



OpenShift Container Platform 4.5

イメージ

OpenShift Container Platform でのイメージおよびイメージストリームの作成および管理

OpenShift Container Platform 4.5 イメージ

OpenShift Container Platform でのイメージおよびイメージストリームの作成および管理

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 © 2022 | You need to change the HOLDER entity in the en-US/Images.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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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

本書では、OpenShift Container Platform でイメージおよびイメージストリームを作成し、管理する方法を説明します。さらに、テンプレートの使用方法についても説明します。

目次

第1章 CLUSTER SAMPLES OPERATOR の設定	6
1.1. 前提条件	6
1.2. CLUSTER SAMPLES OPERATOR について	6
1.2.1. Cluster Samples Operator の管理状態の使用	6
1.2.1.1. ネットワークが制限されたインストール	7
1.2.1.2. 初期のネットワークアクセスが設定された状態でのネットワークが制限されたインストール	7
1.2.2. Cluster Samples Operator でのイメージストリームのインポートの追跡およびエラー回復	8
1.3. 関連情報	8
1.4. CLUSTER SAMPLES OPERATOR の設定パラメーター	8
1.4.1. 設定の制限	9
1.4.2. 条件	10
1.5. CLUSTER SAMPLES OPERATOR 設定へのアクセス	10
関連情報	11
第2章 代替レジストリーでの CLUSTER SAMPLES OPERATOR の使用	12
2.1. ミラーレジストリーについて	12
2.1.1. ミラーホストの準備	12
2.1.2. バイナリーのダウンロードによる CLI のインストール	12
2.1.2.1. Linux への CLI のインストール	13
2.1.2.2. Windows での CLI のインストール	13
2.1.2.3. macOS への CLI のインストール	14
2.2. イメージのミラーリングを可能にする認証情報の設定	14
2.3. OPENSIFT CONTAINER PLATFORM イメージリポジトリーのミラーリング	16
2.4. 代替のレジストリーまたはミラーリングされたレジストリーでの CLUSTER SAMPLES OPERATOR イメージストリームの使用	19
第3章 コンテナ、イメージおよびイメージストリームについて	22
3.1. イメージ	22
3.2. コンテナ	22
3.3. イメージレジストリー	23
3.4. イメージリポジトリー	23
3.5. イメージタグ	23
3.6. イメージ ID	23
3.7. イメージストリームの使用	23
3.8. イメージストリームタグ	24
3.9. イメージストリームイメージ	25
3.10. イメージストリームトリガー	25
3.11. 追加リソース	25
第4章 イメージの作成	26
4.1. コンテナのベストプラクティスについて	26
4.1.1. コンテナイメージの一般的なガイドライン	26
イメージの再利用	26
タグ内の互換性の維持	26
複数プロセスの回避	26
ラッパースクリプトでの exec の使用	26
一時ファイルの消去	27
正しい順序での命令の指定	27
重要なポートのマーク付け	28
環境変数の設定	28
デフォルトのパスワードの回避	28
SSHD の回避	28

永続データ向けのボリュームの使用	28
4.1.2. OpenShift Container Platform 固有のガイドライン	29
Source-To-Image (S2I) 向けのイメージの有効化	29
任意のユーザー ID のサポート	29
イメージ間通信でのサービスの使用	30
共通のライブラリーの提供	30
設定での環境変数の使用	30
イメージのメタデータ設定	31
クラスタリング	31
ロギング	31
Liveness および Readiness プローブ	31
テンプレート	31
4.2. イメージへのメタデータの組み込み	31
4.2.1. イメージメタデータの定義	32
4.3. SOURCE-TO-IMAGE によるソースコードからのイメージの作成	33
4.3.1. Source-to-Image ビルドプロセスについて	33
4.3.2. Source-to-Image スクリプトの作成方法	33
4.4. SOURCE-TO-IMAGE イメージのテストについて	36
4.4.1. テスト要件について	36
4.4.2. スクリプトおよびツールの生成	37
4.4.3. ローカルでのテスト	37
4.4.4. テストの基本的なワークフロー	37
4.4.5. イメージのビルドでの OpenShift Container Platform の使用	38
第5章 イメージの管理	39
5.1. イメージの管理の概要	39
5.1.1. イメージの概要	39
5.2. イメージのタグ付け	39
5.2.1. イメージタグ	39
5.2.2. イメージタグの規則	39
5.2.3. タグのイメージストリームへの追加	40
5.2.4. タグのイメージストリームからの削除	41
5.2.5. イメージストリームでのイメージの参照	41
5.3. イメージプルポリシー	42
5.3.1. イメージプルポリシーの概要	43
5.4. イメージプルシークレットの使用	43
5.4.1. Pod が複数のプロジェクト間でイメージを参照できるようにする設定	44
5.4.2. Pod が他のセキュリティ保護されたレジストリーからイメージを参照できるようにする設定	44
5.4.2.1. 委任された認証を使用したプライベートレジストリーからのプル	45
5.4.3. グローバルクラスタのプルシークレットの更新	45
第6章 イメージストリームの管理	47
6.1. イメージストリームの使用	47
6.2. イメージストリームの設定	48
6.3. イメージストリームイメージ	49
6.4. イメージストリームタグ	49
6.5. イメージストリーム変更トリガー	50
6.6. イメージストリームのマッピング	51
6.7. イメージストリームの使用	53
6.7.1. イメージストリームについての情報の取得	53
6.7.2. タグのイメージストリームへの追加	54
6.7.3. 外部イメージのタグの追加	55
6.7.4. イメージストリームタグの更新	56

6.7.5. イメージストリームタグの削除	56
6.7.6. イメージストリームタグの定期的なインポートの設定	56
6.8. プライベートレジストリーからのイメージおよびイメージストリームのインポート	57
6.8.1. Pod が他のセキュリティー保護されたレジストリーからイメージを参照できるようにする設定	58
第7章 KUBERNETES リソースでのイメージストリームの使用	59
7.1. KUBERNETES リソースでのイメージストリームの有効化	59
第8章 イメージストリームの変更時の更新のトリガー	61
8.1. OPENSIFT CONTAINER PLATFORM リソース	61
8.2. KUBERNETES リソースのトリガー	61
8.3. KUBERNETES リソースでのイメージトリガーの設定	62
第9章 イメージ設定リソース	63
9.1. イメージコントローラー設定パラメーター	63
9.2. イメージ設定内容の設定	65
9.2.1. 特定のレジストリーの追加	66
9.2.2. 特定のレジストリーのブロック	70
9.2.3. 非セキュアなレジストリーの許可	71
9.2.4. イメージレジストリーアクセス用の追加のトラストストアの設定	73
9.2.5. イメージレジストリーのリポジトリミラーリングの設定	74
第10章 テンプレートの使用	79
10.1. テンプレートについて	79
10.2. テンプレートのアップロード	79
10.3. WEB コンソールを使用したアプリケーションの作成	79
10.4. CLI を使用してテンプレートからオブジェクトを作成する手順	80
10.4.1. ラベルの追加	80
10.4.2. パラメーターの一覧表示	80
10.4.3. オブジェクト一覧の生成	81
10.5. アップロードしたテンプレートの変更	82
10.6. インスタントアプリおよびクイックスタートテンプレートの使用	83
10.6.1. クイックスタートのテンプレート	83
10.6.1.1. Web フレームワーククイックスタートのテンプレート	84
10.7. テンプレートの作成	84
10.7.1. テンプレート記述の作成	85
10.7.2. テンプレートラベルの作成	89
10.7.3. テンプレートパラメーターの作成	89
10.7.4. テンプレートオブジェクト一覧の作成	92
10.7.5. テンプレートをバインド可能としてマーキングする	93
10.7.6. テンプレートオブジェクトフィールドの公開	93
10.7.7. テンプレートの準備ができるまで待機する	95
10.7.8. 既存オブジェクトからのテンプレートの作成	97
第11章 RUBY ON RAILS の使用	99
11.1. 前提条件	99
11.2. データベースの設定	99
11.3. アプリケーションの作成	100
11.3.1. Welcome ページの作成	101
11.3.2. OpenShift Container Platform のアプリケーションの設定	101
11.3.3. アプリケーションの Git への保存	102
11.4. アプリケーションの OPENSIFT CONTAINER PLATFORM へのデプロイ	103
11.4.1. データベースサービスの作成	103
11.4.2. フロントエンドサービスの作成	104

11.4.3. アプリケーションのルートの作成	105
第12章 イメージの使用	107
12.1. イメージの使用の概要	107
12.2. JENKINS イメージの設定	107
12.2.1. 設定とカスタマイズ	108
12.2.1.1. OpenShift Container Platform OAuth 認証	108
12.2.1.2. Jenkins 認証	109
12.2.2. Jenkins 環境変数	109
12.2.3. Jenkins へのプロジェクト間のアクセスの提供	113
12.2.4. Jenkins のボリューム間のマウントポイント	114
12.2.5. S2I (Source-To-Image) による Jenkins イメージのカスタマイズ	114
12.2.6. Jenkins Kubernetes プラグインの設定	115
12.2.7. Jenkins パーミッション	117
12.2.8. テンプレートからの Jenkins サービスの作成	118
12.2.9. Jenkins Kubernetes プラグインの使用	119
12.2.10. Jenkins メモリーの要件	121
12.2.11. その他のリソース	121
12.3. JENKINS エージェント	121
12.3.1. Jenkins エージェントイメージ	122
12.3.2. Jenkins エージェントの環境変数	122
12.3.3. Jenkins エージェントのメモリー要件	124
12.3.4. Jenkins エージェントの Gradle ビルド	124
12.3.5. Jenkins エージェント Pod の保持	125
12.4. SOURCE-TO-IMAGE (S2I)	126
12.4.1. Source-to-Image ビルドプロセスの概要	126
12.4.2. 関連情報	126
12.5. SOURCE-TO-IMAGE イメージのカスタマイズ	127
12.5.1. イメージに埋め込まれたスクリプトの呼び出し	127

第1章 CLUSTER SAMPLES OPERATOR の設定

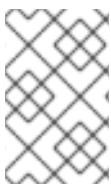
openshift namespace で動作する Cluster Samples Operator は、Red Hat Enterprise Linux (RHEL) ベースの OpenShift Container Platform イメージストリームおよび OpenShift Container Platform テンプレートを実インストールし、更新します。

1.1. 前提条件

- OpenShift Container Platform クラスタをデプロイします。

1.2. CLUSTER SAMPLES OPERATOR について

Operator はインストール時に独自にデフォルト設定オブジェクトを作成し、その後にクイックスタートテンプレートを含む、サンプルのイメージストリームおよびテンプレートを作成します。

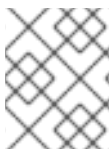


注記

認証情報を必要とする他のレジストリーからのイメージストリームのインポートを容易にするには、クラスタ管理者は、イメージのインポートに必要な Docker **config.json** ファイルの内容を含む追加のシークレットを **openshift** namespace に作成できます。

Cluster Samples Operator 設定はクラスタ全体で使用されるリソースであり、デプロイメントは **openshift-cluster-samples-operator** namespace 内に含まれます。

Cluster Samples Operator のイメージには、関連付けられた OpenShift Container Platform リリースのイメージストリームおよびテンプレートの定義が含まれます。各サンプルが作成または更新されると、Cluster Samples Operator には OpenShift Container Platform のバージョンを示すアノテーションが含まれます。Operator はこのアノテーションを使用して、各サンプルをリリースバージョンに一致させるようにします。このインベントリーの外にあるサンプルは省略されるサンプルであるために無視されます。バージョンのアノテーションが変更または削除されると、Operator が管理するサンプルに変更が加えてもそれらの変更は自動的に元に戻されます。



注記

Jenkins イメージはインストールからのイメージペイロードの一部であり、イメージストリームに直接タグ付けされます。

Cluster Samples Operator 設定リソースには、削除時に以下を消去するファイナライザーが含まれます。

- Operator 管理のイメージストリーム
- Operator 管理のテンプレート
- Operator が生成する設定リソース
- クラスタステータスのリソース

サンプルリソースの削除時に、Cluster Samples Operator はデフォルト設定を使用してリソースを再作成します。

1.2.1. Cluster Samples Operator の管理状態の使用

Cluster Samples Operator はデフォルトで **Managed** としてブートストラップされるか、またはグローバルプロキシが設定されている場合にブートストラップされます。**Managed** 状態で、Cluster Samples Operator は、イメージストリームおよびイメージをレジストリーからプルし、必要なサンプルテンプレートがインストールされた状態になるように、リソースをアクティブに管理し、コンポーネントをアクティブな状態に維持します。

以下を含む特定の状況では、Cluster Samples Operator が **Removed** としてそれ自体をブートストラップします。

- Cluster Samples Operator が、クリーンインストール後の初回起動から 3 分後に registry.redhat.io に到達できない場合。
- Cluster Samples Operator がこれが IPv6 ネットワーク上にあることを検出する場合。

ただし、Cluster Samples Operator が OpenShift Container Platform グローバルプロキシも設定されていることを検知する場合、これらのチェックはバイパスされます。



重要

現在、IPv6 インストールは registry.redhat.io によってサポートされていません。Cluster Samples Operator は、ほとんどのサンプルイメージストリームおよびイメージを registry.redhat.io からプルします。

1.2.1.1. ネットワークが制限されたインストール

registry.redhat.io にアクセスできない場合に **Removed** としてすると、ネットワークの制限がすでに有効な場合にネットワークが制限されたインストールを容易にします。ネットワークアクセスが制限されている場合に **Removed** としてインストールすると、クラスター管理者がサンプルが必要かどうかを判断する時間をより長く確保できます。これは、Cluster Samples Operator は管理状態が **Removed** に設定されている場合にサンプルのイメージストリームのインポートに失敗しているアラートを送信しないためです。Cluster Samples Operator が **Managed** として起動し、サンプルイメージストリームのインストールを試みると、インポートに失敗している場合に初回インストールから 2 時間後にアラートが開始されます。

1.2.1.2. 初期のネットワークアクセスが設定された状態でのネットワークが制限されたインストール

ネットワークが制限された環境のクラスターまたは非接続クラスターとすることが意図されたクラスターが、ネットワークアクセスがある状態で最初にインストールされた場合、Cluster Samples Operator はアクセス可能なために registry.redhat.io からコンテンツをインストールします。Cluster Samples Operator を **Removed** としてブートストラップして、必要なサンプルを判断するまでサンプルのインストールを遅らせる場合、イメージミラーのセットアップなどを行ってから、代替レジストリーで Samples Operator を使用し、ノードをカスタマイズする手順 (どちらも追加のリソースセクションにリンクされている) に従い、Cluster Samples Operator のデフォルト設定を上書きし、最初に **Removed** として起動します。

以下の追加の YAML ファイルを、`openshift-install create manifest` で作成される `openshift` ディレクトリーに置く必要があります。

managementState: Removed が設定された Cluster Samples Operator YAML ファイルのサンプル

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
metadata:
```

```

name: cluster
spec:
  architectures:
  - x86_64
  managementState: Removed

```

1.2.2. Cluster Samples Operator でのイメージストリームのインポートの追跡およびエラー回復

サンプルイメージストリームの作成または更新後に、Cluster Samples Operator はそれぞれのイメージストリームタグのイメージインポートの進捗をモニターします。

インポートが失敗すると、Cluster Samples Operator はイメージストリームイメージインポート API を使用してインポートを再試行します。これは **oc import-image** コマンドで使用されるのと同じ API であり、インポートの成功が確認されるまで約 15 分ごとに、またはイメージストリームのいずれかが **skippedImagestreams** 一覧に追加されるように Cluster Samples Operator の設定が変更されるか、または管理状態が **Removed** に変更される場合に再試行されます。

1.3. 関連情報

- Cluster Samples Operator がインストール時に削除される場合、[Cluster Samples Operator を代替レジストリーと共に使用します](#)。
- Cluster Samples Operator を初期のネットワークアクセスのある状態でネットワークが制限されたインストールで **Removed** としてブートループし、必要なサンプルを判別するまでサンプルのインストールを遅らせる場合、[代替レジストリーで Cluster Samples Operator を使用し、ノードをカスタマイズする](#) 手順に従い、Cluster Samples Operator のデフォルト設定を上書きし、最初に **Removed** として起動します。

1.4. CLUSTER SAMPLES OPERATOR の設定パラメーター

サンプルリソースは以下の設定フィールドを提供します。

パラメーター	説明
managementState	<p>Managed: Cluster Samples Operator は設定の指示に応じてサンプルを更新します。</p> <p>Unmanaged: Cluster Samples Operator は、その設定リソースオブジェクトおよび openshift namespace のイメージストリームまたはテンプレートへの更新を無視します。</p> <p>Removed: Cluster Samples Operator は openshift namespace の一連の Managed 状態のイメージストリームおよびテンプレートを除去します。これは、クラスター管理者によって作成される新規サンプルや、省略された一覧にあるサンプルを無視します。除去されると、Cluster Samples Operator は Unmanaged 状態にあるかのように機能し、サンプルリソース、イメージストリーム、またはテンプレートに対する監視イベントを無視します。</p>

パラメーター	説明
samplesRegistry	<p>イメージコンテンツについてイメージストリームがアクセスするレジストリーを指定できます。samplesRegistry はデフォルトで OpenShift Container Platform の registry.redhat.io に設定されます。</p> <div style="display: flex; align-items: flex-start;"> <div style="width: 30px; height: 150px; border: 1px solid black; background: repeating-linear-gradient(45deg, transparent, transparent 2px, black 2px, black 4px); margin-right: 10px;"></div> <div> <p>注記</p> <p>RHEL コンテンツの作成または更新は、Samples Registry が明示的に設定されていない場合、空の文字列を残す場合やこれが registry.redhat.io に設定されている場合にプルアクセスのシークレットが有効でないと開始されません。いずれの場合も、イメージのインポートは registry.redhat.io の外で機能し、これには認証情報が必要になります。</p> <p>RHEL コンテンツの作成または更新は、Samples Registry が空の文字列または registry.redhat.io 以外の値に上書きされる場合、プルシークレットの存在によって制御されることはありません。</p> </div> </div>
architectures	アーキテクチャーのタイプを選択するためのプレースホルダー。
skippedImagestreams	Cluster Samples Operator のインベントリーにあるものの、クラスター管理者が Operator に無視させるか、または管理させないようにするイメージストリーム。このパラメーターにイメージストリーム名の一覧を追加できます。例: ["httpd","perl"]
skippedTemplates	Cluster Samples Operator のインベントリーにあるものの、クラスター管理者が Operator に無視させるか、または管理させないようにするテンプレート。

シークレット、イメージストリーム、およびテンプレート監視イベントは、初期サンプルリソースオブジェクトの作成前に追加することができ、Cluster Samples Operator はイベントを検出し、再度キューに入れます。

1.4.1. 設定の制限

Cluster Samples Operator が複数のアーキテクチャーをサポートする際に、アーキテクチャーの一覧は、**Managed** 状態の場合は変更できません。

アーキテクチャーの値を変更するために、クラスター管理者は以下を実行する必要があります。

- **Management State** に **Removed** のマークを付け、変更を保存します。
- その後の変更では、アーキテクチャーを編集し、**Management State** を **Managed** に戻します。

Cluster Samples Operator は **Removed** 状態の場合に依然としてシークレットを処理します。**Removed** に切り替える前にシークレットを作成でき、**Managed** に切り替える前の **Removed** 状態で、または **Managed** 状態に切り替えた後にシークレットを作成できます。**Managed** への切り替え後にシークレットを作成する場合に、シークレットイベントが処理されるまでサンプルの作成に遅延が生じます。これは、レジストリーの変更を容易にするために実行されます。ここでは、クリーンな状態にするために、切り替え前にすべてのサンプルを削除することを選択できます。切り替え前にすべてのサンプルを削除する必要はありません。

1.4.2. 条件

サンプルリソースには以下の条件とそのステータスが適用されます。

条件	説明
SamplesExists	サンプルが openshift namespace に作成されていることを示します。
ImageChangesInProgress	<p>True イメージストリームが作成または更新される場合に、タグ仕様の生成およびタグステータスの生成のすべてが一致する訳ではありません。</p> <p>False すべての生成が一致するか、または修復不可能なエラーがインポート時に発生した場合、最後に表示されるエラーは message フィールドに置かれます。保留中のイメージストリームの一覧は reason フィールドに置かれます。</p>
ConfigurationValid	前述の制限された変更のいずれかが送信されるかどうかに応じて True または False になります。
RemovePending	Management State: Removed 設定が保留中であるものの、Cluster Samples Operator は削除が完了するまで待機していることを示します。
ImportImageErrorsExist	<p>イメージストリームのタグのいずれかについて、イメージストリームでイメージインポートフェーズにエラーがあったことを示すインジケーター。</p> <p>True エラーが発生した場合。エラーのあるイメージストリームの一覧は reason フィールドに置かれます。報告されるそれぞれのエラーの詳細は message フィールドに置かれます。</p>
MigrationInProgress	True: バージョンが現在のサンプルセットのインストールに使用した Samples Operator バージョンと異なることを Samples Operator が検知する場合。

1.5. CLUSTER SAMPLES OPERATOR 設定へのアクセス

Cluster Samples Operator は、提供されるパラメーターでファイルを編集して設定できます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。

手順

- Cluster Samples Operator 設定にアクセスします。

```
$ oc edit configs.samples.operator.openshift.io/cluster -o yaml
```

Cluster Samples Operator 設定は以下の例のようになります。

```
apiVersion: samples.operator.openshift.io/v1
kind: Config
...
```

関連情報

認証情報の詳細は、[イメージプルシークレットの使用](#) を参照してください。

第2章 代替レジストリーでの CLUSTER SAMPLES OPERATOR の使用

最初にミラーレジストリーを作成して、別のレジストリーで Cluster Samples Operator を使用できません。



重要

必要なコンテナイメージを取得するには、インターネットへのアクセスが必要です。この手順では、ご使用のネットワークとインターネットのどちらにもアクセスできるミラーホストにミラーレジストリーを配置します。

2.1. ミラーレジストリーについて

OpenShift Container Platform インストールおよびミラーレジストリーに対する後続の製品の更新に必要なイメージをミラーリングできます。これらのアクションは同じプロセスを使用します。コンテンツの説明が含まれるリリースイメージ、およびこれが参照するイメージがすべてミラーリングされます。さらに、Operator カタログソースイメージおよびこれが参照するイメージは、使用する Operator ごとにミラーリングする必要があります。コンテンツをミラーリングした後に、各クラスターをミラーレジストリーからこのコンテンツを取得するように設定します。

ミラーレジストリーには、[Docker v2-2](#) をサポートする任意のコンテナレジストリーを使用できます。すべての主要なクラウドプロバイダーレジストリー、および Red Hat Quay、Artifactory その他には、必要なサポートがあります。これらのレジストリーの1つを使用すると、OpenShift Container Platform で非接続環境で各イメージの整合性を検証できるようになります。

ミラーレジストリーは、プロビジョニングするクラスター内のすべてのマシンから到達できる必要があります。レジストリーに到達できない場合、インストール、更新、またはワークロードの再配置などの通常の操作が失敗する可能性があります。そのため、ミラーレジストリーは可用性の高い方法で実行し、ミラーレジストリーは少なくとも OpenShift Container Platform クラスターの実稼働環境の可用性の条件に一致している必要があります。

ミラーレジストリーを OpenShift Container Platform イメージで設定する場合、2つのシナリオを実行することができます。インターネットとミラーレジストリーの両方にアクセスできるホストがあり、クラスターノードにアクセスできない場合は、そのマシンからコンテンツを直接ミラーリングできます。このプロセスは、**connected mirroring** (接続ミラーリング) と呼ばれます。このようなホストがない場合は、イメージをファイルシステムにミラーリングしてから、そのホストまたはリムーバブルメディアを制限された環境に配置する必要があります。このプロセスは、**disconnected mirroring** (非接続ミラーリング) と呼ばれます。

2.1.1. ミラーホストの準備

ミラーレジストリーを作成する前に、ミラーホストを準備する必要があります。

2.1.2. バイナリーのダウンロードによる CLI のインストール

コマンドラインインターフェイスを使用して OpenShift Container Platform と対話するために CLI (**oc**) をインストールすることができます。**oc** は Linux、Windows、または macOS にインストールできます。



重要

以前のバージョンの **oc** をインストールしている場合、これを使用して OpenShift Container Platform 4.5 のすべてのコマンドを実行することはできません。新規バージョンの **oc** をダウンロードし、インストールします。

2.1.2.1. Linux への CLI のインストール

以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを Linux にインストールできます。

手順

1. Red Hat OpenShift Cluster Manager サイトの [Infrastructure Provider](#) ページに移動します。
2. インフラストラクチャプロバイダーを選択し、(該当する場合は) インストールタイプを選択します。
3. **Command-line interface** セクションで、ドロップダウンメニューの **Linux** を選択し、**Download command-line tools** をクリックします。
4. アーカイブを展開します。

```
$ tar xvzf <file>
```

5. **oc** バイナリーを、**PATH** にあるディレクトリーに配置します。**PATH** を確認するには、以下のコマンドを実行します。

```
$ echo $PATH
```

CLI のインストール後は、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

2.1.2.2. Windows での CLI のインストール

以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを Windows にインストールできます。

手順

1. Red Hat OpenShift Cluster Manager サイトの [Infrastructure Provider](#) ページに移動します。
2. インフラストラクチャプロバイダーを選択し、(該当する場合は) インストールタイプを選択します。
3. **Command-line interface** セクションで、ドロップダウンメニューの **Windows** を選択し、**Download command-line tools** をクリックします。
4. ZIP プログラムでアーカイブを解凍します。
5. **oc** バイナリーを、**PATH** にあるディレクトリーに移動します。**PATH** を確認するには、コマンドプロンプトを開いて以下のコマンドを実行します。

```
C:\> path
```

CLI のインストール後は、**oc** コマンドを使用して利用できます。

```
C:\> oc <command>
```

2.1.2.3. macOS への CLI のインストール

以下の手順を使用して、OpenShift CLI (**oc**) バイナリーを macOS にインストールできます。

手順

1. Red Hat OpenShift Cluster Manager サイトの [Infrastructure Provider](#) ページに移動します。
2. インフラストラクチャプロバイダーを選択し、(該当する場合は) インストールタイプを選択します。
3. **Command-line interface** セクションで、ドロップダウンメニューの **MacOS** を選択し、**Download command-line tools** をクリックします。
4. アーカイブを展開し、解凍します。
5. **oc** バイナリーをパスにあるディレクトリーに移動します。
PATH を確認するには、ターミナルを開き、以下のコマンドを実行します。

```
$ echo $PATH
```

CLI のインストール後は、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

2.2. イメージのミラーリングを可能にする認証情報の設定

Red Hat からミラーへのイメージのミラーリングを可能にするコンテナイメージレジストリーの認証情報ファイルを作成します。

前提条件

- ネットワークが制限された環境で使用するミラーレジストリーを設定していること。

手順

インストールホストで以下の手順を実行します。

1. Red Hat OpenShift Cluster Manager サイトの [Pull Secret](#) ページから **registry.redhat.io** プルシークレットをダウンロードし、これを **.json** ファイルに保存します。
2. ミラーレジストリーの base64 でエンコードされたユーザー名およびパスワードまたはトークンを生成します。

```
$ echo -n '<user_name>:<password>' | base64 -w0 ①  
BGVtbYk3ZHAtqXs=
```

- ① **<user_name>** および **<password>** については、レジストリーに設定したユーザー名およびパスワードを指定します。

3. JSON 形式でプルシークレットのコピーを作成します。

```
$ cat ./pull-secret.text | jq . > <path>/<pull-secret-file> ❶
```

- ❶ プルシークレットを保存するフォルダーへのパスおよび作成する JSON ファイルの名前を指定します。

ファイルの内容は以下の例のようになります。

```
{
  "auths": {
    "cloud.openshift.com": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "quay.io": {
      "auth": "b3BlbnNo...",
      "email": "you@example.com"
    },
    "registry.connect.redhat.com": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

4. 新規ファイルを編集し、レジストリーについて記述するセクションをこれに追加します。

```
"auths": {
  "<mirror_registry>": { ❶
    "auth": "<credentials>", ❷
    "email": "you@example.com"
  },
}
```

- ❶ **<mirror_registry>** については、レジストリードメイン名と、ミラーレジストリーがコンテンツを提供するために使用するポートをオプションで指定します。例:
registry.example.com または **registry.example.com:5000**
- ❷ **<credentials>** については、ミラーレジストリーの base64 でエンコードされたユーザー名およびパスワードを指定します。

ファイルは以下の例のようになります。

```
{
  "auths": {
    "<mirror_registry>": {
      "auth": "<credentials>",
      "email": "you@example.com"
    },
  }
}
```

```

"cloud.openshift.com": {
  "auth": "b3BlbnNo...",
  "email": "you@example.com"
},
"quay.io": {
  "auth": "b3BlbnNo...",
  "email": "you@example.com"
},
"registry.connect.redhat.com": {
  "auth": "NTE3Njg5Nj...",
  "email": "you@example.com"
},
"registry.redhat.io": {
  "auth": "NTE3Njg5Nj...",
  "email": "you@example.com"
}
}
}
}

```

2.3. OPENSIFT CONTAINER PLATFORM イメージリポジトリーのミラーリング

クラスターのインストールまたはアップグレード時に使用するために、OpenShift Container Platform イメージリポジトリーをお使いのレジストリーにミラーリングします。

