# OpenShift Container Platform 4.2

## イメージ

OpenShift Container Platform 4.2 でのイメージおよびイメージストリームの作成および管理

Last Updated: 2020-06-18

# OpenShift Container Platform 4.2 イメージ

OpenShift Container Platform 4.2 でのイメージおよびイメージストリームの作成および管理

## 法律上の通知



## 概要

本書では、OpenShift Container Platform 4.2 でイメージおよびイメージストリームを作成し、管理する方法を説明します。さらに、テンプレートの使用方法についても説明します。

# 目次

# 第1章 SAMPLES OPERATOR の設定

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

**前提条件**

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

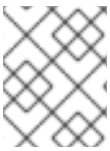
## 1.1. SAMPLES OPERATOR について

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。

Samples Operator は、**registry.redhat.io** からのイメージストリームのインポートを容易にするために、インストールプログラムによってキャプチャーされたプルシークレットを OpenShift namespace にコピーし、シークレット **samples-registry-credentials** の名前を付けます。さらに、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を容易にするために必要な 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。

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。

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



**注記**

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

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
- **samples-registry-credentials** シークレット

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

## 1.2. SAMPLES OPERATOR の設定パラメーター

サンプルリソースは以下の設定フィールドを提供します。

| パラメーター | 説明 |
| --- | --- |
| **managementState** | **Managed**: Samples Operator は設定の指示に応じてサンプルを更新します。<br><br>**Unmanaged**: Samples Operator は、その設定リソースオブジェクトおよび OpenShift namespace のイメージストリームまたはテンプレートへの更新を無視します。<br><br>**Removed**: Samples Operator は OpenShift namespace の一連の**Managed** 状態のイメージストリームおよびテンプレートを除去します。これは、クラスター管理者によって作成される新規サンプルや、省略一覧にあるサンプルを無視します。除去が完了すると、Samples Operator は **Unmanaged** 状態にあるかのように機能し、サンプルリソース、イメージストリーム、またはテンプレートに対する監視イベントを無視します。<br><br>  <br>**注記**<br><br>削除または **ManagementState** の **Removed** への設定は、イメージストリームのインポートが進行中の場合には完了しません。進行中の状態が正常に、またはエラーを出して失敗すると、削除または除去が開始されます。<br><br>シークレット、イメージストリーム、およびテンプレートの監視イベントは、削除または除去が開始されると無視されます。 |
| **samplesRegistry** | イメージのインポート元からレジストリーを上書きします。<br><br>  <br>**注記**<br><br>RHEL コンテンツの作成または更新は、**Samples Registry** が明示的に設定 (空の文字列など) されていない場合やこれが registry.redhat.io に設定される場合にプルアクセスのシークレットが有効でないと開始されません。いずれの場合も、イメージのインポートは registry.redhaいt.io の外で機能し、これには認証情報が必要になります。<br><br>RHEL コンテンツの作成または更新は、**Samples Registry** が空の文字列または registry.redhat.io 以外の値に上書きされる場合、プルシークレットの存在によって制御されることはありません。 |
| **architectures** | アーキテクチャーのタイプを選択するためのプレースホルダー。現時点では、x86 のみがサポートされています。 |
| **skippedImagestreams** | Samples Operator のインベントリーにあるものの、クラスター管理者が Operator に無視させるか、または管理させないようにするイメージストリーム。このパラメーターにイメージストリーム名の一覧を追加できます。例: **["httpd","perl"]** |
| **skippedTemplates** | Samples Operator のインベントリーにあるものの、クラスター管理者が Operator に無視させるか、または管理させないようにするテンプレート。 |

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

### 1.2.1. 設定の制限

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

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

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

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

### 1.2.2. 条件

サンプルリソースには以下の条件とそのステータスが適用されます。

| 条件 | 説明 |
|---|---|
| **SamplesExists** | サンプルが OpenShift namespace に作成されていることを示します。 |
| **ImageChangesInProgress** | **True** イメージストリームが作成または更新される場合に、タグ仕様の生成およびタグステータスの生成のすべてが一致する訳ではありません。<br><br>**False** すべての生成が一致するか、または修復不可能なエラーがインポート時に発生した場合、最後に表示されるエラーは message フィールドに置かれます。保留中のイメージストリームの一覧は reason フィールドに置かれます。 |
| **ImportCredentialsExist** | **samples-registry-credentials** シークレットが OpenShift namespace にコピーされます。 |
| **ConfigurationValid** | 前述の制限された変更のいずれかが送信されるかどうかに応じて **True** または **false** になります。 |
| **RemovePending** | **Management State: Removed** 設定が保留中であるが、進行中のイメージストリームの完了を待機していることを示すインジケーター。 |
| **ImportImageErrorsExist** | イメージストリームのタグのいずれかについて、イメージストリームでイメージインポートフェーズにエラーがあったことを示すインジケーター。<br><br>**True** エラーが発生した場合。エラーのあるイメージストリームの一覧は reason フィールドに置かれます。報告されるそれぞれのエラーの詳細は message フィールドに置かれます。 |

| 条件 | 説明 |
| --- | --- |
| **MigrationInProgress** | **True** バージョンが現在のサンプルセットのインストールに使用した Samples Operator バージョンと異なることを Samples Operator が検知する場合。 |

## 1.3. SAMPLES OPERATOR 設定へのアクセス

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

**前提条件**

- **oc** として知られる OpenShift コマンドラインインターフェース (CLI) のインストール。

**手順**

- Samples Operator 設定にアクセスします。

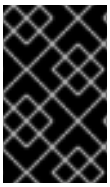
```
$ oc get configs.samples.operator.openshift.io/cluster -o yaml
```

Samples Operator 設定は以下の例のようになります。

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

# 第2章 代替レジストリーでの 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 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。

## 2.1. ミラーレジストリーについて

インストールプログラムを生成するために必要な OpenShift Container Platform レジストリーおよびイメージのコンテンツをミラーリングできます。

ミラーレジストリーは、ネットワークが制限された環境でインストールを実行するために必要となる主要なコンポーネントです。このミラーは、インターネットと制限されたネットワークの両方にアクセスできる bastion ホストで、または制限に対応する他の方法を使用して作成できます。

OpenShift Container Platform がリリースペイロードの整合性を検証する方法により、ローカルレジストリーのイメージ参照は Quay.io 上で Red Hat によってホストされるものと同一になります。インストールのブートストラッププロセスでは、プルされるリポジトリーの種類を問わず、イメージには同じダイジェストがなければなりません。リリースペイロードが同一になるようにするには、イメージをローカルリポジトリーにミラーリングします。

### 2.1.1. ミラーホストの準備

ミラーレジストリーを作成する前に、ミラーホストを準備する必要があります。

### 2.1.2. CLI のインストール

コマンドラインインターフェースを使用して OpenShift Container Platform と対話するために CLI をインストールすることができます。

**重要**

以前のバージョンの **oc** をインストールしている場合、これを使用して OpenShift Container Platform 4.2 のすべてのコマンドを実行することはできません。新規バージョンの **oc**をダウンロードし、インストールします。

**手順**

1. Red Hat OpenShift Cluster Manager サイトの Infrastructure Provider ページから、選択するインストールタイプのページに移動し、**Download Command-line Tools**をクリックします。
2. オペレーティングシステムおよびアーキテクチャーのフォルダーをクリックしてから、圧縮されたファイルをクリックします。

   **注記**

   **oc** は Linux、Windows、または macOS にインストールできます。
3. ファイルをファイルシステムに保存します。

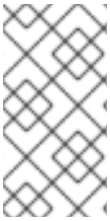
4. 圧縮ファイルを展開します。
5. これを **PATH** にあるディレクトリーに配置します。

CLI のインストール後は、**oc** コマンドを使用して利用できます。

```
$ oc <command>
```

## 2.2. ミラーレジストリーの作成

OpenShift Container Platform のインストールに必要なミラーリングされたコンテンツをホストするためにレジストリーを作成します。



**注記**

以下の手順では、**/opt/registry** フォルダーにデータを保存し、**podman** コンテナーで実行される単純なレジストリーを作成します。Red Hat Quay などの異なるレジストリーソリューションを使用できます。以下の手順で、レジストリーが正常に機能することを確認します。

**前提条件**

- レジストリーホストとして使用する Red Hat Enterprise Linux (RHEL) サーバーがネットワーク上にある。
- レジストリーホストはインターネットにアクセスできる。

**手順**

1. 必要なパッケージをインストールします。

   ```
   # yum -y install podman httpd-tools
   ```

   **podman** パッケージは、レジストリーを実行する際に使用するコンテナーパッケージを提供します。**httpd-tools** パッケージは、ユーザーの作成に使用する **htpasswd** ユーティリティーを提供します。

2. レジストリーのフォルダーを作成します。

   ```
   # mkdir -p /opt/registry/{auth,certs,data}
   ```

   これらのフォルダーはレジストリーコンテナー内にマウントされます。

3. レジストリーの証明書を指定します。既存の信頼される認証局がない場合は、自己署名の証明書を生成することができます。

   ```
   $ cd /opt/registry/certs
   # openssl req -newkey rsa:4096 -nodes -sha256 -keyout domain.key -x509 -days 365 -out
   domain.crt
   ```

   プロンプト時に、証明書に必要な値を指定します。

| | |
|---|---|
| 国名 (2 文字コード) | 場所として 2 文字の ISO 国コードを指定します。ISO 3166 country codes 標準を参照してください。 |
| State or Province Name (正式名) | 都道府県名の正式名を入力します。 |
| Locality Name (例: 市) | 市の名前を入力します。 |
| Organization Name (例: 会社) | 会社名を入力します。 |
| Organizational Unit Name (例: セクション) | 部門名を入力します。 |
| Common Name (例: 本人の名前またはサーバーのホスト名) | レジストリーホストのホスト名を入力します。ホスト名が DNS にあり、予想される IP アドレスに解決されていることを確認します。 |
| Email Address | メールアドレスを入力します。詳細は、OpenSSL ドキュメントの req の説明を参照してください。 |

4. **bcrpt** 形式を使用するレジストリー用のユーザー名およびパスワードを生成します。

```
# htpasswd -bBc /opt/registry/auth/htpasswd <user_name> <password> 1
```

1 **<user_name>** および **<password>** をユーザー名およびパスワードに置き換えます。

5. レジストリーをホストする **mirror-registry** コンテナーを作成します。

```
# podman run --name mirror-registry -p <local_registry_host_port>:5000 \ 1
    -v /opt/registry/data:/var/lib/registry:z \
    -v /opt/registry/auth:/auth:z \
    -e "REGISTRY_AUTH=htpasswd" \
    -e "REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" \
    -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
    -v /opt/registry/certs:/certs:z \
    -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/domain.crt \
    -e REGISTRY_HTTP_TLS_KEY=/certs/domain.key \
    -e REGISTRY_COMPATIBILITY_SCHEMA1_ENABLED=true \
    -d docker.io/library/registry:2
```

1 **<local_registry_host_port>** については、ミラーレジストリーがコンテンツの送信に使用するポートを指定します。

6. レジストリーに必要なポートを開きます。

```
# firewall-cmd --add-port=<local_registry_host_port>/tcp --zone=internal --permanent (1)
# firewall-cmd --add-port=<local_registry_host_port>/tcp --zone=public   --permanent (2)
# firewall-cmd --reload
```

1 2 **<local_registry_host_port>** については、ミラーレジストリーがコンテンツの送信に使用するポートを指定します。

7. 信頼された証明書の一覧に自己署名証明書を追加します。

```
# cp /opt/registry/certs/domain.crt /etc/pki/ca-trust/source/anchors/
# update-ca-trust
```

ミラープロセス中にレジストリーにログインするには、証明書を信頼する必要があります。

8. レジストリーが使用できることを確認します。

```
$ curl -u <user_name>:<password> -k https://<local_registry_host_name>:
<local_registry_host_port>/v2/_catalog (1)

{"repositories":[]}
```

1 **<user_name>** および **<password>**については、レジストリーにユーザー名とパスワードを指定します。**<local_registry_host_name>**については、**registry.example.com** などのように、証明書に指定したレジストリードメイン名を指定します。**<local_registry_host_port>** については、ミラーレジストリーがコンテンツの送信に使用するポートを指定します。

コマンドの出力に空のリポジトリーが表示される場合、レジストリーは利用可能な状態です。

## 2.3. レジストリーのプルシークレットへの追加

OpenShift Container Platform クラスターをネットワークが制限された環境にインストールする前に、OpenShift Container Platform クラスターのプルシークレットを変更してローカルレジストリーを記述します。

