Red Hat OpenShift Service on AWS 4

イメージ

Red Hat OpenShift Service on AWS のイメージ

Last Updated: 2024-04-19

Red Hat OpenShift Service on AWS 4 イメージ

Red Hat OpenShift Service on AWS のイメージ

法律上の通知

概要

このドキュメントでは、イメージとイメージストリームを作成および管理する手順について説明します。さらに、テンプレートの使用方法についても説明します。

目次

第1章 イメージの概要

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

Red Hat OpenShift Service on AWS のコンテナーは OCI または Docker 形式のコンテナーの **イメージ** をベースにしています。イメージは、単一コンテナーを実行するためのすべての要件、およびそのニーズおよび機能を記述するメタデータを含むバイナリーです。

これはパッケージ化テクノロジーとして考えることができます。コンテナーは、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、Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS 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は 作成 および 管理 できます。

1.3. イメージレジストリー

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

```
registry.redhat.io
```

レジストリーには、1 つ以上のタグ付けされたイメージを持つ 1 つ以上のイメージリポジトリーのコレクションが含まれます。Red Hat は、サブスクリプションをお持ちのお客様に対して **registry.redhat.io** でレジストリーを提供しています。Red Hat OpenShift Service on AWS では、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使用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。たとえば、Red Hat OpenShift Service on AWS の Jenkins イメージが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、イメージストリーム内の他のイメージから特定のイメージを識別するリポジトリーのコンテナーイメージに適用されるラベルです。通常、タグはある種のバージョン番号を表します。たとえば、ここでは **:v3.11.59-2** がタグになります。

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

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

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

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

1.7. コンテナー

Red Hat OpenShift Service on AWS アプリケーションの基本単位はコンテナーと呼ばれます。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。このコンテナーという用語は、コンテナーイメージの実行中または一時停止している特定のインスタンスとして定義されています。

数多くのアプリケーションインスタンスは、相互のプロセス、ファイル、ネットワークなどを可視化せずに単一ホストのコンテナ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。Red Hat OpenShift Service on AWS 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。

Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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し、管理するための **docker** コマンドラインツールを置き換えるために使用できます。**podman** を使用すると、Red Hat OpenShift Service on AWS と切り離してコンテナーの実験を行うことができます。

1.8. イメージストリームを使用する理由

イメージストリームとそれに関連付けられたタグは、Red Hat OpenShift Service on AWS 内から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とそのタグを使用して、利用可能なイメージを確認し、リポジトリーのイメージが変更される場合でも必要な特定のイメージを使用

していることを確認できます。

イメージストリームには実際のイメージデータは含まれませんが、イメージリポジトリーと同様に、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。

- Red Hat OpenShift Service on AWS の統合レジストリー
- registry.redhat.io or Quay.io などの外部レジストリー
- Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS はイメージストリームタグを使用してリポジトリーにクエリーを送信し、イメージの関連付けられた ID を特定し、正確な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、Red Hat OpenShift Service on AWS 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ついてセキュリティーを設定することができます。
- クラスターレベルでイメージを読み込んだり、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のイメージに対する名前付きポインターです。イメージストリームタグはコンテナーイメージタグに似ています。

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

1.11. イメージストリームトリガー

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

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。

- 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

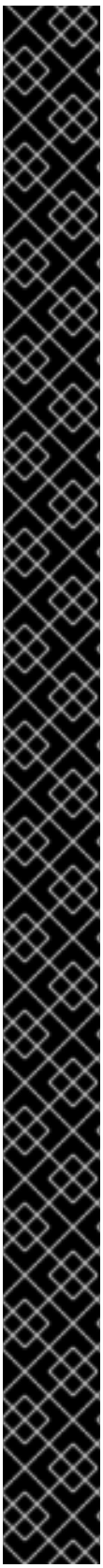
開発者は、Ruby on Rails を使用して次のことができます。

- アプリケーションを作成します。
  - データベースを設定します。
  - ウェルカムページを作成します。
  - Red Hat OpenShift Service on AWS 向け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。
- Red Hat OpenShift Service on AWS 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します。

- アプリケーションのルートを作成します。

第2章 CLUSTER SAMPLES OPERATOR の概要

openshift namespace で動作する Cluster Samples Operator は、Red Hat OpenShift Service on AWS イメージストリームと Red Hat OpenShift Service on AWS テンプレートをインストールおよび更新します。



CLUSTER SAMPLES OPERATOR がダウンサイジングされています

- Red Hat OpenShift Service on AWS 4.13 から、Cluster Samples Operator がダウンサイジングされました。Cluster Samples Operator は、非 Source-to-Image (非 S2I) イメージストリームおよびテンプレートに対する以下の更新の提供を停止します。
  - 新しいイメージストリームとテンプレート
  - CVE 更新でない限り、既存のイメージストリームとテンプレートの更新
- Cluster Samples Operator は、Red Hat OpenShift Service on AWS ライフサイクルポリシーの日付とサポートガイドライン に従って、非 S2I イメージストリームとテンプレートのサポートを提供します。
- Cluster Samples Operator は、S2I ビルダーイメージストリームとテンプレートを引き続きサポートし、更新を受け入れます。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
- Red Hat OpenShift Service on AWS 4.16 以降、Cluster Samples Operator は非 S2I イメージストリームとテンプレートの管理を停止します。要件や将来の計画については、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。さらに、list of the repositories hosting the image stream or templates を参照してください。

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 設定はクラスター全体で使用されるリソースであり、デプロイメントは **openshift-cluster-samples-operator** namespace 内に含められます。

Cluster Samples Operator のイメージには、関連付けられた Red Hat OpenShift Service on AWS リリースのイメージストリームおよびテンプレートの定義が含まれます。各サンプルが作成または更新されると、Cluster Samples Operator に Red Hat OpenShift Service on AWS のバージョンを示すアノテーションが追加されます。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

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

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

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

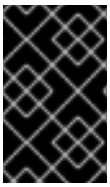
以下を含む特定の状況では、Cluster Samples Operator が **Removed** としてそれ自体をブートストラップします。

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

注記

Red Hat OpenShift Service on AWS の場合、デフォルトのイメージレジストリーは **registry.access.redhat.com** または **quay.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 Red Hat OpenShift Service on AWS 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 registry.redhat.io によってサポートされていません。Cluster Samples Operator は、ほとんどのサンプルイメージストリームおよびイメージを registry.redhat.io からプルします。

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

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

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

関連情報

- Cluster Samples Operator がインストール時に削除される場合、Cluster Samples Operator を代替レジストリーと共に使用 し、コンテンツをインポートし、サンプルを取得するために Cluster Samples Operator を **Managed** に設定できるようにします。

2.2. 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

- **oc** CLI をインストールしていること。

手順

- **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します。

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

出力例

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

関連情報

- 認証情報の詳細は、イメージプルシークレットの使用 を参照してください。

第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。

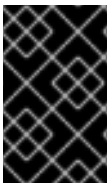
重要

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。この手順では、ご使用のネットワークとインターネットのどちらにもアクセスできるミラーホストにミラーレジストリーを配置します。

3.1. ミラーレジストリーについて

Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS 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 **Red Hat Openshift 導入用のミラーレジストリー** を使用できます。

