OpenShift Container Platform 3.11

イメージの使用

OpenShift Container Platform 3.11 でのイメージの使用ガイド

Last Updated: 2019-01-22

OpenShift Container Platform 3.11 イメージの使用

OpenShift Container Platform 3.11 でのイメージの使用ガイド

法律上の通知

概要

以下のトピックを使用して、OpenShift Container Platform 3.11 でユーザーに提供されている、さまざまな S2I (Source-to-Image)、データベース、Docker イメージについて確認してください。

目次

第1章 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、他のコンテナーイメージについて確認してください。

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 レジストリーの**openshift3** リポジトリー にあります。たとえば、Atomic OpenShift Application Platform イメージは **registry.redhat.io/openshift3/ose** にあります。

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、JBoss EAP イメージの場合は、**registry.redhat.io/jboss-eap-6/eap64-openshift** です。

本書で説明する 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。各イメージの全バージョンに関するコンテンツや用途の詳細が分かります。興味のあ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 のバージョンをもとに、正しいバージョンのコンテナーイメージを確認、使用するようにしてください。

第2章 SOURCE-TO-IMAGE (S2I)

2.1. 概要

以下のトピックには、OpenShift Container Platform のユーザーに提供される、さまざまな S2I (Source-to-Image) 対応イメージに関する情報が含まれます。

2.2. .NET CORE

2.2.1. .NET Core を使用する利点

.NET Core は、自動メモリー管理および最新のプログラム言語が搭載された、汎用の開発プラットフォームです。.NET Core では、効率的に高品質のアプリケーションをビルドできます。.NET Core は、認定済みのコンテナー経由で Red Hat Enterprise Linux (RHEL 7) および OpenShift Container Platform から利用できます。.NET Core は以下の機能を提供します。

- マイクロサービスベースのアプローチに従う機能。この機能では、.NET でビルドされているコンポーネントと、Java でビルドされているコンポーネントがありますが、Red Hat Enterprise Linux や OpenShift Container Platform でサポートされている一般的なプラットフォームにおいてすべて実行可能です。
- Windows で新しい .NET Core ワークロードをより簡単に開発する機能。Red Hat Enterprise Linux または Windows Server のいずれかでデプロイおよび実行が可能です。
- 基礎となるインフラストラクチャーが Windows Server のみに依存することなしに、.NET アプリケーションを実行できる異種のデータセンター。
- OpenShift Container Platform から .NET、Java、Ruby および Python などの一般的な開発フレームワークの多くにアクセスできる機能。

2.2.2. サポートされているバージョン

- .NET Core バージョン 2.2
- .NET Core バージョン 2.1
- .NET Core バージョン 1.1
- .NET Core バージョン 1.0
- Red Hat Enterprise Linux (RHEL) 7 および OpenShift Container Platform バージョン 3.3 以降でサポートされています。

.NET Core バージョン 2.2 関連のリリースの詳細は、『Release Notes for Containers』を参照してください。

.NET Core バージョン 2.1 関連のリリースの詳細は、『Release Notes for Containers』を参照してください。

バージョン 1.1 および 1.0 (**rh-dotnetcore11** および **rh-dotnetcore10**) には **project.json** ビルドシステム (**1.0.0-preview2** SDK) が同梱されています。RHEL システム以外にこの SDK をインストールする方法については、『version 1.1 Release Notes』の既知の問題についての章を参照してください。

2.2.3. イメージ

RHEL 7 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/dotnet/dotnet-22-rhel7
$ docker pull registry.redhat.io/dotnet/dotnet-21-rhel7
$ docker pull registry.redhat.io/dotnet/dotnetcore-11-rhel7
$ docker pull registry.redhat.io/dotnet/dotnetcore-10-rhel7
```

RHEL S2I イメージ上の .NET Core 向けのイメージストリーム定義は、OpenShift Container Platform のインストール時に追加されるようになりました。

2.2.4. ビルドプロセス

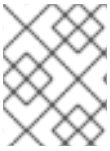
S2I は、ソースコードをコンテナーに挿入し、コンテナーにソースコードの実行を準備をさせることで、実行準備が整ったイメージを生成します。S2I では、以下の手順を実行します。

1. ビルダーイメージからコンテナーを起動します。
2. アプリケーションソースをダウンロードします。
3. ビルダーイメージコンテナーにスクリプトとアプリケーションソースをストリーミングします。
4. (ビルダーイメージから) **assemble** スクリプトを実行します。
5. 最終的なイメージを保存します。

ビルドプロセスの詳細のまとめについては、「S2I ビルドプロセス」を参照してください。

2.2.5. 環境変数

.NET Core イメージは、複数の環境変数をサポートします。この環境変数を設定して、.NET Core アプリケーションのビルド動作を制御することができます。



注記

S2I ビルド設定または **.s2i/environment** ファイルに、ビルドの動作を制御する環境変数を設定して、ビルドの手順で利用できるようにする必要があります。

表2.1 NET Core 環境変数

変数名	説明	デフォルト
DOTNET_STARTUP_PROJECT	実行するプロジェクトを選択します。これは、プロジェクトファイル (例: **csproj** または **fsproj**) か、単一のプロジェクトファイルを含むフォルダーでなければなりません。	**.**

変数名	説明	デフォルト
DOTNET_SDK_VERSION	ビルド時のデフォルトの SDK バージョンを選択します。ソースリポジトリーに **global.json** ファイルがある場合には、これが優先されます。**latest** に設定した場合は、イメージの最新の SDK が使用されます。	イメージで利用可能な、一番古い SDK バージョン
DOTNET_ASSEMBLY_NAME	実行するアセンブリーを選択します。これには、**.dll** の拡張子を **含めないでください**。これは、**csproj** で指定した出力アセンブリー名に設定します (**PropertyGroup**/**AssemblyName**)。	**csproj** ファイルの名前
DOTNET_RESTORE_SOURCES	復元操作時に使用する **NuGet** パッケージソースをスペース区切りの一覧で指定します。これは、**NuGet.config** ファイルで指定したすべてのソースよりも優先されます。	
DOTNET_TOOLS	アプリケーションをビルドする前にインストールする .NET ツールの一覧を指定します。固有のバージョンをインストールするには、パッケージ名の最後に **@<version>** を追加します。	
DOTNET_NPM_TOOLS	アプリケーションをビルドする前にインストールする NPM パッケージの一覧を指定します。	
DOTNET_TEST_PROJECTS	テストするテストプロジェクトの一覧を指定します。これは、プロジェクトファイルか、または単一のプロジェクトファイルを含むフォルダーでなければなりません。**dotnet test** はアイテムごとに呼び出されます。	
DOTNET_CONFIGURATION	**Debug** または **Release** モードでアプリケーションを実行します。この値は、**Release** または **Debug** のいずれかに指定する必要があります。	**Release**

変数名	説明	デフォルト
DOTNET_VERBOSITY	dotnet ビルドコマンドの詳細レベルを指定します。これを設定した場合には、環境変数がビルドの開始時に出力されます。変数は、msbuild の詳細値 (**q[uiet]**、**m[inimal]**, **n[ormal]**、**d[etailed]** および **diag[nostic]**) の 1 つに設定できます。	
HTTP_PROXY、**HTTPS_PROXY**	アプリケーションのビルド時および実行時に使用する HTTP/HTTPS プロキシーを設定します。	
NPM_MIRROR	ビルドプロセス中にパッケージをダウンロードするカスタムの NPM レジストリーミラーを使用します。	
ASPNETCORE_URLS	この変数を **http://*:8080** に設定して ASP.NET Core がイメージが公開するポートを使用するように設定します。この値の変更は推奨 **していません**。	**http://*:8080**
DOTNET_RM_SRC	**true** に設定されると、ソースコードはイメージに組み込まれません。	
DOTNET_SSL_DIRS	追加で信頼する SSL 証明書を使ってフォルダーおよびファイルの一覧を指定するために使用されます。証明書は、ビルド時に実行される各プロセス、およびビルドされたアプリケーションを含め、ビルド後にイメージで実行されるすべてのプロセスで信頼されます。使用される項目には、**/** で開始される絶対パスまたはソースリポジトリーのパスを指定できます (証明書など)。	
DOTNET_RESTORE_DISABLE_PARALLEL	**true** に設定されている場合、複数プロジェクトを並行して復元する操作を無効にします。これにより、ビルドコンテナーが CPU 制限の値が低く設定された状態で実行されている場合にも復元タイムアウトのエラーが減少します。	**false**

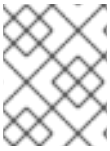
2.2.6. .NET Core ソースからのアプリケーションのクイックデプロイ

重要

.NET イメージストリーム は最初にインストールする必要があります。標準インストールを実行した場合には、イメージストリームは存在します。

サンプルのリポジトリーに対して **oc new-app** を実行すると、イメージを使用してアプリケーションをビルドできます。

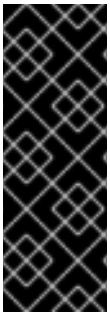
```
$ oc new-app registry.redhat.io/dotnet/dotnet-22-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-2.2
--context-dir=app
$ oc new-app registry.redhat.io/dotnet/dotnet-21-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-2.1
--context-dir=app
$ oc new-app registry.redhat.io/dotnet/dotnetcore-11-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-1.1
--context-dir=app
$ oc new-app registry.redhat.io/dotnet/dotnetcore-10-
rhel7~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-1.0
--context-dir=app
```



注記

oc new-app コマンドは、OpenShift Container Platform 3.3 より .NET Core ソースを検出できます。

2.2.7. .NET Core テンプレート



重要

.NET イメージテンプレート および .NET イメージストリームは先にインストールする必要があります。標準インストールを実行した場合にはテンプレートとイメージストリームは存在します。これは、以下のコマンドで確認できます。

```
$ (oc get -n openshift templates; oc get -n openshift is) |
grep dotnet
```

OpenShift Container Platform には .NET Core イメージのテンプレートが含まれており、サンプルアプリケーションを簡単にデプロイしやすくなります。

dotnet/dotnet-22-rhel7 で実行する .NET Core のサンプルアプリケーション は以下のコマンドでデプロイ可能です。

