OpenShift Container Platform 4.16

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

Last Updated: 2025-12-06

OpenShift Container Platform 4.16 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

Legal Notice

Abstract

このドキュメントでは、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も説明します。

Table of Contents

第1章 イメージの概要

このセクションでは、OpenShift Container Platform のコンテナー、イメージ、イメージストリームの概念を説明します。この情報は、コンテナー化されたアプリケーションが OpenShift Container Platform でどのように機能するかを理解するうえで役立ちます。

1.1. コンテナー、イメージおよびイメージストリームについて

コンテナー、イメージ、およびイメージストリームは、コンテナー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、コンテナーがコンテナー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。

1.2. イメージ

コンテナーイメージは、1 つのコンテナーを実行するために必要なものをすべて含んでいるバイナリーです。イメージを使用すると、アプリケーションをパッケージ化し、OpenShift Container Platform 内の複数のコンテナーとホストにデプロイすることができます。

コンテナーは、作成時にコンテナー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ーにデプロイし、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。

コンテナーイメージを 作成、管理、および 使用 できます。

1.3. IMAGE REGISTRY

イメージレジストリーは、コンテナーイメージを保存および提供するコンテンツサーバーです。レジストリーを使用すると、外部のソースや OpenShift Container Platform の統合レジストリーからコンテナーイメージにアクセスできます。

レジストリーには、1 つ以上のイメージリポジトリー群が含まれています。各リポジトリーには、1 つ以上のタグ付けされたイメージが含まれています。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。OpenShift Container Platform は、カスタムコンテナーイメージを管理するための独自の OpenShift イメージレジストリーを提供することもできます。

1.4. イメージリポジトリー

イメージリポジトリーは、関連するコンテナーイメージとそれらを識別するタグのコレクションです。イメージリポジトリーを使用すると、OpenShift Container Platform 内の関連するコンテナーイメージを整理および管理できます。

たとえば、OpenShift Container Platform Jenkins イメージは次のリポジトリーにあります。

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

1.5. イメージタグ

イメージタグは、イメージの特定のバージョンを区別するために、イメージストリーム内のコンテナーイメージに適用されるラベルです。OpenShift Container Platform においてイメージの特定のバージョンを整理および参照するのに役立ちます。

通常、イメージタグは何らかのバージョン番号を表します。次の例は、**:v3.11.59-2** というタグが付いた **registry.access.redhat.com/openshift3/jenkins-2-rhel7** というイメージを示しています。

```
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 には **oc tag** コマンドがあります。これは **docker tag** コマンドに似ていますが、イメージを直接操作するのではなく、イメージストリームを操作するものです。

1.6. イメージ ID

イメージ ID は、コンテナーイメージを一意に識別する Secure Hash Algorithm (SHA) コードです。イメージ ID を使用すると、決して変更されない特定のバージョンのイメージをプルできます。

たとえば、次のイメージ ID は **docker.io/openshift/jenkins-2-centos7** イメージのものです。

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

1.7. コンテナー

OpenShift Container Platform アプリケーションの基本的な単位はコンテナーと呼ばれています。Linux コンテナーテクノロジー は、指定されたリソースのみと対話するために実行中のプロセスを分離する軽量なメカニズムです。コンテナーという言葉は、**実行中または一時停止中の状態にある、コンテナーイメージの特定のインスタンス** と定義されます。

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ーで実行される可能性があります。通常、コンテナーは任意のワークロードに使用されますが、各コンテナーは 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ー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 コンテナー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、コンテナー形式およびコンテナー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 形式のコンテナー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ー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ー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、コンテナーを直接実行および管理するための Docker コマンドラインツールの代わりとして使用できます。**podman** CLI を使用すると、OpenShift Container Platform と切り離してコンテナーを試すことができます。

1.8. イメージストリームの使用

イメージストリームは、OpenShift Container Platform 内でコンテナー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を使用すると、イメージのバージョンを管理し、ビルドとデプロイを自動化できます。

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ーと同様に、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きます。

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、コンテナーイメージレジストリーのコンテナー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 または 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、リポジトリーではなく、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ー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セキュリティーを設定できます。
- クラスターレベルでイメージを読み込んだり、リスト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。

イメージストリームを 管理 し、Kubernetes リソースでイメージストリームを使用 し、イメージストリームの更新で更新をトリガー できます。

1.9. 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内のイメージに対する名前付きのポインターです。ユーザーは、特定のバージョンのコンテナーイメージを参照するようにイメージストリームタグを設定できます。

1.10. 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コンテナーイメージを取得する API リソースオブジェクトです。イメージストリームイメージを使用すると、OpenShift Container Platform 内にある特定のイメージの SHA 識別子に関するメタデータにアクセスできます。

1.11. イメージストリームトリガー

イメージストリームトリガーは、イメージストリームタグが変更されたときに特定のアクションを開始するものです。ユーザーは、新しいイメージのインポート時にビルドまたはデプロイを自動的に開始するようにトリガーを設定できます。

たとえば、新しいイメージをインポートすると、タグの値が変更されることがあります。このとき、値の変更を待ち受けているデプロイメントやビルドなどのリソースがあると、トリガーが起動します。

1.12. CLUSTER SAMPLES OPERATOR の使用方法

初期の起動時に、Operator はデフォルトサンプルを作成してイメージストリームおよびテンプレートの作成を開始します。Cluster Samples Operator は、**openshift** namespace に保存されるサンプルイメージストリームおよびテンプレートを管理できます。

クラスター管理者は、Cluster Samples Operator を使用して次のことができます。

- Operator の設定
- 代替レジストリーで Operator の使用

1.13. テンプレートについて

テンプレートは、複製されるオブジェクトの定義です。テンプレート を使用して、設定を構築およびデプロイできます。

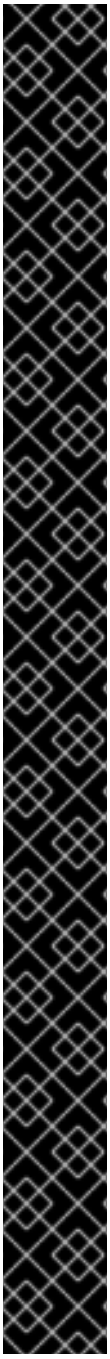
1.14. RUBY ON RAILS の使い方

開発者は、Ruby on Railsを使用して次のことができます。

- アプリケーションの作成:
  - データベースをセットアップする。
  - ウェルカムページを作成します。
  - OpenShift Container Platform 用にアプリケーションを設定します。
  - アプリケーションを Git に保存します。
- OpenShift Container Platform にアプリケーションをデプロイします。
  - データベースサービスを作成します。
  - フロントエンドサービスを作成する。
  - アプリケーションのルートを作成します。

第2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 テンプレートをインストールし、更新します。



重要

- Cluster Samples Operator は非推奨になりました。Cluster Samples Operator には、新しいテンプレート、サンプル、または非 Source-to-Image (非 S2I) イメージストリームは追加されません。ただし今後のリリースで Cluster Samples Operator が削除されるまで、既存の S2I ビルダーイメージストリームとテンプレートは引き続き更新されます。S2I イメージストリームとテンプレートには、次のものが含まれます。
  - Ruby
  - Python
  - Node.js
  - Perl
  - PHP
  - HTTPD
  - Nginx
  - EAP
  - Java
  - Webserver
  - .NET
  - Go
- Cluster Samples Operator は、非 S2I サンプル (イメージストリームとテンプレート) の管理とサポートを停止します。要件や将来の計画は、イメージストリームまたはテンプレートの所有者に問い合わせてください。また、次のリンクも参照してください。
  - イメージストリームまたはテンプレートをホストするリポジトリーのリスト

2.1. 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、クラスター管理者は、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 の設定は、クラスター全体のリソースです。この Operator は、**openshift-cluster-samples-operator** namespace にデプロイされます。

Cluster Samples Operator のイメージには、関連付けられている OpenShift Container Platform リリースのイメージストリームとテンプレート定義が含まれています。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
- クラスターステータスのリソース

サンプルリソースが削除されると、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。

インストール中に Cluster Samples Operator が削除された場合は、別のレジストリーで Cluster Samples Operator を使用することで、コンテンツをインポートできるようになります。その後、Cluster Samples Operator を **Managed** に設定してサンプルを取得できます。以下の手順を使用してください。

- 代替レジストリーでの Cluster Samples Operator の使用

認証情報の設定の詳細は、次のリンクを参照してください。

- イメージプルシークレットの使用

2.2. CLUSTER SAMPLES OPERATOR による管理状態の使用

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、グローバルプロキシーが設定されている場合にブートストラップされます。

Managed 状態では、レジストリーからサンプルイメージストリームとイメージをプルし、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、Cluster Samples Operator がリソースを積極的に管理し、コンポーネントをアクティブな状態に維持します。

次のような特定の状況では、Cluster Samples Operator が **Removed** 状態でブートストラップされます。

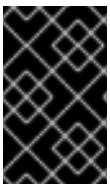
- クリーンインストール後の最初の起動時に、Cluster Samples Operator が 3 分経過してもレジストリーにアクセスできない場合。

- Cluster Samples Operator が IPv6 ネットワーク上にあることが Operator 自身によって検出された場合。
- イメージコントローラーの設定パラメーターにより、デフォルトのイメージレジストリーまたは **samplesRegistry** 設定で指定されたイメージレジストリーを使用してイメージストリームを作成できない場合。詳細は、以下のリンクをご覧ください。
  - イメージコントローラー設定パラメーター
  - Cluster Samples Operator の設定パラメーター

注記

OpenShift Container Platform の場合、デフォルトのイメージレジストリーは **registry.redhat.io** です。

ただし、Cluster Samples Operator が IPv6 ネットワーク上にあることが検出され、かつ OpenShift Container Platform グローバルプロキシーが設定されている場合は、IPv6 チェックがすべてのチェックよりも優先されます。その結果、Cluster Samples Operator はそれ自体を **Removed** としてブートストラップします。



重要

現在、IPv6 インストールはレジストリー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とイメージをレジストリーからプルします。

2.2.1. ネットワークが制限されたインストール

Cluster Samples Operator は、**registry.redhat.io** にアクセスできない場合、自身を **Removed** 状態でブートストラップします。これは、ネットワーク制限がすでに適用されている環境でのインストールを容易にするためです。

Operator が **Removed** 状態でブートストラップされると、クラスター管理者は、サンプルが必要かどうかを判断するための時間をより多く確保できます。管理状態が **Removed** の場合、サンプルイメージストリームのインポートが失敗しているというアラートが Cluster Samples Operator によって送信されないためです。Cluster Samples Operator の管理状態が **Managed** であり、Operator がサンプルイメージストリームをインストールしようとすると、最初のインストールから 2 時間後にインポート失敗のアラートが起動します。

2.2.2. 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

最終的に制限されたネットワーク上で運用されるクラスターであっても、初期インストール時にネットワークアクセスが存在する場合、Cluster Samples Operator は **registry.redhat.io** からコンテンツをインストールします。

この場合、接続環境におけるインストールのデフォルト設定である **Managed** をオーバーライドすることで、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延期することができます。

インストール時にはネットワークアクセスがある環境で Cluster Samples Operator を **Removed** 管理状態でブートストラップする場合は、次の手順を使用して Cluster Samples Operator のデフォルト設定をオーバーライドしてください。

- ノードのカスタマイズ

制限された環境でサンプルをホストするには、次の手順を使用してください。

- 代替レジストリーでの Cluster Samples Operator の使用

また、**openshift-install create manifest** プロセスによって作成された **openshift** ディレクトリーに、次の追加の YAML ファイルを配置する必要があります。

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
```