Red Hat Quay、**Red Hat Openshift 導入用のミラーレジストリー**、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。

Red Hat Openshift 導入用のミラーレジストリー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全マシンから到達可能である必要があります。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、少なくとも Red Hat OpenShift Service on AWS の実稼働環境の可用性の条件を満たしている必要があります。

ミラーレジストリーに Red Hat OpenShift Service on AWS イメージを追加する場合は、2 つの方法で実行できます。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、Red Hat OpenShift Service on AWS でサードパーティーのレジストリーをテストしていません。

3.1.1. ミラーホストの準備

ミラーレジストリーを作成する前に、ミラーホストを準備する必要があります。

3.1.2. バイナリーのダウンロードによる OpenShift CLI のインストール

OpenShift CLI (**oc**) をインストールして、コマンドラインインターフェイスから ROSA と対話できます。**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** をインストールした場合は、それを使用して ROSA のすべてのコマンドを実行することができません。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。

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 Red Hat OpenShift Service on AWS の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 Linux 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。
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>
```

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 Red Hat OpenShift Service on AWS の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 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

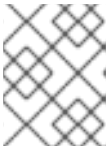
- OpenShift CLI のインストール後に、**oc** コマンドを使用して利用できます。

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

macOC 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 Red Hat OpenShift Service on AWS の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 macOS 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。



注記

macOS arm64 の場合は、**OpenShift v4 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

- 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>
```

3.2. イメージのミラーリングを可能にする認証情報の設定

Red Hat からミラーへのイメージのミラーリングを可能にするコンテナ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。

前提条件

- 使用するミラーレジストリーを設定している。

手順

インストールホストで以下の手順を実行します。

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> 1
```

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

ファイルの内容は以下の例のようになります。

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

3. ミラーレジストリーの base64 でエンコードされたユーザー名およびパスワードまたはトークンを生成します。

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

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

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

```
"auths": {
  "<mirror_registry>": { 1
    "auth": "<credentials>", 2
    "email": "you@example.com"
  }
},
```

1 **<mirror_registry>** については、レジストリードメイン名と、ミラーレジストリーがコンテンツを提供するために使用するポートをオプションで指定します。例: **registry.example.com** または **registry.example.com:8443**

2 **<credentials>** については、ミラーレジストリーの base64 でエンコードされたユーザー名およびパスワードを指定します。

ファイルは以下の例のようになります。

```
{
  "auths": {
    "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. RED HAT OPENSHIFT SERVICE ON AWS 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、Red Hat OpenShift Service on AWS イメージリポジトリーをお使いのレジストリーにミラーリングします。

前提条件

- ミラーホストはインターネットにアクセスできる。

- 使用するミラーレジストリーを設定している。
- 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

ミラーホストで以下の手順を実行します。

1. Red Hat OpenShift Service on AWS のダウンロードページ でインストールする Red Hat OpenShift Service on AWS のバージョンを確認し、Repository Tags ページで対応するタグを確認します。
2. 必要な環境変数を設定します。
   a. リリースバージョンをエクスポートします。

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

      <release_version> には、インストールする Red Hat OpenShift Service on AWS のバージョン (**4.5.4** など) に対応するタグを指定します。

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

   c. ローカルリポジトリー名をエクスポートします。

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

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

   d. ミラーリングするリポジトリーの名前をエクスポートします。

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

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

   e. パスをレジストリープルシークレットにエクスポートします。

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

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

   f. リリースミラーをエクスポートします。

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

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

g. 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> 1
```

1 **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> 1
```

1 最初のスラッシュ (/) 文字を含む完全パスを指定します。

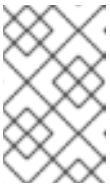
3. バージョンイメージをミラーレジストリーにミラーリングします。

a. 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。

b. 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。

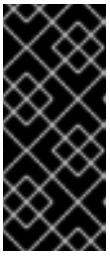


注記

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には、次のコマンドを実行してインストールプログラムを抽出し、リリースに固定します。

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



重要

選択した Red Hat OpenShift Service on AWS 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. インストーラーでプロビジョニングされるインフラストラクチャー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

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。これらについては、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

- **dedicated-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を作成している。

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. クラスターのイメージ設定オブジェクトを作成します。

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

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

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

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

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

注記

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

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

注記

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

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、Red Hat OpenShift Service on AWS 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 (設定されるイメージ) が含まれます。

config map のデータフィールドにおける各エントリーのキーの形式は、**<image_stream_name>_<image_stream_tag_name>** です。

Red Hat OpenShift Service on AWS の非接続インストールを実行すると、Cluster Samples Operator のステータスが **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または切断されている環境でサンプルを使用するには、ネットワークの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 の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。

この config map は、イメージストリームをインポートするためにミラーリングする必要があるイメージの参照情報として使用できます。

- 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。

詳細の手順については、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 について参照してください。

第4章 イメージの作成

使用可能な事前にビルドされたイメージを使用して独自のコンテナーイメージを作成する方法について確認します。このプロセスには、イメージの作成、イメージのメタデータの定義、イメージのテストおよびカスタムビルダーワークフローを使用した Red Hat OpenShift Service on AWS 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。

4.1. コンテナーのベストプラクティスについて

Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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。

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

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

タグ内の互換性の維持
独自のイメージにタグを付ける場合には、タグ内で後方互換性が維持されるようにします。たとえば、**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。

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。Red Hat OpenShift Service on AWS 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。

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。

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、これについては今後変更される可能性はあります。後のレイヤーでファイルが表示されていなくても **rm** コマンドを実行したとしても、ダウンロードするイメージの全体のサイズを縮小することになりません。そのため、**yum clean** の場合のように、可能な場合は後にレイヤーに書き込まれないように、ファイルの作成に使用したのと同じコマンドでファイルを削除することが最も適切と言えます。

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

正しい順序での命令の指定
コンテナービルダーは **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。

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。

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

デフォルトのパスワードの回避
デフォルトのパスワードは設定しないようにしてください。イメージを拡張して、デフォルトのパスワードを削除または変更するのを忘れることが多くあります。これは、実稼働環境で使用するユーザーに誰でも知っているパスワードが割り当てられると、セキュリティ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。

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、Red Hat OpenShift Service on AWS 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。

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、Red Hat

OpenShift Service on AWS 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。

Red Hat OpenShift Service on AWS でのボリュームの使用方法についての詳細は、Kubernetes ドキュメント を参照してください。

注記

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

4.1.2. Red Hat OpenShift Service on AWS 固有のガイドライン

以下は、Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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。

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 Red Hat OpenShift Service on AWS 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。

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ついてのトピックと関連しています。これにより、アプリケーションは、アプリケーションイメージを変更せずに、Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS によるコンテナーイメージの使用が改善され、開発者が Red Hat OpenShift Service on AWS 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。

Red Hat OpenShift Service on AWS 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。Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS によるコンテナーイメージの使用が改善され、開発者が Red Hat OpenShift Service on AWS 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 が使用されます。namespace は、対象のラベルを選択して使用するプロジェクトを反映するように設定されます。Red Hat OpenShift Service on AWS の場合、namespace は **io.openshift** に設定され、Kubernetes の場合、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`

