints:
    - command: ['sleep']
      args: ['infinity']
```

この表は、各ワークスペースコンポーネントのメモリー要件を示しています。

表2.3 ワークスペースメモリー要件および制限の合計

Pod	コンテナ名	デフォルトのメモリー制限	デフォルトのメモリー要求
ワークスペース	theia-ide (デフォルトの <b>cheEditor</b> )	512 MiB	512 MiB
ワークスペース	machine-exec (デフォルトの <b>chePlugin</b> )	128 MiB	128 MiB
ワークスペース	vscode-typescript ( <b>chePlugin</b> )	512 MiB	512 MiB
ワークスペース	frontend ( <b>kubernetes</b> )	1 GiB	512 MiB
JWT プロキシ	verifier	128 MiB	128 MiB
合計		2.25 GiB	1.75 GiB

Pod	コンテナ名	デフォルトの メモリー制限	デフォルトの メモリー要求
-----	-------	------------------	------------------

- devfile に含まれていない場合でも、**theia-ide** および **machine-exec** コンポーネントは暗黙的にワークスペースに追加されます。
- **machine-exec** で必要なリソースは、**chePlugin** のデフォルトです。
- **theia-ide** のリソースは、**cheEditor meta.yaml** で **memoryLimit** が 512 MiB に設定されます。
- Typescript VS Code 拡張は、デフォルトのメモリー制限もオーバーライドします。**meta.yaml** ファイルでは、制限は明示的に 512 MiB に指定されます。
- CodeReady Workspaces は、**kubernetes** コンポーネントタイプのデフォルト（メモリー制限 1 GiB とメモリー要求の 512 MiB）を適用します。これは、**kubernetes** コンポーネントが、リソースの制限や要求のないコンテナ仕様を持つ **Deployment** マニフェストを参照するためです。
- JWT コンテナには 128 MiB のメモリーが必要です。

すべての内容を追加すると、制限が 2.25 GiB の 1.75 GiB のメモリー要求が設定されます。

#### 関連情報

- [1章CodeReady Workspaces アーキテクチャーの概要](#)
- [CodeReady Workspaces インストールの設定](#)
- [CodeReady Workspaces サーバーコンポーネントの詳細な設定オプション](#)
- [devfile 2.0.0 のオーサリング](#)
- [「ユーザーの認証」](#)
- [CodeReady Workspaces プラグインレジストリーリポジトリー](#)

## 第3章 レジストリーのカスタマイズ

本章では、CodeReady Workspaces のカスタムレジストリーをビルドし、実行する方法を説明します。

### 3.1. CODEREADY WORKSPACES レジストリーについて

CodeReady Workspaces は、プラグインレジストリーと devfile レジストリーの 2 つのレジストリーを使用します。これらは、CodeReady Workspaces プラグインおよび devfile のメタデータを公開する静的な Web サイトです。オフラインモードでビルドする場合には、アーティファクトも含まれます。

devfile およびプラグインレジストリーは、2 つの Pod で実行されます。これらのデプロイメントは、CodeReady Workspaces インストールに含まれます。

#### devfile およびプラグインレジストリー

##### devfile レジストリー

devfile レジストリーは、CodeReady Workspaces スタックの定義を保持します。スタックは、**Create Workspace** を選択すると CodeReady Workspaces ユーザーダッシュボードで利用できます。これには、サンプルプロジェクトを含む CodeReady Workspaces の技術的なスタックのサンプルの一覧が含まれます。オフラインモードでビルドする場合には、**zip** ファイルとして devfile で参照されるサンプルプロジェクトがすべて含まれます。

##### プラグインレジストリー

プラグインレジストリーを使用すると、CodeReady Workspaces の同じインスタンスのすべてのユーザーにプラグイン定義を共有できます。オフラインモードでビルドすると、すべてのプラグインまたは拡張アーティファクトも含まれます。

#### 関連情報

- [「カスタムレジストリーイメージのビルド」](#)
- [「カスタムレジストリーの実行」](#)

### 3.2. カスタムレジストリーイメージのビルド

#### 3.2.1. カスタム devfile レジストリーイメージのビルド

本セクションでは、カスタム devfile レジストリーイメージをビルドする方法を説明します。この手順では、devfile を追加する方法を説明します。このイメージには、devfile で参照されるすべてのサンプルプロジェクトが含まれます。

#### 前提条件

- [podman](#) または [docker](#) の実行中のインストール。
- 追加する devfile の有効なコンテンツ。[devfile 2.0.0 の承認](#)を参照してください。

#### 手順

1. devfile レジストリーリポジトリのクローンを作成し、デプロイするバージョンをチェックアウトします。



```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces
$ git checkout crw-2.12-rhel-8
```

2. `./dependencies/che-devfile-registry/devfiles/` ディレクトリーで、サブディレクトリー `<devfile-name>/` を作成し、`devfile.yaml` ファイルおよび `meta.yaml` ファイルを追加します。

#### 例3.1 devfile のファイル編成

```
./dependencies/che-devfile-registry/devfiles/
├── <devfile-name>
│   ├── devfile.yaml
│   └── meta.yaml
```

3. `devfile.yaml` ファイルに有効なコンテンツを追加します。devfile 形式の詳細については、[devfile 2.0.0 の承認](#) を参照してください。
4. `meta.yaml` ファイルが以下の構造に準拠していることを確認します。

表3.1 devfile meta.yaml のパラメーター

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

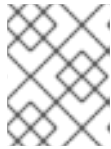
#### 例3.2 devfile の例 meta.yaml

```
displayName: Rust
description: Rust Stack with Rust 1.39
tags: ["Rust"]
icon: https://www.eclipse.org/che/images/logo-eclipseche.svg
globalMemoryLimit: 1686Mi
```

5. カスタム devfile レジストリーイメージをビルドします。

```
$ cd dependencies/che-devfile-registry
$ ./build.sh --organization <my-org> \
  --registry <my-registry> \
```

```
--tag <my-tag>
```



### 注記

**build.sh** スクリプトの詳細なオプションを表示するには **--help** パラメーターを使用します。

### 関連情報

- [devfile 2.0.0 のオーサリング](#)
- [「カスタムレジストリーの実行」](#)

### 3.2.2. カスタムプラグインレジストリーイメージのビルド

本セクションでは、カスタムプラグインレジストリーイメージをビルドする方法を説明します。この手順では、プラグインを追加する方法を説明します。イメージには、プラグインまたは拡張メタデータが含まれます。

### 前提条件

- Node.js 12.x
- yarn の実行中のバージョン。参照: [Installing Yarn](#)
- `./node_modules/.bin` が `PATH` 環境変数にある。
- `podman` または `docker` の実行中のインストール。

### 手順

1. プラグインレジストリーリポジトリのクローンを作成し、デプロイするバージョンをチェックアウトします。

```
$ git clone git@github.com:redhat-developer/codeready-workspaces.git
$ cd codeready-workspaces
$ git checkout crw-2.12-rhel-8
```

2. `./dependencies/che-plugin-registry/` ディレクトリーで、`che-theia-plugins.yaml` ファイルを編集します。
3. `che-theia-plugins.yaml` ファイルに有効なコンテンツを追加します。詳細は、[Adding a VS Code extension to the Che plug-ins registry](#)を参照してください。
4. カスタムプラグインレジストリーイメージをビルドします。

```
$ cd dependencies/che-plugin-registry
$ ./build.sh --organization <my-org> \
  --registry <my-registry> \
  --tag <my-tag>
```



## 注記

**build.sh** スクリプトの詳細なオプションを表示するには **--help** パラメーターを使用します。レジストリーイメージにプラグインバイナリーを含めるには、**--offline** パラメーターを追加します。

- レジストリーのビルド後にコンテナに表示される **./dependencies/che-plugin-registry/v3/plugins/** の内容を確認します。正常なプラグインレジストリービルドから作成されるすべての **meta.yaml** ファイルがここにあります。

```
./dependencies/che-plugin-registry/v3/plugins/
├── <publisher>
│   └── <plugin-name>
│       ├── latest
│       └── meta.yaml
└── latest.txt
```

## 関連情報

- 「[カスタムレジストリーの実行](#)」

## 3.3. カスタムレジストリーの実行

### 前提条件

このセクションで使用される **my-plug-in-registry** イメージおよび **my-devfile-registry** イメージは、**docker** コマンドを使用して構築されます。このセクションでは、これらのイメージが CodeReady Workspaces がデプロイされている OpenShift クラスタで利用できることを想定しています。

これらのイメージは以下にプッシュできます。

- quay.io** または DockerHub などのパブリックコンテナレジストリー。
- プライベートレジストリー

### 3.3.1. OpenShift でのレジストリーのデプロイ

#### 手順

プラグインレジストリーをデプロイする OpenShift テンプレートは、GitHub リポジトリーの **deploy/openshift/** ディレクトリーで利用できます。

- OpenShift テンプレートを使用してプラグインレジストリーをデプロイするには、以下のコマンドを実行します。

```
NAMESPACE=<namespace-name> ①
IMAGE_NAME="my-plug-in-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-plugin-registry.yml \
-n "${NAMESPACE}" \
-p IMAGE="${IMAGE_NAME}" \
-p IMAGE_TAG="${IMAGE_TAG}" \
-p PULL_POLICY="Always"
```

- 1 `crwctl` を使用してインストールされている場合、デフォルトの CodeReady Workspaces プロジェクトは **openshift-workspaces** になります。この OperatorHub のインストール方法
2. `devfile` レジストリーには、GitHub リポジトリの **deploy/openshift/** ディレクトリーに OpenShift テンプレートがあります。これをデプロイするには、以下のコマンドを実行します。

```

NAMESPACE=<namespace-name> 1
IMAGE_NAME="my-devfile-registry"
IMAGE_TAG="latest"
oc new-app -f openshift/che-devfile-registry.yml \
  -n "${NAMESPACE}" \
  -p IMAGE="${IMAGE_NAME}" \
  -p IMAGE_TAG="${IMAGE_TAG}" \
  -p PULL_POLICY="Always"

```

- 1 `crwctl` を使用してインストールされている場合、デフォルトの CodeReady Workspaces プロジェクトは **openshift-workspaces** になります。この OperatorHub のインストール方法では、CodeReady Workspaces を現在のプロジェクトユーザーにデプロイします。

## 検証手順

1. `<plug-in>` プラグインはプラグインレジストリーで利用できます。

例3.3 プラグインレジストリー API を要求する `<plug-in>` を検索します。

```

$ URL=$(oc get route -l app=che,component=plugin-registry \
  -o 'custom-columns=URL:.spec.host' --no-headers)
$ INDEX_JSON=$(curl -sSL http://${URL}/v3/plugins/index.json)
$ echo ${INDEX_JSON} | jq '.[] | select(.name == "<plug-in>")'

```

2. `<devfile>` `devfile` は `devfile` レジストリーで利用できます。

例3.4 `devfile` レジストリー API を要求する `<devfile>` を検索します。

```

$ URL=$(oc get route -l app=che,component=devfile-registry \
  -o 'custom-columns=URL:.spec.host' --no-headers)
$ INDEX_JSON=$(curl -sSL http://${URL}/v3/plugins/index.json)
$ echo ${INDEX_JSON} | jq '.[] | select(.name == "<devfile>")'

```

3. CodeReady Workspaces サーバーはプラグインレジストリーの URL を参照します。

例3.5 `che` ConfigMap の `CHE_WORKSPACE_PLUGIN_REGISTRY_URL` パラメーターの値をプラグインレジストリールートの URL と比較します。

`che` ConfigMap の `CHE_WORKSPACE_PLUGIN_REGISTRY_URL` パラメーターの値を取得します。

```

$ oc get cm/che \
  -o "custom-columns=URL:.data['CHE_WORKSPACE_PLUGIN_REGISTRY_URL']" \
  --no-headers

```

プラグインレジストリールート URL を取得します。

```
$ oc get route -l app=che,component=plugin-registry \
-o 'custom-columns=URL:.spec.host' --no-headers
```

4. CodeReady Workspaces サーバーは devfile レジストリーの URL を参照します。

例3.6 che ConfigMap の `CHE_WORKSPACE_DEVFILE__REGISTRY__URL` パラメーターの値を devfile レジストリールート URL と比較します。

che ConfigMap の `CHE_WORKSPACE_DEVFILE__REGISTRY__URL` パラメーターの値を取得します。

```
$ oc get cm/che \
-o "custom-columns=URL:.data['CHE_WORKSPACE_DEVFILE__REGISTRY__URL']" \
--no-headers
```

devfile レジストリールート URL を取得します。

```
$ oc get route -l app=che,component=devfile-registry \
-o 'custom-columns=URL:.spec.host' --no-headers
```

5. 値が一致しない場合は、ConfigMap を更新し、CodeReady Workspaces サーバーを再起動します。

```
$ oc edit cm/codeready
(...)
$ oc scale --replicas=0 deployment/codeready
$ oc scale --replicas=1 deployment/codeready
```

- プラグインは以下で使用できます。
  - ワークスペースの詳細の Devfile タブの chePlugin コンポーネントの補完
  - ワークスペースの Plugin Che-Theia ビュー
- devfile は、ユーザーダッシュボードの Create Workspace ページの Quick Add および Custom Workspace タブで利用できます。

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

以下のセクションでは、既存の CodeReady Workspaces ワークスペースでのカスタムプラグインレジストリーを追加する 2 つの方法を説明します。

- [コマンドパレットを使用したカスタムプラグインレジストリーの追加](#) - 新しいカスタムプラグインレジストリーを、コマンドパレットコマンドのテキスト入力を使用してすぐに追加します。この方法では、ユーザーがプラグインレジストリーの URL または名前などの既存の情報を編集することはできません。
- [settings.json ファイルを使用してカスタムプラグインレジストリーを追加](#) - 新しいカスタムプラグインレジストリーを追加し、既存のエントリーを編集する場合。

### 3.3.2.1. コマンドパレットを使用してカスタムプラグインレジストリの追加

#### 前提条件

- CodeReady Workspaces のインスタンス

#### 手順

1. CodeReady Workspaces IDE で **F1** キーを押してコマンドパレットを開くか、またはトップメニューで **View → Find Command** に移動します。  
コマンドパレットは、**Ctrl+Shift+p** (または macOS の **Cmd+Shift+p**) を押してアクティブにすることもできます。
2. **Add Registry** コマンドを検索ボックスに入力したら、**Enter** を一度押します。
3. 次の2つのコマンドプロンプトにレジストリー名とレジストリー URL を入力します。
  - 新規プラグインレジストリーの追加後に、**プラグイン** ビューのプラグインの一覧が更新され、新規プラグインレジストリーが有効になっていないと、ユーザーに警告メッセージが表示されます。

### 3.3.2.2. settings.json ファイルを使用したカスタムプラグインレジストリーの追加

以下のセクションでは、メインの CodeReady Workspaces Settings メニューを使用し、**settings.json** ファイルを使用して新規プラグインレジストリーを編集し、追加する方法を説明します。

#### 前提条件

- CodeReady Workspaces のインスタンス

#### 手順

1. メインの CodeReady Workspaces 画面から、**Ctrl+,**を押すか、または左側のバーの歯車アイコンを使用して **Open Preferences** を選択します。
2. **Che Plug-ins** を選択し、**Edit in setting.json** リンクに進みます。  
**setting.json** ファイルが表示されます。
3. 以下のように **chePlugins.repositories** 属性を使用して、新しいプラグインレジストリーを追加します。

```
{
  "application.confirmExit": "never",
  "chePlugins.repositories": {"test": "https://test.com"}
}
```

4. 変更を保存し、既存の CodeReady Workspaces ワークスペースにカスタムプラグインレジストリーを追加します。
  - 新たに追加されたプラグイン検証ツールは、**settings.json** ファイルの **chePlugins.repositories** フィールドに設定された URL 値の正確性をチェックします。
  - 新規プラグインレジストリーの追加後に、**プラグイン** ビューのプラグインの一覧が更新され、新規プラグインレジストリーが有効になっていないと、ユーザーに警告メッセージが

表示されます。このチェックは、コマンドパレットコマンド **Add plugin registry** を使用して追加されたプラグインにも機能します。

## 第4章 CODEREADY WORKSPACES ログの取得

CodeReady Workspaces で各種ログを取得する方法は、以下のセクションを参照してください。

- [「サーバーロギングの設定」](#)
- [「OpenShift での OpenShift イベントへのアクセス」](#)
- [「CodeReady Workspaces サーバーログの表示」](#)
- [「外部サービスログの表示」](#)
- [「プラグインブローカーログの表示」](#)
- [「crwctl を使用したログの収集」](#)

### 4.1. サーバーロギングの設定

CodeReady Workspaces サーバーで利用可能な個別のロガーのログレベルを微調整できます。

CodeReady Workspaces サーバー全体のログレベルは、Operator の **cheLogLevel** 設定プロパティを使用してグローバルに設定されます。Operator によって管理されないインストールでグローバルログレベルを設定するには、**che** ConfigMap で **CHE\_LOG\_LEVEL** 環境変数を指定します。

**CHE\_LOGGER\_CONFIG** 環境変数を使用して、CodeReady Workspaces サーバーで個別のロガーのログレベルを設定できます。

#### 4.1.1. ログレベルの設定

**CHE\_LOGGER\_CONFIG** プロパティの値の形式は、コンマ区切りのキーと値のペアの一覧です。キーは、CodeReady Workspaces サーバーログ出力に表示されるロガーの名前で、値は必要なログレベルになります。

Operator ベースのデプロイメントでは、**CHE\_LOGGER\_CONFIG** 変数はカスタムリソースの **customCheProperties** の下に指定されます。

たとえば、以下のスニペットでは、**WorkspaceManager** が **DEBUG** ログメッセージを生成するようにします。

```
...
server:
  customCheProperties:
    CHE_LOGGER_CONFIG: "org.eclipse.che.api.workspace.server.WorkspaceManager=DEBUG"
```

#### 4.1.2. ロガーの命名

ロガーの名前は、それらのロガーを使用する内部サーバークラスのクラス名に従います。

#### 4.1.3. HTTP トラフィックのロギング

CodeReady Workspaces サーバーと Kubernetes または OpenShift クラスターの API サーバー間の HTTP トラフィックをログに記録できます。これを実行するには、**che.infra.request-logging** ロガーを **TRACE** レベルに設定する必要があります。