```
$ oc new-app --template dotnet-example -p
DOTNET_IMAGE_STREAM_TAG=dotnet:2.2 -p SOURCE_REPOSITORY_REF=dotnetcore-2.2
```

dotnet/dotnetcore-10-rhel7 で実行する .NET Core のサンプルアプリケーション は以下のコマンドでデプロイ可能です。

```
$ oc new-app --template dotnet-example
```

PostgreSQL をデータベースとして使用した .NET Core MusicStore アプリケーション は、以下のコマンドでデプロイ可能です。

```
$ oc new-app --template=dotnet-pgsql-persistent
```

2.3. NODE.JS

2.3.1. 概要

OpenShift Container Platform には 、Node.js アプリケーションのビルドおよび実行用に S2I が有効な Node.js イメージが含まれています。Node.js S2I ビルダーイメージは、必要な依存関係を使用してアプリケーションソースを組み立てて、Node.js アプリケーションを含む新規イメージを作成します。このように作成されたイメージは、OpenShift Container Platform またはコンテナーランタイムで実行可能です。

2.3.2. バージョン

現在、OpenShift Container Platform では、Node.js のバージョン 0.10、4 および 6 を提供しています。

2.3.3. イメージ

これらのイメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/openshift3/nodejs-010-rhel7
$ docker pull registry.redhat.io/rhscl/nodejs-4-rhel7
```

CentOS 7 ベースのイメージ

このイメージは、Docker Hub で入手できます。

```
$ docker pull openshift/nodejs-010-centos7
```

これらのイメージを使用するには、イメージレジストリー から直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリー にプッシュしてください。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 イメージストリーム を作成することもでき、OpenShift Container Platform リソースがこのイメージストリームを参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して イメージストリームの定義例 があります。

2.3.4. ビルドプロセス

S2I は、ソースコードをコンテナーに挿入し、コンテナーにソースコードの実行を準備をさせることで、実行準備が整ったイメージを生成します。S2I では、以下の手順を実行します。

1. ビルダーイメージからコンテナーを起動します。
2. アプリケーションソースをダウンロードします。
3. ビルダーイメージコンテナーにスクリプトとアプリケーションソースをストリーミングします。
4. (ビルダーイメージから) **assemble** スクリプトを実行します。
5. 最終的なイメージを保存します。

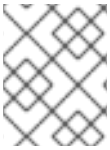
ビルドプロセスの詳細のまとめについては、「S2I ビルドプロセス」を参照してください。

2.3.5. 設定

Node.js イメージは、環境変数を複数サポートし、環境変数を設定することで Node.js のラインタイムの設定や動作を制御できます。

イメージの一部としてこれらの環境変数を設定するには、ソースコードリポジトリーの中にある **.s2i/environment** ファイル に配置するか、ビルド設定の **sourceStrategy** 定義の 環境セクション に定義します。

また、新規アプリケーションの作成時に 既存のイメージを使用するか、デプロイメント設定などの 既存のオブジェクトの環境変数を更新して 環境変数を設定できます。



注記

ビルドの動作を制御する環境変数は、s2i ビルド設定または **.s2i/environment** ファイルの一部として設定して、ビルドの手順で利用できるようにする必要があります。

表2.2 開発モードの環境変数

変数名	説明
DEV_MODE	**true** に設定されている場合には、ホットデプロイを有効にし、デバッグポートを開きます。さらに、ツールに対して、イメージが開発モードであることを指定します。デフォルトは **false** です。
DEBUG_PORT	デバッグポート。**DEV_MODE** が true に設定されている場合のみ有効です。デフォルトは 5858 です。
NPM_MIRROR	カスタムの NPM レジストリーのミラー URL。全 NPM パッケージはビルドプロセス中にミラーリンクからダウンロードされます。

2.3.6. ホットデプロイ

ホットデプロイでは、新しい S2I ビルドを生成する必要なしに、アプリケーションに変更をすばやく加え、デプロイすることができます。アプリケーションのソースコードに加えられた変更を即座に検出するには、環境変数を **DEV_MODE=true** に指定してビルドイメージを実行する必要があります。

新規アプリケーションの作成時 または 既存のオブジェクトの環境変数の更新時 に、新しい環境変数を設定できます。

警告

DEV_MODE=true の環境変数は、開発時またはデバッグ時にのみ使用するようにしてください。この変数の実稼働環境での使用は推奨されていません。

実行中の Pod のソースコードを変更するには、コンテナーに対するリモートシェルを開きます。

```
$ oc rsh <pod_id>
```

実行中のコンテナーに入ると、現在のディレクトリーは、ソースコードが配置されている **/opt/app-root/src** に変わります。

2.4. PERL

2.4.1. 概要

OpenShift Container Platform には 、Perl アプリケーションのビルドおよび実行用に S2I が有効な Perl イメージが含まれています。Perl S2I ビルダーイメージは、必要な依存関係を使用してアプリケーションソースを組み立てて、Perl アプリケーションを含む新規イメージを作成します。このように作成されたイメージは、OpenShift Container Platform またはコンテナーランタイムで実行可能です。

2.4.2. バージョン

現在、OpenShift Container Platform は Perl バージョン 5.16、5.20 および 5.24 をサポートします。

2.4.3. イメージ

イメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/openshift3/perl-516-rhel7
$ docker pull registry.redhat.io/rhscl/perl-520-rhel7
$ docker pull registry.redhat.io/rhscl/perl-524-rhel7
```

CentOS 7 ベースのイメージ

Perl 5.16 の CentOS イメージは、Docker Hub で入手できます。

```
$ docker pull openshift/perl-516-centos7
```

これらのイメージを使用するには、イメージレジストリー から直接アクセスするか、ご自身の

OpenShift Container Platform コンテナーイメージレジストリー にプッシュしてください。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 イメージストリーム を作成することもでき、OpenShift Container Platform リソースがこのイメージストリームを参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して イメージストリームの定義例 があります。

2.4.4. ビルドプロセス

S2I は、ソースコードをコンテナーに挿入し、コンテナーにソースコードの実行を準備をさせることで、実行準備が整ったイメージを生成します。S2I では、以下の手順を実行します。

1. ビルダーイメージからコンテナーを起動します。
2. アプリケーションソースをダウンロードします。
3. ビルダーイメージコンテナーにスクリプトとアプリケーションソースをストリーミングします。
4. (ビルダーイメージから) **assemble** スクリプトを実行します。
5. 最終的なイメージを保存します。

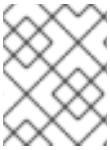
ビルドプロセスの詳細のまとめについては、「S2I ビルドプロセス」を参照してください。

2.4.5. 設定

Perl イメージは多数の環境変数を複数サポートし、環境変数を設定することで Perl のラインタイムの設定や動作を制御できます。

イメージの一部としてこれらの環境変数を設定するには、ソースコードリポジトリーの中にある **.s2i/environment** ファイル に配置するか、ビルド設定の **sourceStrategy** 定義の 環境セクション に定義します。

また、新規アプリケーションの作成時に 既存のイメージを使用するか、デプロイメント設定などの 既存のオブジェクトの環境変数を更新して 環境変数を設定できます。



注記

ビルドの動作を制御する環境変数は、s2i ビルド設定または **.s2i/environment** ファイルの一部として設定して、ビルドの手順で利用できるようにする必要があります。

表2.3 Perl 環境変数

変数名	説明
ENABLE_CPAN_TEST	**true** に設定している場合は、この変数は全 cpan モジュールをインストールして、そのテストを実行します。デフォルトでは、モジュールのテストはオフになっています。
CPAN_MIRROR	この変数は、cpanminus が依存関係のインストールに使用するミラーの URL を指定します。デフォルトではこの URL は指定されていません。

変数名	説明
PERL_APACHE2_RELOAD	これを **true** に設定すると、変更した Perl モジュールの自動再読み込みが有効になります。デフォルトでは、自動再読み込みはオフになっています。
HTTPD_START_SERVERS	StartServers ディレクティブは、起動時に作成される子サーバープロセスの数を設定します。デフォルトは 8 です。
HTTPD_MAX_REQUEST_WORKERS	Apache により処理される同時要求の数。デフォルトは 256 ですが、メモリーに制限がある場合は自動的に数値が下がります。

2.4.6. ログへのアクセス

アクセスログは、標準出力にストリーミングされるので、**oc logs** コマンドを使用して表示可能です。エラーログは **/tmp/error_log** ファイルに保存されているので、コンテナーにアクセスする **oc rsh** コマンドを使用して表示できます。

2.4.7. ホットデプロイ

ホットデプロイでは、新しい S2I ビルドを生成する必要なしに、アプリケーションに変更をすばやく加え、デプロイすることができます。このイメージでホットデプロイを有効化するには、**PERL_APACHE2_RELOAD** 環境変数を **true** に設定する必要があります。たとえば、**oc new-app** コマンドを参照してください。**oc set env** コマンドを使用して、既存オブジェクトの環境変数を更新できます。

警告

このオプションは、開発またはデバッグの時にだけ使用するようにしてください。実稼働環境でこの設定をオンにすることは推奨しません。

実行中の Pod でソースコードを変更するには、**oc rsh** コマンドを使用して、コンテナーに入ります。

```
$ oc rsh <pod_id>
```

実行中のコンテナーに入った後に、現在のディレクトリーを、ソースコードが配置されている **/opt/app-root/src** に設定します。

2.5. PHP

2.5.1. 概要

OpenShift Container Platform には 、PHP アプリケーションのビルドおよび実行用に S2I が有効な PHP イメージが含まれています。PHP S2I ビルダーイメージは、必要な依存関係を使用してアプリケーションソースを組み立てて、PHP アプリケーションを含む新規イメージを作成します。このように

作成されたイメージは、OpenShift Container Platform またはコンテナーランタイムで実行可能です。

2.5.2. バージョン

現在、OpenShift Container Platform では、PHP のバージョン 5.5、5.6 および 7.0 を提供しています。

2.5.3. イメージ

これらのイメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/openshift3/php-55-rhel7
$ docker pull registry.redhat.io/rhscl/php-56-rhel7
$ docker pull registry.redhat.io/rhscl/php-70-rhel7
```