### 前提条件

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

### 手順

bastion ホストで以下の手順を実行します。

1. Red Hat OpenShift Cluster Manager サイトの Pull Secret ページから **registry.redhat.io** プルシークレットをダウンロードします。

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

```
$ echo -n '<user_name>:<password>' | base64 -w0 (1)

BGVtbYk3ZHAtqXs=
```

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

3. JSON 形式でプルシークレットのコピーを作成します。

```
$ cat ./pull-secret.text | jq .  > <path>/<pull-secret-file> (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"
    }
  }
}
```

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

```
  "auths": {
...
    "<local_registry_host_name>:<local_registry_host_port>": { (1)
      "auth": "<credentials>", (2)
      "email": "you@example.com"
  },
...
```

(1) **<local_registry_host_name>** については、証明書に指定したレジストリードメイン名を指定し、**<local_registry_host_port>** については、ミラーレジストリーがコンテンツを送るために使用するポートを指定します。

2 **<credentials>** については、生成した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": {
    "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"
    },
    "<local_registry_host_name>:<local_registry_host_port>": {
      "auth": "<credentials>",
      "email": "you@example.com"
    },
    "registry.redhat.io": {
      "auth": "NTE3Njg5Nj...",
      "email": "you@example.com"
    }
  }
}
```

## 2.4. OPENSHIFT CONTAINER PLATFORM イメージリポジトリーのミラーリング

クラスターのインストールまたはアップグレードに使用する OpenShift Container Platform イメージリポジトリーをミラーリングします。

### 前提条件

- ネットワークが制限された環境で使用するミラーレジストリーを設定し、設定した証明書および認証情報にアクセスできること。
- Red Hat OpenShift Cluster Manager のサイトの「Pull Secret」ページからプルシークレットをダウンロードしており、ミラーリポジトリーに認証を組み込むようにこれを変更している。

### 手順

bastion ホストで以下の手順を実行します。

1. 「OpenShift Container Platform ダウンロード」ページを確認し、インストールする必要のある OpenShift Container Platform のバージョンを判別します。
2. 必要な環境変数を設定します。

```
$ export OCP_RELEASE=<release_version> 1
```

```
$ export LOCAL_REGISTRY='<local_registry_host_name>:<local_registry_host_port>' 2
$ export LOCAL_REPOSITORY='<repository_name>' 3
$ export PRODUCT_REPO='openshift-release-dev' 4
$ export LOCAL_SECRET_JSON='<path_to_pull_secret>' 5
$ export RELEASE_NAME="ocp-release" 6
```

1. **<release_version>** については、インストールする OpenShift Container Platform のバージョン番号 (例: **4.2.0**) を指定します。
2. **<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。
3. **<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。
4. ミラーリングするリポジトリー。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。
5. **<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およびファイル名の絶対パスを指定します。
6. リリースミラー。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。

3. リポジトリーをミラーリングします。

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

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。

4. 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。

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

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

**重要**

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

インターネット接続のあるマシンで、このステップを実行する必要があります。

## 2.5. 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 SAMPLES OPERATOR イメージストリームの使用

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

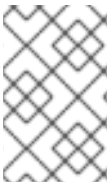
**重要**

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

Sample Operator 設定ファイルの **samplesRegistry** フィールドの registry.redhat.io への設定は、これはすでに Jenkins イメージおよびイメージストリーム以外のすべての registry.redhat.io に送信されているため不要になります。また、Jenkins イメージストリームのインストールペイロードも破損します。

Samples Operator は Jenkins イメージストリームについて以下のレジストリーの使用を防ぎます。

- docker.io
- registry.redhat.io
- registry.access.redhat.com
- quay.io.



**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、Samples Operator によって管理されません。これらについては、この手順で扱いません。

**前提条件**

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

**手順**

1. ミラーリングする特定のイメージストリームのイメージにアクセスします。

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

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

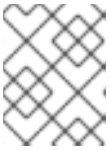
```
$ oc image mirror registry.redhat.io/rhscl/ruby-25-rhel7:latest ${MIRROR_ADDR}/rhscl/ruby-25-rhel7:latest
```

3. 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 create configmap registry-config --from-
file=${MIRROR_ADDR_HOSTNAME}..5000=$path/ca.crt -n openshift-config
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-config"}}}' --type=merge
```

4. 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。

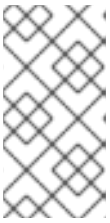
```
$ oc get configs.samples.operator.openshift.io -n openshift-cluster-samples-operator
```



**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。

5. 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、Samples Operator を Samples Operator 設定オブジェクトの **Removed** に設定します。



**注記**

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

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

# 第3章 コンテナー、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。

## 3.1. イメージ

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

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

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

## 3.2. コンテナー

OpenShift Container Platform アプリケーションの基本的な単位は **コンテナー** と呼ばれています。Linux コンテナーテクノロジーは、指定されたリソースのみとの対話に制限されるように、実行中のプロセスを分離する軽量なメカニズムです。このコンテナーという用語は、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ースを開発しました。さらに最近では、Open Container Initiative により、コンテナー形式およびコンテナーランタイムのオープン標準が策定されています。OpenShift Container Platform および Kubernetes は複数ホストのインストール間で OCI および Docker 形式のコンテナーのオーケストレーションを実行する機能を追加しています。

OpenShift Container Platform を使用する際にコンテナーランタイムと直接対話することはありませんが、それらの OpenShift Container Platform における役割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、OpenShift Container Platform と切り離してコンテナーの実験を行うことができます。

## 3.3. イメージレジストリー

**イメージレジストリー** は、コンテナーイメージを保管し、提供するコンテナーサーバーです。以下は例になります。

```
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。また、OpenShift Container Platform はカスタムコンテナーイメージを管理するための独自の内部レジストリーも提供しています。

## 3.4. イメージリポジトリー

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

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

## 3.5. イメージタグ

イメージタグは、イメージストリーム内の他のイメージから特定のイメージを識別するリポジトリ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。

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

## 3.6. イメージ ID

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

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

## 3.7. イメージストリームの使用

イメージストリームおよびその関連付けられたタグは、OpenShift Container Platform 内でコンテナ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を提示します。

ビルドおよびデプロイメントは、イメージストリームで新規イメージの追加時の通知の有無を監視するように設定し、ビルドまたはデプロイメントをそれぞれ実行してこれに対応す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。

ただし、デプロイメントまたはビルドで使用されるイメージストリームタグが更新されない場合は、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おそらくは既知の正常なイメージをそのまま使用します。

ソースイメージは以下のいずれかに保存できます。

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

(ビルドまたはデプロイメント設定などの) イメージストリームタグを参照するオブジェクトを定義する場合は、Docker リポジトリーではなく、イメージストリームタグを参照します。アプリケーションのビルドまたはデプロイ時に、OpenShift Container Platform はイメージストリームタグを使用して Docker 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、OpenShift Container Platform には他のリソースの汎用トリガーがあります (Kubernetes オブジェクトなど)。
- 定期的な再インポートを実行するためにタグにマークを付けることができます。ソースイメージが変更された場合、その変更は認識され、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のいずれかに応じて、ビルドおよび/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ても、イメージストリームタグはイメージの既知の正常なバージョンを参照したままになり、アプリケーションに予期しない障害が発生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のセキュリティー設定を行うことができます。
- クラスターレベルでイメージを読み込んだり、一覧表示するパーミッションのないユーザーでも、イメージストリームを使用してプロジェクトでタグ付けされたイメージを取得できます。

### 3.7.1. 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。

## 3.8. 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。

## 3.9. イメージストリームトリガー

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

## 3.10. 追加リソース

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

# 第4章 イメージの作成

使用可能な事前にビルドされたイメージを使用して独自のコンテナ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 OpenShift Container Platform で使用できるイメージの作成のベストプラクティスを理解することが含まれます。イメージを作成した後には、これを内部レジストリーにプッシュできます。

## 4.1. コンテナーのベストプラクティスについて

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

### 4.1.1. コンテナーイメージの一般的なガイドライン

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

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

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

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

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

**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** を使用するはずです。**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、以下で説明するように望ましいアクションではありません。

たとえば、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。コンテナーを起動する (たとえば、**podman run -i** を使用する) 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。ここで、**CTRL+C** でコンテナーを強制終了する必要があるとします。ラッパースクリプトが exec を使用してサーバープロセスを開始した場合、**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、何も生じなかったかのようにプロセスは実行し続けます。

また、コンテナー内で実行すると、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。

その他の影響については、Docker and the PID 1 zombie reaping problem のブログ記事を参照してください。また、PID 1 および **init** システムの詳細については、Demystifying the init system (PID 1) のブログ記事も参照してください。

**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する必要があります。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** コマンドを実行することを強く推奨します。

**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て **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も表示されます。
- 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ことが最善の策です。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るようにする必要があります。

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

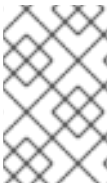
するには、**podman exec** または **docker exec** コマンドを使用できます。または、**oc exec** コマンドまたは **oc rsh** コマンドを使用して、OpenShift Container Platform クラスターで実行中のコンテナーにアクセスできます。イメージで **sshd** をインストールし、実行すると、攻撃の経路が増え、セキュリティー修正が必要になります。

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

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

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

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



**注記**

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

**追加リソース**

- Docker ドキュメント - Best practices for writing Dockerfiles
- Project Atomic ドキュメント - Guidance for Container Image Authors

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

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

**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。

たとえば、この Python イメージは S2I スクリプトを定義して、Python アプリケーションのさまざまなバージョンをビルドします。

**任意のユーザー ID のサポート**
デフォルトでは OpenShift Container Platform は、任意に割り当てられたユーザー ID を使用してコンテナーを実行します。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可能性のあ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
```

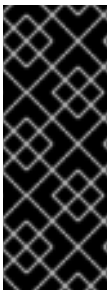
コンテナーユーザーは常に root グループのメンバーであるため、コンテナーユーザーはこれらのファイルに対する読み取り、書き込みが可能です。

**警告**

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

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

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



**重要**

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

**イメージ間通信でのサービスの使用**
データの保存や取得のためにデータベースイメージにアクセスする必要のある Web フロントエンドイメージなど、別のイメージが提供するサービスとイメージが通信する場合には、イメージは OpenShift Container Platform サービスを使用する必要があります。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。

**共通のライブラリーの提供**
サードパーティーが提供するアプリケーションコードの実行を目的とするイメージの場合は、プラットフォーム用として共通に使用されるライブラリーをイメージに含めるようにしてください。とくに、プラットフォームで使用する共通のデータベース用のデータベースドライバーを設定してください。たとえば、Java フレームワークイメージを作成する場合に、MySQL や PostgreSQL には JDBC ドライ

バーを設定します。このように設定することで、アプリケーションのアセンブリー時に共通の依存関係をダウンロードする必要がなくなり、アプリケーションイメージのビルドがスピードアップします。また、すべての依存関係の要件を満たすためのアプリケーション開発者の作業が簡素化されます。

**設定での環境変数の使用**
イメージのユーザーは、ダウンストリームイメージをイメージに基づいて作成しなくても、イメージの設定が行えるようにしてください。つまり、ランタイム設定は環境変数を使用して処理される必要があります。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する必要がある点で、これはイメージ間の通信でのサービスの使用についてのトピックと関連しています。これにより、アプリケーションは、アプリケーションイメージを変更せずに、OpenShift Container Platform 環境に定義されているデータソースサービスを動的に使用できます。

さらに、コンテナーの **cgroups** 設定を確認して、調整を行う必要があります。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する必要があります。

コンテナーの **cgroup** クォータを管理する方法については、以下の資料を参照してください。

- ブログ記事 - Resource management in Docker
- Docker ドキュメント - Runtime metrics
- ブログ記事 - Memory inside Linux containers

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

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

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

**ロギング**
すべてのロギングを標準出力に送信することが推奨されます。OpenShift Container Platform はコンテナーから標準出力を収集し、表示が可能な中央ロギングサービスに送信します。ログコンテンツを分離する必要がある場合には、出力のプレフィックス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。

**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。

**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。

**追加リソース**

- Docker basics
- Dockerfile reference
- Project Atomic Guidance for Container Image Authors

## 4.2. イメージへのメタデータの組み込み

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

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

### 4.2.1. イメージメタデータの定義

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

**LABEL** 命令に関する詳細は、Docker ドキュメントを参照してください。

ラベル名には、通常 namespace を指定する必要があります。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してください。OpenShift Container Platform の場合は、namespace は **io.openshift** に、Kubernetes の場合は、namespace は **io.k8s** に設定してください。