```
...
server:
  customCheProperties:
    CHE_LOGGER_CONFIG: "che.infra.request-logging=TRACE"
```

## 4.2. OPENSIFT での OPENSIFT イベントへのアクセス

OpenShift プロジェクトのハイレベルのモニタリングについては、プロジェクトが実行する OpenShift イベントを表示します。

本セクションでは、OpenShift Web コンソールでこれらのイベントにアクセスする方法を説明します。

### 前提条件

- 実行中の OpenShift Web コンソール。

### 手順

1. OpenShift Web コンソールの左側のパネルで、**Home → Events** をクリックします。
2. 特定のプロジェクトのイベントの一覧を表示するには、一覧からプロジェクトを選択します。
3. 現在のプロジェクトのイベントの詳細が表示されます。

### 関連情報

- OpenShift イベントの一覧については、OpenShift ドキュメントの [Comprehensive List of Events in OpenShift documentation](#) を参照してください。

## 4.3. OPENSIFT 4 CLI ツールを使用した CODEREADY WORKSPACES クラスタデプロイメントの状態の表示

本セクションでは、OpenShift 4 CLI ツールを使用して CodeReady Workspaces クラスタのデプロイメントの状態を表示する方法を説明します。

### 前提条件

- OpenShift で実行している Red Hat CodeReady Workspaces のインスタンス。
- OpenShift コマンドラインツール **oc** のインストール

### 手順

1. 以下のコマンドを実行して **crw** プロジェクトを選択します。

```
$ oc project <project_name>
```

2. 以下のコマンドを実行して、選択したプロジェクトで実行されている Pod の名前およびステータスを取得します。

```
$ oc get pods
```

- すべての Pod のステータスが **Running** であることを確認します。

#### 例4.1 ステータスが **Running** の Pod

```

NAME                READY   STATUS    RESTARTS   AGE
codeready-8495f4946b-jrzdc    0/1     Running   0           86s
codeready-operator-578765d954-99szc  1/1     Running   0           42m
keycloak-74fbfb9654-g9vp5     1/1     Running   0           4m32s
postgres-5d579c6847-w6wx5     1/1     Running   0           5m14s

```

- CodeReady Workspaces クラスターデプロイメントの状態を表示するには、以下を実行します。

```
$ oc logs --tail=10 -f `(oc get pods -o name | grep operator)`
```

#### 例4.2 Operator のログ:

```

time="2019-07-12T09:48:29Z" level=info msg="Exec successfully completed"
time="2019-07-12T09:48:29Z" level=info msg="Updating eclipse-che CR with status:
provisioned with OpenShift identity provider: true"
time="2019-07-12T09:48:29Z" level=info msg="Custom resource eclipse-che updated"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: ConfigMap, name:
che"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: ConfigMap, name:
custom"
time="2019-07-12T09:48:29Z" level=info msg="Creating a new object: Deployment,
name: che"
time="2019-07-12T09:48:30Z" level=info msg="Updating eclipse-che CR with status:
CodeReady Workspaces API: Unavailable"
time="2019-07-12T09:48:30Z" level=info msg="Custom resource eclipse-che updated"
time="2019-07-12T09:48:30Z" level=info msg="Waiting for deployment che. Default
timeout: 420 seconds"

```

## 4.4. CODEREADY WORKSPACES サーバーログの表示

本セクションでは、コマンドラインを使用して CodeReady Workspaces サーバーログを表示する方法を説明します。

### 4.4.1. OpenShift CLI を使用した CodeReady Workspaces サーバーログの表示

本セクションでは、OpenShift CLI (コマンドラインインターフェース) を使用して CodeReady Workspaces サーバーログを表示する方法を説明します。

#### 手順

- ターミナルで以下のコマンドを実行し、Pod を取得します。

```
$ oc get pods
```

#### 例

```
$ oc get pods
NAME          READY STATUS RESTARTS AGE
codeready-11-j4w2b 1/1 Running 0      3m
```

2. デプロイメントのログを取得するには、以下のコマンドを実行します。

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

### 例

```
$ oc logs codeready-11-j4w2b
```

## 4.5. 外部サービスログの表示

本セクションでは、CodeReady Workspaces サーバーに関連する外部サービスのログを表示する方法を説明します。

### 4.5.1. RH-SSO ログの表示

RH-SSO OpenID プロバイダーは、サーバーと IDE の 2 つの部分で構成されます。これは診断またはエラー情報を複数のログに書き込みます。

#### 4.5.1.1. RH-SSO サーバーログの表示

このセクションでは、RH-SSO OpenID プロバイダーサーバーのログを表示する方法について説明します。

#### 手順

1. OpenShift Web コンソールで **Deployments** をクリックします。
2. **Filter by label** 検索フィールドに **keycloak** を入力し、RH-SSO ログを表示します。

**Deployment Configs** セクションで、**keycloak** リンクをクリックしてこれを開きます。

1. **History** タブで、アクティブな RH-SSO デプロイメントについての **View log** リンクをクリックします。
2. RH-SSO ログが表示されます。

#### 関連情報

- RH-SSO IDE サーバーに関連する診断およびエラーメッセージについては、「[CodeReady Workspaces サーバーログの表示](#)」を参照してください。

#### 4.5.1.2. Firefox での RH-SSO クライアントログの表示

このセクションでは、Firefox **WebConsole** で RH-SSO IDE クライアント診断またはエラー情報を表示する方法を説明します。

#### 手順

- **Menu > WebDeveloper > WebConsole** をクリックします。

### 4.5.1.3. Google Chrome での RH-SSO クライアントログの表示

このセクションでは、Google Chrome **Console** タブで RH-SSO IDE クライアントの診断またはエラー情報を表示する方法を説明します。

#### 手順

1. **Menu** > **More Tools** > **Developer Tools** の順にクリックします。
2. **Console** タブをクリックします。

### 4.5.2. CodeReady Workspaces データベースログの表示

本セクションでは、PostgreSQL サーバーログなどのデータベースログを CodeReady Workspaces に表示する方法を説明します。

#### 手順

1. OpenShift Web コンソールで **Deployments** をクリックします。
2. **Find by label** 検索フィールドに以下を入力します。
  - **app=che** および **Enter** を押してください。
  - **component=postgres** および **Enter** を押してください。  
OpenShift Web コンソールは、これら 2 つのキーでベースを検索し、PostgreSQL ログを表示するようになりました。
3. **postgres** デプロイメントをクリックして開きます。
4. アクティブな PostgreSQL デプロイメントの **View log** リンクをクリックします。  
OpenShift Web コンソールには、データベースログが表示されます。

#### 関連情報

- PostgreSQL サーバーに関連する診断またはエラーメッセージは、アクティブな CodeReady Workspaces デプロイメントログにある場合があります。アクティブな CodeReady Workspaces デプロイメントログへのアクセスに関する詳細は、[「CodeReady Workspaces サーバーログの表示」](#) セクションを参照してください。

### 4.6. プラグインブローカーログの表示

このセクションでは、プラグインブローカーログを表示する方法を説明します。

**che-plugin-broker** Pod 自体は作業が完了すると削除されます。そのため、イベントログはワークスペースの起動時にのみ利用できます。

#### 手順

一時 Pod からのログイベントを表示するには、以下を実行します。

1. CodeReady Workspaces ワークスペースを起動します。
2. メインの OpenShift Container Platform 画面から、**Workload** → **Pods** に移動します。
3. Pod の **Terminal** タブにある OpenShift ターミナルコンソールを使用します。

## 検証手順

- ワークスペースの起動中に OpenShift ターミナルコンソールがプラグインブローカーログを表示します

## 4.7. CRWCTL を使用したログの収集

**crwctl** ツールを使用して OpenShift クラスターからすべての Red Hat CodeReady Workspaces ログを取得できます。

- **crwctl server:deploy** は、Red Hat CodeReady Workspaces のインストール時に Red Hat CodeReady Workspaces サーバーログの収集を自動的に開始します。
- **crwctl server:logs** は、既存の Red Hat CodeReady Workspaces サーバーログを収集します。
- **crwctl workspace:logs** はワークスペースログを収集します

## 第5章 CODEREADY WORKSPACES の監視

本章では、CodeReady Workspaces を設定してメトリクスを公開する方法と、CodeReady Workspaces でメトリクスとして公開されるデータを処理するために外部ツールを使用してモニタリングスタックのサンプルを構築する方法を説明します。

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

本セクションでは、CodeReady Workspaces メトリクスを有効にし、公開する方法を説明します。

#### 手順

1. che-master ホストで **8087** ポートをサービスとして公開する **CHE\_METRICS\_ENABLED=true** 環境変数を設定します。

Red Hat CodeReady Workspaces が OperatorHub でインストールされると、デフォルトの **CheCluster** CR が使用されている場合に環境変数が自動的に設定されます。

[Eclipse Che](#) > Create Che Cluster

## Create Che Cluster

Create by manually entering YAML or JSON definitions, or by dragging and dropping a file into the editor.

```
1  apiVersion: org.eclipse.che/v1
2  kind: CheCluster
3  metadata:
4    name: eclipse-che
5    namespace: che-metrics
6  spec:
7    server:
8      cheImageTag: nightly
9      devfileRegistryImage: 'quay.io/eclipse/che-devfile-registry:nightly'
10     pluginRegistryImage: 'quay.io/eclipse/che-plugin-registry:nightly'
11     tlsSupport: true
12     selfSignedCert: false
13   database:
14     externalDb: false
15     chePostgresHostName: ''
16     chePostgresPort: ''
17     chePostgresUser: ''
18     chePostgresPassword: ''
19     chePostgresDb: ''
20   auth:
21     openShiftoAuth: true
22     identityProviderImage: 'quay.io/eclipse/che-keycloak:nightly'
23     externalIdentityProvider: false
24     identityProviderURL: ''
25     identityProviderRealm: ''
26     identityProviderClientId: ''
27   storage:
28     pvcStrategy: per-workspace
29     pvcClaimSize: 1Gi
30     preCreateSubPaths: true
31   metrics:
32     enable: true
33
```

```
spec:
  metrics:
    enable: true
```

## 5.2. PROMETHEUS を使用した CODEREADY WORKSPACES メトリクスの収集

このセクションでは、Prometheus モニタリングシステムを使用して、CodeReady Workspaces に関するメトリクスを収集し、保存し、クエリーする方法を説明します。

## 前提条件

- CodeReady Workspaces はポート **8087** でメトリクスを公開している。[che メトリクスの有効化および公開](#)について参照してください。
- Prometheus 2.9.1 以降が実行中である。Prometheus コンソールは、対応する **service** と **route** のあるポート **9090** で実行されている。[Prometheus を初めて実行するための手順](#)について参照してください。

## 手順

- **8087** ポートからメトリクスを収集するように Prometheus を設定する。

### 例5.1 Prometheus 設定の例

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |-
    global:
      scrape_interval: 5s      ❶
      evaluation_interval: 5s  ❷
    scrape_configs:           ❸
      - job_name: 'che'
        static_configs:
          - targets: [['che-host']:8087] ❹
  
```

- ❶ ターゲットが収集されるレート。
- ❷ 記録およびアラートルールを再チェックするレート (現時点ではシステムで使用されていません)。
- ❸ Prometheus モニターするリソース。デフォルト設定では、CodeReady Workspaces サーバーによって公開される時系列データを取得する **che** という単一のジョブがあります。
- ❹ **8087** ポートからメトリクスを収集します。

## 検証手順

- Prometheus コンソールを使用して、メトリクスをクエリーし、表示します。メトリックは、<http://<che-server-url>:9090/metrics>で入手できます。

詳細は、Prometheus ドキュメントの「[Using the expression browser](#)」を参照してください。

## 関連情報

- [Prometheus での最初のステップ](#)
- [Prometheus の設定](#)
- [Prometheus のクエリー](#)



- [Prometheus メトリクスのタイプ](#)

## 第6章 CODEREADY WORKSPACES のトレース

トレースは、マイクロサービスアーキテクチャーのレイテンシーの問題をトラブルシューティングするためにタイミングデータを収集するのに役立ち、分散システムで伝播されるため、完全なトランザクションまたはワークフローを理解するのに役立ちます。すべてのトランザクションでは、新規サービスが独立したチームによって導入されると、早い段階でパフォーマンスの異常を反映する可能性があります。

CodeReady Workspaces アプリケーションの追跡は、ワークスペースの作成、ワークスペースの起動、サブ操作の実行期間の分解、ボトルネックの特定、プラットフォーム全体の状態を改善など、さまざまな操作の実行を分析するのに役立ちます。

トレーサーはアプリケーションに存在します。これらは、発生する操作に関するタイミングとメタデータを記録します。多くの場合、それらはライブラリーをインストルメント化し、使用がユーザーに破棄されるようにします。たとえば、インストルメント化された Web サーバーは、要求の受信時や応答の送信時を記録します。収集されるトレースデータは **スパン** と呼ばれます。スパンには、トレースやスパン識別子などの情報や、次のステップに伝播できる他の種類の情報が含まれるコンテキストがあります。

### 6.1. トレース API

CodeReady Workspaces はインストルメント化に [OpenTracing API](#) (ベンダーに依存しないフレームワーク) を使用します。これは、開発者が別のトレースバックエンドを試す場合、新規の分散トレースシステムのインストルメンテーションプロセスを繰り返すのではなく、開発者は単にトレーサーのバックエンドの設定を変更できることを意味します。

### 6.2. バックエンドの追跡

デフォルトでは、CodeReady Workspaces は Jaeger をトレースバックエンドとして使用します。Jaeger は Dapper および OpenZipkin によって提供され、Uber Technologies によってオープンソースとしてリリースされた分散トレーシングシステムです。Jaeger は、大規模な要求およびパフォーマンスに対応する、より複雑なアーキテクチャーを拡張します。

### 6.3. JAEGER トレースツールのインストール

以下のセクションでは、Jaeger トレーシングツールのインストール方法を説明します。その後、Jaeger は CodeReady Workspaces でメトリクスを収集するために使用できます。

利用可能なインストール方法:

- [「OpenShift 4 での OperatorHub を使用した Jaeger のインストール」](#)
- [「OpenShift 4 での CLI を使用した Jaeger のインストール」](#)

Jaeger を使用して CodeReady Workspaces インスタンスをトレースするには、バージョン 1.12.0 以降が必要になります。Jaeger の詳細は、[Jaeger の Web サイト](#) を参照してください。

#### 6.3.1. OpenShift 4 での OperatorHub を使用した Jaeger のインストール

このセクションでは、実稼働環境でテストおよび評価目的で Jaeger トレースツールを使用する方法についての情報を提供します。

OpenShift Container Platform の OperatorHub インターフェースから Jaeger トレースツールをインストールするには、以下の手順を実行します。

## 前提条件

- ユーザーが OpenShift Container Platform Web コンソールにログインしている。
- CodeReady Workspaces インスタンスはプロジェクトで利用できます。

## 手順

1. OpenShift Container Platform コンソールを開きます。
2. メインの OpenShift Container Platform 画面の左側のメニューから、**Operators** → **OperatorHub** に移動します。
3. **Search by keyword** 検索バーに、**Jaeger Operator** と入力します。
4. **Jaeger Operator** タイルをクリックします。
5. **Jaeger Operator** ポップアップウィンドウで **Install** ボタンをクリックします。
6. インストール方法を選択します。**A specific project on the cluster** の場合、CodeReady Workspaces はデプロイされ、残りをデフォルト値のままにします。
7. **Subscribe** ボタンをクリックします。
8. メインの OpenShift Container Platform 画面の左側のメニューから、**Operators** → **Installed Operators** ページに移動します。
9. Red Hat CodeReady Workspaces は、**InstallSucceeded** ステータスで示唆されるようにインストールされた Operator として表示されます。
10. インストールされた Operator の一覧で、**Jaeger Operator** 名をクリックします。
11. **Overview** タブに移動します。
12. ページ下部の **Conditions** セクションで、メッセージ **install strategy completed with no errors** が表示されるのを待機します。
13. **Jaeger Operator** および追加の **Elasticsearch Operator** がインストールされています。
14. **Operators** → **Installed Operators** セクションに移動します。
15. インストールされた Operator の一覧で **Jaeger Operator** をクリックします。
16. **Jaeger Cluster** ページが表示されます。
17. ウィンドウの左下にある **Create Instance** をクリックします。
18. **保存** をクリックします。
19. OpenShift は Jaeger クラスタ **jaeger-all-in-one-inmemory** を作成します。
20. [メトリクスコレクションの有効化](#) についての手順に従い、以下の手順を完了します。

### 6.3.2. OpenShift 4 での CLI を使用した Jaeger のインストール

このセクションでは、テストおよび評価の目的で Jaeger トレースツールを使用する方法について説明します。

OpenShift Container Platform の CodeReady Workspaces プロジェクトから Jaeger トレースツールをインストールするには、本セクションの手順に従います。

### 前提条件

- ユーザーが OpenShift Container Platform Web コンソールにログインしている。
- OpenShift Container Platform クラスターの CodeReady Workspaces のインスタンス。

### 手順

1. OpenShift Container Platform クラスターの CodeReady Workspaces インストールプロジェクトで、**oc** クライアントを使用して Jaeger デプロイメントの新規アプリケーションを作成します。

```
$ oc new-app -f /${CHE_LOCAL_GIT_REPO}/deploy/openshift/templates/jaeger-all-in-one-template.yml:
```

```
--> Deploying template "<project_name>/jaeger-template-all-in-one" for "/home/user/crw-projects/crw/deploy/openshift/templates/jaeger-all-in-one-template.yml" to project <project_name>
```

```
Jaeger (all-in-one)
```

```
-----
```

```
Jaeger Distributed Tracing Server (all-in-one)
```

```
* With parameters:
```

```
* Jaeger Service Name=jaeger
```

```
* Image version=latest
```

```
* Jaeger Zipkin Service Name=zipkin
```

```
--> Creating resources ...
```

```
deployment.apps "jaeger" created
```

```
service "jaeger-query" created
```

```
service "jaeger-collector" created
```

```
service "jaeger-agent" created
```

```
service "zipkin" created
```

```
route.route.openshift.io "jaeger-query" created
```

```
--> Success
```

```
Access your application using the route: 'jaeger-query-<project_name>.apps.ci-ln-whx0352-d5d6b.origin-ci-int-aws.dev.rhcloud.com'
```

```
Run 'oc status' to view your app.
```

2. メインの OpenShift Container Platform 画面の左側のメニューから **Workloads** → **Deployments** を使用して、Jaeger デプロイメントが正常に終了するまで監視します。
3. メインの OpenShift Container Platform 画面の左側のメニューから **Networking** → **Routes** を選択し、URL リンクをクリックして Jaeger ダッシュボードにアクセスします。
4. [メトリクスコレクションの有効化](#) についての手順に従い、以下の手順を完了します。

## 6.4. メトリクス収集の有効化

### 前提条件

- Jaeger v1.12.0 以降がインストールされている。「[Jaeger トレースツールのインストール](#)」の手順を参照してください。

## 手順

Jaeger トレースを機能させるには、CodeReady Workspaces デプロイメントで以下の環境変数を有効にします。

```
# Activating CodeReady Workspaces tracing modules
CHE_TRACING_ENABLED=true

# Following variables are the basic Jaeger client library configuration.
JAEGER_ENDPOINT="http://jaeger-collector:14268/api/traces"

# Service name
JAEGER_SERVICE_NAME="che-server"

# URL to remote sampler
JAEGER_SAMPLER_MANAGER_HOST_PORT="jaeger:5778"

# Type and param of sampler (constant sampler for all traces)
JAEGER_SAMPLER_TYPE="const"
JAEGER_SAMPLER_PARAM="1"

# Maximum queue size of reporter
JAEGER_REPORTER_MAX_QUEUE_SIZE="10000"
```