CentOS 7 ベースのイメージ

PHP 5.5 および 5.6 の CentOS イメージは、Docker Hub で入手できます。

```
$ docker pull openshift/php-55-centos7
$ docker pull openshift/php-56-centos7
```

これらのイメージを使用するには、イメージレジストリー から直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリー にプッシュしてください。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 イメージストリーム を作成することもでき、OpenShift Container Platform リソースがこのイメージストリームを参照できるようになります。

提供される全 OpenShift Container Platform イメージに対して イメージストリームの定義例 があります。

2.5.4. ビルドプロセス

S2I は、ソースコードをコンテナーに挿入し、コンテナーにソースコードの実行を準備をさせることで、実行準備が整ったイメージを生成します。S2I では、以下の手順を実行します。

1. ビルダーイメージからコンテナーを起動します。
2. アプリケーションソースをダウンロードします。
3. ビルダーイメージコンテナーにスクリプトとアプリケーションソースをストリーミングします。
4. (ビルダーイメージから) **assemble** スクリプトを実行します。
5. 最終的なイメージを保存します。

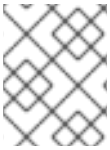
ビルドプロセスの詳細のまとめについては、「S2I ビルドプロセス」を参照してください。

2.5.5. 設定

PHP イメージは数多くの環境変数を複数サポートし、環境変数を設定することで PHP のラインタイムの設定や動作を制御できます。

イメージの一部としてこれらの環境変数を設定するには、ソースコードリポジトリーの中にある **.s2i/environment** ファイル に配置するか、ビルド設定の **sourceStrategy** 定義の 環境セクション に定義します。

また、新規アプリケーションの作成時に 既存のイメージを使用するか、デプロイメント設定などの 既存のオブジェクトの環境変数を更新して 環境変数を設定できます。



注記

ビルドの動作を制御する環境変数は、s2i ビルド設定または **.s2i/environment** ファイルの一部として設定して、ビルドの手順で利用できるようにする必要があります。

以下の環境変数は、**php.ini** ファイルに同等のプロパティー値を設定します。

表2.4 PHP 環境変数

変数名	説明	デフォルト
ERROR_REPORTING	PHP で対応する必要のあるエラー、警告、注意を PHP に通知します。	**E_ALL & ~E_NOTICE**
DISPLAY_ERRORS	PHP がエラー、注意、警告を出力するかどうか、さらに出力先を制御します。	**ON**
DISPLAY_STARTUP_ERRORS	PHP の起動シーケンス時に発生した表示エラーを通常の表示エラーとは別に処理するようにします。	**OFF**
TRACK_ERRORS	**$php_errormsg** (boolean) に最後のエラー/警告メッセージを保存します。	**OFF**
HTML_ERRORS	対象のエラーに関連するドキュメントにエラーをリンクします。	**ON**
INCLUDE_PATH	PHP ソースファイルへのパス	**.:/opt/openshift/src:/opt/rh/php55/root/usr/share/pear**
SESSION_PATH	セッションデータファイルの場所	**/tmp/sessions**
DOCUMENTROOT	アプリケーションのドキュメントルートを定義するパス (例: **/public**)	**/**

以下の環境変数は、**opcche.ini** ファイルに同等のプロパティー値を設定します。

表2.5 PHP の他の設定

変数名	説明	デフォルト
OPCACHE_MEMORY_CONSUMPTION	OPcache 共有メモリーのストレージサイズ	**16M**
OPCACHE_REVALIDATE_FREQ	更新のスクリプトタイムスタンプをどの頻度で確認するかを秒単位で指定します。**0** に指定すると、OPcache はすべての要求の更新を確認します。	**2**

以下を設定して PHP 設定の読み込みに使用するディレクトリー全体を上書きすることも可能です。

表2.6 PHP の他の設定

変数名	説明
PHPRC	**php.ini** ファイルにパスを設定します。
PHP_INI_SCAN_DIR	追加の **.ini** 設定ファイル

デフォルトの 'packagist.org' ではなく、カスタムの Composer リポジトリーのミラー URL を使用して、パッケージをダウンロードできます。

表2.7 Composer 環境変数

変数名	説明	COMPOSER_MIRROR

2.5.5.1. Apache 設定

アプリケーションの **DocumentRoot** がソースディレクトリーの **/opt/openshift/src** にネストされている場合には、独自の **.htaccess** ファイルで、デフォルトの Apache の動作を置き換え、アプリケーションの要求の処理方法を指定することができます。**.htaccess** ファイルは、アプリケーションソースのルートに配置する必要があります。

2.5.6. ログへのアクセス

アクセスログは、標準出力にストリーミングされるので、**oc logs** コマンドを使用して表示可能です。エラーログは **/tmp/error_log** ファイルに保存されているので、コンテナーにアクセスする **oc rsh** コマンドを使用して表示できます。

2.5.7. ホットデプロイ

ホットデプロイでは、新しい S2I ビルドを生成する必要なしに、アプリケーションに変更をすばやく加え、デプロイすることができます。アプリケーションのソースコードに加えられた変更を即座に検出するには、環境変数を **OPCACHE_REVALIDATE_FREQ=0** に指定してビルドイメージを実行する必要があります。

たとえば、**oc new-app** コマンドを参照してください。**oc env** コマンドを使用して、既存オブジェクトの環境変数を更新できます。

警告

このオプションは、開発またはデバッグの時にだけ使用するようにしてください。実稼働環境でこの設定をオンにすることは推奨しません。

実行中の Pod でソースコードを変更するには、**oc rsh** コマンドを使用して、コンテナーに入ります。

```
$ oc rsh <pod_id>
```

実行中のコンテナーに入った後に、現在のディレクトリーを、ソースコードが配置されている **/opt/app-root/src** に設定します。

2.6. PYTHON

2.6.1. 概要

OpenShift Container Platform には 、Python アプリケーションのビルドおよび実行用に S2I が有効な Python イメージが含まれています。Python S2I ビルダーイメージは、必要な依存関係を使用してアプリケーションソースを組み立てて、Python アプリケーションを含む新規イメージを作成します。このように作成されたイメージは、OpenShift Container Platform またはコンテナーランタイムで実行可能です。

2.6.2. バージョン

現在、OpenShift Container Platform では、Python のバージョン 2.7、3.3、3.4 および 3.5 を提供しています。

2.6.3. イメージ

これらのイメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/rhscl/python-27-rhel7
$ docker pull registry.redhat.io/openshift3/python-33-rhel7
$ docker pull registry.redhat.io/rhscl/python-34-rhel7
$ docker pull registry.redhat.io/rhscl/python-35-rhel7
```

CentOS 7 ベースのイメージ

これらのイメージは、Docker Hub で入手できます。

```
$ docker pull centos/python-27-centos7
$ docker pull openshift/python-33-centos7
$ docker pull centos/python-34-centos7
$ docker pull centos/python-35-centos7
```

これらのイメージを使用するには、イメージレジストリー から直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリー にプッシュしてください。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 イメージストリーム を作成することもでき、OpenShift Container Platform リソースがこのイメージストリームを参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して イメージストリームの定義例 があります。

2.6.4. ビルドプロセス

S2I は、ソースコードをコンテナーに挿入し、コンテナーにソースコードの実行を準備をさせることで、実行準備が整ったイメージを生成します。S2I では、以下の手順を実行します。

1. ビルダーイメージからコンテナーを起動します。
2. アプリケーションソースをダウンロードします。
3. ビルダーイメージコンテナーにスクリプトとアプリケーションソースをストリーミングします。
4. (ビルダーイメージから) **assemble** スクリプトを実行します。
5. 最終的なイメージを保存します。

ビルドプロセスの詳細のまとめについては、「S2I ビルドプロセス」を参照してください。

2.6.5. 設定

Python イメージは多数の環境変数を複数サポートし、環境変数を設定することで Python のラインタイムの設定や動作を制御できます。

イメージの一部としてこれらの環境変数を設定するには、ソースコードリポジトリーの中にある **.s2i/environment** ファイル に配置するか、ビルド設定の **sourceStrategy** 定義の 環境セクション に定義します。

また、新規アプリケーションの作成時に 既存のイメージを使用するか、デプロイメント設定などの 既存のオブジェクトの環境変数を更新して 環境変数を設定できます。

注記

ビルドの動作を制御する環境変数は、s2i ビルド設定または **.s2i/environment** ファイルの一部として設定して、ビルドの手順で利用できるようにする必要があります。

表2.8 Python 環境変数

変数名	説明

変数名	説明
APP_FILE	この変数は、アプリケーションを起動する Python インタープリターに渡すファイル名を指定します。デフォルトでは、この変数は **app.py** に設定されています。
APP_MODULE	この変数は WSGI 呼び出し可能なオブジェクトを指定します。この変数のパターンは **$(MODULE_NAME):$(VARIABLE_NAME)** で、モジュール名はドットのフルパスに指定し、変数名は指定のモジュール内の関数を参照します。アプリケーションのインストールに **setup.py** を使用した場合に、モジュール名はファイルから読み込むことができ、変数は **application** に設定されます。setup-test-app のサンプルがあります。
APP_CONFIG	この変数は、gunicorn 設定 で有効な Python ファイルへのパスを指定します。
DISABLE_COLLECTSTATIC	空でない値に設定して、ビルド時に **manage.py collectstatic** が実行されないようにします。これは Django プロジェクトに対してのみ影響があります。
DISABLE_MIGRATE	空でない値に設定して、生成されたイメージの実行時に **manage.py migrate** が実行されないようにします。これは Django プロジェクトに対してのみ影響があります。
PIP_INDEX_URL	ビルドプロセス時に必要なパッケージをダウンロードするための、カスタムのインデックス URL またはミラーを使用するように、この変数を設定します。これは、**requirements.txt** ファイルに記載のパッケージにのみ影響があります。
WEB_CONCURRENCY	これを設定して、ワーカー 数のデフォルト設定を変更します。デフォルトでは、これは利用可能なコアに 4 をかけた数字に設定されています。

2.6.6. ホットデプロイ

ホットデプロイでは、新しい S2I ビルドを生成する必要なしに、アプリケーションに変更をすばやく加え、デプロイすることができます。Django を使用する場合は、ホットデプロイメントはカスタマイズなしに使用できます。

Gunicorn を使用したホットデプロイメントを有効にするには、**reload** オプション を **true** に設定して、リポジトリーに Gunicorn 設定ファイルが配置されているようにします。設定ファイルは、**APP_CONFIG** 環境変数を使用して指定します。たとえば、**oc new-app** コマンドを参照してください。**oc set env** コマンドを使用して、既存オブジェクトの環境変数を更新できます。

警告

このオプションは、開発またはデバッグの時にだけ使用するようにしてください。実稼働環境でこの設定をオンにすることは推奨しません。

実行中の Pod でソースコードを変更するには、**oc rsh** コマンドを使用して、コンテナーに入ります。