2.3. CLUSTER SAMPLES OPERATOR によるイメージストリーム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。

インポートが失敗した場合、Cluster Samples Operator は、次のいずれかが発生するまで、約 15 分ごとにイメージストリームイメージインポート API を介してインポートを再試行します。

- インポートが成功する。
- Cluster Samples Operator 設定が変更され、イメージストリームが **skippedImagestreams** リストに追加されるか、管理状態が **Removed** に変更される。

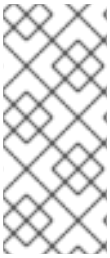
2.3.1. ミラーリングの Cluster Samples Operator のサポート

インストール中に、OpenShift Container Platform は、**openshift-cluster-samples-operator** namespace に **imagestreamtag-to-image** という名前の config map を作成します。

imagestreamtag-to-image config map には、各イメージストリームタグのエントリー (入力されるイメージ) が含まれています。

この config map に含まれるデータフィールドの各エントリーのキーは、**<image_stream_name>_<image_stream_tag_name>** という形式です。

OpenShift Container Platform の非接続インストール時に、Cluster Samples Operator のステータスは **Removed** に設定されます。これを **Managed** に変更することを選択した場合、サンプルがインストールされます。



注記

ネットワークが制限されている環境や非接続環境でサンプルを使用するには、ネットワーク外部のサービスにアクセスする必要がある場合があります。サービスの例には、Github、Maven Central、npm、RubyGems、PyPi などがあります。場合によっては、Cluster Samples Operator オブジェクトが必要なサービスに到達できるようにするために、追加の手順を実行する必要があります。

イメージストリームによるインポート用に、どのイメージをミラーリングする必要があるかを判断する際には、次の原則を使用してください。

- Cluster Samples Operator が **Removed** に設定される場合、ミラーリングされたレジストリーを作成するか、使用する必要のある既存のミラーリングされたレジストリーを判別できます。
- 新しい config map をガイドとして使用し、ミラーリングされたレジストリーに必要なサンプルをミラーリングします。
- Cluster Samples Operator 設定オブジェクトの **skippedImagestreams** リストに、ミラーリングされていないイメージストリームを追加します。
- Cluster Samples Operator 設定オブジェクトの **samplesRegistry** をミラーリングされたレジストリーに設定します。
- 次に、Cluster Samples Operator を **Managed** に設定し、ミラーリングしたイメージストリームをインストールします。

2.4. 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	**Managed**: Cluster Samples Operator は設定の指示に応じてサンプルを更新します。 **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。 **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、省略されたリストにあるサンプルを無視します。除去されると、Cluster Samples Operator は **Unmanaged** 状態にあるかのうように機能し、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。

パラメーター	説明
samplesRegistry	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。  **注記** 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です。 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。
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、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リストを追加できます。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、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。

2.4.1. 設定の制限

Cluster Samples Operator が複数のアーキテクチャーのサポートを開始すると、Operator が **Managed** 状態にある間はアーキテクチャーのリストを変更できなくなります。

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** に切り替えた後にシークレットを作成すると、シークレットイベントが処理されるまでの間、サンプルの作成に遅延が発生します。この仕組みは、レジストリーの変更作業を容易にするものであり、レジストリーを切り替える前にすべてのサンプルを削除し、まっさらな状態を確保する場合に役立ちます。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。

2.4.2. サンプルリソースの条件

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	**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。 **False** すべての生成が一致するか、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リストは reason フィールドに置かれます。 これは OpenShift Container Platform では非推奨にされています。
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	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。 **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リストは 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。
MigrationInProgress	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 Cluster Samples Operator のバージョンと異なることを Cluster Samples Operator が検出した場合、**True** になります。 これは OpenShift Container Platform では非推奨にされています。

2.5. 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**) がインストールされている。

手順

- 次のコマンドを実行して、Cluster Samples Operator 設定にアクセスします。

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

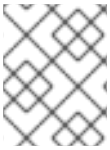
Cluster Samples Operator 設定は以下の例のようになります。

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

2.6. CLUSTER SAMPLES OPERATOR からの非推奨のイメージストリームタグの削除

Cluster Samples Operator は、ユーザーが非推奨のイメージストリームタグを使用するデプロイメントを持っている可能性があるため、非推奨のイメージストリームタグをイメージストリームに残します。

oc tag コマンドでイメージストリームを編集して、非推奨のイメージストリームタグを削除できます。



注記

サンプルプロバイダーがイメージストリームから削除した非推奨のイメージストリームタグは初期インストールに含まれません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

- 次の **oc tag** コマンドを使用してイメージストリームを編集し、非推奨のイメージストリームタグを削除します。

```
$ oc tag -d <image_stream_name:tag>
```

出力例

```
Deleted tag default/<image_stream_name:tag>.
```

第3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 を使用できます。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。

3.1. ミラーレジストリーについて

必要なコンテナー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。代替レジストリーを使用すると、お使いのネットワークとインターネットの両方に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することになります。

OpenShift Container Platform のインストールとその後の製品更新に必要なイメージは、Red Hat Quay、JFrog Artifactory、Sonatype Nexus Repository、Harbor などのコンテナーミラーレジストリーにミラーリングできます。大規模なコンテナーレジストリーにアクセスできない場合は、OpenShift Container Platform サブスクリプションに含まれる小規模なコンテナーレジストリーである **mirror registry for Red Hat OpenShift** を使用できます。

Red Hat Quay、**mirror registry for Red Hat OpenShift**、Artifactory、Sonatype Nexus リポジトリー、Harbor など、Dockerv2-2 をサポートする任意のコンテナーレジストリーを使用できます。選択したレジストリーに関係なく、インターネット上の Red Hat がホストするサイトから分離されたイメージレジストリーにコンテンツをミラーリングする手順は同じです。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。

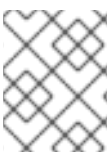
重要

OpenShift イメージレジストリーはターゲットレジストリーとして使用できません。これは、ミラーリングプロセスで必要となるタグを使わないプッシュをサポートしないためです。

mirror registry for Red Hat OpenShift以外のコンテナーレジストリーを選択する場合は、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そのレジストリーにアクセス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。

ミラーリングされたレジストリーの場合は、プルされたイメージのソースを表示するには、CRI-O ログで **Trying to access** のログエントリーを確認する必要があります。ノードで **crictl images** コマンドを使用するなど、イメージのプルソースを表示する他の方法では、イメージがミラーリングされた場所からプルされている場合でも、ミラーリングされていないイメージ名を表示します。



注記

Red Hat は、OpenShift Container Platform を使用してサードパーティーのレジストリーをテストしません。

3.1.1. OpenShift CLI のインストール

OpenShift CLI (**oc**) をインストールすると、コマンドラインインターフェイスから OpenShift Container Platform を操作できます。**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 4.16 のすべてのコマンドを実行することはできません。新しいバージョンの **oc** をダウンロードしてインストールしてください。

3.1.1.1. Linux への OpenShift 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 カスタマーポータルの OpenShift Container Platform ダウンロードページ に移動します。
2. **Product Variant** ドロップダウンリストからアーキテクチャーを選択します。
3. **バージョン** ドロップダウンリストから適切なバージョンを選択します。
4. **OpenShift v4.16 Linux Clients** エントリーの横にある **Download Now** をクリックして、ファイルを保存します。
5. アーカイブを展開します。

```
$ tar xvf <file>
```

6. **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
```

検証

- OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

3.1.1.2. Windows への OpenShift 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 カスタマーポータルの OpenShift Container Platform ダウンロードページ に移動します。
2. **バージョン** ドロップダウンリストから適切なバージョンを選択します。

3. **OpenShift v4.16 Windows Client** エントリーの横にある **Download Now** をクリックして、ファイルを保存します。
4. ZIP プログラムでアーカイブを展開します。
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
```

検証

- OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

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

3.1.1.3. macOS への OpenShift 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 カスタマーポータルの OpenShift Container Platform ダウンロードページ に移動します。
2. **バージョン** ドロップダウンリストから適切なバージョンを選択します。
3. **OpenShift v4.16 macOS Clients** エントリーの横にある **Download Now** をクリックして、ファイルを保存します。

注記

macOS arm64 の場合は、**OpenShift v4.16 macOS arm64 Client** エントリーを選択します。

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
```

検証

- **oc** コマンドを使用してインストールを確認します。

```
$ oc <command>
```

3.2. 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 からミラーにイメージをミラーリングできるように、コンテナー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。インストールホストで以下の手順を実行します。

前提条件

- 切断された環境で使用するミラーレジストリーを設定しました。

手順

1. **registry.redhat.io** プルシークレットを Red Hat OpenShift Cluster Manager からダウンロードします。
2. 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 | jq . > <path>/<pull_secret_file_in_json>
```

プルシークレットを保存するディレクトリ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"
    }
  }
}
```

3. 次のコマンドを実行して、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と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
```

<user_name> および **<password>** には、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。

出力例

```
BGVtbYk3ZHAtqXs=
```

4. JSON ファイルを編集し、レジストリーを記述するセクションをこれに追加します。

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

- < **mirror_registry** > 値については、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。たとえば、**registry.example.com** または **registry.example.com:8443** です。
- < **credentials> 値** 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。

変更済みのプルシークレットの例

```
{
 "auths": {
  "registry.example.com": {
   "auth": "BGVtbYk3ZHAtqXs=",
   "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"
  }
 }
}
```

3.3. OPENSHIFT 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。ミラーホストで以下の手順を実行します。

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 からプルシークレット をダウンロードし、ミラーリポジトリーへの認証を組み込むように変更している。
- 自己署名証明書を使用する場合は、証明書にサブジェクトの別名を指定しています。

手順

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. 次の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>
      ```

      &lt **;release_version&gt**; 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： **4.20.1**）。

   b. ローカルレジストリー名とホスト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は、**ocp-release** を指定する必要があります。

g. クラスターのアーキテクチャーのタイプをエクスポートします。

```
$ ARCHITECTURE=<cluster_architecture>
```

x86_64、**aarch64**、**s390x**、または **ppc64le** など、クラスターのアーキテクチャーを指定します。

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>
```

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ミラー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ーに一意であり、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。

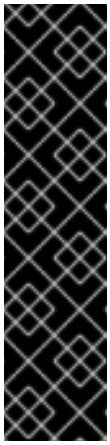
iv. イメージをリムーバブルメディア上のディレクトリーにミラーリングします。