次の環境変数を有効にするには、以下を実行します。

1. CodeReady Workspaces デプロイメントの **yaml** ソースコードで、**spec.server.customCheProperties** に以下の設定変数を追加します。

```
customCheProperties:
  CHE_TRACING_ENABLED: 'true'
  JAEGER_SAMPLER_TYPE: const
  DEFAULT_JAEGER_REPORTER_MAX_QUEUE_SIZE: '10000'
  JAEGER_SERVICE_NAME: che-server
  JAEGER_ENDPOINT: 'http://jaeger-collector:14268/api/traces'
  JAEGER_SAMPLER_MANAGER_HOST_PORT: 'jaeger:5778'
  JAEGER_SAMPLER_PARAM: '1'
```

2. **JAEGER\_ENDPOINT** の値を編集して、デプロイメントの Jaeger コレクターサービスの名前に一致するようにします。

メインの OpenShift Container Platform 画面の左側のメニューから、**Networking → Services** に移動して **JAEGER\_ENDPOINT** の値を取得します。または、以下の **oc** コマンドを実行します。

```
$ oc get services
```

要求された値は **collector** 文字列が含まれるサービス名に含まれます。

## 関連情報

- カスタム環境プロパティや、CheCluster カスタムリソースでの定義方法に関する詳細は、[Advanced configuration options for the CodeReady Workspaces server component](#) を参照してください。
- Jaeger のカスタム設定については、[Jaeger クライアント環境変数の一覧](#)を参照してください。

## 6.5. JAEGER UI での CODEREADY WORKSPACES トレースの表示

このセクションでは、Jaeger UI を使用して CodeReady Workspaces 操作の追跡についての概要を示す方法を説明します。

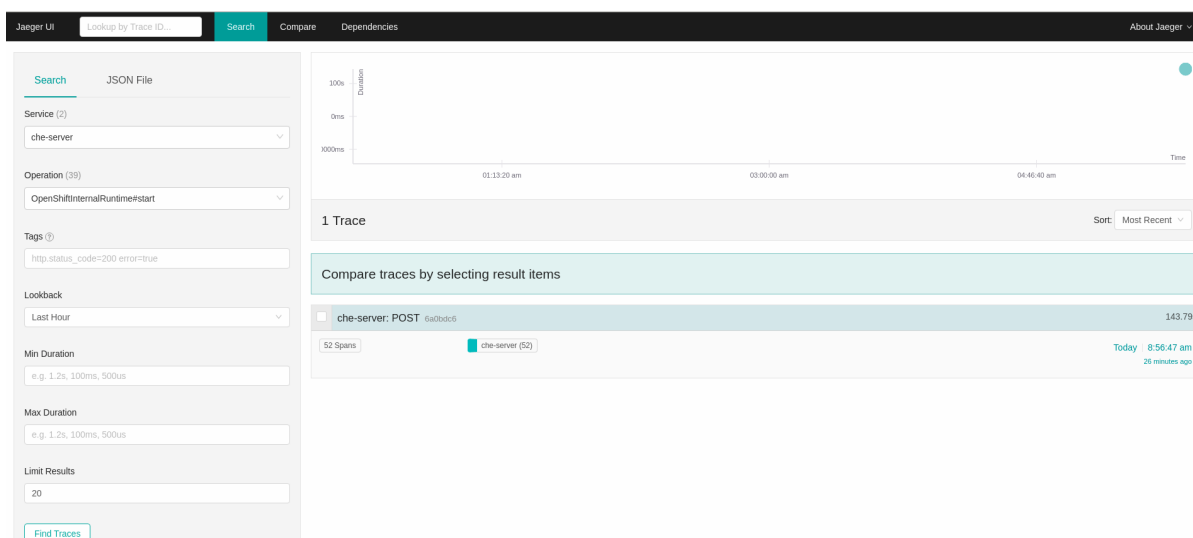
### 手順

この例では、CodeReady Workspaces インスタンスはしばらく実行されており、1つのワークスペースが起動しています。

ワークスペース開始のトレースを検査するには、以下を実行します。

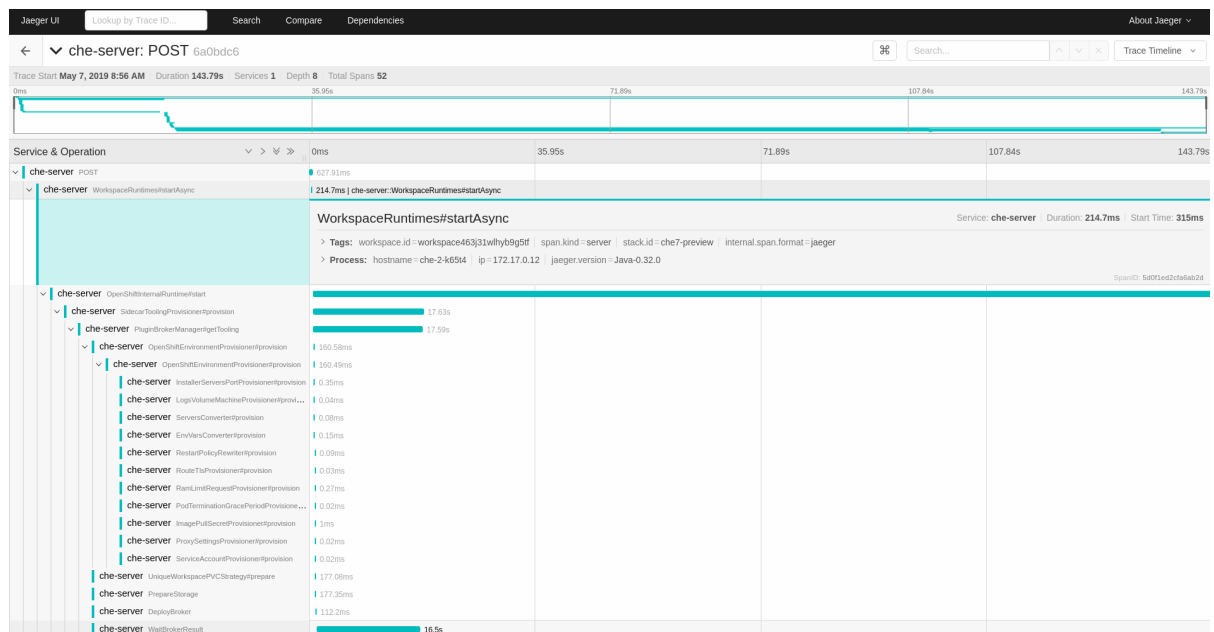
1. 左側の **Search** パネルで、操作名 (スパン名)、タグ、または期間でスパンをフィルターします。

図6.1 Jaeger UI を使用した CodeReady Workspaces の追跡



2. トレースを選択して拡張し、ネストされたスパンのツリーと、タグや期間などの強調表示されたスパンに関する追加情報を表示します。

図6.2 拡張されたトレースツリー



## 6.6. CODEREADY WORKSPACES トレーシングコードベースの概要および拡張ガイド

CodeReady Workspaces のトレース実装のコアの部分は、**che-core-tracing-core** および **che-core-tracing-web** モジュールにあります。

トレース API へのすべての HTTP 要求には独自のトレースがあります。これは、サーバーアプリケーション全体にバインドされる [OpenTracing ライブラリー](#) から **TracingFilter** で実行されます。**@Traced** アノテーションをメソッドに追加すると、**TracingInterceptor** はトレーススパンを追加します。

### 6.6.1. タグ付け

スパンには、操作名、スパンの起点、エラー、およびユーザーのスパンのクエリーやフィルターに役立つその他のタグなど、標準のタグが含まれる場合があります。ワークスペース関連の操作（ワークスペースの開始または停止など）には、**userId**、**workspaceId**、**stackId** などの追加のタグが使用されます。**TracingFilter** によって作成されたスパンには、HTTP ステータスコードタグもあります。

トレースメソッドでのタグの宣言は、**TracingTags** クラスからフィールドを設定して静的に実行されます。

```
TracingTags.WORKSPACE_ID.set(workspace.getId());
```

**TracingTags** は、それぞれの **AnnotationAware** タグ実装のように、一般的に使用されるすべてのタグが宣言されるクラスです。

### 関連情報

Jaeger UI の使用方法についての詳細は、Jaeger ドキュメントの『[Jaeger Getting Started Guide](#)』を参照してください。

## 第7章 バックアップおよび障害復旧

Red Hat CodeReady Workspaces Operator は、CodeReady Workspaces インスタンスのバックアップを作成し、必要な場合はバックアップスナップショットから復元できます。本章では、このようなバックアップを準備し、そのバックアップの使用方法をフォローアップの復旧フェーズで使用方法を説明します。

- [「crwctl を使用したバックアップの管理」](#)
- [「カスタムリソースを使用したバックアップの管理」](#)

### 注意

- CodeReady Workspaces の標準バックアップメカニズムは、ユーザーのワークスペースの内容をバックアップしません。ローカルの変更を保持するには、[「永続ボリュームのバックアップ」](#) を参照してください。
- バックアップスナップショットは独自のクラスターにバインドされ、そこのみを使用する必要があります。

### 関連情報

- [「外部データベースの設定」](#)

## 7.1. バックアップサーバーの設定

以下のセクションでは、サポートされる CodeReady Workspaces バックアップサーバーを説明し、その設定に関する情報を提供します。

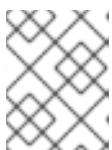


### 注記

- Red Hat CodeReady Workspaces Operator は、同じクラスター内でバックアップサーバーを自動的に設定できますが、実稼働環境での使用には推奨されません。
- CodeReady Workspaces インストールと同じ OpenShift プロジェクト内のデータをバックアップするという決定に起因する制限に同意したユーザーは、このセクションをスキップできます。

CodeReady Workspaces は **restic** ツールを使用して以下を行うようにします。

- バックアップスナップショットの管理
- バックアップサーバーからバックアップデータをプッシュ、またはバックアップサーバーにプルします。



### 注記

**restic** バックアップツールは、[BSD 2-Clause](#) ライセンスでライセンスが適用されています。

現在、CodeReady Workspaces でバックアップサーバーがサポートされています。

### REST



REST サーバーは、**restic** ツールと連携するソリューションです。[REST サーバーのセットアップ方法](#)を参照してください。

### Amazon S3 および API と互換性のある代替手段

[AWS S3 Simple Storage Serviceのドキュメント](#)、またはAWSと互換性のあるAPIを持つ代替サービスのドキュメントを参照してください。

### SFTP

[How to configure an SFTP server](#) を参照してください。

## 7.2. CRWCTL を使用したバックアップの管理

以下のセクションでは、CodeReady Workspaces インストールのバックアップを作成し、`crwctl` を使用して復元を行う方法を説明します。



### 前提条件

- Red Hat CodeReady Workspaces Operator は、同じクラスター内でバックアップサーバーを自動的に設定できますが、実稼働環境での使用には推奨されません。
- CodeReady Workspaces インストールと同じ OpenShift プロジェクト内のデータをバックアップするという決定に起因する制限に同意したユーザーは、このセクションをスキップできます。

- [バックアップサーバーを設定する](#)
- [バックアップサーバーを使用するように `crwctl` を設定する](#)

### 手順

- [「新規バックアップの作成」](#)
- [「バックアップからの復元」](#)

### 7.2.1. 新規バックアップの作成

1. バックアップを作成し、事前に設定されたバックアップサーバーに送信します。

```
$ crwctl server:backup --repository-url=<repository-url> --repository-  
password=<repository-password>
```

- 引数なしで **server:backup** コマンドを使用して、同じバックアップサーバーに他のバックアップを作成できます。
- CodeReady Workspaces のインストール時に最初に引数を指定せずに **server:backup** コマンドを使用すると、内部バックアップサーバーが設定され、使用されます。

### 7.2.2. バックアップからの復元

1. CodeReady Workspaces の直前の状態を復元します。

```
$ crwctl server:restore --repository-url=<repository-url> --repository-  
password=<repository-password>
```

- バックアップサーバーが以前使用されていたときに、**server:restore** コマンドは引数なしで実行できます。
2. ID を指定して特定のスナップショットから復元します。

```
$ crwctl server:restore --snapshot-id=<snapshot-id>
```

### 7.3. バックアップサーバーを使用するように CRWCTL を設定

次のセクションでは、crwctl ツールを使用して、特定のバックアップサーバーの環境変数を定義する方法を説明します。

#### 手順

1. **バックアップサーバータイプ**とサーバー URL を確認します。[restic リポジトリドキュメント](#) は参考として使用してください。  
URL は **-r** パラメーターで指定するか、**BACKUP\_REPOSITORY\_URL** 環境変数を使用して定義できます。
2. バックアップリポジトリのパスワードを取得または作成します。  
パスワードは、**-p** パラメーターで指定するか、**BACKUP\_REPOSITORY\_PASSWORD** 環境変数を使用して定義できます。



#### 警告

バックアップデータは、このパスワードで暗号化されます。バックアップリポジトリのパスワードが失われると、データが失われます。

3. 選択した **バックアップサーバータイプ**に、以下の環境変数を設定します。

#### REST

任意の認証が有効な場合は、**REST\_SERVER\_USERNAME** および **REST\_SERVER\_PASSWORD** 環境変数をエクスポートします。

#### AWS S3

AWS ユーザー認証情報で **AWS\_ACCESS\_KEY\_ID** および **AWS\_SECRET\_ACCESS\_KEY** 環境変数をエクスポートします。

#### SFTP

パスワードなしのログインの場合は、対応する SSH 鍵を持つファイルへのパスを保持する **SSH\_KEY\_FILE** 環境変数をエクスポートするか、**--ssh-key-file** パラメーターを指定します。

または、SSH キー自体を保持する **SSH\_KEY** 環境変数を使用することもできます。



## 注記

--**backup-server-config-name** パラメーターまたは **BACKUP\_SERVER\_CONFIG\_NAME** 環境変数を使用して、バックアップサーバー設定オブジェクトを直接参照することができます。この場合、上記のすべての設定は必要ありません。詳細は、「[カスタムリソースを使用したバックアップの管理](#)」を参照してください。

## 7.4. カスタムリソースを使用したバックアップの管理

本セクションでは、CodeReady Workspaces インストールのバックアップを作成し、カスタムリソースオブジェクトを使用して直接復元する方法を説明します。



### 前提条件

- Red Hat CodeReady Workspaces Operator は、同じクラスター内でバックアップサーバーを自動的に設定できますが、実稼働環境での使用には推奨されません。
- CodeReady Workspaces インストールと同じ OpenShift プロジェクト内のデータをバックアップするという決定に起因する制限に同意したユーザーは、このセクションをスキップできます。

- [バックアップサーバーの設定](#)
- [バックアップサーバーを使用するように crwctl を設定](#)。

### 手順

- [「新規バックアップの作成」](#)
- [「バックアップからの復元」](#)

### 7.4.1. 新規バックアップの作成

1. **CheClusterBackup** オブジェクトを作成し、新規バックアップを作成します。

```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: CodeReady Workspaces-backup
spec:
  backupServerConfigRef: backup-server-configuration 1
```

- 1 使用するバックアップサーバーを定義する **CheBackupServerConfiguration** オブジェクトの名前。

- **CheClusterBackup** オブジェクトの作成は、新規バックアップを開始します。
- 新規バックアップオブジェクトに同じ名前を再利用する前に、古いオブジェクトを削除します。

```
oc delete CheClusterBackup <name> -n openshift-workspaces
```



## 注記

**CheClusterBackup** オブジェクトの編集は機能しません。

## 代替方法

内部バックアップサーバーを使用するには、CodeReady Workspaces Operator から自動設定を要求します。上記の準備は必要ありません。

- 自動設定を構成し、バックアップを内部バックアップサーバーに送信します。

```
apiVersion: org.eclipse.che/v1
kind: CheClusterBackup
metadata:
  name: CodeReady Workspaces-backup
spec:
  useInternalBackupServer: true
```

## 7.4.2. バックアップからの復元

1. **CheClusterRestore** の新規オブジェクトを作成し、バックアップから CodeReady Workspaces インストールを復元します。

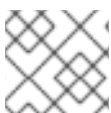
```
apiVersion: org.eclipse.che/v1
kind: CheClusterRestore
metadata:
  name: CodeReady Workspaces-restore
spec:
  backupServerConfigRef: backup-server-configuration 1
  snapshotId: ba92c7e0 2
```

- 1** 使用するバックアップサーバーを定義する **CheBackupServerConfiguration** オブジェクトの名前。
- 2** 復元元となるスナップショット ID を定義するオプションパラメーター。デフォルト値は、バックアップサーバーの最後のスナップショットです。

1. 新規リカバリーを要求する新しい **CheClusterRestore** オブジェクトを作成します。
  - 新規バックアップオブジェクトに同じ名前を再利用する前に、古いオブジェクトを削除します。

```
oc delete CheClusterBackup <name> -n openshift-workspaces
```

2. リカバリープロセスが完了するまで待ちます。  
リカバリー後にブラウザーでエラーが発生した場合は、CodeReady Workspaces ドメインのブラウザーデータをクリーンアップします。



## 注記

**CheClusterRestore** オブジェクトの編集は機能しません。

## 検証

## 1. バックアッププロセスの状態を確認します。

- a. **CheClusterBackup** オブジェクトの **status** セクションを読み取り、バックアッププロセスを確認します。

```
status:
  message: 'Backup is in progress. Start time: <timestamp>' 1
  stage: Collecting CodeReady Workspaces installation data 2
  state: InProgress 3
  snapshotId: ba92c7e0 4
```

- 1 全体の状態またはエラーメッセージを表示します。
- 2 バックアッププロセスの現在のフェーズは、人間が判読できる形式で行います。
- 3 バックアッププロセスの状態。 **InProgress**、 **Succeeded**、 または **Failed** のいずれか。
- 4 作成されたバックアップスナップショットの ID。フィールドは、 **state** が **Succeeded** の場合にのみ表示されます。

## 2. 復元プロセスの状態の確認

- a. **CheClusterRestore** オブジェクトの **status** セクションを読み取り、復元プロセスを確認します。

```
status:
  message: 'Restore is in progress. Start time: <timestamp>' 1
  stage: Restoring CodeReady Workspaces related cluster objects 2
  state: InProgress 3
```

- 1 全体的な状態またはエラーメッセージ
- 2 人間が判読できる形式で復元プロセスの現在のフェーズ。
- 3 復元プロセスの状態。 **InProgress**、 **Succeeded**、 または **Failed** のいずれか。