```
$ oc rsh <pod_id>
```

実行中のコンテナーに入った後に、現在のディレクトリーを、ソースコードが配置されている **/opt/app-root/src** に設定します。

2.7. RUBY

2.7.1. 概要

OpenShift Container Platform には、Ruby アプリケーションのビルドおよび実行用に S2I が有効な Ruby イメージが含まれています。Ruby S2I ビルダーイメージは、必要な依存関係を使用してアプリケーションソースを組み立てて、Ruby アプリケーションを含む新規イメージを作成します。このように作成されたイメージは、OpenShift Container Platform またはコンテナーランタイムで実行可能です。

2.7.2. バージョン

現時点で、OpenShift Container Platform では、Ruby のバージョン 2.0、2.2 および 2.3 を提供しています。

2.7.3. イメージ

これらのイメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/openshift3/ruby-20-rhel7
$ docker pull registry.redhat.io/rhscl/ruby-22-rhel7
$ docker pull registry.redhat.io/rhscl/ruby-23-rhel7
```

CentOS 7 ベースのイメージ

これらのイメージは、Docker Hub で入手できます。

```
$ docker pull openshift/ruby-20-centos7
$ docker pull openshift/ruby-22-centos7
$ docker pull centos/ruby-23-centos7
```

これらのイメージを使用するには、イメージレジストリー から直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリー にプッシュしてください。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 イメージストリーム を作成することもでき、OpenShift Container Platform リソースがこのイメージストリームを参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して イメージストリームの定義例 があります。

2.7.4. ビルドプロセス

S2I は、ソースコードをコンテナーに挿入し、コンテナーにソースコードの実行を準備をさせることで、実行準備が整ったイメージを生成します。S2I では、以下の手順を実行します。

1. ビルダーイメージからコンテナーを起動します。
2. アプリケーションソースをダウンロードします。
3. ビルダーイメージコンテナーにスクリプトとアプリケーションソースをストリーミングします。
4. (ビルダーイメージから) **assemble** スクリプトを実行します。
5. 最終的なイメージを保存します。

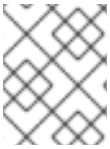
ビルドプロセスの詳細のまとめについては、「S2I ビルドプロセス」を参照してください。

2.7.5. 設定

Ruby イメージは多数の環境変数を複数サポートし、環境変数を設定することで Ruby のランタイムの設定や動作を制御できます。

イメージの一部としてこれらの環境変数を設定するには、ソースコードリポジトリーの中にある **.s2i/environment** ファイル に配置するか、ビルド設定の **sourceStrategy** 定義の 環境セクション に定義します。

また、新規アプリケーションの作成時に 既存のイメージを使用するか、デプロイメント設定などの 既存のオブジェクトの環境変数を更新して 環境変数を設定できます。



注記

ビルドの動作を制御する環境変数は、s2i ビルド設定または **.s2i/environment** ファイルの一部として設定して、ビルドの手順で利用できるようにする必要があります。

表2.9 Ruby 環境変数

変数名	説明

変数名	説明
RACK_ENV	この変数は、Ruby アプリケーションがデプロイされる環境を指定します。たとえば **production**、**development** または **test** などです。ロギングの詳細レベル、エラーページ、Ruby gem インストールなど、レベルごとに動作が異なります。アプリケーションのアセットは、**RACK_ENV** が **production** に設定されている場合にのみコンパイルされます。デフォルト値は **production** です。
RAILS_ENV	この変数は、Ruby on Rails アプリケーションがデプロイされる環境を指定します。たとえば **production**、**development** または **test** などです。ロギングの詳細レベル、エラーページ、Ruby gem インストールなど、レベルごとに動作が異なります。アプリケーションのアセットは、**RAILS_ENV** が **production** に設定されている場合にのみコンパイルされます。デフォルトではこの変数は **${RACK_ENV}** に設定されています。
DISABLE_ASSET_COMPILATION	この変数は **true** に設定されている場合には、アセットのコンパイルプロセスを無効にします。アセットのコンパイルは、アプリケーションが実稼働環境で実行されている場合にのみ行われます。そのため、アセットがコンパイル済みの場合は、この変数を使用できます。
PUMA_MIN_THREADS、**PUMA_MAX_THREADS**	この変数は、Puma のスレッドプールで利用可能な最小および最大スレッド数を指定します。
PUMA_WORKERS	この変数は、Puma の クラスターモード で起動されるワーカープロセスの数を示します (Puma が 3 つ以上のプロセスを実行する場合)。明示的に設定されていない場合には、デフォルトの動作で **PUMA_WORKERS** が、コンテナーで利用可能なメモリーや、ホスト上のコア数に適した値に設定されます。
RUBYGEM_MIRROR	ビルドプロセス時に必要な gem パッケージをダウンロードするための、カスタムの RubyGems ミラー URL を使用するようにこの変数を設定します。注意: この環境変数は、Ruby 2.2+ イメージでのみ利用可能です。

2.7.6. ホットデプロイ

ホットデプロイでは、新しい S2I ビルドを生成する必要なしに、アプリケーションに変更をすばやく加え、デプロイすることができます。このイメージでホットデプロイメントを有効にする方法は、アプリケーションの種類により異なります。

Ruby on Rails アプリケーション

Ruby on Rails アプリケーションの場合は、**RAILS_ENV=development** 環境変数を実行中の Pod に渡して、ビルド済みの Rails アプリケーションを実行します。既存のデプロイメント設定では、**oc set env** コマンドを使用してください。

```
$ oc set env dc/rails-app RAILS_ENV=development
```

他のタイプの Ruby アプリケーション (Sinatra、Padrino など)

他のタイプの Ruby アプリケーションでは、アプリケーションは実行中のコンテナー内でソースコードが変更されるたびに、サーバーを再読み込みできる gem でビルドする必要があります。

- Shotgun
- Rerun
- Rack-livereload

開発モードでアプリケーションを実行できるようにするには、選択した gem で Web サーバーを起動し、ソースコードへの変更の有無を確認するように、S2I **run** スクリプト を変更する必要があります。

カスタマイズした S2I **run** スクリプト でアプリケーションをビルドした後に、**RACK_ENV=development** 環境変数でイメージを実行します。たとえば、**oc new-app** コマンドを確認します。**oc set env** コマンドを使用して、既存のオブジェクトの環境変数を更新することができます。

警告

このオプションは、開発またはデバッグの時にだけ使用するようにしてください。実稼働環境でこの設定をオンにすることは推奨しません。

実行中の Pod でソースコードを変更するには、**oc rsh** コマンドを使用して、コンテナーに入ります。

```
$ oc rsh <pod_id>
```

実行中のコンテナーに入った後に、現在のディレクトリーを、ソースコードが配置されている **/opt/app-root/src** に設定します。

2.8. S2I イメージのカスタマイズ

2.8.1. 概要

S2I ビルダーイメージには通常、**assemble** と **run** スクリプトが含まれていますが、これらのスクリプトのデフォルトの動作は全ユーザーに対して適切であるとは限りません。以下のトピックでは、デフォルトのスクリプトなど、S2I ビルダーの動作をカスタマイズする方法を何点か見ていきます。

2.8.2. イメージに埋め込まれたスクリプトの呼び出し

一般的に、ビルダーイメージでは、最も一般的なユースケースを含む、独自の S2I スクリプトが提供されます。これらのスクリプトで各自のニーズが満たされない場合に向け、S2I には **.s2i/bin** ディレクトリーにカスタムのスクリプトを追加して上書きできる手段があります。ただし、カスタムのスクリプトを追加すると、標準のスクリプトを完全に置き換えてしまいます。これは許容範囲の場合もありますが、シナリオによっては、イメージに含まれるスクリプトのロジックを保持しつつ、スクリプトの前

(または後) にコマンドをいくつか実行する必要がある場合があります。そのような場合には、カスタムのロジックを実行し、イメージ内のデフォルトのスクリプトにさらなる作業を委譲するラッパースクリプトを作成することができます。

ビルダーイメージ内のスクリプトの場所を判断するには、**io.openshift.s2i.scripts-url** ラベルの値を確認します。以下のように **docker inspect** を使用してください。

```
$ docker inspect --format='{{ index .Config.Labels
"io.openshift.s2i.scripts-url" }}' openshift/wildfly-100-centos7
image:///usr/libexec/s2i
```

openshift/wildfly-100-centos7 ビルダーイメージを確認し、対象のスクリプトが **/usr/libexec/s2i** ディレクトリーにあることを確認できます。

この情報を基にして、呼び出しをラップし、独自のスクリプトからこれらのスクリプトを呼び出します。

例2.1 .s2i/bin/assemble スクリプト

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

/usr/libexec/s2i/assemble
rc=$?