変数	説明
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
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。

スクリプト	説明
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。

S2I スクリプトの例

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

assemble スクリプト:

```
#!/bin/bash

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

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

# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run スクリプト:

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

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

save-artifacts スクリプト:

```
#!/bin/bash

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

usage スクリプト:

```
#!/bin/bash

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

関連情報

- S2I イメージ作成のチュートリアル

4.4. SOURCE-TO-IMAGE イメージのテストについて

Source-to-Image (S2I) ビルダーイメージの作成者は、S2I イメージをローカルでテストして、自動テストや継続的な統合に Red Hat OpenShift Service on AWS 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、使用についての情報がコンソールに出力されるようになります。

S2I イメージのテストは、ベースのコンテナ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、Red Hat OpenShift Service on AWS 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 Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS 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。

Red Hat OpenShift Service on AWS 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

Red Hat OpenShift Service on AWS では、イメージのレジストリーの場所、イメージのレジストリーに関する認証要件、ビルドとデプロイメントの動作方法に応じて、イメージを操作し、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

次のセクションでは、Red Hat OpenShift Service on AWS のイメージストリームとそ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方法の概要と手順を説明します。

5.2.1. イメージタグ

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

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

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

Red Hat OpenShift Service on AWS 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。

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. タグのイメージストリームへの追加

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

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

手順

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

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

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

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

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

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

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

注記

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

- また、**--scheduled=true** フラグを追加して、宛先タグが定期的に更新 (再インポート) されるようにもできます。期間はシステムレベルでグローバルに設定できます。
- **--reference** フラグはインポートされないイメージストリームを作成します。このタグはソースの場所を参照しますが、これを永続的に参照します。
  タグ付けされたイメージを統合レジストリーから常に取得するように Red Hat OpenShift Service on AWS に指示する場合は、**--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またはタグ付けする際に、Red Hat OpenShift Service on AWS 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

Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS はイメージのタグに基づいてこのパラメーターを設定します。

1. タグが **latest** の場合、Red Hat OpenShift Service on AWS は **imagePullPolicy** を **Always** にデフォルト設定します。
2. それ以外の場合は、Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS 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

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

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

Docker 認証情報ファイルと関連するプルシークレットには、同じレジストリーに対して、それぞれに独自の認証情報セットがある、複数の参照を含めることができます。

config.json ファイルのサンプル

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

手順

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

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

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

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

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

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

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

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

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

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

手順

1. 委任された認証サーバーのシークレットを作成します。

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

secret/redhat-connect-sso
```

2. プライベートレジストリーのシークレットを作成します。

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

secret/private-registry
```

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を使用する理由

イメージストリームとそれに関連付けられたタグは、Red Hat OpenShift Service on AWS 内から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とそのタグを使用して、利用可能なイメージを確認し、リポジトリーのイメージが変更される場合でも必要な特定のイメージを使用していることを確認できます。

イメージストリームには実際のイメージデータは含まれませんが、イメージリポジトリーと同様に、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。

- Red Hat OpenShift Service on AWS の統合レジストリー
- registry.redhat.io or Quay.io などの外部レジストリー
- Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS はイメージストリームタグを使用してリポジトリーにクエリーを送信し、イメージの関連付けられた ID を特定し、正確な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、Red Hat OpenShift Service on AWS 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ついてセキュリティーを設定することができます。
- クラスターレベルでイメージを読み込んだり、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ついてのメタデータを取得できます。

イメージストリームにイメージをインポートまたはタグ付けすると、イメージストリームのイメージオブジェクトが Red Hat OpenShift Service on AWS 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、Red Hat OpenShift Service on AWS に付属する **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設定されています。

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

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

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

6.5. イメージストリーム変更トリガー

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

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

6.6. イメージストリームの使用

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

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。

次のデフォルトプロジェクトは、高い特権があるとみなされます (**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)。Pod セキュリティーアドミッション、セキュリティーコンテキスト制約、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。

6.6.1. イメージストリームについての情報の取得

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

手順

- イメージストリームに関する一般情報と、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
```

6.6.2. タグのイメージストリームへの追加

追加タグをイメージストリームに追加できます。

手順

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

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

以下に例を示します。

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

出力例

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

- イメージストリームに、外部コンテナ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
```

6.6.3. 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

6.6.4. 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.
```

6.6.5. 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についての詳細は、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。

6.6.6. 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.
```

このコマンドを実行すると、Red Hat OpenShift Service on AWS がこの特定のイメージストリームタグを定期的に更新するようになります。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。

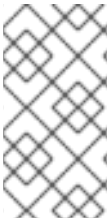
```
$ oc tag <repositiory/image> <image-name:tag>
```

6.7. 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について説明します。

6.7.1. 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中、Red Hat OpenShift Service on AWS はシークレットを取得し、リモートパーティーに提供します。

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

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

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

Docker 認証情報ファイルと関連するプルシークレットには、同じレジストリーに対して、それぞれに独自の認証情報セットがある、複数の参照を含めることができます。

config.json ファイルのサンプル

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

手順

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

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

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

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

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

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

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

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

6.7.2. 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。

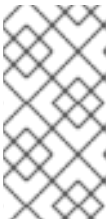
```
$ oc import-image <multiarch-image-stream-tag>  --from=
<registry>/<project_name>/<image-name> \
--import-mode='PreserveOriginal' --reference-policy=local --confirm
```

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**。

制限事項
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。

6.7.2.1. 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

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
```

6.7.2.2. 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
```

6.7.3. --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'
```

6.7.4. --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	Description

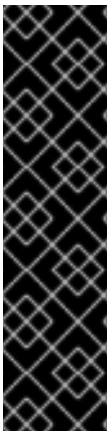
パラメーター	Description
Legacy	**--import-mode** のデフォルトオプション。指定すると、マニフェストリストが破棄され、単一のサブマニフェストがインポートされます。プラットフォームは、以下の優先順位で選択されます。 1. タグのアノテーション 2. コントロールプレーンアーキテクチャー 3. Linux/AMD64 4. 一覧の最初のマニフェスト
PreserveOriginal	指定すると、元のマニフェストが保持されます。マニフェスト一覧の場合は、マニフェストの一覧とそのすべてのサブマニフェストがインポートされます。

第7章 KUBERNETES リソースでのイメージストリームの使用

イメージストリームは、Red Hat OpenShift Service on AWS のネイティブリソースであり、**Build** リソースや **DeploymentConfigs** リソースなど、Red Hat OpenShift Service on AWS で利用可能なすべてのネイティブリソースと連携します。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。

次のデフォルトプロジェクトは、高い特権があるとみなされます (**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)。Pod セキュリティーアドミッション、セキュリティーコンテキスト制約、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

Red Hat OpenShift Service on AWS では、イメージストリームタグが新しいイメージを参照するように更新されたときに、古いイメージを使用していたリソースに新しいイメージをロールアウトするアクションを自動的に実行できます。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. RED HAT OPENSHIFT SERVICE ON AWS のリソース

Red Hat OpenShift Service on AWS のデプロイメント設定とビルド設定は、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できます。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。代わりに、Red Hat OpenShift Service on AWS のアノテーションを使用してトリガーを要求できます。