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、カンマ区切りの文字列値の一覧として表現されているタグの一覧が含まれます。タグを使用して、コンテナ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。<br><br>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一覧を指定します。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。<br><br>LABEL io.openshift.wants mongodb,redis |
| **io.k8s.description** | このラベルは、コンテナーイメージの利用者に、このイメージが提供するサービスや機能に関する詳細情報を提供するのに使用できます。UI は、この説明とコンテナーイメージ名を使用して、人間が解読しやすい情報をエンドユーザーに提供します。<br><br>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旨を示します。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基本的に **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。<br><br>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します。この値は、Kubernetes の数量と互換性がある必要があります。<br><br>LABEL io.openshift.min-memory 8Gi<br>LABEL io.openshift.min-cpu 4 |

## 4.3. S2I イメージのテスト

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

S2I ビルドを正常に実行するには、S2I に **assemble** と **run** スクリプトが必要です。S2I 外のコンテナーイメージを実行した場合に、**save-artifacts** スクリプトがあると、ビルドのアーティファクトが再利用され、**usage** スクリプトがあると、使用に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。

### 4.3.1. テスト要件について

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

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

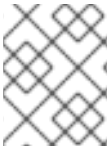
S2I は、アプリケーションのソースコードおよびビルダーイメージを統合します。これをテストするには、ソースが実行可能なコンテナーイメージに変換されることを検証するためのサンプルアプリケーションのソースが必要です。サンプルアプリケーションは単純なものである必要がありますが、**assemble** および **run** スクリプトの重要な手順を実行できる必要があります。

### 4.3.2. 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。

### 4.3.3. 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
```

### 4.3.4. 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。

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

新しい S2I ビルダーイメージを構成する **Dockerfile** と他のアーティファクトが準備できたら、それらを git リポジトリーに配置して、OpenShift Container Platform を使用し、イメージをビルドしてプッシュします。その後は、お使いのリポジトリーを参照する Docker ビルドを定義することのみが必要になります。

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

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

# 第5章 イメージの管理

## 5.1. イメージの管理の概要

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

### 5.1.1. イメージの概要

イメージストリームは、タグで識別される数多く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を監視することにより、ビルドおよびデプロイメントは新規イメージの追加または変更時に通知を受信し、それぞれビルドまたはデプロイメントを実行してこれに対応します。

## 5.2. イメージのタグ付け

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

### 5.2.1. イメージタグ

イメージタグは、イメージストリーム内の他のイメージから特定のイメージを識別するリポジトリ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。

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

### 5.2.2. イメージタグの規則

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

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

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

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

**表5.1 イメージタグの名前付け規則**

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

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

### 5.2.3. タグのイメージストリームへの追加

OpenShift Container Platform のイメージストリームは、タグで識別される 0 個以上のコンテナ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_tag 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。

**追跡** 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はありません。

- 宛先タグがソースタグの変更時に常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、**追跡**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、インポートされていないイメージストリームタグを作成します。このタグはソースの場所を参照しますが、これを永続的に参照します。
  統合レジストリーのタグ付けされたイメージを常にフェッチするように OpenShift に指示するには、**--reference-policy=local** を使用します。レジストリーはプルスルー (pull-through) 機能を使用してイメージをクライアントに提供します。デフォルトで、イメージ Blob はレジストリーによってローカルにミラーリングされます。その結果、それらが次回必要となる場合により迅速にプルされます。また、このフラグは **--insecure-registry** をコンテナーランタイムに指定しなくても、イメージストリームに非セキュアなアノテーションがあるか、またはタグに非セキュアなインポートポリシーがある限り、非セキュアなレジストリーからのプルを許可します。

### 5.2.4. タグのイメージストリームからの削除

タグをイメージストリームから削除できます。

**手順**

タグをイメージストリームから完全に削除するには、以下を実行します。

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

または、以下を実行します。

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

### 5.2.5. イメージストリームでのイメージの参照

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

**表5.2 イメージストリームの参照タイプ**

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

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

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

これは、**ImageStreamImage** オブジェクトが、イメージをイメージストリームにインポートしたり、タグ付けしたりする際に OpenShift Container Platform に自動的に作成されるためです。イメージストリームを作成するために使用するイメージストリーム定義に **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>
```

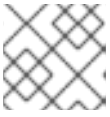
- 所定のイメージストリームおよびイメージの **sha** ID のイメージを参照するには、**ImageStreamImage** を使用します。

```
<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@sha256:3a335d7d8a452970c5b4054ad7118ff134b3a6b50a2bb6d0c07c746e8986b2
8e
```

### 5.2.6. 追加情報

- CentOS イメージストリームのイメージストリーム定義の例。

## 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含まれます。イメージを作成してレジストリーにプッシュすると、このイメージを Pod で参照できます。

### 5.3.1. イメージプルポリシーの概要

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

**表5.3 imagePullPolicy の値**

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

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

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

## 5.4. イメージプルシークレットの使用

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

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

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

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

**手順**

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

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

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

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

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

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

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

OpenShift Container Platform の内部レジストリーにないセキュリティー保護されたコンテナ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。

**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
```

- セキュリティー保護されたレジストリーの Docker 認証情報がない場合は、以下を実行してシークレットを作成できます。

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

- Pod のイメージのプルにシークレットを使用するには、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。**default** はデフォルトのサービスアカウントです。

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

- ビルドイメージのプッシュおよびプルにシークレットを使用するには、シークレットは Pod 内でマウント可能である必要があります。以下でこれを実行できます。

```
$ oc secrets link builder <pull_secret_name>
```

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

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

**手順**

1. 委任された認証サーバーのシークレットを作成します。

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

secret/redhat-connect-sso
```

2. プライベートレジストリーのシークレットを作成します。

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

secret/private-registry
```

### 5.4.3. グローバルクラスターのプルシークレットの更新

クラスターのグローバルプルシークレットを更新できます。

**警告**

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

**前提条件**

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

**手順**

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

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

1 新規プルシークレットファイルへのパスを指定します。

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

# 第6章 イメージストリームの管理

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

## 6.1. イメージストリームの使用

イメージストリームおよびその関連付けられたタグは、OpenShift Container Platform 内でコンテナーイメージを参照するための抽象化を提供します。イメージストリーム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を提示します。

ビルドおよびデプロイメントは、イメージストリームで新規イメージの追加時の通知の有無を監視するように設定し、ビルドまたはデプロイメントをそれぞれ実行してこれに対応す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。

ただし、デプロイメントまたはビルドで使用されるイメージストリームタグが更新されない場合は、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おそらくは既知の正常なイメージをそのまま使用します。

ソースイメージは以下のいずれかに保存できます。

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

(ビルドまたはデプロイメント設定などの) イメージストリームタグを参照するオブジェクトを定義する場合は、Docker リポジトリーではなく、イメージストリームタグを参照します。アプリケーションのビルドまたはデプロイ時に、OpenShift Container Platform はイメージストリームタグを使用して Docker 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、OpenShift Container Platform には他のリソースの汎用トリガーがあります (Kubernetes オブジェクトなど)。
- 定期的な再インポートを実行するためにタグにマークを付けることができます。ソースイメージが変更された場合、その変更は認識され、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のいずれかに応じて、ビルドおよび/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ても、イメージストリームタグはイメージの既知の正常なバージョンを参照したままになり、アプリケーションに予期しない障害が発生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のセキュリティー設定を行うことができます。
- クラスターレベルでイメージを読み込んだり、一覧表示するパーミッションのないユーザーでも、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

イメージストリームオブジェクトには以下の要素が含まれます。

**イメージストリームオブジェクト定義**

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

(1) イメージストリームの名前です。

(2) 新規イメージをこのイメージストリームで追加/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 このイメージストリームタグが現在参照している 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。

## 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内から特定のイメージ ID を参照します。

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

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

イメージストリームイメージはリポジトリーからのイメージストリーム名およびイメージ 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>
```

イメージストリーム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** として省略されることが多くあります。イメージストリームタグは、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または取得するために使用されます。

イメージストリームタグは、ローカルの、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 番目の位置などに置かれます。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、イメージストリームオブジェクトからのものです。

**履歴内に 2 つのイメージを持つイメージストリームタグ**

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

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

イメージストリームタグは**永続**タグまたは**追跡**タグにすることができます。

- **永続タグ** 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。
- **追跡タグ** は別のイメージストリームタグをフォローする参照タグで、シンボリックリンクのように、フォローするイメージを変更するために今後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ない点に注意してください。
  たとえば、OpenShift Container Platform に同梱される **latest** イメージストリームタグは追跡タグです。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、いつでも **v3.11** に変更される可能性があります。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**

追跡タグは単一のイメージストリームに制限され、他のイメージストリームを参照できません。

各自のニーズに合わせて独自のイメージストリームタグを作成できます。

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

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

たとえば、上記のイメージストリーム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、アップストリームイメージの新規バージョンが利用可能になると自動的に起動します。

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

## 6.6. イメージストリームのマッピング

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

**注記**

イメージストリームのマッピング設定は高度な機能です。

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

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

**イメージストリームマッピングオブジェクト定義**

```
apiVersion: v1
kind: ImageStreamMapping
metadata:
  creationTimestamp: null
  name: origin-ruby-sample
  namespace: test
tag: latest
image:
  dockerImageLayers:
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ee1dd2cb6df21971f4af6de0f1d7782b81fb63156801cfde2bb47b4247c23c29
    size: 196634330
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
    size: 0
  - name: sha256:ca062656bff07f18bff46be00f40cfbb069687ec124ac0aa038fd676cfaea092
    size: 177723024
  - name: sha256:63d529c59c92843c395befd065de516ee9ed4995549f8218eac6ff088bfa6b6e
    size: 55679776
  - name: sha256:92114219a04977b5563d7dff71ec4caa3a37a15b266ce42ee8f43dba9798c966
    size: 11939149
  dockerImageMetadata:
    Architecture: amd64
    Config:
      Cmd:
      - /usr/libexec/s2i/run
      Entrypoint:
      - container-entrypoint
      Env:
      - RACK_ENV=production
      - OPENSHIFT_BUILD_NAMESPACE=test
      - OPENSHIFT_BUILD_SOURCE=https://github.com/openshift/ruby-hello-world.git
      - EXAMPLE=sample-app
```

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

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

## 6.7. イメージストリームの使用

以下のセクションでは、イメージストリームおよびイメージストリームタグの使用方法を説明します。

### 6.7.1. イメージストリームについての情報の取得

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

**手順**

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

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

以下は例になります。

```
$ oc describe is/python

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

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

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

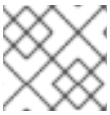
- 特定のイメージストリームタグについての利用可能な情報をすべて取得します。

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

以下は例になります。

```
$ oc describe istag/python:latest

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



**注記**

表示されている以上の情報が出力されます。

### 6.7.2. タグのイメージストリームへの追加

追加タグをイメージストリームに追加できます。

**手順**

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

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

以下は例になります。

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

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

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

latest
  tagged from
python@sha256:49c18358df82f4577386404991c51a9559f243e0b1bdc366df25

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

3.5
  tagged from centos/python-35-centos7

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

### 6.7.3. 外部イメージのタグの追加

外部イメージのタグを追加することができます。

**手順**

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

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

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

+

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

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

### 6.7.4. イメージストリームタグの更新

別のタグをイメージストリームに反映するようタグを更新できます。

**手順**

- タグを更新します。

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

たとえば、以下では **3.6** タグをイメージストリームに反映させるように **latest** タグを更新します。

```
$ oc tag python:3.6 python:latest
Tag python:latest set to
python@sha256:438208801c4806548460b27bd1fbcb7bb188273d13871ab43f.
```