if [ $rc -eq 0 ]; then
    echo "After successful assembling"
else
    echo "After failed assembling"
fi

exit $rc
```

以下の例では、メッセージを出力するカスタムの **assemble** スクリプトを表示し、イメージから標準の **assemble** スクリプトを実行して、**assemble** スクリプトの終了コードに応じて別のメッセージを出力します。

run スクリプトをラップする場合には、スクリプトの呼び出しに **exec** を実行して、シグナルが正しく処理されるようにする必要があります。残念ながら、**exec** を使用すると、デフォルトのイメージ実行スクリプトを呼び出した後に追加でコマンドを実行できなくなります。

例2.2 .s2i/bin/run スクリプト

```
#!/bin/bash
echo "Before running application"
exec /usr/libexec/s2i/run
```

第3章 データベースイメージ

3.1. 概要

以下のトピックには、OpenShift Container Platform ユーザーに提供される、さまざまなデータベースイメージに関する情報が含まれます。

注記

現在、テクノロジープレビューとして、データベースイメージ用にクラスター化を有効にできますが、この機能は実稼働環境での使用を目的としていません。

3.2. MYSQL

3.2.1. 概要

OpenShift Container Platform には、MySQL の実行用のコンテナーイメージがあります。このイメージでは、設定で指定されるユーザー名、パスワード、データベース名に基づいてデータベースサービスが提供されます。

3.2.2. バージョン

現時点で、OpenShift Container Platform では、MySQL のバージョン 5.6 および 5.7 を提供しています。

3.2.3. イメージ

イメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/rhscl/mysql-56-rhel7
$ docker pull registry.redhat.io/rhscl/mysql-57-rhel7
```

CentOS 7 ベースのイメージ

MySQL 5.6 および 5.7 の CentOS イメージは、Docker Hub で入手できます。

```
$ docker pull centos/mysql-56-centos7
$ docker pull centos/mysql-57-centos7
```

これらのイメージを使用するには、イメージレジストリーから直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリーにプッシュしてください。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、OpenShift Container Platform リソースがこの ImageStream を参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して ImageStream の 定義例 があります。

3.2.4. 設定および用途

3.2.4.1. データベースの初期化

共有ボリュームを初めて使用する場合には、データベース、データベースの管理ユーザー、MySQL root ユーザー (**MYSQL_ROOT_PASSWORD** 環境変数を指定した場合) が作成され、次に MySQL デーモンが起動します。別のコンテナーにボリュームを再アタッチする場合には、データベース、データベースユーザー、管理者ユーザーは作成されず、MySQL デーモンが開始されます。

以下のコマンドは、新しいデータベースの Pod を作成し、さらにコンテナー内で MySQL を実行します。

```
$ oc new-app \
    -e MYSQL_USER=<username> \
    -e MYSQL_PASSWORD=<password> \
    -e MYSQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/mysql-56-rhel7
```

3.2.4.2. コンテナーでの MySQL コマンドの実行

OpenShift Container Platform は Software Collections (SCL) を使用して、MySQL をインストールし、起動します。(デバッグ用に) 実行中のコンテナー内で MySQL コマンドを実行する場合には bash を使用して呼び出す必要があります。

これを実行するには、まず Pod 名を特定します。たとえば、現在のプロジェクトで Pod の一覧を表示できます。

```
$ oc get pods
```

次に、Pod に対してリモートシェルセッションを開始します。

```
$ oc rsh <pod>
```

コンテナーに入ると、必要な SCL が自動的に有効になります。

Bash シェルから **mysql** コマンドを実行し、MySQL の対話セッションを開始して通常の MySQL 操作が実行できるようになりました。たとえば、データベースユーザーとして認証するには、以下を実行します。

```
bash-4.2$ mysql -u $MYSQL_USER -p$MYSQL_PASSWORD -h $HOSTNAME
$MYSQL_DATABASE
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.6.37 MySQL Community Server (GPL)
...
mysql>
```

完了したら、**quit** または **exit** を入力して MySQL セッションを終了します。

3.2.4.3. 環境変数

MySQL ユーザー名、パスワード、データベース名は、以下の環境変数で設定する必要があります。

表3.1 MySQL 環境変数

変数名	説明
MYSQL_USER	アプリケーションで使用するために作成されたデータベースユーザーのユーザー名を指定します。
MYSQL_PASSWORD	**MYSQL_USER** のパスワード
MYSQL_DATABASE	**MYSQL_USER** が完全な権限を持つデータベースの名前
MYSQL_ROOT_PASSWORD	root ユーザーの任意のパスワード。これが設定されていない場合には、root アカウントにリモートログインできません。コンテナーからはいつでも、パスワードなしにローカル接続が可能です。
MYSQL_SERVICE_HOST	Kubernetes が自動作成したサービスホストの変数
MYSQL_SERVICE_PORT	Kubernetes が自動作成したサービスポートの変数

警告

ユーザー名、パスワード、データベース名を指定する必要があります。この 3 つすべてを指定しない場合には、Pod は起動に失敗し、OpenShift Container Platform は Pod の再起動を継続的に試行します。

MySQL 設定は、以下の環境変数で設定できます。

表3.2 MySQL の他の設定

変数名	説明	デフォルト
MYSQL_LOWER_CASE_TABLE_NAMES	テーブル名の保存および比較方法を設定します。	0
MYSQL_MAX_CONNECTIONS	クライアントが同時に接続可能な最大数	151
MYSQL_MAX_ALLOWED_PACKET	生成された文字列/中間文字列または 1 つのパケットの最大サイズ	200M

変数名	説明	デフォルト
MYSQL_FT_MIN_WORD_LEN	FULLTEXT インデックスに含める文字の最小長	4
MYSQL_FT_MAX_WORD_LEN	FULLTEXT インデックスに含める文字の最大長	20
MYSQL_AIO	ネイティブの AIO が壊れている場合に **innodb_use_native_aio** の設定値を制御します。	1
MYSQL_TABLE_OPEN_CACHE	全スレッド用に開くテーブル数	400
MYSQL_KEY_BUFFER_SIZE	インデックスブロックに使用するバッファーサイズ	32M (または利用可能なメモリーの 10%)
MYSQL_SORT_BUFFER_SIZE	分類に使用するバッファーサイズ	256K
MYSQL_READ_BUFFER_SIZE	シーケンススキャンに使用するバッファーサイズ	8M (または利用可能メモリーの 5%)
MYSQL_INNODB_BUFFER_POOL_SIZE	InnoDB がテーブルやインデックスデータをキャッシュするバッファープールのサイズ	32M (または利用可能なメモリーの 50%)
MYSQL_INNODB_LOG_FILE_SIZE	ロググループにある各ログファイルのサイズ	8M (または利用可能メモリーの 15%)

変数名	説明	デフォルト
MYSQL_INNODB_LOG_BUFFER_SIZE	InnoDB がディスクのログファイルへの書き込みに使用するバッファーサイズ	8M (または利用可能メモリーの 15%)

メモリー関連のパラメーターによっては、デフォルト値が 2 つあるものもあります。コンテナーにメモリーの制限が割り当てられていない場合には、固定値が使用されます。他の値は、コンテナーの起動中に利用可能なメモリーを基に動的に計算されます。

3.2.4.4. ボリュームのマウントポイント

MySQL イメージは、マウントしたボリュームで実行して、データベース用に永続ストレージを有効化できます。

- **/var/lib/mysql/data**: これは、MySQL がデータベースのファイルを保存するデータディレクトリーです。

3.2.4.5. パスワードの変更

パスワードはイメージ設定の一部であるため、データベースユーザー (**MYSQL_USER**) と **root** ユーザーのパスワードを変更する唯一のサポートされている方法として、環境変数 **MYSQL_PASSWORD** と **MYSQL_ROOT_PASSWORD** をそれぞれ変更することができます。

現在のパスワードは、Pod またはデプロイメント設定を Web コンソールで表示するか、CLI で環境変数を表示して、確認できます。

```
$ oc set env pod <pod_name> --list
```

MYSQL_ROOT_PASSWORD が設定されている場合は常に、**root** ユーザーに特定のパスワードを指定してリモートアクセスを有効にできます。また、設定されていない場合には、**root** ユーザーのリモートアクセスが無効になります。これは、一般ユーザー **MYSQL_USER** には影響なく、常にリモートからアクセスできます。また、**root** ユーザーのローカルアクセスにも影響なく、**localhost** でパスワードなしにいつでもログインできます。

SQL ステートメントや、前述した環境変数以外の方法でデータベースのパスワードを変更すると、変数に保存されている値と、実際のパスワードが一致しなくなる可能性があります。データベースコンテナーが起動するたびに、パスワードは環境変数に保存されている値にリセットされます。

これらのパスワードを変更するには、**oc set env** コマンドを使用して、関連するデプロイメント設定の任意の環境変数 1 つまたは両方を更新します。たとえば、テンプレートからアプリケーションを作成する場合など、複数のデプロイメント設定がこれらの環境変数を使用する場合には、デプロイメント設定ごとに変数を更新し、パスワードがどこでも同期されるようにします。これは、すべて同じコマンドで実行できます。

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  MYSQL_PASSWORD=<new_password> \
  MYSQL_ROOT_PASSWORD=<new_root_password>
```

重要

アプリケーションによっては、アプリケーションの他の場所にあるパスワードの他の環境変数を更新して一致させる必要があるものもあります。たとえば、フロントエンド Pod のより一般的な **DATABASE_USER** 変数などは、データベースユーザーのパスワードと一致する必要がある場合があります。必要とされる環境変数すべてにおいて、パスワードがアプリケーションごとに一致しているようにしてください。一致しない場合には、トリガーされた時点で、Pod の再デプロイメントが失敗する場合があります。

設定変更トリガーが設定されている場合には、環境変数を更新すると、データベースサーバーの再デプロイメントがトリガーされます。設定されていない場合には、新しいデプロイメントを手動で起動して、パスワードの変更を適用する必要があります。

新規パスワードが有効になっていることを確認するには、まず実行中の MySQL Pod に対してリモートシェルセッションを開きます。