```
$ 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ー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}
```

REMOVABLE_MEDIA_PATH 変数の場合は、イメージのミラーリング時に指定したものと同じパスを使用する必要があります。



重要

oc image mirror コマンドを実行すると、**error: unable to retrieve source image** エラーが発生する場合があります。このエラーは、イメージレジストリーに存在しなくなったイメージへの参照がイメージインデックスに含まれている場合に発生します。イメージインデックスは、それらのイメージを実行しているユーザーがアップグレードグラフの新しいポイントへのアップグレードパスを実行できるように、古い参照を保持する場合があります。一時的な回避策として、**--skip-missing** オプションを使用してエラーを回避し、イメージインデックスのダウンロードを続行できます。詳細は、Service Mesh Operator mirroring failed を参照してください。

- 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、クラスター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ーに一意であり、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に基づくインストールプログラムを作成するために、インストールプログラムを展開して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 ${LOCAL_SECRET_JSON} --icsp-file=<file> --
   command=openshift-install
   "${LOCAL_REGISTRY}/${LOCAL_REPOSITORY}:${OCP_RELEASE}-
   ${ARCHITECTURE}" \
   --insecure=true
   ```

オプション: ターゲットレジストリーの信頼を設定しない場合は、**--insecure=true** フラグを追加します。

- 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 のバージョンに適したイメージを確実に使用するために、ミラーリングし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。

5. installer-provisioned infrastructure を使用するクラスターの場合は、以下のコマンドを実行します。

```
$ openshift-install
```

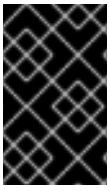
3.4. 代替のレジストリーまたはミラーリングされたレジストリーでの CLUSTER SAMPLES OPERATOR イメージストリームの使用

Red Hat レジストリーを使用する代わりに、代替またはミラーレジストリーを使用してイメージストリームをホストすることができます。

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。

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。これは、この手順で扱いません。



重要

Cluster Samples Operator は、非接続環境では **Managed** に設定する必要があります。イメージストリームをインストールするには、ミラーリングされたレジストリーが必要です。

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/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-
25-rhel7:latest
```

3. 次のコマンドを実行して、クラスターのイメージ設定オブジェクトを作成します。

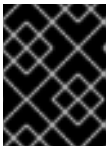
```
$ oc create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
```

4. 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。

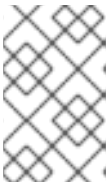
```
$ oc edit configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```



重要

現時点では、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で、この手順が必要です。

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

Cluster Samples Operator は、イメージストリームのインポートに失敗した場合にアラートを発行しますが、Cluster Samples Operator は定期的に再試行する場合もあれば、それらを再試行していないように見える場合もあります。

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。**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パージできます。これにより、イメージストリームが欠落しているためにテンプレートが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。

3.4.1. ミラーリングの Cluster Samples Operator のサポート

インストール中に、OpenShift Container Platform は、**openshift-cluster-samples-operator** namespace に **imagestreamtag-to-image** という名前の config map を作成します。

imagestreamtag-to-image config map には、各イメージストリームタグのエントリー (入力されるイメージ) が含まれています。

この config map に含まれるデータフィールドの各エントリーのキーは、**<image_stream_name>_<image_stream_tag_name>** という形式です。

OpenShift Container Platform の非接続インストール時に、Cluster Samples Operator のステータスは **Removed** に設定されます。これを **Managed** に変更することを選択した場合、サンプルがインストールされます。

注記

ネットワークが制限されている環境や非接続環境でサンプルを使用するには、ネットワーク外部のサービスにアクセスする必要がある場合があります。サービスの例には、Github、Maven Central、npm、RubyGems、PyPi などがあります。場合によっては、Cluster Samples Operator オブジェクトが必要なサービスに到達できるようにするために、追加の手順を実行する必要があります。

イメージストリームによるインポート用に、どのイメージをミラーリングする必要があるかを判断する際には、次の原則を使用してください。

- Cluster Samples Operator が **Removed** に設定される場合、ミラーリングされたレジストリーを作成するか、使用する必要のある既存のミラーリングされたレジストリーを判別できます。
- 新しい config map をガイドとして使用し、ミラーリングされたレジストリーに必要なサンプルをミラーリングします。
- Cluster Samples Operator 設定オブジェクトの **skippedImagestreams** リストに、ミラーリングされていないイメージストリームを追加します。
- Cluster Samples Operator 設定オブジェクトの **samplesRegistry** をミラーリングされたレジストリーに設定します。
- 次に、Cluster Samples Operator を **Managed** に設定し、ミラーリングしたイメージストリームをインストールします。

3.5. 関連情報

- イメージのプルソースの表示
- 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

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ーイメージを作成する方法を確認します。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ら、それを OpenShift イメージレジストリーにプッシュできます。

4.1. コンテナー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ー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. コンテナー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、コンテナーイメージの作成時に一般的に適用されます。

4.1.1.1. 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。

4.1.1.2. 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、**image** 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、**image: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 **image: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、**image: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。**image: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。

4.1.1.3. 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 つのコンテナー内で起動しないようにしてください。コンテナーは軽量で、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、コンテナーはネットワークの 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。

4.1.1.4. 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、コンテナー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ーを起動すると、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** でコンテナー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、コンテナー内で実行されると、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、コンテナー全体が停止され、**PID 1** プロセスから起動した子プロセスが終了します。

4.1.1.5. 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、**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。

現在のコンテナー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、これは今後変更される可能性があります。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。

4.1.1.6. 正しい順序での命令の指定

コンテナービルダーは **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ときや、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、同じイメージの次回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。

4.1.1.7. 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ーおよび他のコンテナー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ーに関連付けられる **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す。
- 公開されるポートは、1つのコンテナーを別のコンテナーにリンクする際にリンクされます。

4.1.1.8. 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。

4.1.1.9. 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、セキュリティー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、コンテナー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。

4.1.1.10. 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ー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 クラスターで実行中のコンテナー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、セキュリティー修正が必要になります。

4.1.1.11. 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、コンテナー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、コンテナー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、コンテナーが再起動されたり、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ー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。

コンテナーが破棄された後も保存する必要の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。コンテナーエンジンはコンテナーの **readonly** フラグをサポートしており、このフラグを使用して、コンテナー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ーイメージの作成時に適用されるガイドラインです。

4.1.2.1. 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。

4.1.2.2. 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ーを実行します。こうすることで、コンテナーエンジンの脆弱性が原因でコンテナー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
```

コンテナーユーザーは常に root グループのメンバーであるため、コンテナー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

コンテナーの機密領域のディレクトリーとファイルパーミッションを変更する場合は注意が必要です。**/etc/passwd** ファイルなどの機密領域に変更を適用すると、意図しないユーザーによるこれらのファイルの変更が許可され、コンテナーまたはホストがセキュリティーリスクにさらされる可能性があります。CRI-O は、コンテナーの **/etc/passwd** ファイルへに対する任意のユーザー ID の挿入をサポートしています。そのため、パーミッションを変更する必要はありません。

また、いずれのコンテナーイメージにも **/etc/passwd** ファイルが存在しないはずです。存在する場合、CRI-O コンテナーランタイムはランダムな UID を **/etc/passwd** ファイルに挿入できません。このような場合、コンテナーがアクティブな UID を解決する際に問題が発生する可能性があります。この要件を満たさない場合、特定のコンテナー化されたアプリケーションの機能に影響が及ぶ可能性があります。

さらに、コンテナー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 **anyuid** 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。

4.1.2.3. 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、コンテナーが停止、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。

4.1.2.4. 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。

4.1.2.5. 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。

注記

環境変数を使用して、コンテナー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、コンテナー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ー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関する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、コンテナーの **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。

4.1.2.6. 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 によるコンテナー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。

4.1.2.7. クラスタリング

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、クラスタリングスキームを動的にしておくことが重要です。

4.1.2.8. 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 はコンテナー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ーに入り、ログファイルを取得または表示する必要があります。

4.1.2.9. 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ー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ー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。

4.1.2.10. 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 によるコンテナー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ー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ー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 が付けられています。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リストとして表現されているタグのリストが含まれます。タグを使用して、コンテナー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ーイメージを提案しやすくなります。 `LABEL io.openshift.tags mongodb,mongodb24,nosql`
io.openshift.wants	コンテナー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リストを指定します。たとえば、コンテナーイメージに **mysql** と **redis** が必要で、コンテナーイメージに **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。 `LABEL io.openshift.wants mongodb,redis`
io.k8s.description	このラベルは、コンテナー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ー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。 `LABEL io.k8s.description The MySQL 5.5 Server with master-slave replication support`
io.openshift.non-scalable	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。 `LABEL io.openshift.non-scalable true`

io.openshift.min-memory および **io.openshift.min-cpu**	このラベルは、コンテナー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、このコンテナー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。
	
	`LABEL io.openshift.min-memory 16Gi`
	`LABEL io.openshift.min-cpu 4`

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、アセンブ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ー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

ビルドプロセスは次の 3 つの基本要素で構成されます。これらを組み合わせて最終的なコンテナー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

スクリプト	説明
assemble	**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。 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	**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。
save-artifacts	**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。 ● 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 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。
usage	**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。
test/run	**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。 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。  **注記** **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ーの **test/test-app** ディレクトリーです。

スクリプト	説明

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。各例では、**tar** の内容が **/tmp/s2i** ディレクトリーに展開されていることを前提としています。

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 外のコンテナー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、使用に関する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、ベースのコンテナー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ー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. コンテナーを実行します。
   - 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. コンテナーが実行され、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が分かります。

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 リポジトリーに配置して、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ーを参照する 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 リポジトリー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、タグによって識別される任意の数のコンテナーイメージで構成されます。これはコンテナーイメージリポジトリー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ー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を説明します。

5.2.1. イメージタグ

イメージタグは、イメージの特定のバージョンを区別するために、イメージストリーム内のコンテナーイメージに適用されるラベルです。OpenShift Container Platform においてイメージの特定のバージョンを整理および参照するのに役立ちます。

通常、イメージタグは何らかのバージョン番号を表します。次の例は、**:v3.11.59-2** というタグが付いた **registry.access.redhat.com/openshift3/jenkins-2-rhel7** というイメージを示しています。

```
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 には **oc tag** コマンドがあります。これは **docker tag** コマンドに似ていますが、イメージを直接操作するのではなく、イメージストリームを操作するものです。

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ー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多くな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	**myimage:v2.0.1**
アーキテクチャー	**myimage:v2.0-x86_64**
ベースイメージ	**myimage:v1.2-centos7**
最新 (不安定な可能性がある)	**myimage:latest**
最新 (安定性がある)	**myimage:stable**

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増大する可能性があります。

5.2.3. イメージストリームへのタグの追加

OpenShift Container Platform のイメージストリームは、タグで識別される 0 個以上のコンテナーイメージで構成されます。

各種のタグを利用できます。デフォルトの動作では、特定の時点のイメージを指す **permanent** タグが使用されます。**permanent** タグが使用されている場合、参照元が変更されても、参照先のタグは変更されません。

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>
```

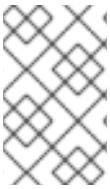
たとえば、**ruby** イメージストリームの **static-2.0** タグを **ruby** イメージストリーム **2.0** タグの現行のイメージを常に参照するように設定するには、以下を実行します。

```
$ 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** をコンテナー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、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、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>
```

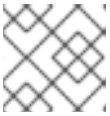
- 所定のイメージストリームおよびイメージの **sha** ID のイメージを参照するには、**ImageStreamImage** を使用します。