### 6.7.5. イメージストリームタグの削除

古いタグをイメージストリームから削除できます。

**手順**

- 古いタグをイメージストリームから削除します。

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

以下は例になります。

```
$ oc tag -d python:3.5

Deleted tag default/python:3.5.
```

### 6.7.6. イメージストリームタグの定期的なインポートの設定

外部コンテナ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 <repositiory/image> <image-name:tag> --scheduled
```

以下は例になります。

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

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

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

2. 定期的なチェックを削除するには、上記のコマンド再実行しますが、**--scheduled** フラグを省略します。これにより、その動作がデフォルトに再設定されます。

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

# 第7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。

## 7.1. イメージコントローラー設定パラメーター

**image.config.openshift.io/cluster** リソースは以下の設定パラメーターを提供します。

| パラメーター | 説明 |
| --- | --- |
| **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。<br><br>**spec**: 設定についてのユーザーが設定できる値を保持します。**spec** サブセクションを編集できます。<br><br>**status**: クラスターから監視される値を保持します。 |
| **ImageSpec** | **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。この一覧を、有効なイメージを含むものとしてユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。イメージまたは **ImageStreamMappings** を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのはクラスター管理者のみです。<br><br>**additionalTrustedCA**: **ImageStream import**、**pod image pull**、**openshift-image-registry pullthrough**、およびビルドの処理時に信頼される必要のある追加の CA が含まれる ConfigMap の参照です。<br><br>この ConfigMap の namespace は **openshift-config** です。ConfigMap 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。<br><br>**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。 |
| **ImageStatus** | **internalRegistryHostname**: **internalRegistryHostname** を制御するイメージレジストリー Operator によって設定されます。これはデフォルトの内部イメージレジストリーのホスト名を設定します。値は **hostname[:port]** 形式の値である必要があります。後方互換性を確保す るために、**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。<br><br>**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 |
| --- | --- |
| **RegistryLocation** | 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。ドメイン名にはワイルドカードが含まれる可能性があります。<br><br>**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。<br><br>**insecure**: insecure はレジストリーがセキュアか、または非セキュアであるかを示します。指定がない場合には、デフォルトでレジストリーはセキュアであることが想定されます。 |
| **RegistrySources** | レジストリー設定を処理する方法についてのクラスター全体の情報を保持します。<br><br>**insecureRegistries**: 有効な TLS 証明書を持たないか、または HTTP 接続のみをサポートするレジストリーです。<br><br>**blockedRegistries**: イメージのプルおよびプッシュアクションについてブラックリスト化されます。他のすべてのレジストリーは許可されます。<br><br>**allowedRegistries**: イメージのプルおよびプッシュアクションについてホワイトリスト化されます。他のすべてのレジストリーはブロックされます。<br><br>**blockedRegistries** または **allowedRegistries** のいずれかのみを設定できます。 |

**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** および **quay.io** 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、Pod の失敗を防ぐために、お使いの環境内のペイロードイメージで必要なソースレジストリー **registry.redhat.io** および **quay.io** を宣言します。非接続クラスターの場合、ミラーレジストリーも追加する必要があります。

## 7.2. イメージ設定内容の設定

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

**手順**

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

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

以下は、**image.config.openshift.io/cluster** リソースの例になります。

```
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)
    insecureRegistries: (5)
    - insecure.com
    blockedRegistries: (6)
    - untrusted.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。この一覧を、有効なイメージを含むものとしてユーザーが信頼し、アプリケーションのインポート元となるレジストリーに設定します。イメージまたは **ImageStreamMappings** を API 経由で作成するパーミッションを持つユーザーは、このポリシーによる影響を受けません。通常、これらのパーミッションを持っているのはクラスター管理者のみです。

(3) **additionalTrustedCA**: **ImageStream import**、**pod image pull**、 **openshift-image-registry pullthrough**、およびビルドの処理時に信頼される必要のある追加の CA が含まれる ConfigMap の参照です。この ConfigMap の namespace は **openshift-config** です。ConfigMap 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、base64 エンコード証明書を値として使用します。

(4) **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。

(5) **insecureRegistries**: 有効な TLS 証明書を持たないか、または HTTP 接続のみをサポートするレジストリーです。

(6) **blockedRegistries**: イメージのプルおよびプッシュアクションについてブラックリスト化されます。他のすべてのレジストリーは許可されます。

### 7.2.1. 非セキュアなレジストリーのインポートと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、非セキュアなレジストリーを追加するか、レジストリーをブロックできます。OpenShift Container Platform は、この CR への変更をクラスター内のすべてのノードに適用します。

有効な TLS 証明書を持たない、または HTTP 接続のみをサポートするレジストリーなど、非セキュアな外部レジストリーは使用しないでください。

**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** 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:
  allowedRegistriesForImport:
    - domainName: quay.io
      insecure: false
  additionalTrustedCA:
    name: myconfigmap
  registrySources:
    insecureRegistries: (1)
    - insecure.com
    blockedRegistries: (2)
    - untrusted.com
    allowedRegistries:
    - quay.io (3)
status:
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

(1) 非セキュアなレジストリーを指定します。

(2) イメージのプルおよびプッシュアクションについてブラックリストに指定する必要のあるレジストリーを指定します。他のすべてのレジストリーは許可されます。**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。

(3) イメージのプルおよびプッシュアクションについて許可する必要のあるレジストリーを指定します。他のすべてのレジストリーは拒否されます。**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。

Machine Config Operator (MCO) は、レジストリーへの変更について **image.config.openshift.io/cluster** を監視し、変更を検出するとノードを再起動します。レジストリーへの変更は、各ノードの **/host/etc/containers/registries.conf** ファイルに表示されます。

```
cat /host/etc/containers/registries.conf
[registries]
  [registries.search]
    registries = ["registry.access.redhat.com", "docker.io"]
  [registries.insecure]
    registries = ["insecure.com"]
  [registries.block]
    registries = ["untrusted.com"]
```

### 7.2.2. 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

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

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

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

- イメージプルには、レジストリーのダウンタイムに対する回復性があります
- ネットワークが制限された環境のクラスターは、重要な場所 (quay.io など) からイメージをプルするように要求でき、会社のファイアウォールの背後にあるレジストリーが要求されたイメージを提供するようにできます。
- イメージのプル要求時にレジストリーへの接続が特定の順序で試行され、通常は永続レジストリーが最後に試行されます。
- 入力したミラー情報は、OpenShift Container Platform クラスターの全ノードの **/etc/containers/registries.conf** ファイルに追加されます。
- ノードがソースリポジトリ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。

- OpenShift Container Platform インストール時: OpenShift Container Platform が必要とする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配置すると、制限されたネットワーク内にあるデータセンターに OpenShift Container Platform をインストールできます。詳細は、OpenShift Container Platform イメージレジストリーのミラーリングについて参照してください。
- OpenShift Container Platform のインストール後: OpenShift Container Platform インストール時にミラーリングを設定しなくても、**ImageContentSourcePolicy** オブジェクトを使用して後で設定することができます。

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

- ミラーリングするコンテナ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。

**前提条件**

- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。

**手順**

1. ミラーリングされたリポジトリーを設定します。これを実行するには、以下のいずれかを行います。
   - 「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、ソースディレク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/ubi8/ubi-minimal@sha256:c505667389712dc337986e29ffcb65116879ef27629dc3ce6e1b17727c06e78f \
       docker://example.io/ubi8/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 **ubi8/ubi-minimal** イメージをコピーします。レジストリーを作成した後、OpenShift Container Platform クラスターを設定して、ソースリポジトリーで作成される要求をミラーリングされたリポジトリーにリダイレクトできます。
2. OpenShift Container Platform クラスターにログインします。
3. **ImageContentSourcePolicy** ファイル (例: **registryrepomirror.yaml**) を作成し、ソースとミラーを固有のレジストリー、およびリポジトリーのペアとイメージのものに置き換えます。

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: ImageContentSourcePolicy
   metadata:
     name: ubi8repo
   spec:
     repositoryDigestMirrors:
     - mirrors:
       - example.io/example/ubi-minimal (1)
       source: registry.access.redhat.com/ubi8/ubi-minimal (2)
     - mirrors:
   ```

```
    - example.com/example/ubi-minimal
    source: registry.access.redhat.com/ubi8/ubi-minimal
~
```

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

4. 新しい **ImageContentSourcePolicy** を作成します。

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

**ImageContentSourcePolicy** が作成されると、新しい設定が各ノードにデプロイされ、ソースリポジトリーへの要求のためにミラーリングされたリポジトリーの使用をすぐに開始します。

5. ミラーリングされた設定が機能することを確認するには、ノードのいずれかに移動します。以下は例になります。

   a. ノードの一覧を表示します。

```
$ oc get node
NAME                         STATUS                     ROLES    AGE  VERSION
ip-10-0-137-44.ec2.internal    Ready                      worker   7m   v1.14.6+90fadebfa
ip-10-0-138-148.ec2.internal   Ready                      master   11m  v1.14.6+90fadebfa
ip-10-0-139-122.ec2.internal   Ready                      master   11m  v1.14.6+90fadebfa
ip-10-0-147-35.ec2.internal    Ready,SchedulingDisabled   worker   7m
v1.14.6+90fadebfa
ip-10-0-153-12.ec2.internal    Ready                      worker   7m   v1.14.6+90fadebfa
ip-10-0-154-10.ec2.internal    Ready                      master   11m  v1.14.6+90fadebfa
```

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

   b. **/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。

```
$ oc debug node/ip-10-0-147-35.ec2.internal
Starting pod/ip-10-0-147-35ec2internal-debug ...
To use host binaries, run `chroot /host`

sh-4.2# chroot /host
sh-4.2# cat /etc/containers/registries
unqualified-search-registries = ["registry.access.redhat.com", "docker.io"]
[[registry]]
  location = "registry.access.redhat.com/ubi8/"
  insecure = false
  blocked = false
  mirror-by-digest-only = true
  prefix = ""

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

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

c. ソースからノードにイメージをプルし、ミラーによって実際に解決されているかどうかを確認します。

```
sh-4.2# podman pull --log-level=debug registry.access.redhat.com/ubi8/ubi-minimal
```

### 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** 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。*

# 第8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。

## 8.1. テンプレートについて

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

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

## 8.2. テンプレートのアップロード

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

**手順**

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

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

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

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

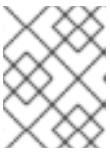
テンプレートは、Web コンソールまたは CLI を使用して選択できるようになりました。

## 8.3. WEB コンソールを使用したアプリケーションの作成

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

**手順**

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



**注記**

以下に示すように、**builder** タグがアノテーションに列挙されるイメージストリームタグのみがこの一覧に表示されます。

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby"
  creationTimestamp: null
spec:
  dockerImageRepository: "registry.redhat.io/openshift3/ruby-20-rhel7"
  tags:
    -
      name: "2.0"
      annotations:
        description: "Build and run Ruby 2.0 applications"
        iconClass: "icon-ruby"
        tags: "builder,ruby" (1)
        supports: "ruby:2.0,ruby"
        version: "2.0"
```

(1) ここに **builder** を含めると、この **ImageStreamTag** がビルダーとして Web コンソールに表示されます。

3. 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。

## 8.4. 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。

### 8.4.1. 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
```

### 8.4.2. パラメーターの一覧表示

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

**手順**

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、パラメーターを一覧表示することができます。

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

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

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

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

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

### 8.4.3. オブジェクト一覧の生成

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

**手順**

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

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

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

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

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

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

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

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

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

   a. テンプレートからのオブジェクト一覧の作成

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

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

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

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

```
$ cat postgres.env
POSTGRESQL_USER=bob
POSTGRESQL_DATABASE=mydatabase
$ oc process -f my-rails-postgresql --param-file=postgres.env
```

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

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

## 8.5. 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>
```

## 8.6. インスタントアプリおよびクイックスタートテンプレートの使用

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

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

**手順**

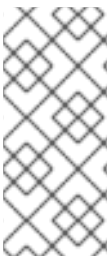
1. 以下のように、利用可能なデフォルトのインスタントアプリとクイックスタートテンプレートを一覧表示できます。

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

2. ソースを変更し、アプリケーションの独自のバージョンをビルドするには、以下を実行します。

   a. テンプレートのデフォルト **SOURCE_REPOSITORY_URL** パラメーターが参照するリポジトリーをフォークします。

   b. テンプレートから作成する場合には、**SOURCE_REPOSITORY_URL** パラメーターの値を上書きします。 デフォルト値ではなく、フォークを指定してください。
   これにより、テンプレートで作成したビルド設定はアプリケーションコードのフォークを参照するようになり、コードを変更し、アプリケーションを自由に再ビルドできます。



**注記**

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

### 8.6.1. クイックスタートのテンプレート

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

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

クイックスタートは、アプリケーションのソースコードを含むソースリポジトリーを参照します。クイックスタートをカスタマイズするには、リポジトリーをフォークし、テンプレートからアプリケーションを作成する時に、デフォルトのソースリポジトリー名をフォークしたリポジトリーに置き換えま

す。これにより、提供されたサンプルのソースではなく、独自のソースコードを使用してビルドが実行されます。ソースリポジトリーでコードを更新し、新しいビルドを起動して、デプロイされたアプリケーションで変更が反映されていることを確認できます。

#### 8.6.1.1. Web フレームワーククイックスタートのテンプレート

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

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

## 8.7. テンプレートの作成

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

以下は、単純なテンプレートオブジェクト定義 (YAML) の例です。

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