前提条件

- ミラーホストがインターネットにアクセスできる。
- ネットワークが制限された環境で使用するミラーレジストリーを設定し、設定した証明書および認証情報にアクセスできる。
- Red Hat OpenShift Cluster Manager のサイトの [Pull Secret](#) ページからプルシークレットをダウンロードしており、ミラーリポジトリーに認証を組み込むようにこれを変更している。
- Subject Alternative Name が設定されていない自己署名証明書を使用する場合は、この手順の **oc** コマンドの前に **GODEBUG=x509ignoreCN=0** を追加する必要があります。この変数を設定しない場合、**oc** コマンドは以下のエラーを出して失敗します。

```
x509: certificate relies on legacy Common Name field, use SANs or temporarily enable
Common Name matching with GODEBUG=x509ignoreCN=0
```

手順

ミラーホストで以下の手順を実行します。

1. [OpenShift Container Platform ダウンロード](#) ページを確認し、インストールする必要のある OpenShift Container Platform のバージョンを判別し、[Repository Tags](#) ページで対応するタグを判別します。
2. 必要な環境変数を設定します。
 - a. リリースバージョンをエクスポートします。

```
$ OCP_RELEASE=<release_version>
```

<release_version> について、インストールする OpenShift Container Platform のバージョンに対応するタグを指定します (例: 4.5.4)。

- b. ローカルレジストリー名とポートをエクスポートします。

```
$ LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>'
```

<local_registry_host_name> については、ミラーレジストリーのレジストリードメイン名を指定し、<local_registry_host_port> については、コンテンツの送信に使用するポートを指定します。

- c. ローカルリポジトリー名をエクスポートします。

```
$ LOCAL_REPOSITORY='<local_repository_name>'
```

<local_repository_name> については、**ocp4/openshift4** などのレジストリーに作成するリポジトリーの名前を指定します。

- d. ミラーリングするリポジトリーの名前をエクスポートします。

```
$ PRODUCT_REPO='openshift-release-dev'
```

実稼働環境のリリースの場合には、**openshift-release-dev** を指定する必要があります。

- e. パスをレジストリープルシークレットにエクスポートします。

```
$ LOCAL_SECRET_JSON='<path_to_pull_secret>'
```

<path_to_pull_secret> については、作成したミラーレジストリーのプルシークレットの絶対パスおよびファイル名を指定します。

- f. リリースミラーをエクスポートします。

```
$ RELEASE_NAME="ocp-release"
```

実稼働環境のリリースについては、**ocp-release** を指定する必要があります。

- g. サーバーのアーキテクチャーのタイプをエクスポートします (例: **x86_64**)。

```
$ ARCHITECTURE=<server_architecture>
```

- h. ミラーリングされたイメージをホストするためにディレクトリーへのパスをエクスポートします。

```
$ REMOVABLE_MEDIA_PATH=<path> 1
```

1 最初のスラッシュ (/) 文字を含む完全パスを指定します。

3. バージョンイメージを内部コンテナレジストリーにミラーリングします。

- ミラーホストがインターネットにアクセスできない場合は、以下の操作を実行します。
 - i. リムーバブルメディアをインターネットに接続しているシステムに接続します。

- ii. ミラーリングするイメージおよび設定マニフェストを確認します。

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON} \
  --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
  ${ARCHITECTURE} \
  --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
  --to-release-
  image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
  ${ARCHITECTURE} --dry-run
```

- iii. 直前のコマンドの出力の **imageContentSources** セクション全体を記録します。ミラーの情報はミラーリングされたリポジトリに一意であり、インストール時に **imageContentSources** セクションを **install-config.yaml** ファイルに追加する必要があります。
- iv. イメージをリムーバブルメディア上のディレクトリにミラーリングします。

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON} --to-
  dir=${REMOVABLE_MEDIA_PATH}/mirror
  quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
  ${ARCHITECTURE}
```

- v. メディアをネットワークが制限された環境に移し、イメージをローカルコンテナレジストリーにアップロードします。

```
$ oc image mirror -a ${LOCAL_SECRET_JSON} --from-
  dir=${REMOVABLE_MEDIA_PATH}/mirror
  "file://openshift/release:${OCP_RELEASE}*"
  ${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} ❶
```

- ❶ **REMOVABLE_MEDIA_PATH** の場合、イメージのミラーリング時に指定した同じパスを使用する必要があります。

- ローカルコンテナレジストリーがミラーホストに接続されている場合は、以下の操作を実行します。
 - i. 以下のコマンドを使用して、リリースイメージをローカルレジストリーに直接プッシュします。

```
$ oc adm release mirror -a ${LOCAL_SECRET_JSON} \
  --from=quay.io/${PRODUCT_REPO}/${RELEASE_NAME}:${OCP_RELEASE}-
  ${ARCHITECTURE} \
  --to=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY} \
  --to-release-
  image=${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
  ${ARCHITECTURE}
```

このコマンドは、リリース情報をダイジェストとしてプルします。その出力には、クラスタのインストール時に必要な **imageContentSources** データが含まれます。

- ii. 直前のコマンドの出力の **imageContentSources** セクション全体を記録します。ミラーの情報はミラーリングされたリポジトリに一意であり、インストール時に **imageContentSources** セクションを **install-config.yaml** ファイルに追加する必要があります。



注記

ミラーリングプロセス中にイメージ名に Quay.io のパッチが適用され、podman イメージにはブートストラップ仮想マシンのレジストリーに Quay.io が表示されます。

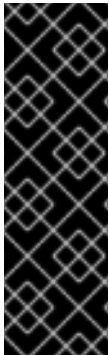
4. ミラーリングしたコンテンツをベースとしているインストールプログラムを作成するには、これを展開し、リリースに固定します。

- ミラーホストがインターネットにアクセスできない場合は、以下のコマンドを実行します。

```
$ oc adm release extract -a ${LOCAL_SECRET_JSON} --command=openshift-install
"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}"
```

- ローカルコンテナーレジストリーがミラーホストに接続されている場合は、以下のコマンドを実行します。

```
$ oc adm release extract -a ${LOCAL_SECRET_JSON} --command=openshift-install
"${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
${ARCHITECTURE}"
```



重要

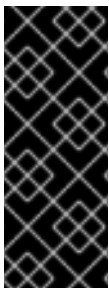
選択した OpenShift Container Platform バージョンに適したイメージを使用するには、ミラーリングされたコンテンツからインストールプログラムを展開する必要があります。

インターネット接続のあるマシンで、このステップを実行する必要があります。

非接続環境を使用している場合には、`must-gather` の一部として `--image` フラグを使用し、ペイロードイメージを参照します。

2.4. 代替のレジストリーまたはミラーリングされたレジストリーでの CLUSTER SAMPLES OPERATOR イメージストリームの使用

Cluster Samples Operator によって管理される `openshift` namespace のほとんどのイメージストリームは、Red Hat レジストリーの registry.redhat.io にあるイメージを参照します。



重要

`jenkins`、`jenkins-agent-maven`、および `jenkins-agent-nodejs` イメージストリームは、インストールペイロードからのもので、Samples Operator によって管理されます。

Sample Operator 設定ファイルの `samplesRegistry` フィールドの registry.redhat.io への設定は、これはすでに Jenkins イメージおよびイメージストリーム以外のすべての registry.redhat.io に送信されているため不要になります。



注記

`cli`、`installer`、`must-gather`、および `tests` イメージストリームはインストールペイロードの一部ですが、Cluster Samples Operator によって管理されません。これらについては、この手順で扱いません。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- ミラーレジストリーのプルシークレットの作成。

手順

1. ミラーリングする特定のイメージストリームのイメージにアクセスします。

```
$ oc get is <imagestream> -n openshift -o json | jq .spec.tags[].from.name | grep registry.redhat.io
```

2. 必要なイメージストリームに関連付けられた registry.redhat.io のイメージをミラーリングします。

```
$ oc image mirror registry.redhat.io/rhsc/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhsc/ruby-25-rhel7:latest
```

3. クラスターのイメージ設定オブジェクトを作成します。

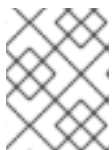
```
$ oc create configmap registry-config --from-file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. クラスターのイメージ設定オブジェクトに、ミラーに必要な信頼される CA を追加します。

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":{"name":"registry-config"}}}' --type=merge
```

5. Cluster Samples Operator 設定オブジェクトの **samplesRegistry** フィールドを、ミラー設定で定義されたミラーの場所の **hostname** の部分を含むように更新します。

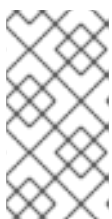
```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```



注記

これは、イメージストリームのインポートプロセスでミラーまたは検索メカニズムが使用されないのが必要になります。

6. Cluster Samples Operator 設定オブジェクトの **skippedImagestreams** フィールドにミラーリングされないイメージストリームを追加します。または、サンプルイメージストリームのいずれもサポートする必要がない場合は、Cluster Samples Operator を Cluster Samples Operator 設定オブジェクトの **Removed** に設定します。



注記

省略されないミラーリングされないイメージがあるか、または Samples Operator が **Removed** に変更されない場合、Samples Operator はイメージストリームのインポートが失敗し始めてから 2 時間後に **Degraded** ステータスを報告します。

openshift namespace のテンプレートの多くはイメージストリームを参照します。そのた

め、**Removed** を使用してイメージストリームとテンプレートの両方を除去すると、イメージストリームのいずれかが欠落しているためにテンプレートが正常に機能しない場合にテンプレートの使用を試行する可能性がなくなります。

第3章 コンテナ、イメージおよびイメージストリームについて

コンテナ、イメージ、およびイメージストリームは、コンテナ化されたソフトウェアを作成し、管理する際に理解しておくべき重要な概念です。イメージは、コンテナがコンテナイメージの実行中のインスタンスである場合に、実行の準備ができている一連のソフトウェアを保持します。イメージストリームは、同一の基本的なイメージの異なるバージョンを保存する1つの方法です。それらの異なるバージョンは、同じイメージ名の異なるタグによって表されます。

3.1. イメージ

OpenShift Container Platform のコンテナは OCI または Docker 形式のコンテナのイメージをベースにしています。イメージは、単一コンテナを実行するためのすべての要件、およびそのニーズおよび機能を記述するメタデータを含むバイナリです。

これはパッケージ化テクノロジーとして考えることができます。コンテナには、作成時にコンテナに追加のアクセスを付与しない限り、イメージで定義されるリソースにのみアクセスできます。同じイメージを複数のホストにまたがって複数のコンテナにデプロイし、それらの間で負荷を分散することにより、OpenShift Container Platform はイメージにパッケージ化されたサービスの冗長性および水平的なスケーリングを提供できます。

イメージをビルドするために `podman` または `docker` CLI を直接使用することはできますが、OpenShift Container Platform は、コードまたは設定を既存イメージに追加して新規イメージの作成を支援するビルダーイメージも提供します。

アプリケーションは一定期間をかけて開発されるため、単一のイメージ名が同じイメージの数多くの異なるバージョンを参照する場合があります。それぞれの異なるイメージは、通常は 12 文字 (例: `fd44297e2ddb`) に省略されるそのハッシュ (`fd44297e2ddb050ec4f...` などの長い 16 進数) で一意に参照されます。

3.2. コンテナ

OpenShift Container Platform アプリケーションの基本的な単位は コンテナと呼ばれています。[Linux コンテナテクノロジー](#) は、指定されたリソースのみとの対話に制限されるように、実行中のプロセスを分離する軽量なメカニズムです。このコンテナという用語は、コンテナイメージの実行中または一時停止している特定のインスタンスとして定義されています。

数多くのアプリケーションインスタンスは、相互のプロセス、ファイル、ネットワークなどを可視化せずに単一ホストのコンテナで実行される可能性があります。通常、コンテナは任意のワークロードに使用されますが、各コンテナは Web サーバーまたはデータベースなどの (通常はマイクロサービスと呼ばれることの多い) 単一サービスを提供します。

Linux カーネルは数年にわたりコンテナテクノロジーの各種機能を統合してきました。Docker プロジェクトはホスト上の Linux コンテナの便利な管理インターフェイスを開発しました。さらに最近では、[Open Container Initiative](#) により、コンテナ形式およびコンテナランタイムのオープン標準が策定されています。OpenShift Container Platform および Kubernetes は複数ホストのインストール間で OCI および Docker 形式のコンテナのオーケストレーションを実行する機能を追加しています。

OpenShift Container Platform を使用する際にコンテナランタイムと直接対話することはありませんが、それらの OpenShift Container Platform におけるロールやコンテナ内でのアプリケーションの機能を理解する上で、それらの機能および用語を理解しておくことは重要です。

`podman` などのツールは、コンテナを直接実行し、管理するための `docker` コマンドラインツールを置き換えるために使用できます。`podman` を使用すると、OpenShift Container Platform と切り離してコンテナの実験を行うことができます。

3.3. イメージレジストリー

イメージレジストリーは、コンテナイメージを保管し、提供するコンテナサーバーです。以下に例を示します。

```
registry.redhat.io
```

レジストリーには、1つ以上のタグ付けされたイメージを持つ1つ以上のイメージリポジトリのコレクションが含まれます。Red Hat は、サブスクリプションをお持ちのお客様に対して **registry.redhat.io** でレジストリーを提供しています。また、OpenShift Container Platform はカスタムコンテナイメージを管理するための独自の内部レジストリーも提供しています。

3.4. イメージリポジトリ

イメージリポジトリは、関連するコンテナイメージおよびそれらを特定するタグのコレクションです。たとえば、OpenShift Container Platform Jenkins イメージはリポジトリにあります。

```
docker.io/openshift/jenkins-2-centos7
```

3.5. イメージタグ

イメージタグは、イメージストリーム内の他のイメージから特定のイメージを識別するリポジトリのコンテナイメージに適用されるラベルです。通常、タグはある種のバージョン番号を表します。たとえば、ここでは **:v3.11.59-2** がタグになります。

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

イメージにタグを追加することができます。たとえば、イメージには **:v3.11.59-2** および **:latest** というタグが割り当てられる可能性があります。

OpenShift Container Platform は **docker tag** コマンドに似た **oc tag** コマンドを提供しますが、これらはイメージ上で直接動作するのではなくイメージストリームで動作します。

3.6. イメージ ID

イメージ ID は、イメージをプルするために使用できる SHA (Secure Hash Algorithm) コードです。SHA イメージ ID は変更できません。特定の SHA ID は同一のコンテナイメージコンテンツを常に参照します。以下に例を示します。

```
docker.io/openshift/jenkins-2-centos7@sha256:ab312bda324
```

3.7. イメージストリームの使用

イメージストリームおよびその関連付けられたタグは、OpenShift Container Platform 内でコンテナイメージを参照するための抽象化を提供します。イメージストリームとそのタグを使用して、利用可能なイメージを確認し、リポジトリのイメージが変更される場合でも必要な特定のイメージを使用していることを確認できます。

イメージストリームには実際のイメージデータは含まれませんが、イメージリポジトリと同様に、関連するイメージの単一の仮想ビューが提示されます。

ビルドおよびデプロイメントをそれぞれ実行し、ビルドおよびデプロイメントを、新規イメージが追加される際やこれに対応する際の通知をイメージストリームで確認できるように設定できます。

たとえば、デプロイメントで特定のイメージを使用していて、そのイメージの新規バージョンが作成される場合、デプロイメントを、そのイメージの新規バージョンを選択できるように自動的に実行します。

デプロイメントまたはビルドで使用するイメージストリームタグが更新されない場合には、コンテナイメージレジストリーのコンテナイメージが更新されても、ビルドまたはデプロイメントは以前の、既知でおそらく適切であると予想されるイメージをそのまま使用します。

ソースイメージは以下のいずれかに保存できます。

- OpenShift Container Platform の統合レジストリー
- registry.redhat.io or Quay.io などの外部レジストリー
- OpenShift Container Platform クラスターの他のイメージストリーム

ビルドまたはデプロイメント設定などのイメージストリームタグを参照するオブジェクトを定義する場合には、リポジトリではなく、イメージストリームタグを参照します。アプリケーションのビルドまたはデプロイ時に、OpenShift Container Platform がこのイメージストリームタグを使用してリポジトリに対してクエリーし、対象のイメージに関連付けられた ID を特定して、そのイメージを使用します。

イメージストリームメタデータは他のクラスター情報と共に etcd インスタンスに保存されます。

イメージストリームの使用には、いくつかの大きな利点があります。

- コマンドラインを使用して再プッシュすることなく、タグ付けや、タグのロールバック、およびイメージの迅速な処理を実行できます。
- 新規イメージがレジストリーにプッシュされると、ビルドおよびデプロイメントをトリガーできます。また、OpenShift Container Platform には他のリソースの汎用トリガーがあります (Kubernetes オブジェクトなど)。
- 定期的な再インポートを実行するためにタグにマークを付けることができます。ソースイメージが変更されると、その変更は選択され、イメージストリームに反映されます。これにより、ビルドまたはデプロイメント設定に応じてビルドまたはデプロイメントフローがトリガーされます。
- 詳細なアクセス制御を使用してイメージを共有し、チーム間でイメージを迅速に分散できます。
- ソースイメージが変更されると、イメージストリームタグはイメージの既知の適切なバージョンをポイントしたままになり、アプリケーションが予期せずには損傷しないようにします。
- イメージストリームオブジェクトのパーミッションを使用して、イメージを表示し、使用できるユーザーについてセキュリティを設定することができます。
- クラスターレベルでイメージを読み込んだり、一覧表示するパーミッションのないユーザーは、イメージストリームを使用してプロジェクトでタグ付けされたイメージを取得できます。

3.8. イメージストリームタグ

イメージストリームタグは、イメージストリームのイメージに対する名前付きポインターです。イメージストリームタグはコンテナイメージタグに似ています。

3.9. イメージストリームイメージ

イメージストリームイメージは、これがタグ付けされている特定のイメージストリームから特定のコンテナイメージを取得できるようにします。イメージストリームイメージは、特定のイメージの SHA ID についてのメタデータをプルする API リソースオブジェクトです。

3.10. イメージストリームトリガー

イメージストリームのトリガーは、イメージストリームタグの変更時に特定のアクションを生じさせます。たとえば、インポートにより、タグの値が変更され、これによりデプロイメント、ビルドまたはそれらをリッスンする他のリソースがある場合にトリガーが実行されます。

3.11. 追加リソース

- イメージストリームの使用方法についての詳細は、[イメージストリームの管理](#) を参照してください。

第4章 イメージの作成

使用可能な事前にビルドされたイメージを使用して独自のコンテナイメージを作成する方法について確認します。このプロセスには、イメージの作成、イメージのメタデータの定義、イメージのテストおよびカスタムビルダーワークフローを使用した OpenShift Container Platform で使用するイメージの作成のベストプラクティスを理解することが含まれます。イメージを作成した後は、これを内部レジストリーにプッシュできます。

4.1. コンテナのベストプラクティスについて

OpenShift Container Platform で実行するコンテナイメージを作成する場合には、イメージの作成者は、イメージの使いやすさの点で数多くのベストプラクティスを考慮する必要があります。イメージは変更不可で、そのままの状態で使用されることが意図されているため、以下のガイドラインは、イメージを使用しやすく、OpenShift Container Platform で簡単に使用できるようにするのに役立ちます。

4.1.1. コンテナイメージの一般的なガイドライン

以下のガイドラインは、イメージが OpenShift Container Platform で使用されるかどうかにかかわらず、コンテナイメージの作成時に一般的に適用されます。

イメージの再利用

可能な場合は、**FROM** ステートメントを使用し、適切なアップストリームイメージをベースとしてイメージを設定します。これにより、依存関係を直接更新する必要なく、イメージが更新時にアップストリームイメージからセキュリティー修正を簡単に取得できるようになります。

さらに、**FROM** 命令 (例: `rhel:rhel7`) のタグを使用して、お使いのイメージがどのバージョンのイメージをベースとしているかを明確にします。アップストリームイメージの **latest** バージョンを使用すると互換性に影響のある変更が組み込まれる可能性があるため、**latest** 以外のタグを使用することができます。

タグ内の互換性の維持

独自のイメージにタグを付ける場合には、タグ内で後方互換性が維持されるようにします。たとえば、**foo** という名前のイメージがあり、現時点でバージョン **1.0** が含まれている場合には、**foo:v1** のタグを指定します。イメージの更新時には、元のイメージとの互換性がある限り、新しいイメージに **foo:v1** のタグを付けることができ、このタグのダウンストリームのコンシューマーは、互換性に関する影響を被ることなく更新を取得できるようになります。

互換性のない更新を後にリリースした場合には、**foo:v2** などの新しいタグに切り替えます。これにより、ダウンストリームのコンシューマーはいつでも新しいバージョンに移行できますが、意図せずにこの互換性のない新規イメージによる影響を受けることはありません。**foo:latest** を使用するダウンストリームコンシューマーには、互換性のない変更が導入されるリスクがあります。

複数プロセスの回避

データベースや **SSHD** など複数のサービスを1つのコンテナ内で起動しないようにしてください。コンテナは軽量で、複数のプロセスをオーケストレーションするために簡単にリンクできるので、複数プロセスの実行は不要です。OpenShift Container Platform では、関連のあるイメージを1つの Pod にグループ化して、簡単に共存させ、共同管理することができます。

このように共存させることで、コンテナはネットワークの namespace とストレージを通信用に共有できるようになります。また、イメージの更新頻度が低く、個別に更新されるので、更新による中断の可能性が低くなります。シグナル処理フローは、複数の起動したプロセスへのルーティングシグナルを管理する必要がないので、単一プロセスによって明確になります。

ラッパースクリプトでの **exec** の使用

多くのイメージはラッパースクリプトを使用して、実行されるソフトウェアのプロセスを開始する前に

いくつかの設定を行います。イメージがこのようなスクリプトを使用する場合、そのスクリプトは、スクリプトのプロセスがソフトウェアによって置き換えられるように **exec** を使用します。**exec** を使用しない場合、コンテナランタイムによって送信されるシグナルが、ソフトウェアのプロセスではなくラッパースクリプトに送られます。これは望ましい動作ではありません。

一部のサーバーのプロセスを開始するラッパースクリプトがあるとします。**podman run -i** などを使用してコンテナを起動すると、それによりラッパースクリプトが実行され、次にプロセスが開始されます。**CTRL+C** でコンテナを閉じる必要があるとします。ラッパースクリプトがサーバープロセスを開始するために **exec** を使用している場合、**podman** は SIGINT をサーバープロセスに送信し、すべてが予想通りに機能します。ラッパースクリプトで **exec** を使用しなかった場合、**podman** はラッパースクリプトのプロセスに SIGINT を送信し、プロセスは何も生じなかったかのように実行し続けます。

また、コンテナ内で実行されると、プロセスは **PID 1** として実行される点に留意してください。つまり、主なプロセスが中断された場合には、コンテナ全体が停止され、**PID 1** プロセスから起動した子プロセスが終了します。

一時ファイルの消去

ビルドプロセスで作成される一時ファイルはすべて削除します。これには、**ADD** コマンドで追加したファイルも含まれます。たとえば、**yum install** の操作を実行してから、**yum clean** コマンドを実行します。

yum キャッシュがイメージレイヤーに残らないように、以下のように **RUN** ステートメントを作成します。

```
RUN yum -y install mypackage && yum -y install myotherpackage && yum clean all -y
```

以下のように記述した場合には注意してください。

```
RUN yum -y install mypackage  
RUN yum -y install myotherpackage && yum clean all -y
```

上記のように記述すると、最初の **yum** 呼び出しにより、対象のレイヤーに追加のファイルが残ります。**yum clean** 操作を後に実行してもこれらのファイルは削除できません。これらの追加ファイルは最終イメージでは確認できませんが、下位レイヤーには存在します。

現在のコンテナビルドプロセスでは、前のレイヤーで何かが削除された場合でも、後のレイヤーでコマンドを実行してイメージが使用する容量を縮小できません。ただし、これについては今後変更される可能性はあります。後のレイヤーでファイルが表示されていなくても **rm** コマンドを実行したとしても、ダウンロードするイメージの全体のサイズを縮小することになりません。そのため、**yum clean** の場合のように、可能な場合は後にレイヤーに書き込まれないように、ファイルの作成に使用したのと同じコマンドでファイルを削除することが最も適切と言えます。

また、単一の **RUN** ステートメントで複数のコマンドを実行すると、イメージのレイヤー数が減り、ダウンロードと実行時間が短縮されます。

正しい順序での命令の指定

コンテナビルダーは **Dockerfile** を読み取り、トップダウンで命令を実行します。命令が正常に実行されると、同じイメージが次回ビルドされる時や、別のイメージがビルドされる時に再利用することができるレイヤーが作成されます。**Dockerfile** の上部にほとんど変更されない命令を配置することは非常に重要です。こうすることで、上位レイヤーで加えられた変更によってキャッシュが無効にならないので、同じイメージの次のビルドをすばやく実行できます。

たとえば、反復するファイルをインストールするための **ADD** コマンドと、パッケージを **yum install** する **RUN** コマンドが含まれる **Dockerfile** で作業を行う場合には、**ADD** コマンドを最後に配置することが最善の方法です。

```
FROM foo
RUN yum -y install mypackage && yum clean all -y
ADD myfile /test/myfile
```

これにより、**myfile** を編集して **podman build** または **docker build** を返すたびに、システムは **yum** コマンドのキャッシュされたレイヤーを再利用し、**ADD** 操作に対してのみ新規レイヤーを生成します。

代わりに **Dockerfile** を以下のように作成した場合:

```
FROM foo
ADD myfile /test/myfile
RUN yum -y install mypackage && yum clean all -y
```

myfile を変更して、**podman build** または **docker build** を再実行するたびに、**ADD** 操作は **RUN** レイヤーのキャッシュを無効にするので、**yum** 操作も再実行する必要があります。

重要なポートのマーク付け

EXPOSE 命令は、ホストシステムで利用できるコンテナおよび他のコンテナにポートを作成します。ポートを **podman run** の起動で公開されるように指定できますが、**Dockerfile** で EXPOSE 命令を使用すると、ソフトウェアが実行する必要のあるポートを明示的に宣言することで、人間とソフトウェアの両方がイメージをより簡単に使用できるようになります。

- 公開されるポートは、イメージから作成されるコンテナに関連付けられる **podman ps** の下に表示されます。
- 公開されるポートは、**podman inspect** によって返されるイメージのメタデータに表示されません。
- 公開されるポートは、1つのコンテナを別のコンテナにリンクする際にリンクされます。

環境変数の設定

ENV 命令で環境変数を設定することが適切です。一例として、プロジェクトのバージョンを設定するなどが挙げられます。バージョンを設定することで、**Dockerfile** を確認せずにバージョンを簡単に見つけ出すことができます。別の例としては、**JAVA_HOME** など、別のプロセスで使用可能なシステムでパスを公開する場合などです。

デフォルトのパスワードの回避

デフォルトのパスワードは設定しないようにしてください。イメージを拡張して、デフォルトのパスワードを削除または変更するのを忘れることが多くあります。これは、実稼働環境で使用するユーザーに誰でも知っているパスワードが割り当てられると、セキュリティの問題に発展する可能性があります。パスワードは、環境変数を使用して設定できます。

デフォルトのパスワードを設定することにした場合には、コンテナの起動時に適切な警告メッセージが表示されるようにしてください。メッセージはデフォルトパスワードの値をユーザーに通知し、環境変数の設定など、パスワードの変更方法を説明するものである必要があります。

SSHD の回避

イメージで **sshd** を実行しないようにしてください。ローカルホストで実行中のコンテナにアクセスするには、**podman exec** または **docker exec** コマンドを使用できます。または、**oc exec** コマンドまたは **oc rsh** コマンドを使用して、OpenShift Container Platform クラスタで実行中のコンテナにアクセスできます。イメージで **sshd** をインストールし、実行すると、攻撃の経路が増え、セキュリティ修正が必要になります。

永続データ向けのボリュームの使用

イメージは、永続データ用に **ボリューム** を使用する必要があります。こうすることで、OpenShift

Container Platform により、コンテナを実行するノードにネットワークストレージがマウントされ、コンテナが新しいノードに移動した場合に、ストレージはそのノードに再度割り当てられます。永続ストレージのすべての要件に対応するようにボリュームを使用することで、コンテナが再起動されたり、移動されたりしても、コンテンツは保存されます。イメージがコンテナ内の任意の場所にデータを書き込む場合には、コンテンツは保存されない可能性があります。

コンテナが破棄された後も保存する必要があるデータはすべて、ボリュームに書き込む必要があります。コンテナエンジンはコンテナの **readonly** フラグをサポートしており、このフラグを使用して、コンテナの一時ストレージにデータが決して記述されないようにすることができます。イメージをこの機能に基づいて設計すると、この機能を後に利用することがより簡単になります。

Dockerfile でボリュームを明示的に定義すると、イメージの利用者がイメージの実行時に定義する必要のあるボリュームがどれかを簡単に理解できるようになります。

OpenShift Container Platform でのボリュームの使用方法についての詳細は、[Kubernetes ドキュメント](#) を参照してください。



注記

永続ボリュームでも、イメージの各インスタンスには独自のボリュームがあり、ファイルシステムはインスタンス間で共有されません。つまり、ボリュームを使用してクラスターの状態を共有できません。

4.1.2. OpenShift Container Platform 固有のガイドライン

以下は、OpenShift Container Platform で使用するためのコンテナイメージの作成時に適用されるガイドラインです。

Source-To-Image (S2I) 向けのイメージの有効化

開発者が提供した Ruby コードを実行するように設計された Ruby イメージなど、サードパーティー提供のアプリケーションコードを実行することが目的のイメージの場合には、イメージを [Source-to-Image \(S2I\)](#) ビルドツールと連携できるようにすることができます。S2I は、インプットとして、アプリケーションのソースコードを受け入れるイメージを簡単に記述でき、アセンブルされたアプリケーションをアウトプットとして実行する新規イメージを簡単に生成することができるフレームワークです。

任意のユーザー ID のサポート

デフォルトでは OpenShift Container Platform は、任意に割り当てられたユーザー ID を使用してコンテナを実行します。こうすることで、コンテナエンジンの脆弱性が原因でコンテナから出ていくプロセスに対して追加のセキュリティーを設定でき、ホストノードでパーミッションのエスカレーションが可能になります。

イメージが任意ユーザーとしての実行をサポートできるように、イメージ内のプロセスで記述されるディレクトリーやファイルは、root グループが所有し、このグループに対して読み取り/書き込みの権限を割り当てる必要があります。実行予定のファイルには、グループの実行権限も必要です。

以下を Dockerfile に追加すると、root グループのユーザーがビルドされたイメージでアクセスできるように、ディレクトリーおよびファイルのパーミッションが設定されます。