```
<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@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b28e
```

5.3. イメージプルポリシー

Pod のそれぞれのコンテナーにはコンテナー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 はコンテナーを作成する際に、コンテナーの **imagePullPolicy** を使用して、コンテナー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。

コンテナーの **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 イメージ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。

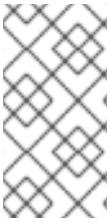
ただし、OpenShift Container Platform プロジェクト全体でイメージを参照する場合や、セキュリティー保護されたレジストリーからイメージを参照するなどの他のシナリオでは、追加の設定手順が必要になります。

イメージの プルシークレットは、Red Hat OpenShift Cluster Manager から取得 できます。このプルシークレットは **pullSecret** と呼ばれます。

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ーイメージを提供する組み込まれた各種の認証局 (Quay.io および registry.redhat.io) によって提供されるサービスで認証できます。

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

OpenShift イメージ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。



注記

Pod サービスアカウントまたは namespace を作成するときは、サービスアカウントが Docker プルシークレットでプロビジョニングされるまで待ちます。サービスアカウントが完全にプロビジョニングされる前に Pod を作成すると、Pod は OpenShift イメージレジストリーにアクセスできません。

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
```

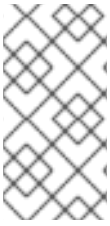
5.4.2. Pod が他のセキュリティー保護されたレジストリーからイメージを参照できるようにする設定

他のプライベートレジストリーレジストリーまたは保護されたレジストリーから保護されたコンテナーをプルするには、Docker や Podman などのコンテナークライアント認証情報からプルシークレットを作成し、それをサービスアカウントに追加する必要があります。

Docker と Podman は設定ファイルを使用して、保護されたレジストリーまたは保護されていないレジストリーへのログインに使用する認証の詳細を保存します。

- **Docker**: Docker は、デフォルトで **$HOME/.docker/config.json** を使用します。
- **Podman**: Podman は、デフォルトで **$HOME/.config/containers/auth.json** を使用します。

以前に保護されたレジストリーまたは保護されていないレジストリーにログインしたことがある場合、これらのファイルには認証情報が保存されます。



注記

quay.io や **quay.io/<example_repository>** のような一意のパスがある場合、Docker と Podman の認証情報ファイルおよび関連するプルシークレットには、同一レジストリーへの複数の参照を含めることができます。ただし、Docker および Podman のいずれも、まったく同じレジストリーパスの複数エントリーはサポートしていません。

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"
    },
    "quay.io/repository-main":{
      "auth":"b3Blb=",
      "email":"you@example.com"
    }
  }
}
```

プルシークレットの例

```
apiVersion: v1
data:
  .dockerconfigjson:
ewogICAiYXV0aHMiOnsKICAgICAgIm0iOnsKICAgICAgIsKICAgICAgICAgImF1dGgiOiJiM0JsYj0iLAogI
CAgICAgICAiZW1haWwiOiJ5b3VAZXhhbXBsZS5jb20iCiAgICAgIH0KICAgfQp9Cg==
kind: Secret
metadata:
  creationTimestamp: "2021-09-09T19:10:11Z"
  name: pull-secret
  namespace: default
  resourceVersion: "37676"
  uid: e2851531-01bc-48ba-878c-de96cfe31020
type: Opaque
```

5.4.2.1. プルシークレットの作成

手順

- 既存の認証ファイルからシークレットを作成します。
  - **.docker/config.json** を使用する Docker クライアントの場合は、次のコマンドを入力します。

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

  - **.config/containers/auth.json** を使用する Podman クライアントの場合は、次のコマンドを入力します。

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

- セキュアなレジストリーの Docker 認証情報ファイルがまだない場合は、次のコマンドを実行してシークレットを作成できます。

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

5.4.2.2. ワークロードでのプルシークレットの使用

プルシークレットを使用すると、ワークロードが次のいずれかの方法でプライベートレジストリーからイメージをプルできるようになります。

- シークレットを **ServiceAccount** にリンクする。これにより、そのサービスアカウントを使用するすべての Pod にシークレットが自動的に適用されます。
- ワークロード設定で **imagePullSecrets** を直接定義する。これは GitOps や ArgoCD などの環境に役立ちます。

手順

- サービスアカウントにシークレットを追加することで、Pod のイメージをプルするためのシークレットを使用できます。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と一致する必要があることに注意してください。デフォルトのサービスアカウントは **default** です。
  - 次のコマンドを入力してプルシークレットを **ServiceAccount** にリンクします。

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

  - 変更を確認するために、次のコマンドを入力します。

    ```
    $ oc get serviceaccount default -o yaml
    ```

    出力例

    ```
    apiVersion: v1
    imagePullSecrets:
    - name: default-dockercfg-123456
    - name: <pull_secret_name>
    kind: ServiceAccount
    metadata:
      annotations:
        openshift.io/internal-registry-pull-secret-ref: <internal_registry_pull_secret>
      creationTimestamp: "2025-03-03T20:07:52Z"
      name: default
      namespace: default
      resourceVersion: "13914"
      uid: 9f62dd88-110d-4879-9e27-1ffe269poe3
    secrets:
    - name: <pull_secret_name>
    ```

- シークレットをサービスアカウントにリンクする代わりに、Pod またはワークロード定義で直接参照することもできます。これは ArgoCD などの GitOps ワークフローに役立ちます。以下に例を示します。

Pod 仕様の例

```
apiVersion: v1
kind: Pod
metadata:
  name: <secure_pod_name>
spec:
  containers:
  - name: <container_name>
    image: quay.io/my-private-image
  imagePullSecrets:
  - name: <pull_secret_name>
```

ArgoCD ワークフローの例

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: <example_workflow>
spec:
  entrypoint: <main_task>
  imagePullSecrets:
  - name: <pull_secret_name>
```

5.4.2.3. 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

現在のプルシークレットを置き換えるか、新しいプルシークレットを追加することで、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。

重要

クラスターを別の所有者に譲渡するには、OpenShift Cluster Manager で譲渡を開始してから、クラスターのプルシークレットを更新する必要があります。OpenShift Cluster Manager で委譲を開始せずに、クラスターのプルシークレットを更新すると、クラスターは OpenShift Cluster Manager での Telemetry メトリクスの報告を停止します。

詳細は、Red Hat OpenShift Cluster Manager ドキュメントの「クラスター所有権の譲渡」を参照してください。

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. オプション: 既存のプルシークレットに新しいプルシークレットを追加するには、以下の手順を実行します。
   a. 以下のコマンドを入力してプルシークレットをダウンロードします。

```
$ oc get secret/pull-secret -n openshift-config --template='{{index .data
".dockerconfigjson" | base64decode}}' > <pull_secret_location> 1
```

   1 **<pull_secret_location>**: プルシークレットファイルへのパスを指定します。

   b. 以下のコマンドを実行して、新しいプルシークレットを追加します。

```
$ oc registry login --registry="<registry>" \ 1
--auth-basic="<username>:<password>" \ 2
--to=<pull_secret_location> 3
```

   1 **<registry>**: 新しいレジストリーを指定します。同じレジストリー内に複数のリポジトリーを指定できます (例: **--registry="<registry/my-namespace/my-repository>**)。

   2 **<username>:<password>**: 新しいレジストリーの認証情報を指定します。

   3 **<pull_secret_location>**: プルシークレットファイルへのパスを指定します。

   プルシークレットファイルを手動で更新することもできます。

2. 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 **<pull_secret_location>**: 新しい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。

注記

OpenShift Container Platform 4.7.4 の時点で、グローバルプルシークレットへの変更によってノードの drain (Pod の退避) の実行または再起動がトリガーされなくなりました。

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ー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は、OpenShift Container Platform 内でコンテナー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を使用すると、イメージのバージョンを管理し、ビルドとデプロイを自動化できます。

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ーと同様に、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きます。

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、コンテナーイメージレジストリーのコンテナー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 または 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、リポジトリーではなく、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ー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セキュリティーを設定できます。
- クラスターレベルでイメージを読み込んだり、リスト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: image.openshift.io/v1
kind: ImageStream
metadata:
  annotations:
    openshift.io/generated-by: OpenShiftNewApp
  labels:
    app: ruby-sample-build
    template: application-template-stibuild
  name: origin-ruby-sample (1)
  namespace: test
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: 2017-09-01T13:40:11Z
      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 リポジトリーパスです。

(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関する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のイメージは、リポジトリーからのイメージストリーム名とイメージ ID で構成されており、@ 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のようになります。

```
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、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

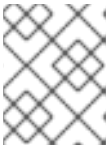
```
kind: ImageStream
apiVersion: image.openshift.io/v1
metadata:
  name: my-image-stream
# ...
  tags:
  - items:
    - created: 2017-09-02T10:15:09Z
      dockerImageReference: 172.30.56.218:5000/test/origin-ruby-
sample@sha256:47463d94eb5c049b2d23b03a9530bf944f8f967a0fe79147dd6b9135bf7dd13d
      generation: 2
      image: sha256:909de62d1f609a717ec433cc25ca5cf00941545c83a01fb31527771e1fab3fc5
```

```
    - created: 2017-09-01T13:40:11Z
      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 リポジトリー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構成されています。

```
<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関する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: image.openshift.io/v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    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
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
      - OPENSHIFT_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
      - OPENSHIFT_BUILD_NAME=ruby-sample-build-1
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_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を説明します。

重要

デフォルトプロジェクトでワークロードを実行したり、デフォルトプロジェクトへのアクセスを共有したりしないでください。デフォルトのプロジェクトは、コアクラスターコンポーネントを実行するために予約されています。

デフォルトプロジェクトである **default**、**kube-public**、**kube-system**、**openshift**、**openshift-infra**、**openshift-node**、および **openshift.io/run-level** ラベルが **0** または **1** に設定されているその他のシステム作成プロジェクトは、高い特権があるとみなされます。Pod セキュリティーアドミッション、Security Context Constraints、クラスターリソースクォータ、イメージ参照解決などのアドミッションプラグインに依存する機能は、高い特権を持つプロジェクトでは機能しません。

6.7.1. イメージストリームに関する情報の取得

イメージストリームに関する一般的な情報およびこれがポイントするすべてのタグ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関する一般情報と、それが指しているすべてのタグに関する詳細情報を取得するには、次のコマンドを入力します。

```
$ 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関して利用可能なすべての情報を取得するには、次のコマンドを入力します。

```
$ 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。

- 次のコマンドを入力して、イメージストリームタグがサポートするアーキテクチャーまたはオペレーティングシステムを検出します。

```
$ oc get istag <image-stream-tag> -ojsonpath="{range .image.dockerImageManifests[*]}
{.os}/{.architecture}{'\n'}{end}"
```

以下に例を示します。

```
$ oc get istag busybox:latest -ojsonpath="{range .image.dockerImageManifests[*]}
{.os}/{.architecture}{'\n'}{end}"
```

出力例

```
linux/amd64
linux/arm
linux/arm64
linux/386
linux/mips64le
linux/ppc64le
linux/riscv64
linux/s390x
```

6.7.2. イメージストリームへのタグの追加

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、外部コンテナー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用し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@sha256:438208801c4806548460b27bd1fbcb7bb188273d13871ab43f.
```