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ている場合、Red Hat OpenShift Service on AWS は、トリガーによって参照されるイメージストリームタグに現在関連付けられているイメージを使用してオブジェクトを更新しようとします。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

以下の手順でイメージレジストリーを設定します。

9.1. イメージコントローラー設定パラメーター

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

注記

DisableScheduledImport、**MaxImagesBulkImportedPerRepository**、**MaxScheduledImportsPerMinute**、**ScheduledImageImportMinimumIntervalSeconds**、**InternalRegistryHostname** などのパラメーターは設定できません。

パラメーター	説明
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。
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 についてレジストリーのホスト名をキーとして使用し、PEM エンコード証明書を値として使用します。
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。

パラメーター	説明
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を設定します。内部クラスターレジストリーの設定は含まれません。 **insecureRegistries**: 有効な TLS 証明書を持たないか、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 パラメーターが定義されると、明示的に一覧表示されない限り、**registry.redhat.io** レジストリーと **quay.io** レジストリー、およびデフォルトの 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。パラメーターを使用する場合は、Pod の失敗を防ぐために、**registry.redhat.io** レジストリーと **quay.io** レジストリー、および **internalRegistryHostname** を含むすべてのレジストリーを **allowedRegistries** 一覧に追加します。これらは、お使いの環境内のペイロードイメージで必要とされます。非接続クラスターの場合、ミラーレジストリーも追加する必要があります。

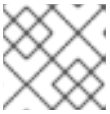
image.config.openshift.io/cluster リソースの **status** フィールドは、クラスターから観察される値を保持します。

パラメーター	説明
internalRegistryHostname	**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. 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。レジストリーへの変更が **image.config.openshift.io/cluster** CR に適用されると、Machine Config Operator (MCO) は以下の一連のアクションを実行します。

1. ノードを封鎖します
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。

手順

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

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

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

```
apiVersion: config.openshift.io/v1
kind: Image (1)
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 1
  name: cluster
  resourceVersion: "8302"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: e34555da-78a9-11e9-b92b-06d6c7da38dc
spec:
  allowedRegistriesForImport: (2)
    - domainName: quay.io
```

```
    insecure: false
  additionalTrustedCA: 3
    name: myconfigmap
  registrySources: 4
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - image-registry.openshift-image-registry.svc:5000
    - reg1.io/myrepo/myapp:latest
    insecureRegistries:
    - insecure.com
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1 **Image**: イメージの処理方法についてのクラスター全体の情報を保持します。正規名および唯一の有効な名前となるのは **cluster** です。

2 **allowedRegistriesForImport**: 標準ユーザーがイメージのインポートに使用するコンテナ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。

3 **additionalTrustedCA**: イメージストリームのインポート、Pod のイメージプル、**openshift-image-registry** プルスルー、およびビルド時に信頼される追加の認証局 (CA) が含まれる 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 についてレジストリーのホスト名をキーとして使用し、PEM 証明書を値として使用します。

4 **registrySources**: ビルドおよび Pod のイメージにアクセスする際に、コンテナーランタイムが個々のレジストリーを許可するかブロックするかを決定する設定が含まれます。**allowedRegistries** パラメーターまたは **blockedRegistries** パラメーターのいずれかを設定できますが、両方を設定することはできません。安全でないレジストリーまたはイメージの短い名前を使用するレジストリーを許可するレジストリーへのアクセスを許可するかどうかを定義することもできます。この例では、使用が許可されるレジストリーを定義する **allowedRegistries** パラメーターを使用します。安全でないレジストリー **insecure.com** も許可されます。**registrySources** パラメーターには、内部クラスターレジストリーの設定は含まれません。

注記

allowedRegistries パラメーターが定義されると、明示的に一覧表示されない限り、registry.redhat.io レジストリーと quay.io レジストリー、およびデフォルトの 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。パラメーターを使用する場合は、Pod の失敗を防ぐために、**registry.redhat.io** レジストリーと **quay.io** レジストリー、および **internalRegistryHostname** を **allowedRegistries** 一覧に追加する必要があります。これらは、お使いの環境内のペイロードイメージで必要とされます。**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。

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**

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

2. 変更が適用されたことを確認するには、ノードを一覧表示します。

```
$ oc get nodes
```

出力例

```
NAME                                         STATUS                     ROLES                  AGE   VERSION
ip-10-0-137-182.us-east-2.compute.internal   Ready,SchedulingDisabled   worker                 65m   v1.28.5
ip-10-0-139-120.us-east-2.compute.internal   Ready,SchedulingDisabled   control-plane          74m   v1.28.5
ip-10-0-176-102.us-east-2.compute.internal   Ready                      control-plane          75m   v1.28.5
ip-10-0-188-96.us-east-2.compute.internal    Ready                      worker                 65m   v1.28.5
ip-10-0-200-59.us-east-2.compute.internal    Ready                      worker                 63m   v1.28.5
ip-10-0-223-123.us-east-2.compute.internal   Ready                      control-plane          73m   v1.28.5
```

9.2.1. 特定のレジストリーの追加

image.config.openshift.io/cluster カスタムリソース (CR) を編集してイメーのプおよびプッシュアクションで許可されるレジストリーのリスト、およびオプションでレジストリー内の個別のリポジトリーを追加できます。Red Hat OpenShift Service on AWS 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。一覧に含まれていないレジストリーはブロックされます。

警告

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

手順

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

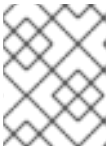
```
$ oc edit image.config.openshift.io/cluster
```

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: 1
    allowedRegistries: 2
    - 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. 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。内部クラスターレジストリーの設定は含まれません。

2. レジストリー、およびイメージのプルおよびプッシュアクションに使用するレジストリー内のリポジトリーを指定します。他のすべてのレジストリーはブロックされます。



注記

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

Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** 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、これはノードをドレイン (解放) し、その変更を適用してノードの遮断を解除します。ノードが **Ready** 状態に戻った後に、許可されるレジストリーリストは、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。

注記

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

以下に例を示します。

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

9.2.2. 特定のレジストリーのブロック

image.config.openshift.io/cluster カスタムリソース (CR) を編集してレジストリー、およびオプションでレジストリー内の個別のリポジトリーをブロックできます。Red Hat OpenShift Service on AWS 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** レジストリーおよび **quay.io** レジストリーを **blockedRegistries** リストに追加しないでください。これらは、お使いの環境内のペイロードイメージで必要とされます。

手順

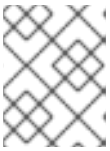
- **image.config.openshift.io/cluster** カスタムリソースを編集します。

```
$ 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: 1
    blockedRegistries: 2
    - untrusted.com
    - reg1.io/myrepo/myapp:latest
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 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。内部クラスターレジストリーの設定は含まれません。
2. レジストリー、およびオプションでイメージのプルおよびプッシュアクションに使用できないレジストリー内のリポジトリーを指定します。他のすべてのレジストリーは許可されます。



注記

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

Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** 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、これはノードをドレイン (解放) し、その変更を適用してノードの遮断を解除します。ノードが **Ready** 状態に戻った後に、ブロックされたレジストリーへの変更は各ノードの **/etc/containers/registries.conf** ファイルに表示されます。