```
RUN chgrp -R 0 /some/directory && \
    chmod -R g=u /some/directory
```

コンテナユーザーは常に root グループのメンバーであるため、コンテナユーザーはこれらのファイルに対する読み取り、書き込みが可能です。



警告

コンテナの慎重に扱うべき分野のディレクトリーおよびファイルパーミッションを変更する場合には注意が必要です (通常のシステムの扱いと同様です)。

`/etc/passwd` などの慎重に扱うべき分野に適用されると、意図しないユーザーによるこのようなファイルの変更が可能となり、コンテナやホストにセキュリティ上のリスクが生じる可能性があります。CRI-O は、ランダムユーザー ID のコンテナの `/etc/passwd` への挿入をサポートするため、そのパーミッションを変更する必要はありません。

さらに、コンテナで実行中のプロセスは、特権のあるユーザーとして実行されていないので、特権のあるポート (1024 未満のポート) をリッスンできません。



重要

S2I イメージに、ユーザーを数値で指定した **USER** 宣言が含まれない場合には、デフォルトで、ビルドが失敗します。名前が指定されたユーザーや `root (0)` ユーザーを使用するイメージを OpenShift Container Platform でビルドできるようにするには、プロジェクトのビルダーサービスアカウント (`system:serviceaccount:<your-project>:builder`) を **privileged** SCC (security context constraint) に追加できます。または、すべてのイメージをどのユーザーでも実行できるようにできます。

イメージ間通信でのサービスの使用

データの保存や取得のためにデータベースイメージにアクセスする必要のある Web フロントエンドイメージなど、別のイメージが提供するサービスとイメージが通信する場合には、イメージは OpenShift Container Platform サービスを使用します。サービスは、コンテナが停止、開始、または移動しても変更されない静的アクセスエンドポイントを提供します。さらに、サービスにより、要求が負荷分散されます。

共通のライブラリーの提供

サードパーティーが提供するアプリケーションコードの実行を目的とするイメージの場合は、プラットフォーム用として共通に使用されるライブラリーをイメージに含めるようにしてください。とくに、プラットフォームで使用する共通のデータベース用のデータベースドライバーを設定してください。たとえば、Java フレームワークイメージを作成する場合に、MySQL や PostgreSQL には JDBC ドライバーを設定します。このように設定することで、アプリケーションのアセンブリー時に共通の依存関係をダウンロードする必要がなくなり、アプリケーションイメージのビルドがスピードアップします。また、すべての依存関係の要件を満たすためのアプリケーション開発者の作業が簡素化されます。

設定での環境変数の使用

イメージのユーザーは、ダウンストリームイメージをイメージに基づいて作成しなくても、イメージを設定できます。つまり、ランタイム設定は環境変数を使用して処理されます。単純な設定の場合、実行中のプロセスは環境変数を直接使用できます。より複雑な設定や、これをサポートしないランタイムの場合、起動時に処理されるテンプレート設定ファイルを定義してランタイムを設定します。このプロセス時に、環境変数を使用して渡される値は設定ファイルで置き換えることも、この値を使用して、設定ファイルに指定するオプションを決定することもできます。

環境変数を使用して、コンテナに証明書やキーなどのシークレットを渡すこともでき、これは推奨されています。環境変数を使用することで、シークレット値がイメージにコミットされたり、コンテナイメージレジストリーに漏洩されることはありません。

環境変数を指定することで、イメージの利用者は、イメージ上に新しいレイヤーを作成することなく、データベースの設定、パスワード、パフォーマンスチューニングなどの動作をカスタマイズできます。Pod の定義時に環境変数の値を定義するだけで、イメージの再ビルドなしに設定を変更できます。

非常に複雑なシナリオの場合、ランタイム時にコンテナにマウントされるボリュームを使用して設定を指定することも可能です。ただし、この方法を使用する場合には、必要なボリュームや設定が存在しない場合に明確なエラーメッセージが起動時に表示されるように、イメージが設定されている必要があります。

サービスエンドポイントの情報を渡す環境変数としてデータソースなどの設定を定義される点で、これはイメージ間の通信でのサービスの使用についてのトピックと関連しています。これにより、アプリケーションは、アプリケーションイメージを変更せずに、OpenShift Container Platform 環境に定義されているデータソースサービスを動的に使用できます。

さらに、コンテナの **cgroups** 設定を確認して、調整します。これにより、イメージは利用可能なメモリー、CPU、他のリソースに合わせてチューニングが可能になります。たとえば、Java ベースのイメージは、制限を超えず、メモリー不足のエラーが表示されないように、**cgroup** の最大メモリーパラメーターを基にヒープをチューニングします。

イメージのメタデータ設定

イメージのメタデータを定義することで、OpenShift Container Platform によるコンテナイメージの使用が改善され、開発者が OpenShift Container Platform でイメージを使用しやすくなります。たとえば、メタデータを追加して、イメージに関する役立つ情報を提供したり、必要とされる他のイメージを提案したりできます。

クラスターリング

イメージの複数のインスタンスを実行するとはどういうことかを十分に理解しておく必要があります。最も単純な例では、サービスの負荷分散機能は、イメージのすべてのインスタンスにトラフィックをルーティングします。ただし、セッションの複製などで、リーダーの選択やフェイルオーバーの状態を実行するには、多くのフレームワークが情報を共有する必要があります。

OpenShift Container Platform での実行時に、インスタンスでこのような通信を実現する方法を検討します。Pod 同士は直接通信できますが、Pod が起動、停止、移動するたびに IP アドレスが変更されます。そのため、クラスターリングスキームを動的にしておくことが重要です。

ロギング

すべてのロギングを標準出力に送信することが推奨されます。OpenShift Container Platform はコンテナから標準出力を収集し、表示が可能な中央ロギングサービスに送信します。ログコンテンツを分離する必要がある場合には、出力の接頭辞に適切なキーワードを指定して、メッセージをフィルターリングできるようにしてください。

お使いのイメージがファイルにロギングをする場合には、手動で実行中のコンテナに入り、ログファイルを取得または表示する必要があります。

Liveness および Readiness プローブ

イメージで使用可能な liveness および readiness プローブの例をまとめます。これらのプローブにより、処理の準備ができるまでトラフィックがコンテナにルーティングされず、プロセスが正常でない状態になる場合にコンテナが再起動されるので、ユーザーはイメージを安全にデプロイできます。

テンプレート

イメージと共にテンプレートサンプルを提供することも検討してください。テンプレートがあると、ユーザーは、正しく機能する設定を指定してイメージをすばやく簡単にデプロイできるようになります。完全を期すため、テンプレートには、イメージに関連して記述した liveness および readiness プローブを含めるようにしてください。

4.2. イメージへのメタデータの組み込み

イメージのメタデータを定義することで、OpenShift Container Platform によるコンテナイメージの使用が改善され、開発者が OpenShift Container Platform でイメージを使用しやすくなります。たとえば、メタデータを追加して、イメージに関する役立つ情報を提供したり、必要とされる可能性のある他のイメージを提案したりできます。

このトピックでは、現在の一連のユースケースに必要なメタデータのみを定義します。他のメタデータまたはユースケースは、今後追加される可能性があります。

4.2.1. イメージメタデータの定義

Dockerfile で **LABEL** 命令を使用して、イメージのメタデータを定義することができます。ラベルは、イメージやコンテナに割り当てるキーと値のペアである点で環境変数と似ています。ただし、ラベルは、実行中のアプリケーションに表示されず、イメージやコンテナをすばやく検索する場合にも使用できる点で、環境変数とは異なります。

LABEL 命令に関する詳細は、[Docker ドキュメント](#) を参照してください。

通常、ラベル名には namespace が使用されます。namespace は、対象のラベルを選択して使用するプロジェクトを反映するように設定されます。OpenShift Container Platform の場合、namespace は **io.openshift** に、Kubernetes の場合は、namespace は **io.k8s** に設定されます。

形式に関する詳細は、[Docker のカスタムメタデータ](#) に関するドキュメントを参照してください。

表4.1 サポートされるメタデータ

変数	説明
io.openshift.tags	このラベルには、コンマ区切りの文字列値の一覧として表現されているタグの一覧が含まれます。タグを使用して、コンテナイメージを幅広い機能エリアに分類します。タグを使用すると、UI および生成ツールがアプリケーションの作成プロセスで適切なコンテナイメージを提案しやすくなります。 <pre>LABEL io.openshift.tags mongodb,mongodb24,nosql</pre>
io.openshift.wants	コンテナイメージにすでにタグが指定されていない場合に、生成ツールと UI が適切な提案を行うのに使用するタグの一覧を指定します。たとえば、コンテナイメージに mysql と redis が必要で、コンテナイメージに redis タグが指定されていない場合には、UI はこのイメージをデプロイメントに追加するように提案する可能性があります。 <pre>LABEL io.openshift.wants mongodb,redis</pre>
io.k8s.description	このラベルは、コンテナイメージの利用者に、このイメージが提供するサービスや機能に関する詳細情報を提供するのに使用できます。UI は、この説明とコンテナイメージ名を使用して、人間が解読しやすい情報をエンドユーザーに提供します。 <pre>LABEL io.k8s.description The MySQL 5.5 Server with master-slave replication support</pre>

変数	説明
io.openshift.non-scalable	<p>イメージは、この変数を使用して、スケーリングがサポートされていないことを示す場合があります。その後、UI はこれをそのイメージのコンシューマーに通知します。スケーリング不可にした場合は replicas の値を最初に 1 よりも大きい値に設定することはできません。</p> <pre data-bbox="518 389 1058 450">LABEL io.openshift.non-scalable true</pre>
io.openshift.min-memory および io.openshift.min-cpu	<p>このラベルは、コンテナイメージが正しく機能するにはどの程度リソースが必要かを提案します。UI でユーザーに対し、このコンテナイメージをデプロイすると、ユーザークォータを超過する可能性があることを警告する場合があります。この値は、Kubernetes の数量と互換性がある必要があります。</p> <pre data-bbox="518 703 1018 786">LABEL io.openshift.min-memory 8Gi LABEL io.openshift.min-cpu 4</pre>

4.3. SOURCE-TO-IMAGE によるソースコードからのイメージの作成

Source-to-Image (S2I) は、アプリケーションのソースコードを入力として取り、アSEMBルされたアプリケーションを出力として実行する新規イメージを生成するイメージを簡単に作成できるようにするフレームワークです。

再生成可能なコンテナイメージのビルドに S2I を使用する主な利点として、開発者の使い勝手の良さが挙げられます。ビルダーイメージの作成者は、イメージが最適な S2I パフォーマンスを実現できるように、ビルドプロセスと S2I スクリプトの基本的なコンセプト 2 点を理解する必要があります。

4.3.1. Source-to-Image ビルドプロセスについて

ビルドプロセスは、以下の 3 つの要素で設定されており、これら 3 つを組み合わせると最終的なコンテナイメージが作成されます。

- ソース
- Source-to-Image (S2I) スクリプト
- ビルダーイメージ

S2I は、最初の **FROM** 命令として、ビルダーイメージで Dockerfile を生成します。S2I によって生成される Dockerfile は Buildah に渡されます。

4.3.2. Source-to-Image スクリプトの作成方法

Source-to-Image (S2I) スクリプトは、ビルダーイメージ内でスクリプトを実行できる限り、どのプログラム言語でも記述できます。S2I は **assemble/run/save-artifacts** スクリプトを提供する複数のオプションをサポートします。ビルドごとに、これらの場所はすべて、以下の順番にチェックされます。

1. ビルド設定に指定されるスクリプト
2. アプリケーションソースの **.s2i/bin** ディレクトリーにあるスクリプト

3. `io.openshift.s2i.scripts-url` ラベルを含むデフォルトの URL にあるスクリプト

イメージで指定した `io.openshift.s2i.scripts-url` ラベルも、ビルド設定で指定したスクリプトも、以下の形式のいずれかを使用します。

- `image:///path_to_scripts_dir`: S2I スクリプトが配置されているディレクトリーへのイメージ内の絶対パス。
- `file:///path_to_scripts_dir`: S2I スクリプトが配置されているディレクトリーへのホスト上の相対パスまたは絶対パス。
- `http(s)://path_to_scripts_dir`: S2I スクリプトが配置されているディレクトリーの URL。

表4.2 S2I スクリプト

スクリプト	説明
assemble	<p>assemble スクリプトは、ソースからアプリケーションアーティファクトをビルドし、イメージ内の適切なディレクトリーに配置します。このスクリプトが必要です。このスクリプトのワークフローは以下のとおりです。</p> <ol style="list-style-type: none"> 1. オプション: ビルドのアーティファクトを復元します。増分ビルドをサポートする必要がある場合、save-artifacts も定義するようにしてください (オプション)。 2. 任意の場所に、アプリケーションソースを配置します。 3. アプリケーションのアーティファクトをビルドします。 4. 実行に適した場所に、アーティファクトをインストールします。
run	<p>run スクリプトはアプリケーションを実行します。このスクリプトが必要です。</p>
save-artifacts	<p>save-artifacts スクリプトは、次に続くビルドプロセスを加速できるようにすべての依存関係を収集します。このスクリプトはオプションです。以下に例を示します。</p> <ul style="list-style-type: none"> ● Ruby の場合は、Bundler でインストールされる gems ● Java の場合は、.m2 のコンテンツ <p>これらの依存関係は tar ファイルに集められ、標準出力としてストリーミングされます。</p>
usage	<p>usage スクリプトでは、ユーザーに、イメージの正しい使用方法を通知します。このスクリプトはオプションです。</p>

スクリプト	説明
test/run	<p>test/run スクリプトでは、イメージが正しく機能しているかどうかを確認するためのプロセスを作成できます。このスクリプトはオプションです。このプロセスの推奨フローは以下のとおりです。</p> <ol style="list-style-type: none"> 1. イメージをビルドします。 2. イメージを実行して usage スクリプトを検証します。 3. s2i build を実行して assemble スクリプトを検証します。 4. オプション: 再度 s2i build を実行して、save-artifacts と assemble スクリプトの保存、復元アーティファクト機能を検証します。 5. イメージを実行して、テストアプリケーションが機能していることを確認します。 <div style="display: flex; align-items: flex-start; margin-top: 20px;"> <div style="flex: 1; text-align: center; margin-right: 10px;">  </div> <div> <p>注記</p> <p>test/run スクリプトでビルドしたテストアプリケーションを配置するための推奨される場所は、イメージリポジトリの test/test-app ディレクトリーです。</p> </div> </div>

S2I スクリプトの例

以下の S2I スクリプトの例は Bash で記述されています。それぞれの例では、**tar** の内容は **/tmp/s2i** ディレクトリーに展開されることが前提とされています。

assemble スクリプト:

```
#!/bin/bash

# restore build artifacts
if [ "$(ls /tmp/s2i/artifacts/ 2>/dev/null)" ]; then
  mv /tmp/s2i/artifacts/* $HOME/.
fi

# move the application source
mv /tmp/s2i/src $HOME/src

# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run スクリプト:

```
#!/bin/bash
```



```
# run the application
/opt/application/run.sh
```

save-artifacts スクリプト:

```
#!/bin/bash

pushd ${HOME}
if [ -d deps ]; then
  # all deps contents to tar stream
  tar cf - deps
fi
popd
```

usage スクリプト:

```
#!/bin/bash

# inform the user how to use the image
cat <<EOF
This is a S2I sample builder image, to use it, install
https://github.com/openshift/source-to-image
EOF
```

関連情報

- [S2I イメージ作成のチュートリアル](#)

4.4. SOURCE-TO-IMAGE イメージのテストについて

Source-to-Image (S2I) ビルダーイメージの作成者は、S2I イメージをローカルでテストして、自動テストや継続的な統合に OpenShift Container Platform ビルドシステムを使用できます。

S2I ビルドを正常に実行するには、S2I に **assemble** と **run** スクリプトが必要です。S2I 外のコンテナイメージを実行した場合に、**save-artifacts** スクリプトがあると、ビルドのアーティファクトが再利用され、**usage** スクリプトがあると、使用についての情報がコンソールに出力されるようになります。

S2I イメージのテストは、ベースのコンテナイメージを変更したり、コマンドが使用するツールが更新されたりした場合でも、上記のコマンドが正しく機能することを確認するのが目的です。

4.4.1. テスト要件について

test スクリプトは、基本的に **test/run** に配置されます。このスクリプトは、OpenShift Container Platform S2I イメージビルダーが呼び出し、単純な Bash スクリプトか静的な Go バイナリーのいずれかの形式を取ることができます。

test/run スクリプトは S2I ビルドを実行するので、S2I バイナリーを **\$PATH** で利用可能にしておく必要があります。必要に応じて、[S2I README](#) のインストール手順に従います。

S2I は、アプリケーションのソースコードおよびビルダーイメージを統合します。これをテストするには、ソースが実行可能なコンテナイメージに変換されることを検証するためのサンプルアプリケーションのソースが必要です。サンプルアプリケーションは単純なものである必要があります

が、**assemble** および **run** スクリプトの重要な手順を実行できる必要があります。

4.4.2. スクリプトおよびツールの生成

S2I ツールは、新しい S2I イメージの作成プロセスを加速化する強力な生成ツールと共に提供されます。**s2i create** コマンドでは、**Makefile** 以外に、必要とされる S2I スクリプトとテストツールすべてが生成されます。

```
$ s2i create <image name> <destination directory>
```

生成された **test/run** スクリプトは、より使いやすくするために調整する必要がありますが、このスクリプトを開発の開始段階で使用することが推奨されます。



注記

s2i create コマンドで生成した **test/run** スクリプトでは、サンプルアプリケーションのソースを **test/test-app** ディレクトリーに配置しておく必要があります。

4.4.3. ローカルでのテスト

S2I イメージテストをローカルに実行する最も簡単な方法として、生成した **Makefile** を使用することができます。

s2i create コマンドを使用しない場合には、以下の **Makefile** テンプレートをコピーして、**IMAGE_NAME** パラメーターをお使いのイメージ名に置き換えることができます。

Makefile の例

```
IMAGE_NAME = openshift/ruby-20-centos7
CONTAINER_ENGINE := $(shell command -v podman 2> /dev/null | echo docker)

build:
  ${CONTAINER_ENGINE} build -t $(IMAGE_NAME) .

.PHONY: test
test:
  ${CONTAINER_ENGINE} build -t $(IMAGE_NAME)-candidate .
  IMAGE_NAME=$(IMAGE_NAME)-candidate test/run
```

4.4.4. テストの基本的なワークフロー

test スクリプトは、テストするイメージをすでにビルドしていることが前提です。必要に応じて、以下のコマンドで S2I イメージを先にビルドしてください。以下のいずれかのコマンドを実行してください。

- Podman を使用する場合は、以下のコマンドを実行します。

```
$ podman build -t <builder_image_name>
```

- Docker を使用する場合は、以下のコマンドを実行します。

```
$ docker build -t <builder_image_name>
```

以下の手順では、S2I イメージビルダーをテストするデフォルトのワークフローを説明しています。

1. **usage** スクリプトが機能していることを確認します。

- Podman を使用する場合は、以下のコマンドを実行します。

```
$ podman run <builder_image_name> .
```

- Docker を使用する場合は、以下のコマンドを実行します。

```
$ docker run <builder_image_name> .
```

2. イメージをビルドします。

```
$ s2i build file:///path-to-sample-app _<BUILDER_IMAGE_NAME>_  
_<OUTPUT_APPLICATION_IMAGE_NAME>_
```

3. オプション: **save-artifacts** をサポートする場合には、再度手順 2 を実行して、保存して復元するアーティファクトが正しく機能することを確認します。

4. コンテナを実行します。

- Podman を使用する場合は、以下のコマンドを実行します。

```
$ podman run <output_application_image_name>
```

- Docker を使用する場合は、以下のコマンドを実行します。

```
$ docker run <output_application_image_name>
```

5. コンテナが実行され、アプリケーションが応答していることを確認します。

これらの手順を実行すると、通常はビルダーイメージが予想通りに機能しているかどうか分かります。

4.4.5. イメージのビルドでの OpenShift Container Platform の使用

新しい S2I ビルダーイメージを設定する **Dockerfile** と他のアーティファクトが準備できたら、それらを git リポジトリに配置して、OpenShift Container Platform を使用し、イメージをビルドしてプッシュします。お使いのリポジトリを参照する Docker ビルドを定義します。

OpenShift Container Platform インスタンスが公開 IP アドレスでホストされる場合、ビルドは、S2I ビルダーイメージ GitHub リポジトリにプッシュするたびにトリガーされます。

ImageChangeTrigger を使用して、更新した S2I ビルダーイメージに基づくアプリケーションの再ビルドをトリガーすることもできます。

第5章 イメージの管理

5.1. イメージの管理の概要

OpenShift Container Platform では、イメージのレジストリーが置かれる場所やレジストリー関連の認証要件、およびビルドとデプロイメントで必要とされる動作に応じてイメージと対話し、イメージストリームをセットアップできます。

5.1.1. イメージの概要

イメージストリームは、タグで識別される数多くのコンテナイメージで設定されます。これはコンテナイメージリポジトリのように関連イメージの単一仮想ビューを提供します。

イメージストリームの監視により、ビルドおよびデプロイメントは新規イメージの追加または変更時に通知を受信し、それぞれビルドまたはデプロイメントを実行してこれに対応します。

5.2. イメージのタグ付け

以下のセクションでは、OpenShift Container Platform イメージストリームおよびそれらのタグを操作するためにコンテナイメージのコンテキストでイメージタグを使用する概要および方法について説明します。

5.2.1. イメージタグ

イメージタグは、イメージストリーム内の他のイメージから特定のイメージを識別するリポジトリのコンテナイメージに適用されるラベルです。通常、タグはある種のバージョン番号を表します。たとえば、ここでは **:v3.11.59-2** がタグになります。

```
registry.access.redhat.com/openshift3/jenkins-2-rhel7:v3.11.59-2
```

イメージにタグを追加することができます。たとえば、イメージには **:v3.11.59-2** および **:latest** というタグが割り当てられる可能性があります。

OpenShift Container Platform は **docker tag** コマンドに似た **oc tag** コマンドを提供しますが、これらはイメージ上で直接動作するのではなくイメージストリームで動作します。

5.2.2. イメージタグの規則

イメージは時間の経過と共に変化するもので、それらのタグはその変化を反映します。ほとんどの場合、イメージタグはビルドされる最新イメージを常に参照します。

v2.0.1-may-2019 のように、タグ名に非常に多くの情報が組み込まれる場合、タグはイメージの単一のリビジョンのみを参照し、更新されることはありません。デフォルトのイメージのプルニングオプションを使用しても、このようなイメージは削除されません。非常に大規模なクラスターでは、イメージが修正されるたびに新規タグが作成される設定の場合、古くなって久しいイメージの余分のタグメタデータで etcd データストアが一杯になる可能性があります。

タグの名前が **v2.0** である場合はイメージのリビジョンの数が増えることが予想されます。これによりタグ履歴が長くなるため、イメージプルーナーが古くなり使われなくなったイメージを削除する可能性が高くなります。

タグの名前付け規則は各自で定めることができますが、ここでは **<image_name>:<image_tag>** 形式のいくつかの例を見てみましょう。

表5.1 イメージタグの名前付け規則

説明	例
リビジョン	<code>myimage:v2.0.1</code>
アーキテクチャー	<code>myimage:v2.0-x86_64</code>
ベースイメージ	<code>myimage:v1.2-centos7</code>
最新 (不安定な可能性がある)	<code>myimage:latest</code>
最新 (安定性がある)	<code>myimage:stable</code>

タグ名に日付を含める必要がある場合、古くなり使用されなくなったイメージおよび **istags** を定期的に検査し、これらを削除してください。そうしないと、古いイメージを保持して、リソースの使用量が增大する可能性があります。

5.2.3. タグのイメージストリームへの追加

OpenShift Container Platform のイメージストリームは、タグで識別される 0 個以上のコンテナイメージで設定されます。

各種のタグを利用できます。デフォルト動作では、特定の時点の特定のイメージを参照する 永続 タグを使用します。 **permanent** タグが使用され、ソースが変更される場合、タグは宛先について変更されません。

tracking タグの場合は、宛先タグのメタデータがソースタグのインポート時に更新されます。

手順

- **oc tag** コマンドを使用して、タグをイメージストリームに追加できます。

```
$ oc tag <source> <destination>
```

たとえば、**ruby** イメージストリームの **static-2.0** タグを **ruby** イメージストリーム **2.0** タグの現行のイメージを常に参照するように設定するには、以下を実行します。

```
$ oc tag ruby:2.0 ruby:static-2.0
```

これにより、**ruby** イメージストリームに **static-2.0** という名前のイメージストリームタグが新たに作成されます。この新規タグは、**oc tag** の実行時に **ruby:2.0** イメージストリームタグが参照したイメージ ID を直接参照し、これが参照するイメージが変更されることがありません。

- 宛先タグがソースタグの変更時に更新されるようにするには、**--alias=true** フラグを使用します。

```
$ oc tag --alias=true <source> <destination>
```



注記

永続的なエイリアス (**latest** または **stable** など) を作成するには、tracking タグを使用します。このタグは単一イメージストリーム内でのみ適切に機能します。複数のイメージストリーム間で使用されるエイリアスを作成しようとするとエラーが生じます。

- また、**--scheduled=true** フラグを追加して、宛先タグが定期的に更新 (再インポート) されるようにもできます。期間はシステムレベルでグローバルに設定できます。
- **--reference** フラグはインポートされないイメージストリームを作成します。このタグはソースの場所を参照しますが、これを永続的に参照します。
統合レジストリーのタグ付けされたイメージを常にフェッチするように OpenShift Container Platform に指示するには、**--reference-policy=local** を使用します。レジストリーはプルスルー (pull-through) 機能を使用してイメージをクライアントに提供します。デフォルトで、イメージ Blob はレジストリーによってローカルにミラーリングされます。その結果、それらが次回必要となる場合により迅速にプルされます。また、このフラグは **--insecure-registry** をコンテナランタイムに指定しなくても、イメージストリームに非セキュアなアノテーションがあるか、またはタグに非セキュアなインポートポリシーがある限り、非セキュアなレジストリーからのプルを許可します。

5.2.4. タグのイメージストリームからの削除

タグをイメージストリームから削除できます。

手順

- タグをイメージストリームから完全に削除するには、以下を実行します。

```
$ oc delete istag/ruby:latest
```

または、以下を実行します。

```
$ oc tag -d ruby:latest
```

5.2.5. イメージストリームでのイメージの参照

タグを使用してイメージストリームのイメージを参照するには、以下の参照タイプを使用します。

表5.2 イメージストリームの参照タイプ

参照タイプ	説明
ImageStreamTag	ImageStreamTag は、所定のイメージストリームおよびタグのイメージを参照し、取得するために使用されます。
ImageStreamImage	ImageStreamImage は、所定のイメージストリームおよびイメージ sha ID を参照するか、または取得するために使用されます。

参照タイプ	説明
DockerImage	DockerImage は、所定の外部レジストリーのイメージを参照または取得するために使用されます。この名前は、標準の Docker pull specification に基づいて付けられます。

イメージストリーム定義のサンプルを表示すると、これらには **ImageStreamTag** の定義と **DockerImage** の参照が含まれていますが、**ImageStreamImage** に関連するものは何も含まれていないことに気づくでしょう。

これは、**ImageStreamImage** オブジェクトが、イメージをイメージストリームにインポートしたり、タグ付けしたりする際に OpenShift Container Platform に自動的に作成されるためです。イメージストリームを作成するために使用するイメージストリーム定義で **ImageStreamImage** オブジェクトを明示的に定義する必要はありません。

手順

- 所定のイメージストリームおよびタグのイメージを参照するには、**ImageStreamTag** を使用します。

```
<image_stream_name>:<tag>
```

- 所定のイメージストリームおよびイメージの **sha** ID のイメージを参照するには、**ImageStreamImage** を使用します。

```
<image_stream_name>@<id>
```

<id> は、ダイジェストとも呼ばれる特定イメージのイミュータブルな ID です。

- 所定の外部レジストリーのイメージを参照または取得するには、**DockerImage** を使用します。

```
openshift/ruby-20-centos7:2.0
```



注記

タグが指定されていない場合、**latest** タグが使用されることが想定されます。

サードパーティーのレジストリーを参照することもできます。

```
registry.redhat.io/rhel7:latest
```

またはダイジェストでイメージを参照できます。

```
centos/ruby-22-  
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b2  
8e
```