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:3.6
```

出力例

```
Deleted tag default/python:3.6
```

Cluster Samples Operator による非推奨のイメージストリームタグの処理方法の詳細は、Cluster Samples Operator からの非推奨のイメージストリームタグの削除 を参照してください。

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

外部コンテナー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。

2. 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。これにより、その動作がデフォルトに再設定されます。

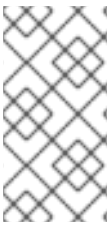
```
$ oc tag <repositiory/image> <image-name:tag>
```

6.8. イメージとイメージストリームのインポートと操作

次のセクションでは、イメージストリームをインポートして操作する方法を説明します。

6.8.1. 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と、このシークレットは使用されません。

手順

1. 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
```

2. 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、**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。

6.8.2. マニフェストリストの操作

--import-mode フラグを追加することにより、**oc import-image** または **oc tag** CLI コマンドを使用するときに、マニフェストリストの 1 つのサブマニフェストまたはすべてのマニフェストをインポートできます。

単一のサブマニフェストまたはマルチアーキテクチャーイメージを含むイメージストリームを作成するには、以下のコマンドを参照してください。

手順

- 次のコマンドを入力して、マルチアーキテクチャーイメージを含むイメージストリームを作成し、インポートモードを **PreserveOriginal** に設定します。

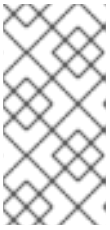
```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal' --reference-policy=local --confirm
```

出力例

```
---
Arch:           <none>
Manifests:      linux/amd64
sha256:6e325b86566fafd3c4683a05a219c30c421fbccbf8d87ab9d20d4ec1131c3451
                linux/arm64
sha256:d8fad562ffa75b96212c4a6dc81faf327d67714ed85475bf642729703a2b5bf6
                linux/ppc64le
sha256:7b7e25338e40d8bdeb1b28e37fef5e64f0afd412530b257f5b02b30851f416e1
---
```

- または、次のコマンドを入力して、マニフェストリストを破棄し、単一のサブマニフェストをインポートする **Legacy** インポートモードでイメージをインポートします。

```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='Legacy' --confirm
```



注記

--import-mode= のデフォルト値は **Legacy** です。この値を除外するか、**Legacy** または **PreserveOriginal** のいずれかを指定しないと、単一のサブマニフェストがインポートされます。無効なインポートモードは次のエラーを返します: **error: valid ImportMode values are Legacy or PreserveOriginal**。

6.8.2.1. 制限

マニフェストリストの操作には、次の制限があります。

- 場合によっては、ユーザーがサブマニフェストを直接使用したい場合があります。**oc adm prune images** が実行されている場合、または **CronJob** プルーナーが実行されている場合、サブマニフェストリストが使用されていることを検出できません。その結果、**oc adm prune images** または **CronJob** プルーナーを使用する管理者は、サブマニフェストを含むマニフェストリスト全体を削除する可能性があります。
この制限を回避するには、代わりにタグ別またはダイジェスト別のマニフェストリストを使用できます。

6.8.2.2. マニフェストリストの定期的なインポートの設定

マニフェストリストを定期的に再インポートするには、**--scheduled** フラグを使用できます。

手順

- 次のコマンドを入力して、マニフェストリストを定期的に更新するようにイメージストリームを設定します。

```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal' --scheduled=true
```

6.8.2.3. マニフェストリストのインポート時の SSL/TSL の設定

マニフェストリストをインポートするときに SSL/TSL を設定するには、**--insecure** フラグを使用できます。

手順

- **--insecure=true** を設定すると、マニフェストリストのインポートで SSL/TSL 検証がスキップされます。以下に例を示します。

```
$ oc import-image <multiarch-image-stream-tag> --from=<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal' --insecure=true
```

6.8.3. --import-mode のアーキテクチャーの指定

--import-mode= フラグを除外または含めることで、インポートしたイメージストリームをマルチアーキテクチャーとシングルアーキテクチャーの間で入れ替えることができます。

手順

- 次のコマンドを実行して、**--import-mode=** フラグを除外して、イメージストリームをマルチアーキテクチャーからシングルアーキテクチャーに更新します。

```
$ oc import-image <multiarch-image-stream-tag> --from=<registry>/<project_name>/<image-name>
```

- 次のコマンドを実行して、イメージストリームをシングルアーキテクチャーからマルチアーキテクチャーに更新します。

```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal'
```

6.8.4. --import-mode の設定フィールド

次の表に、**--import-mode=** フラグで使用できるオプションを示します。

パラメーター	説明

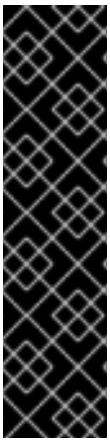
パラメーター	説明
レガシー	**--import-mode** のデフォルトオプション。指定すると、マニフェストリストが破棄され、単一のサブマニフェストがインポートされます。プラットフォームは、以下の優先順位で選択されます。 1. タグのアノテーション 2. コントロールプレーンアーキテクチャー 3. Linux/AMD64 4. 一覧の最初のマニフェスト
PreserveOriginal	指定すると、元のマニフェストが保持されます。マニフェスト一覧の場合は、マニフェストの一覧とそのすべてのサブマニフェストがインポートされます。

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、**Build** リソース、**DeploymentConfigs** リソースなどの OpenShift Container Platform で利用可能なネイティブリソースすべてで動作します。これらは、**Job** リソース、**ReplicationController** リソース、**ReplicaSet** リソース、Kubernetes **Deployment** リソースなどのネイティブ 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、**ruby:2.5** など、単一セグメントの値で構成されている必要があります。この場合、**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です。



重要

デフォルトプロジェクトでワークロードを実行したり、デフォルトプロジェクトへのアクセスを共有したりしないでください。デフォルトのプロジェクトは、コアクラスターコンポーネントを実行するために予約されています。

デフォルトプロジェクトである **default**、**kube-public**、**kube-system**、**openshift**、**openshift-infra**、**openshift-node**、および **openshift.io/run-level** ラベルが **0** または **1** に設定されているその他のシステム作成プロジェクトは、高い特権があるとみなされます。Pod セキュリティーアドミッション、Security Context Constraints、クラスターリソースクォータ、イメージ参照解決などのアドミッションプラグインに依存する機能は、高い特権を持つプロジェクトでは機能しません。

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: image.openshift.io/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。

2. 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. OPENSHIFT 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。

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    image.openshift.io/triggers:
      [
       {
         "from": {
           "kind": "ImageStreamTag", (1)
           "name": "example:latest", (2)
           "namespace": "myapp" (3)
         },
         "fieldPath": "spec.template.spec.containers[?(@.name==\"web\")].image", (4)
         "paused": false (5)
       },
      # ...
      ]
# ...
```

(1) 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。

(2) 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。

(3) 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。

(4) 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ーに正確に一致する 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** です。

(5) 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** コンテナー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
```

トリガーアノテーションを使用したデプロイメントの例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    image.openshift.io/triggers: '[{"from":
{"kind":"ImageStreamTag","name":"example:latest"},"fieldPath":"spec.template.spec.containers[
?(@.name==\"container\")].image"}]'
# ...
```

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

コンテナーイメージを保存し、提供するようにイメージレジストリーを設定できます。

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 リソースの **仕様** でクラスター全体でイメージを処理する特定のパラメーターを設定できます。

注記

次の設定不可能なパラメーターは、表にリストされていません。

- **DisableScheduledImport**
- **MaxImagesBulkImportedPerRepository**
- **MaxScheduledImportsPerMinute**
- **ScheduledImageImportMinimumIntervalSeconds**
- **InternalRegistryHostname**

表9.1 イメージコントローラー設定パラメーター

フィールド名	説明
kind.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。この CR の正規名および唯一の有効な名前は **cluster** です。
allowedRegistriesForImport	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ーイメージレジストリーを制限します。このリストを、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。 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。 **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。 **insecure**: insecure はレジストリーがセキュアか、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。

フィールド名	説明
additionalTrustedCA	**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 config map の参照です。 この config map の namespace は **openshift-config** です。config map の形式では、信頼する追加のレジストリー CA に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。
externalRegistryHostnames	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。
registrySources	コンテナー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などです。内部クラスターレジストリーの設定は含まれません。 **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。すべてのサブドメインを指定するには、ドメイン名に接頭辞としてアスタリスク (*) ワイルドカード文字を追加します。例: ***.example.com**レジストリー内で個別のリポジトリーを指定できます。例: **reg1.io/myrepo/myapp:latest** **blockedRegistries**: イメージのプルおよびプッシュアクションが拒否されるレジストリーです。すべてのサブドメインを指定するには、ドメイン名に接頭辞としてアスタリスク (*) ワイルドカード文字を追加します。例: ***.example.com**レジストリー内で個別のリポジトリーを指定できます。例: **reg1.io/myrepo/myapp:latest**他のすべてのレジストリーは許可されます。 **allowedRegistries**: イメージのプルおよびプッシュアクションが許可されるレジストリーです。すべてのサブドメインを指定するには、ドメイン名に接頭辞としてアスタリスク (*) ワイルドカード文字を追加します。例: ***.example.com**レジストリー内で個別のリポジトリーを指定できます。例: **reg1.io/myrepo/myapp:latest**他のすべてのレジストリーはブロックされます。 **containerRuntimeSearchRegistries**: イメージの短縮名を使用したイメージのプルおよびプッシュアクションが許可され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。 **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。

警告

allowedRegistries パラメーターを定義すると、明示的に一覧表示されない限り、**registry.redhat.io**、**quay.io**、およびデフォルトの OpenShift 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。ペイロードイメージが必要とするすべてのレジストリーを **allowedRegistries** 一覧に追加する必要があります。たとえば、**registry.redhat.io**、**quay.io**、および **internalRegistryHostname** レジストリーを一覧表示します。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。そうしないと、Pod の障害が伴います。

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。

表9.2 イメージコントローラーのステータスフィールドのパラメーター

パラメーター	説明
internalRegistryHostname	**internalRegistryHostname** を制御する Image Registry Operator によって設定されます。これはデフォルトの OpenShift 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、**OPENSHIFT_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。
externalRegistryHostnames	Image Registry Operator によって設定され、イメージレジストリーが外部に公開されるときに、イメージレジストリーの外部ホスト名を提供します。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。

9.2. MACHINE CONFIG OPERATOR の動作およびレジストリーの変更

Machine Config Operator (MCO)は、**image.config.openshift.io/cluster** カスタムリソース(CR)でレジストリーへの変更の有無を監視し、レジストリーが変更される際に特定の手順を実行します。

レジストリーへの変更が **image.config.openshift.io/cluster** CR に適用されると、MCO は以下の一連のアクションを実行します。

1. ノードを遮断します。特定のパラメーターによりノードがドレイン（解放）され、その他のパラメーターはドレインされない
2. CRI-O を再起動して変更を適用します
3. ノードを解放します

注記

MCO は、変更を検出してもノードを再起動しません。この期間中、サービスが利用できなくなる可能性があります。

9.2.1. レジストリーソースを許可およびブロックする場合

MCO は **image.config.openshift.io/cluster** リソースでレジストリーへの変更の有無を監視します。MCO が変更を検出すると、machine config pool (MCP) 内のノードでロールアウトがトリガーされます。許可されたレジストリーのリストは、各ノードの **/etc/containers/policy.json** ファイルでイメージ署名ポリシーを更新するために使用されます。**/etc/containers/policy.json** ファイルへの変更において、ノードをドレインする必要はありません。

9.2.2. containerRuntimeSearchRegistries パラメーターを使用する場合

ノードが **Ready** 状態に戻った後に、**containerRuntimeSearchRegistries** パラメーターが追加されると、MCO はリスト表示されるレジストリーで各ノードの **/etc/containers/registries.conf.d** ディレクトリーにファイルを作成します。このファイルは、**/etc/containers/registries.conf** ファイルの非修飾検索レジストリーのデフォルトリストをオーバーライドします。修飾されていない検索レジストリーのデフォルトリストにフォールバックする方法はありません。

重要

containerRuntimeSearchRegistries パラメーターは、Podman および CRI-O コンテナーエンジンを使用する場合のみ機能します。リストのレジストリーは、ビルドおよびイメージストリームではなく、Pod 仕様でのみ使用できます。

9.3. イメージレジストリーの設定

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。

手順

1. 以下のコマンドを実行して **image.config.openshift.io/cluster** CR 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
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:
    - domainName: quay.io
      insecure: false
  additionalTrustedCA:
    name: myconfigmap
  registrySources:
    allowedRegistries:
```

```
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

注記

allowedRegistries、**blockedRegistries** または **insecureRegistries** パラメーターを使用する場合、レジストリー内に個別のリポジトリーを指定できます。例: **reg1.io/myrepo/myapp:latest**

起こりうるセキュリティー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の使用を避けてください。

検証

1. 変更を確認するには、以下のコマンドを実行してノードを一覧表示します。