9.2.3. 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、およびオプションでレジストリー内の個別のリポジトリーを追加できます。Red Hat OpenShift Service on AWS は、この CR への変更をクラスター内のすべてのノードに適用します。

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

警告

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

手順

- **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、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: 1
    insecureRegistries: 2
    - insecure.com
    - reg4.io/myrepo/myapp:latest
    allowedRegistries:
    - example.com
    - quay.io
    - registry.redhat.io
    - insecure.com 3
    - reg4.io/myrepo/myapp:latest
    - image-registry.openshift-image-registry.svc:5000
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

1. 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。内部クラスターレジストリーの設定は含まれません。
2. 非セキュアなレジストリーを指定します。そのレジストリーでリポジトリーを指定できます。
3. 非セキュアなレジストリーが **allowedRegistries** 一覧に含まれていることを確認します。

注記

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

Machine Config Operator (MCO) は、**image.config.openshift.io/cluster** CR でレジストリーへの変更の有無を監視し、変更を検出するとノードをドレイン (解放) し、遮断を解除します。ノードが **Ready** 状態に戻った後に、非セキュアな、およびブロックされたレジストリーへの変更は、各ノードの **/etc/containers/registries.conf** ファイルに表示されます。

9.2.4. 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。Red Hat OpenShift Service on AWS 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。

イメージの短縮名を使用して、プル仕様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。

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る可能性があります。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。

警告

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。

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。

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。

Machine Config Operator (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、これはノードをドレイン (解放) し、その変更を適用してノードの遮断を解除します。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。

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。

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
```

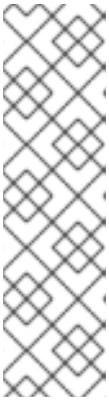
以下は、**image.config.openshift.io/cluster** CR の例になります。

```
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: (1)
    - reg1.io
    - reg2.io
    - reg3.io
    allowedRegistries: (2)
    - 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) イメージの短縮名で使用するレジストリーを指定します。セキュリティー上のリスクが発生する可能性を軽減するために、内部レジストリーまたはプライベートレジストリーでのみイメージの短縮名を使用する必要があります。

(2) **containerRuntimeSearchRegistries** に一覧表示されるレジストリーが **allowedRegistries** 一覧に含まれることを確認します。



注記

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

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

前提条件

- 認証局 (CA) は PEM でエンコードされている。

手順

openshift-config namespace で config map を作成し、 **image.config.openshift.io** カスタムリソースの **AdditionalTrustedCA** でその名前を使用して、外部レジストリーにアクセスするときに信頼する必要がある追加の CA を提供できます。

config map のキーは、この CA を信頼するポートがあるレジストリーのホスト名であり、値は各追加レジストリー CA が信頼する証明書のコンテンツです。

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: | 1
    -----BEGIN CERTIFICATE-----
    ...
    -----END CERTIFICATE-----
```

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

以下の手順で追加の CA を設定できます。

- 追加の CA を設定するには、以下を実行します。

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

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

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

9.3. 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し、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この要求を解決するように Red Hat OpenShift Service on AWS クラスターを設定する。

- 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する。

Red Hat OpenShift Service on AWS のリポジトリーミラーリングには、次の特性があります。

- 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が、Red Hat OpenShift Service on AWS クラスター内のすべてのノードの **/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します。成功すると、イメージはノードにプルされます。

リポジトリーミラーリングのセットアップは次の方法で実行できます。

- Red Hat OpenShift Service on AWS のインストール時:
  Red Hat OpenShift Service on AWS 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内側に取り込むことで、非接続環境にあるデータセンターに Red Hat OpenShift Service on AWS をインストールできます。
- Red Hat OpenShift Service on AWS のインストール後:
  Red Hat OpenShift Service on AWS のインストール時にミラーリングを設定しなかった場合は、インストール後に次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ミラーリングを設定できます。
  - **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

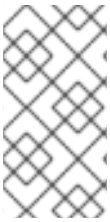
ImageContentSourcePolicy (ICSP) オブジェクトを使用してリポジトリーミラーリングを設定することは、非推奨の機能です。非推奨の機能は依然として Red Hat OpenShift Service on AWS に含まれており、引き続きサポートされますが、この製品の今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。**ImageContentSourcePolicy** オブジェクトの作成に使用した既存の YAML ファイルがある場合は、**oc adm migrate icsp** コマンドを使用して、それらのファイルを **ImageDigestMirrorSet** YAML ファイルに変換できます。詳細については、次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
- ソースリポジトリーから要求されたコンテンツを提供する各ミラーリポジトリーの個別のエントリー。

新しいクラスターの場合は、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、引き続き期待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 merge 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。これについては、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。

9.3.1. 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

- **dedicated-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。
   - Repository Mirroring in Red Hat Quay で説明されているように、Red Hat Quay でミラーリングされたリポジトリーを設定します。Red Hat Quay を使用すると、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、これらのリポジトリーを一定期間繰り返し自動的に同期することもできます。

- **skopeo** などのツールを使用して、ソース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、Red Hat Enterprise Linux (RHEL 7 または RHEL 8) システムに skopeo RPM パッケージをインストールした後、以下の例に示すように **skopeo** コマンドを使用します。

```
$ skopeo copy \
docker://registry.access.redhat.com/ubi9/ubi-minimal:latest@sha256:5cf... \
docker://example.io/example/ubi-minimal
```

  この例では、**example.io** 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、そこに **registry.access.redhat.com** から **ubi9/ubi-minimal** イメージをコピーします。レジストリーを作成した後、ソースリポジトリーに対する要求をリポジトリーにリダイレクトするように Red Hat OpenShift Service on AWS を設定できます。

2. Red Hat OpenShift Service on AWS クラスターにログインします。

3. 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 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 1
kind: ImageDigestMirrorSet 2
metadata:
  name: ubi9repo
spec:
  imageDigestMirrors: 3
  - mirrors:
    - example.io/example/ubi-minimal 4
    - example.com/example/ubi-minimal 5
    source: registry.access.redhat.com/ubi9/ubi-minimal 6
    mirrorSourcePolicy: AllowContactingSource 7
  - mirrors:
    - mirror.example.com/redhat
    source: registry.redhat.io/openshift4 8
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.com
    source: registry.redhat.io 9
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/image
    source: registry.example.com/example/myimage 10
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net
    source: registry.example.com/example 11
    mirrorSourcePolicy: AllowContactingSource
  - mirrors:
    - mirror.example.net/registry-example-com
    source: registry.example.com 12
    mirrorSourcePolicy: AllowContactingSource
```