5.3. イメージプルポリシー

Pod のそれぞれのコンテナにはコンテナイメージがあります。イメージを作成し、これをレジストリーにプッシュすると、イメージを Pod で参照できます。

5.3.1. イメージプルポリシーの概要

OpenShift Container Platform はコンテナを作成する際に、コンテナの **imagePullPolicy** を使用して、コンテナの起動前にイメージをプルする必要があるかどうかを判別します。**imagePullPolicy** には以下の3つの値があります。

表5.3 **imagePullPolicy** の値

値	説明
Always	常にイメージをプルします。
IfNotPresent	イメージがノード上にない場合にのみイメージをプルします。
Never	イメージをプルしません。

コンテナの **imagePullPolicy** パラメーターが指定されていない場合、OpenShift Container Platform はイメージのタグに基づいてこれを設定します。

1. タグが **latest** の場合、OpenShift Container Platform は **imagePullPolicy** を **Always** にデフォルト設定します。
2. それ以外の場合に、OpenShift Container Platform は **imagePullPolicy** を **IfNotPresent** にデフォルト設定します。

5.4. イメージプルシークレットの使用

OpenShift Container Platform の内部レジストリーを使用し、同じプロジェクトにあるイメージストリームからプルしている場合は、Pod のサービスアカウントに適切なパーミッションがすでに設定されているために追加のアクションは不要です。

ただし、OpenShift Container Platform プロジェクト全体でイメージを参照する場合や、セキュリティー保護されたレジストリーからイメージを参照するなどの他のシナリオでは、追加の設定手順が必要になります。

Red Hat OpenShift Cluster Manager サイトの [Pull Secret](#) ページからイメージプルシークレット **pullSecret** を取得することもできます。

このプルシークレットを使用し、OpenShift Container Platform コンポーネントのコンテナイメージを提供する、[Quay.io](#) および [registry.redhat.io](#) を含む、組み込まれた各種の認証局によって提供されるサービスで認証できます。

config.json ファイルのサンプル

```
{
  "auths": {
    "cloud.openshift.com": {
      "auth": "b3Blb=",
      "email": "you@example.com"
    }
  }
}
```

```

    },
    "quay.io":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    }
  }
}

```

5.4.1. Pod が複数のプロジェクト間でイメージを参照できるようにする設定

内部レジストリーを使用している場合で **project-a** の Pod が **project-b** のイメージを参照できるようにするには、**project-a** のサービスアカウントが **project-b** の **system:image-puller** ロールにバインドされている必要があります。

手順

1. **project-a** の Pod が **project-b** のイメージを参照できるようにするには、**project-a** のサービスアカウントを **project-b** の **system:image-puller** ロールにバインドします。

```

$ oc policy add-role-to-user \
  system:image-puller system:serviceaccount:project-a:default \
  --namespace=project-b

```

このロールを追加すると、デフォルトのサービスアカウントを参照する **project-a** の Pod は **project-b** からイメージをプルできるようになります。

2. **project-a** のすべてのサービスアカウントにアクセスを許可するには、グループを使用します。

```

$ oc policy add-role-to-group \
  system:image-puller system:serviceaccounts:project-a \
  --namespace=project-b

```

5.4.2. Pod が他のセキュリティー保護されたレジストリーからイメージを参照できるようにする設定

Docker クライアントの **.dockercfg \$HOME/.docker/config.json** ファイルは、セキュア/非セキュアなレジストリーに事前にログインしている場合に認証情報を保存する Docker 認証情報ファイルです。

OpenShift Container Platform の内部レジストリーにないセキュリティー保護されたコンテナイメージをプルするには、Docker 認証情報でプルシークレットを作成し、これをサービスアカウントに追加する必要があります。

手順

- セキュリティー保護されたレジストリーの **.dockercfg** ファイルがすでにある場合は、以下を実行してそのファイルからシークレットを作成できます。

```

$ oc create secret generic <pull_secret_name> \
  --from-file=.dockercfg=<path/to/.dockercfg> \
  --type=kubernetes.io/dockercfg

```

- または、**\$HOME/.docker/config.json** ファイルがある場合は以下を実行します。


```
$ oc create secret generic <pull_secret_name> \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```

- セキュアなレジストリーについての Docker 認証情報ファイルがまだない場合には、以下のコマンドを実行してシークレットを作成することができます。

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

- Pod のイメージをプルするためのシークレットを使用するには、そのシークレットをサービスアカウントに追加する必要があります。この例では、サービスアカウントの名前は、Pod が使用するサービスアカウントの名前に一致している必要があります。デフォルトのサービスアカウントは **default** です。

```
$ oc secrets link default <pull_secret_name> --for=pull
```

5.4.2.1. 委任された認証を使用したプライベートレジストリーからのプル

プライベートレジストリーは認証を別個のサービスに委任できます。この場合、イメージプルシークレットは認証およびレジストリーのエンドポイントの両方に対して定義される必要があります。

手順

1. 委任された認証サーバーのシークレットを作成します。

```
$ oc create secret docker-registry \
  --docker-server=sso.redhat.com \
  --docker-username=developer@example.com \
  --docker-password=***** \
  --docker-email=unused \
  redhat-connect-sso

secret/redhat-connect-sso
```

2. プライベートレジストリーのシークレットを作成します。

```
$ oc create secret docker-registry \
  --docker-server=privateregistry.example.com \
  --docker-username=developer@example.com \
  --docker-password=***** \
  --docker-email=unused \
  private-registry

secret/private-registry
```

5.4.3. グローバルクラスターのプルシークレットの更新

クラスターのグローバルプルシークレットを更新できます。



警告

クラスターリソースは新規のプルシークレットに合わせて調整する必要がありますが、これにより、クラスターのユーザービリティが一時的に制限される可能性があります。



警告

グローバルプルシークレットを更新すると、Machine Config Operator (MCO) が変更を同期している間にノードが再起動します。

前提条件

- アップロードする新規または変更されたプルシークレットファイルがある。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

- 以下のコマンドを実行して、クラスターのグローバルプルシークレットを更新します。

```
$ oc set data secret/pull-secret -n openshift-config --from-file=.dockerconfigjson=<pull-secret-location> ①
```

- ① 新規プルシークレットファイルへのパスを指定します。

この更新はすべてのノードにロールアウトされます。これには、クラスターのサイズに応じて多少時間がかかる場合があります。この間に、ノードがドレイン (解放) され、Pod は残りのノードで再スケジューリングされます。

第6章 イメージストリームの管理

イメージストリームは、継続的な方法でコンテナイメージの作成および更新を行う手段を提供します。イメージの改良により、タグを使用して新規バージョン番号を割り当て、変更を追跡できるようになりました。本書では、イメージストリームの管理方法について説明します。

6.1. イメージストリームの使用

イメージストリームおよびその関連付けられたタグは、OpenShift Container Platform 内でコンテナイメージを参照するための抽象化を提供します。イメージストリームとそのタグを使用して、利用可能なイメージを確認し、リポジトリのイメージが変更される場合でも必要な特定のイメージを使用していることを確認できます。

イメージストリームには実際のイメージデータは含まれませんが、イメージリポジトリと同様に、関連するイメージの単一の仮想ビューが提示されます。

ビルドおよびデプロイメントをそれぞれ実行し、ビルドおよびデプロイメントを、新規イメージが追加される際やこれに対応する際の通知をイメージストリームで確認できるように設定できます。

たとえば、デプロイメントで特定のイメージを使用していて、そのイメージの新規バージョンが作成される場合、デプロイメントを、そのイメージの新規バージョンを選択できるように自動的に実行します。

デプロイメントまたはビルドで使用するイメージストリームタグが更新されない場合には、コンテナイメージレジストリーのコンテナイメージが更新されても、ビルドまたはデプロイメントは以前の、既知でおそらく適切であると予想されるイメージをそのまま使用します。

ソースイメージは以下のいずれかに保存できます。

- OpenShift Container Platform の統合レジストリー
- registry.redhat.io or Quay.io などの外部レジストリー
- OpenShift Container Platform クラスターの他のイメージストリーム

ビルドまたはデプロイメント設定などのイメージストリームタグを参照するオブジェクトを定義する場合には、リポジトリではなく、イメージストリームタグを参照します。アプリケーションのビルドまたはデプロイ時に、OpenShift Container Platform がこのイメージストリームタグを使用してリポジトリに対してクエリーし、対象のイメージに関連付けられた ID を特定して、そのイメージを使用します。

イメージストリームメタデータは他のクラスター情報と共に etcd インスタンスに保存されます。

イメージストリームの使用には、いくつかの大きな利点があります。

- コマンドラインを使用して再プッシュすることなく、タグ付けや、タグのロールバック、およびイメージの迅速な処理を実行できます。
- 新規イメージがレジストリーにプッシュされると、ビルドおよびデプロイメントをトリガーできます。また、OpenShift Container Platform には他のリソースの汎用トリガーがあります (Kubernetes オブジェクトなど)。
- 定期的な再インポートを実行するためにタグにマークを付けることができます。ソースイメージが変更されると、その変更は選択され、イメージストリームに反映されます。これにより、ビルドまたはデプロイメント設定に応じてビルドまたはデプロイメントフローがトリガーされます。

- 詳細なアクセス制御を使用してイメージを共有し、チーム間でイメージを迅速に分散できません。
- ソースイメージが変更されると、イメージストリームタグはイメージの既知の適切なバージョンをポイントしたままになり、アプリケーションが予期せずに損傷しないようにします。
- イメージストリームオブジェクトのパーミッションを使用して、イメージを表示し、使用できるユーザーについてセキュリティーを設定することができます。
- クラスターレベルでイメージを読み込んだり、一覧表示するパーミッションのないユーザーは、イメージストリームを使用してプロジェクトでタグ付けされたイメージを取得できます。

6.2. イメージストリームの設定

ImageStream オブジェクトには以下の要素が含まれます。

イメージストリームオブジェクト定義

```

apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  creationTimestamp: 2017-09-29T13:33:49Z
  generation: 1
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample 1
  namespace: test
  resourceVersion: "633"
  selflink: /oapi/v1/namespaces/test/imagestreams/origin-ruby-sample
  uid: ee2b9405-c68c-11e5-8a99-525400f25e34
spec: {}
status:
  dockerImageRepository: 172.30.56.218:5000/test/origin-ruby-sample 2
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d 3
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5 4
    - created: 2017-09-29T13:40:11Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
      generation: 1
      image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
      tag: latest 5

```

1 イメージストリームの名前です。

2 新規イメージをこのイメージストリームで追加または更新するためにプッシュできる Docker リポジトリパスです。

- 3 イメージストリームが現在参照する SHA ID です。このイメージストリームタグを参照するリソースはこの ID を使用します。
- 4 このイメージストリームタグが以前に参照した SHA ID です。古いイメージにロールバックするために使用できます。
- 5 イメージストリームタグ名です。

6.3. イメージストリームイメージ

イメージストリームイメージは、イメージストリームから特定のイメージ ID をポイントします。

イメージストリームイメージにより、タグ付けされている特定のイメージストリームからイメージについてのメタデータを取得できます。

イメージストリームイメージオブジェクトは、イメージをイメージストリームにインポートしたり、タグ付けしたりする場合には OpenShift Container Platform に常に自動的に作成されます。イメージストリームを作成するために使用するイメージストリームイメージオブジェクトをイメージストリーム定義に明示的に定義する必要はありません。

イメージストリームイメージはリポジトリからのイメージストリーム名およびイメージ ID で設定されており、@ 記号で区切られています。

```
<image-stream-name>@<image-id>
```

ImageStream オブジェクトのサンプルでイメージを参照する際、イメージストリームのイメージは以下ようになります。

```
origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
```

6.4. イメージストリームタグ

イメージストリームタグは、イメージストリームのイメージに対する名前付きポインターです。これは **istag** として省略されます。イメージストリームタグは、指定のイメージストリームおよびタグのイメージを参照するか、または取得するために使用されます。

イメージストリームタグは、ローカル、または外部で管理されるイメージを参照できます。これには、タグが参照したすべてのイメージのスタックとして表されるイメージの履歴が含まれます。新規または既存のイメージが特定のイメージストリームタグでタグ付けされる場合はいつでも、これは履歴スタックの最初の位置に置かれます。これまで先頭の位置を占めていたイメージは 2 番目の位置に置かれます。これにより、タグを過去のイメージに再び参照させるよう簡単にロールバックできます。

以下のイメージストリームタグは、**ImageStream** オブジェクトからのものです。

履歴の 2 つのイメージを持つイメージストリームタグ

```
tags:
- items:
- created: 2017-09-02T10:15:09Z
  dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
  generation: 2
```

```

image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
- created: 2017-09-29T13:40:11Z
dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
generation: 1
image: sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
tag: latest

```

イメージストリームタグには permanent タグまたは tracking タグを使用できます。

- Permanent タグは、Python 3.5 などの特定バージョンのイメージを参照するバージョン固有のタグです。
- tracking タグは別のイメージストリームタグに従う参照タグで、シンボリックリンクなどのように、フォローするイメージを変更するために更新される可能性があります。このような新規レベルでは後方互換性が確保されません。

たとえば、OpenShift Container Platform に同梱される **latest** イメージストリームタグはトラッキングタグです。これは、**latest** イメージストリームタグのコンシューマーが、新規レベルが利用可能になるとイメージで提供されるフレームワークの最新レベルに更新されることを意味します。**v3.10** への **latest** イメージストリームタグは **v3.11** に変更される可能性が常にあります。これらの **latest** イメージストリームタグは Docker **latest** タグと異なる動作をすることに注意してください。この場合、**latest** イメージストリームタグは Docker リポジトリの最新イメージを参照しません。これは別のイメージストリームタグを参照し、これはイメージの最新バージョンではない可能性があります。たとえば、**latest** イメージストリームタグがイメージの **v3.10** を参照する場合、**3.11** バージョンがリリースされても **latest** タグは **v3.11** に自動的に更新されず、これが **v3.11** イメージストリームタグを参照するように手動で更新されるまで **v3.10** を参照したままになります。



注記

トラッキングタグは単一のイメージストリームに制限され、他のイメージストリームを参照できません。

各自のニーズに合わせて独自のイメージストリームタグを作成できます。

イメージストリームタグは、コロンで区切られた、イメージストリームの名前とタグで設定されています。

```
<imagestream name>:<tag>
```

たとえば、上記の **ImageStream** オブジェクトのサンプルで **sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d** イメージを参照するには、イメージストリームタグは以下のようになります。

```
origin-ruby-sample:latest
```

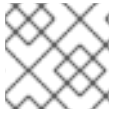
6.5. イメージストリーム変更トリガー

イメージストリームトリガーにより、ビルドおよびデプロイメントは、アップストリームの新規バージョンが利用可能になると自動的に起動します。

たとえば、ビルドおよびデプロイメントは、イメージストリームタグの変更時に自動的に起動します。これは、特定のイメージストリームタグをモニターし、変更の検出時にビルドまたはデプロイメントに通知することで実行されます。

6.6. イメージストリームのマッピング

統合レジストリーが新規イメージを受信する際、これは OpenShift Container Platform にマップするイメージストリームを作成し、送信し、イメージのプロジェクト、名前、タグおよびイメージメタデータを提供します。



注記

イメージストリームのマッピングの設定は高度な機能です。

この情報は、新規イメージを作成する際 (すでに存在しない場合) やイメージをイメージストリームにタグ付けする際に使用されます。OpenShift Container Platform は、コマンド、エントリーポイント、および環境変数などの各イメージについての完全なメタデータを保存します。OpenShift Container Platform のイメージはイミュータブル (変更不可能) であり、名前の最大長さは 63 文字です。

以下のイメージストリームマッピングのサンプルにより、イメージが **test/origin-ruby-sample:latest** としてタグ付けされます。

イメージストリームマッピングオブジェクト定義

```

apiVersion: v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddf10ace3c6ef
    size: 0
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
    size: 11939149
  dockerImageMetadata:
    Architecture: amd64
    Config:
      Cmd:
      - /usr/libexec/s2i/run
    Entrypoint:
      - container-entrypoint
    Env:
      - RACK_ENV=production
      - OPENSIFT_BUILD_NAMESPACE=test
      - OPENSIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app

```

```

- OPENSIFT_BUILD_NAME=ruby-sample-build-1
- PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
- STI_SCRIPTS_URL=image:///usr/libexec/s2i
- STI_SCRIPTS_PATH=/usr/libexec/s2i
- HOME=/opt/app-root/src
- BASH_ENV=/opt/app-root/etc/scl_enable
- ENV=/opt/app-root/etc/scl_enable
- PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
- RUBY_VERSION=2.2
ExposedPorts:
  8080/tcp: {}
Labels:
  build-date: 2015-12-23
  io.k8s.description: Platform for building and running Ruby 2.2 applications
  io.k8s.display-name: 172.30.56.218:5000/test/origin-ruby-sample:latest
  io.openshift.build.commit.author: Ben Parees <bparees@users.noreply.github.com>
  io.openshift.build.commit.date: Wed Jan 20 10:14:27 2016 -0500
  io.openshift.build.commit.id: 00cadc392d39d5ef9117cbc8a31db0889eedd442
  io.openshift.build.commit.message: 'Merge pull request #51 from php-coder/fix_url_and_sti'
  io.openshift.build.commit.ref: master
  io.openshift.build.image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
  io.openshift.build.source-location: https://github.com/openshift/ruby-hello-world.git
  io.openshift.builder-base-version: 8d95148
  io.openshift.builder-version: 8847438ba06307f86ac877465eadc835201241df
  io.openshift.s2i.scripts-url: image:///usr/libexec/s2i
  io.openshift.tags: builder,ruby,ruby22
  io.s2i.scripts-url: image:///usr/libexec/s2i
  license: GPLv2
  name: CentOS Base Image
  vendor: CentOS
User: "1001"
WorkingDir: /opt/app-root/src
Container: 86e9a4a3c760271671ab913616c51c9f3cea846ca524bf07c04a6f6c9e103a76
ContainerConfig:
  AttachStdout: true
  Cmd:
  - /bin/sh
  - -c
  - tar -C /tmp -xf - && /usr/libexec/s2i/assemble
  Entrypoint:
  - container-entrypoint
  Env:
  - RACK_ENV=production
  - OPENSIFT_BUILD_NAME=ruby-sample-build-1
  - OPENSIFT_BUILD_NAMESPAC=test
  - OPENSIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
  - EXAMPLE=sample-app
  - PATH=/opt/app-root/src/bin:/opt/app-
root/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
  - STI_SCRIPTS_URL=image:///usr/libexec/s2i
  - STI_SCRIPTS_PATH=/usr/libexec/s2i
  - HOME=/opt/app-root/src
  - BASH_ENV=/opt/app-root/etc/scl_enable
  - ENV=/opt/app-root/etc/scl_enable

```



```

- PROMPT_COMMAND=. /opt/app-root/etc/scl_enable
- RUBY_VERSION=2.2
ExposedPorts:
  8080/tcp: {}
Hostname: ruby-sample-build-1-build
Image: centos/ruby-22-
centos7@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
OpenStdin: true
StdinOnce: true
User: "1001"
WorkingDir: /opt/app-root/src
Created: 2016-01-29T13:40:00Z
DockerVersion: 1.8.2.fc21
Id: 9d7fd5e2d15495802028c569d544329f4286dcd1c9c085ff5699218dbaa69b43
Parent: 57b08d979c86f4500dc8cad639c9518744c8dd39447c055a3517dc9c18d6fccd
Size: 441976279
apiVersion: "1.0"
kind: DockerImage
dockerImageMetadataVersion: "1.0"
dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d

```

6.7. イメージストリームの使用

以下のセクションでは、イメージストリームおよびイメージストリームタグを使用する方法について説明します。

6.7.1. イメージストリームについての情報の取得

イメージストリームについての一般的な情報およびこれがポイントするすべてのタグについての詳細情報を取得することができます。

手順

- イメージストリームについての一般的な情報およびこれがポイントするすべてのタグについての詳細情報を取得します。

```
$ oc describe is/<image-name>
```

以下に例を示します。

```
$ oc describe is/python
```

出力例

```

Name: python
Namespace: default
Created: About a minute ago
Labels: <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup: local=false
Unique Images: 1

```

Tags: 1

3.5

tagged from centos/python-35-centos7

* centos/python-35-centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
About a minute ago

- 特定のイメージストリームタグについて利用可能な情報をすべて取得します。

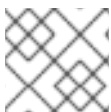
```
$ oc describe istag/<image-stream>:<tag-name>
```

以下に例を示します。

```
$ oc describe istag/python:latest
```

出力例

```
Image Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Docker Image: centos/python-35-centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Name: sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
Created: 2 minutes ago
Image Size: 251.2 MB (first layer 2.898 MB, last binary layer 72.26 MB)
Image Created: 2 weeks ago
Author: <none>
Arch: amd64
Entrypoint: container-entrypoint
Command: /bin/sh -c $STI_SCRIPTS_PATH/usage
Working Dir: /opt/app-root/src
User: 1001
Exposes Ports: 8080/tcp
Docker Labels: build-date=20170801
```



注記

表示されている以上の情報が出力されます。

6.7.2. タグのイメージストリームへの追加

追加タグをイメージストリームに追加できます。

手順

- 既存タグのいずれかを参照するタグを追加するには、`oc tag` コマンドを使用できます。

```
$ oc tag <image-name:tag1> <image-name:tag2>
```

以下に例を示します。

```
$ oc tag python:3.5 python:latest
```

出力例

```
Tag python:latest set to
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25.
```

- イメージストリームに、外部コンテナイメージを参照するタグ (3.5) と、この最初のタグに基づいて作成されているために同じイメージを参照する別のタグ (**latest**) の2つのタグが含まれることを確認します。

```
$ oc describe is/python
```

出力例

```
Name: python
Namespace: default
Created: 5 minutes ago
Labels: <none>
Annotations: openshift.io/image.dockerRepositoryCheck=2017-10-02T17:05:11Z
Docker Pull Spec: docker-registry.default.svc:5000/default/python
Image Lookup: local=false
Unique Images: 1
Tags: 2
```

```
latest
  tagged from
  python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
```

```
* centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
  About a minute ago
```

```
3.5
  tagged from centos/python-35-centos7
```

```
* centos/python-35-
centos7@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25
  5 minutes ago
```

6.7.3. 外部イメージのタグの追加

外部イメージのタグを追加することができます。

手順

- タグ関連のすべての操作に **oc tag** コマンドを使用して、内部または外部イメージをポイントするタグを追加します。

```
$ oc tag <repository/image> <image-name:tag>
```

たとえば、このコマンドは **docker.io/python:3.6.0** イメージを **python** イメージストリームの **3.6** タグにマップします。

```
$ oc tag docker.io/python:3.6.0 python:3.6
```

出力例

```
Tag python:3.6 set to docker.io/python:3.6.0.
```

外部イメージのセキュリティーが保護されている場合、そのレジストリーにアクセスするために認証情報を使ってシークレットを作成する必要があります

6.7.4. イメージストリームタグの更新

別のタグをイメージストリームに反映するようタグを更新できます。

手順

- タグを更新します。

```
$ oc tag <image-name:tag> <image-name:latest>
```

たとえば、以下は **latest** タグを更新し、**3.6** タグをイメージタグに反映させます。

```
$ oc tag python:3.6 python:latest
```

出力例

```
Tag python:latest set to  
python@sha256:438208801c4806548460b27bd1fbc7bb188273d13871ab43f.
```

6.7.5. イメージストリームタグの削除

古いタグをイメージストリームから削除できます。

手順

- 古いタグをイメージストリームから削除します。

```
$ oc tag -d <image-name:tag>
```

以下に例を示します。

```
$ oc tag -d python:3.5
```

出力例

```
Deleted tag default/python:3.5.
```

6.7.6. イメージストリームタグの定期的なインポートの設定

外部コンテナイメージレジストリーを使用している場合、(最新のセキュリティー更新を取得する場合などに) イメージを定期的に再インポートするには、**--scheduled** フラグを使用します。

手順

1. イメージインポートのスケジュール

```
$ oc tag <repository/image> <image-name:tag> --scheduled
```

以下に例を示します。

```
$ oc tag docker.io/python:3.6.0 python:3.6 --scheduled
```

出力例

```
Tag python:3.6 set to import docker.io/python:3.6.0 periodically.
```

このコマンドにより、OpenShift Container Platform はこの特定のイメージストリームタグを定期的に更新します。この期間はクラスター全体のデフォルトで15分に設定されます。

- 定期的なチェックを削除するには、上記のコマンド再実行しますが、**--scheduled** フラグを省略します。本リリースでは、その動作をデフォルトに設定しています。

```
$ oc tag <repository/image> <image-name:tag>
```

6.8. プライベートレジストリーからのイメージおよびイメージストリームのインポート

イメージストリームは、プライベートレジストリーからタグおよびイメージメタデータをインポートするように設定できます。これには認証が必要です。この手順は、Cluster Samples Operator が registry.redhat.io 以外からコンテンツをプルするために使用するレジストリーを変更する場合に適用されます。



注記

セキュアでないレジストリーからインポートする場合には、シークレットに定義されたレジストリーの URL に **:80** ポートの接尾辞を追加するようにしてください。追加していない場合にレジストリーからインポートしようとする、このシークレットは使用されません。

手順

- 以下のコマンドを入力して、認証情報を保存するために使用する **secret** オブジェクトを作成する必要があります。

```
$ oc create secret generic <secret_name> --from-file=.dockerconfigjson=  
<file_absolute_path> --type=kubernetes.io/dockerconfigjson
```

- シークレットが設定されたら、新規イメージストリームを作成するか、または **oc import-image** コマンドを入力します。

```
$ oc import-image <imagestreamtag> --from=<image> --confirm
```

インポートプロセスで OpenShift Container Platform はシークレットを取得してリモートパーティーに提供します。

6.8.1. Pod が他のセキュリティー保護されたレジストリーからイメージを参照できるようにする設定

Docker クライアントの `$.dockerconfig $HOME/.docker/config.json` ファイルは、セキュア/非セキュアなレジストリーに事前にログインしている場合に認証情報を保存する Docker 認証情報ファイルです。

OpenShift Container Platform の内部レジストリーにないセキュリティー保護されたコンテナイメージをプルするには、Docker 認証情報でプルシークレットを作成し、これをサービスアカウントに追加する必要があります。

手順

- セキュリティー保護されたレジストリーの `$.dockerconfig` ファイルがすでにある場合は、以下を実行してそのファイルからシークレットを作成できます。

```
$ oc create secret generic <pull_secret_name> \
  --from-file=.dockerconfig=<path/to/.dockerconfig> \
  --type=kubernetes.io/dockerconfig
```

- または、`$.docker/config.json` ファイルがある場合は以下を実行します。

```
$ oc create secret generic <pull_secret_name> \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```

- セキュアなレジストリーについての Docker 認証情報ファイルがまだない場合には、以下のコマンドを実行してシークレットを作成することができます。

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

- Pod のイメージをプルするためのシークレットを使用するには、そのシークレットをサービスアカウントに追加する必要があります。この例では、サービスアカウントの名前は、Pod が使用するサービスアカウントの名前に一致する必要があります。デフォルトのサービスアカウントは **default** です。

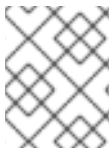
```
$ oc secrets link default <pull_secret_name> --for=pull
```

第7章 KUBERNETES リソースでのイメージストリームの使用

OpenShift Container Platform のネイティブリソースであるイメージストリームは、ビルドまたはデプロイメントなどの OpenShift Container Platform で利用可能な残りのネイティブリソースのすべてと共に追加の設定なしで機能します。これらは、ジョブ、レプリケーションコントローラー、レプリカセットまたは Kubernetes デプロイメントなどのネイティブ Kubernetes リソースと共に機能することもできます。

7.1. KUBERNETES リソースでのイメージストリームの有効化

Kubernetes リソースでイメージストリームを使用する場合、リソースと同じプロジェクトにあるイメージストリームのみを参照できます。イメージストリームの参照は、単一セグメントの値で設定される必要があります。たとえば **ruby:2.5** の場合、**ruby** は **2.5** という名前のタグを持ち、参照するリソースと同じプロジェクトにあるイメージストリームの名前になります。



注記

この機能は、**default** の namespace や **openshift-** または **kube-** の namespace では使用できません。

Kubernetes リソースでイメージストリームを有効にする方法は 2 つあります。

- 特定のリソースでイメージストリームの解決を有効にする。これにより、このリソースのみがイメージフィールドのイメージストリーム名を使用できます。
- イメージストリームでイメージストリームの解決を有効にする。これにより、このイメージストリームを参照するすべてのリソースがイメージフィールドのイメージストリーム名を使用できます。

手順

oc set image-lookup を使用して、特定のリソース上のイメージストリームの解決またはイメージストリーム上のイメージストリームの解決を有効にすることができます。

1. すべてのリソースが **mysql** という名前のイメージストリームを参照できるようにするには、以下のコマンドを入力します。

```
$ oc set image-lookup mysql
```

これにより、**Imagestream.spec.lookupPolicy.local** フィールドが **true** に設定されます。

イメージルックアップが有効なイメージストリーム

```
apiVersion: v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/display-name: mysql
  name: mysql
  namespace: myproject
spec:
  lookupPolicy:
    local: true
```

有効な場合には、この動作はイメージストリーム内のすべてのタグに対して有効化されます。

- 次に、イメージストリームをクエリーし、このオプションが設定されているかどうかを確認できます。

```
$ oc set image-lookup imagestream --list
```

特定のリソースでイメージルックアップを有効にすることができます。

- **mysql** という名前の Kubernetes デプロイメントがイメージストリームを使用できるようにするには、以下のコマンドを実行します。

```
$ oc set image-lookup deploy/mysql
```