```
$ oc get nodes
```

出力例

```
NAME                                         STATUS                     ROLES                  AGE   VERSION
ip-10-0-137-182.us-east-2.compute.internal   Ready,SchedulingDisabled   worker                 65m   v1.29.4
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane          74m   v1.29.4
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane          75m   v1.29.4
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker                 65m   v1.29.4
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker                 63m   v1.29.4
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane          73m   v1.29.4
```

9.3.1. 許可リストへの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、イメージのプルおよびプッシュアクションのレジストリー内に、レジストリーの許可リストまたは個別のリポジ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。

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リスト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リストを作成している場合、コンテナーラ

ンタイムはそれらのレジストリーのみを検索します。許可リスト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 パラメーターを定義すると、明示的に一覧表示されない限り、**registry.redhat.io**、**quay.io**、およびデフォルトの OpenShift 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。ペイロードイメージが必要とするすべてのレジストリーを **allowedRegistries** 一覧に追加する必要があります。たとえば、**registry.redhat.io**、**quay.io**、および **internalRegistryHostname** レジストリーを一覧表示します。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。そうしないと、Pod の障害が伴います。

手順

- 以下のコマンドを実行して、**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、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:
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 設定の更新を行ったら、以下のコマンドを実行してノードを一覧表示します。

```
$ oc get nodes
```

出力例

```
NAME              STATUS   ROLES                AGE   VERSION
<node_name>       Ready    control-plane,master   37m   v1.27.8+4fab27b
```

2. 次のコマンドを実行して、ノードでデバッグモードに入ります。

```
$ oc debug node/<node_name>
```

<node_name> はノード名に置き換えてください。

3. プロンプトが表示されたら、ターミナルに **chroot /host** を入力します。

```
sh-4.4# chroot /host
```

検証

1. 次のコマンドを実行して、レジストリーがポリシーファイルにあることを確認します。

```
sh-5.1# cat /etc/containers/policy.json | jq '.'
```

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にのみアクセスできることを示しています。

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
```

9.3.2. 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またはレジストリー内の個々のリポジ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リスト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リストを作成した場合、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** 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。環境内のペイロードイメージには、これらのレジストリーへのアクセスが必要です。

手順

- 以下のコマンドを実行して、**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、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:
    blockedRegistries:
    - untrusted.com
    - reg1.io/myrepo/myapp:latest
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

blockedRegistries および **allowedRegistries** パラメーターの両方を設定することはできません。どちらか一方を選択する必要があります。

1. 次のコマンドを実行して、ノードの一覧を取得します。

   ```
   $ oc get nodes
   ```

   出力例

   ```
   NAME              STATUS   ROLES                AGE   VERSION
   <node_name>       Ready    control-plane,master   37m   v1.27.8+4fab27b
   ```

2. 次のコマンドを実行し、ノード上でデバッグモードに入ります。

   ```
   $ oc debug node/<node_name>
   ```

   <node_name> は、詳細が必要なノードの名前に置き換えます。

3. プロンプトが表示されたら、ターミナルに **chroot /host** を入力します。

   ```
   sh-4.4# chroot /host
   ```

検証

1. 次のコマンドを実行して、レジストリーがポリシーファイルにあることを確認します。

   ```
   sh-5.1# cat 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ブロックされてい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 = "untrusted.com"
     blocked = true
   ```

9.3.3. ペイロードレジストリーのブロック

ミラーリング設定では、**ImageContentSourcePolicy** (ICSP)オブジェクトを使用して、切断された環境でアップストリームペイロードレジストリーをブロックできます。以下の手順例は、**quay.io/openshift-payload** ペイロードレジストリーをブロックする方法を示しています。

手順

1. **ImageContentSourcePolicy** (ICSP) オブジェクトを使用してミラー設定を作成し、ペイロードをインスタンスのレジストリーにミラーリングします。以下の ICSP ファイルの例は、ペイロード **internal-mirror.io/openshift-payload** をミラーリングします。

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: my-icsp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - internal-mirror.io/openshift-payload
    source: quay.io/openshift-payload
```

2. オブジェクトがノードにデプロイされたら、**/etc/containers/registries.conf** カスタムリソース (CR)をチェックして、ミラー設定が設定されていることを確認します。

   出力例

```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  mirror-by-digest-only = true

[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

3. 以下のコマンドを使用して **image.config.openshift.io** CR を編集します。

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

4. ペイロードレジストリーをブロックするには、次の設定を **image.config.openshift.io** CR に追加します。

```
spec:
  registrySources:
    blockedRegistries:
     - quay.io/openshift-payload
```

検証

- ノードの **/etc/containers/registries.conf** ファイルをチェックして、上流のペイロードレジストリーがブロックされていることを確認します。

/etc/containers/registries.conf ファイルの例

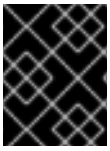
```
[[registry]]
  prefix = ""
  location = "quay.io/openshift-payload"
  blocked = true
  mirror-by-digest-only = true

[[registry.mirror]]
  location = "internal-mirror.io/openshift-payload"
```

9.3.4. 非セキュアなレジストリー

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、またはレジストリー内に個別のリポジ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。



重要

起こりうるセキュリティー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の使用を避けてください。

+ :leveloffset: +1

警告

allowedRegistries パラメーターを定義すると、明示的に一覧表示されない限り、**registry.redhat.io**、**quay.io**、およびデフォルトの OpenShift 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。ペイロードイメージが必要とするすべてのレジストリーを **allowedRegistries** 一覧に追加する必要があります。たとえば、**registry.redhat.io**、**quay.io**、および **internalRegistryHostname** レジストリーを一覧表示します。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。そうしないと、Pod の障害が伴います。

手順

- 以下のコマンドを実行して、**image.config.openshift.io/cluster** カスタムリソース(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:
    insecureRegistries:
    - insecure.com
    - reg4.io/myrepo/myapp:latest
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com
    - reg4.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

検証

- ノードで以下のコマンドを実行して、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します。

```
$ cat /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
```

9.4. イメージの短縮名を許可するレジストリーの追加について

イメージの短縮名を使用すると、pull **spec** パラメーターに完全修飾ドメイン名を追加せずに、イメージを検索できます。

たとえば、**registry.access.redhat.com/rhe7/etcd** の代わりに **rhel7/etcd** を使用できます。**image.config.openshift.io/cluster** カスタムリソース (CR) を編集して、イメージの短縮名を検索するためにレジストリーを追加できます。

完全パスを使用することが実際的ではない場合に、短縮名を使用できる場合があります。たとえば、ク

ラスターが DNS が頻繁に変更される複数の内部レジストリーを参照する場合、毎回の変更ごとにプル仕様の完全修飾ドメイン名を更新する必要が生じます。この場合は、イメージの短縮名を使用した方が良いでしょう。

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リスト表示されるレジストリーを検索します。短縮名を使用してイメージをプル際に、**containerRuntimeSearchRegistries** パラメーターでレジストリーのリストを作成している場合、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す。

9.4.1. イメージの短縮名を使用しない場合

公開レジストリーで認証が必要な場合、イメージがデプロイされない可能性があるため、公開レジストリーでイメージの短縮名を使用することは推奨しません。公開レジストリーで完全修飾イメージ名を使用します。

通常、Red Hat の内部レジストリーまたはプライベートレジストリーは、イメージの短縮名の使用をサポートしています。

各パブリックレジストリーが異なる認証情報を必要とし、クラスターでグローバルプルシークレットにパブリックレジストリーがリストされない場合には、**containerRuntimeSearchRegistries** パラメーターの下に複数のパブリックレジストリーをリストできません。

認証が必要なパブリックレジストリーの場合、レジストリーの認証情報がグローバルプルシークレットに格納されている場合にのみ、イメージの短縮名を使用できます。

警告

containerRuntimeSearchRegistries パラメーター (**registry.redhat.io**、**docker.io**、および **quay.io** レジストリーを含む) にパブリックレジストリーをリスト表示する場合、認証情報はリスト上のすべてのレジストリーに公開され、ネットワークおよびレジストリーの攻撃にされされるリスクが生じます。イメージをプルするためのプルシークレットは 1 つしかないため、グローバルプルシークレットで定義されているように、そのシークレットは、そのリスト内のすべてのレジストリーに対して認証するために使用されます。したがって、リストにパブリックレジストリーを含めると、セキュリティーリスクが発生します。

9.4.2. イメージの短縮名を許可する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、イメージの短縮名を検索するために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。

警告

allowedRegistries パラメーターを定義すると、明示的に一覧表示されない限り、**registry.redhat.io**、**quay.io**、およびデフォルトの OpenShift 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。ペイロードイメージが必要とするすべてのレジストリーを **allowedRegistries** 一覧に追加する必要があります。たとえば、**registry.redhat.io**、**quay.io**、および **internalRegistryHostname** レジストリーを一覧表示します。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。そうしないと、Pod の障害が伴います。

手順

- **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
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:
    - domainName: quay.io
      insecure: false
  additionalTrustedCA:
    name: myconfigmap
  registrySources:
    containerRuntimeSearchRegistries:
    - reg1.io
    - reg2.io
    - reg3.io
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - reg1.io
    - reg2.io
    - reg3.io
    - image-registry.openshift-image-registry.svc:5000
...
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 次のコマンドを実行して、ノードの一覧を取得します。

```
$ oc get nodes
```

出力例

```
NAME              STATUS   ROLES                AGE   VERSION
<node_name>       Ready    control-plane,master   37m   v1.27.8+4fab27b
```