```
$ oc rsh <pod>
```

Bash シェルから、データベースユーザーの新規パスワードを確認します。

```
bash-4.2$ mysql -u $MYSQL_USER -p<new_password> -h $HOSTNAME
$MYSQL_DATABASE -te "SELECT * FROM (SELECT database()) db CROSS JOIN
(SELECT user()) u"
```

パスワードが正しく変更された場合には、以下のような表が表示されるはずです。

```
+------------+---------------------+
| database() | user()              |
+------------+---------------------+
| sampledb   | user0PG@172.17.42.1 |
+------------+---------------------+
```

root ユーザーの新規パスワードを確認するには、以下を実行します。

```
bash-4.2$ mysql -u root -p<new_root_password> -h $HOSTNAME $MYSQL_DATABASE
-te "SELECT * FROM (SELECT database()) db CROSS JOIN (SELECT user()) u"
```

パスワードが正しく変更された場合には、以下のような表が表示されるはずです。

```
+------------+------------------+
| database() | user()           |
+------------+------------------+
| sampledb   | root@172.17.42.1 |
+------------+------------------+
```

3.2.5. テンプレートからのデータベースサービスの作成

OpenShift Container Platform には テンプレート が含まれており、新規データベースサービスの作成を簡素化します。テンプレートには、必須の環境変数をすべて定義するパラメーターフィールドがあり (ユーザー、パスワード、データベース名など)、自動生成されたパスワード値など、事前定義済みのデフォルト値が設定されます。また、テンプレートは デプロイメント設定 および サービス の両方を定義します。

MySQL テンプレートは、クラスターの初期設定時にクラスター管理者により、デフォルトの **openshift** プロジェクトに登録しておく必要があります。詳細については、必要に応じて、「デフォルトのイメージストリームおよびテンプレートの読み込み」を参照してください。

利用可能なテンプレートは以下の 2 種類です。

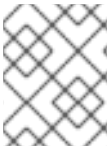
- **mysql-ephemeral** は、データベースのコンテンツ用に一時ストレージを使用するので、開発またはテスト目的にのみ使用します。つまり、Pod が別の Pod に移動されたり、デプロイメント設定が更新され、再デプロイがトリガーされたりなど、データベース Pod が何らかの理由で再起動された場合には、データはすべて失われます。

- **mysql-persistent** は、データベースのデータ用に永続ボリュームストアを使用するので、データは Pod が再起動されても残ります。永続ボリュームを使用する場合には、OpenShift Container Platform デプロイメントで定義された永続ボリュームプールが必要です。プールの設定に関するクラスター管理者向けの説明は、こちらを参照してください。

この説明に従い、テンプレートをインスタンス化できます。

サービスをインスタンス化したら、データベースにアクセスする予定のある別のコンポーネントのデプロイメント設定に、ユーザー名、パスワード、データベース名の環境変数をコピーできます。このコンポーネントは、定義したサービスを使用してこのデータベースにアクセスできます。

3.2.6. MySQL のレプリケーションの使用



注記

現在、テクノロジープレビューとして、データベースイメージ用にクラスター化を有効にできますが、この機能は実稼働環境での使用を目的としていません。

Red Hat は、MySQL のマスター、スレーブのレプリケーション (クラスタリング) 用に概念実証テンプレートを提供します。GitHub からサンプルテンプレート を入手できます。

現在のプロジェクトのテンプレートライブラリーにテンプレートのサンプルをアップロードするには、以下を実行します。

```
$ oc create -f \
    https://raw.githubusercontent.com/sclorg/mysql-
container/master/examples/replica/mysql_replica.json
```

以下のセクションでは、サンプルのテンプレートに定義されているオブジェクト、およびそれらのオブジェクトが連携してマスターとスレーブのレプリケーションを実装する MySQL サーバークラスターをどのように起動するのかを詳しく説明します。これは、MySQL 向けに推奨されるレプリケーションストラテジーです。

3.2.6.1. MySQL マスターのデプロイメント設定の作成

MySQL レプリケーションを設定するには、デプロイメント設定 を、レプリケーションコントローラー を定義するテンプレート例に定義します。MySQL のマスターとスレーブレプリケーションには、デプロイメント設定が 2 つ必要です。1 つ目のデプロイメント設定では、MySQL **マスター** サーバーを、2 つ目で MySQL **スレーブ** サーバーを定義します。

MySQL サーバーに対してマスターとして機能するように指示するには、デプロイメント設定のコンテナー定義にある **command** フィールドに、**run-mysqld-master** を設定する必要があります。このスクリプトは、MySQL イメージの別のエントリーポイントとして機能し、MySQL サーバーがレプリケーションのマスターとして実行するように設定します。

MySQL レプリケーションでは、マスターとスレーブ間のデータをリレーする特別ユーザーが必要です。この目的で使用できるように、以下の環境変数をテンプレートに定義します。

変数名	説明	デフォルト
MYSQL_MASTER_USER	レプリケーションユーザーのユーザー名	**master**
MYSQL_MASTER_PASSWORD	レプリケーションユーザーのパスワード	**generated**

例3.1 サンプルテンプレートでの MySQL マスターデプロイメント設定のオブジェクト定義

```
kind: "DeploymentConfig"
apiVersion: "v1"
metadata:
  name: "mysql-master"
spec:
  strategy:
    type: "Recreate"
  triggers:
    - type: "ConfigChange"
  replicas: 1
  selector:
    name: "mysql-master"
  template:
    metadata:
      labels:
        name: "mysql-master"
    spec:
      volumes:
        - name: "mysql-master-data"
          persistentVolumeClaim:
            claimName: "mysql-master"
      containers:
        - name: "server"
          image: "openshift/mysql-56-centos7"
          command:
            - "run-mysqld-master"
          ports:
            - containerPort: 3306
              protocol: "TCP"
          env:
            - name: "MYSQL_MASTER_USER"
              value: "${MYSQL_MASTER_USER}"
            - name: "MYSQL_MASTER_PASSWORD"
              value: "${MYSQL_MASTER_PASSWORD}"
            - name: "MYSQL_USER"
              value: "${MYSQL_USER}"
            - name: "MYSQL_PASSWORD"
              value: "${MYSQL_PASSWORD}"
            - name: "MYSQL_DATABASE"
              value: "${MYSQL_DATABASE}"
```

```
            - name: "MYSQL_ROOT_PASSWORD"
              value: "${MYSQL_ROOT_PASSWORD}"
          volumeMounts:
            - name: "mysql-master-data"
              mountPath: "/var/lib/mysql/data"
          resources: {}
          terminationMessagePath: "/dev/termination-log"
          imagePullPolicy: "IfNotPresent"
          securityContext:
            capabilities: {}
            privileged: false
      restartPolicy: "Always"
      dnsPolicy: "ClusterFirst"
```

デプロイメント設定で永続ボリュームを要求し、MySQL マスターサーバー用に全データを永続化したため、ストレージを要求できる永続ボリュームを作成するように、クラスター管理者に依頼する必要があります。

デプロイメント設定を作成し、MySQL マスターサーバーが指定された Pod を起動した後には、**MYSQL_DATABASE** で定義されたデータベースが作成され、このデータベースをスレーブに複製するようにサーバーが設定されます。

提供されているサンプルでは、MySQL マスターサーバーのレプリカ 1 つのみが定義されているため、OpenShift Container Platform はサーバーの 1 つのインスタンスのみを起動します。複数のインスタンス (マルチマスター) はサポートされていないため、このレプリケーションコントローラーはスケーリングできません。

テンプレートにデプロイメント設定を定義して、MySQL マスターで作成したデータベースを複製します。このデプロイメント設定は、**command** フィールドが **run-mysqld-slave** に設定されている、MySQL イメージを起動するレプリケーションコントローラーを作成します。このもう 1 つのエントリーポイントでは、データベースの初期化をスキップし、MySQL サーバーが **mysql-master** サービスに接続するように設定します。これについても、サンプルのテンプレートに定義されています。

例3.2 サンプルテンプレートでの MySQL スレーブデプロイメント設定のオブジェクト定義

```
kind: "DeploymentConfig"
apiVersion: "v1"
metadata:
  name: "mysql-slave"
spec:
  strategy:
    type: "Recreate"
  triggers:
    - type: "ConfigChange"
  replicas: 1
  selector:
    name: "mysql-slave"
  template:
    metadata:
      labels:
        name: "mysql-slave"
    spec:
      containers:
        - name: "server"
```

```
          image: "openshift/mysql-56-centos7"
          command:
            - "run-mysqld-slave"
          ports:
            - containerPort: 3306
              protocol: "TCP"
          env:
            - name: "MYSQL_MASTER_USER"
              value: "${MYSQL_MASTER_USER}"
            - name: "MYSQL_MASTER_PASSWORD"
              value: "${MYSQL_MASTER_PASSWORD}"
            - name: "MYSQL_DATABASE"
              value: "${MYSQL_DATABASE}"
          resources: {}
          terminationMessagePath: "/dev/termination-log"
          imagePullPolicy: "IfNotPresent"
          securityContext:
            capabilities: {}
            privileged: false
        restartPolicy: "Always"
        dnsPolicy: "ClusterFirst"
```

このデプロイメント設定のサンプルでは、最初のレプリカ数を **1** に設定して、レプリケーションコントローラーを開始します。アカウントのリソース容量に達するまで、両方向にこのレプリケーションコントローラーをスケーリングできます。

3.2.6.2. ヘッドレスサービスの作成

MySQL スレーブのレプリケーションコントローラーで作成した Pod は、レプリケーションを登録するために、MySQL マスターサーバーに到達する必要があります。この目的のために、サンプルテンプレートでは、**mysql-master** と呼ばれるヘッドレスサービスを定義します。このサービスは、レプリケーションだけに使用するのではなく、クライアントは MySQL ホストとして **mysql-master:3306** にクエリーも送信します。

ヘッドレスサービスを含めるには、サービス定義の **portalIP** パラメーターを **None** に設定します。このように設定すると、DNS クエリーを使用して、このサービスの現在のエンドポイントを表す Pod の IP アドレス一覧を取得できるようになります。

例3.3 サンプルテンプレートでのヘッドレスサービスのオブジェクト定義