## 7.5. バックアップサーバーを使用するように CODEREADY WORKSPACES を設定

CodeReady Workspaces のバックアップサーバーを設定するには、ユーザーは `openshift-workspaces` namespace に **CheBackupServerConfiguration** カスタムリソースオブジェクトを作成する必要があります。オブジェクトの **spec** プロパティは複数のセクションに分割され、それぞれは特定の [バックアップサーバータイプ](#) に対応します。

- [REST](#)
- [AWS S3 または API 互換](#)
- [SFTP](#)



## 注記

- **openshift-workspaces** namespace に保存されるカスタムリソースオブジェクトには、**spec** プロパティに1つのセクションのみを設定する必要があります。
- 必要な数だけバックアップサーバーを設定できますが、各バックアップサーバーは個別の **Custom Resource** で設定できます。
- 各サーバータイプの参照シークレットが存在し、必須フィールドを指定する必要があります。対応するサーバータイプの章で各シークレットの説明を参照してください。

### 7.5.1. REST サーバーの設定

```

apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
spec:
  rest:
    protocol: http
    hostname: my-domain.net
    port: 1234
    repositoryPath: CodeReady Workspaces-backups
    repositoryPasswordSecretRef: backup-encryption-password-secret
    credentialsSecretRef: rest-server-auth-secret

```

- 1 使用するプロトコルを指定する任意のプロパティ。デフォルト値は**https**で、2番目に許可されるオプションは**http**です。
- 2 バックアップサーバーのホスト名。
- 3 バックアップサーバーが実行しているポートを指定する任意のプロパティ。デフォルト値は**8000**です。
- 4 バックアップスナップショットが保存されるバックアップサーバーのパス。
- 5 **repo-password** フィールドに保存されるリポジトリパスワードを含むシークレット名。シークレットに1つのフィールドのみが含まれる場合、その名前は任意です。パスワードは、バックアップスナップショットデータを暗号化および復号するために使用されます。
- 6 **username** フィールドおよび **password** フィールドに保存されている REST サーバーユーザー認証情報でシークレットの名前を指定する任意のプロパティ。

### 7.5.2. AWS S3 または API 互換サーバーの設定

```

apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
spec:
  awss3:

```

```

protocol: https
hostname: my-domain.net
port: 1234
repositoryPath: CodeReady Workspaces-backups
repositoryPasswordSecretRef: backup-encryption-password-secret
awsAccessKeyIdSecretRef: aws-user-credentials-secret

```

- ① 使用するプロトコルを指定する任意のプロパティ。デフォルト値は**https**で、2番目に許可されるオプションは**http**です。
- ② S3 ホスト名を指定する任意のプロパティ。デフォルト値は **s3.amazonaws.com** です。
- ③ バックアップサーバーが実行しているポートを指定する任意のプロパティ。
- ④ バックアップスナップショットが保存されるバケットリソースの名前。バケットリソースは手動で事前に作成する必要があります。
- ⑤ **repo-password** フィールドに保存されるリポジトリパスワードが含まれるシークレットの名前。シークレットに1つのフィールドのみが含まれる場合、この名前は任意です。パスワードは、バックアップスナップショットデータを暗号化および復号するために使用されます。
- ⑥ **awsAccessKeyId** フィールドおよび **awsSecretAccessKey** フィールドに保存されているユーザー認証情報が含まれるシークレットの名前。

### 7.5.3. SFTP サーバーの設定

```

apiVersion: org.eclipse.che/v1
kind: CheBackupServerConfiguration
metadata:
  name: backup-server-configuration
spec:
  awss3:
    username: user
    hostname: my-domain.net
    port: 1234
    repositoryPath: CodeReady Workspaces-backups
    repositoryPasswordSecretRef: backup-encryption-password-secret
    sshKeySecretRef: ssh-key-secret

```

- ① SSH プロトコルを使用してログインするためのリモートサーバーのユーザー名。
- ② リモートサーバーのホスト名。
- ③ SFTP サーバーが実行しているポートを指定する任意のプロパティ。デフォルト値は**22**です。
- ④ バックアップスナップショットが保存されるサーバー上の絶対パスまたは相対パス。
- ⑤ **repo-password** フィールドに保存されるリポジトリパスワードが含まれるシークレットの名前。シークレットに1つのフィールドのみが含まれる場合、この名前は任意です。パスワードは、バックアップスナップショットデータを暗号化および復号するために使用されます。
- ⑥ **ssh-privatekey** フィールドに保存される SSH 秘密鍵が含まれるシークレットの名前。この SSH キーは、SFTP サーバーにパスワードなしでログインを実行するために使用できます。

## 7.6. 永続ボリュームのバックアップ

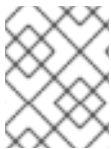
永続ボリューム (PV) は、ローカルのハードドライブのデスクトップ IDE 用にワークスペースのデータを保存する方法と同様に、CodeReady Workspaces ワークスペースデータを保存します。

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

### 7.6.1. 推奨されるバックアップツール: Velero

Velero は、OpenShift アプリケーションおよびそれらの PV をバックアップするオープンソースツールです。Velero を使用すると、以下を実行できます。

- クラウドまたはオンプレミスでデプロイします。
- データ損失が発生した場合にクラスターをバックアップし、復元します。
- クラスタリソースを他のクラスターに移行します。
- 実稼働クラスターを開発およびテスト用に複製します。



#### 注記

または、基礎となるストレージシステムに依存するバックアップソリューションを使用できます。たとえば、Gluster や Ceph 固有のソリューションなどがこれに含まれます。

#### 関連情報

- [永続ボリュームのドキュメント](#)
- [Gluster ドキュメント](#)
- [Ceph ドキュメント](#)
- [Velero on GitHub](#)

## 7.7. 外部データベースの設定

PostgreSQL データベースは、CodeReady Workspaces の状態に関するデータを永続化させるために、CodeReady Workspaces サーバーによって使用されます。これには、ユーザーアカウント、ワークスペース、設定についての情報、およびその他の詳細情報が含まれます。

デフォルトで、CodeReady Workspaces Operator はデータベースデプロイメントを作成し、管理します。

ただし、CodeReady Workspaces Operator はバックアップやリカバリーなどの完全なライフサイクル機能をサポートしません。

ビジネスに不可欠な環境では、以下の推奨される障害復旧オプションを使用して外部データベースを設定します。

- 高可用性 (HA)
- PITR (Point In Time Recovery)



オンプレミスの外部 PostgreSQL インスタンスを設定するか、または Amazon Relational Database Service (Amazon RDS) などのクラウドサービスを使用します。Amazon RDS を使用すると、通常のおよびオンデマンドのスナップショットを使用して、回復性のある障害復旧ストラテジーの Multi-Availability Zone 設定で実稼働データベースをデプロイできます。

データベースサンプルの設定例は以下のようになります。

パラメーター	値
インスタンスクラス	db.t2.small
vCPU	1
RAM	2 GB
Multi-az	true、2つのレプリカ
エンジンのバージョン	9.6.11
TLS	enabled
自動化されたバックアップ	有効 (30 日)

### 7.7.1. 外部 PostgreSQL の設定

外部 PostgreSQL を設定すると、ワークスペースメタデータおよびユーザー情報を永続化できます。

#### 手順

- 以下のプレースホルダーの値を定義します。
  - <database-user>** は、CodeReady Workspaces サーバーデータベースのユーザー名です。
  - <database-password>** は、CodeReady Workspaces サーバーデータベースのパスワードです。
  - <database>** は、CodeReady Workspaces サーバーデータベースの名前です。
- 以下の SQL スクリプトを使用して、CodeReady Workspaces サーバーのユーザーおよびデータベースを作成し、ワークスペースのメタデータを永続化します。

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

- 以下のプレースホルダーの値を定義します。
  - <identity-database-password>** は RH-SSO データベースのパスワードです。
- 以下の SQL スクリプトを使用して、RH-SSO バックエンドのデータベースを作成して、ユーザー情報を永続化します。

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

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

### 前提条件

- **oc** ツールが利用できる。

### 手順

1. CodeReady Workspaces のプロジェクトを事前に作成します。

```
$ oc create namespace openshift-workspaces
```

2. CodeReady Workspaces サーバーデータベースの認証情報を保存するためにシークレットを作成します。

```
$ oc create secret generic <server-database-credentials> \ 1
--from-literal=user=<database-user> \ 2
--from-literal=password=<database-password> \ 3
-n openshift-workspaces
```

- 1** CodeReady Workspaces サーバーデータベースの認証情報を保存するためのシークレットの名前
- 2** CodeReady Workspaces サーバーデータベースのユーザー名
- 3** CodeReady Workspaces サーバーデータベースのパスワード

3. RH-SSO データベース認証情報を保存するためのシークレットを作成します。

```
$ oc create secret generic <identity-database-credentials> \ 1
--from-literal=password=<identity-database-password> \ 2
-n openshift-workspaces
```

- 1** RH-SSO データベースの認証情報を保存するためのシークレット名
- 2** RH-SSO データベースのパスワード

4. パッチを適用して **crwctl** コマンドを実行し、Red Hat CodeReady Workspaces をデプロイします。例:

```
$ crwctl server:deploy --che-operator-cr-patch-yaml=patch.yaml ...
```

patch.yaml には、Operator がデータベースのデプロイを省略し、既存のデータベースの接続詳細を CodeReady Workspaces サーバーに渡すことができるようにするために以下が含まれる必要があります。

```
spec:
```

```
database:
  externalDb: true
  chePostgresHostName: <hostname>
  chePostgresPort: <port>
  chePostgresSecret: <server-database-credentials>
  chePostgresDb: <database>
spec:
  auth:
    identityProviderPostgresSecret: <identity-database-credentials>
```

- ① 外部データベースのホスト名
- ② 外部データベースポート
- ③ CodeReady Workspaces サーバーデータベースの認証情報を含むシークレットの名前
- ④ CodeReady Workspaces サーバーデータベースの名前
- ⑤ RH-SSO データベースの認証情報が含まれるシークレットの名前

#### 関連情報

- [PostgreSQL](#)
- [RDS](#)

## 第8章 ワークスペースの起動を迅速化するイメージのキャッシュ

CodeReady Workspaces ワークスペースの起動時間のパフォーマンスを改善するには、Image Puller を使用して OpenShift クラスターのイメージの事前プルに使用できる CodeReady Workspaces に依存しないコンポーネントを使用します。Image Puller は、関連する CodeReady Workspaces ワークスペースイメージを各ノードで事前にプルするように設定できる **DaemonSet** を作成する追加の OpenShift デプロイメントです。これらのイメージは、CodeReady Workspaces ワークスペースの起動時にすでに利用可能なため、ワークスペースの開始時間が改善されています。

Image Puller は、設定用に以下のパラメーターを提供します。

表8.1 Image Puller パラメーター

パラメーター	使用法	デフォルト
<b>CACHING_INTERVAL_HOURS</b>	デーモンセットのヘルスチェック間隔（時間単位）	"1"
<b>CACHING_MEMORY_REQUEST</b>	Puller の実行時にキャッシュされる各イメージのメモリー要求。 <a href="#">「Image Puller のメモリーパラメーターの定義」</a> を参照してください。	10Mi
<b>CACHING_MEMORY_LIMIT</b>	Puller の実行時にキャッシュされる各イメージのメモリー制限。 <a href="#">「Image Puller のメモリーパラメーターの定義」</a> を参照してください。	20Mi
<b>CACHING_CPU_REQUEST</b>	Puller の実行時にキャッシュされる各イメージのプロセッサ要求	.05 または 50 ミリコア
<b>CACHING_CPU_LIMIT</b>	Puller の実行時にキャッシュされる各イメージのプロセッサ制限	.2 または 200 ミリコア
<b>DAEMONSET_NAME</b>	作成するデーモンセットの名前	kubernetes-image-puller
<b>DEPLOYMENT_NAME</b>	作成するデプロイメントの名前	kubernetes-image-puller
<b>NAMESPACE</b>	作成するデーモンセットが含まれる OpenShift プロジェクト	k8s-image-puller
<b>IMAGES</b>	プルするイメージのセミコロンで区切られた一覧 ( <code>&lt;name1&gt;=&lt;image1&gt;;&lt;name2&gt;=&lt;image2&gt;</code> の形式) <a href="#">「プルするイメージの一覧の定義」</a> を参照してください。	

パラメーター	使用法	デフォルト
<b>NODE_SELECTOR</b>	デーモンセットによって作成される Pod に適用するノードセクター	'{}'
<b>AFFINITY</b>	DaemonSet によって作成される Pod に適用されるアフィニティー	'{}'
<b>IMAGE_PULL_SECRETS</b>	DeamonSet で作成される Pod に追加する <b>pullsecret1;...</b> 形式のイメージプルシークレットの一覧。これらのシークレットはイメージ puller の namespace に配置し、クラスター管理者はそれらを作成する必要があります。	""

## 関連情報

- [「プルするイメージの一覧の定義」](#)
- [「Image Puller のメモリーパラメーターの定義」](#)
- [「CodeReady Workspaces Operator を使用した Image Puller のインストール」](#)
- [「OperatorHub を使用した OpenShift 4 での Image Puller のインストール」](#)
- [「OpenShift テンプレートを使用した OpenShift への Image Puller のインストール」](#)
- [Kubernetes Image Puller ソースコードリポジトリ](#)

## 8.1. プルするイメージの一覧の定義

Image Puller は、**che-machine-exec** などの scratch イメージを含むほとんどのイメージを事前プルできます。ただし、**traefik** などの Dockerfile にボリュームをマウントするイメージは、OpenShift 3.11 における事前プルではサポートされません。

ワークスペースの起動に関連するイメージの事前プルが行われると、ワークスペースの起動時間が短縮されます。例:

- Che-Theia
- ブローカーイメージ
- プラグインサイドカーイメージ

## 前提条件

- **curl** ツールが利用できる。[curl ホームページ](#)を参照してください。
- **jq** ツールが利用できる。[jq ホームページ](#)を参照してください。

- **yq** ツールが利用できる。[yq ホームページ](#)を参照してください。

## 手順

1. OpenShift プラットフォームの関連するコンテナイメージの一覧を収集します。

### 例8.1 CodeReady Workspaces 2.12 のすべてのイメージ一覧の取得

```
$ curl -sSL - https://raw.githubusercontent.com/redhat-developer/codeready-workspaces-
images/crw-2.12-rhel-8/codeready-workspaces-operator-metadata-
generated/manifests/codeready-workspaces.csv.yaml \
| yq -r '.spec.relatedImages[]'
```

2. ワークスペースの起動フェーズに関連するイメージを保持します。

- **eap**
- **machineexec**
- **mongodb**
- **pluginbroker**
- **plugin-**
- **stacks**
- **theia**
- **ubi-minimal**

3. ターゲットプラットフォームで対応していないコンテナイメージの一覧から除外します。

- AMD64 および Intel 64 (x86\_64)の場合は、**openj9** イメージを除外します。

### 例8.2 openj9 イメージを除く AMD64 および Intel 64 (x86\_64)のイメージ一覧

```
che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-
workspaces/pluginbroker-artifacts-
rhel8@sha256:3436bacdc249e8dcc289b67e89340bdc2c5b6694304fe369af751f684a1
40d60;
che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:e709e89bd1ab1612baad039ad6b824b027a070ff0a37be3395202faa55
518e09;
codeready_workspaces_machineexec_plugin_registry_image_gixdcmqk=registry.redhat
.io/codeready-workspaces/machineexec-
rhel8@sha256:93dc77a7f33171b36dbc112e7e732da4044c4c75a48b7a25907521c529
170c0c;
codeready_workspaces_plugin_java11_devfile_registry_image_gixdcmqk=registry.redha
t.io/codeready-workspaces/plugin-java11-
rhel8@sha256:1ca806f7a094fd524b9012d593eafeb5096233c2871e21205d705e46091
e6de7;
codeready_workspaces_plugin_java11_plugin_registry_image_gixdcmqk=registry.redha
t.io/codeready-workspaces/plugin-java11-
rhel8@sha256:1ca806f7a094fd524b9012d593eafeb5096233c2871e21205d705e46091
```

```
e6de7;
codeready_workspaces_plugin_java8_devfile_registry_image_gixdcmqk=registry.redhat
io/codeready-workspaces/plugin-java8-
rhel8@sha256:eff4d7987d332b0b3baf22ba1e056b339063ce92d1e84f8c9027cb3c28e1
aea6;
codeready_workspaces_plugin_java8_plugin_registry_image_gixdcmqk=registry.redhat.
o/codeready-workspaces/plugin-java8-
rhel8@sha256:eff4d7987d332b0b3baf22ba1e056b339063ce92d1e84f8c9027cb3c28e1
aea6;
codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcmqk=registry.r
edhat.io/codeready-workspaces/plugin-kubernetes-
rhel8@sha256:827cd9b9c30e14ec3958445dc3ef1a087794a1f0259601e868619af746d
aee89;
codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcmqk=registry.red
hat.io/codeready-workspaces/plugin-openshift-
rhel8@sha256:d794597b27e15e47560ce63596226932d498243ac4327dfe5514af1645
e3c23a;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcmqk=registry.redhat.i
o/codeready-workspaces/stacks-cpp-
rhel8@sha256:871edac168ea6d670943cb7a7f3e51f2ad2aff5d3ccf1fc22e3b1872db60d
8d7;
codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcmqk=registry.redhat.ic
/codeready-workspaces/stacks-cpp-
rhel8@sha256:871edac168ea6d670943cb7a7f3e51f2ad2aff5d3ccf1fc22e3b1872db60d
8d7;
codeready_workspaces_stacks_dotnet_devfile_registry_image_gixdcmqk=registry.redha
t.io/codeready-workspaces/stacks-dotnet-
rhel8@sha256:60007e969d7e3dbc526f3be6b5ebb6aadac9018448c6f2d967ade8400d
158531;
codeready_workspaces_stacks_dotnet_plugin_registry_image_gixdcmqk=registry.redha
t.io/codeready-workspaces/stacks-dotnet-
rhel8@sha256:60007e969d7e3dbc526f3be6b5ebb6aadac9018448c6f2d967ade8400d
158531;
codeready_workspaces_stacks_golang_devfile_registry_image_gixdcmqk=registry.redh
at.io/codeready-workspaces/stacks-golang-
rhel8@sha256:a2cbdc901b1f52e2f11482cdf5c2ee367858ba5ab3789c7330f1e79c561
5688;
codeready_workspaces_stacks_golang_plugin_registry_image_gixdcmqk=registry.redha
t.io/codeready-workspaces/stacks-golang-
rhel8@sha256:a2cbdc901b1f52e2f11482cdf5c2ee367858ba5ab3789c7330f1e79c561
5688;
codeready_workspaces_stacks_php_devfile_registry_image_gixdcmqk=registry.redhat.i
o/codeready-workspaces/stacks-php-
rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fde74935134cfc807a4c97e
6631;
codeready_workspaces_stacks_php_plugin_registry_image_gixdcmqk=registry.redhat.ic
/codeready-workspaces/stacks-php-
rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fde74935134cfc807a4c97e
6631;
codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcmqk=registry.redh
at.io/codeready-workspaces/theia-endpoint-
rhel8@sha256:2eb5fd69e08e3a1a8f557fe1d41170ee9dac89d15c5947aabc477d4e210
7a9ea;
codeready_workspaces_theia_plugin_registry_image_gixdcmqk=registry.redhat.io/coder
eady-workspaces/theia-
rhel8@sha256:7c1fcf0a972e2905a0758d53e2155534643d0fbcbb688d57b54f05baf32
```

```

3582;
jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqi_____
_=registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-
rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c8923
58632d;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi_____
_=registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-
rhel8@sha256:3875b2ee2826a6d8134aa3b80ac0c8b5ebc4a7f718335d76dfc3461b79f
93d19;
pvc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:54ef2173bba7384dc7609e8affbae1c36f8a3ec137cacc0866116d65dd
4b9afe;
rhsc_l_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhsc_l/mo
ngodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0
dc73;