2. 次のコマンドを実行し、ノード上でデバッグモードに入ります。

```
$ oc debug node/<node_name>
```

3. プロンプトが表示されたら、ターミナルに **chroot /host** を入力します。

```
sh-4.4# chroot /host
```

検証

1. 次のコマンドを実行して、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します。

```
sh-5.1# cat /etc/containers/registries.conf.d/01-image-searchRegistries.conf
```

出力例

```
unqualified-search-registries = ['reg1.io', 'reg2.io', 'reg3.io']
```

9.4.3. イメージレジストリーアクセス用の追加トラストストアの設定

image.config.openshift.io/cluster カスタムリソース(CR)へのイメージレジストリーアクセス時に信頼される追加の認証局(CA)を持つ設定マップへの参照を追加できます。

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

1. **openshift-config** namespace に設定マップを作成し、**image.config.openshift.io** CR の **AdditionalTrustedCA** パラメーターで設定マップ名を使用します。これにより、クラスターが外部イメージレジストリーと通信する際に信頼される必要のある CA が追加されます。

イメージレジストリー CA の config map 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: |
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

ここでは、以下のようになります。

data:registry.example.com:

この CA が信頼されるレジストリーのホスト名の例。

data:registry-with-port.example.com..5000:

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の例。レジストリーに **registry-with-port.example.com:5000 などのポートがある場合、:** は **..** に置き換える必要があります。
PEM 証明書の内容は、信頼する追加のレジストリー CA の値です。

2. 任意。次のコマンドを実行して、追加の CA を設定します。

```
$ 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
```

9.5. イメージレジストリーリポジトリーのミラーリングについて

コンテナーレジストリーのリポジトリーのミラーリングを設定すると、次のタスクを実行できます。

- ソースイメージのレジストリーのリポジトリー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ーで解決できるようにします。
- 各ターゲットリポジトリーに対して複数のミラーリングされたリポジトリーを特定し、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。

OpenShift Container Platform のリポジトリーミラーリングには、以下の属性が含まれます。

- 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。
- 切断された環境のクラスターは、**quay.io** などの重要な場所からイメージをプルし、会社のファイアウォールの背後にあるレジストリーに要求されたイメージを提供することができます。
- 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ー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ー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ーに対して試行する。成功すると、イメージはノードにプルされる。

次の方法でリポジトリーミラーリングを設定できます。

- OpenShift Container Platform のインストール時:
  OpenShift Container Platform に必要なコンテナーイメージをプルし、それらのイメージを会社のファイアウォールの背後に配置することで、非接続環境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 のインストール中にミラーリングを設定しなかった場合は、以下のカスタムリソース (CR) オブジェクトのいずれかを使用して、インストール後に設定できます。

  - **ImageDigestMirrorSet** (IDMS)。このオブジェクトを使用すると、ダイジェスト仕様を使用して、ミラーリングされたレジストリーからイメージを取得できます。IDMS CR を使用すると、イメージのプルが失敗した場合に、ソースレジストリーからのプルの継続的な試行を許可または停止するフォールバックポリシーを設定できます。

  - **ImageTagMirrorSet** (ITMS)。このオブジェクトを使用すると、イメージタグを使用して、ミラーリングされたレジストリーからイメージをプルできます。ITMS CR を使用すると、イメージのプルが失敗した場合に、ソースレジストリーからのプルの継続的な試行を許可または停止するフォールバックポリシーを設定できます。

  - **ImageContentSourcePolicy** (ICSP)。このオブジェクトを使用すると、ダイジェスト仕様を使用して、ミラーリングされたレジストリーからイメージを取得できます。ミラーが機能しない場合、ICSP CR は必ずソースレジストリーにフォールバックします。

重要

ImageContentSourcePolicy (ICSP) オブジェクトを使用してリポジトリーミラーリングを設定することは、非推奨の機能です。非推奨の機能は OpenShift Container Platform に引き続き含まれており、サポートが継続されます。これは今後のリリースで削除されるため、新しいデプロイメントには推奨されません。

ImageContentSourcePolicy オブジェクトの作成に使用した既存の YAML ファイルがある場合は、**oc adm migrate icsp** コマンドを使用して、それらのファイルを **ImageDigestMirrorSet** YAML ファイルに変換できます。詳細については、イメージレジストリーリポジトリーミラーリング用の ImageContentSourcePolicy (ICSP)ファイルの変換を参照してください。

これらのカスタムリソースオブジェクトはそれぞれ、次の情報を識別します。

- ミラーリングするコンテナーイメージリポジトリーのソース

- コンテンツを提供する各ミラーリポジトリーの個別のエントリー

次のアクションと、それがノードの drain 動作にどのように影響するかに注意してください。

- IDMS または ICSP CR オブジェクトを作成すると、MCO はノードの drain またはリブートを実行しません。
- ITMS CR オブジェクトを作成すると、MCO はノードの drain とリブートを実行します。
- ITMS、IDMS、または ICSP CR オブジェクトを削除すると、MCO はノードの drain とリブートを実行します。
- ITMS、IDMS、または ICSP CR オブジェクトを変更すると、MCO はノードの drain とリブートを実行します。

重要

- MCO が以下の変更のいずれかを検出すると、ノードの drain (Pod の退避) の実行または再起動を行わずに更新を適用します。
  - マシン設定の **spec.config.passwd.users.sshAuthorizedKeys** パラメーターの SSH キーの変更。
  - **openshift-config** namespace でのグローバルプルシークレットまたはプルシークレットへの変更。
  - Kubernetes API Server Operator による **/etc/kubernetes/kubelet-ca.crt** 認証局 (CA) の自動ローテーション。
- MCO は、**/etc/containers/registries.conf** ファイルへの変更 (**ImageDigestMirrorSet**、**ImageTagMirrorSet**、または **ImageContentSourcePolicy** オブジェクトの編集など) を検出すると、対応するノードの drain (Pod の退避) を実行し、変更を適用して、ノードをスケジューリング対象に戻します。次の変更ではノードの drain (Pod の退避) の実行は発生しません。
  - **pull-from-mirror = "digest-only"** パラメーターがミラーごとに設定されたレジストリーの追加。
  - **pull-from-mirror = "digest-only"** パラメーターがレジストリーに設定されたミラーの追加。
  - **unqualified-search-registries** へのアイテムの追加。

新規クラスターの場合、必要に応じて IDMS、ITMS、および ICSP CR オブジェクトを使用できます。ただし、IDMS と ITMS の使用を推奨します。

クラスターをアップグレードした場合、既存の ICSP オブジェクトは安定を維持し、IDMS オブジェクトと ICSP オブジェクトの両方がサポートされるようになります。ICSP オブジェクトを使用するワークロードは引き続き想定どおりに機能します。一方、IDMS CR で導入されたフォールバックポリシーを利用する場合は、**oc adm migrate icsp** コマンドを使用して、現在のワークロードを IDMS オブジェクトに移行できます。これは、後述の **イメージレジストリーリポジトリーミラーリング用の ImageContentSourcePolicy (ICSP) ファイルの変換** セクションで説明しています。IDMS オブジェクトへの移行に、クラスターの再起動は必要ありません。

注記

クラスターで **ImageDigestMirrorSet**、**ImageTagMirrorSet**、または **ImageContentSourcePolicy** オブジェクトを使用してリポジトリーミラーリングを設定する場合、ミラーリングされたレジストリーにはグローバルプルシークレットのみを使用できます。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。

9.5.1. イメージレジストリーのリポジトリーミラーリングの設定

インストール後のミラー設定カスタムリソース (CR) を作成して、ソースイメージレジストリーからミラーリングされたイメージレジストリーにイメージプル要求をリダイレクトできます。

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リポジトリーを設定します。以下のいずれかを実行します。
   - Red Hat Quay でミラーリングされたリポジトリーのセットアップRed Hat Quay を使用して、あるリポジトリーから別のリポジトリーにイメージをコピーでき、それらのリポジトリーを時間の経過とともに繰り返し同期することもできます。
     - Red Hat Quay リポジトリーミラーリング
   - **skopeo** などのツールを使用して、ソースリポジトリーからミラーリングされたリポジトリーにイメージを手動でコピーします。
     たとえば、{op-system-base-full 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 --all \
     docker://registry.access.redhat.com/ubi9/ubi-minimal:latest@sha256:5cf... \
     docker://example.io/example/ubi-minimal
     ```

     この例では、**example.io** という名前のコンテナー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ーがあります。**ubi9/ubi-minimal** イメージを **registry.access.redhat.com** から **example.io** にコピーします。ミラーリングされた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に送信された要求をミラーリングされたリポジトリーにリダイレクトできます。
2. 次の例のいずれかを使用して、インストール後のミラー設定のカスタムリソース(CR)を作成します。
   - 必要に応じて **ImageDigestMirrorSet** または **ImageTagMirrorSet** CR を作成し、ソースとミラーを独自のレジストリーとリポジトリーのペアとイメージに置き換えます。

     ```
     apiVersion: config.openshift.io/v1
     kind: ImageDigestMirrorSet
     metadata:
       name: ubi9repo
     spec:
       imageDigestMirrors:
       - mirrors:
         - example.io/example/ubi-minimal
     ```

```
    - example.com/example2/ubi-minimal
    source: registry.access.redhat.com/ubi9/ubi-minimal
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.com/redhat
    source: registry.example.com/redhat
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.com
    source: registry.example.com
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/image
    source: registry.example.com/example/myimage
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net
    source: registry.example.com/example
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/registry-example-com
    source: registry.example.com
    mirrorSourcePolicy: AllowContactingSource
```

- **ImageContentSourcePolicy** カスタムリソースを作成し、ソースとミラーを独自のレジストリーとリポジトリーのペアとイメージに置き換えます。

```
apiVersion: operator.openshift.io/v1alpha1
kind: ImageContentSourcePolicy
metadata:
  name: mirror-ocp
spec:
  repositoryDigestMirrors:
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-release
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

ここでは、以下のようになります。

- mirror.registry.com:443/ocp/release

ミラーイメージレジストリーおよびリポジトリーの名前を指定します。

source: quay.io/openshift-release-dev/ocp-release

ミラーリングされるコンテンツが含まれるオンラインレジストリーおよびリポジトリーを指定します。

3. 次のコマンドを実行して、新しいオブジェクトを作成します。

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

オブジェクトの作成後、Machine Config Operator (MCO) は **ImageTagMirrorSet** オブジェクトのみのノードの drain (Pod の退避) を実行します。MCO は、**ImageDigestMirrorSet** オブ

ジェクトと **ImageContentSourcePolicy** オブジェクトのノードの drain (Pod の退避) を実行しません。

4. 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。

   a. 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   v1.29.4
   ip-10-0-138-148.ec2.internal   Ready                     master   11m  v1.29.4
   ip-10-0-139-122.ec2.internal   Ready                     master   11m  v1.29.4
   ip-10-0-147-35.ec2.internal    Ready                    worker   7m   v1.29.4
   ip-10-0-153-12.ec2.internal    Ready                    worker   7m   v1.29.4
   ip-10-0-154-10.ec2.internal    Ready                    master   11m  v1.29.4
   ```

   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. ルートディレクトリーを **/host** に変更します。