1. この CR で使用する API を示します。これは **config.openshift.io/v1** である必要があります。
2. プルタイプに応じてオブジェクトの種類を示します。
   - **ImageDigestMirrorSet**: ダイジェスト参照イメージをプルします。
   - **ImageTagMirrorSet**: タグ参照イメージをプルします。
3. 次のいずれかのイメージプルメソッドのタイプを示します。
   - **imageDigestMirrors**: **ImageDigestMirrorSet** CR に使用します。
   - **imageTagMirrors**: **ImageTagMirrorSet** CR に使用します。
4. 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を示します。
5. オプション: 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を示します。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別のミラーを使用できます。
6. イメージプル仕様で参照されるリポジトリーである、レジストリーおよびリポジトリーソースを示します。
7. オプション: 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します。
   - **AllowContactingSource**: 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を許可します。これはデフォルトになります。
   - **NeverContactSource**: 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を防ぎます。
8. オプション: レジストリー内の namespace を示します。これにより、その namaspace 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。
9. オプション: レジストリーを示し、そのレジストリー内の任意のイメージを使用できるようにします。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。
10. 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。
11. 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。
12. ミラーレジストリー **example.net/registry-example-com/myimage@sha256:...** からイメージ **registry.example.com/myimage@sha256** をプルします。

- **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 (1)
    source: quay.io/openshift-release-dev/ocp-release (2)
  - mirrors:
    - mirror.registry.com:443/ocp/release
    source: quay.io/openshift-release-dev/ocp-v4.0-art-dev
```

(1) 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。

(2) 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。

4. 新規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をドレインします。MCO は、**ImageDigestMirrorSet** オブジェクトと **ImageContentSourcePolicy** オブジェクトのノードをドレインしません。

5. 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   v1.28.5
ip-10-0-138-148.ec2.internal   Ready                         master   11m  v1.28.5
ip-10-0-139-122.ec2.internal   Ready                         master   11m  v1.28.5
ip-10-0-147-35.ec2.internal    Ready                        worker   7m   v1.28.5
ip-10-0-153-12.ec2.internal    Ready                        worker   7m   v1.28.5
ip-10-0-154-10.ec2.internal    Ready                        master   11m  v1.28.5
```

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 が適用された **registries.conf** ファイルを表しています。最後の 2 つのエントリーは、それぞれ **digest-only** および **tag-only** とマークされています。

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" (1)

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal" (2)
    pull-from-mirror = "digest-only" (3)

  [[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.redhat.io"

  [[registry.mirror]]
```

```
    location = "mirror.example.com"
    pull-from-mirror = "digest-only"

[[registry]]
  prefix = ""
  location = "registry.redhat.io/openshift4"

  [[registry.mirror]]
    location = "mirror.example.com/redhat"
    pull-from-mirror = "digest-only"
[[registry]]
  prefix = ""
  location = "registry.access.redhat.com/ubi9/ubi-minimal"
  blocked = true (4)

  [[registry.mirror]]
    location = "example.io/example/ubi-minimal-tag"
    pull-from-mirror = "tag-only" (5)
```

1. プルスペックで参照されるリポジトリーを示します。
2. そのリポジトリーのミラーを示します。
3. ミラーからプルされた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ます。
4. 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。
5. ミラーからプルされたイメージ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...
```

リポジトリーのミラーリングの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動作方法についての以下の情報を使用して、問題のトラブルシューティングを行うことができます。

- 最初に機能するミラーは、プルされるイメージを指定するために使用されます。
- メインレジストリーは、他のミラーが機能していない場合にのみ使用されます。
- システムコンテキストによって、**Insecure** フラグがフォールバックとして使用されます。
- **/etc/containers/registries.conf** ファイルの形式が最近変更されました。現在のバージョンはバージョン 2 で、TOML 形式です。

9.3.2. 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 Red Hat OpenShift Service on AWS に引き続き含まれており、

引き続きサポートされます。ただし、この製品の今後のリリースでは削除される予定であるため、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については、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

- **dedicated-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。

第10章 テンプレートの使用

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

10.1. テンプレートについて

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

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

10.2. テンプレートのアップロード

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

手順

- 次のいずれかの方法を使用してテンプレートをアップロードします。
  - 現在のプロジェクトのテンプレートライブラリーにテンプレートをアップロードするには、JSON または YAML ファイルを以下のコマンドで渡します。

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

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

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

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

10.3. WEB コンソールを使用したアプリケーションの作成

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

手順

1. Web コンソールのナビゲーションメニューの上部にあるコンテキストセレクターから **Developer** を選択します。
2. 目的のプロジェクト内で、**+Add** をクリックします。
3. **Developer Catalog** タイルの **All services** をクリックします。

4. **Type** の下の **Builder Images** をクリックして、利用可能なビルダーイメージを表示します。

注記

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

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby"
  creationTimestamp: null
spec:
# ...
  tags:
    - name: "2.6"
      annotations:
        description: "Build and run Ruby 2.6 applications"
        iconClass: "icon-ruby"
        tags: "builder,ruby" 1
        supports: "ruby:2.6,ruby"
        version: "2.6"
# ...
```

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

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

10.4. CLI を使用してテンプレートからオブジェクトを作成する手順

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

10.4.1. ラベルの追加

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

手順

- コマンドラインからテンプレートにラベルを追加します。

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

10.4.2. パラメーターのリスト表示

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

手順

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

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

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

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

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

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

出力例

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

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

10.4.3. オブジェクトリストの生成

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

手順

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

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

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

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

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

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

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

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

3. 上書きする **<name>=<value>** の各ペアに、**-p** オプションを追加することで、ファイルに定義されたパラメーターの値を上書きできます。パラメーター参照は、テンプレートアイテム内のテキストフィールドに表示されます。
たとえば、テンプレートの以下の **POSTGRESQL_USER** および **POSTGRESQL_DATABASE** パラメーターを上書きし、カスタマイズされた環境変数の設定を出力します。

   a. テンプレートからのオブジェクトリストの作成

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

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

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

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

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

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

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

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

-

10.5. アップロードしたテンプレートの変更

すでにプロジェクトにアップロードされているテンプレートを編集できます。

手順

- すでにアップロードされているテンプレートを変更します。

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

10.6. テンプレートの作成

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

以下は、単純なテンプレートオブジェクト定義 (YAML) の例です。

```
apiVersion: template.openshift.io/v1
kind: Template
metadata:
  name: redis-template
  annotations:
    description: "Description"
    iconClass: "icon-redis"
    tags: "database,nosql"
objects:
- apiVersion: v1
  kind: Pod
  metadata:
    name: redis-master
  spec:
    containers:
    - env:
      - name: REDIS_PASSWORD
        value: ${REDIS_PASSWORD}
      image: dockerfile/redis
      name: master
      ports:
      - containerPort: 6379
        protocol: TCP
parameters:
- description: Password used for Redis authentication
  from: '[A-Z0-9]{8}'
  generate: expression
  name: REDIS_PASSWORD
labels:
  redis: master
```

10.6.1. テンプレート記述の作成

テンプレートの記述により、テンプレートの内容に関する情報を提供でき、Web コンソールでの検索時

に役立ちます。テンプレート名以外のメタデータは任意ですが、使用できると便利です。メタデータには、一般的な説明などの情報以外にタグのセットも含まれます。便利なタグにはテンプレートで使用する言語名などがあります (例: Java、PHP、Ruby)。

以下は、テンプレート記述メタデータの例です。

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: cakephp-mysql-example (1)
  annotations:
    openshift.io/display-name: "CakePHP MySQL Example (Ephemeral)" (2)
    description: >-
      An example CakePHP application with a MySQL database. For more information
      about using this template, including OpenShift considerations, see
      https://github.com/sclorg/cakephp-ex/blob/master/README.md.


      WARNING: Any data stored will be lost upon pod destruction. Only use this
      template for testing." (3)
    openshift.io/long-description: >-
      This template defines resources needed to develop a CakePHP application,
      including a build configuration, application DeploymentConfig, and
      database DeploymentConfig.  The database is stored in
      non-persistent storage, so this configuration should be used for
      experimental purposes only. (4)
    tags: "quickstart,php,cakephp" (5)
    iconClass: icon-php (6)
    openshift.io/provider-display-name: "Red Hat, Inc." (7)
    openshift.io/documentation-url: "https://github.com/sclorg/cakephp-ex" (8)
    openshift.io/support-url: "https://access.redhat.com" (9)
message: "Your admin credentials are ${ADMIN_USERNAME}:${ADMIN_PASSWORD}" (10)
```