```

- IBM Z および IBM Power Systems の場合は、**java8** および **java11** の **openj9** バージョンを使用し **dotnet** を除外します。

### 例8.3 IBM Z および IBM Power Systems のイメージ一覧 **java8** および **java11** の **openj9** バージョンを使用し、**dotnet** を除く

```

che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-
workspaces/pluginbroker-artifacts-
rhel8@sha256:3436bacdc249e8dcc289b67e89340bdc2c5b6694304fe369af751f684a1
40d60;
che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:e709e89bd1ab1612baad039ad6b824b027a070ff0a37be3395202faa55
518e09;
codeready_workspaces_machineexec_plugin_registry_image_gixdcmqk=registry.redhat
.io/codeready-workspaces/machineexec-
rhel8@sha256:93dc77a7f33171b36dbc112e7e732da4044c4c75a48b7a25907521c529
170c0c;
codeready_workspaces_plugin_java11_openj9_devfile_registry_image_gixdcmqk=regist
ry.redhat.io/codeready-workspaces/plugin-java11-openj9-
rhel8@sha256:b91184021db6e0d86741017a7d723c3204fda0dac532a8a2baa4b92462
85116f;
codeready_workspaces_plugin_java11_openj9_plugin_registry_image_gixdcmqk=regist
y.redhat.io/codeready-workspaces/plugin-java11-openj9-
rhel8@sha256:b91184021db6e0d86741017a7d723c3204fda0dac532a8a2baa4b92462
85116f;
codeready_workspaces_plugin_java8_openj9_devfile_registry_image_gixdcmqk=registr
y.redhat.io/codeready-workspaces/plugin-java8-openj9-
rhel8@sha256:7f5fc6ee8b61bff9b1643b3ce94cb276d75d0d63ba2cf6aa7c9ab9e87384
23bc;
codeready_workspaces_plugin_java8_openj9_plugin_registry_image_gixdcmqk=registry
.redhat.io/codeready-workspaces/plugin-java8-openj9-
rhel8@sha256:7f5fc6ee8b61bff9b1643b3ce94cb276d75d0d63ba2cf6aa7c9ab9e87384
23bc;
codeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcmqk=registry.r
edhat.io/codeready-workspaces/plugin-kubernetes-
rhel8@sha256:827cd9b9c30e14ec3958445dc3ef1a087794a1f0259601e868619af746d
aee89;

```



```
codeready_workspaces_plugin_openshift_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/plugin-openshift-rhel8@sha256:d794597b27e15e47560ce63596226932d498243ac4327dfe5514af1645e3c23a;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:871edac168ea6d670943cb7a7f3e51f2ad2aff5d3ccf1fc22e3b1872db60d8d7;
codeready_workspaces_stacks_cpp_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:871edac168ea6d670943cb7a7f3e51f2ad2aff5d3ccf1fc22e3b1872db60d8d7;
codeready_workspaces_stacks_golang_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:a2cbdc901b1f52e2f11482cdf5c2ee367858ba5ab3789c7330f1e79c5615688;
codeready_workspaces_stacks_golang_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:a2cbdc901b1f52e2f11482cdf5c2ee367858ba5ab3789c7330f1e79c5615688;
codeready_workspaces_stacks_php_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fde74935134cfc807a4c97e6631;
codeready_workspaces_stacks_php_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fde74935134cfc807a4c97e6631;
codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/theia-endpoint-rhel8@sha256:2eb5fd69e08e3a1a8f557fe1d41170ee9dac89d15c5947aabc477d4e2107a9ea;
codeready_workspaces_theia_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:7c1fcf0a972e2905a0758d53e2155534643d0fbcbb688d57b54f05baf323582;
jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqi_____registry.redhat.io/jboss-eap-7/eap74-openjdk8_openshift-rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d;
jboss_eap_7_eap_xp3_openj9_11_openshift_devfile_registry_image_gmxdacq_____registry.redhat.io/jboss-eap-7/eap-xp3-openj9-11_openshift-rhel8@sha256:896c1a9baf21400e8bc75e8b7fb22fc3a829aa3fee68ca9f8373111b7c21e27d;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi_____registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11_openshift-rhel8@sha256:3875b2ee2826a6d8134aa3b80ac0c8b5ebc4a7f718335d76dfc3461b79f93d19;
pvc_jobs=registry.redhat.io/ubi8/ubi-minimal@sha256:54ef2173bba7384dc7609e8affbae1c36f8a3ec137cacc0866116d65dd4b9afe;
rhsc MongoDB_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhsc/mongodb-36-rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73;
```

4. プル前の一覧からイメージを判別します。  
ワークスペースの起動時間を短縮するには、ワークスペース関連のイメージを事前にプルすることを検討してください。
- **theia-rhel8**
  - **theia-endpoint-rhel8**
  - **pluginbroker-artifacts-rhel8**
  - **pluginbroker-metadata-rhel8**
  - **stacks-\*-rhel8**
  - **plugin-\*-rhel8**
    - スタックイメージの一覧: [コンテナイメージ - スタック](#)
    - プラグインイメージの一覧: [コンテナイメージ - プラグイン](#)

## 関連情報

- [「Image Puller のメモリーパラメーターの定義」](#)
- [「OperatorHub を使用した OpenShift 4 での Image Puller のインストール」](#)
- [「OpenShift テンプレートを使用した OpenShift への Image Puller のインストール」](#)

## 8.2. IMAGE PULLER のメモリーパラメーターの定義

メモリー要求および制限パラメーターを定義して、コンテナをプルし、プラットフォームに実行するのに十分なメモリーがあることを確認します。

### 前提条件

- [「プルするイメージの一覧の定義」](#)

### 手順

1. **CACHING\_MEMORY\_REQUEST** または **CACHING\_MEMORY\_LIMIT** の最小値を定義するには、プルする各コンテナイメージの実行に必要なメモリー容量を考慮してください。
2. **CACHING\_MEMORY\_REQUEST** または **CACHING\_MEMORY\_LIMIT** の最大値を定義するには、クラスターのデーモンセット Pod に割り当てられるメモリーの合計を考慮します。

$$\text{(memory limit)} * \text{(number of images)} * \text{(number of nodes in the cluster)}$$

コンテナのメモリー制限が **20Mi** の 20 ノードで 5 つのイメージをプルする場合、**2000Mi** のメモリーが必要です。

## 関連情報

- [「OperatorHub を使用した OpenShift 4 での Image Puller のインストール」](#)
- [「OpenShift テンプレートを使用した OpenShift への Image Puller のインストール」](#)

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

本セクションでは、CodeReady Workspaces Operatorを使用して、テクノロジープレビュー状態でコミュニティがサポートする機能であるImage Pullerをインストールする方法を説明します。

### 前提条件

- [「プルするイメージの一覧の定義」](#)
- [「Image Puller のメモリーパラメーターの定義」](#)
- Operator Lifecycle Manager および OperatorHub が OpenShift インスタンスで利用できる。OpenShift は、バージョン 4.2 以降のバージョンを提供します。
- CodeReady Workspaces Operator が利用できる。[OperatorHub を使用した OpenShift 4 への CodeReady Workspaces のインストール](#)を参照してください。

### 手順

1. `.spec.imagePuller.enable` を `true` に設定して、**CheCluster** カスタムリソースの Image Puller を有効にします。

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

2. **CheCluster** カスタムリソースに Image Puller を設定します。

```
apiVersion: org.eclipse.che/v1
kind: CheCluster
metadata:
  name: codeready-workspaces
spec:
  ...
  imagePuller:
    enable: true
  spec:
    configMapName: <kubernetes-image-puller>
    daemonsetName: <kubernetes-image-puller>
    deploymentName: <kubernetes-image-puller>
    images: 'che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-
workspaces/pluginbroker-artifacts-
rhel8@sha256:3436bacdc249e8dcc289b67e89340bdc2c5b6694304fe369af751f684a140d60;c
he_workspace_plugin_broker_metadata=registry.redhat.io/codeready-
workspaces/pluginbroker-metadata-
rhel8@sha256:e709e89bd1ab1612baad039ad6b824b027a070ff0a37be339520faa55518e09;
codeready_workspaces_machineexec_plugin_registry_image_gixdcmqk=registry.redhat.io/code
ready-workspaces/machineexec-
rhel8@sha256:93dc77a7f33171b36dbc112e7e732da4044c4c75a48b7a25907521c529170c0c;
```

```
codeready_workspaces_plugin_java11_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/plugin-java11-rhel8@sha256:1ca806f7a094fd524b9012d593eafeb5096233c2871e21205d705e46091e6de7;
codeready_workspaces_plugin_java11_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/plugin-java11-rhel8@sha256:1ca806f7a094fd524b9012d593eafeb5096233c2871e21205d705e46091e6de7;
codeready_workspaces_plugin_java8_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/plugin-java8-rhel8@sha256:eff4d7987d332b0b3baf22ba1e056b339063ce92d1e84f8c9027cb3c28e1aea6;c
odeready_workspaces_plugin_java8_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/plugin-java8-rhel8@sha256:eff4d7987d332b0b3baf22ba1e056b339063ce92d1e84f8c9027cb3c28e1aea6;c
odeready_workspaces_plugin_kubernetes_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/plugin-kubernetes-rhel8@sha256:827cd9b9c30e14ec3958445dc3ef1a087794a1f0259601e868619af746dae89;c
odeready_workspaces_plugin_openshift_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/plugin-openshift-rhel8@sha256:d794597b27e15e47560ce63596226932d498243ac4327dfe5514af1645e3c23a;
codeready_workspaces_stacks_cpp_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:871edac168ea6d670943cb7a7f3e51f2ad2aff5d3ccf1fc22e3b1872db60d8d7;co
deready_workspaces_stacks_cpp_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-cpp-rhel8@sha256:871edac168ea6d670943cb7a7f3e51f2ad2aff5d3ccf1fc22e3b1872db60d8d7;co
deready_workspaces_stacks_dotnet_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-dotnet-rhel8@sha256:60007e969d7e3dbc526f3be6b5ebb6aadac9018448c6f2d967ade8400d158531;
codeready_workspaces_stacks_dotnet_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-dotnet-rhel8@sha256:60007e969d7e3dbc526f3be6b5ebb6aadac9018448c6f2d967ade8400d158531;
codeready_workspaces_stacks_golang_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:a2cbdc901b1f52e2f11482cdf5c2ee367858ba5ab3789c7330f1e79c5615688;co
deready_workspaces_stacks_golang_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-golang-rhel8@sha256:a2cbdc901b1f52e2f11482cdf5c2ee367858ba5ab3789c7330f1e79c5615688;co
deready_workspaces_stacks_php_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fe74935134cfc807a4c97e6631;co
deready_workspaces_stacks_php_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fe74935134cfc807a4c97e6631;co
deready_workspaces_theia_endpoint_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/theia-endpoint-rhel8@sha256:2eb5fd69e08e3a1a8f557fe1d41170ee9dac89d15c5947aabc477d4e2107a9ea;c
odeready_workspaces_theia_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:7c1fcf0a972e2905a0758d53e2155534643d0fbcbb688d57b54f05baf323582;jb
oss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqi_____ =regist
ry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-
rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d;
jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi_____ =regist
ry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-
rhel8@sha256:3875b2ee2826a6d8134aa3b80ac0c8b5ebc4a7f718335d76dfc3461b79f93d19;p
vc_jobs=registry.redhat.io/ubi8/ubi-
minimal@sha256:54ef2173bba7384dc7609e8affbae1c36f8a3ec137cacc0866116d65dd4b9afe;
```

```
rhsc1_mongodb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhsc1/mongodb-36-
rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73;
```

## 注記

サポートされている Image Puller を使用するには、これを KubernetesImagePuller Operator とは別にインストールします。Red Hat が提供する追加のテストおよび検証により、Red Hat の公式ビルドの利点。

CodeReady Workspaces のインストール時に Operator Hub での KubernetesImagePuller の使用を有効にし、使用するコミュニティでサポートされるバージョンを設定します。

- [コミュニティビルド](#)
- [Red Hat の公式ビルド](#)

## デフォルトイメージ

- CodeReady Workspaces Operator は、**CheCluster** カスタムリソースを作成する前にイメージがこのフィールドに追加されていない場合に、ワークスペースの起動（Theia イメージ、プラグインブローカーイメージ、サイドカープラグインイメージ）に使用されるデフォルトイメージを **.spec.imagePuller.spec.images** フィールドに入力します。CodeReady Workspaces Operator は、CodeReady Workspaces のロールアウト更新後に **.spec.imagePuller.spec.images** フィールドのデフォルトイメージを更新します。ただし、**CheCluster** カスタムリソースを作成する前に、イメージが **.spec.imagePuller.spec.images** フィールドに追加された場合、CodeReady Workspaces Operator はデフォルトイメージを追加しません。
- **CheCluster** カスタムリソースの作成後にユーザーによって提供されるイメージが **.spec.imagePuller.spec.images** フィールドに追加されると、CodeReady Workspaces Operator は後続の CodeReady Workspaces ロールアウトの更新時にデフォルトイメージを更新します。ロールアウトの更新後も、デフォルト以外のイメージは **.spec.imagePuller.spec.images** フィールドで変更されないままになります。

## 検証

- OpenShift は **kubernetes-image-puller-operator** サブスクリプションを作成します。
- **eclipse-che namespace** には コミュニティでサポートされる **community supported Kubernetes Image Puller Operator ClusterServiceVersion** が含まれます。

```
$ oc get clusterserviceversions
```

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

```
$ oc get deployments
```

- コミュニティがサポートする Kubernetes Image Puller Operator は **KubernetesImagePuller** カスタムリソースを作成します。

```
$ oc get kubernetesimagepullers
```

## CodeReady Workspaces Operator を使用した Image Puller のアンインストール

1. **CheCluster** カスタムリソースを編集し、`.spec.imagePuller.enable` を **false** に編集します。
2. **CheCluster** カスタムリソースを編集し、`.spec.imagePuller.spec` を、CodeReady Workspaces Operator のオプションの Image Puller パラメーターを設定できるように設定します。

## 8.4. OPERATORHUB を使用した OPENSIFT 4 での IMAGE PULLER のインストール

この手順では、Operator を使用してコミュニティでサポートされる Kubernetes Image Puller Operator を OpenShift 4 にインストールする方法について説明します。

### 前提条件

- OpenShift 4 の実行中のインスタンスの管理者アカウント
- [「プルするイメージの一覧の定義」](#)
- [「Image Puller のメモリーパラメーターの定義」](#)

### 手順

1. Image Puller をホストする OpenShift プロジェクト <kubernetes-image-puller> を作成するには、OpenShift Web コンソールを開き、**Home** → **Projects** セクションに移動し、**Create Project** をクリックします。
2. プロジェクトの詳細を指定します。
  - **Name:** <kubernetes-image-puller>
  - **Display Name:** <Image Puller>
  - **Description:** <Kubernetes Image Puller>
3. **Operators** → **OperatorHub** に移動します。
4. **Filter by keyword** ボックスを使用して、**community supported Kubernetes Image Puller Operator** を検索します。**community supported Kubernetes Image Puller Operator** をクリックします。
5. Operator の説明を確認します。**Continue** → **Install** をクリックします。
6. **Installation Mode** について **A specific project on the cluster** を選択します。ドロップダウンで、OpenShift プロジェクト <kubernetes-image-puller> を見つけます。**Subscribe** をクリックします。
7. コミュニティがサポートする Kubernetes Image Puller Operator のインストールを待機します。**KubernetesImagePuller** → **Create instance** をクリックします。
8. YAML エディターのあるリダイレクトされるウィンドウで、**KubernetesImagePuller** カスタムリソースに変更を加え、**Create** をクリックします。
9. <kubernetes-image-puller> OpenShift プロジェクトの **Workloads** および **Pods** メニューに移動します。Image Puller が利用可能であることを確認します。

## 8.5. OPENSIFT テンプレートを使用した OPENSIFT への IMAGE PULLER のインストール

この手順では、OpenShift テンプレートを使用して Kubernetes Image Puller を OpenShift にインストールする方法を説明します。

### 前提条件

- 実行中の OpenShift クラスター。
- **oc** ツールが利用できる。
- [「プルするイメージの一覧の定義」](#)
- [「Image Puller のメモリーパラメーターの定義」](#)

### 手順

1. Image Puller リポジトリのクローンを作成し、OpenShift テンプレートが含まれるディレクトリを取得します。

```
$ git clone https://github.com/che-incubator/kubernetes-image-puller
$ cd kubernetes-image-puller/deploy/openshift
```

2. 以下のパラメーターを使用して、**app.yaml**、**configmap.yaml** および **serviceaccount.yaml** OpenShift テンプレートを設定します。

表8.2 app.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用法	デフォルト
<b>DEPLOYMENT_NAME</b>	ConfigMap の <b>DEPLOYMENT_NAME</b> の値	<b>kubernetes-image-puller</b>
<b>IMAGE</b>	<b>kubernetes-image-puller</b> デプロイメントに使用されるイメージ	<b>registry.redhat.io/codeready-workspaces/imagepuller-rhel8:2.12</b>
<b>IMAGE_TAG</b>	プルするイメージタグ	<b>latest</b>
<b>SERVICEACCOUNT_NAME</b>	デプロイメントで作成され、使用される ServiceAccount の名前	<b>kubernetes-image-puller</b>

表8.3 configmap.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用法	デフォルト
<b>CACHING_CPU_LIMIT</b>	ConfigMap の <b>CACHING_CPU_LIMIT</b> の値	<b>.2</b>