```
    name: REDIS_PASSWORD
  labels:
    redis: master
```

### 8.7.1. テンプレート記述の作成

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

以下は、テンプレート記述メタデータの例です。

```
kind: Template
apiVersion: v1
metadata:
  name: cakephp-mysql-example (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ー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できます。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。カテゴリーはクラスター全体に対してカスタマイズすることもできます。

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。可能な場合は、既存のロゴアイコンから選択します。また、FontAwesome からもアイコンを使用できます。または、テンプレート

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。

### 8.7.2. テンプレートラベルの作成

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

以下は、テンプレートオブジェクトのラベルの例です。

```
kind: "Template"
apiVersion: "v1"
...
labels:
  template: "cakephp-mysql-example" 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。

### 8.7.3. テンプレートパラメーターの作成

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

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

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

**${{PARAMETER_NAME}}** 構文のみを使用する場合は、単一のパラメーター参照のみが許可され、先頭文字や終了文字は使用できません。結果の値は、置換後に結果が有効な json オブジェクトの場合は引用

されません。結果が有効な json 値でない場合に、結果の値は引用され、標準の文字列として処理されます。

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

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

以下は、明示的な値をデフォルト値として設定する例です。

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

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

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

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

利用可能な構文は、完全な正規表現構文ではありません。ただし、**\w**、**\d**、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。

以下は、パラメーター定義と参照を含む完全なテンプレートの例です。

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

```
        ref: "${SOURCE_REPOSITORY_REF}"
      contextDir: "${CONTEXT_DIR}"
  - kind: DeploymentConfig
    apiVersion: v1
    metadata:
      name: frontend
    spec:
      replicas: "${{REPLICA_COUNT}}" 2
parameters:
  - name: SOURCE_REPOSITORY_URL 3
    displayName: Source Repository URL 4
    description: The URL of the repository with your application source code 5
    value: https://github.com/sclorg/cakephp-ex.git 6
    required: true 7
  - name: GITHUB_WEBHOOK_SECRET
    description: A secret string used to configure the GitHub webhook
    generate: expression 8
    from: "[a-zA-Z0-9]{40}" 9
  - name: REPLICA_COUNT
    description: Number of replicas to run
    value: "2"
    required: true
message: "... The GitHub webhook secret is ${GITHUB_WEBHOOK_SECRET} ..." 10
```

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

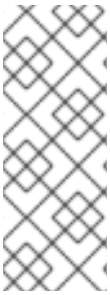
### 8.7.4. テンプレートオブジェクト一覧の作成

テンプレートの主な部分は、テンプレートがインスタンス化される時に作成されるオブジェクトの一覧です。これには、**BuildConfig**、**DeploymentConfig**、**Service** など、有効な API オブジェクトを使用できます。オブジェクトはここで定義された通りに作成され、パラメーターの値は作成前に置換されます。これらのオブジェクトの定義では、以前に定義したパラメーターを参照できます。

以下は、オブジェクト一覧の例です。

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

1 **Service** 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。

### 8.7.5. テンプレートをバインド可能としてマーキングする

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

**手順**

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

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

### 8.7.6. テンプレートオブジェクトフィールドの公開

テンプレートの作成者は、テンプレートに含まれる特定のオブジェクトのフィールドを公開すべきかどうかを指定できます。テンプレートサービスブローカーは、ConfigMap、Secret、Service、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、プレフィックスが **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、プレフィックス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/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プレフィックスを取り除いた値を有効な環境変数名として使用することが推奨されます。先頭に**A-Z**、**a-z** または _ を指定して、その後に、ゼロか、他の文字 **A-Z**、**a-z**、**0-9** または _ を指定してください。

**注記**

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

以下は、公開されるさまざまなオブジェクトのフィールドの例です。

```
kind: Template
apiVersion: v1
metadata:
  name: my-template
objects:
- kind: ConfigMap
  apiVersion: v1
  metadata:
    name: my-template-config
    annotations:
      template.openshift.io/expose-username: "{.data['my\\.username']}"
  data:
    my.username: foo
- kind: Secret
  apiVersion: v1
  metadata:
    name: my-template-config-secret
    annotations:
      template.openshift.io/base64-expose-password: "{.data['password']}"
  stringData:
```

```
    password: bar
- kind: Service
  apiVersion: v1
  metadata:
    name: my-template-service
    annotations:
      template.openshift.io/expose-service_ip_port: "{.spec.clusterIP}:{.spec.ports[?
(.name==\"web\")].port}"
  spec:
    ports:
    - name: "web"
      port: 8080
- kind: Route
  apiVersion: v1
  metadata:
    name: my-template-route
    annotations:
      template.openshift.io/expose-uri: "http://{.spec.host}{.spec.path}"
  spec:
    path: mypath
```

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

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

**手順**

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

### 8.7.7. テンプレートの準備ができるまで待機する

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

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

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

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

テンプレートのインスタンス化は失敗します。

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

| 種類 | 準備状態 (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** | オブジェクトが新しい ReplicaSet やデプロイメントが利用可能であることを報告する (これはオブジェクトに定義された readiness プローブに従います) | 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** | オブジェクトが新しい ReplicaController やデプロイメントが利用可能であると報告する (これはオブジェクトに定義された readiness プローブに従います) | 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** | オブジェクトがすべてのレプリカが準備状態にあることを報告する (これはオブジェクトに定義された readiness プローブに従います) | 該当なし |

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

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

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

**その他の推奨事項**

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

### 8.7.8. 既存オブジェクトからのテンプレートの作成

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

**手順**

1. オブジェクトを YAML 形式でプロジェクトにエクスポートします。

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

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

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

   - BuildConfig
   - Build
   - DeploymentConfig
   - ImageStream
   - Pod
   - ReplicationController
   - Route

- Service

# 第9章 RUBY ON RAILS の使用

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

**警告**

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

**前提条件**

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

## 9.1. 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。

## 9.2. 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
Successfully installed rails-4.2.0
1 gem installed
```

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

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

3. 新規アプリケーションディレクトリーに切り替えます。

```
$ cd rails-app
```

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

```
gem 'pg'
```

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

```
$ bundle install
```

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

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

7. アプリケーションの開発およびテスト用のデータベースを作成します。

```
$ rake db:create
```

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

### 9.2.1. 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 **index** アクションの **ビュー** 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。

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

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

**手順**

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

```
<% 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"] %>
```

### 9.2.3. アプリケーションの Git への保存

通常 OpenShift Container Platform でアプリケーションをビルドする場合、ソースコードを git リポジトリ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
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。

1. 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
```

2. 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
```

## 9.3. アプリケーションの OPENSHIFT CONTAINER PLATFORM へのデプロイ

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

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

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

- OpenShift Container Platform の PostgreSQL イメージからデータベースサービスを作成します。
- データベースサービスと連動する OpenShift Container Platform の Ruby 2.0 ビルダーイメージおよび 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。

**手順**

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

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

### 9.3.1. データベースサービスの作成

Rails アプリケーションには実行中のデータベースサービスが必要です。このサービスには、PostgeSQL 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
```

### 9.3.2. フロントエンドサービスの作成

アプリケーションを OpenShift Container Platform にデプロイするには、アプリケーションが置かれるリポジトリ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
```

   このコマンドでは、OpenShift Container Platform は指定された環境変数を使用してソースコードの取得、ビルダーのセットアップ、アプリケーションイメージのビルド、新規に作成されたイメージのデプロイを実行します。このアプリケーションには **rails-app** という名前を指定します。

2. **rails-app** DeploymentConfig 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。

```
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. ビルドが完了すると、OpenShift Container Platform で Pod が実行されていることを確認します。

```
$ oc get pods
```

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

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

### 9.3.3. 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。

**手順**

- **www.example.com** などの外部からアクセスできるホスト名を指定してサービスを公開するには、OpenShift Container Platform のルートを使用します。この場合は、以下を入力してフロントエンドサービスを公開する必要があります。

```
$ oc expose service rails-app --hostname=www.example.com
```

**警告**

指定するホスト名がルーターの IP アドレスに解決することを確認します。

# 第10章 イメージの使用

## 10.1. イメージの使用の概要

以下のトピックを使用して、OpenShift Container Platform ユーザーに提供されているさまざまな Source-to-Image (S2I)、データベース、その他のコンテナーイメージを確認します。

Red Hat の公式コンテナーイメージは、registry.redhat.io の Red Hat レジストリーで提供されています。OpenShift Container Platform がサポートする S2I、データベース、Jenkins イメージは、Red Hat Quay レジストリーの **openshift4** リポジトリーにあります。たとえば、**quay.io/openshift-release-dev/ocp-v4.0-<address>** は OpenShift Application Platform イメージの名前です。

xPaaS ミドルウェアイメージは、Red Hat レジストリーの適切な製品リポジトリーで提供されていますが、サフィックスとして **-openshift** が付いています。たとえば、**registry.redhat.io/jboss-eap-6/eap64-openshift** は JBoss EAP イメージの名前です。

このセクションで説明する Red Hat がサポートするイメージはすべて Red Hat Container Catalog に記載されています。各イメージのすべてのバージョンについて、そのコンテンツや用途の詳細を確認できます。関連するイメージを参照または検索してください。

**重要**

コンテナーイメージの新しいバージョンは、OpenShift Container Platform の以前のバージョンとは互換性がありません。お使いの OpenShift Container Platform のバージョンに基づいて、正しいバージョンのコンテナーイメージを確認し、使用するようにしてください。

## 10.2. JENKINS イメージの設定

OpenShift Container Platform には、Jenkins 実行用のコンテナーイメージがあります。このイメージには Jenkins サーバーインスタンスが含まれており、このインスタンスを使用して継続的なテスト、統合、デリバリーの基本フローを設定することができます。

イメージは、Red Hat Universal Base Images (UBI) に基づいています。

OpenShift Container Platform は、Jenkins の LTS リリースに従います。OpenShift Container Platform は、Jenkins 2.x を含むイメージを提供します。

OpenShift Container Platform Jenkins イメージは、**quay.io** または **registry.redhat.io** で利用できます。

以下は例になります。

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins:<v4.2.0>
```

これらのイメージを使用するには、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュできます。さらに、コンテナーイメージレジストリーまたは外部の場所に、対象イメージを参照する ImageStream を作成することもできます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。

ただし便宜上、OpenShift Container Platform はコア Jenkins イメージの **openshift** namespace に ImageStreams を提供するほか、OpenShift Container Platform を Jenkins と統合するために提供されるエージェントイメージのサンプルも提供します。

### 10.2.1. 設定とカスタマイズ

Jenkins 認証は、以下の 2 つの方法で管理できます。

- OpenShift ログインプラグインが提供する OpenShift Container Platform OAuth 認証
- Jenkins が提供する標準認証。

#### 10.2.1.1. OpenShift Container Platform OAuth 認証

OAuth 認証は、Jenkins UI の **Configure Global Security** パネルでオプションを設定するか、Jenkins **デプロイメント設定**の **OPENSHIFT_ENABLE_OAUTH** 環境変数を **false** 以外に設定して、有効にします。これにより、OpenShift Container Platform ログインプラグインが有効になり、Pod データからか、または OpenShift Container Platform API サーバーと対話して設定情報を取得します。

有効な認証情報は、OpenShift Container Platform アイデンティティープロバイダーが制御します。

Jenkins はブラウザーおよびブラウザー以外のアクセスの両方をサポートします。

OpenShift Container Platform **Roles** でユーザーに割り当てられる固有の Jenkins パーミッションが指定されている場合、有効なユーザーは、ログイン時に自動的に Jenkins 認証マトリックスに追加されます。デフォルトで使用される **Roles** は、事前に定義される **admin**、**edit**、および **view** です。ログインプラグインは、Jenkins が実行されている **Project** または namespace のそれらの **Roles** に対して自己 SAR 要求 (self-SAR request)を実行します。

**admin** ロールを持つユーザーには、従来の Jenkins 管理ユーザーパーミッションがあります。ユーザーのパーミッションは、ロールが **edit**、**view** になるほど少なくなります。

OpenShift Container Platform のデフォルトの **admin**、**edit**、**view** の **Roles**、これらの **Roles** が Jenkins インスタンスに割り当てられている Jenkins パーミッションは設定可能です。

OpenShift Container Platform **Pod** で Jenkins を実行する場合、ログインプラグインは Jenkins が実行されている namespace で **openshift-jenkins-login-plugin-config** という名前の **ConfigMap** を検索します。