これにより、**alpha.image.policy.openshift.io/resolve-names** アノテーションがデプロイメントに設定されます。

イメージルックアップが有効にされたデプロイメント

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  namespace: myproject
spec:
  replicas: 1
  template:
    metadata:
      annotations:
        alpha.image.policy.openshift.io/resolve-names: '*'
    spec:
      containers:
        - image: mysql:latest
          imagePullPolicy: Always
          name: mysql
```

イメージルックアップを無効にすることができます。

- イメージルックアップを無効にするには、**--enabled=false** を渡します。

```
$ oc set image-lookup deploy/mysql --enabled=false
```


第8章 イメージストリームの変更時の更新のトリガー

イメージストリームタグが新規イメージを参照するように更新される場合、OpenShift Container Platform は、古いイメージを使用していたリソースに新規イメージをロールアウトするためのアクションを自動的に実行します。イメージストリームタグを参照しているリソースのタイプに応じ、この動作はさまざまな方法で設定できます。

8.1. OPENSIFT CONTAINER PLATFORM リソース

OpenShift Container Platform デプロイメントの設定およびビルド設定は、イメージストリームタグの変更によって自動的にトリガーされます。トリガーされたアクションは更新されたイメージストリームタグで参照されるイメージの新規の値を使用して実行できます。

8.2. KUBERNETES リソースのトリガー

API 定義の一部としてトリガーを制御するためのフィールドセットを含むデプロイメントおよびビルド設定とは異なり、Kubernetes リソースにはトリガー用のフィールドがありません。その代わりに、OpenShift Container Platform でアノテーションを使用してトリガーを要求できるようにします。

アノテーションは以下のように定義されます。

```
Key: image.openshift.io/triggers
Value:
[
  {
    "from": {
      "kind": "ImageStreamTag", ❶
      "name": "example:latest", ❷
      "namespace": "myapp" ❸
    },
    "fieldPath": "spec.template.spec.containers[?(@.name==\"web\")].image", ❹
    "paused": "false" ❺
  },
  ...
]
```

- ❶ 必須: **kind** は、トリガーするリソースであり、**ImageStreamTag** である必要があります。
- ❷ 必須: **name** はイメージストリームタグの名前である必要があります。
- ❸ オプション: **namespace** はデフォルトでオブジェクトの namespace に設定されます。
- ❹ 必須: **fieldPath** は変更する JSON パスです。このフィールドは制限され、ID またはインデックスでコンテナに正確に一致する JSON パス式のみを受け入れます。Pod の場合、JSON パスは `spec.containers[?(@.name='web')].image` になります。
- ❺ オプション: **paused** はトリガーが一時停止されるかどうかを意味し、デフォルト値は **false** です。このトリガーを一時的に無効にするには、**paused** を **true** に設定します。

コア Kubernetes リソースの1つに Pod テンプレートとこのアノテーションの両方が含まれる場合、OpenShift Container Platform は現時点でトリガーで参照されるイメージストリームタグに関連付けられているイメージを使用してオブジェクトの更新を試行します。この更新は、指定の **fieldPath** に対して実行されます。

Pod テンプレートおよびアノテーションの両方が含まれるコア Kubernetes リソースの例には、以下が含まれます。

- **CronJobs**
- **Deployments**
- **StatefulSets**
- **DaemonSets**
- **Jobs**
- **ReplicationControllers**
- **Pods**

8.3. KUBERNETES リソースでのイメージトリガーの設定

イメージトリガーをデプロイメントに追加する際に、**oc set triggers** コマンドを使用できます。たとえば、この手順のコマンド例は、イメージ変更トリガーを **example** という名前のデプロイメントに追加し、**example:latest** イメージストリームタグの更新時に、デプロイメント内の **web** コンテナが新規の値で更新されるようにします。このコマンドは、デプロイメントリソースに正しい **image.openshift.io/triggers** アノテーションを設定します。

手順

- **oc set triggers** コマンドを入力して Kubernetes リソースをトリガーします。

```
$ oc set triggers deploy/example --from-image=example:latest -c web
```

デプロイメントが一時停止されない限り、この Pod テンプレートの更新により、デプロイメントはイメージの新規の値で自動的に実行されます。

第9章 イメージ設定リソース

以下の手順でイメージレジストリーを設定します。

9.1. イメージコントローラー設定パラメーター

image.config.openshift.io/cluster resource は、イメージの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。spec は以下の設定パラメーターを提供します。

パラメーター	説明
allowedRegistriesForImport	<p>標準ユーザーがイメージのインポートに使用できるコンテナイメージレジストリーを制限します。この一覧を、有効なイメージを含むものとしてユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。イメージまたは ImageStreamMappings を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのはクラスター管理者のみです。</p> <p>このリストのすべての要素に、レジストリーのドメイン名で指定されるレジストリーの場所が含まれます。</p> <p>domainName: レジストリーのドメイン名を指定します。レジストリーが標準以外の (80 または 443) ポートを使用する場合、ポートはドメイン名にも含まれる必要があります。</p> <p>insecure: insecure はレジストリーがセキュアか、または非セキュアであるかを示します。指定がない場合には、デフォルトでレジストリーはセキュアであることが想定されます。</p>
additionalTrustedCA	<p>image stream import、pod image pull、openshift-image-registry pullthrough、およびビルド時に信頼される必要のある追加の CA が含まれる設定マップの参照です。</p> <p>この設定マップの namespace は openshift-config です。設定マップの形式では、信頼する追加のレジストリー CA についてレジストリーのホスト名をキーとして使用し、PEM エンコード証明書を値として使用します。</p>
externalRegistryHostnames	<p>デフォルトの外部イメージレジストリーのホスト名を指定します。外部ホスト名は、イメージレジストリーが外部に公開される場合にのみ設定される必要があります。最初の値は、イメージストリームの publicDockerImageRepository フィールドで使用されます。値は hostname[:port] 形式の値である必要があります。</p>

パラメーター	説明
registrySources	<p>コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。たとえば、非セキュアなアクセスを許可するかどうかを設定します。内部クラスターレジストリーの設定は含まれません。</p> <p>insecureRegistries: 有効な TLS 証明書を持たないか、または HTTP 接続のみをサポートするレジストリーです。</p> <p>blockedRegistries: イメージのプルおよびプッシュアクションについて拒否リストに指定します。他のすべてのレジストリーは許可されます。</p> <p>allowedRegistries: イメージのプルおよびプッシュアクションについて許可リストに指定します。他のすべてのレジストリーはブロックされます。</p> <p>blockedRegistries または allowedRegistries のいずれかを設定できますが、両方を設定することはできません。</p>



警告

allowedRegistries パラメーターが定義されると、明示的に一覧表示されない限り、**registry.redhat.io** レジストリーと **quay.io** レジストリー、およびデフォルトの内部イメージレジストリーを含むすべてのレジストリーがブロックされます。パラメーターを使用する場合は、Pod の失敗を防ぐために、**registry.redhat.io** レジストリーと **quay.io** レジストリー、および **internalRegistryHostname** を含むすべてのレジストリーを **allowedRegistries** 一覧に追加します。これらは、お使いの環境内のペイロードイメージで必要とされます。非接続クラスターの場合、ミラーレジストリーも追加する必要があります。

image.config.openshift.io/cluster リソースの **status** フィールドは、クラスターから観察される値を保持します。

パラメーター	説明
internalRegistryHostname	<p>internalRegistryHostname を制御するイメージレジストリー Operator によって設定されます。これはデフォルトの内部イメージレジストリーのホスト名を設定します。値は hostname[:port] 形式の値である必要があります。後方互換性を確保するために、OPENSIFT_DEFAULT_REGISTRY 環境変数を依然として使用できますが、この設定によってこの環境変数は上書きされます。</p>
externalRegistryHostnames	<p>イメージレジストリー Operator によって設定され、外部に公開される際にイメージレジストリーの外部のホスト名を指定します。最初の値は、イメージストリームの publicDockerImageRepository フィールドで使用されます。値は hostname[:port] 形式の値である必要があります。</p>

9.2. イメージ設定内容の設定

image.config.openshift.io/cluster カスタムリソース (CR) を編集してイメージレジストリーの設定を行うことができます。Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** CR でレジストリーへの変更の有無を監視し、変更を検出するとノードを再起動します。

手順

1. **image.config.openshift.io/cluster** カスタムリソースを編集します。

```
$ oc edit image.config.openshift.io/cluster
```

以下は、**image.config.openshift.io/cluster** CR の例になります。

```
apiVersion: config.openshift.io/v1
kind: Image 1
metadata:
  annotations:
    release.openshift.io/create-only: "true"
    creationTimestamp: "2019-05-17T13:44:26Z"
    generation: 1
    name: cluster
    resourceVersion: "8302"
    selfLink: /apis/config.openshift.io/v1/images/cluster
    uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: 2
    - domainName: quay.io
      insecure: false
  additionalTrustedCA: 3
    name: myconfigmap
  registrySources: 4
    allowedRegistries:
      - example.com
      - quay.io
      - registry.redhat.io
      - image-registry.openshift-image-registry.svc:5000
    insecureRegistries:
      - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

- 1 Image:** イメージの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。
- 2 allowedRegistriesForImport:** 標準ユーザーがイメージのインポートに使用するコンテナイメージレジストリーを制限します。この一覧を、有効なイメージを含むものとしてユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。イメージまたは **ImageStreamMappings** を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのはクラスター管理者のみです。
- 3 additionalTrustedCA:** イメージストリームのインポート、Pod のイメージプル、**openshift-image-registry** プルスルー、およびビルド時に信頼される追加の認証局 (CA) が含まれる設定マップの参照です。この設定マップの namespace は **openshift-**

config です。設定マップの形式では、信頼する追加のレジストリー CA についてレジストリーのホスト名をキーとして使用し、PEM 証明書を値として使用します。

- 4 **registrySources**: コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。たとえば、非セキュアなアクセスを許可するかどうかを設定します。内部クラスターレジストリーの設定は含まれません。この例では、使用可能なレジストリーを定義する **allowedRegistries** を一覧表示します。一覧表示されたレジストリーのいずれかが非セキュアです。

2. 変更が適用されたことを確認するには、ノードを一覧表示します。

```
$ oc get nodes
```

出力例

```

NAME                                STATUS              ROLES    AGE   VERSION
ci-ln-j5cd0qt-f76d1-vfj5x-master-0  Ready              master   98m   v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-master-1  Ready,SchedulingDisabled  master   99m   v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-master-2  Ready              master   98m   v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-worker-b-nsnd4  Ready              worker   90m   v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-worker-c-5z2gz  NotReady,SchedulingDisabled  worker   90m   v1.19.0+7070803
ci-ln-j5cd0qt-f76d1-vfj5x-worker-d-stsjv  Ready              worker   90m   v1.19.0+7070803

```

9.2.1. 特定のレジストリーの追加

image.config.openshift.io/cluster カスタムリソース (CR) を編集してイメージのプおよびプッシュアクションで許可されるレジストリーの一覧を追加できます。OpenShift Container Platform は、この CR への変更をクラスター内のすべてのノードに適用します。

イメージをプルまたはプッシュする場合、コンテナランタイムは **image.config.openshift.io/cluster** CR の **registrySources** パラメーターの下に一覧表示されるレジストリーを検索します。 **allowedRegistries** パラメーターの下にレジストリーの一覧を作成している場合、コンテナランタイムはそれらのレジストリーのみを検索します。一覧に含まれていないレジストリーはブロックされます。



警告

allowedRegistries パラメーターが定義されると、明示的に一覧表示されない限り、**registry.redhat.io** レジストリーと **quay.io** レジストリー、およびデフォルトの内部イメージレジストリーを含むすべてのレジストリーがブロックされます。パラメーターを使用する場合は、Pod の失敗を防ぐために、**registry.redhat.io** レジストリーと **quay.io** レジストリー、および **internalRegistryHostname** を **allowedRegistries** 一覧に追加します。これらは、お使いの環境内のペイロードイメージで必要とされます。非接続クラスターの場合、ミラーレジストリーも追加する必要があります。

手順

1. **image.config.openshift.io/cluster** CR を編集します。

```
$ oc edit image.config.openshift.io/cluster
```

以下は、許可リストを含む **image.config.openshift.io/cluster** リソースの例になります。

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: ①
  allowedRegistries: ②
  - example.com
  - quay.io
  - registry.redhat.io
  - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

- ① **registrySources**: コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。内部クラスターレジストリーの設定は含まれません。
- ② **allowedRegistries**: イメージのプルおよびプッシュアクションに使用するレジストリーです。他のすべてのレジストリーはブロックされます。



注記

allowedRegistries パラメーターまたは **blockedRegistries** パラメーターのいずれかを設定できますが、両方を設定することはできません。

Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** CR でレジストリーへの変更の有無を監視し、変更を検出するとノードを再起動します。許可されるレジストリーへの変更により、各ノードの **/host/etc/containers/policy.json** ファイルのイメージ署名ポリシーが作成または更新されます。

2. レジストリーがポリシーファイルに追加されていることを確認するには、ノードで以下のコマンドを使用します。

```
$ cat /host/etc/containers/policy.json
```

以下のポリシーは、イメージのプルおよびプッシュで、example.com、quay.io、および registry.redhat.io レジストリーからのイメージのみを許可されることを示しています。

例9.1 イメージ署名ポリシーファイルの例

```
{
  "default":[
    {
      "type":"reject"
    }
  ],
  "transports":{
    "atomic":{
      "example.com":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "image-registry.openshift-image-registry.svc:5000":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "insecure.com":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "quay.io":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "reg4.io/myrepo/myapp:latest":[
        {
          "type":"insecureAcceptAnything"
        }
      ],
      "registry.redhat.io":[

```



```
        "type":"insecureAcceptAnything"
      }
    ]
  },
  "docker":{
    "example.com":[
      {
        "type":"insecureAcceptAnything"
      }
    ],
    "image-registry.openshift-image-registry.svc:5000":[
      {
        "type":"insecureAcceptAnything"
      }
    ],
    "insecure.com":[
      {
        "type":"insecureAcceptAnything"
      }
    ],
    "quay.io":[
      {
        "type":"insecureAcceptAnything"
      }
    ],
    "reg4.io/myrepo/myapp:latest":[
      {
        "type":"insecureAcceptAnything"
      }
    ],
    "registry.redhat.io":[
      {
        "type":"insecureAcceptAnything"
      }
    ]
  },
  "docker-daemon":{
    "":[
      {
        "type":"insecureAcceptAnything"
      }
    ]
  }
}
```



注記

クラスターが **registrySources.insecureRegistries** パラメーターを使用する場合、非セキュアなレジストリーが許可リストに含まれることを確認します。

以下に例を示します。

```
spec:
  registrySources:
    insecureRegistries:
      - insecure.com
    allowedRegistries:
      - example.com
      - quay.io
      - registry.redhat.io
      - insecure.com
      - image-registry.openshift-image-registry.svc:5000
```

9.2.2. 特定のレジストリーのブロック

image.config.openshift.io/cluster カスタムリソース (CR) を編集してレジストリーをブロックできます。OpenShift Container Platform は、この CR への変更をクラスター内のすべてのノードに適用します。

イメージをプルまたはプッシュする場合、コンテナーランタイムは **image.config.openshift.io/cluster** CR の **registrySources** パラメーターの下に一覧表示されるレジストリーを検索します。**blockedRegistries** パラメーターの下にレジストリーの一覧を作成した場合、コンテナーランタイムはそれらのレジストリーを検索しません。他のすべてのレジストリーは許可されます。



警告

Pod の失敗を防ぐために、**registry.redhat.io** レジストリーおよび **quay.io** レジストリーを **blockedRegistries** 一覧に追加しないでください。これらは、お使いの環境内のペイロードイメージで必要とされます。

手順

1. **image.config.openshift.io/cluster** CR を編集します。

```
$ oc edit image.config.openshift.io/cluster
```

以下は、ブロックリストを含む **image.config.openshift.io/cluster** CR の例です。

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
```

```

name: cluster
resourceVersion: "8302"
selfLink: /apis/config.openshift.io/v1/images/cluster
uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: ❶
  blockedRegistries: ❷
    - untrusted.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000

```

- ❶ **registrySources**: コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。内部クラスターレジストリーの設定は含まれません。
- ❷ イメージのプルおよびプッシュアクションについて使用できないレジストリーを指定します。他のすべてのレジストリーは許可されます。



注記

blockedRegistries レジストリーまたは **allowedRegistries** レジストリーのいずれかを設定できますが、両方を設定することはできません。

Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** CR でレジストリーへの変更の有無を監視し、変更を検出するとノードを再起動します。ブロックされたレジストリーへの変更は、各ノードの **/etc/containers/registries.conf** ファイルに表示されます。

2. レジストリーがポリシーファイルに追加されていることを確認するには、ノードで以下のコマンドを使用します。

```
$ cat /host/etc/containers/registries.conf
```

以下の例では、**untrusted.com** レジストリーからのイメージが、イメージのプルおよびプッシュで許可されないことを示しています。

出力例

```

unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "untrusted.com"
  blocked = true

```

9.2.3. 非セキュアなレジストリーの許可

image.config.openshift.io/cluster カスタムリソース (CR) を編集して、非セキュアなレジストリーを追加できます。OpenShift Container Platform は、この CR への変更をクラスター内のすべてのノードに適用します。

有効な SSL 証明書を使用しないレジストリー、または HTTPS 接続を必要としないレジストリーは、非セキュアであると見なされます。

**警告**

セキュリティ上のリスクを軽減するために、非セキュアな外部レジストリーは回避する必要があります。

手順

1. **image.config.openshift.io/cluster** CR を編集します。

```
$ oc edit image.config.openshift.io/cluster
```

以下は、非セキュアなレジストリーのリストを含む **image.config.openshift.io/cluster** CR の例になります。

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
    creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  registrySources: ❶
  insecureRegistries: ❷
  - insecure.com
  allowedRegistries:
  - example.com
  - quay.io
  - registry.redhat.io
  - insecure.com ❸
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

- ❶ **registrySources**: コンテナランタイムがビルドおよび Pod のイメージへのアクセス時に個々のレジストリーを処理する方法を決定する設定が含まれます。内部クラスターレジストリーの設定は含まれません。
- ❷ 非セキュアなレジストリーを指定します。
- ❸ 非セキュアなレジストリーが **allowedRegistries** 一覧に含まれていることを確認します。



注記

allowedRegistries パラメーターが定義されると、明示的に一覧表示されない限り、registry.redhat.io レジストリーと quay.io レジストリー、およびデフォルトの内部イメージレジストリーを含むすべてのレジストリーがブロックされます。パラメーターを使用する場合は、Pod の失敗を防ぐために、**registry.redhat.io** レジストリーと **quay.io** レジストリー、および **internalRegistryHostname** を含むすべてのレジストリーを **allowedRegistries** 一覧に追加します。これらは、お使いの環境内のペイロードイメージが必要とされます。非接続クラスターの場合、ミラーレジストリーも追加する必要があります。

Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** CR でレジストリーへの変更の有無を監視し、変更を検出するとノードを再起動します。非セキュアでブロックされたレジストリーへの変更は、各ノードの **/etc/containers/registries.conf** ファイルに表示されます。

2. レジストリーがポリシーファイルに追加されていることを確認するには、ノードで以下のコマンドを使用します。

```
$ cat /host/etc/containers/registries.conf
```

以下の例は、**insecure.com** レジストリーからのイメージが非セキュアであり、イメージのプルおよびプッシュで許可されることを示しています。

出力例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]

[[registry]]
  prefix = ""
  location = "insecure.com"
  insecure = true
```

9.2.4. イメージレジストリーアクセス用の追加のトラストストアの設定

image.config.openshift.io/cluster カスタムリソースには、イメージレジストリーのアクセス時に信頼される追加の認証局が含まれる設定マップへの参照を含めることができます。

前提条件

- 認証局 (CA) は PEM でエンコードされている必要があります。

手順

設定マップを **openshift-config** namespace に作成し、その名前を **image.config.openshift.io** カスタムリソースの **AdditionalTrustedCA** で使用し、追加の CA を指定することができます。

設定マップキーは、この CA が信頼されるポートを持つレジストリーのホスト名であり、base64 エンコード証明書が信頼する追加の各レジストリー CA についての値になります。

イメージレジストリー CA の設定マップの例

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```

name: my-registry-ca
data:
  registry.example.com: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
  registry-with-port.example.com:5000: | ❶
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----

```

- ❶ レジストリーにポートがある場合 (例: **registry-with-port.example.com:5000**)、: は .. に置き換える必要があります。

以下の手順で追加の CA を設定することができます。

- 追加の CA を設定するには、以下を実行します。

```
$ oc create configmap registry-config --from-file=<external_registry_address>=ca.crt -n openshift-config
```

```
$ oc edit image.config.openshift.io cluster
```

```
spec:
  additionalTrustedCA:
    name: registry-config
```

9.2.5. イメージレジストリーのリポジトリミラーリングの設定

コンテナレジストリーのリポジトリミラーリングの設定により、以下が可能になります。

- ソースイメージのレジストリーのリポジトリからイメージをプルする要求をリダイレクトするように OpenShift Container Platform クラスターを設定し、これをミラーリングされたイメージレジストリーのリポジトリで解決できるようにします。
- 各ターゲットリポジトリに対して複数のミラーリングされたリポジトリを特定し、1つのミラーがダウンした場合に別のミラーを使用できるようにします。

以下は、OpenShift Container Platform のリポジトリミラーリングの属性の一部です。

- イメージプルには、レジストリーのダウンタイムに対する回復性があります。
- ネットワークが制限された環境のクラスターは、重要な場所 (quay.io など) からイメージをプルでき、会社のファイアウォールの背後にあるレジストリーが要求されたイメージを提供するようにできます。
- イメージのプル要求時にレジストリーへの接続が特定の順序で試行され、通常は永続レジストリーが最後に試行されます。
- 入力したミラー情報は、OpenShift Container Platform クラスターの全ノードの `/etc/containers/registries.conf` ファイルに追加されます。
- ノードがソースリポジトリからイメージの要求を行うと、要求されたコンテンツを見つけるまで、ミラーリングされた各リポジトリに対する接続を順番に試行します。すべてのミラー

で障害が発生した場合、クラスターはソースリポジトリに対して試行します。成功すると、イメージはノードにプルされます。

リポジトリミラーリングのセットアップは次の方法で実行できます。

- OpenShift Container Platform のインストール時:
OpenShift Container Platform が必要とするコンテナイメージをプルし、それらのイメージを会社のファイアウォールの内側に配置すると、制限されたネットワーク内にあるデータセンターに OpenShift Container Platform をインストールできます。
- OpenShift Container Platform の新規インストール後:
OpenShift Container Platform インストール時にミラーリングを設定しなくても、**ImageContentSourcePolicy** オブジェクトを使用して後で設定することができます。

以下の手順では、インストール後のミラーを設定し、以下を識別する **ImageContentSourcePolicy** オブジェクトを作成します。

- ミラーリングするコンテナイメージリポジトリのソース
- ソースリポジトリから要求されたコンテンツを提供する各ミラーリポジトリの個別のエントリー。



注記

ImageContentSourcePolicy オブジェクトを持つクラスターのグローバルプルシークレットのみを設定できます。プロジェクトにプルシークレットを追加することはできません。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. ミラーリングされたりポジトリを設定します。以下のいずれかを実行します。
 - [Repository Mirroring in Red Hat Quay](#) で説明されているように、Red Hat Quay でミラーリングされたりポジトリを設定します。Red Hat Quay を使用すると、あるリポジトリから別のリポジトリにイメージをコピーでき、これらのリポジトリを一定期間繰り返し自動的に同期することもできます。
 - **skopeo** などのツールを使用して、ソースディレクトリからミラーリングされたりポジトリにイメージを手動でコピーします。
たとえば、Red Hat Enterprise Linux (RHEL 7 または RHEL 8) システムに **skopeo** RPM パッケージをインストールした後、以下の例に示すように **skopeo** コマンドを使用します。

```
$ skopeo copy \
docker://registry.access.redhat.com/ubi8/ubi-
minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
a6 \
docker://example.io/example/ubi-minimal
```

この例では、**example.io** という名前のコンテナイメージレジストリーと **example** という名前のイメージリポジトリがあり、そこに **registry.access.redhat.com** から **ubi8/ubi-**

minimal イメージをコピーします。レジストリーを作成した後、OpenShift Container Platform クラスターを設定して、ソースリポジトリーで作成される要求をミラーリングされたりポジトリーにリダイレクトできます。

- OpenShift Container Platform クラスターにログインします。
- ImageContentSourcePolicy** ファイル (例: **registryrepomirror.yaml**) を作成し、ソースとミラーを固有のレジストリー、およびリポジトリーのペアとイメージのものに置き換えます。

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: ubi8repo
spec:
  repositoryDigestMirrors:
  - mirrors:
    - example.io/example/ubi-minimal ❶
    source: registry.access.redhat.com/ubi8/ubi-minimal ❷
  - mirrors:
    - example.com/example/ubi-minimal
    source: registry.access.redhat.com/ubi8/ubi-minimal
```

- ❶ イメージレジストリーおよびリポジトリーの名前を示します。
- ❷ ミラーリングされているコンテンツが含まれるレジストリーおよびリポジトリーを示します。

- 新しい **ImageContentSourcePolicy** オブジェクトを作成します。

```
$ oc create -f registryrepomirror.yaml
```

ImageContentSourcePolicy オブジェクトが作成されると、新しい設定が各ノードにデプロイされ、クラスターはソースリポジトリーへの要求のためにミラーリングされたりポジトリーの使用を開始します。

- ミラーリングされた設定が適用されていることを確認するには、ノードのいずれかで以下を実行します。
 - ノードの一覧を表示します。

```
$ oc get node
```

出力例

NAME	STATUS	ROLES	AGE	VERSION
ip-10-0-137-44.ec2.internal	Ready	worker	7m	v1.18.3
ip-10-0-138-148.ec2.internal	Ready	master	11m	v1.18.3
ip-10-0-139-122.ec2.internal	Ready	master	11m	v1.18.3
ip-10-0-147-35.ec2.internal	Ready,SchedulingDisabled	worker	7m	v1.18.3
ip-10-0-153-12.ec2.internal	Ready	worker	7m	v1.18.3
ip-10-0-154-10.ec2.internal	Ready	master	11m	v1.18.3

変更が適用されているため、各ワーカーノードのスケジューリングが無効にされていることを確認できます。

- b. デバッグプロセスを開始し、ノードにアクセスします。

```
$ oc debug node/ip-10-0-147-35.ec2.internal
```

出力例

```
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`
```

- c. ノードのファイルにアクセスします。

```
sh-4.2# chroot /host
```

- d. `/etc/containers/registries.conf` ファイルをチェックして、変更が行われたことを確認します。

```
sh-4.2# cat /etc/containers/registries.conf
```

出力例

```
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
[[registry]]
location = "registry.access.redhat.com/ubi8/"
insecure = false
blocked = false
mirror-by-digest-only = true
prefix = ""

[[registry.mirror]]
location = "example.io/example/ubi8-minimal"
insecure = false

[[registry.mirror]]
location = "example.com/example/ubi8-minimal"
insecure = false
```

- e. ソースからノードにイメージダイジェストをプルし、ミラーによって解決されているかどうかを確認します。 **ImageContentSourcePolicy** オブジェクトはイメージダイジェストのみをサポートし、イメージタグはサポートしません。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-
minimal@sha256:5cfbaf45ca96806917830c183e9f37df2e913b187adb32e89fd83fa455eba
a6
```

リポジトリのミラーリングのトラブルシューティング

リポジトリのミラーリング手順が説明どおりに機能しない場合は、リポジトリミラーリングの動作方法についての以下の情報を使用して、問題のトラブルシューティングを行うことができます。

- 最初に機能するミラーは、プルされるイメージを指定するために使用されます。
- メインレジストリーは、他のミラーが機能していない場合にのみ使用されます。
- システムコンテキストによって、**Insecure** フラグがフォールバックとして使用されます。

- **/etc/containers/registries.conf** ファイルの形式が最近変更されました。現在のバージョンはバージョン 2 で、TOML 形式です。

関連情報

グローバルプルシークレットについての詳細は、[グローバルクラスタープルシークレットの更新](#) について参照してください。

第10章 テンプレートの使用

以下のセクションでは、テンプレートの概要と共に、それらを使用し、作成する方法についての概要を説明します。

10.1. テンプレートについて

テンプレートでは、パラメーター化や処理が可能な一連のオブジェクトを記述し、OpenShift Container Platform で作成するためのオブジェクトの一覧を生成します。テンプレートは、サービス、ビルド設定およびデプロイメント設定など、プロジェクト内で作成パーミッションがあるすべてのものを作成するために処理できます。また、テンプレートではラベルのセットを定義して、これをテンプレート内に定義されたすべてのオブジェクトに適用できます。

オブジェクトの一覧は CLI を使用してテンプレートから作成することも、テンプレートがプロジェクトまたはグローバルテンプレートライブラリーにアップロードされている場合、Web コンソールを使用することもできます。

10.2. テンプレートのアップロード

テンプレートを定義する JSON または YAML ファイルがある場合は、この例にあるように、CLI を使用してプロジェクトにテンプレートをアップロードできます。こうすることで、プロジェクトにテンプレートが保存され、対象のプロジェクトに対して適切なアクセス権があるユーザーがこれを繰り返し使用できます。独自のテンプレートの記述については、このトピックで後ほど説明します。

手順

- 現在のプロジェクトのテンプレートライブラリーにテンプレートをアップロードするには、JSON または YAML ファイルを以下のコマンドで渡します。