値	使用法	デフォルト
<b>CACHING_CPU_REQUEST</b>	ConfigMap の <b>CACHING_CPU_REQUEST</b> の値	.05
<b>CACHING_INTERVAL_HOURS</b>	ConfigMap の <b>CACHING_INTERVAL_HOURS</b> の値	"1"
<b>CACHING_MEMORY_LIMIT</b>	ConfigMap の <b>CACHING_MEMORY_LIMIT</b> の値	"20Mi"
<b>CACHING_MEMORY_REQUEST</b>	ConfigMap の <b>CACHING_MEMORY_REQUEST</b> の値	"10Mi"
<b>DAEMONSET_NAME</b>	ConfigMap の <b>DAEMONSET_NAME</b> の値	kubernetes-image-puller
<b>DEPLOYMENT_NAME</b>	ConfigMap の <b>DEPLOYMENT_NAME</b> の値	kubernetes-image-puller
<b>IMAGES</b>	ConfigMap の <b>IMAGES</b> の値	'che_workspace_plugin_broker_artifacts=registry.redhat.io/codeready-workspaces/pluginbroker-artifacts-rhel8@sha256:3436bacdc249e8dcc289b67e89340bd c2c5b6694304fe369af751f684a140d60;che_workspace_plugin_broker_metadata=registry.redhat.io/codeready-workspaces/pluginbroker-metadata-rhel8@sha256:e709e89bd1ab1612baad039ad6b824b027a070ff0a37be339520faa55518e09;codeready_workspaces_machineexec_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/machineexec-rhel8@sha256:93dc77a7f33171b36dbc112e7e732da4044c4c75a48b7a25907521c529170c0c;codeready_workspaces_plugin_java11_d



値	使用法	デフォルト
		<pre> evfile_registry_image_gix cmqk=registry.redhat.io/ codeready- workspaces/plugin- java11- rhel8@sha256:1ca806f7a0 94fd524b9012d593eafeb50 96233c2871e21205d705e46 091e6de7;codeready_wor kspaces_plugin_java11_pl ugin_registry_image_gixd cmqk=registry.redhat.io/c odeready- workspaces/plugin- java11- rhel8@sha256:1ca806f7a0 94fd524b9012d593eafeb50 96233c2871e21205d705e46 091e6de7;codeready_wor kspaces_plugin_java8_de vfile_registry_image_gixd cmqk=registry.redhat.io/c odeready- workspaces/plugin-java8- rhel8@sha256:eff4d7987d 332b0b3baf22ba1e056b33 9063ce92d1e84f8c9027cb3 c28e1aea6;codeready_wor kspaces_plugin_java8_plu gin_registry_image_gixdc mqk=registry.redhat.io/co deready- workspaces/plugin-java8- rhel8@sha256:eff4d7987d 332b0b3baf22ba1e056b33 9063ce92d1e84f8c9027cb3 c28e1aea6;codeready_wor kspaces_plugin_kubern es_plugin_registry_image _gixdcmqk=registry.redha t.io/codeready- workspaces/plugin- kubernetes- rhel8@sha256:827cd9b9c 30e14ec3958445dc3ef1a08 7794a1f0259601e868619af 746daee89;codeready_wo rkspaces_plugin_openshif t_plugin_registry_image_ gixdcmqk=registry.redhat. io/codeready- workspaces/plugin- openshift- rhel8@sha256:d794597b2 7e15e47560ce63596226932 d498243ac4327dfe5514af1 </pre>

値	使用法	645e3c23a;codeready_wor kspaces_stacks_cpp_devf ile_registry_image_gixdc mqq=registry.redhat.io/co deready-
		workspaces/stacks-cpp- rhel8@sha256:871edac168 ea6d670943cb7a7f3e51f2a d2aff5d3ccf1fc22e3b1872d b60d8d7;codeready_work spaces_stacks_cpp_plugi n_registry_image_gixdcm qk=registry.redhat.io/code ready-workspaces/stacks- cpp- rhel8@sha256:871edac168 ea6d670943cb7a7f3e51f2a d2aff5d3ccf1fc22e3b1872d b60d8d7;codeready_work spaces_stacks_dotnet_de vfile_registry_image_gixd cmqk=registry.redhat.io/c odeready- workspaces/stacks- dotnet- rhel8@sha256:60007e969d 7e3dbc526f3be6b5ebb6aa dac9018448c6f2d967ade84 00d158531;codeready_wo rkspaces_stacks_dotnet_ plugin_registry_image_gix dcmqk=registry.redhat.io/ codeready- workspaces/stacks- dotnet- rhel8@sha256:60007e969d 7e3dbc526f3be6b5ebb6aa dac9018448c6f2d967ade84 00d158531;codeready_wo rkspaces_stacks_golang_ devfile_registry_image_gi xdcmqk=registry.redhat.io /codeready- workspaces/stacks- golang- rhel8@sha256:a2cbdc901 b1f52e2f11482cdf5c2ee36 7858ba5ab3789c7330f1e79 c5615688;codeready_work spaces_stacks_golang_pl ugin_registry_image_gixd cmqk=registry.redhat.io/c odeready- workspaces/stacks- golang- rhel8@sha256:a2cbdc901

値	使用法	b1f52e2f11482cdf5c2ee36 デフォルト 78366a3ab3789c7330f1e79
		c5615688;codeready_workspaces_stacks_php_devfile_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fde74935134cfc807a4c97e6631;codeready_workspaces_stacks_php_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/stacks-php-rhel8@sha256:32d28aef989cadf8dda497dd3375de2c65fd0fde74935134cfc807a4c97e6631;codeready_workspaces_theia_endpoint_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/theia-endpoint-rhel8@sha256:2eb5fd69e08e3a1a8f557fe1d41170ee9dac89d15c5947aabc477d4e2107a9ea;codeready_workspaces_theia_plugin_registry_image_gixdcmqk=registry.redhat.io/codeready-workspaces/theia-rhel8@sha256:7c1fcf0a972e2905a0758d53e2155534643d0fbcbb688d57b54f05baf323582;jboss_eap_7_eap74_openjdk8_openshift_rhel7_devfile_registry_image_g4xdilrqi_____registry.redhat.io/jboss-eap-7/eap74-openjdk8-openshift-rhel7@sha256:b4a113c4d4972d142a3c350e2006a2b297dc883f8ddb29a88db19c892358632d;jboss_eap_7_eap_xp3_openjdk11_openshift_devfile_registry_image_gmxdaljzbi_____registry.redhat.io/jboss-eap-7/eap-xp3-openjdk11-openshift-rhel8@sha256:3875b2ee28

値	使用法	デフォルト
		26a6d8134aa3b80ac0c8b5 b79f93d19;pvc_jobs=registry.redhat.io/ubi8/ubi-minimal@sha256:54ef2173bba7384dc7609e8affbae1c36f8a3ec137cacc0866116d65dd4b9afe;rhscldb_36_rhel7_devfile_registry_image_gewtkmak=registry.redhat.io/rhscldb_36_rhel7@sha256:9f799d356d7d2e442bde9d401b720600fd9059a3d8eefea6f3b2ffa721c0dc73;'
<b>NAMESPACE</b>	ConfigMap の <b>NAMESPACE</b> の値	k8s-image-puller
<b>NODE_SELECTOR</b>	ConfigMap の <b>NODE_SELECTOR</b> の値	"{}"

表8.4 serviceaccount.yaml の Image Puller OpenShift テンプレートパラメーター

値	使用法	デフォルト
<b>SERVICEACCOUNT_NAME</b>	デプロイメントで作成され、使用される ServiceAccount の名前	kubernetes-image-puller

- Image Puller をホストする OpenShift プロジェクトを作成します。

```
$ oc new-project <k8s-image-puller>
```

- テンプレートを処理してから適用し、Puller をインストールします。

```
$ oc process -f serviceaccount.yaml | oc apply -f -
$ oc process -f configmap.yaml | oc apply -f -
$ oc process -f app.yaml | oc apply -f -
```

## 検証手順

- <kubernetes-image-puller> デプロイメントおよび <kubernetes-image-puller> デモンセットがあることを確認します。デモンセットでは、クラスター内の各ノードに Pod が必要です。

```
$ oc get deployment,daemonset,pod --namespace <k8s-image-puller>
```

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

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

## 第9章 ID および承認の管理

このセクションでは、Red Hat CodeReady Workspaces の ID および承認の管理についての各種の側面について説明します。

- [「ユーザーの認証」](#)
- [「ユーザーの認証」](#)
- [「認証の設定」](#)
- [「ユーザーデータの削除」](#)
- [「OpenShift OAuth の設定」](#)

### 9.1. ユーザーの認証

以下では、CodeReady Workspaces サーバー上とワークスペース内の両方で、Red Hat CodeReady Workspaces のユーザー認証のすべての側面について説明します。これには、すべての REST API エンドポイント、WebSocket または JSON RPC 接続、一部の Web リソースのセキュリティーを保護することが含まれます。

すべての認証タイプは、[JWT オープン標準](#)を、ユーザー ID 情報を転送するコンテナとして使用します。さらに、CodeReady Workspaces サーバー認証は、[RH-SSO](#) によってデフォルトで提供される [OpenID Connect](#) プロトコル実装に基づいて行われます。

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

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

##### 9.1.1.1. 他の認証の実装を使用した CodeReady Workspaces サーバーに対する認証

この手順では、RH-SSO 以外の OpenID Connect (OIDC) 認証の実装を使用する方法について説明します。

#### 手順

1. **multiuser.properties** ファイルに保存されている認証設定パラメーターを更新します (例: クライアント ID、認証 URL、レルム名)。
2. 単一のフィルターまたはフィルターチェーンを作成してトークンを検証し、CodeReady Workspaces ダッシュボードでユーザーを作成し、**subject** オブジェクトを作成します。
3. 新規の認証プロバイダーが OpenID プロトコルをサポートする場合、設定エンドポイントで利用可能な OIDC JS クライアントライブラリーを使用してください。これは特定の实装から分離されるためです。
4. 選択されたプロバイダーがユーザー (名前および姓、肩書き) についての追加データを保存する場合、この情報を提供するプロバイダーに固有の **ProfileDao** 実装を作成することが推奨されます。

##### 9.1.1.2. OAuth を使用した CodeReady Workspaces サーバーに対する認証

サードパーティーサービスとのユーザーの対話を容易にするために、CodeReady Workspaces サーバーは OAuth 認証をサポートします。OAuth トークンは、GitHub 関連のプラグインにも使用されます。

OAuth 認証には、2 つの主要なフローがあります。

#### delegated

デフォルトです。OAuth 認証を RH-SSO サーバーに委譲します。

#### embedded

ビルトイン CodeReady Workspaces サーバーメカニズムを使用して OAuth プロバイダーと通信します。

2 つの実装間で切り替えるには、**che.oauth.service\_mode=<embedded|delegated>** 設定プロパティを使用します。

OAuth API の主な REST エンドポイントは **/api/oauth** であり、以下が含まれます。

- OAuth 認証フローを開始できる認証メソッドの **/authenticate**。
- プロバイダーからのコールバックを処理するコールバックメソッドの **/callback**。
- 現行ユーザーの OAuth トークンを取得するためのトークン GET メソッドの **/token**。
- 現行ユーザーの OAuth トークンを無効にするためのトークン DELETE メソッドの **/token**。
- 設定済みのアイデンティティプロバイダーの一覧を取得する GET メソッドの **/**。

#### 9.1.1.3. Swagger または REST クライアントを使用したクエリーの実行

ユーザーの RH-SSO トークンを使用して、REST クライアントでユーザーの代わりにセキュアな API に対してクエリーを実行します。有効なトークンは、**Request** ヘッダーまたは **?token=\$token** クエリーパラメーターとして割り当てする必要があります。

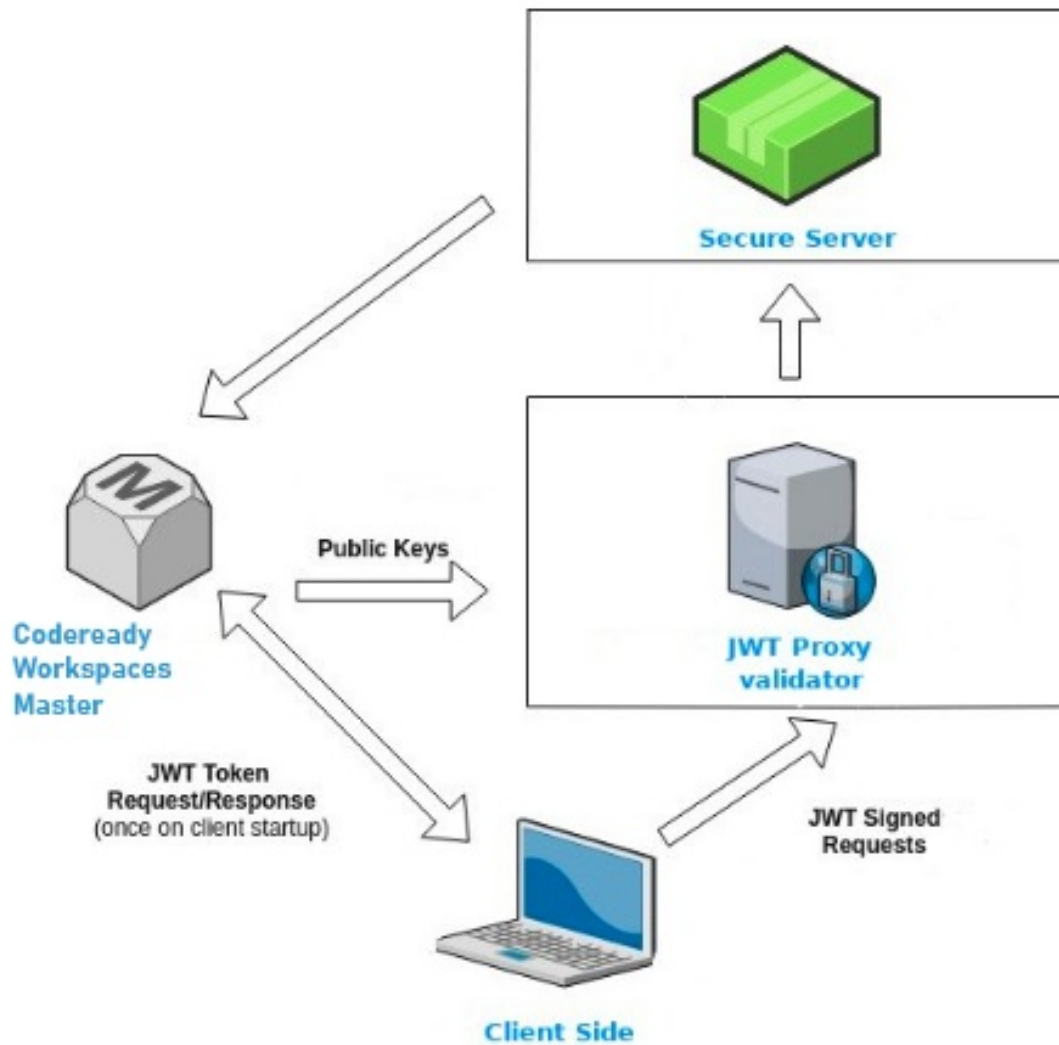
CodeReady Workspaces Swagger インターフェース **\https://codeready-<openshift\_deployment\_name>.<domain\_name>/swagger** にアクセスします。アクセストークンが **Request** ヘッダーに含まれるように、ユーザーは RH-SSO で署名する必要があります。

#### 9.1.2. CodeReady Workspaces ワークスペースでの認証

ワークスペースコンテナには、認証で保護される必要のあるサービスが含まれる場合があります。このように保護されるサービスは、**セキュア** なサービスと呼ばれます。これらのサービスのセキュリティを保護するには、マシンの認証メカニズムを使用します。

JWT トークンを使用すると、RH-SSO トークンをワークスペースコンテナに渡す必要がなくなります（セキュアではなくなる可能性があります）。また、RH-SSO トークンの有効期間は比較的短く、定期的な更新またはリフレッシュが必要になる場合があります。このため、クライアントの同じユーザーセッショントークンを管理し、これらとの同期を維持するのは容易ではありません。

図9.1 ワークスペース内の認証



### 9.1.2.1. セキュアなサーバーの作成

CodeReady Workspaces ワークスペースでセキュアなサーバーを作成するには、`devfile` の `dockerimage` タイプコンポーネントで、エンドポイントの `secure` 属性を `true` に設定します。

#### セキュアなサーバーの devfile スニペット

```
components:
- type: dockerimage
  endpoints:
  - attributes:
    secure: 'true'
```

### 9.1.2.2. ワークスペース JWT トークン

ワークスペーストークンは、要求に以下の情報が含まれる JSON Web トークン (JWT) です。

- **uid:** このトークンを所有するユーザーの ID
- **uname:** このトークンを所有するユーザーの名前
- **wsid:** このトークンでクエリーできるワークスペースの ID

すべてのユーザーには、各ワークスペースに固有の個人用トークンが提供されます。トークンと署名の構造は、RH-SSO の場合とは異なります。以下は、トークンビューの例です。

```
# Header
{
  "alg": "RS512",
  "kind": "machine_token"
}
# Payload
{
  "wsid": "workspacekrh99xjenek3h571",
  "uid": "b07e3a58-ed50-4a6e-be17-fcf49ff8b242",
  "uname": "john",
  "jti": "06c73349-2242-45f8-a94c-722e081bb6fd"
}
# Signature
{
  "value": "RSASHA256(base64UrlEncode(header) + . + base64UrlEncode(payload))"
}
```

RSA アルゴリズムを使用した SHA-256 暗号は、JWT トークンの署名に使用されます。これは設定不可です。また、トークンの署名に使用されるキーペアの公開部分を配布するパブリックサービスはありません。

### 9.1.2.3. マシントークンの検証

マシントークン (JWT トークン) の検証は、別の Pod で実行される **JWTProxy** と共に専用の per-workspace サービスを使用して実行されます。ワークスペースが起動すると、このサービスは CodeReady Workspaces サーバーから SHA キーの公開部分を受信します。セキュアなサーバーごとに個別の検証エンドポイントが作成されます。トラフィックがそのエンドポイントに到達すると、**JWTProxy** は cookie またはヘッダーからトークンを抽出し、公開鍵の部分を使用して検証します。

CodeReady Workspaces サーバーをクエリーするには、ワークスペースサーバーは **CHE\_MACHINE\_TOKEN** 環境変数で提供されるマシントークンを使用できます。このトークンは、ワークスペースを起動するユーザーのトークンです。このような要求の範囲は、現在のワークスペースにのみ制限されます。許可される操作の一覧も厳密に制限されます。

## 9.2. ユーザーの認証

CodeReady Workspaces でのユーザー認証は、パーミッションモデルに基づいて行われます。パーミッションは、ユーザーの許可されるアクションを制御し、セキュリティモデルを確立するために使用されます。すべての要求は、認証にパスした後に、現行ユーザーのサブジェクトに必要なパーミッションがあるかどうかについて検証されます。CodeReady Workspaces が管理するリソースを制御し、ユーザーにパーミッションを割り当てることで特定のアクションを許可できます。

パーミッションは以下のエンティティに適用できます。

- ワークスペース
- システム

すべてのパーミッションは、提供される REST API を使用して管理できます。API は、`\https://codeready-<openshift_deployment_name>.<domain_name>/swagger/#!/permissions` で Swagger を使用して文書化されます。