このプラグインが検出し、その **ConfigMap** で読み取り可能な場合には、**Role** を Jenkins パーミッションマッピングに定義できます。具体的には以下を実行します。

- ログインプラグインは、**ConfigMap** のキーと値のペアを OpenShift Role マッピングに対する Jenkins パーミッションとして処理します。
- キーは Jenkins パーミッショングループの短い ID と Jenkins パーミッションの短い ID で、この 2 つはハイフンで区切られています。
- OpenShift Container Platform **Role** に **Overall Jenkins Administer** パーミッションを追加する場合は、キーは **Overall-Administer** である必要があります。
- パーミッショングループおよびパーミッション ID が利用可能であるかどうかを把握するには、Jenkins コンソールのマトリックス認証ページに移動し、グループの ID とグループが提供するテーブルの個々のパーミッションを確認します。
- キーと値ペアの値は、パーミッションが適用される必要がある OpenShift Container Platform **Roles** の一覧で、各ロールはカンマで区切られています。
- **Overall Jenkins Administer** パーミッションをデフォルトの **admin** および **edit Roles** の両方に追加し、作成した新規の jenkins ロールも追加する場合は、キーの **Overall-Administer** の値は **admin,edit,jenkins** になります。

**注記**

OpenShift Container Platform OAuth が使用されている場合、管理者権限で OpenShift Container Platform Jenkins イメージに事前に設定されている **admin** ユーザーには、これらの権限は割り当てられません。これらのパーミッションを付与するには、OpenShift Container Platform クラスター管理者は OpenShift Container Platform アイデンティティープロバイダーでそのユーザーを明示的に定義し、**admin** ロールをユーザーに割り当てる必要があります。

保存される Jenkins ユーザーのパーミッションは、初回のユーザー作成後に変更できます。OpenShift ログインプラグインは、OpenShift Container Platform API サーバーをポーリングしてパーミッションを取得し、ユーザーごとに Jenkins に保存されているパーミッションを、OpenShift Container Platform から取得したパーミッションに更新します。Jenkins UI を使用して Jenkins ユーザーのパーミッションを更新する場合には、プラグインが次回に OpenShift Container Platform をポーリングするタイミングで、パーミッションの変更が上書きされます。

ポーリングの頻度は **OPENSHIFT_PERMISSIONS_POLL_INTERVAL** 環境変数で制御できます。デフォルトのポーリングの間隔は 5 分です。

OAuth 認証を使用して新しい Jenkins サービスを作成するには、テンプレートを使用するのが最も簡単な方法です。

#### 10.2.1.2. Jenkins 認証

テンプレートを使用せず、イメージが直接実行される場合には、デフォルトで Jenkins 認証が使用されます。

Jenkins の初回起動時には、設定、管理ユーザーおよびパスワードが作成されます。デフォルトのユーザー認証情報は、**admin** と **password** です。標準の Jenkins 認証を使用する場合のみ、**JENKINS_PASSWORD** 環境変数を設定してデフォルトのパスワードを設定します。

**手順**

- 標準の Jenkins 認証を使用する Jenkins アプリケーションを作成します。

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    openshift4/ose-jenkins
```

### 10.2.2. Jenkins 環境変数

Jenkins サーバーは、以下の環境変数で設定できます。

| 変数 | 定義 | 値と設定の例 |
| --- | --- | --- |
| **OPENSHIFT_ENABLE_OAUTH** | Jenkins へのログイン時に OpenShift ログインプラグインが認証を管理するかどうかを決定します。有効にするには、**true** に設定します。 | デフォルト: **false** |

| 変数 | 定義 | 値と設定の例 |
| --- | --- | --- |
| **JENKINS_PASSWORD** | 標準の Jenkins 認証を使用する際の **admin** ユーザーのパスワード。**OPENSHIFT_ENABLE_OAUTH** が **true** に設定されている場合には該当しません。 | デフォルト: **password** |
| **JAVA_MAX_HEAP_PARAM**、**CONTAINER_HEAP_PERCENT**、**JENKINS_MAX_HEAP_UPPER_BOUND_MB** | これらの値は Jenkins JVM の最大ヒープサイズを制御します。**JAVA_MAX_HEAP_PARAM** が設定されている場合は、その値が優先されます。設定されていない場合は、最大ヒープサイズは、コンテナーメモリー制限の **CONTAINER_HEAP_PERCENT** として動的に計算され、オプションで **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB を上限とします。<br><br>デフォルトでは Jenkins JVM の最大ヒープサイズは、上限なしでコンテナーメモリー制限の 50% に設定されます。 | **JAVA_MAX_HEAP_PARAM** の設定例: **-Xmx512m**<br><br>**CONTAINER_HEAP_PERCENT** のデフォルト: **0.5** (50%)<br><br>**JENKINS_MAX_HEAP_UPPER_BOUND_MB** の設定例: **512 MiB** |
| **JAVA_INITIAL_HEAP_PARAM**、**CONTAINER_INITIAL_PERCENT** | これらの値は Jenkins JVM の初期ヒープサイズを制御します。**JAVA_INITIAL_HEAP_PARAM** が設定されている場合は、その値が優先されます。設定されていない場合は、初期ヒープサイズは、動的に計算される最大ヒープサイズの **CONTAINER_INITIAL_PERCENT** として動的に計算されます。<br><br>デフォルトでは、JVM は初期のヒープサイズを設定します。 | **JAVA_INITIAL_HEAP_PARAM** の設定例: **-Xmx32m**<br><br>**CONTAINER_INITIAL_PERCENT** の設定例: **0.1** (10%) |
| **CONTAINER_CORE_LIMIT** | 設定されている場合には、内部の JVM スレッドのサイジング数に使用するコアの数を整数で指定します。 | 設定例: **2** |

| 変数 | 定義 | 値と設定の例 |
| --- | --- | --- |
| **JAVA_TOOL_OPTIONS** | このコンテナーで実行中のすべての JVM に適用するオプションを指定します。この値の上書きは推奨していません。 | デフォルト: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true** |
| **JAVA_GC_OPTS** | Jenkins JVM ガーベッジコレクションのパラメーターを指定します。この値の上書きは推奨していません。 | デフォルト: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90** |
| **JENKINS_JAVA_OVERRIDES** | Jenkins JVM の追加オプションを指定します。これらのオプションは、上記の Java オプションなどその他すべてのオプションに追加され、必要に応じてそれらの値のいずれかを上書きするのに使用できます。追加オプションがある場合には、スペースで区切ります。オプションにスペース文字が含まれる場合には、バックスラッシュでエスケープしてください。 | 設定例: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value** |
| **JENKINS_OPTS** | Jenkins への引数を指定します。 | |
| **INSTALL_PLUGINS** | コンテナーが初めて実行された場合や、**OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** が **true** に設定されている場合に、インストールする追加の Jenkins プラグインを指定します。プラグインは、名前:バージョンのペアのカンマ区切りの一覧で指定されます。 | 設定例: **git:3.7.0,subversion:2.10.2** |
| **OPENSHIFT_PERMISSIONS_POLL_INTERVAL** | OpenShift ログインプラグインが Jenkins に定義されているユーザーごとに関連付けられたパーミッションを OpenShift Container Platform でポーリングする間隔をミリ秒単位で指定します。 | デフォルト: **300000** - 5 分 |

| 変数 | 定義 | 値と設定の例 |
| --- | --- | --- |
| **OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG** | Jenkins 設定ディレクトリー用に OpenShift Container Platform 永続ボリュームを使用してこのイメージを実行する場合、永続ボリュームは Persistent Volume Claim (PVC、永続ボリューム要求）の作成時に割り当てられるため、イメージから永続ボリュームに設定が転送されるのは、イメージの初回起動時のみです。このイメージを拡張し、初回起動後にカスタムイメージの設定を更新するカスタムイメージを作成する場合、この環境変数を **true** に設定しない限り、設定はコピーされません。 | デフォルト: **false** |
| **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** | Jenkins 設定ディレクトリー用に OpenShift Container Platform 永続ボリュームを使用してこのイメージを実行する場合、永続ボリュームは Persistent Volume Claim (PVC、永続ボリューム要求) の作成時に割り当てられるため、イメージから永続ボリュームにプラグインが転送されるのは、イメージの初回起動時のみです。このイメージを拡張し、初回起動後にカスタムイメージのプラグインを更新するカスタムイメージを作成する場合、この環境変数を **true** に設定しない限り、プラグインはコピーされません。 | デフォルト: **false** |
| **ENABLE_FATAL_ERROR_LOG_FILE** | Jenkins 設定ディレクトリー用に OpenShift Container Platform の Persistent Volume Claim (PVC、永続ボリューム要求) を使用してこのイメージを実行する場合に、この環境変数は致命的なエラーが生じる際に致命的なエラーのログファイルが永続することを可能にします。致命的なエラーのファイルは **/var/lib/jenkins/logs** に保存されます。 | デフォルト: **false** |

| 変数 | 定義 | 値と設定の例 |
| --- | --- | --- |
| **NODEJS_SLAVE_IMAGE** | この値を設定すると、デフォルトの NodeJS エージェント Pod 設定に使用されるイメージが上書きされます。**jenkins-agent-nodejs** という名前の関連イメージストリームタグがプロジェクトにあります。この変数は、有効にするために Jenkins の初回の起動前に設定される必要があります。 | Jenkins サーバーのデフォルトの NodeJS エージェントイメージ: **image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-nodejs:latest** |
| **MAVEN_SLAVE_IMAGE** | この値を設定すると、デフォルトの maven エージェント Pod 設定に使用されるイメージが上書きされます。**jenkins-agent-maven** という名前の関連イメージストリームタグがプロジェクトにあります。この変数は、有効にするために Jenkins の初回の起動前に設定される必要があります。 | Jenkins サーバーのデフォルトの Maven エージェントイメージ: **image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-maven:latest** |

### 10.2.3. Jenkins へのプロジェクト間のアクセスの提供

同じプロジェクト以外で Jenkins を実行する場合には、プロジェクトにアクセスするために、Jenkins にアクセストークンを提供する必要があります。

**手順**

1. サービスアカウントのシークレットを特定します。そのアカウントには、Jenkins がアクセスする必要のあるプロジェクトにアクセスするための適切なパーミッションがあります。

```
$ oc describe serviceaccount jenkins
Name:       default
Labels:     <none>
Secrets:    {  jenkins-token-uyswp    }
            {  jenkins-dockercfg-xcr3d    }
Tokens:     jenkins-token-izv1u
            jenkins-token-uyswp
```

   ここでは、シークレットの名前は **jenkins-token-uyswp** です。

2. シークレットからトークンを取得します。

```
$ oc describe secret <secret name from above>
Name:       jenkins-token-uyswp
Labels:     <none>
Annotations:    kubernetes.io/service-account.name=jenkins,kubernetes.io/service-
account.uid=32f5b661-2a8f-11e5-9528-3c970e3bf0b7
Type:   kubernetes.io/service-account-token
Data
```

```
====
ca.crt: 1066 bytes
token:  eyJhbGc..<content cut>....wRA
```

トークンパラメーターには、Jenkins がプロジェクトにアクセスするために必要とするトークンの値が含まれます。

### 10.2.4. Jenkins のボリューム間のマウントポイント

Jenkins イメージはマウントしたボリュームで実行して、設定用に永続ストレージを有効にできます。

- **/var/lib/jenkins** - これは、Jenkins がジョブ定義などの設定ファイルを保存するデータディレクトリーです。

### 10.2.5. S2I (Source-To-Image) による Jenkins イメージのカスタマイズ

正式な OpenShift Container Platform Jenkins イメージをカスタマイズするには、イメージを Source-To-Image (S2I) ビルダーとしてイメージを使用できます。

S2I を使用して、カスタムの Jenkins ジョブ定義をコピーしたり、プラグインを追加したり、同梱の **config.xml** ファイルを独自のカスタムの設定に置き換えたりできます。

Jenkins イメージに変更を追加するには、以下のディレクトリー構造の Git リポジトリーが必要です。

**plugins**
　このディレクトリーには、Jenkins にコピーするバイナリーの Jenkins プラグインを含めます。

**plugins.txt**
　このファイルは、以下の構文を使用して、インストールするプラグインを一覧表示します。