```
kind: "Service"
apiVersion: "v1"
metadata:
  name: "mysql-master"
  labels:
    name: "mysql-master"
spec:
  ports:
    - protocol: "TCP"
      port: 3306
      targetPort: 3306
      nodePort: 0
  selector:
    name: "mysql-master"
```

```
    portalIP: "None"
    type: "ClusterIP"
    sessionAffinity: "None"
  status:
    loadBalancer: {}
```

3.2.6.3. MySQL スレーブのスケーリング

クラスターのメンバー数を増やすには、以下を実行します。

```
$ oc scale rc mysql-slave-1 --replicas=<number>
```

このコマンドは、レプリケーションコントローラーに対して、新しい MySQL スレーブ Pod を作成するように指示します。新しいスレーブが作成されると、スレーブのエントリーポイントが最初に **mysql-master** サービスに問い合わせして、レプリケーションセットに登録しようとします。これが完了すると、MySQL マスターサーバーはスレーブに複製されたデータベースを送信します。

スケールダウン時には、MySQL スレーブがシャットダウンされ、スレーブに永続ストレージが定義されていないので、スレーブ上の全データが失われます。MySQL マスターサーバーは、スレーブに到達できないことを検出し、自動的にレプリケーションからそのスレーブを取り除きます。

3.2.7. トラブルシューティング

以下のセクションでは、発生する可能性のある問題と、考えられる解決策を説明します。

3.2.7.1. Linux ネイティブの AIO の障害

現象

MySQL コンテナーが起動に失敗し、以下のようなログを出力します。

```
151113  5:06:56 InnoDB: Using Linux native AIO
151113  5:06:56  InnoDB: Warning: io_setup() failed with EAGAIN. Will make
5 attempts before giving up.
InnoDB: Warning: io_setup() attempt 1 failed.
InnoDB: Warning: io_setup() attempt 2 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 3 failed.
InnoDB: Warning: io_setup() attempt 4 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 5 failed.
151113  5:06:59  InnoDB: Error: io_setup() failed with EAGAIN after 5
attempts.
InnoDB: You can disable Linux Native AIO by setting innodb_use_native_aio
= 0 in my.cnf
151113  5:06:59 InnoDB: Fatal error: cannot initialize AIO sub-system
151113  5:06:59 [ERROR] Plugin 'InnoDB' init function returned error.
151113  5:06:59 [ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE
failed.
151113  5:06:59 [ERROR] Unknown/unsupported storage engine: InnoDB
151113  5:06:59 [ERROR] Aborting
```

説明

MySQL のストレージエンジンは、リソース制限が原因で、カーネルの AIO (非同期 I/O) 機能を使用できませんでした。

解決策

環境変数 **MYSQL_AIO** の値を **0** に設定して、AIO の使用を完全に停止します。今後のデプロイメントでは、この設定により MySQL の設定変数 **innodb_use_native_aio** の値が **0** に指定されます。

または **aio-max-nr** カーネルリソースを増やします。以下の例では、現在の **aio-max-nr** の値を検証して、この値を 2 倍にします。

```
$ sysctl fs.aio-max-nr
fs.aio-max-nr = 1048576
# sysctl -w fs.aio-max-nr=2097152
```

これはノードごとの解決策であるため、次にノードが再起動されるまで有効です。

3.3. POSTGRESQL

3.3.1. 概要

OpenShift Container Platform には、PostgreSQL の実行用のコンテナーイメージがあります。このイメージでは、設定で指定されるユーザー名、パスワード、データベース名を基にデータベースサービスが提供されます。

3.3.2. バージョン

現時点で、OpenShift Container Platform は PostgreSQL バージョン 9.4 および 9.5 をサポートします。

3.3.3. イメージ

これらのイメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/rhscl/postgresql-94-rhel7
$ docker pull registry.redhat.io/rhscl/postgresql-95-rhel7
```

CentOS 7 ベースのイメージ

これらのイメージは、Docker Hub で入手できます。

```
$ docker pull centos/postgresql-94-centos7
$ docker pull centos/postgresql-95-centos7
```

これらのイメージを使用するには、イメージレジストリーから直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリーにプッシュしてください。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、OpenShift Container Platform リソースがこの ImageStream を参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して ImageStream の 定義例 があります。

3.3.4. 設定および用途

3.3.4.1. データベースの初期化

共有ボリュームを初めて使用する場合には、データベース、データベースの管理ユーザー、PostgreSQL root ユーザー (**POSTGRESQL_ADMIN_PASSWORD** 環境変数を指定した場合) が作成され、次に PostgreSQL デーモンが起動します。別のコンテナーにボリュームを再アタッチする場合には、データベース、データベースユーザー、管理者ユーザーは作成されず、PostgreSQL デーモンが起動します。

以下のコマンドは、新しいデータベースの Pod を作成し、さらにコンテナー内で PostgreSQL を実行します。

```
$ oc new-app \
    -e POSTGRESQL_USER=<username> \
    -e POSTGRESQL_PASSWORD=<password> \
    -e POSTGRESQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/postgresql-95-rhel7
```

3.3.4.2. コンテナーでの PostgreSQL コマンドの実行

OpenShift Container Platform は Software Collections (SCL) を使用して、PostgreSQL をインストール、起動します。(デバッグ用に) 実行中のコンテナー内で PostgreSQL コマンドを実行する場合には bash を使用して呼び出す必要があります。

これには、まず、実行中の PostgreSQL Pod の名前を特定します。たとえば、現在のプロジェクトで Pod の一覧を表示できます。

```
$ oc get pods
```

次に、任意の Pod に対してリモートシェルセッションを開きます。

```
$ oc rsh <pod>
```

コンテナーに入ると、必要な SCL が自動的に有効になります。

Bash シェルから **psql** コマンドを実行し、PostgreSQL の対話セッションを開始して通常の PostgreSQL 操作が実行できるようになりました。たとえば、データベースユーザーとして認証するには、以下を実行します。

```
bash-4.2$ PGPASSWORD=$POSTGRESQL_PASSWORD psql -h postgresql
$POSTGRESQL_DATABASE $POSTGRESQL_USER
psql (9.5.16)
Type "help" for help.

default=>
```

完了したら、**\q** と入力して PostgreSQL セッションを終了します。

3.3.4.3. 環境変数

PostgreSQL ユーザー名、パスワード、データベース名は、以下の環境変数で設定する必要があります。

表3.3 PostgreSQL 環境変数

変数名	説明
POSTGRESQL_USER	作成予定の PostgreSQL アカウントのユーザー名。このユーザーには、対象のデータベースに対する完全な権限があります。
POSTGRESQL_PASSWORD	ユーザーアカウントのパスワード
POSTGRESQL_DATABASE	データベース名
POSTGRESQL_ADMIN_PASSWORD	**postgres** 管理ユーザーの任意パスワード。これが設定されていない場合には、**postgres** アカウントにリモートからログインができません。コンテナーからはいつでも、パスワードなしにローカル接続が可能です。

警告

ユーザー名、パスワード、データベース名を指定する必要があります。この 3 つすべてを指定しない場合には、Pod は起動に失敗し、OpenShift Container Platform は Pod の再起動を継続的に試行します。

PostgreSQL 設定は、以下の環境変数で設定できます。

表3.4 PostgreSQL の他の設定

変数名	説明	デフォルト
POSTGRESQL_MAX_CONNECTIONS	許容範囲の最大クライアント接続数	100
POSTGRESQL_MAX_PREPARED_TRANSACTIONS	「準備」状態にすることのできる最大トランザクション数。準備状態のトランザクションを使用する場合には、値は **POSTGRESQL_MAX_CONNECTIONS** 以上に指定する必要があります。	0
POSTGRESQL_SHARED_BUFFERS	データのキャッシュ用に PostgreSQL 専用に割り当てられたメモリー量	32M

変数名	説明	デフォルト
POSTGRESQL_EFFECTIVE_CACHE_SIZE	オペレーティングシステム別または PostgreSQL 自体で、ディスクキャッシュに利用可能な予想メモリー量	128M

3.3.4.4. ボリュームのマウントポイント

PostgreSQL イメージは、マウントしたボリュームで実行して、データベース用に永続ストレージを有効化できます。

- **/var/lib/pgsql/data**: これは、PostgreSQL がデータベースファイルを保存するデータベースクラスターのディレクトリーです。

3.3.4.5. パスワードの変更

パスワードはイメージ設定の一部であるため、データベースユーザー (**POSTGRESQL_USER**) と **postgres** 管理者ユーザーのパスワードを変更する唯一のサポートされている方法として、環境変数 **POSTGRESQL_PASSWORD** と **POSTGRESQL_ADMIN_PASSWORD** をそれぞれ変更することができます。

現在のパスワードは、Pod またはデプロイメント設定を Web コンソールで表示するか、CLI で環境変数を表示して、確認できます。

```
$ oc set env pod <pod_name> --list
```

SQL ステートメントや、前述した環境変数以外の方法でデータベースのパスワードを変更すると、変数に保存されている値と、実際のパスワードが一致しなくなる可能性があります。データベースコンテナーが起動するたびに、パスワードは環境変数に保存されている値にリセットされます。

これらのパスワードを変更するには、**oc set env** コマンドを使用して、関連するデプロイメント設定の任意の環境変数 1 つまたは両方を更新します。たとえば、テンプレートからアプリケーションを作成する場合など、複数のデプロイメント設定がこれらの環境変数を使用する場合には、デプロイメント設定ごとに変数を更新し、パスワードがどこでも同期されるようにします。これは、すべて同じコマンドで実行できます。