```
$ oc create -f <filename>
```

- **-n** オプションを使用してプロジェクト名を指定することで、別のプロジェクトにテンプレートをアップロードできます。

```
$ oc create -f <filename> -n <project>
```

テンプレートは、Web コンソールまたは CLI を使用して選択できるようになりました。

10.3. WEB コンソールを使用したアプリケーションの作成

Web コンソールを使用して、テンプレートからアプリケーションを作成することができます。

手順

1. 必要なプロジェクトで **Add to Project** をクリックします。
2. プロジェクト内にあるイメージの一覧またはサービスカタログからビルダーイメージを選択します。



注記

以下に示すように、**builder** タグがアノテーションに一覧表示されているイメージストリームタグのみが一覧に表示されます。

```

kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby"
  creationTimestamp: null
spec:
  dockerImageRepository: "registry.redhat.io/rhscv/ruby-26-rhel7"
  tags:
  -
    name: "2.6"
    annotations:
      description: "Build and run Ruby 2.6 applications"
      iconClass: "icon-ruby"
      tags: "builder,ruby" ❶
      supports: "ruby:2.6,ruby"
      version: "2.6"

```

- ❶ ここに **builder** を含めると、このイメージストリームがビルダーとして Web コンソールに表示されます。

- 新規アプリケーション画面で設定を変更し、オブジェクトをアプリケーションをサポートするように設定します。

10.4. CLI を使用してテンプレートからオブジェクトを作成する手順

CLI を使用して、テンプレートを処理し、オブジェクトを作成するために生成された設定を使用できません。

10.4.1. ラベルの追加

ラベルは、Pod などの生成されたオブジェクトを管理し、整理するために使用されます。テンプレートで指定されるラベルは、テンプレートから生成されるすべてのオブジェクトに適用されます。

手順

- コマンドラインからテンプレートにラベルを追加します。

```
$ oc process -f <filename> -l name=otherLabel
```

10.4.2. パラメーターの一覧表示

上書きできるパラメーターの一覧は、テンプレートの **parameters** セクションに表示されます。

手順

- CLI で以下のコマンドを使用し、使用するファイルを指定して、パラメーターを一覧表示することができます。

```
$ oc process --parameters -f <filename>
```

または、テンプレートがすでにアップロードされている場合には、以下を実行します。

```
$ oc process --parameters -n <project> <template_name>
```

たとえば、デフォルトの **openshift** プロジェクトにあるクイックスタートテンプレートのいずれかに対してパラメーターを一覧表示する場合に、以下のような出力が表示されます。

```
$ oc process --parameters -n openshift rails-postgresql-example
```

出力例

```
NAME                DESCRIPTION
GENERATOR           VALUE
SOURCE_REPOSITORY_URL  The URL of the repository with your application source
code                https://github.com/sclorg/rails-ex.git
SOURCE_REPOSITORY_REF  Set this to a branch name, tag or other ref of your
repository if you are not using the default branch
CONTEXT_DIR           Set this to the relative path to your project if it is not in the root of
your repository
APPLICATION_DOMAIN     The exposed hostname that will route to the Rails service
rails-postgresql-example.openshiftapps.com
GITHUB_WEBHOOK_SECRET A secret string used to configure the GitHub webhook
expression           [a-zA-Z0-9]{40}
SECRET_KEY_BASE       Your secret key for verifying the integrity of signed cookies
expression           [a-z0-9]{127}
APPLICATION_USER       The application user that is used within the sample application
to authorize access on pages           openshift
APPLICATION_PASSWORD   The application password that is used within the sample
application to authorize access on pages           secret
DATABASE_SERVICE_NAME Database service name
postgresql
POSTGRESQL_USER        database username
expression             user[A-Z0-9]{3}
POSTGRESQL_PASSWORD    database password
expression             [a-zA-Z0-9]{8}
POSTGRESQL_DATABASE    database name
root
POSTGRESQL_MAX_CONNECTIONS database max connections
10
POSTGRESQL_SHARED_BUFFERS database shared buffers
12MB
```

この出力から、テンプレートの処理時に正規表現のようなジェネレーターで生成された複数のパラメーターを特定できます。

10.4.3. オブジェクト一覧の生成

CLI を使用して、標準出力にオブジェクト一覧を返すテンプレートを定義するファイルを処理できます。

手順

1. 標準出力にオブジェクト一覧を返すテンプレートを定義するファイルを処理します。

```
$ oc process -f <filename>
```

または、テンプレートがすでに現在のプロジェクトにアップロードされている場合には以下を実行します。

```
$ oc process <template_name>
```

2. テンプレートを処理し、**oc create** の出力をパイプして、テンプレートからオブジェクトを作成します。

```
$ oc process -f <filename> | oc create -f -
```

または、テンプレートがすでに現在のプロジェクトにアップロードされている場合には以下を実行します。

```
$ oc process <template> | oc create -f -
```

3. 上書きする **<name>=<value>** の各ペアに、**-p** オプションを追加することで、ファイルに定義されたパラメーターの値を上書きできます。パラメーター参照は、テンプレートアイテム内のテキストフィールドに表示されます。

たとえば、テンプレートの以下の **POSTGRESQL_USER** および **POSTGRESQL_DATABASE** パラメーターを上書きし、カスタマイズされた環境変数の設定を出力します。

- a. テンプレートからのオブジェクト一覧の作成

```
$ oc process -f my-rails-postgresql \
  -p POSTGRESQL_USER=bob \
  -p POSTGRESQL_DATABASE=mydatabase
```

- b. JSON ファイルは、ファイルにリダイレクトすることも、**oc create** コマンドで処理済みの出力をパイプして、テンプレートをアップロードせずに直接適用することも可能です。

```
$ oc process -f my-rails-postgresql \
  -p POSTGRESQL_USER=bob \
  -p POSTGRESQL_DATABASE=mydatabase \
  | oc create -f -
```

- c. 多数のパラメーターがある場合は、それらをファイルに保存してからそのファイルを **oc process** に渡すことができます。

```
$ cat postgres.env
POSTGRESQL_USER=bob
POSTGRESQL_DATABASE=mydatabase
```

```
$ oc process -f my-rails-postgresql --param-file=postgres.env
```

- d. **--param-file** の引数として **"-"** を使用して、標準入力から環境を読み込むこともできます。

```
$ sed s/bob/alice/ postgres.env | oc process -f my-rails-postgresql --param-file=-
```

10.5. アップロードしたテンプレートの変更

すでにプロジェクトにアップロードされているテンプレートを編集できます。

手順

- すでにアップロードされているテンプレートを変更します。

```
$ oc edit template <template>
```

10.6. インスタントアプリおよびクイックスタートテンプレートの使用

OpenShift Container Platform では、デフォルトで、インスタントアプリとクイックスタートテンプレートを複数提供しており、各種言語で簡単に新規アプリの構築を開始できます。Rails (Ruby)、Django (Python)、Node.js、CakePHP (PHP) および Dancer (Perl) 用のテンプレートを利用できます。クラスター管理者は、これらのテンプレートを利用できるようにデフォルトのグローバル **openshift** プロジェクトにこれらのテンプレートを作成している必要があります。

デフォルトで、テンプレートビルドは必要なアプリケーションコードが含まれる GitHub の公開ソースリポジトリを使用して行われます。

手順

1. 以下のように、利用可能なデフォルトのインスタントアプリとクイックスタートテンプレートを一覧表示できます。

```
$ oc get templates -n openshift
```

2. ソースを変更し、アプリケーションの独自のバージョンをビルドするには、以下を実行します。
 - a. テンプレートのデフォルト **SOURCE_REPOSITORY_URL** パラメーターが参照するリポジトリをフォークします。
 - b. テンプレートから作成する場合には、**SOURCE_REPOSITORY_URL** パラメーターの値を上書きします。デフォルト値ではなく、フォークを指定してください。これにより、テンプレートで作成したビルド設定はアプリケーションコードのフォークを参照するようになり、コードを変更し、アプリケーションを自由に再ビルドできます。



注記

一部のインスタントアプリおよびクイックスタートのテンプレートで、データベースのデプロイメント設定を定義します。テンプレートが定義する設定では、データベースコンテナ用に一時ストレージを使用します。データベース Pod が何らかの理由で再起動されると、データベースの全データが失われてしまうので、これらのテンプレートはデモ目的でのみ使用する必要があります。

10.6.1. クイックスタートのテンプレート

クイックスタートは、OpenShift Container Platform で実行するアプリケーションの基本的なサンプルです。クイックスタートはさまざまな言語やフレームワークが含まれており、サービスのセット、ビルド設定およびデプロイメント設定などで設定されるテンプレートで定義されています。このテンプレートは、必要なイメージやソースリポジトリを参照して、アプリケーションをビルドし、デプロイします。

クイックスタートを確認するには、テンプレートからアプリケーションを作成します。管理者がこれらのテンプレートを OpenShift Container Platform クラスターにすでにインストールしている必要がありますが、その場合には、Web コンソールからこれを簡単に選択できます。

クイックスタートは、アプリケーションのソースコードを含むソースリポジトリを参照します。クイックスタートをカスタマイズするには、リポジトリをフォークし、テンプレートからアプリケーションを作成する時に、デフォルトのソースリポジトリ名をフォークしたリポジトリに置き換えます。これにより、提供されたサンプルのソースではなく、独自のソースコードを使用してビルドが実行されます。ソースリポジトリでコードを更新し、新しいビルドを起動して、デプロイされたアプリケーションで変更が反映されていることを確認できます。

10.6.1.1. Web フレームワーククイックスタートのテンプレート

以下のクイックスタートテンプレートでは、指定のフレームワークおよび言語の基本アプリケーションを提供します。

- CakePHP: PHP Web フレームワーク (MySQL データベースを含む)
- Dancer: Perl Web フレームワーク (MySQL データベースを含む)
- Django: Python Web フレームワーク (PostgreSQL データベースを含む)
- NodeJS: NodeJS web アプリケーション (MongoDB データベースを含む)
- Rails: Ruby Web フレームワーク (PostgreSQL データベースを含む)

10.7. テンプレートの作成

アプリケーションの全オブジェクトを簡単に再作成するために、新規テンプレートを定義できます。テンプレートでは、作成するオブジェクトと、これらのオブジェクトの作成をガイドするメタデータを定義します。

以下は、単純なテンプレートオブジェクト定義 (YAML) の例です。

```
apiVersion: v1
kind: Template
metadata:
  name: redis-template
annotations:
  description: "Description"
  iconClass: "icon-redis"
  tags: "database,nosql"
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: redis-master
  spec:
    containers:
    - env:
      - name: REDIS_PASSWORD
        value: ${REDIS_PASSWORD}
      image: dockerfile/redis
      name: master
      ports:
      - containerPort: 6379
        protocol: TCP
  parameters:
  - description: Password used for Redis authentication
```



```

from: '[A-Z0-9]{8}'
generate: expression
name: REDIS_PASSWORD
labels:
  redis: master

```

10.7.1. テンプレート記述の作成

テンプレートの記述により、テンプレートの内容に関する情報を提供でき、Web コンソールでの検索時に役立ちます。テンプレート名以外のメタデータは任意ですが、使用できると便利です。メタデータには、一般的な説明などの情報以外にタグのセットも含まれます。便利なタグにはテンプレートで使用する言語名などがあります (例: Java、PHP、Ruby)。

以下は、テンプレート記述メタデータの例です。

```

kind: Template
apiVersion: v1
metadata:
  name: cakephp-mysql-example ❶
  annotations:
    openshift.io/display-name: "CakePHP MySQL Example (Ephemeral)" ❷
    description: >-
      An example CakePHP application with a MySQL database. For more information
      about using this template, including OpenShift considerations, see
      https://github.com/sclorg/cakephp-ex/blob/master/README.md.

    WARNING: Any data stored will be lost upon pod destruction. Only use this
    template for testing." ❸
    openshift.io/long-description: >-
      This template defines resources needed to develop a CakePHP application,
      including a build configuration, application DeploymentConfig, and
      database DeploymentConfig. The database is stored in
      non-persistent storage, so this configuration should be used for
      experimental purposes only. ❹
  tags: "quickstart,php,cakephp" ❺
  iconClass: icon-php ❻
  openshift.io/provider-display-name: "Red Hat, Inc." ❼
  openshift.io/documentation-url: "https://github.com/sclorg/cakephp-ex" ❽
  openshift.io/support-url: "https://access.redhat.com" ❾
message: "Your admin credentials are ${ADMIN_USERNAME}:${ADMIN_PASSWORD}" ❿

```

- ❶ テンプレートの一意の名前。
- ❷ ユーザーインターフェイスで利用できるように、ユーザーに分かりやすく、簡単な名前。
- ❸ テンプレートの説明。デプロイされる内容、デプロイ前に知っておく必要のある注意点をユーザーが理解できるように詳細を追加します。README ファイルなど、追加情報へのリンクも追加します。パラグラフを作成するには、改行を追加できます。
- ❹ 追加の説明。たとえば、サービスカタログに表示されます。
- ❺

検索およびグループ化を実行するためにテンプレートに関連付けられるタグ。これを指定されるカタログカテゴリのいずれかに組み込むタグを追加します。コンソールの定数ファイルの

6 Web コンソールでテンプレートと一緒に表示されるアイコン。

例10.1 利用可能なアイコン

- **icon-3scale**
- **icon-aerogear**
- **icon-amq**
- **icon-angularjs**
- **icon-ansible**
- **icon-apache**
- **icon-beaker**
- **icon-camel**
- **icon-capedwarf**
- **icon-cassandra**
- **icon-catalog-icon**
- **icon-clojure**
- **icon-codeigniter**
- **icon-cordova**
- **icon-datagrid**
- **icon-datavirt**
- **icon-debian**
- **icon-decisionserver**
- **icon-django**
- **icon-dotnet**
- **icon-drupal**
- **icon-eap**
- **icon-elastic**
- **icon-erlang**
- **icon-fedora**
- **icon-freebsd**

- **icon-git**
- **icon-github**
- **icon-gitlab**
- **icon-glassfish**
- **icon-go-gopher**
- **icon-golang**
- **icon-grails**
- **icon-hadoop**
- **icon-haproxy**
- **icon-helm**
- **icon-infinispan**
- **icon-jboss**
- **icon-jenkins**
- **icon-jetty**
- **icon-joomla**
- **icon-jruby**
- **icon-js**
- **icon-knative**
- **icon-kubevirt**
- **icon-laravel**
- **icon-load-balancer**
- **icon-mariadb**
- **icon-mediawiki**
- **icon-memcached**
- **icon-mongodb**
- **icon-mssql**
- **icon-mysql-database**
- **icon-nginx**
- **icon-nodejs**

- **icon-openjdk**
- **icon-openliberty**
- **icon-openshift**
- **icon-openstack**
- **icon-other-linux**
- **icon-other-unknown**
- **icon-perl**
- **icon-phalcon**
- **icon-php**
- **icon-play**
- **iconpostgresql**
- **icon-processserver**
- **icon-python**
- **icon-quarkus**
- **icon-rabbitmq**
- **icon-rails**
- **icon-redhat**
- **icon-redis**
- **icon-rh-integration**
- **icon-rh-spring-boot**
- **icon-rh-tomcat**
- **icon-ruby**
- **icon-scala**
- **icon-serverlessfx**
- **icon-shadowman**
- **icon-spring-boot**
- **icon-spring**
- **icon-sso**
- **icon-stackoverflow**

- **icon-suse**
- **icon-symfony**
- **icon-tomcat**
- **icon-ubuntu**
- **icon-vertx**
- **icon-wildfly**
- **icon-windows**
- **icon-wordpress**
- **icon-xamarin**
- **icon-zend**

- 7 テンプレートを提供する人または組織の名前
- 8 テンプレートに関する他のドキュメントを参照する URL
- 9 テンプレートに関するサポートを取得できる URL
- 10 テンプレートがインスタンス化された時に表示される説明メッセージ。このフィールドで、新規作成されたリソースの使用方法をユーザーに通知します。生成された認証情報や他のパラメーターを出力に追加できるように、メッセージの表示前にパラメーターの置換が行われます。ユーザーが従うべき次の手順が記載されたドキュメントへのリンクを追加してください。

10.7.2. テンプレートラベルの作成

テンプレートにはラベルのセットを追加できます。これらのラベルは、テンプレートがインスタンス化される時に作成されるオブジェクトごとに追加します。このようにラベルを定義すると、特定のテンプレートから作成された全オブジェクトの検索、管理が簡単になります。

以下は、テンプレートオブジェクトのラベルの例です。

```
kind: "Template"
apiVersion: "v1"
...
labels:
  template: "cakephp-mysql-example" ❶
  app: "${NAME}" ❷
```

- ❶ このテンプレートから作成する全オブジェクトに適用されるラベル
- ❷ パラメーター化されたラベル。このラベルは、このテンプレートを基に作成された全オブジェクトに適用されます。パラメーターは、ラベルキーおよび値の両方で拡張されます。

10.7.3. テンプレートパラメーターの作成

パラメーターにより、テンプレートがインスタンス化される時に値を生成するか、ユーザーが値を指定できるようになります。パラメーターが参照されると、値が置換されます。参照は、オブジェクト一覧フィールドであればどこでも定義できます。これは、無作為にパスワードを作成したり、テンプレートのカスタマイズに必要なユーザー固有の値やホスト名を指定したりできるので便利です。パラメーターは、2種類の方法で参照可能です。

- 文字列の値として、テンプレートの文字列フィールドに `${PARAMETER_NAME}` の形式で配置する
- JSON/YAML の値として、テンプレートのフィールドに `${{PARAMETER_NAME}}` の形式で配置する

`${PARAMETER_NAME}` 構文を使用すると、複数のパラメーター参照を1つのフィールドに統合でき、`"http://${PARAMETER_1}${PARAMETER_2}"` などのように、参照を固定データ内に埋め込むことができます。どちらのパラメーター値も置換されて、引用された文字列が最終的な値になります。

`${PARAMETER_NAME}` 構文のみを使用する場合は、単一のパラメーター参照のみが許可され、先頭文字や終了文字は使用できません。結果の値は、置換後に結果が有効な JSON オブジェクトの場合は引用されません。結果が有効な JSON 値でない場合に、結果の値は引用され、標準の文字列として処理されます。

単一のパラメーターは、テンプレート内で複数回参照でき、1つのテンプレート内で両方の置換構文を使用して参照することができます。

デフォルト値を指定でき、ユーザーが別の値を指定していない場合に使用されます。

以下は、明示的な値をデフォルト値として設定する例です。

```
parameters:
- name: USERNAME
  description: "The user name for Joe"
  value: joe
```

パラメーター値は、パラメーター定義に指定したルールを基に生成することも可能です。以下は、パラメーター値の生成例です。

```
parameters:
- name: PASSWORD
  description: "The random user password"
  generate: expression
  from: "[a-zA-Z0-9]{12}"
```

上記の例では、処理後に、英字の大文字、小文字、数字すべてを含む 12 文字長のパスワードが無作為に作成されます。

利用可能な構文は、完全な正規表現構文ではありません。ただし、`\w`、`\d`、`\a`、および `\A` 修飾子を使用できます。

- `[w]{10}` は、10 桁の英字、数字、およびアンダースコアを生成します。これは PCRE 標準に準拠し、`[a-zA-Z0-9_]{10}` に相当します。
- `[d]{10}` は 10 桁の数字を生成します。これは `[0-9]{10}` に相当します。
- `[a]{10}` は 10 桁の英字を生成します。これは `[a-zA-Z]{10}` に相当します。

- `[A]{10}` は 10 の句読点または記号文字を生成します。これは `[~!@#$%^&*()-_+={}\|<,>.?/";:']{10}` に相当します。

注記

テンプレートが YAML または JSON で記述されているかどうか、また修飾子が組み込まれている文字列のタイプによっては、2 番目のバックスラッシュでバックスラッシュをエスケープする必要がある場合があります。以下は例になります。

修飾子を含む YAML テンプレートの例

```
parameters:
- name: singlequoted_example
  generate: expression
  from: '[A]{10}'
- name: doublequoted_example
  generate: expression
  from: "[\A]{10}"
```

修飾子を含む JSON テンプレートの例

```
{
  "parameters": [
    {
      "name": "json_example",
      "generate": "expression",
      "from": "[\A]{10}"
    }
  ]
}
```

以下は、パラメーター定義と参照を含む完全なテンプレートの例です。

```
kind: Template
apiVersion: v1
metadata:
  name: my-template
objects:
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: cakephp-mysql-example
  annotations:
    description: Defines how to build the application
  spec:
    source:
      type: Git
      git:
        uri: "${SOURCE_REPOSITORY_URL}" 1
        ref: "${SOURCE_REPOSITORY_REF}"
        contextDir: "${CONTEXT_DIR}"
- kind: DeploymentConfig
  apiVersion: v1
  metadata:
    name: frontend
```

```

spec:
  replicas: "${REPLICA_COUNT}" 2
parameters:
- name: SOURCE_REPOSITORY_URL 3
  displayName: Source Repository URL 4
  description: The URL of the repository with your application source code 5
  value: https://github.com/sclorg/cakephp-ex.git 6
  required: true 7
- name: GITHUB_WEBHOOK_SECRET
  description: A secret string used to configure the GitHub webhook
  generate: expression 8
  from: "[a-zA-Z0-9]{40}" 9
- name: REPLICA_COUNT
  description: Number of replicas to run
  value: "2"
  required: true
message: "... The GitHub webhook secret is ${GITHUB_WEBHOOK_SECRET} ..." 10

```

- 1 この値は、テンプレートがインスタンス化された時点で **SOURCE_REPOSITORY_URL** パラメーターに置き換えられます。
- 2 この値は、テンプレートがインスタンス化された時点で、**REPLICA_COUNT** パラメーターの引用なしの値に置き換えられます。
- 3 パラメーター名。この値は、テンプレート内でパラメーターを参照するのに使用します。
- 4 分かりやすいパラメーターの名前。これは、ユーザーに表示されます。
- 5 パラメーターの説明。期待値に対する制約など、パラメーターの目的を詳細にわたり説明します。説明には、コンソールのテキスト標準に従い、完結した文章を使用するようにしてください。表示名と同じ内容を使用しないでください。
- 6 テンプレートをインスタンス化する時に、ユーザーにより値が上書きされない場合に使用されるパラメーターのデフォルト値。パスワードなどのデフォルト値の使用を避けるようにしてください。シークレットと組み合わせた生成パラメーターを使用するようにしてください。
- 7 このパラメーターが必須であることを示します。つまり、ユーザーは空の値で上書きできません。パラメーターでデフォルト値または生成値が指定されていない場合には、ユーザーは値を指定する必要があります。
- 8 値が生成されるパラメーター
- 9 ジェネレーターへの入力。この場合、ジェネレーターは、大文字、小文字を含む 40 桁の英数字の値を生成します。
- 10 パラメーターはテンプレートメッセージに含めることができます。これにより、生成された値がユーザーに通知されます。

10.7.4. テンプレートオブジェクト一覧の作成

テンプレートの主な部分は、テンプレートがインスタンス化される時に作成されるオブジェクトの一覧です。これには、ビルド設定、デプロイメント設定、またはサービスなどの有効な API オブジェクトを使用できます。オブジェクトはここで定義された通りに作成され、パラメーターの値は作成前に置換されます。これらのオブジェクトの定義では、以前に定義したパラメーターを参照できます。

以下は、オブジェクト一覧の例です。

```
kind: "Template"
apiVersion: "v1"
metadata:
  name: my-template
objects:
- kind: "Service" 1
  apiVersion: "v1"
  metadata:
    name: "cakephp-mysql-example"
    annotations:
      description: "Exposes and load balances the application pods"
  spec:
    ports:
      - name: "web"
        port: 8080
        targetPort: 8080
    selector:
      name: "cakephp-mysql-example"
```

1 サービスの定義。このテンプレートにより作成されます。



注記

オブジェクト定義のメタデータに **namespace** フィールドの固定値が含まれる場合、フィールドはテンプレートのインスタンス化の際に定義から取り除かれます。**namespace** フィールドにパラメーター参照が含まれる場合には、通常のパラメーター置換が行われ、パラメーターの置換による値の解決が実行された namespace で、オブジェクトが作成されます。この場合、ユーザーは対象の namespace でオブジェクトを作成するパーミッションがあることが前提になります。

10.7.5. テンプレートをバインド可能としてマーキングする

テンプレートサービスブローカーは、認識されているテンプレートオブジェクトごとに、カタログ内にサービスを1つ公開します。デフォルトでは、これらのサービスはそれぞれバインド可能として公開され、エンドユーザーがプロビジョニングしたサービスに対してバインドできるようにします。

手順

テンプレートの作成者は、エンドユーザーが指定テンプレートからプロビジョニングされたサービスに対してバインディングすることを防ぐことができます。

- **template.openshift.io/bindable: "false"** のアノテーションをテンプレートに追加して、エンドユーザーが指定のテンプレートからプロビジョニングされるサービスをバインドできないようにできます。

10.7.6. テンプレートオブジェクトフィールドの公開

テンプレートの作成者は、テンプレートに含まれる特定のオブジェクトのフィールドを公開すべきかどうかを指定できます。テンプレートサービスブローカーは、**ConfigMap**、**Secret**、**Service**、および **Route** オブジェクトに公開されたフィールドを認識し、ユーザーがブローカーでサポートされているサービスをバインドする際に公開されたフィールドの値を返します。

オブジェクトのフィールドを1つまたは複数公開するには、テンプレート内のオブジェクトに、接頭辞が **template.openshift.io/expose-** または **template.openshift.io/base64-expose-** のアノテーションを追加します。

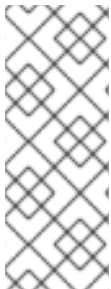
各アノテーションキーは、**bind** 応答のキーになるように、接頭辞が削除されてパススルーされます。

各アノテーションの値は Kubernetes JSONPath 式の値であり、バインド時に解決され、**bind** 応答で返される値が含まれるオブジェクトフィールドを指定します。



注記

Bind 応答のキーと値のペアは、環境変数として、システムの他の場所で使用できます。そのため、アノテーションキーで接頭辞を取り除いた値を有効な環境変数名として使用することが推奨されます。先頭に **A-Z**、**a-z** または **_** を指定して、その後に、ゼロか、他の文字 **A-Z**、**a-z**、**0-9** または **_** を指定してください。



注記

バックスラッシュでエスケープしない限り、Kubernetes の JSONPath 実装は表現内のどの場所に使用されていても、**.**、**@** などはメタ文字として解釈します。そのため、たとえば、**my.key** という名前の **ConfigMap** のデータを参照するには、JSONPath 式は **{.data['my\\.key']}** とする必要があります。JSONPath 式が YAML でどのように記述されているかによって、**"{.data['my\\.key']}"** などのように、追加でバックスラッシュが必要になる場合があります。

以下は、公開されるさまざまなオブジェクトのフィールドの例です。

```
kind: Template
apiVersion: v1
metadata:
  name: my-template
objects:
- kind: ConfigMap
  apiVersion: v1
  metadata:
    name: my-template-config
    annotations:
      template.openshift.io/expose-username: "{.data['my\\.username']}"
  data:
    my.username: foo
- kind: Secret
  apiVersion: v1
  metadata:
    name: my-template-config-secret
    annotations:
      template.openshift.io/base64-expose-password: "{.data['password']}"
  stringData:
    password: bar
- kind: Service
  apiVersion: v1
  metadata:
    name: my-template-service
    annotations:
      template.openshift.io/expose-service_ip_port: "{.spec.clusterIP};{.spec.ports[?
(.name==\"web\")].port}"
```

```
spec:
  ports:
    - name: "web"
      port: 8080
  - kind: Route
    apiVersion: v1
    metadata:
      name: my-template-route
    annotations:
      template.openshift.io/expose-uri: "http://{.spec.host}{.spec.path}"
spec:
  path: mypath
```

上記の部分的なテンプレートでの **bind** 操作に対する応答例は以下のようになります。

```
{
  "credentials": {
    "username": "foo",
    "password": "YmFy",
    "service_ip_port": "172.30.12.34:8080",
    "uri": "http://route-test.router.default.svc.cluster.local/mypath"
  }
}
```

手順

- **template.openshift.io/expose-** アノテーションを使用して、値を文字列として返します。これは、任意のバイナリーデータを処理しないものの、便利な方法です。
- バイナリーデータを返す必要がある場合、**template.openshift.io/base64-expose-** アノテーションを使用して、データが返される前にデータを base64 でエンコードします。

10.7.7. テンプレートの準備ができるまで待機する

テンプレートの作成者は、テンプレート内の特定のオブジェクトがサービスカタログ、Template Service Broker または **TemplateInstance** API によるテンプレートのインスタンス化が完了したとされるまで待機する必要があるかを指定できます。

この機能を使用するには、テンプレート内の **Build**、**BuildConfig**、**Deployment**、**DeploymentConfig**、**Job** または **StatefulSet** のオブジェクト 1 つ以上に、次のアノテーションでマークを付けてください。