```
pluginId:pluginVersion
```

**configuration/jobs**
　このディレクトリーには、Jenkins ジョブ定義が含まれます。

**configuration/config.xml**
　このファイルには、カスタムの Jenkins 設定が含まれます。

**configuration/** ディレクトリーのコンテンツは **/var/lib/jenkins/** ディレクトリーにコピーされるので、このディレクトリーに **credentials.xml** などのファイルをさらに追加することもできます。

以下のビルド設定のサンプルは、OpenShift Container Platform で Jenkins イメージをカスタマイズします。

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:                    1
    git:
      uri: https://github.com/custom/repository
    type: Git
  strategy:                  2
    sourceStrategy:
```

```
      from:
        kind: ImageStreamTag
        name: jenkins:2
        namespace: openshift
    type: Source
  output:                    3
    to:
      kind: ImageStreamTag
      name: custom-jenkins:latest
```

1. **source** パラメーターは、上記のレイアウトでソースの Git リポジトリーを定義します。
2. **strategy** パラメーターは、ビルドのソースイメージとして使用するための元の Jenkins イメージを定義します。
3. **output** パラメーターは、結果として生成される、カスタマイズした Jenkins イメージを定義します。これは、公式の Jenkins イメージの代わりに、デプロイメント設定で使用できます。

### 10.2.6. Jenkins Kubernetes プラグインの設定

OpenShift Container Platform Jenkins イメージには、事前にインストール済みの Kubernetes プラグインが含まれ、Kubernetes および OpenShift Container Platform を使用して、Jenkins エージェントを複数のコンテナーホストで動的にプロビジョニングできるようにします。

OpenShift Container Platform は、Kubernetes プラグインを使用するために、Jenkins エージェントとして使用するのに適したイメージ (**Base**、**Maven**、および **Node.js** イメージ) を提供します。

Maven および Node.js のエージェントイメージは、Kubernetes プラグイン用の OpenShift Container Platform Jenkins イメージの設定内で、Kubernetes Pod テンプレートイメージとして自動的に設定されます。この設定にはイメージごとのラベルが含まれており、**Restrict where this project can be run** の設定にある Jenkins ジョブのいずれかに適用できます。ラベルが適用されている場合、ジョブはそれぞれのエージェントイメージを実行する OpenShift Container Platform Pod の下で実行されます。

Jenkins イメージは、Kubernetes プラグインの追加のエージェントイメージの自動検出および自動設定を実行します。

OpenShift Container Platform 同期プラグインでは、Jenkins イメージは、Jenkins の起動時に、実行中のプロジェクトまたはプラグインの設定に具体的に一覧表示されているプロジェクト内で以下を検索します。

- ラベル **role** が **jenkins-slave** に設定されているイメージストリーム。
- アノテーション **role** が **jenkins-slave** に設定されているイメージストリームタグ。
- ラベル **role** が **jenkins-slave** に設定されている ConfigMap

適切なラベルが付いたイメージストリーム、または適切なアノテーションが付いたイメージストリームタグが見つかると、対応する Kubernetes プラグイン設定が生成されるため、イメージストリームから提供されるコンテナーイメージを実行する Pod で、Jenkins ジョブを実行するように割り当てることができます。

イメージストリームまたはイメージストリームタグの名前およびイメージ参照が、Kubernetes プラグインの Pod テンプレートにある名前およびイメージフィールドにマッピングされます。Kubernetes プラグインの Pod テンプレートのラベルフィールドは、**slave-label** キーを使用し、イメージストリーム

またはイメージストリームタグのオブジェクトにアノテーションを設定して制御できます。これらを使用しない場合には、名前をラベルとして使用します。

**注記**

Jenkins コンソールにログインして、Pod テンプレート設定を変更しないでください。Pod テンプレートが作成された後にこれを行い、OpenShift 同期プラグインが ImageStream または ImageStreamTag に関連付けられたイメージが変更されたことを検知した場合、Pod テンプレートを置き換え、これらの設定変更を上書きします。新しい設定を既存の設定とマージすることはできません。

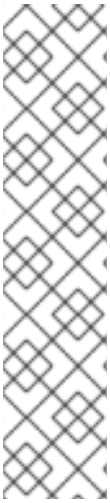
より複雑な設定が必要な場合は、ConfigMap を使用する方法を検討してください。

適切なラベルが指定された ConfigMap が検出される場合は、ConfigMap のキーと値のデータペイロードの値に、Jenkins および Kubernetes プラグインの Pod テンプレートの設定形式に準拠する XML が含まれることが想定されます。イメージストリームまたはイメージストリームタグではなく、ConfigMap を使用する際に注意すべき主な違いとして、Kubernetes プラグインの Pod テンプレートのすべてのパラメーターを制御できる点に留意してください。

**jenkins-agent** の ConfigMap の例:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-slave
data:
  template1: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
      <inheritFrom></inheritFrom>
      <name>template1</name>
      <instanceCap>2147483647</instanceCap>
      <idleMinutes>0</idleMinutes>
      <label>template1</label>
      <serviceAccount>jenkins</serviceAccount>
      <nodeSelector></nodeSelector>
      <volumes/>
      <containers>
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>jnlp</name>
          <image>openshift/jenkins-agent-maven-35-centos7:v3.10</image>
          <privileged>false</privileged>
          <alwaysPullImage>true</alwaysPullImage>
          <workingDir>/tmp</workingDir>
          <command></command>
          <args>${computer.jnlpmac} ${computer.name}</args>
          <ttyEnabled>false</ttyEnabled>
          <resourceRequestCpu></resourceRequestCpu>
          <resourceRequestMemory></resourceRequestMemory>
          <resourceLimitCpu></resourceLimitCpu>
          <resourceLimitMemory></resourceLimitMemory>
          <envVars/>
        </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
      </containers>
```

```
    <envVars/>
    <annotations/>
    <imagePullSecrets/>
    <nodeProperties/>
  </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```



**注記**

Jenkins コンソールにログインして、Pod テンプレートの作成後に Pod テンプレート設定をさらに変更し、OpenShift 同期プラグインが ConfigMap が変更されたことを検知すると、これは Pod テンプレートを置き換え、この設定変更を上書きします。新しい設定を既存の設定とマージすることはできません。

Jenkins コンソールにログインして、Pod テンプレート設定を変更しないでください。Pod テンプレートが作成された後にこれを行い、OpenShift 同期プラグインが ImageStream または ImageStreamTag に関連付けられたイメージが変更されたことを検知した場合、Pod テンプレートを置き換え、これらの設定変更を上書きします。新しい設定を既存の設定とマージすることはできません。

より複雑な設定が必要な場合は、ConfigMap を使用する方法を検討してください。

インストールされた後、OpenShift 同期プラグインは **ImageStreams**、**ImageStreamTags**、および **ConfigMaps** に更新がないか、OpenShift Container Platform の API サーバーをモニタリングして、Kubernetes プラグインの設定を調整します。

以下のルールが適用されます。

- **ConfigMap**、**ImageStream**、または **ImageStreamTag** からラベルまたはアノテーションを削除すると、既存の **PodTemplate** が Kubernetes プラグインの設定から削除されます。
- これらのオブジェクトが削除されると、対応する設定が Kubernetes プラグインから削除されます。
- 適切なラベルおよびアノテーションが付いた **ConfigMap**、**ImageStream**、または **ImageStreamTag** オブジェクトを作成するか、初回作成後にラベルを追加すると、Kubernetes プラグイン設定に **PodTemplate** が作成されます。
- **ConfigMap** フォームの **PodTemplate** の場合には、**PodTemplate** の **ConfigMap** データへの変更は、Kubernetes プラグイン設定の **PodTemplate** 設定に適用され、**ConfigMap** に変更を加えてから次に変更を加えるまでの間に、Jenkins UI で加えた **PodTemplate** の変更が上書きされます。

Jenkins エージェントとしてコンテナーイメージを使用するには、イメージは、エントリーポイントとしてスレーブエージェントを実行する必要があります。これに関する詳細情報は、公式の Jenkins ドキュメントを参照してください。

### 10.2.7. Jenkins パーミッション

ConfigMap で、Pod テンプレート XML の **<serviceAccount>** 要素が結果として作成される Pod に使用される OpenShift Container Platform サービスアカウントである場合、サービスアカウントの認証情報が Pod にマウントされます。パーミッションはサービスアカウントに関連付けられ、OpenShift Container Platform マスターに対するどの操作が Pod から許可されるかについて制御します。

Pod に使用されるサービスアカウントについて以下のシナリオを考慮してください。この Pod は、OpenShift Container Platform Jenkins イメージで実行される Kubernetes プラグインによって起動されます。

OpenShift Container Platform で提供される Jenkins のテンプレートサンプルを使用する場合には、**jenkins** サービスアカウントが、Jenkins が実行するプロジェクトの **edit** ロールで定義され、マスター Jenkins Pod にサービスアカウントがマウントされます。

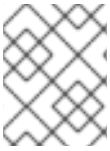
Jenkins 設定に挿入される 2 つのデフォルトの Maven および NodeJS Pod テンプレートは、Jenkins マスターと同じサービスアカウントを使用するように設定されます。

- イメージストリームまたはイメージストリームタグに必要なラベルまたはアノテーションがあるために OpenShift 同期プラグインで自動的に検出されるすべての Pod テンプレートは、Jenkins マスターのサービスアカウントをサービスアカウントとして使用するように設定されます。
- Pod テンプレートの定義を Jenkins と Kubernetes プラグインに渡す他の方法として、使用するサービスアカウントを明示的に指定する必要があります。他の方法には、Jenkins コンソール、Kubernetes プラグインで提供される **podTemplate** パイプライン DSL、または Pod テンプレートの XML 設定をデータとする ConfigMap のラベル付けなどが含まれます。
- サービスアカウントの値を指定しない場合には、**default** のサービスアカウントが使用されます。
- 使用するサービスアカウントが何であっても、必要なパーミッション、ロールなどを OpenShift Container Platform 内で定義して、Pod から操作するプロジェクトをすべて操作できるようにする必要があります。

### 10.2.8. テンプレートからの Jenkins サービスの作成

テンプレートには各種パラメーターフィールドがあり、事前定義されたデフォルト値ですべての環境変数を定義できます。OpenShift Container Platform では、新規の Jenkins サービスを簡単に作成できるようにテンプレートが提供されています。Jenkins テンプレートは、クラスター管理者が、クラスターの初期設定時に、デフォルトの **openshift** プロジェクトに登録する必要があります。

使用可能な 2 つのテンプレートは共にデプロイメント設定とサービスを定義します。テンプレートはストレージストラテジーに応じて異なります。これは、Jenkins コンテンツが Pod の再起動時に永続するかどうかに影響を与えます。



**注記**

Pod は、別のノードへの移動時や、デプロイメント設定の更新で再デプロイメントがトリガーされた時に再起動される可能性があります。

- **jenkins-ephemeral** は一時ストレージを使用します。Pod が再起動すると、すべてのデータが失われます。このテンプレートは、開発またはテストにのみ役立ちます。
- **jenkins-persistent** は永続ボリュームストアを使用します。データは Pod が再起動されても保持されます。

永続ボリュームストアを使用するには、クラスター管理者は OpenShift Container Platform デプロイメントで永続ボリュームプールを定義する必要があります。

必要なテンプレートを選択したら、テンプレートをインスタンス化して Jenkins を使用できるようにする必要があります。

**手順**

1. 以下の方法のいずれかを使用して、新しい Jenkins アプリケーションを作成します。
   - 永続ボリューム:

     ```
     $ oc new-app jenkins-persistent
     ```

   - または、Pod の再起動で設定が維持されない **emptyDir** タイプボリューム:

     ```
     $ oc new-app jenkins-ephemeral
     ```

### 10.2.9. Jenkins Kubernetes プラグインの使用

以下の例では、**openshift-jee-sample** BuildConfig により、Jenkins Maven エージェント Pod が動的にプロビジョニングされます。Pod は Java ソースコードをクローンし、WAR ファイルを作成して、2 番目の BuildConfig (**openshift-jee-sample-docker**) を実行します。2 番目の BuildConfig は、新しい WAR ファイルをコンテナーイメージに階層化します。

以下の例は、Jenkins Kubernetes プラグインを使用する BuildConfig です。

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample-docker
  spec:
    strategy:
      type: Docker
    source:
      type: Docker
      dockerfile: |-
        FROM openshift/wildfly-101-centos7:latest
        COPY ROOT.war /wildfly/standalone/deployments/ROOT.war
        CMD $STI_SCRIPTS_PATH/run
      binary:
        asFile: ROOT.war
    output:
      to:
        kind: ImageStreamTag
        name: openshift-jee-sample:latest
- kind: BuildConfig
  apiVersion: v1
  metadata:
    name: openshift-jee-sample
  spec:
    strategy:
      type: JenkinsPipeline
      jenkinsPipelineStrategy:
```