   ```
   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
   ```

   次の出力は、インストール後のミラー設定 CR が適用される **registries.conf** ファイルを示しています。

   出力例

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

   [[registry]]
     prefix = ""
     location = "registry.access.redhat.com/ubi9/ubi-minimal"

     [[registry.mirror]]
       location = "example.io/example/ubi-minimal"
       pull-from-mirror = "digest-only"
   ```

```
  [[registry.mirror]]
    location = "example.com/example/ubi-minimal"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com"

  [[registry.mirror]]
    location = "mirror.example.net/registry-example-com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/example"

  [[registry.mirror]]
    location = "mirror.example.net"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/example/myimage"

  [[registry.mirror]]
    location = "mirror.example.net/image"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com"

  [[registry.mirror]]
    location = "mirror.example.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.example.com/redhat"

  [[registry.mirror]]
    location = "mirror.example.com/redhat"
    pull-from-mirror = "digest-only"
[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi9/ubi-minimal"
  blocked = true

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal-tag"
    pull-from-mirror = "tag-only"
```

[[registry]].location = "registry.access.redhat.com/ubi9/ubi-minimal":: プル仕様にリストされているリポジトリー。**[[registry.mirror]].location = "example.io/example/ubi-minimal"**:: そのリポジトリーのミラーを示します。**[[registry.mirror]].pull-from-mirror =**

"digest-only":: ミラーからイメージがダイジェスト参照イメージであることを示しています。**[[registry]].blocked = true**:: このリポジトリーに **NeverContactSource** パラメーターが設定されていることを示します。**[[registry.mirror]].pull-from-mirror = "tag-only"**:: ミラーからのイメージプルがタグ参照イメージであることを示します。

e. ソースからノードにイメージをプルし、ミラーによって解決されるかどうかを確認します。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi9/ubi-minimal@sha256:5cf...
```

トラブルシューティング

リポジトリーのミラーリング手順が説明どおりに機能しない場合は、リポジトリーミラーリングの仕組みに関する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。

9.5.2. イメージレジストリーリポジトリーのミラーリング設定パラメーター

ミラーリング用にイメージリポジトリーを設定するときに、以下の表を使用してパラメーターに関する情報を取得できます。

パラメーター	値と情報
apiVersion:	必須。値は **config.openshift.io/v1** API である必要があります。
kind:	プルタイプに応じたオブジェクトの種類。**ImageDigestMirrorSet** タイプはダイジェスト参照イメージをプルします。**ImageTagMirrorSet** タイプはタグ参照イメージをプルします。
spec: imageDigestMirrors:	イメージのプル方法のタイプ。**ImageDigestMirrorSet CR に imageDigestMirrors を使用します。ImageTagMirrorSet** CR には **imageTagMirrors** を使用します。
- mirrors: - example.io/example/ubi-minimal	ミラーリングされたイメージのレジストリーとリポジトリーの名前。
- mirrors: - example.com/example2/ubi-minimal	このパラメーターの値は、各ターゲットリポジトリーのセカンダリーミラーリポジトリーの名前です。1 つのミラーがダウンした場合、ターゲットリポジトリーはセカンダリーミラーを使用できます。

パラメーター	値と情報
ソース：registry.access.redhat.com/ubi9/ubi-minimal	レジストリーおよびリポジトリーソース。ソースは、イメージのプル仕様にリストされているリポジトリーです。
mirrorSourcePolicy: AllowContactingSource	イメージのプルが失敗した場合のフォールバックポリシーを示すオプションのパラメーター。**AllowContactingSource** 値により、ソースリポジトリーからのイメージのプルの継続的な試行が可能になります。デフォルト値。**NeverContactSource** は、ソースリポジトリーからイメージのプルが継続しようとするのを防ぎます。
ソース：registry.example.com/redhat: レジストリー内の namespace を示すオプションのパラメーター。レジストリーに namespace を設定すると、その namespace で任意のイメージを使用できます。レジストリードメインをソースとして使用する場合、オブジェクトはレジストリーからすべてのリポジトリーに適用されます。	**ソース：registry.example.com**
レジストリーを示す任意のパラメーターです。そのレジストリー内の任意のイメージを許可します。レジストリー名を指定すると、ソースレジストリーからミラーレジストリーまでのすべてのリポジトリーにオブジェクトが適用されます。	**ソース：registry.example.com/example/myimage**
イメージ **registry.example.com/example/myimage@sha256:...** をミラー **mirror.example.net/image@sha256:..** からプルします。	**ソース：registry.example.com/example**
ミラー **mirror.example.net/image@sha256:...** からソースレジストリー namespace のイメージ **registry.example.com/example/image@sha256:...** をプルします。	**ソース：registry.example.com**

9.5.3. イメージレジストリーリポジトリーミラーリング用の ImageContentSourcePolicy (ICSP) ファイルの変換

ImageContentSourcePolicy (ICSP) オブジェクトを使用してリポジトリーミラーリングを設定することは、非推奨の機能です。

この機能は引き続き OpenShift Container Platform に含まれており、引き続きサポートされます。ただし、この製品の将来のリリースでは削除される予定であり、新しいデプロイメントには推奨されません。

ICSP オブジェクトは、リポジトリーミラーリングを設定するために **ImageDigestMirrorSet** および **ImageTagMirrorSet** オブジェクトに置き換えられています。**ImageContentSourcePolicy** オブジェクトの作成に使用した既存の YAML ファイルがある場合は、**oc adm migrate icsp** コマンドを使用して、それらのファイルを **ImageDigestMirrorSet** YAML ファイルに変換できます。このコマンドは、API を現在のバージョンに更新し、**kind** 値を **ImageDigestMirrorSet** に変更し、**spec.repositoryDigestMirrors** を **spec.imageDigestMirrors** に変更します。ファイルの残りの部分は変更されません。

移行によって **registries.conf** ファイルは変更されないため、クラスターを再起動する必要はありません。

ImageDigestMirrorSet または **ImageTagMirrorSet** オブジェクトの詳細は、前のセクションの「イメージレジストリーリポジトリーミラーリングの設定」を参照してください。

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。
- クラスターに **ImageContentSourcePolicy** オブジェクトがあることを確認します。

手順

1. 次のコマンドを使用して、1 つ以上の **ImageContentSourcePolicy** YAML ファイルを **ImageDigestMirrorSet** YAML ファイルに変換します。

```
$ oc adm migrate icsp <file_name>.yaml <file_name>.yaml <file_name>.yaml --dest-dir
<path_to_the_directory>
```

ここでは、以下のようになります。

<file_name>

ソース **ImageContentSourcePolicy** YAML の名前を指定します。複数のファイル名をリストできます。

--dest-dir

オプション: 出力 **ImageDigestMirrorSet** YAML のディレクトリーを指定します。設定されていない場合、ファイルは現在のディレクトリーに書き込まれます。
たとえば、次のコマンドは **icsp.yaml** および **icsp-2.yaml** ファイルを変換し、新しい YAML ファイルを **idms-files** ディレクトリーに保存します。

```
$ oc adm migrate icsp icsp.yaml icsp-2.yaml --dest-dir idms-files
```

出力例

```
wrote ImageDigestMirrorSet to idms-
files/imagedigestmirrorset_ubi8repo.5911620242173376087.yaml
wrote ImageDigestMirrorSet to idms-
```

```
files/imagedigestmirrorset_ubi9repo.6456931852378115011.yaml
```

2. 次のコマンドを実行して CR オブジェクトを作成します。

```
$ oc create -f <path_to_the_directory>/<file-name>.yaml
```

ここでは、以下のようになります。

<path_to_the_directory>
--dest-dir フラグを使用した場合は、ディレクトリーへのパスを指定します。

<file_name>
ImageDigestMirrorSet YAML の名前を指定します。

3. IDMS オブジェクトがロールアウトされた後、ICSP オブジェクトを削除します。

9.6. 関連情報

- マニフェストリストの操作
- フィーチャーゲートについて
- グローバルクラスタープルシークレットの更新

第10章 イメージの使用

10.1. イメージの使用の概要

OpenShift Container Platform で提供されている Source-to-Image (S2I)、データベース、およびその他のコンテナーイメージを使用できます。これらのイメージは、クラスター上でコンテナー化されたアプリケーションをビルドおよびデプロイするのに必要です。

Red Hat の公式コンテナーイメージは、registry.redhat.io の Red Hat レジストリーで提供されます。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** リポジトリーにあります。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ー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 のサポート対象イメージは、すべて Red Hat Ecosystem Catalog のコンテナー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

コンテナー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ーイメージを確認し、使用するようにしてください。

10.2. SOURCE-TO-IMAGE

Source-to-Image (S2I) イメージは、Node.js、Python、Java といった言語用のランタイムベースイメージの特別なバージョンです。ランタイム環境を設定することなく、S2I イメージにコードを挿入してコンテナー化されたアプリケーションを作成できます。

Red Hat Software Collections のイメージは、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できます。

Java を使用するランタイム環境のリファレンスとして、OpenShift 用 source-to-image の概要 ドキュメントを使用できます。

S2I イメージは、Cluster Samples Operator からも入手できます。

10.2.1. OpenShift Container Platform Developer Console での S2I ビルダーイメージへのアクセス

Web コンソールの Developer Console から S2I ビルダーイメージにアクセスできます。このイメージは、ソースコードからコンテナー化されたアプリケーションをビルドするのに必要です。

手順

1. ログイン認証情報を使用して OpenShift Container Platform Web コンソールにログインします。OpenShift Container Platform Web コンソールのデフォルトビューは **Administrator** パースペクティブです。

2. パースペクティブスイッチャーを使用して、**Developer** パースペクティブに切り替えます。
3. **+Add** ビューで、**Project** ドロップダウンリストを使用して既存プロジェクトを選択するか、新規プロジェクトを作成します。
4. **Developer Catalog** タイルの **All services** をクリックします。
5. **Type** の下の **Builder Images** をクリックして、利用可能な S2I イメージを表示します。

10.2.2. 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ーイメージに挿入するビルドプロセスです。S2I は、アプリケーションのソースコードからすぐに実行できるコンテナーイメージの作成を自動化します。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 は次にコンテナーイメージを作成します。

10.2.3. 関連情報

- Cluster Samples Operator の設定
- ビルドストラテジーの使用
- Source-to-Image (S2I) プロセスのトラブルシューティング
- 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 イメージのテストについて
- Source-to-Image によるソースコードからのイメージの作成

10.3. SOURCE-TO-IMAGE イメージのカスタマイズ

デフォルトのアセンブルおよび実行スクリプトの動作を変更するには、Source-to-Image (S2I) ビルダーイメージをカスタマイズします。デフォルトのスクリプトが適切でない場合は、特定のアプリケーション要件に合わせて S2I ビルダーを調整できます。

10.3.1. イメージに埋め込まれたスクリプトの呼び出し

埋め込まれた S2I イメージスクリプトを呼び出すには、カスタムロジックとデフォルトスクリプトを実行するラッパースクリプトを作成します。サポートされているスクリプトロジックとアップグレードの互換性を維持しながら、ビルダーイメージの動作を拡張するには、これを実行する必要があります。

手順

1. ビルダー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

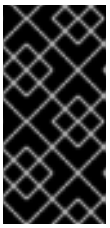
```
#!/bin/bash
echo "Before assembling"

/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
```