(1) テンプレートの一意の名前。

(2) ユーザーインターフェイスで利用できるように、ユーザーに分かりやすく、簡単な名前。

(3) テンプレートの説明。デプロイされる内容、デプロイ前に知っておく必要のある注意点をユーザーが理解できるように詳細を追加します。README ファイルなど、追加情報へのリンクも追加します。パラグラフを作成するには、改行を追加できます。

(4) 追加の説明。たとえば、サービスカタログに表示されます。

(5) 検索およびグループ化を実行するためにテンプレートに関連付けられるタグ。これを指定されるカタログカテゴリーのいずれかに組み込むタグを追加します。コンソールの定数ファイルの **CATALOG_CATEGORIES** で **id** および **categoryAliases** を参照してください。

(6) Web コンソールでテンプレートと一緒に表示されるアイコン。

例10.1 利用可能なアイコン

- **icon-3scale**
- **icon-aerogear**

- **icon-amq**
- **icon-angularjs**
- **icon-ansible**
- **icon-apache**
- **icon-beaker**
- **icon-camel**
- **icon-capedwarf**
- **icon-cassandra**
- **icon-catalog-icon**
- **icon-clojure**
- **icon-codeigniter**
- **icon-cordova**
- **icon-datagrid**
- **icon-datavirt**
- **icon-debian**
- **icon-decisionserver**
- **icon-django**
- **icon-dotnet**
- **icon-drupal**
- **icon-eap**
- **icon-elastic**
- **icon-erlang**
- **icon-fedora**
- **icon-freebsd**
- **icon-git**
- **icon-github**
- **icon-gitlab**
- **icon-glassfish**
- **icon-go-gopher**

- **icon-golang**
- **icon-grails**
- **icon-hadoop**
- **icon-haproxy**
- **icon-helm**
- **icon-infinispan**
- **icon-jboss**
- **icon-jenkins**
- **icon-jetty**
- **icon-joomla**
- **icon-jruby**
- **icon-js**
- **icon-knative**
- **icon-kubevirt**
- **icon-laravel**
- **icon-load-balancer**
- **icon-mariadb**
- **icon-mediawiki**
- **icon-memcached**
- **icon-mongodb**
- **icon-mssql**
- **icon-mysql-database**
- **icon-nginx**
- **icon-nodejs**
- **icon-openjdk**
- **icon-openliberty**
- **icon-openshift**
- **icon-openstack**
- **icon-other-linux**

- **icon-other-unknown**
- **icon-perl**
- **icon-phalcon**
- **icon-php**
- **icon-play**
- **iconpostgresql**
- **icon-processserver**
- **icon-python**
- **icon-quarkus**
- **icon-rabbitmq**
- **icon-rails**
- **icon-redhat**
- **icon-redis**
- **icon-rh-integration**
- **icon-rh-spring-boot**
- **icon-rh-tomcat**
- **icon-ruby**
- **icon-scala**
- **icon-serverlessfx**
- **icon-shadowman**
- **icon-spring-boot**
- **icon-spring**
- **icon-sso**
- **icon-stackoverflow**
- **icon-suse**
- **icon-symfony**
- **icon-tomcat**
- **icon-ubuntu**
- **icon-vertx**

- **icon-wildfly**
- **icon-windows**
- **icon-wordpress**
- **icon-xamarin**
- **icon-zend**

7 テンプレートを提供する人または組織の名前

8 テンプレートに関する他のドキュメントを参照する URL

9 テンプレートに関するサポートを取得できる URL

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

10.6.2. テンプレートラベルの作成

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

以下は、テンプレートオブジェクトのラベルの例です。

```
kind: "Template"
apiVersion: "v1"
...
labels:
  template: "cakephp-mysql-example" 1
  app: "${NAME}" 2
```

1 このテンプレートから作成する全オブジェクトに適用されるラベル

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

10.6.3. テンプレートパラメーターの作成

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

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

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

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

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

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

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

以下は、明示的な値をデフォルト値として設定する例です。

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

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

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

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

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

- **[\w]{10}** は、10 桁の英字、数字、およびアンダースコアを生成します。これは PCRE 標準に準拠し、**[a-zA-Z0-9_]{10}** に相当します。
- **[\d]{10}** は 10 桁の数字を生成します。これは **[0-9]{10}** に相当します。
- **[\a]{10}** は 10 桁の英字を生成します。これは **[a-zA-Z]{10}** に相当します。
- **[\A]{10}** は 10 の句読点または記号文字を生成します。これは **[~!@#$%\^&*()\-_+={}\[\]\\|<,>.?/"';:`]{10}** に相当します。

注記

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

修飾子を含む YAML テンプレートの例

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

修飾子を含む JSON テンプレートの例

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

以下は、パラメーター定義と参照を含む完全なテンプレートの例です。

```
kind: Template
apiVersion: template.openshift.io/v1
metadata:
  name: my-template
objects:
  - kind: BuildConfig
    apiVersion: build.openshift.io/v1
    metadata:
      name: cakephp-mysql-example
      annotations:
        description: Defines how to build the application
    spec:
      source:
        type: Git
        git:
          uri: "${SOURCE_REPOSITORY_URL}" 1
          ref: "${SOURCE_REPOSITORY_REF}"
        contextDir: "${CONTEXT_DIR}"
  - kind: DeploymentConfig
    apiVersion: apps.openshift.io/v1
    metadata:
      name: frontend
    spec:
      replicas: "${{REPLICA_COUNT}}" 2
```

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

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

10.6.4. テンプレートオブジェクトリストの作成

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

以下は、オブジェクトリストの例です。

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

1 サービスの定義。 このテンプレートにより作成されます。

注記

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

10.6.5. テンプレートをバインド可能としてマーキングする

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

手順

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

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

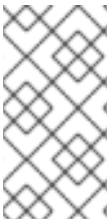
10.6.6. テンプレートオブジェクトフィールドの公開

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

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

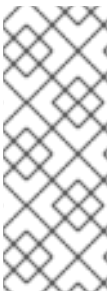
各アノテーションキーは、**bind** 応答のキーになるように、接頭辞が削除されてパススルーされます。