```
      jenkinsfile: |-
        node("maven") {
          sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
          sh "mvn -B -Popenshift package"
          sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
        }
    triggers:
    - type: ConfigChange
```

動的に作成された Jenkins エージェント Pod の仕様を上書きすることも可能です。以下は、コンテナーメモリーを上書きして、環境変数を指定するように先の例を変更しています。

以下の例は、Jenkins Kubernetes プラグインで、メモリー制限および環境変数を指定する BuildConfig です。

```
kind: BuildConfig
apiVersion: v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", 1
                cloud: "openshift", 2
                inheritFrom: "maven", 3
                containers: [
          containerTemplate(name: "jnlp", 4
                    image: "openshift/jenkins-agent-maven-35-centos7:v3.10", 5
                    resourceRequestMemory: "512Mi", 6
                    resourceLimitMemory: "512Mi", 7
                    envVars: [
            envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25") 8
          ])
        ]) {
          node("mypod") { 9
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
        }
    triggers:
    - type: ConfigChange
```

1. **mypod** という名前の新しい Pod テンプレートが動的に定義されます。この新しい Pod テンプレート名はノードのスタンザで参照されます。
2. **cloud** の値は **openshift** に設定する必要があります。
3. 新しい Pod テンプレートは、既存の Pod テンプレートから設定を継承できます。この場合、OpenShift Container Platform で事前定義された Maven Pod テンプレートから継承されます。
4. 

この例では、既存のコンテナーの値を上書きし、名前で指定する必要があります。OpenShift Container Platform に含まれる Jenkins エージェントイメージはすべて、コンテナー名として **jnlp**

5 再びコンテナーイメージ名を指定します。これは既知の問題です。

6 **512 Mi** のメモリー要求を指定します。

7 **512 Mi** のメモリー制限を指定します。

8 環境変数 **CONTAINER_HEAP_PERCENT** に値 **0.25** を指定します。

9 ノードスタンザは、定義された Pod テンプレート名を参照します。

デフォルトで、Pod はビルドの完了時に削除されます。この動作は、プラグインを使用して、またはパイプライン Jenkinsfile 内で変更できます。

### 10.2.10. Jenkins メモリーの要件

提供される Jenkins の一時また永続テンプレートでデプロイする場合にはデフォルトのメモリー制限は **1 Gi** です。

デフォルトで、Jenkins コンテナーで実行する他のすべてのプロセスは、合計の **512 MiB** を超えるメモリーを使用することはできません。メモリーがさらに必要になると、コンテナーは停止します。そのため、パイプラインが可能な場合に、エージェントコンテナーで外部コマンドを実行することを強く推奨されます。

また、**Project** クォータがこれを許可する場合は、Jenkins マスターがメモリーの観点から必要とするものについて、Jenkins ドキュメントの推奨事項を参照してください。この推奨事項では、Jenkins マスターにさらにメモリーを割り当てることを禁止しています。

Jenkins Kubernetes プラグインによって作成されるエージェントコンテナーで、メモリー要求および制限の値を指定することが推奨されます。管理者ユーザーは、Jenkins 設定を使用して、エージェントのイメージごとにデフォルト値を設定できます。メモリー要求および制限パラメーターは、コンテナーごとに上書きすることもできます。

Jenkins で利用可能なメモリー量を増やすには、Jenkins の一時テンプレートまたは永続テンプレートをインスタンス化する際に **MEMORY_LIMIT** パラメーターを上書きします。

### 10.2.11. その他のリソース

- Red Hat Universal Base Images (UBI) の詳細は、「ベースイメージのオプション」を参照してください。

## 10.3. JENKINS エージェント

OpenShift Container Platform では、Jenkins エージェントとして使用するのに適したイメージを 3 つのイメージ (**Base**、**Maven**、および **Node.js**) を提供します。

1 番目は、Jenkins エージェントのベースイメージです。

- 必須のツール (ヘッドレス Java、Jenkins JNLP クライアント) と有用なツール (**git**、**tar**、**zip**、**nss** など) の両方を取り入れます。
- JNLP エージェントをエントリーポイントとして確立します。

- Jenkins ジョブ内からコマンドラインの操作を呼び出す **oc** クライアントツールが含まれます。
- Red Hat Enterprise Linux (RHEL) および **localdev** イメージの両方の Dockerfile を提供します。

ベースイメージを拡張するイメージがさらに 2 つ提供されます。

- Maven v3.5 イメージ
- Node.js v8 イメージ

Maven および Node.js Jenkins エージェントイメージは、新しいエージェントイメージをビルドする際に参照できる Universal Base Image (UBI) 用の Dockerfile を提供します。**contrib** および **contrib/bin** サブディレクトリーにも注目してください。このサブディレクトリーでは、イメージの設定ファイルや実行可能なスクリプトの挿入が可能です。

**重要**

OpenShift Container Platform のバージョンに適したエージェントイメージバージョンを使用し、継承します。エージェントイメージに埋め込まれた **oc** クライアントバージョンが OpenShift Container Platform バージョンと互換性がない場合、予期しない動作が発生する可能性があります。

### 10.3.1. Jenkins エージェントイメージ

OpenShift Container Platform Jenkins エージェントイメージは **quay.io** または **registry.redhat.io** で利用できます。

Jenkins イメージは、Red Hat レジストリーから入手できます。

```
$ docker pull registry.redhat.io/openshift4/ose-jenkins:<v4.2.0>
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-nodejs:<v4.2.0>
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-maven:<v4.2.0>
$ docker pull registry.redhat.io/openshift4/ose-jenkins-agent-base:<v4.2.0>
```

これらのイメージを使用するには、**quay.io** または **registry.redhat.io** から直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュします。

### 10.3.2. Jenkins エージェントの環境変数

各 Jenkins エージェントコンテナーは、以下の環境変数で設定できます。

| 変数 | 定義 | 値と設定の例 |
|---|---|---|

| 変数 | 定義 | 値と設定の例 |
| --- | --- | --- |
| **JAVA_MAX_HEAP_PARAM**、**CONTAINER_HEAP_PERCENT**、**JENKINS_MAX_HEAP_UPPER_BOUND_MB** | これらの値は Jenkins JVM の最大ヒープサイズを制御します。**JAVA_MAX_HEAP_PARAM** が設定されている場合は、その値が優先されます。設定されていない場合は、最大ヒープサイズは、コンテナーメモリー制限の **CONTAINER_HEAP_PERCENT** として動的に計算され、オプションで **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB を上限とします。<br><br>デフォルトでは Jenkins JVM の最大ヒープサイズは、上限なしでコンテナーメモリー制限の 50% に設定されます。 | **JAVA_MAX_HEAP_PARAM** の設定例: **-Xmx512m**<br><br>**CONTAINER_HEAP_PERCENT** のデフォルト: **0.5** (50%)<br><br>**JENKINS_MAX_HEAP_UPPER_BOUND_MB** の設定例: **512 MiB** |
| **JAVA_INITIAL_HEAP_PARAM**、**CONTAINER_INITIAL_PERCENT** | これらの値は Jenkins JVM の初期ヒープサイズを制御します。**JAVA_INITIAL_HEAP_PARAM** が設定されている場合は、その値が優先されます。設定されていない場合は、初期ヒープサイズは、動的に計算される最大ヒープサイズの **CONTAINER_INITIAL_PERCENT** として動的に計算されます。<br><br>デフォルトでは、JVM は初期のヒープサイズを設定します。 | **JAVA_INITIAL_HEAP_PARAM** の設定例: **-Xmx32m**<br><br>**CONTAINER_INITIAL_PERCENT** の設定例: **0.1** (10%) |
| **CONTAINER_CORE_LIMIT** | 設定されている場合には、内部の JVM スレッドのサイジング数に使用するコアの数を整数で指定します。 | 設定例: **2** |
| **JAVA_TOOL_OPTIONS** | このコンテナーで実行中のすべての JVM に適用するオプションを指定します。この値の上書きは推奨していません。 | デフォルト: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true** |

| 変数 | 定義 | 値と設定の例 |
| --- | --- | --- |
| **JAVA_GC_OPTS** | Jenkins JVM ガーベッジコレクションのパラメーターを指定します。この値の上書きは推奨していません。 | デフォルト: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90** |
| **JENKINS_JAVA_OVERRIDES** | Jenkins JVM の追加オプションを指定します。これらのオプションは、上記の Java オプションを含む、その他すべてのオプションに追加され、必要に応じてそれらのいずれかを上書きするために使用できます。追加オプションがある場合には、スペースで区切ります。オプションにスペース文字が含まれる場合には、バックスラッシュでエスケープしてください。 | 設定例: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value** |

### 10.3.3. Jenkins エージェントのメモリー要件

JVM はすべての Jenkins エージェントで使用され、Jenkins JNLP エージェントをホストし、**javac**、Maven、または Gradle などの Java アプリケーションを実行します。

デフォルトで、Jenkins JNLP エージェントの JVM はヒープにコンテナーメモリー制限の 50% を使用します。この値は、**CONTAINER_HEAP_PERCENT** の環境変数で変更可能です。上限を指定することも、すべて上書きすることも可能です。

デフォルトでは、シェルスクリプトや **oc** コマンドをパイプラインから実行するなど、Jenkins エージェントコンテナーで実行される他のプロセスはすべて、OOM kill を呼び出さずに残りの 50% メモリー制限を超えるメモリーを使用することはできません。

デフォルトでは、Jenkins エージェントコンテナーで実行される他の各 JVM プロセスは、最大でコンテナーメモリー制限の 25% をヒープに使用します。多くのビルドワークロードにおいて、この制限の調整が必要になる場合があります。

### 10.3.4. Jenkins エージェントの Gradle ビルド

OpenShift Container Platform の Jenkins エージェントで Gradle ビルドをホストすると、Jenkins JNLP エージェントおよび Gradle JVM に加え、テストが指定されている場合に Gradle が 3 番目の JVM を起動してテストを実行するので、さらに複雑になります。

以下の設定は、OpenShift Container Platform でメモリーに制約がある Jenkins エージェントの Gradle ビルドを実行する場合の開始点として推奨されます。必要に応じて、これらの設定を変更することができます。

- **gradle.properties** ファイルに **org.gradle.daemon=false** を追加して、有効期間の長い (long-lived) Gradle デーモンを無効にするようにします。

- **gradle.properties** ファイルで **org.gradle.parallel=true** が設定されていないこと、また、コマンドラインの引数として **--parallel** が設定されていないことを確認して、並行ビルドの実行を無効にします。
- **java { options.fork = false }** を **build.gradle** ファイルに設定して、プロセス以外で Java がコンパイルされないようにします.
- **build.gradle** ファイルで **test { maxParallelForks = 1 }** が設定されていることを確認して、複数の追加テストプロセスを無効にします。
- **GRADLE_OPTS**、**JAVA_OPTS**、または **JAVA_TOOL_OPTIONS** 環境変数で、Gradle JVM メモリーパラメーターを上書きします。
- **buile.gradle** の **maxHeapSize** および **jvmArgs** 設定を定義するか、**-Dorg .gradle. jvmargs** コマンドライン引数を使用して、Gradle テスト JVM に最大ヒープサイズと JVM の引数を設定します。

### 10.3.5. Jenkins エージェント Pod の保持

Jenkins エージェント Pod (別名: スレーブ Pod) は、ビルドが完了するか、または停止された後にデフォルトで削除されます。この動作は、Kubernetes プラグインの **Pod Retention** (Pod の保持) 設定で変更できます。Pod の保持は、すべての Jenkins ビルドについて各 Pod テンプレートの上書きで設定できます。以下の動作がサポートされます。

- **Always** は、ビルドの結果に関係なくビルド Pod を維持します。
- **Default** はプラグイン値を使用します (Pod テンプレートのみ)。
- **Never** は常に Pod を削除します。
- **On Failure** は Pod がビルド時に失敗した場合に Pod を維持します。

Pod の保持はパイプライン Jenkinsfile で上書きできます。

```
podTemplate(label: "mypod",
  cloud: "openshift",
  inheritFrom: "maven",
  podRetention: onFailure(), 1
  containers: [
    ...
  ]) {
  node("mypod") {
    ...
  }
}
```

1 **podRetention** に許可される値は、**never()**、**onFailure()**、**always()**、および **default()** です。

**警告**

保持される Pod は実行し続け、リソースクォータに対してカウントされる可能性があります。