### 9.2.1. CodeReady Workspaces ワークスペースパーミッション

ワークスペースを作成するユーザーはワークスペースの所有者です。デフォルトで、ワークスペースの所有者には、パーミッション **read**、**use**、**run**、**configure**、**setPermissions**、および **delete** があります。ワークスペースの所有者は、ユーザーをワークスペースに招待し、他のユーザーのワークスペースのパーミッションを制御できます。

ワークスペースには以下のパーミッションが関連付けられています。

表9.1 CodeReady Workspaces ワークスペースパーミッション

パーミッション	説明
read	ワークスペース設定の読み取りを許可します。
use	ワークスペースの使用や、これとの対話を許可します。
run	ワークスペースの開始および停止を許可します。
configure	ワークスペース設定の定義および変更を許可します。
setPermissions	その他のユーザーのワークスペースパーミッションの更新を許可します。
削除	ワークスペースの削除を許可します。

### 9.2.2. CodeReady Workspaces システムパーミッション

CodeReady Workspaces のシステムパーミッションは、CodeReady Workspaces インストール全体のさまざまな側面を制御します。以下のパーミッションがシステムに適用されます。

表9.2 CodeReady Workspaces システムパーミッション

パーミッション	説明
manageSystem	システムおよびワークスペースの制御を許可します。
setPermissions	システムでのユーザーのパーミッションの更新を許可します。
manageUsers	ユーザーの作成および管理を許可します。
monitorSystem	サーバーの状態の監視に使用するエンドポイントへのアクセスを許可します。

すべてのシステムパーミッションは管理ユーザーに付与されます。管理ユーザーを設定するには、**CHE\_SYSTEM\_ADMIN\_NAME** プロパティを使用します。デフォルト値は **admin** です。シス

テムのパーミッションは CodeReady Workspaces サーバーの起動時に付与されます。ユーザーのレコードが CodeReady Workspaces ユーザーデータベースにない場合、ユーザーには最初のログイン後にパーミッションが付与されます。

### 9.2.3. manageSystem パーミッション

manageSystem パーミッションを持つユーザーは、以下のサービスにアクセスできます。

パス	HTTP メソッド	説明
/resource/free/	GET	空きリソース制限を取得します。
/resource/free/{accountId}	GET	指定のアカウントの空きリソース制限を取得します。
/resource/free/{accountId}	POST	指定のアカウントの空きリソース制限を編集します。
/resource/free/{accountId}	DELETE	指定のアカウントの空きリソース制限を削除します。
/installer/	POST	インストーラーをレジストリーに追加します。
/installer/{key}	PUT	レジストリーでインストーラーを更新します。
/installer/{key}	DELETE	レジストリーからインストーラーを削除します。
/logger/	GET	CodeReady Workspaces サーバーのロギング設定を取得します。
/logger/{name}	GET	CodeReady Workspaces サーバーで、ロガーの名前でロガーの設定を取得します。
/logger/{name}	PUT	CodeReady Workspaces サーバーでロガーを作成します。
/logger/{name}	POST	CodeReady Workspaces サーバーでロガーを編集します。
/resource/{accountId}/details	GET	指定のアカウントのリソースに関する詳細情報を取得します。
/system/stop	POST	すべてのシステムサービスをシャットダウンし、CodeReady Workspaces の停止に向けて準備します。

## 9.2.4. monitorSystem パーミッション

monitorSystem パーミッションを持つユーザーは、以下のサービスにアクセスできます。

パス	HTTP メソッド	説明
/activity	GET	一定期間に特定の状態に置かれているワークスペースを取得します。

## 9.2.5. CodeReady Workspaces パーミッションの一覧表示

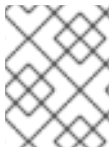
特定の **リソース** に適用される CodeReady Workspaces パーミッションを一覧表示するには、**GET /permissions** 要求を実行します。

**user** に適用されるパーミッションを一覧表示するには、**GET /permissions/{domain}** 要求を実行します。

**すべてのユーザー** に適用されるパーミッションを一覧表示するには、**GET /permissions/{domain}/all** 要求を実行します。この情報を表示するには、ユーザーに **manageSystem** パーミッションが必要です。

適切なドメイン値は次のとおりです。

- system
- organization
- workspace



### 注記

ドメインは任意です。ドメインを指定しないと、API はすべてのドメインについて想定されるすべてのパーミッションを返します。

## 9.2.6. CodeReady Workspaces パーMISSIONの割り当て

リソースにパーMISSIONを割り当てるには、**POST /permissions** 要求を実行します。適切なドメイン値は次のとおりです。

- system
- organization
- workspace

以下は、**userId** を持つユーザーの **workspaceID** を持つワークスペースに対するパーMISSIONを要求するメッセージの本体です。

### CodeReady Workspaces ユーザーパーMISSIONの要求

```
{
  "actions": [
    "read",
```

```

    "use",
    "run",
    "configure",
    "setPermissions"
  ],
  "userId": "userID",      ❶
  "domainId": "workspace",
  "instanceId": "workspaceID" ❷
}

```

- ❶ `userId` パラメーターは、特定のパーミッションが付与されたユーザーの ID です。
- ❷ `instanceId` パラメーターは、すべてのユーザーのパーミッションを取得するリソースの ID です。

## 9.3. 認証の設定

**CodeReady Workspaces** は、ユーザー認可のパーミッションモデルを使用します。

### 9.3.1. 認証およびユーザー管理

Red Hat CodeReady Workspaces は **RH-SSO** を使用してユーザーの作成、インポート、管理、削除、および認証を行います。RH-SSO は、ビルトイン認証メカニズムとユーザーストレージを使用します。サードパーティーのアイデンティティ管理システムを使用してユーザーを作成し、認証できます。CodeReady Workspaces リソースへのアクセスを要求する場合に、Red Hat CodeReady Workspaces には RH-SSO トークンが必要です。

ローカルユーザーおよびインポートされたフェデレーションユーザーは、プロファイルにメールアドレスが必要です。

デフォルトの RH-SSO 認証情報は **admin:admin** です。Red Hat CodeReady Workspaces への初回ログイン時に、**admin:admin** 認証情報を使用できます。これにはシステム権限があります。

**RH-SSO URL を特定します。**

OpenShift Web コンソールおよび RH-SSO プロジェクトに移動します。

### 9.3.2. RH-SSO と連携する CodeReady Workspaces の設定

デプロイメントスクリプトは RH-SSO を設定します。以下のフィールドを使用して **codeready-public** クライアントを作成します。

- **Valid Redirect URIs:** この URL を使用して CodeReady Workspaces にアクセスします。
- **Web Origins**

以下は、RH-SSO と連携するように CodeReady Workspaces を設定する場合の一般的なエラーです。

#### 無効な `redirectURI` エラー

エイリアスの **myhost** で CodeReady Workspaces にアクセスし、元の **CHE\_HOST** が **1.1.1.1** の場合に発生します。このエラーが発生した場合は、RH-SSO 管理コンソールに移動し、有効なリダイレクト URI が設定されていることを確認してください。


#### CORS エラー

無効な Web Origin がある場合に発生します。


### 9.3.3. RH-SSO トークンの設定


ユーザートークンの有効期間は、デフォルトで 30 分後に切れます。


以下の RH-SSO トークン設定を変更することができます。


Che 


General Login Keys Email Themes Cache **Tokens** Client Registration Security Defenses


Revoke Refresh Token   OFF


SSO Session Idle    ▼


SSO Session Max    ▼


Offline Session Idle    ▼


Access Token Lifespan    ▼


Access Token Lifespan For Implicit Flow    ▼

Client login timeout    ▼

Login timeout    ▼

Login action timeout    ▼

User-Initiated Action Lifespan    ▼

Default Admin-Initiated Action Lifespan    ▼

### 9.3.4. ユーザーフェデレーションの設定

RH-SSO は、外部ユーザーデータベースのフェデレーションを行い、LDAP および Active Directory をサポートします。ストレージプロバイダーを選択する前に、接続をテストし、ユーザーを認証できます。

プロバイダーの追加方法については、RH-SSO ドキュメントの[ユーザーストレージのフェデレーション](#)についてのページを参照してください。

複数の LDAP サーバーを指定するには、RH-SSO ドキュメントの[LDAP および Active Directory](#) についてのページを参照してください。

### 9.3.5. ソーシャルアカウントおよびブローカーを使用した認証の有効化

RH-SSO は、GitHub、OpenShift、および Facebook や Twitter などの最も一般的に使用されるソーシャルネットワークの組み込みサポートを提供します。[GitHub でログインを有効にする](#)方法については、RH-SSO ドキュメントを参照してください。

#### 9.3.5.1. GitHub OAuth の設定

GitHub の OAuth では、GitHub への SSH キーの自動アップロードを許可します。

## 前提条件

- **oc** ツールが利用できる。

## 手順

- CodeReady Workspaces URL をアプリケーションの **Homepage URL** の値として、また RH-SSO GitHub エンドポイント URL を認証コールバック URL の値として使用して [GitHub に OAuth アプリケーション](#) を作成します。デフォルト値は、それぞれ **https://codeready-openshift-workspaces.<DOMAIN>** および **https://keycloak-openshift-workspaces.<DOMAIN>/auth/realms/codeready/broker/github/endpoint** です。<DOMAIN>はOpenShiftクラスタードメインです。
  1. CodeReady Workspaces がデプロイされているプロジェクトで新規シークレットを作成します。

```
$ oc apply -f - <<EOF
kind: Secret
apiVersion: v1
metadata:
  name: github-oauth-config
  namespace: <...> ❶
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: github
type: Opaque
data:
  id: <...> ❷
  secret: <...> ❸
EOF
```

- ❶ CodeReady Workspaces namespace。デフォルトは openshift-workspaces です。
- ❷ base64 でエンコードされた GitHub OAuth クライアント ID
- ❸ base64 でエンコードされた GitHub OAuth クライアントシークレット

2. CodeReady Workspaces がすでにインストールされている場合は、RH-SSO コンポーネントのロールアウトが完了するまで待機します。

### 9.3.5.2. 自己署名 TLS 証明書を使用する Bitbucket サーバーの設定

本章では、CodeReady Workspaces サーバーおよびワークスペースコンポーネントが BB で信頼される接続を確立できるように、自己署名 TLS 証明書を使用する Bitbucket (BB)サーバーを設定する方法を説明します。

- 追加の TLS 証明書および **gitSelfSign** 証明書用の ConfigMap の作成。これにより、以下が可能になります。
  - devfile URL を使用したファクトリーの起動。
  - プロジェクトのインポートおよびクローン作成。



## 注記

- BB サーバー側で OAuth1 認証を設定します。詳細は、[Bitbucket Server OAuth 1 の設定](#)を参照してください。
- BB サーバーが自己署名 TLS 証明書で設定されている場合にのみ、追加の証明書をインポートする ConfigMap を作成する必要があります。これらの証明書は、CodeReady Workspaces サーバーおよびワークスペース内の適切な機能に必要であり、特定のリポジトリに関連する Git 操作を実行するために使用します。

## 前提条件

- Base64 ASCII 形式でエクスポートされた BB サーバーの認証局(CA)の値。また、**ca.crt** ファイルに保存されます。
- CodeReady Workspaces のインスタンス

## 手順

1. BB サーバーの CA を CodeReady Workspaces サーバーにプロビジョニングし、BB サーバーに保存されている devfile を読み取ることができるようにします。これを実行するには、以下の ConfigMap を **openshift-workspaces** プロジェクトに追加します。

```
$ oc create configmap bitbucket-ca-cert-for-factory --from-file=ca.crt -n openshift-workspaces
```

```
$ oc label configmap bitbucket-ca-cert-for-factory app.kubernetes.io/part-of=che.eclipse.org
app.kubernetes.io/component=ca-bundle -n openshift-workspaces
```

2. Git 操作を使用できるように、BB サーバーの CA を CodeReady Workspaces サーバーにプロビジョニングします。これを実行するには、新規 ConfigMap を **openshift-workspaces** プロジェクトに追加します。

```
$ oc create configmap che-git-self-signed-cert --from-file=ca.crt --from-literal=githost=<bitbucket_server_url> -n openshift-workspaces
```

3. CheCluster カスタムリソース(CR)を編集して、CodeReady Workspaces サーバーを設定します。

```
spec:
  server:
    # ...
    gitSelfSignedCert: <boolean> ①
```

- ① 自己署名証明書を使用する BB サーバーには **true** を使用します。デフォルト値は **false** です。

- 詳細は、[CodeReady Workspaces サーバーコンポーネントの高度な設定オプション](#)を参照してください。

## 参照資料

- Bitbucket CA 証明書を CodeReady Workspaces に追加します。[Importing untrusted TLS certificates to CodeReady Workspaces](#)を参照してください。

### 9.3.5.3. OAuth1を使用するように Bitbucket および CodeReady Workspaces 統合を設定

以下のセクションでは、Bitbucket (BB)リポジトリで読み取りおよび書き込み操作を実行するのに必要な OAuth1 認証の設定を説明します。**clone**、**push** などの許可された Git 操作で BB リポジトリを使用するには、最初に CodeReady Workspaces で BB エンドポイントを登録し、OAuth1 認証を設定します。



#### 注記

この手順では、以下が必要です。

- RSA 鍵ペアの生成
- コンシューマーのキーペアの生成
- BB 側でのアプリケーションリンクの作成
- CodeReady Workspaces サーバー側での BB の設定

この手順では、Bitbucket Server の OAuth1 をアクティベートして以下を実行する方法を説明します。

- Bitbucket Server でホストされる devfile を使用します。
- これは CodeReady Workspaces が [Bitbucket Server Personal アクセストークン](#) を取得し、更新できるようにします。

#### 前提条件

- **oc** ツールが利用できる。
- Bitbucket サーバーは、CodeReady Workspaces サーバーから利用できます。
- CodeReady Workspaces のインスタンス

#### 手順

1. RSA キーペアと公開鍵の省略バージョンを生成します。

```
$ openssl genrsa -out <private.pem> 2048
```

```
$ openssl rsa -in <private.pem> -pubout > <public.pub>
```

```
$ openssl pkcs8 -topk8 -inform pem -outform pem -nocrypt -in <private.pem> -out <privatepkcs8.pem>
```

```
$ cat <public.pub> | sed 's/-----BEGIN PUBLIC KEY-----//g' | sed 's/-----END PUBLIC KEY-----//g' | tr -d '\n' > <public-stripped.pub>
```

2. コンシューマーキーと共有シークレットを生成します。

```
$ openssl rand -base64 24 > <bitbucket_server_consumer_key>
```

```
$ openssl rand -base64 24 > <bitbucket_shared_secret>
```



3. Bitbucket で [アプリケーションリンク](#) を設定し、CodeReady Workspaces から Bitbucket サーバーへの通信を有効にします。
  - a. Bitbucket Server で上部のナビゲーションバーをクリックし、**Administration > Application Links** に移動します。
  - b. アプリケーション URL: `\https://codeready-<openshift_deployment_name>.<domain_name>` を入力し、**Create new link** ボタンをクリックします。
  - c. **No response was received from the URL** という警告メッセージが表示されたら **Continue** ボタンをクリックします。
  - d. **Link Applications** フォームを入力し、**Continue** ボタンをクリックします。

Application Name

<CodeReady Workspaces>

Application Type

Generic Application

Service Provider Name

<CodeReady Workspaces>

Consumer Key

<bitbucket\_server\_consumer\_key> ファイルの内容を貼り付けます。

Shared secret

<bitbucket\_shared\_secret> ファイルの内容を貼り付けます。

Request Token URL

<Bitbucket Server URL>/plugins/servlet/oauth/request-token

アクセストークン URL

<Bitbucket Server URL>/plugins/servlet/oauth/access-token

Authorize URL

<Bitbucket Server URL>/plugins/servlet/oauth/access-token

Create incoming link

Enabled

- e. **Link Applications** フォームを入力し、**Continue** ボタンをクリックします。

Consumer Key

<bitbucket\_server\_consumer\_key> ファイルの内容を貼り付けます。

Consumer name

<CodeReady Workspaces>

Public Key

<public-stripped.pub> ファイルの内容を貼り付けます。

4. コンシューマーおよびプライベートキーが含まれる OpenShift シークレットを CodeReady Workspaces プロジェクトに作成します。

```
$ oc apply -f - <<EOF
kind: Secret
apiVersion: v1
metadata:
```

```

name: bitbucket-oauth-config
namespace: <CodeReady Workspaces-namespace> ❶
labels:
  app.kubernetes.io/component: oauth-scm-configuration
  app.kubernetes.io/part-of: che.eclipse.org
annotations:
  che.eclipse.org/oauth-scm-server: bitbucket
  che.eclipse.org/scm-server-endpoint: '<scm-server-endpoint>' ❷
type: Opaque
data:
  private.key: '<user-private-key>' ❸
  consumer.key: '<bitbucket_server_consumer_key>' ❹
EOF

```

- ❶ CodeReady Workspaces namespace。デフォルトは openshift-workspaces です。
- ❷ Bitbucket サーバー URL
- ❸ 最初の行と最後の行のない <privatepkcs8.pem> ファイルの base64 でエンコードされたコンテンツ。
- ❹ <bitbucket\_server\_consumer\_key> ファイルの base64 でエンコードされたコンテンツ。

## 例

```

#!/usr/bin/env bash

NS=${1:-eclipse-che}
CONSUMER_KEY=$(cat ./certs/bitbucket_server_consumer_key)
PRIVATE_KEY=$(cat ./certs/privatepkcs8.pem | sed 's/-----BEGIN PRIVATE KEY-----//g' | sed 's/-----END PRIVATE KEY-----//g' | tr -d '\n')
BITBUCKET_HOST='<your-bitbucket-host-here>'
unameOut="$(uname -s)"

case "$unameOut" in
  Linux*)    BASE64_FUNC='base64 -w 0';;
  Darwin*)   BASE64_FUNC='base64';;
  CYGWIN*)   BASE64_FUNC='base64 -w 0';;
  MINGW*)   BASE64_FUNC='base64 -w 0';;
  *)         BASE64_FUNC='base64 -w 0'
esac

cat <<EOF | oc apply -n $NS -f -
kind: Secret
apiVersion: v1
metadata:
  name: bitbucket-oauth-config
  labels:
    app.kubernetes.io/part-of: che.eclipse.org
    app.kubernetes.io/component: oauth-scm-configuration
  annotations:
    che.eclipse.org/oauth-scm-server: bitbucket
    che.eclipse.org/scm-server-endpoint: https://$BITBUCKET_HOST

```

```

type: Opaque
data:
  private.key: $(echo -n $PRIVATE_KEY | $BASE64_FUNC)
  consumer.key: $(echo -n $CONSUMER_KEY | $BASE64_FUNC)
EOF