```
"template.alpha.openshift.io/wait-for-ready": "true"
```

テンプレートのインスタンス化は、アノテーションのマークが付けられたすべてのオブジェクトが準備できたと報告されるまで、完了しません。同様に、アノテーションが付けられたオブジェクトが失敗したと報告されるか、固定タイムアウトである 1 時間以内にテンプレートの準備が整わなかった場合に、テンプレートのインスタンス化は失敗します。

インスタンス化の目的で、各オブジェクトの種類の準備状態および失敗は以下のように定義されます。

種類	準備状態 (Readiness)	失敗 (Failure)
Build	オブジェクトが Complete (完了) フェーズを報告する	オブジェクトが Canceled (キャンセル)、Error (エラー)、または Failed (失敗) を報告する
BuildConfig	関連付けられた最新のビルドオブジェクトが Complete (完了) フェーズを報告する	関連付けられた最新のビルドオブジェクトが Canceled (キャンセル)、Error (エラー)、または Failed (失敗) を報告する
Deployment	オブジェクトは、新しいレプリカセットとデプロイメントが利用可能であると報告する。これにより、オブジェクトで定義される readiness プロブが有効になります。	オブジェクトで、Progressing (進捗中) の状態が false であると報告される
DeploymentConfig	オブジェクトは新規レプリケーションコントローラーおよびデプロイメントが利用可能であると報告する。これにより、オブジェクトで定義される readiness プロブが有効になります。	オブジェクトで、Progressing (進捗中) の状態が false であると報告される
Job	オブジェクトが完了 (completion) を報告する	オブジェクトが1つ以上の失敗が発生したことを報告する
StatefulSet	オブジェクトはすべてのレプリカが Ready (準備状態) であることを報告する。これにより、オブジェクトで定義される readiness プロブが有効になります。	該当なし

以下は、テンプレートサンプルを一部抜粋したものです。この例では、**wait-for-ready** アノテーションが使用されています。他のサンプルは、OpenShift Container Platform クイックスタートテンプレートにあります。

```
kind: Template
apiVersion: v1
metadata:
  name: my-template
objects:
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: ...
  annotations:
    # wait-for-ready used on BuildConfig ensures that template instantiation
    # will fail immediately if build fails
    template.alpha.openshift.io/wait-for-ready: "true"
  spec:
    ...
- kind: DeploymentConfig
```

```

apiVersion: v1
metadata:
  name: ...
  annotations:
    template.alpha.openshift.io/wait-for-ready: "true"
spec:
  ...
- kind: Service
  apiVersion: v1
  metadata:
    name: ...
  spec:
    ...

```

その他の推奨事項

- アプリケーションにスムーズに実行するのに十分なリソースが提供されるようにメモリー、CPU、およびストレージのデフォルトサイズを設定します。
- **latest** タグが複数のメジャーバージョンで使用されている場合には、イメージからこのタグを参照しないようにします。新規イメージがそのタグにプッシュされると、実行中のアプリケーションが破損してしまう可能性があります。
- 適切なテンプレートの場合、テンプレートのデプロイ後に変更する必要なしに、ビルドおよびデプロイが正常に行われます。

10.7.8. 既存オブジェクトからのテンプレートの作成

テンプレートをゼロから作成するのではなく、プロジェクトから既存のオブジェクトをYAML形式でエクスポートして、パラメーターを追加したり、テンプレート形式としてカスタマイズしたりして、YAML形式を変更することもできます。

手順

- オブジェクトをYAML形式でプロジェクトにエクスポートします。

```
$ oc get -o yaml --export all > <yaml_filename>
```

all ではなく、特定のリソースタイプや複数のリソースを置き換えることも可能です。他の例については、**oc get -h** を実行してください。

以下は、**oc get --export all** に含まれるオブジェクトタイプです。

- **BuildConfig**
- **Build**
- **DeploymentConfig**
- **ImageStream**
- **Pod**
- **ReplicationController**

- **Route**
- **Service**

第11章 RUBY ON RAILS の使用

Ruby on Rails は Ruby で記述される Web フレームワークです。本書では、OpenShift Container Platform での Rails 4 の使用について扱います。



警告

チュートリアル全体をチェックして、OpenShift Container Platform でアプリケーションを実行するために必要なすべての手順を概観してください。問題に直面した場合には、チュートリアル全体を振り返り、もう一度問題に対応してください。またチュートリアルは、実行済みの手順を確認し、すべての手順が適切に実行されていることを確認するのに役立ちます。

11.1. 前提条件

- Ruby および Rails の基本知識
- Ruby 2.0.0+、Rubygems、Bundler のローカルにインストールされたバージョン
- Git の基本知識
- OpenShift Container Platform v4 の実行インスタンス
- OpenShift Container Platform のインスタンスが実行中であり、利用可能であることを確認してください。さらに、**oc** CLI クライアントがインストールされており、コマンドがコマンドシェルからアクセスできることを確認し、メールアドレスおよびパスワードを使用してログインする際にこれを使用できるようにします。

11.2. データベースの設定

Rails アプリケーションはほぼ常にデータベースと併用されます。ローカル開発の場合は、PostgreSQL データベースを使用します。

手順

1. データベースをインストールします。

```
$ sudo yum install -y postgresql postgresql-server postgresql-devel
```

2. データベースを初期化します。

```
$ sudo postgresql-setup initdb
```

このコマンドで **/var/lib/pgsql/data** ディレクトリーが作成され、このディレクトリーにデータが保存されます。

3. データベースを起動します。

```
$ sudo systemctl start postgresql.service
```

4. データベースが実行されたら、**rails** ユーザーを作成します。

```
$ sudo -u postgres createuser -s rails
```

作成をしたユーザーのパスワードは作成されていない点に留意してください。

11.3. アプリケーションの作成

Rails アプリケーションをゼロからビルドするには、Rails gem を先にインストールする必要があります。その後、アプリケーションを作成することができます。

手順

1. Rails gem をインストールします。

```
$ gem install rails
```

出力例

```
Successfully installed rails-4.3.0  
1 gem installed
```

2. Rails gem のインストール後に、PostgreSQL をデータベースとして指定して新規アプリケーションを作成します。

```
$ rails new rails-app --database=postgresql
```

3. 新規アプリケーションディレクトリーに切り替えます。

```
$ cd rails-app
```

4. アプリケーションがすでにある場合には **pg** (postgresql) gem が **Gemfile** に配置されていることを確認します。配置されていない場合には、gem を追加して **Gemfile** を編集します。

```
gem 'pg'
```

5. すべての依存関係を含む **Gemfile.lock** を新たに生成します。

```
$ bundle install
```

6. **pg** gem で **postgresql** データベースを使用するほか、**config/database.yml** が **postgresql** アダプターを使用していることを確認する必要があります。

config/database.yml ファイルの **default** セクションを以下のように更新するようにしてください。

```
default: &default  
  adapter: postgresql  
  encoding: unicode  
  pool: 5  
  host: localhost  
  username: rails  
  password:
```


7. アプリケーションの開発およびテスト用のデータベースを作成します。

```
$ rake db:create
```

これで PostgreSQL サーバーに **development** および **test** データベースが作成されます。

11.3.1. Welcome ページの作成

Rails 4 では静的な **public/index.html** ページが実稼働環境で提供されなくなったので、新たに root ページを作成する必要があります。

Welcome ページをカスタマイズするには、以下の手順を実行する必要があります。

- index アクションでコントローラーを作成します。
- welcome コントローラーの index アクションの view ページを作成します。
- 作成したコントローラーとビューと共にアプリケーションの root ページを提供するルートを作成します。

Rails には、これらの必要な手順をすべて実行するジェネレーターがあります。

手順

1. Rails ジェネレーターを実行します。

```
$ rails generate controller welcome index
```

すべての必要なファイルが作成されます。

2. 以下のように **config/routes.rb** ファイルの 2 行目を編集します。

```
root 'welcome#index'
```

3. rails server を実行して、ページが利用できることを確認します。

```
$ rails server
```

ブラウザで <http://localhost:3000> に移動してページを表示してください。このページが表示されない場合は、サーバーに出力されるログを確認してデバッグを行ってください。

11.3.2. OpenShift Container Platform のアプリケーションの設定

アプリケーションが OpenShift Container Platform で実行中の PostgreSQL データベースサービスと通信できるようにするには、後のデータベースサービスの作成時に定義する必要がある環境変数を使用できるように **config/database.yml** の **default** セクションを編集する必要があります。

手順

- 以下のように事前に定義した変数で、**config/database.yml** の **default** セクションを編集します。

config/database YAML ファイルのサンプル

```

<% user = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ? "root" :
ENV["POSTGRESQL_USER"] %>
<% password = ENV.key?("POSTGRESQL_ADMIN_PASSWORD") ?
ENV["POSTGRESQL_ADMIN_PASSWORD"] : ENV["POSTGRESQL_PASSWORD"] %>
<% db_service = ENV.fetch("DATABASE_SERVICE_NAME", "").upcase %>

default: &default
  adapter: postgresql
  encoding: unicode
  # For details on connection pooling, see rails configuration guide
  # http://guides.rubyonrails.org/configuring.html#database-pooling
  pool: <%= ENV["POSTGRESQL_MAX_CONNECTIONS"] || 5 %>
  username: <%= user %>
  password: <%= password %>
  host: <%= ENV["#{db_service}_SERVICE_HOST"] %>
  port: <%= ENV["#{db_service}_SERVICE_PORT"] %>
  database: <%= ENV["POSTGRESQL_DATABASE"] %>

```

11.3.3. アプリケーションの Git への保存

通常 OpenShift Container Platform でアプリケーションをビルドする場合、ソースコードを git リポジトリに保存する必要があるため、**git** がない場合にはインストールしてください。

前提条件

- git をインストールします。

手順

1. **ls -l** コマンドを実行して、Rails アプリケーションのディレクトリーで操作を行っていることを確認します。コマンドの出力は以下のようになります。

```
$ ls -l
```

出力例

```

app
bin
config
config.ru
db
Gemfile
Gemfile.lock
lib
log
public
Rakefile
README.rdoc
test
tmp
vendor

```

2. Rails app ディレクトリーで以下のコマンドを実行して、コードを初期化し、git にコミットします。

```
$ git init
```

```
$ git add .
```

```
$ git commit -m "initial commit"
```

アプリケーションがコミットされたら、これをリモートリポジトリにプッシュする必要があります。新規リポジトリを作成する GitHub アカウントです。

3. お使いの **git** リポジトリを参照するリモートを設定します。

```
$ git remote add origin git@github.com:<namespace/repository-name>.git
```

4. アプリケーションをリモートの git リポジトリにプッシュします。

```
$ git push
```

11.4. アプリケーションの OPENSIFT CONTAINER PLATFORM へのデプロイ

OpenShift Container Platform にアプリケーションをデプロイすることができます。

rails-app プロジェクトの作成後、新規プロジェクトの namespace に自動的に切り替えられます。

OpenShift Container Platform へのアプリケーションのデプロイでは 3 つの手順を実行します。

- OpenShift Container Platform の PostgreSQL イメージからデータベースサービスを作成します。
- データベースサービスと連動する OpenShift Container Platform の Ruby 2.0 ビルダイメージおよび Ruby on Rails ソースコードのフロントエンドサービスを作成します。
- アプリケーションのルートを作成します。

手順

- Ruby on Rails アプリケーションをデプロイするには、アプリケーション用に新規のプロジェクトを作成します。

```
$ oc new-project rails-app --description="My Rails application" --display-name="Rails Application"
```

11.4.1. データベースサービスの作成

Rails アプリケーションには実行中のデータベースサービスが必要です。このサービスには、PostgreSQL データベースイメージを使用します。

データベースサービスを作成するために、**oc new-app** コマンドを使用します。このコマンドには、必要な環境変数を渡す必要があります。この環境変数は、データベースコンテナ内で使用します。これらの環境変数は、ユーザー名、パスワード、およびデータベースの名前を設定するために必要です。これらの環境変数の値を任意の値に変更できます。変数は以下のようになります。

- POSTGRESQL_DATABASE
- POSTGRESQL_USER
- POSTGRESQL_PASSWORD

これらの変数を設定すると、以下を確認できます。

- 指定の名前のデータベースが存在する
- 指定の名前のユーザーが存在する
- ユーザーは指定のパスワードで指定のデータベースにアクセスできる

手順

1. データベースサービスを作成します。

```
$ oc new-app postgresql -e POSTGRESQL_DATABASE=db_name -e  
POSTGRESQL_USER=username -e POSTGRESQL_PASSWORD=password
```

データベース管理者のパスワードを設定するには、直前のコマンドに以下を追加します。

```
-e POSTGRESQL_ADMIN_PASSWORD=admin_pw
```

2. 進行状況を確認します。

```
$ oc get pods --watch
```

11.4.2. フロントエンドサービスの作成

アプリケーションを OpenShift Container Platform にデプロイするには、アプリケーションが置かれるリポジトリを指定する必要があります。

手順

1. フロントエンドサービスを作成し、データベースサービスの作成時に設定されたデータベース関連の環境変数を指定します。

```
$ oc new-app path/to/source/code --name=rails-app -e POSTGRESQL_USER=username -e  
POSTGRESQL_PASSWORD=password -e POSTGRESQL_DATABASE=db_name -e  
DATABASE_SERVICE_NAME=postgresql
```

このコマンドでは、OpenShift Container Platform は指定された環境変数を使用してソースコードの取得、ビルダーのセットアップ、アプリケーションイメージのビルド、新規に作成されたイメージのデプロイを実行します。このアプリケーションには **rails-app** という名前を指定します。

2. **rails-app** デプロイメント設定の JSON ドキュメントを参照して、環境変数が追加されたかどうかを確認できます。

```
$ oc get dc rails-app -o json
```

以下のセクションが表示されるはずですが。

3. 出力例

```
env": [
  {
    "name": "POSTGRESQL_USER",
    "value": "username"
  },
  {
    "name": "POSTGRESQL_PASSWORD",
    "value": "password"
  },
  {
    "name": "POSTGRESQL_DATABASE",
    "value": "db_name"
  },
  {
    "name": "DATABASE_SERVICE_NAME",
    "value": "postgresql"
  }
],
```

1. ビルドプロセスを確認します。

```
$ oc logs -f build/rails-app-1
```

2. ビルドが完了すると、OpenShift Container Platform で Pod が実行されていることを確認します。

```
$ oc get pods
```

myapp-<number>-<hash> で始まる行が表示されますが、これは OpenShift Container Platform で実行中のアプリケーションです。

3. データベースの移行スクリプトを実行してデータベースを初期化してからでないと、アプリケーションは機能しません。これを実行する 2 種類の方法があります。

- 実行中のフロントエンドコンテナーから手動で実行する
 - **rsh** コマンドでフロントエンドコンテナーに **exec** を実行します。

```
$ oc rsh <frontend_pod_id>
```

- コンテナー内から移行を実行します。

```
$ RAILS_ENV=production bundle exec rake db:migrate
```

development または **test** 環境で Rails アプリケーションを実行する場合には、**RAILS_ENV** の環境変数を指定する必要はありません。

- デプロイメント前のライフサイクルフックをテンプレートに追する

11.4.3. アプリケーションのルートを作成

アプリケーションのルートを作成するためにサービスを公開できます。

手順

- **www.example.com** などの外部からアクセスできるホスト名を指定してサービスを公開するには、OpenShift Container Platform のルートを使用します。この場合は、以下を入力してフロントエンドサービスを公開する必要があります。

```
$ oc expose service rails-app --hostname=www.example.com
```



警告

指定するホスト名がルーターの IP アドレスに解決することを確認します。

第12章 イメージの使用

12.1. イメージの使用の概要

以下のトピックを使用して、OpenShift Container Platform ユーザーに提供されているさまざまな Source-to-Image (S2I)、データベース、その他のコンテナイメージを確認します。

Red Hat の公式コンテナイメージは、registry.redhat.io の Red Hat レジストリーで提供されています。OpenShift Container Platform がサポートする S2I、データベース、Jenkins イメージは、Red Hat Quay レジストリーの **openshift4** リポジトリにあります。たとえば、**quay.io/openshift-release-dev/ocp-v4.0-<address>** は OpenShift Application Platform イメージの名前です。

xPaaS ミドルウェアイメージは、Red Hat レジストリーの適切な製品リポジトリで提供されていますが、接尾辞として **-openshift** が付いています。たとえば、**registry.redhat.io/jboss-eap-6/eap64-openshift** は JBoss EAP イメージの名前です。

このセクションで説明する Red Hat がサポートするイメージはすべて [Red Hat Ecosystem Catalog のコンテナイメージのセクション](#) に記載されています。各イメージのすべてのバージョンについて、そのコンテンツや用途の詳細を確認できます。関連するイメージを参照または検索してください。



重要

コンテナイメージの新しいバージョンは、OpenShift Container Platform の以前のバージョンとは互換性がありません。お使いの OpenShift Container Platform のバージョンに基づいて、正しいバージョンのコンテナイメージを確認し、使用するようにしてください。

12.2. JENKINS イメージの設定

OpenShift Container Platform には、Jenkins 実行用のコンテナイメージがあります。このイメージには Jenkins サーバーインスタンスが含まれており、このインスタンスを使用して継続的なテスト、統合、デリバリーの基本フローを設定することができます。

イメージは、Red Hat Universal Base Images (UBI) に基づいています。

OpenShift Container Platform は、Jenkins の [LTS](#) リリースに従います。OpenShift Container Platform は、Jenkins 2.x を含むイメージを提供します。

OpenShift Container Platform Jenkins イメージは、[Quay.io](https://quay.io) または registry.redhat.io で利用できます。

以下に例を示します。

```
$ podman pull registry.redhat.io/openshift4/ose-jenkins:<v4.3.0>
```

これらのイメージを使用するには、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナイメージレジストリーにプッシュできます。さらに、コンテナイメージレジストリーまたは外部の場所で、対象イメージを参照するイメージストリームを作成することもできます。その後、OpenShift Container Platform リソースがイメージストリームを参照できます。

ただし便宜上、OpenShift Container Platform はコア Jenkins イメージの **openshift** namespace にイメージストリームを提供するほか、OpenShift Container Platform を Jenkins と統合するために提供されるエージェントイメージのサンプルも提供します。

12.2.1. 設定とカスタマイズ

Jenkins 認証は、以下の 2 つの方法で管理できます。

- OpenShift Container Platform ログインプラグインが提供する OpenShift Container Platform OAuth 認証
- Jenkins が提供する標準認証。

12.2.1.1. OpenShift Container Platform OAuth 認証

OAuth 認証は、Jenkins UI の **Configure Global Security** パネルでオプションを設定するか、Jenkins デプロイメント設定の **OPENSIFT_ENABLE_OAUTH** 環境変数を **false** 以外に設定して、有効にします。これにより、OpenShift Container Platform ログインプラグインが有効になり、Pod データからか、または OpenShift Container Platform API サーバーと対話して設定情報を取得します。

有効な認証情報は、OpenShift Container Platform アイデンティティプロバイダーが制御します。

Jenkins はブラウザおよびブラウザ以外のアクセスの両方をサポートします。

OpenShift Container Platform ロールでユーザーに割り当てられる固有の Jenkins パーミッションが指定されている場合、有効なユーザーは、ログイン時に自動的に Jenkins 認証マトリックスに追加されます。デフォルトで使用されるロールは、事前に定義される **admin**、**edit**、および **view** です。ログインプラグインは、Jenkins が実行されているプロジェクトまたは namespace のそれらのロールに対して自己 SAR 要求 (self-SAR request) を実行します。

admin ロールを持つユーザーには、従来の Jenkins 管理ユーザーパーミッションがあります。ユーザーのパーミッションは、ロールが **edit**、**view** になるほど少なくなります。

OpenShift Container Platform のデフォルトの **admin**、**edit**、**view** のロール、これらのロールが Jenkins インスタンスに割り当てられている Jenkins パーミッションは設定可能です。

OpenShift Container Platform Pod で Jenkins を実行する場合、ログインプラグインは Jenkins が実行されている namespace で **openshift-jenkins-login-plugin-config** という名前の設定マップを検索します。

このプラグインが検出し、その設定マップで読み取り可能な場合には、ロールを Jenkins パーミッションマッピングに定義できます。具体的には以下を実行します。

- ログインプラグインは、設定マップのキーと値のペアを OpenShift Container Platform のロールのマッピングに対する Jenkins パーミッションとして処理します。
- キーは Jenkins パーミッショングループの短い ID と Jenkins パーミッションの短い ID で、この 2 つはハイフンで区切られています。
- OpenShift Container Platform ロールに **Overall Jenkins Administer** パーミッションを追加する場合は、キーは **Overall-Administer** である必要があります。
- パーミッショングループおよびパーミッション ID が利用可能かどうかを把握するには、Jenkins コンソールのマトリックス認証ページに移動し、グループの ID とグループが提供するテーブルの個々のパーミッションを確認します。
- キーと値ペアの値は、パーミッションが適用される必要がある OpenShift Container Platform ロールの一覧で、各ロールはコマンドで区切られています。

- **Overall Jenkins Administer** パーミッションをデフォルトの **admin** および **edit** ロールの両方に追加し、作成した新規の jenkins ロールも追加する場合は、キーの **Overall-Administer** の値は **admin,edit,jenkins** になります。



注記

OpenShift Container Platform OAuth が使用されている場合、管理者権限で OpenShift Container Platform Jenkins イメージに事前に設定されている **admin** ユーザーには、これらの権限は割り当てられません。これらのパーミッションを付与するには、OpenShift Container Platform クラスター管理者は OpenShift Container Platform アイデンティティプロバイダーでそのユーザーを明示的に定義し、**admin** ロールをユーザーに割り当てる必要があります。

保存される Jenkins ユーザーのパーミッションは、初回のユーザー作成後に変更できます。OpenShift Container Platform ログインプラグインは、OpenShift Container Platform API サーバーをポーリングしてパーミッションを取得し、ユーザーごとに Jenkins に保存されているパーミッションを、OpenShift Container Platform から取得したパーミッションに更新します。Jenkins UI を使用して Jenkins ユーザーのパーミッションを更新する場合には、プラグインが次回に OpenShift Container Platform をポーリングするタイミングで、パーミッションの変更が上書きされます。

ポーリングの頻度は **OPENSIFT_PERMISSIONS_POLL_INTERVAL** 環境変数で制御できます。デフォルトのポーリングの間隔は 5 分です。

OAuth 認証を使用して新しい Jenkins サービスを作成するには、テンプレートを使用するのが最も簡単な方法です。

12.2.1.2. Jenkins 認証

テンプレートを使用せず、イメージが直接実行される場合には、デフォルトで Jenkins 認証が使用されます。

Jenkins の初回起動時には、設定、管理ユーザーおよびパスワードが作成されます。デフォルトのユーザー認証情報は、**admin** と **password** です。標準の Jenkins 認証を使用する場合は、**JENKINS_PASSWORD** 環境変数を設定してデフォルトのパスワードを設定します。

手順

- 標準の Jenkins 認証を使用する Jenkins アプリケーションを作成します。

```
$ oc new-app -e \
  JENKINS_PASSWORD=<password> \
  openshift4/ose-jenkins
```

12.2.2. Jenkins 環境変数

Jenkins サーバーは、以下の環境変数で設定できます。

変数	定義	値と設定の例
----	----	--------

変数	定義	値と設定の例
OPENSIFT_ENABLE_OAUTH	Jenkins へのログイン時に OpenShift Container Platform ログインプラグインが認証を管理するかどうかを決定します。有効にするには、 true に設定します。	デフォルト: false
JENKINS_PASSWORD	標準の Jenkins 認証を使用する際の admin ユーザーのパスワード。 OPENSIFT_ENABLE_OAUTH が true に設定されている場合には該当しません。	デフォルト: password
JAVA_MAX_HEAP_PARAM、CONTAINER_HEAP_PERCENT、JENKINS_MAX_HEAP_UPPER_BOUND_MB	これらの値は Jenkins JVM の最大ヒープサイズを制御します。 JAVA_MAX_HEAP_PARAM が設定されている場合は、その値が優先されます。設定されていない場合は、最大ヒープサイズは、コンテナメモリー制限の CONTAINER_HEAP_PERCENT として動的に計算され、オプションで JENKINS_MAX_HEAP_UPPER_BOUND_MB MiB を上限とします。 デフォルトでは Jenkins JVM の最大ヒープサイズは、上限なしでコンテナメモリー制限の 50% に設定されます。	JAVA_MAX_HEAP_PARAM の設定例: -Xmx512m CONTAINER_HEAP_PERCENT のデフォルト: 0.5 (50%) JENKINS_MAX_HEAP_UPPER_BOUND_MB の設定例: 512 MiB
JAVA_INITIAL_HEAP_PARAM、CONTAINER_INITIAL_PERCENT	これらの値は Jenkins JVM の初期ヒープサイズを制御します。 JAVA_INITIAL_HEAP_PARAM が設定されている場合は、その値が優先されます。設定されていない場合は、初期ヒープサイズは、動的に計算される最大ヒープサイズの CONTAINER_INITIAL_PERCENT として動的に計算されません。 デフォルトでは、JVM は初期のヒープサイズを設定します。	JAVA_INITIAL_HEAP_PARAM の設定例: -Xmx32m CONTAINER_INITIAL_PERCENT の設定例: 0.1 (10%)
CONTAINER_CORE_LIMIT	設定されている場合には、内部の JVM スレッドのサイジング数に使用するコアの数を整数で指定します。	設定例: 2

変数	定義	値と設定の例
JAVA_TOOL_OPTIONS	このコンテナで実行中のすべての JVM に適用するオプションを指定します。この値の上書きは推奨していません。	デフォルト: - XX:+UnlockExperimentalVM Options - XX:+UseCGroupMemoryLimitForHeap - Dsun.zip.disableMemoryMapping=true
JAVA_GC_OPTS	Jenkins JVM ガーベージコレクションのパラメーターを指定します。この値の上書きは推奨していません。	デフォルト: - XX:+UseParallelGC - XX:MinHeapFreeRatio=5 - XX:MaxHeapFreeRatio=10 - XX:GCTimeRatio=4 - XX:AdaptiveSizePolicyWeight=90
JENKINS_JAVA_OVERRIDES	Jenkins JVM の追加オプションを指定します。これらのオプションは、上記の Java オプションなど、その他すべてのオプションに追加され、必要に応じてそれらの値のいずれかを上書きするのに使用できます。追加オプションがある場合には、スペースで区切ります。オプションにスペース文字が含まれる場合には、バックスラッシュでエスケープしてください。	設定例: -Dfoo -Dbar;- Dfoo=first\ value - Dbar=second\ value
JENKINS_OPTS	Jenkins への引数を指定します。	
INSTALL_PLUGINS	コンテナが初めて実行された場合や、 OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS が true に設定されている場合に、インストールする追加の Jenkins プラグインを指定します。プラグインは、名前: バージョンのペアのコンマ区切りの一覧で指定されます。	設定例: git:3.7.0,subversion:2.10.2
OPENSIFT_PERMISSIONS_POLL_INTERVAL	OpenShift Container Platform ログインプラグインが Jenkins に定義されているユーザーごとに関連付けられたパーミッションを OpenShift Container Platform でポーリングする間隔をミリ秒単位で指定します。	デフォルト: 300000 - 5 分

変数	定義	値と設定の例
OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG	Jenkins 設定ディレクトリー用に OpenShift Container Platform 永続ボリューム (PV) を使用してこのイメージを実行する場合、PV は永続ボリューム要求 (PVC) の作成時に割り当てられるため、イメージから PV に設定が転送されるのは、イメージの初回起動時のみです。このイメージを拡張し、初回起動後にカスタムイメージの設定を更新するカスタムイメージを作成する場合、この環境変数を true に設定しない限り、設定はコピーされません。	デフォルト: false
OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS	Jenkins 設定ディレクトリー用に OpenShift Container Platform PV を使用してこのイメージを実行する場合、PV は PVC の作成時に割り当てられるため、イメージから PV にプラグインが転送されるのは、イメージの初回起動時のみです。このイメージを拡張し、初回起動後にカスタムイメージのプラグインを更新するカスタムイメージを作成する場合、この環境変数を true に設定しない限り、プラグインはコピーされません。	デフォルト: false
ENABLE_FATAL_ERROR_LOG_FILE	Jenkins 設定ディレクトリー用に OpenShift Container Platform PVC を使用してこのイメージを実行する場合に、この環境変数は致命的なエラーが生じる際に致命的なエラーのログファイルが永続することを可能にします。致命的なエラーのファイルは /var/lib/jenkins/logs に保存されます。	デフォルト: false
NODEJS_SLAVE_IMAGE	この値を設定すると、デフォルトの Node.js エージェント Pod 設定に使用されるイメージが上書きされます。 jenkins-agent-nodejs という名前の関連するイメージストリームタグがプロジェクトにあります。この変数は、有効にするために Jenkins の初回の起動前に設定される必要があります。	Jenkins サーバーのデフォルトの Node.js エージェントイメージ: image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-nodejs:latest

トークンパラメーターには、Jenkins がプロジェクトにアクセスするために必要とするトークンの値が含まれます。

12.2.4. Jenkins のボリューム間のマウントポイント

Jenkins イメージはマウントしたボリュームで実行して、設定用に永続ストレージを有効にできます。

- **/var/lib/jenkins** - Jenkins がジョブ定義などの設定ファイルを保存するデータディレクトリーです。

12.2.5. S2I (Source-To-Image) による Jenkins イメージのカスタマイズ

正式な OpenShift Container Platform Jenkins イメージをカスタマイズするには、イメージを Source-To-Image (S2I) ビルダーとしてイメージを使用できます。

S2I を使用して、カスタムの Jenkins ジョブ定義をコピーしたり、プラグインを追加したり、同梱の **config.xml** ファイルを独自のカスタムの設定に置き換えたりできます。

Jenkins イメージに変更を追加するには、以下のディレクトリー構造の Git リポジトリーが必要です。

plugins

このディレクトリーには、Jenkins にコピーするバイナリーの Jenkins プラグインを含めます。

plugins.txt

このファイルは、以下の構文を使用して、インストールするプラグインを一覧表示します。