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



注記

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



注記

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

以下は、公開されるさまざまなオブジェクトのフィールドの例です。

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

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

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

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

手順

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

10.6.7. テンプレートの準備ができるまで待機する

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

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

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

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

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

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

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

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

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

その他の推奨事項

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

10.6.8. 既存オブジェクトからのテンプレートの作成

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

手順

- オブジェクトを YAML 形式でプロジェクトにエクスポートします。

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

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

  oc get -o yaml all に含まれるオブジェクトタイプは以下の通りです。

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

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

注記

コンテンツはクラスターやバージョンによって異なる可能性があるため、**all** エイリアスの使用は推奨されません。代わりに、必要なすべてのリソースを指定してください。

第11章 RUBY ON RAILS の使用

Ruby on Rails は Ruby で記述される Web フレームワークです。本書では、Red Hat OpenShift Service on AWS での Rails 4 の使用について説明します。

警告

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

11.1. 前提条件

- Ruby および Rails の基本知識
- Ruby 2.0.0+、Rubygems、Bundler のローカルにインストールされたバージョン
- Git の基本知識
- Red Hat OpenShift Service on AWS 4 のインスタンスの実行
- Red Hat OpenShift Service on AWS のインスタンスが実行中であり、利用可能であることを確認してください。さらに、**oc** CLI クライアントがインストールされており、コマンドがコマンドシェルからアクセスできることを確認し、メールアドレスおよびパスワードを使用してログインする際にこれを使用できるようにします。

11.2. データベースの設定

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

手順

1. データベースをインストールします。

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

2. データベースを初期化します。

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

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

3. データベースを起動します。

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

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

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

作成をしたユーザーのパスワードは作成されていない点に留意してください。

11.3. アプリケーションの作成

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

手順

1. Rails gem をインストールします。

```
$ gem install rails
```

出力例

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

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

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

3. 新規アプリケーションディレクトリーに切り替えます。

```
$ cd rails-app
```

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

```
gem 'pg'
```

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

```
$ bundle install
```

6. **pg** gem で **postgresql** データベースを使用するほか、**config/database.yml** が **postgresql** アダプターを使用していることを確認する必要があります。
config/database.yml ファイルの **default** セクションを以下のように更新するようにしてください。

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

7. アプリケーションの開発およびテスト用のデータベースを作成します。

```
$ rake db:create
```

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

11.3.1. Welcome ページの作成

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

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

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

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

手順

1. Rails ジェネレーターを実行します。

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

すべての必要なファイルが作成されます。

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

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

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

```
$ rails server
```

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

11.3.2. Red Hat OpenShift Service on AWS のアプリケーションの設定

アプリケーションが Red Hat OpenShift Service on AWS で実行されている PostgreSQL データベースサービスと通信できるようにするには、データベースサービスの作成時に、**config/database.yml** の **default** セクションを、環境変数を使用するように編集する必要があります。この環境変数は後で定義する必要があります。

手順

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

config/database YAML ファイルのサンプル

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

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

11.3.3. アプリケーションの Git への保存

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

前提条件

- git をインストールします。

手順

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

```
$ ls -1
```

出力例

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

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

```
$ git init
```

```
$ git add .
```

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

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

3. お使いの **git** リポジトリーを参照するリモートを設定します。

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

4. アプリケーションをリモートの git リポジトリーにプッシュします。

```
$ git push
```

11.4. RED HAT OPENSHIFT SERVICE ON AWS へのアプリケーションのデプロイ

アプリケーションを Red Hat OpenShift Service on AWS にデプロイできます。

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

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

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

11.4.1. データベースサービスの作成

手順

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

データベースサービスを作成するために、**oc new-app** コマンドを使用します。このコマンドには、必要な環境変数を渡す必要があります。この環境変数は、データベースコンテナー内で使用します。これらの環境変数は、ユーザー名、パスワード、およびデータベースの名前を設定するために必要です。これらの環境変数の値を任意の値に変更できます。変数は以下のようになります。

- POSTGRESQL_DATABASE
- POSTGRESQL_USER
- POSTGRESQL_PASSWORD

これらの変数を設定すると、以下を確認できます。

- 指定の名前のデータベースが存在する
- 指定の名前のユーザーが存在する
- ユーザーは指定のパスワードで指定のデータベースにアクセスできる

手順

1. データベースサービスを作成します。

```
$ oc new-app postgresql -e POSTGRESQL_DATABASE=db_name -e
POSTGRESQL_USER=username -e POSTGRESQL_PASSWORD=password
```

データベース管理者のパスワードを設定するには、直前のコマンドに以下を追加します。

```
-e POSTGRESQL_ADMIN_PASSWORD=admin_pw
```

2. 進行状況を確認します。

```
$ oc get pods --watch
```

11.4.2. フロントエンドサービスの作成

アプリケーションを Red Hat OpenShift Service on AWS にデプロイするには、アプリケーションが置かれるリポジトリーを指定する必要があります。

手順

1. フロントエンドサービスを作成し、データベースサービスの作成時に設定されたデータベース関連の環境変数を指定します。

```
$ oc new-app path/to/source/code --name=rails-app -e POSTGRESQL_USER=username -e
POSTGRESQL_PASSWORD=password -e POSTGRESQL_DATABASE=db_name -e
DATABASE_SERVICE_NAME=postgresql
```

このコマンドを実行すると、Red Hat OpenShift Service on AWS が指定された環境変数を使用して、ソースコードの取得、ビルダーのセットアップ、アプリケーションイメージのビルド、新しく作成されたイメージのデプロイを実行します。このアプリケーションには **rails-app** という名前を指定します。

2. **rails-app** デプロイメント設定の JSON ドキュメントを参照して、環境変数が追加されたかどうかを確認できます。

```
$ oc get dc rails-app -o json
```

以下のセクションが表示されるはずです。

出力例

```
env": [
    {
        "name": "POSTGRESQL_USER",
        "value": "username"
```

```
    },
    {
      "name": "POSTGRESQL_PASSWORD",
      "value": "password"
    },
    {
      "name": "POSTGRESQL_DATABASE",
      "value": "db_name"
    },
    {
      "name": "DATABASE_SERVICE_NAME",
      "value": "postgresql"
    }

  ],
```

3. ビルドプロセスを確認します。

```
$ oc logs -f build/rails-app-1
```

4. ビルドが完了したら、Red Hat OpenShift Service on AWS で実行中の Pod を確認します。

```
$ oc get pods
```

myapp-<number>-<hash> で始まる行が表示されますが、これは Red Hat OpenShift Service on AWS で実行中のアプリケーションです。

5. データベースの移行スクリプトを実行してデータベースを初期化してからでないと、アプリケーションは機能しません。これを実行する 2 種類の方法があります。

- 実行中のフロントエンドコンテナーから手動で実行する
  - **rsh** コマンドでフロントエンドコンテナーに exec を実行します。

    ```
    $ oc rsh <frontend_pod_id>
    ```

  - コンテナー内から移行を実行します。

    ```
    $ RAILS_ENV=production bundle exec rake db:migrate
    ```

    development または **test** 環境で Rails アプリケーションを実行する場合には、**RAILS_ENV** の環境変数を指定する必要はありません。

- デプロイメント前のライフサイクルフックをテンプレートに追する

11.4.3. アプリケーションのルートの作成

アプリケーションのルートを作成するためにサービスを公開できます。