```

- この [GitHub サンプル](#) のスクリプト全体を参照してください。

## 関連情報

- [Bitbucket Server overview](#)
- [Download Bitbucket Server](#)
- [Bitbucket Server Personal access tokens](#)
- [How to generate public key to application link 3rd party applications](#)
- [Using AppLinks to link to other applications](#)
- [SCM サーバーのプライベートリポジトリでのユーザーの認証](#)

### 9.3.5.4. GitLab サーバーの設定

GitLab サーバーをプロジェクトソースサプライヤーとして使用するには、**CHE\_INTEGRATION\_GITLAB\_SERVER\_ENDPOINTS** プロパティを使用して GitLab サーバーの URL を CodeReady Workspaces に登録し、登録するサーバーのホスト名を指定します。

## 例

```
https://gitlab.apps.cluster-2ab2.2ab2.example.opentlc.com/
```

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

- Operator - [Operator](#) を使用した CodeReady Workspaces サーバーの詳細設定についてを参照

## 関連情報

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

### 9.3.5.5. Configuring GitLab OAuth2

GitLab の OAuth2 では、プライベート GitLab リポジトリからファクトリーを受け入れることができます。

## 前提条件

- GitLab サーバーが実行中であり、CodeReady Workspaces から利用可能

## 手順

- CodeReady Workspaces をアプリケーションの **Name** として使用し、RH-SSO GitLab エンドポイント URL を **Redirect URI** の値として使用して、GitLab に [承認済み OAuth2 アプリケーション](#) を作成します。コールバック URL のデフォルト値は **https://keycloak-openshift-**

`workspaces.<DOMAIN>/auth/realms/codeready/broker/gitlab/endpoint` です。ここで、`<DOMAIN>` は OpenShift クラスタドメインです。 **Application ID** および **Secret** の値を保存します。GitLab OAuth 2 アプリケーションの 3 つのタイプはすべて、ユーザー所有、グループ所有、およびインスタンス全体に対応します。

1. GitLab サーバーを参照する RH-SSO でカスタム OIDC プロバイダーリンクを作成します。以下のフィールドに入力します。

#### クライアント ID

直前の手順で GitLab サーバーにより提供される **Application ID** フィールドの値。

#### クライアントのシークレット

直前の手順で GitLab サーバーにより提供される **Secret** フィールドの値。

#### 認証 URL

`https://<GITLAB_DOMAIN>/oauth/authorize` 形式の URL

#### トークン URL

`https://<GITLAB_DOMAIN>/oauth/token` 形式の URL

#### スコープ

`api write_repository openid` のセットを含める必要のある（ただしこれに限定されない）一連のスコープ

#### トークンの保存

有効にする必要があります。

#### 読み取り可能なトークンの保存

有効にする必要があります。



#### 注記

- `<GITLAB_DOMAIN>` を GitLab インストールの URL およびポートに置き換えます。

2. `CHE_INTEGRATION_GITLAB_OAUTH_ENDPOINT` プロパティを使用して、GitLab インスタンス URL を CodeReady Workspaces で有効にされた OAuth 2 サポートに登録します。



#### 警告

- GitLab インスタンスの URL は、設定された GitLab 統合エンドポイントの一覧に存在し、`CHE_INTEGRATION_GITLAB_SERVER_ENDPOINTS` プロパティで設定します。

## 関連情報

CodeReady Workspaces が TLS キーに関連する GitLab にアクセスする場合は、以下のドキュメントを参照してください。

- [信頼できない TLS 証明書の CodeReady Workspaces へのインポート](#)

- [自己署名証明書を使用した Git リポジトリをサポートする CodeReady Workspaces のデプロイ](#)

### 9.3.6. プロトコルベースのプロバイダーの使用

RH-SSO は [SAML v2.0](#) プロトコルおよび [OpenID Connect v1.0](#) プロトコルをサポートします。

### 9.3.7. RH-SSO を使用したユーザーの管理

ユーザーインターフェースでユーザーを追加し、削除し、編集できます。詳細は、[RH-SSO ユーザー管理](#)について参照してください。

### 9.3.8. 外部 RH-SSO インストールを使用するように CodeReady Workspaces を設定する

デフォルトでは、CodeReady Workspaces インストールには、専用の RH-SSO インスタンスのデプロイメントが含まれます。ただし、外部の RH-SSO を使用することも可能です。このオプションは、すでに定義されたユーザーを含む既存の RH-SSO インスタンス (複数のアプリケーションが使用する会社全体の RH-SSO サーバーなど) がある場合に役立ちます。

表9.3 サンプルで使用されるプレースホルダー

<b>&lt;provider-realm-name&gt;</b>	CodeReady Workspaces で使用することが意図された RH-SSO レalm 名
<b>&lt;oidc-client-name&gt;</b>	<b>&lt;provider-realm-name&gt;</b> で定義された <b>oidc</b> クライアントの名前
<b>&lt;auth-base-url&gt;</b>	外部 RH-SSO サーバーのベース URL

#### 前提条件

- RH-SSO の外部インストールの管理コンソールで、CodeReady Workspaces に接続することが意図されたユーザーが含まれる [レalm](#) を定義します。

The screenshot shows the configuration page for a realm named 'realm-for-users'. The left sidebar contains navigation options: Realm, Settings, Clients, Client Scopes, Roles, Identity, Providers, User, Federation, Authentication, and Manage. The main content area is titled 'Realm-for-users' and has tabs for General, Login, Keys, Email, Themes, Cache, Tokens, Client Registration, and Security Defenses. The 'General' tab is active, showing the following fields:

- Name**: realm-for-users
- Display name**: (empty)
- HTML Display name**: (empty)
- Frontend URL**: (empty)
- Enabled**: ON
- User-Managed Access**: OFF
- Endpoints**: OpenID Endpoint Configuration, SAML 2.0 Identity Provider Metadata

Buttons for 'Save' and 'Cancel' are visible at the bottom.

- この **realm** では、CodeReady Workspaces がユーザーの認証に使用する [OIDC クライアント](#) を定義します。以下は、正しい設定のあるクライアントの例です。

The screenshot shows the 'Public-client' configuration page in the Admin Console. The left sidebar contains navigation options like 'Configure', 'Manage', and 'Clients'. The main content area is titled 'Public-client' and has tabs for 'Settings', 'Roles', 'Client Scopes', 'Mappers', 'Scope', 'Revocation', and 'Sessions'. The 'Settings' tab is active, showing various configuration fields. Several fields are highlighted with red boxes: 'Client Protocol' (set to 'openid-connect'), 'Access Type' (set to 'public'), 'Valid Redirect URIs' (containing two entries: 'http://che-eclipse-che.apps-crc.testing/\*' and 'https://che-eclipse-che.apps-crc.testing/\*'), and 'Web Origins' (containing two entries: 'http://che-eclipse-che.apps-crc.testing' and 'https://che-eclipse-che.apps-crc.testing'). Other fields include 'Client ID' (public-client), 'Name', 'Description', 'Enabled' (ON), 'Consent Required' (OFF), 'Login Theme', 'Standard Flow Enabled' (ON), 'Implicit Flow Enabled' (OFF), 'Direct Access Grants Enabled' (ON), 'Root URL', 'Base URL', and 'Admin URL'.

## 注記

- Client Protocol は **openid-connect** である必要があります。
- Access Type は **public** である必要があります。CodeReady Workspaces は **public** アクセスタイプのみをサポートします。
- Valid Redirect URIs には、**http** プロトコルを使用する URL と **https** を使用する URL の 2 つ以上の CodeReady Workspaces サーバーに関連する URI が含まれる必要があります。これらの URI には CodeReady Workspaces サーバーのベース URL (この後に /\* ワイルドカードが続く) が含まれる必要があります。
- Web Origins には、**http** プロトコルを使用する URI と **https** を使用する URI の 2 つ以上の CodeReady Workspaces サーバーに関連する URI が含まれる必要があります。これらの URI には、CodeReady Workspaces サーバーのベース URL (ホストの後はパスがない) が含まれる必要があります。URI の数は、インストールされている製品ツールの数によって異なります。

- デフォルトの OpenShift OAuth サポートを使用する CodeReady Workspaces では、ユーザー

認証は、OpenShift OAuth と RH-SSO の統合に依存します。これにより、ユーザーは OpenShift ログインで CodeReady Workspaces にログインでき、独自のワークスペースを個人の OpenShift プロジェクトに作成することができます。

これには、OpenShift の「RH-SSO Identity Provider」を設定する必要があります。外部 RH-SSO を使用する場合は、手動で RH-SSO を設定します。手順については、[OpenShift 3](#) または [OpenShift 4](#) のいずれかに該当する RH-SSO ドキュメントを参照してください。

- 設定した RH-SSO のオプションには、**Store Tokens** および **Stored Tokens Readable** が有効になっています。

## 手順

1. **CheCluster** カスタムリソース (CR) に以下のプロパティを設定します。

```
spec:
  auth:
    externalIdentityProvider: true
    identityProviderURL: <auth-base-url>
    identityProviderRealm: <provider-realm-name>
    identityProviderClientId: <oidc-client-name>
```

2. OpenShift OAuth サポートを有効にして CodeReady Workspaces をインストールする場合は、**CheCluster** カスタムリソース (CR) に以下のプロパティを設定します。

```
spec:
  auth:
    openShifttoAuth: true
    # Note: only if the OpenShift "RH-SSO Identity Provider" alias is different from 'openshift-v3'
    # or 'openshift-v4'
  server:
    customCheProperties:
      CHE_INFRA_OPENSHIFT_OAUTHIDENTITYPROVIDER: <OpenShift "RH-SSO Identity
      Provider" alias>
```

### 9.3.9. SMTP およびメール通知の設定

Red Hat CodeReady Workspaces は事前に設定された SMTP サーバーを提供しません。

RH-SSO で SMTP サーバーを有効にするには、以下を実行します。

1. **che realm settings > Email** の順に移動します。
2. ホスト、ポート、ユーザー名、およびパスワードを指定します。

Red Hat CodeReady Workspaces は、登録、メールの確認、パスワードの復旧、およびログインの失敗についてのデフォルトのテーマを使用します。

### 9.3.10. 自己登録の有効化

自己登録により、ユーザーは CodeReady Workspaces サーバー URL にアクセスして、CodeReady Workspaces インスタンスに自己登録できます。

OpenShift OAuth サポートなしでインストールされた CodeReady Workspaces では、自己登録はデフォルトで無効にされるため、ログインページで新規ユーザーを登録するオプションは利用できません。

## 前提条件

- 管理者としてログインしている。

## 手順

ユーザーの自己登録を有効にするには、以下を実行します。

1. 左側の **Realm Settings** メニューに移動し、**Login** タブを開きます。
2. **User registration** オプションを **On** に設定します。

## 9.4. OPENSIFT OAUTH の設定

ユーザーが OpenShift と対話できるようにするには、まず OpenShift クラスターに対して認証する必要があります。OpenShift OAuth は、ユーザーが取得された OAuth アクセストークンを使って API 経由でクラスターに対して自らを証明するプロセスです。

[OpenShift Connector の概要](#) を使用した認証は、CodeReady Workspaces ユーザーが OpenShift クラスターで認証できるようにする方法になります。

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

### 9.4.1. 初期ユーザーでの OpenShift OAuth の設定

#### 前提条件

- **oc** ツールが利用できる。
- **crwctl** 管理ツールが利用可能である。[crwctl 管理ツールの使用](#) を参照してください。

#### 手順

- クラスターで OpenShift アイデンティティプロバイダーを設定します。[アイデンティティプロバイダー設定について](#) 参照してください。  
ユーザーが OpenShift 「RH-SSO Identity Provider」 の設定ステップを省略し、OpenShift クラスターに設定済みの RH-SSO が含まれていない場合、CodeReady Workspaces は **HTPasswd** アイデンティティプロバイダーの初期 OpenShift ユーザーを作成します。このユーザーの認証情報は、**openshift-config** namespace にある **openshift-oauth-user-credentials** シークレットに保存されます。

OpenShift クラスターおよび CodeReady Workspaces インスタンスにログインするための認証情報を取得します。

1. OpenShift ユーザー名を取得します。

```
$ oc get secret openshift-oauth-user-credentials -n openshift-config -o json | jq -r '.data.user' | base64 -d
```

2. OpenShift ユーザーパスワードを取得します。

```
$ oc get secret openshift-oauth-user-credentials -n openshift-config -o json | jq -r '.data.password' | base64 -d
```



- [OperatorHub](#) または `crwctl` を使用して CodeReady Workspaces をデプロイする場合は、[crwctl server:deploy 仕様](#) についての章を参照してください。OpenShift OAuth はデフォルトで有効にされます。

### 9.4.2. OpenShift の初期 OAuth ユーザーをプロビジョニングせずに OpenShift OAuth を設定する

以下の手順では、初期 OAuth ユーザーをプロビジョニングせずに OpenShift OAuth を設定する方法を説明します。

#### 前提条件

- `crwctl` 管理ツールが利用可能である。[crwctl 管理ツールの使用](#) を参照してください。

#### 手順

1. Operator を使用して CodeReady Workspaces をインストールしている場合は、`codeready-workspaces` カスタムリソースに以下の値を設定します。

```
spec:
  auth:
    openShifttoAuth: true
    initialOpenShiftOAuthUser: "
```

2. `crwctl` ツールを使用して CodeReady Workspaces をインストールしている場合は、`--che-operator-cr-patch-yaml` フラグを使用します。

```
$ crwctl server:deploy --che-operator-cr-patch-yaml=patch.yaml ...
```

`patch.yaml` ファイルには以下を含める必要があります。

```
spec:
  auth:
    openShifttoAuth: true
    initialOpenShiftOAuthUser: "
```

### 9.4.3. OpenShift 初期 OAuth ユーザーの削除

以下の手順では、Red Hat CodeReady Workspaces がプロビジョニングする OpenShift の初期の OAuth ユーザーを削除する方法を説明します。

#### 前提条件

- `oc` ツールがインストールされている。
- OpenShift で実行している Red Hat CodeReady Workspaces のインスタンス。
- `oc` ツールを使用して OpenShift クラスターにログインしている。

#### 手順

1. `codeready-workspaces` カスタムリソースを更新します。

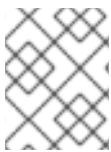
```
$ oc patch checluster/codeready-workspaces -n openshift-workspaces --type=json -p \
[{"op": "replace", "path": "/spec/auth/initialOpenShiftOAuthUser", "value": false}]
```

## 9.5. ユーザーデータの削除

### 9.5.1. GDPR に準拠したユーザーデータの削除

一般データ保護規則 (GDPR: General Data Protection Regulation) では、個人データの消去を求める個人の権利が施行されています。

以下の手順では、クラスターおよび RH-SSO データベースからユーザーのデータを削除する方法を説明します。



#### 注記

以下のコマンドは、**-n** オプションのユーザー例として、デフォルトの OpenShift プロジェクト **openshift-workspaces** を使用します。

#### 前提条件

- ユーザーまたは管理者の認証トークン。ユーザーアカウントにバインドされているデータ以外のデータを削除する場合は、**admin** 権限が必要になります。**admin** は、**CHE\_SYSTEM\_ADMIN\_NAME** および **CHE\_SYSTEM\_SUPER\_PRIVILEGED\_MODE = true** カスタムリソース定義を使用して事前に作成され、有効にされる特別な CodeReady Workspaces 管理者アカウントです。

```
spec:
  server:
    customCheProperties:
      CHE_SYSTEM_SUPER_PRIVILEGED_MODE: 'true'
      CHE_SYSTEM_ADMIN_NAME: '<admin-name>'
```

必要に応じて、以下のコマンドを使用して **admin** ユーザーを作成します。

```
$ oc patch checluster/codeready-workspaces \
  --type merge \
  -p '{"spec": {"server": {"customCheProperties": {"CHE_SYSTEM_SUPER_PRIVILEGED_MODE": "true"} }}}' \
  -n openshift-workspaces
```

```
$ oc patch checluster/codeready-workspaces \
  --type merge \
  -p '{"spec": {"server": {"customCheProperties": {"CHE_SYSTEM_ADMIN_NAME": "<admin-name>"}}} \
  -n openshift-workspaces
```



## 注記

すべてのシステムパーミッションは管理ユーザーに付与されます。管理ユーザーを設定するには、**CHE\_SYSTEM\_ADMIN\_NAME** プロパティを使用します。デフォルト値は **admin** です。システムのパーミッションは CodeReady Workspaces サーバーの起動時に付与されます。ユーザーレコードが CodeReady Workspaces ユーザーデータベースにない場合は、ユーザーの最初のログイン後にパーミッションが付与されます。

### 認証トークンの権限:

- **admin**: すべてのユーザーのすべての個人データを削除できます。
  - **user**: ユーザーに関連するデータのみを削除できます。
- ユーザーまたは管理者が、CodeReady Workspaces がデプロイされた状態で OpenShift クラスターにログインしている。
  - ユーザー ID が取得されます。以下のコマンドを使用してユーザー ID を取得します。
    - 現行ユーザーの場合:

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>/api/user'
```

- 名前でユーザーを検索するには、以下を実行します。

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>/api/user/find?name=<username>'
```

- メールでユーザーを検索するには、以下を実行します。

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://<codeready-<openshift_deployment_name>.<domain_name>/api/user/find?email=<email>'
```

### ユーザー ID を取得する例

この例では、ローカルユーザーの名前として **vparfono** を使用します。

```
$ curl -X GET \
  --header 'Authorization: Bearer <user-token>' \
  'https://che-vp-che.apps.che-dev.x6e0.p1.openshiftapps.com/api/user/find?name=vparfono'
```

ユーザー ID は、curl コマンド出力の下部にあります。

```
{
  "name": "vparfono",
  "links": [
    {
```

```

    .
    .
    .
  }
],
"email": "vparfono@redhat.com",
"id": "921b6f33-2657-407e-93a6-fb14cf2329ce"
}

```

## 手順

1. **codeready-workspaces CheCluster Custom** リソース (CR) 定義を更新して、RH-SSO データベースからユーザーのデータの削除を許可します。

```

$ oc patch checluster/codeready-workspaces \
  --patch '{"spec":{"server":{"customCheProperties":\
{"CHE_KEYCLOAK_CASCADE_USER_REMOVAL_ENABLED": "true"}}}}' \
  --type=merge -n openshift-workspaces

```

2. API を使用してデータを削除します。

```

$ curl -i -X DELETE \
  --header 'Authorization: Bearer <user-token>' \
  https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/<user-
id>

```

## 検証

以下のコマンドを実行すると、コード **204** が API 応答として返されます。

```

$ curl -i -X DELETE \
  --header 'Authorization: Bearer <user-token>' \
  https://<codeready-<openshift_deployment_name>.<domain_name>>/api/user/<user-id>

```

## 関連情報

すべてのユーザーのデータを削除するには、[CodeReady Workspaces のアンインストール 手順](#)に従います。