```
pluginId:pluginVersion
```

configuration/jobs

このディレクトリーには、Jenkins ジョブ定義が含まれます。

configuration/config.xml

このファイルには、カスタムの Jenkins 設定が含まれます。

configuration/ディレクトリーのコンテンツは **/var/lib/jenkins/**ディレクトリーにコピーされるので、このディレクトリーに **credentials.xml** などのファイルをさらに追加することもできます。

ビルド設定のサンプルは、OpenShift Container Platform で Jenkins イメージをカスタマイズします。

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source: 1
    git:
      uri: https://github.com/custom/repository
      type: Git
  strategy: 2
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jenkins:2
        namespace: openshift
```

```

type: Source
output:
  to:
    kind: ImageStreamTag
    name: custom-jenkins:latest

```

3

- 1 **source** パラメーターは、上記のレイアウトでソースの Git リポジトリを定義します。
- 2 **strategy** パラメーターは、ビルドのソースイメージとして使用するための元の Jenkins イメージを定義します。
- 3 **output** パラメーターは、結果として生成される、カスタマイズした Jenkins イメージを定義します。これは、公式の Jenkins イメージの代わりに、デプロイメント設定で使用できます。

12.2.6. Jenkins Kubernetes プラグインの設定

OpenShift Container Platform Jenkins イメージには、事前にインストール済みの [Kubernetes プラグイン](#) が含まれ、Kubernetes および OpenShift Container Platform を使用して、Jenkins エージェントを複数のコンテナホストで動的にプロビジョニングできるようにします。

OpenShift Container Platform は、Kubernetes プラグインを使用するために、Jenkins エージェントとして使用するのに適したイメージ (Base、Maven、および Node.js イメージ) を提供します。

Maven および Node.js のエージェントイメージは、Kubernetes プラグイン用の OpenShift Container Platform Jenkins イメージの設定内で、Kubernetes Pod テンプレートイメージとして自動的に設定されます。この設定にはイメージごとのラベルが含まれており、Restrict where this project can be run の設定にある Jenkins ジョブのいずれかに適用できます。ラベルが適用されている場合、ジョブはそれぞれのエージェントイメージを実行する OpenShift Container Platform Pod の下で実行されます。

Jenkins イメージは、Kubernetes プラグインの追加のエージェントイメージの自動検出および自動設定を実行します。

OpenShift Container Platform 同期プラグインでは、Jenkins イメージは、Jenkins の起動時に、実行中のプロジェクトまたはプラグインの設定に具体的に一覧表示されているプロジェクト内で以下を検索します。

- ラベル **role** が **jenkins-agent** に設定されているイメージストリーム
- アノテーション **role** が **jenkins-agent** に設定されているイメージストリーム
- ラベル **role** が **jenkins-agent** に設定されている設定マップ

適切なラベルまたは、適切なアノテーションが付いたイメージストリームタグが見つかったら、適切な Kubernetes プラグイン設定が生成され、イメージストリーム提供のコンテナイメージを実行する Pod で、Jenkins ジョブを実行するように割り当てることができます。

イメージストリームまたはイメージストリームタグのイメージ参照および名前が、Kubernetes プラグインの Pod テンプレートにある名前およびイメージフィールドにマッピングされます。Kubernetes プラグインの Pod テンプレートのラベルフィールドは、イメージストリームにアノテーションを設定するか、イメージストリームタグオブジェクトに **agent-label** キーを設定して制御できます。これらを使用しない場合には、名前をラベルとして使用します。



注記

Jenkins コンソールにログインして、Pod テンプレート設定を変更しないでください。Pod テンプレートが作成された後にこれを行い、OpenShift Container Platform 同期プラグインがイメージストリームまたはイメージストリームタグに関連付けられたイメージが変更されたことを検知した場合、Pod テンプレートを置き換え、これらの設定変更を上書きします。新しい設定を既存の設定とマージすることはできません。

より複雑な設定が必要な場合は、設定マップを使用する方法を検討してください。

適切なラベルが指定された設定マップが検出される場合は、設定マップのキーと値のデータペイロードの値に、Jenkins および Kubernetes プラグインの Pod テンプレートの設定形式に準拠する XML (Extensible Markup Language) が含まれることが想定されます。設定マップを使用時に注意すべき主な違いは、イメージストリームまたはイメージストリームタグではなく、Kubernetes プラグインの Pod テンプレートのパラメーターすべてを制御できます。

jenkins-agent の設定マップの例:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
labels:
  role: jenkins-agent
data:
  template1: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
    <inheritFrom></inheritFrom>
    <name>template1</name>
    <instanceCap>2147483647</instanceCap>
    <idleMinutes>0</idleMinutes>
    <label>template1</label>
    <serviceAccount>jenkins</serviceAccount>
    <nodeSelector></nodeSelector>
    <volumes/>
    <containers>
    <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
    <name>jnlp</name>
    <image>openshift/jenkins-agent-maven-35-centos7:v3.10</image>
    <privileged>>false</privileged>
    <alwaysPullImage>>true</alwaysPullImage>
    <workingDir>/tmp</workingDir>
    <command></command>
    <args>${computer.jnlpMac} ${computer.name}</args>
    <ttyEnabled>>false</ttyEnabled>
    <resourceRequestCpu></resourceRequestCpu>
    <resourceRequestMemory></resourceRequestMemory>
    <resourceLimitCpu></resourceLimitCpu>
    <resourceLimitMemory></resourceLimitMemory>
    <envVars/>
    </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
    </containers>
    <envVars/>
    <annotations/>
```



```
<imagePullSecrets/>
<nodeProperties/>
</org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

注記

Jenkins コンソールにログインして、Pod テンプレートの作成後に Pod テンプレート設定をさらに変更し、OpenShift Container Platform 同期プラグインが設定マップが変更されたことを検知すると、これは Pod テンプレートを置き換え、この設定変更を上書きします。新しい設定を既存の設定とマージすることはできません。

Jenkins コンソールにログインして、Pod テンプレート設定を変更しないでください。Pod テンプレートが作成された後にこれを行い、OpenShift Container Platform 同期プラグインがイメージストリームまたはイメージストリームタグに関連付けられたイメージが変更されたことを検知した場合、Pod テンプレートを置き換え、これらの設定変更を上書きします。新しい設定を既存の設定とマージすることはできません。

より複雑な設定が必要な場合は、設定マップを使用する方法を検討してください。

インストールされた後、OpenShift Container Platform 同期プラグインは、イメージストリーム、イメージストリームタグ、および設定マップに更新がないか、OpenShift Container Platform の API サーバーをモニターリングして、Kubernetes プラグインの設定を調整します。

以下のルールが適用されます。

- 設定マップ、イメージストリーム、またはイメージストリームタグからラベルまたはアノテーションを削除すると、既存の **PodTemplate** が Kubernetes プラグインの設定から削除されます。
- これらのオブジェクトが削除されると、対応する設定が Kubernetes プラグインから削除されます。
- 適切なラベルおよびアノテーションが付いた **ConfigMap**、**ImageStream**、または **ImageStreamTag** オブジェクトを作成するか、初回作成後にラベルを追加すると、Kubernetes プラグイン設定に **PodTemplate** が作成されます。
- 設定マップフォームの **PodTemplate** の場合には、**PodTemplate** の設定マップデータへの変更は、Kubernetes プラグイン設定の **PodTemplate** 設定に適用され、設定マップに変更を加えてから次に変更を加えるまでの間に、Jenkins UI で加えた **PodTemplate** の変更が上書きされます。

Jenkins エージェントとしてコンテナイメージを使用するには、イメージは、エントリーポイントとしてエージェントを実行する必要があります。これに関する詳細情報は、公式の [Jenkins ドキュメント](#) を参照してください。

12.2.7. Jenkins パーミッション

設定マップで、Pod テンプレート XML の **<serviceAccount>** 要素が結果として作成される Pod に使用される OpenShift Container Platform サービスアカウントである場合、サービスアカウントの認証情報が Pod にマウントされます。パーミッションはサービスアカウントに関連付けられ、OpenShift Container Platform マスターに対するどの操作が Pod から許可されるかについて制御します。

Pod に使用されるサービスアカウントについて以下のシナリオを考慮してください。この Pod は、OpenShift Container Platform Jenkins イメージで実行される Kubernetes プラグインによって起動されます。

OpenShift Container Platform で提供される Jenkins のテンプレートサンプルを使用する場合には、**jenkins** サービスアカウントが、Jenkins が実行するプロジェクトの **edit** ロールで定義され、マスター Jenkins Pod にサービスアカウントがマウントされます。

Jenkins 設定に挿入される 2 つのデフォルトの Maven および NodeJS Pod テンプレートは、Jenkins マスターと同じサービスアカウントを使用するように設定されます。

- イメージストリームまたはイメージストリームタグに必要なラベルまたはアノテーションがあるために OpenShift Container Platform 同期プラグインで自動的に検出されるすべての Pod テンプレートは、Jenkins マスターのサービスアカウントをサービスアカウントとして使用するよう設定されます。
- Pod テンプレートの定義を Jenkins と Kubernetes プラグインに渡す他の方法として、使用するサービスアカウントを明示的に指定する必要があります。他の方法には、Jenkins コンソール、Kubernetes プラグインで提供される **podTemplate** パイプライン DSL、または Pod テンプレートの XML 設定をデータとする設定マップのラベル付けなどが含まれます。
- サービスアカウントの値を指定しない場合には、**default** のサービスアカウントが使用されます。
- 使用するサービスアカウントが何であっても、必要なパーミッション、ロールなどを OpenShift Container Platform 内で定義して、Pod から操作するプロジェクトをすべて操作できるようにする必要があります。

12.2.8. テンプレートからの Jenkins サービスの作成

テンプレートには各種パラメーターフィールドがあり、事前定義されたデフォルト値ですべての環境変数を定義できます。OpenShift Container Platform では、新規の Jenkins サービスを簡単に作成できるようにテンプレートが提供されています。Jenkins テンプレートは、クラスター管理者が、クラスターの初期設定時に、デフォルトの **openshift** プロジェクトに登録する必要があります。

使用可能な 2 つのテンプレートは共にデプロイメント設定とサービスを定義します。テンプレートはストレージストラテジーに応じて異なります。これは、Jenkins コンテンツが Pod の再起動時に永続するかどうかに影響を与えます。



注記

Pod は、別のノードへの移動時や、デプロイメント設定の更新で再デプロイメントがトリガーされた時に再起動される可能性があります。

- **jenkins-ephemeral** は一時ストレージを使用します。Pod が再起動すると、すべてのデータが失われます。このテンプレートは、開発またはテストにのみ役立ちます。
- **jenkins-persistent** は永続ボリューム (PV) ストアを使用します。データは Pod が再起動されても保持されます。

PV ストアを使用するには、クラスター管理者は OpenShift Container Platform デプロイメントで PV プールを定義する必要があります。

必要なテンプレートを選択したら、テンプレートをインスタンス化して Jenkins を使用できるようにする必要があります。

手順

1. 以下の方法のいずれかを使用して、新しい Jenkins アプリケーションを作成します。

- PV:

```
$ oc new-app jenkins-persistent
```

- または、Pod の再起動で設定が維持されない **emptyDir** タイプボリューム:

```
$ oc new-app jenkins-ephemeral
```

12.2.9. Jenkins Kubernetes プラグインの使用

以下の例では、**openshift-jee-sample BuildConfig** オブジェクトにより、Jenkins Maven エージェント Pod が動的にプロビジョニングされます。Pod は Java ソースコードをクローンし、WAR ファイルを作成して、2 番目の **BuildConfig**、**openshift-jee-sample-docker** を実行します。2 番目の **BuildConfig** は、新しい WAR ファイルをコンテナイメージに階層化します。

Jenkins Kubernetes プラグインを使用した BuildConfig の例

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample-docker
  spec:
    strategy:
      type: Docker
    source:
      type: Docker
      dockerfile: |-
        FROM openshift/wildfly-101-centos7:latest
        COPY ROOT.war /wildfly/standalone/deployments/ROOT.war
        CMD $STI_SCRIPTS_PATH/run
    binary:
      asFile: ROOT.war
    output:
      to:
        kind: ImageStreamTag
        name: openshift-jee-sample:latest
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
  spec:
    strategy:
      type: JenkinsPipeline
      jenkinsPipelineStrategy:
        jenkinsfile: |-
          node("maven") {
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
```

```

    sh "mvn -B -Popenshift package"
    sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
  }
triggers:
- type: ConfigChange

```

動的に作成された Jenkins エージェント Pod の仕様を上書きすることも可能です。以下は、コンテナメモリーを上書きして、環境変数を指定するように先の例を変更しています。

Jenkins Kubernetes プラグインを使用した BuildConfig の例 (メモリー制限および環境変数の指定)

```

kind: BuildConfig
apiVersion: v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", ❶
          cloud: "openshift", ❷
          inheritFrom: "maven", ❸
          containers: [
            containerTemplate(name: "jnlp", ❹
              image: "openshift/jenkins-agent-maven-35-centos7:v3.10", ❺
              resourceRequestMemory: "512Mi", ❻
              resourceLimitMemory: "512Mi", ❼
              envVars: [
                envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25") ❽
              ]
            ]
          ) {
            node("mypod") { ❾
              sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
              sh "mvn -B -Popenshift package"
              sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
            }
          }
      }
triggers:
- type: ConfigChange

```

- ❶ **mypod** という名前の新しい Pod テンプレートが動的に定義されます。この新しい Pod テンプレート名はノードのスタンザで参照されます。
- ❷ **cloud** の値は **openshift** に設定する必要があります。
- ❸ 新しい Pod テンプレートは、既存の Pod テンプレートから設定を継承できます。この場合、OpenShift Container Platform で事前定義された Maven Pod テンプレートから継承されます。
- ❹ この例では、既存のコンテナの値を上書きし、名前で指定する必要があります。OpenShift Container Platform に含まれる Jenkins エージェントイメージはすべて、コンテナ名として **jnlp** を使用します。

- 5 再びコンテナイメージ名を指定します。これは既知の問題です。
- 6 **512 Mi** のメモリー要求を指定します。
- 7 **512 Mi** のメモリー制限を指定します。
- 8 環境変数 **CONTAINER_HEAP_PERCENT** に値 **0.25** を指定します。
- 9 ノードスタンプは、定義された Pod テンプレート名を参照します。

デフォルトで、Pod はビルドの完了時に削除されます。この動作は、プラグインを使用して、またはパイプライン Jenkinsfile 内で変更できます。

12.2.10. Jenkins メモリーの要件

提供される Jenkins の一時また永続テンプレートでデプロイする場合にはデフォルトのメモリー制限は **1 Gi** です。

デフォルトで、Jenkins コンテナで実行する他のすべてのプロセスは、合計の **512 MiB** を超えるメモリーを使用することはできません。メモリーがさらに必要になると、コンテナは停止します。そのため、パイプラインが可能な場合に、エージェントコンテナで外部コマンドを実行することを強く推奨されます。

また、**Project** クォータがこれを許可する場合は、Jenkins マスターがメモリーの観点から必要とするものについて、Jenkins ドキュメントの推奨事項を参照してください。この推奨事項では、Jenkins マスターにさらにメモリーを割り当てることを禁止しています。

Jenkins Kubernetes プラグインによって作成されるエージェントコンテナで、メモリー要求および制限の値を指定することが推奨されます。管理者ユーザーは、Jenkins 設定を使用して、エージェントのイメージごとにデフォルト値を設定できます。メモリー要求および制限パラメーターは、コンテナごとに上書きすることもできます。

Jenkins で利用可能なメモリー量を増やすには、Jenkins の一時テンプレートまたは永続テンプレートをインスタンス化する際に **MEMORY_LIMIT** パラメーターを上書きします。

12.2.11. その他のリソース

- [Red Hat Universal Base Images \(UBI\)](#) の詳細は、[ベースイメージのオプション](#) を参照してください。

12.3. JENKINS エージェント

OpenShift Container Platform では、Jenkins エージェントとして使用するのに適したイメージを3つのイメージ (**Base**、**Maven**、および **Node.js**) を提供します。

1番目は、Jenkins エージェントのベースイメージです。

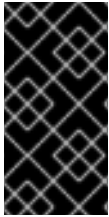
- 必須のツール (ヘッドレス Java、Jenkins JNLP クライアント) と有用なツール (**git**、**tar**、**zip**、**nss** など) の両方を取り入れます。
- JNLP エージェントをエントリーポイントとして確立します。
- Jenkins ジョブ内からコマンドラインの操作を呼び出す **oc** クライアントツールが含まれます。

- Red Hat Enterprise Linux (RHEL) および **localdev** イメージの両方の Dockerfile を提供します。

ベースイメージを拡張するイメージがさらに 2 つ提供されます。

- Maven v3.5 イメージ
- Node.js v10 イメージおよび Node.js v12 イメージ

Maven および Node.js Jenkins エージェントイメージは、新しいエージェントイメージをビルドする際に参照できる Universal Base Image (UBI) 用の Dockerfile を提供します。**contrib** および **contrib/bin** サブディレクトリーにも注目してください。このサブディレクトリーでは、イメージの設定ファイルや実行可能なスクリプトの挿入が可能です。



重要

OpenShift Container Platform のバージョンに適したエージェントイメージバージョンを使用し、継承します。エージェントイメージに埋め込まれた **oc** クライアントバージョンが OpenShift Container Platform バージョンと互換性がない場合、予期しない動作が発生する可能性があります。

12.3.1. Jenkins エージェントイメージ

OpenShift Container Platform Jenkins エージェントイメージは [Quay.io](https://quay.io) または registry.redhat.io で利用できます。

Jenkins イメージは、Red Hat レジストリーから入手できます。

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins:<v4.5.0>
```

```
$ docker pull registry.redhat.io/openshift4/jenkins-agent-nodejs-10-rhel7:<v4.5.0>
```

```
$ docker pull registry.redhat.io/openshift4/jenkins-agent-nodejs-12-rhel7:<v4.5.0>
```

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-maven:<v4.5.0>
```

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-base:<v4.5.0>
```

これらのイメージを使用するには、[Quay.io](https://quay.io) または registry.redhat.io から直接アクセスするか、これらを OpenShift Container Platform コンテナイメージレジストリーにプッシュします。

12.3.2. Jenkins エージェントの環境変数

各 Jenkins エージェントコンテナは、以下の環境変数で設定できます。

変数	定義	値と設定の例
----	----	--------

変数	定義	値と設定の例
JAVA_MAX_HEAP_PARAM、CONTAINER_HEAP_PERCENT、JENKINS_MAX_HEAP_UPPER_BOUND_MB	<p>これらの値は Jenkins JVM の最大ヒープサイズを制御します。JAVA_MAX_HEAP_PARAM が設定されている場合は、その値が優先されます。設定されていない場合は、最大ヒープサイズは、コンテナメモリー制限の CONTAINER_HEAP_PERCENT として動的に計算され、オプションで JENKINS_MAX_HEAP_UPPER_BOUND_MB MiB を上限とします。</p> <p>デフォルトでは Jenkins JVM の最大ヒープサイズは、上限なしでコンテナメモリー制限の 50% に設定されます。</p>	<p>JAVA_MAX_HEAP_PARAM の設定例: -Xmx512m</p> <p>CONTAINER_HEAP_PERCENT のデフォルト: 0.5 (50%)</p> <p>JENKINS_MAX_HEAP_UPPER_BOUND_MB の設定例: 512 MiB</p>
JAVA_INITIAL_HEAP_PARAM、CONTAINER_INITIAL_PERCENT	<p>これらの値は Jenkins JVM の初期ヒープサイズを制御します。JAVA_INITIAL_HEAP_PARAM が設定されている場合は、その値が優先されます。設定されていない場合は、初期ヒープサイズは、動的に計算される最大ヒープサイズの CONTAINER_INITIAL_PERCENT として動的に計算されます。</p> <p>デフォルトでは、JVM は初期のヒープサイズを設定します。</p>	<p>JAVA_INITIAL_HEAP_PARAM の設定例: -Xmx32m</p> <p>CONTAINER_INITIAL_PERCENT の設定例: 0.1 (10%)</p>
CONTAINER_CORE_LIMIT	<p>設定されている場合には、内部の JVM スレッドのサイジング数に使用するコアの数を整数で指定します。</p>	<p>設定例: 2</p>
JAVA_TOOL_OPTIONS	<p>このコンテナで実行中のすべての JVM に適用するオプションを指定します。この値の上書きは推奨していません。</p>	<p>デフォルト: - XX:+UnlockExperimentalVMOptions - XX:+UseCGroupMemoryLimitForHeap - Dsun.zip.disableMemoryMapping=true</p>

変数	定義	値と設定の例
JAVA_GC_OPTS	Jenkins JVM ガーベージコレクションのパラメーターを指定します。この値の上書きは推奨していません。	デフォルト: - XX:+UseParallelGC - XX:MinHeapFreeRatio=5 - XX:MaxHeapFreeRatio=10 - XX:GCTimeRatio=4 - XX:AdaptiveSizePolicyWeight=90
JENKINS_JAVA_OVERRIDES	Jenkins JVM の追加オプションを指定します。これらのオプションは、上記の Java オプションを含む、その他すべてのオプションに追加され、必要に応じてそれらのいずれかを上書きするために使用できます。追加オプションがある場合には、スペースで区切ります。オプションにスペース文字が含まれる場合には、バックslashでエスケープしてください。	設定例: -Dfoo -Dbar; - Dfoo=first\ value - Dbar=second\ value

12.3.3. Jenkins エージェントのメモリー要件

JVM はすべての Jenkins エージェントで使用され、Jenkins JNLP エージェントをホストし、**javac**、Maven、または Gradle などの Java アプリケーションを実行します。

デフォルトで、Jenkins JNLP エージェントの JVM はヒープにコンテナメモリー制限の 50% を使用します。この値は、**CONTAINER_HEAP_PERCENT** の環境変数で変更可能です。上限を指定することも、すべて上書きすることも可能です。

デフォルトでは、シェルスクリプトや **oc** コマンドをパイプラインから実行するなど、Jenkins エージェントコンテナで実行される他のプロセスはすべて、OOM kill を呼び出さずに残りの 50% メモリー制限を超えるメモリーを使用することはできません。

デフォルトでは、Jenkins エージェントコンテナで実行される他の各 JVM プロセスは、最大でコンテナメモリー制限の 25% をヒープに使用します。多くのビルドワークロードにおいて、この制限の調整が必要になる場合があります。

12.3.4. Jenkins エージェントの Gradle ビルド

OpenShift Container Platform の Jenkins エージェントで Gradle ビルドをホストすると、Jenkins JNLP エージェントおよび Gradle JVM に加え、テストが指定されている場合に Gradle が 3 番目の JVM を起動してテストを実行するので、さらに複雑になります。

以下の設定は、OpenShift Container Platform でメモリーに制約がある Jenkins エージェントの Gradle ビルドを実行する場合の開始点として推奨されます。必要に応じて、これらの設定を変更することができます。

- **gradle.properties** ファイルに **org.gradle.daemon=false** を追加して、有効期間の長い (long-lived) Gradle デーモンを無効にします。

- **gradle.properties** ファイルで **org.gradle.parallel=true** が設定されていないこと、また、コマンドラインの引数として **--parallel** が設定されていないことを確認して、並行ビルドの実行を無効にします。
- **java { options.fork = false }** を **build.gradle** ファイルに設定して、プロセス以外で Java がコンパイルされないようにします。
- **build.gradle** ファイルで **test { maxParallelForks = 1 }** が設定されていることを確認して、複数の追加テストプロセスを無効にします。
- **GRADLE_OPTS**、**JAVA_OPTS**、または **JAVA_TOOL_OPTIONS** 環境変数で、Gradle JVM メモリーパラメーターを上書きします。
- **build.gradle** の **maxHeapSize** および **jvmArgs** 設定を定義するか、**-Dorg.gradle.jvmargs** コマンドライン引数を使用して、Gradle テスト JVM に最大ヒープサイズと JVM の引数を設定します。

12.3.5. Jenkins エージェント Pod の保持

Jenkins エージェント Pod は、ビルドが完了するか、または停止された後にデフォルトで削除されます。この動作は、Kubernetes プラグインの Pod の保持設定で変更できます。Pod の保持は、すべての Jenkins ビルドについて各 Pod テンプレートの上書きで設定できます。以下の動作がサポートされます。

- **Always** は、ビルドの結果に関係なくビルド Pod を維持します。
- **Default** はプラグイン値を使用します (Pod テンプレートのみ)。
- **Never** は常に Pod を削除します。
- **On Failure** は Pod がビルド時に失敗した場合に Pod を維持します。

Pod の保持はパイプライン Jenkinsfile で上書きできます。

```
podTemplate(label: "mypod",
  cloud: "openshift",
  inheritFrom: "maven",
  podRetention: onFailure(), ❶
  containers: [
    ...
  ]) {
  node("mypod") {
    ...
  }
}
```

- ❶ **podRetention** に許可される値は、**never()**、**onFailure()**、**always()**、および **default()** です。



警告

保持される Pod は実行し続け、リソースクォータに対してカウントされる可能性があります。

12.4. SOURCE-TO-IMAGE (S2I)

[Red Hat Software Collections](#) イメージを、Node.js、Perl、Python などの特定のランタイム環境に依存するアプリケーションの基盤として使用することができます。これらのランタイムベースイメージの特殊なバージョンは Source-to-image (S2I) イメージと呼ばれています。S2I イメージを使用して、コードを、そのコードを実行できるベースイメージ環境に挿入することができます。

S2I イメージには、以下が含まれます。

- Java
- Node.js
- Perl
- PHP
- Python
- Ruby

S2I イメージは、OpenShift Container Platform Web UI の **Catalog** → **Developer Catalog** を選択することにより直接使用することができます。

S2I イメージは、[Cluster Samples Operator](#) を設定して利用可能にすることもできます。

12.4.1. Source-to-Image ビルドプロセスの概要

Source-to-Image (S2I) は、ソースコードをコンテナに挿入してソースコードを実行可能にすることで、実行可能な状態のイメージを生成します。S2I では、以下の手順を実行します。

1. **FROM <builder image>** コマンドを実行します。
2. ソースコードをビルダーイメージの定義された場所にコピーします。
3. ビルダーイメージから assemble スクリプトを実行します。
4. デフォルトコマンドとしてビルダーイメージに run スクリプトを設定します。

Buildah は次にコンテナイメージを作成します。

12.4.2. 関連情報

- Cluster Samples Operator の使用方法は、[Cluster Samples Operator の設定](#) を参照してください。

- S2I ビルドについての詳細は、[S2I ビルドについてのビルドストラテジーのドキュメント](#) を参照してください。
- S2I プロセスのトラブルシューティングについては、[Source-to-Image \(S2I\) プロセスのトラブルシューティング](#) を参照してください。
- S2I でのイメージの作成の概要については、[source-to-image からのイメージの作成](#) を参照してください。
- S2I イメージのテストの概要については、[S2I イメージのテストについて](#) を参照してください。

12.5. SOURCE-TO-IMAGE イメージのカスタマイズ

Source-to-Image (S2I) ビルダークイイメージには、assemble および run スクリプトが含まれますが、それらのスクリプトのデフォルト動作はすべてのユーザーに適している訳ではありません。デフォルトのスクリプトを含む S2I ビルダークイ動作をカスタマイズできます。

12.5.1. イメージに埋め込まれたスクリプトの呼び出し

ビルダークイイメージは、最も一般的なユースケースを含む、独自のバージョンの source-to-image (S2I) スクリプトを提供します。これらのスクリプトで各自のニーズが満たされない場合に向け、S2I には **.s2i/bin** ディレクトリーにカスタムスクリプトを追加して上書きできる手段があります。ただし、カスタムスクリプトを追加すると、標準スクリプトを完全に置き換えてしまいます。スクリプトの置き換えは許容できる場合もありますが、場合によっては、イメージに含まれるスクリプトのロジックを保持しつつ、スクリプトの前後にコマンドをいくつか実行できる場合があります。標準的なスクリプトを再利用するには、カスタムのロジックを実行し、イメージ内のデフォルトのスクリプトに追加の作業を委任するラッパースクリプトを作成できます。

手順

1. ビルダークイイメージ内のスクリプトの場所を判別するには、**io.openshift.s2i.scripts-url** ラベルの値を確認します。

```
$ podman inspect --format='{{ index .Config.Labels "io.openshift.s2i.scripts-url" }}'
wildfly/wildfly-centos7
```

出力例

```
image:///usr/libexec/s2i
```

wildfly/wildfly-centos7 ビルダークイイメージを検査し、スクリプトが **/usr/libexec/s2i** ディレクトリーにあることを確認できます。

2. 他のコマンドでラップされた標準スクリプトのいずれかの呼び出しを含むスクリプトを作成します。

.s2i/bin/assemble スクリプト

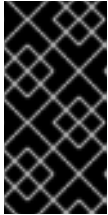
```
#!/bin/bash
echo "Before assembling"

/usr/libexec/s2i/assemble
rc=$?
```

```
if [ $rc -eq 0 ]; then
  echo "After successful assembling"
else
  echo "After failed assembling"
fi

exit $rc
```

以下の例では、メッセージを出力するカスタムの assemble スクリプトを表示し、イメージから標準の assemble スクリプトを実行して、assemble スクリプトの終了コードに応じて別のメッセージを出力します。



重要

run スクリプトをラップする場合には、スクリプトの呼び出しに **exec** を実行して、シグナルが正しく処理されるようにする必要があります。**exec** を使用すると、デフォルトのイメージ実行スクリプトを呼び出した後に追加でコマンドを実行できなくなります。

.s2i/bin/run スクリプト

```
#!/bin/bash
echo "Before running application"
exec /usr/libexec/s2i/run
```