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  POSTGRESQL_PASSWORD=<new_password> \
  POSTGRESQL_ADMIN_PASSWORD=<new_admin_password>
```

重要

アプリケーションによっては、アプリケーションの他の場所にあるパスワードの他の環境変数を更新して一致させる必要があるものもあります。たとえば、フロントエンド Pod のより一般的な **DATABASE_USER** 変数などは、データベースユーザーのパスワードと一致する必要がある場合があります。必要とされる環境変数すべてにおいて、パスワードがアプリケーションごとに一致しているようにしてください。一致しない場合には、トリガーされた時点で、Pod の再デプロイメントが失敗する場合があります。

設定変更トリガーが設定されている場合には、環境変数を更新すると、データベースサーバーの再デプロイメントがトリガーされます。設定されていない場合には、新しいデプロイメントを手動で起動して、パスワードの変更を適用する必要があります。

新規パスワードが有効になっていることを確認するには、まず、実行中の PostgreSQL Pod に対してリモートシェルセッションを開きます。

```
$ oc rsh <pod>
```

Bash シェルから、データベースユーザーの新規パスワードを確認します。

```
bash-4.2$ PGPASSWORD=<new_password> psql -h postgresql
$POSTGRESQL_DATABASE $POSTGRESQL_USER -c "SELECT * FROM (SELECT
current_database()) cdb CROSS JOIN (SELECT current_user) cu"
```

パスワードが正しく変更された場合には、以下のような表が表示されるはずです。

```
 current_database | current_user
------------------+--------------
 default          | django
(1 row)
```

Bash シェルから **postgres** 管理者ユーザーの新規パスワードを検証します。

```
bash-4.2$ PGPASSWORD=<new_admin_password> psql -h postgresql
$POSTGRESQL_DATABASE postgres -c "SELECT * FROM (SELECT
current_database()) cdb CROSS JOIN (SELECT current_user) cu"
```

パスワードが正しく変更された場合には、以下のような表が表示されるはずです。

```
 current_database | current_user
------------------+--------------
 default          | postgres
(1 row)
```

3.3.5. テンプレートからのデータベースサービスの作成

OpenShift Container Platform には テンプレート が含まれており、新規データベースサービスの作成を簡素化します。テンプレートには、必須の環境変数をすべて定義するパラメーターフィールドがあり (ユーザー、パスワード、データベース名など)、自動生成されたパスワード値など、事前定義済みのデフォルト値が設定されます。また、テンプレートは デプロイメント設定 および サービス の両方を定義します。

PostgreSQL テンプレートは、クラスターの初期設定時にクラスター管理者により、デフォルトの **openshift** プロジェクトに登録しておく必要があります。詳細については、必要に応じて、「デフォルトのイメージストリームおよびテンプレートの読み込み」を参照してください。

利用可能なテンプレートは以下の 2 種類です。

- **PostgreSQL-ephemeral** は、データベースのコンテンツ用に一時ストレージを使用するので、開発またはテスト目的にのみ使用します。つまり、Pod が別の Pod に移動されたり、デプロイメント設定が更新され、再デプロイがトリガーされたりなど、データベース Pod が何らかの理由で再起動された場合には、データはすべて失われます。

- **PostgreSQL-persistent** は、データベースのデータ用に永続ボリュームストアを使用するので、データは Pod が再起動されても残ります。永続ボリュームを使用する場合には、OpenShift Container Platform デプロイメントで定義された永続ボリュームプールが必要です。プールの設定に関するクラスター管理者向けの説明は、こちらを参照してください。

この説明に従い、テンプレートをインスタンス化できます。

サービスをインスタンス化したら、データベースにアクセスする予定のある別のコンポーネントのデプロイメント設定に、ユーザー名、パスワード、データベース名の環境変数をコピーできます。このコンポーネントは、定義したサービスを使用してこのデータベースにアクセスできます。

3.4. MONGODB

3.4.1. 概要

OpenShift Container Platform には、MongoDB の実行用のコンテナーイメージがあります。このイメージでは、設定で指定されるユーザー名、パスワード、データベース名に基づくデータベースサービスが提供されます。

3.4.2. バージョン

現時点で、OpenShift Container Platform では、MongoDB のバージョン 2.6、3.2 および 3.4 を提供しています。

3.4.3. イメージ

これらのイメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/rhscl/mongodb-26-rhel7
$ docker pull registry.redhat.io/rhscl/mongodb-32-rhel7
$ docker pull registry.redhat.io/rhscl/mongodb-34-rhel7
```

CentOS 7 ベースのイメージ

これらのイメージは、Docker Hub で入手できます。

```
$ docker pull centos/mongodb-26-centos7
$ docker pull centos/mongodb-32-centos7
$ docker pull centos/mongodb-34-centos7
```

これらのイメージを使用するには、イメージレジストリーから直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリーにプッシュしてください。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、OpenShift Container Platform リソースがこの ImageStream を参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して ImageStream の 定義例 があります。

3.4.4. 設定および用途

3.4.4.1. データベースの初期化

MongoDB は、一時ボリュームまたは永続ボリュームで設定できます。ボリュームを初めて使用する場合には、データベースとデータベースの管理ユーザーが作成され、次に MongoDB デーモンが起動します。別のコンテナーにボリュームを再アタッチする場合には、データベース、データベースユーザー、管理者ユーザーは作成されず、MongoDB デーモンが起動します。

以下のコマンドは、新しいデータベースの Pod を作成し、さらに一時ボリュームが含まれるコンテナー内で MongoDB を実行します。

```
$ oc new-app \
    -e MONGODB_USER=<username> \
    -e MONGODB_PASSWORD=<password> \
    -e MONGODB_DATABASE=<database_name> \
    -e MONGODB_ADMIN_PASSWORD=<admin_password> \
    registry.redhat.io/rhscl/mongodb-26-rhel7
```

3.4.4.2. コンテナーでの MongoDB コマンドの実行

OpenShift Container Platform は Software Collections (SCL) を使用して、MongoDB をインストールし、起動します。(デバッグ用に) 実行中のコンテナー内で MongoDB コマンドを実行する場合には bash を使用して呼び出す必要があります。

これを実行するには、まず 実行中の MongoDB Pod の名前を特定します。たとえば、現在のプロジェクトで Pod の一覧を表示できます。

```
$ oc get pods
```

次に、任意の Pod に対してリモートシェルセッションを開きます。

```
$ oc rsh <pod>
```

コンテナーに入ると、必要な SCL が自動的に有効になります。

Bash シェルから **mongo** コマンドを実行し、MongoDB の対話セッションを開始して通常の MongoDB 操作が実行できるようになりました。たとえば、**sampledb** データベースに切り替えてデータベースユーザーとして認証するには、以下を実行します。

```
bash-4.2$ mongo -u $MONGODB_USER -p $MONGODB_PASSWORD $MONGODB_DATABASE
MongoDB shell version: 2.6.9
connecting to: sampledb
>
```

完了したら、**CTRL+D** を押して、MongoDB セッションを終了します。

3.4.4.3. 環境変数

MongoDB ユーザー名、パスワード、データベース名および **admin** のパスワードは、以下の環境変数で設定する必要があります。

表3.5 MongoDB 環境変数

変数名	説明
MONGODB_USER	作成する MongoDB アカウントのユーザー名

変数名	説明
MONGODB_PASSWORD	ユーザーアカウントのパスワード
MONGODB_DATABASE	データベース名
MONGODB_ADMIN_PASSWORD	**admin** ユーザーのパスワード

警告

ユーザー名、パスワード、データベース名および **admin** パスワードを指定する必要があります。この 4 つすべてを指定しない場合には、Pod は起動できず、OpenShift Container Platform は継続して Pod の再起動を試行します。

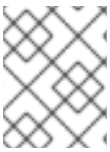
注記

管理者のユーザー名は **admin** に設定されます。admin のパスワードは、**MONGODB_ADMIN_PASSWORD** 環境変数で設定する必要があります。このプロセスは、データベースの初期化の実行時に行います。

MongoDB 設定は、以下の環境変数で設定できます。

表3.6 MongoDB の他の設定

変数名	説明	デフォルト
MONGODB_NOPREALLOC	データファイルの事前割り当てを無効にします。	**true**
MONGODB_SMALLFILES	MongoDB がより小さなデータファイルサイズを使用するようにデフォルト設定します。	**true**
MONGODB_QUIET	MongoDB を Quiet モードで実行して、出力量を制限しようとします。	**true**



注記

テキスト検索は、MongoDB バージョン 2.6 以降ではデフォルトで有効になっているので、設定可能なパラメーターはありません。

3.4.4.4. ボリュームのマウントポイント

MongoDB イメージはマウントしたボリュームで実行して、データベース用に永続ストレージを有効化できます。

- **/var/lib/mongodb/data**: これは、MongoDB がデータベースファイルを保存するデータ

ベースのディレクトリーです。

3.4.4.5. パスワードの変更

パスワードはイメージ設定の一部であるため、データベースユーザー (**MONGODB_USER**) と **admin** ユーザーのパスワードを変更するための唯一のサポートされている方法とし、環境変数 **MONGODB_PASSWORD** と **MONGODB_ADMIN_PASSWORD** をそれぞれ変更することができます。

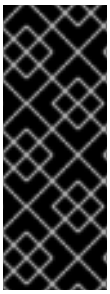
現在のパスワードは、Pod またはデプロイメント設定を Web コンソールで表示するか、CLI で環境変数を表示して、確認できます。

```
$ oc set env pod <pod_name> --list
```

MongoDB で直接データベースのパスワードを変更すると、変数に保存されている値と実際のパスワードが一致しなくなる可能性があります。データベースコンテナーが起動するたびに、パスワードは環境変数に保存されている値にリセットされます。

これらのパスワードを変更するには、**oc set env** コマンドを使用して、関連するデプロイメント設定の任意の環境変数 1 つまたは両方を更新します。たとえば、テンプレートからアプリケーションを作成する場合など、複数のデプロイメント設定がこれらの環境変数を使用する場合には、デプロイメント設定ごとに変数を更新し、パスワードがどこでも同期されるようにします。これは、すべて同じコマンドで実行できます。

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  MONGODB_PASSWORD=<new_password> \
  MONGODB_ADMIN_PASSWORD=<new_admin_password>
```



重要

アプリケーションによっては、アプリケーションの他の場所にあるパスワードの他の環境変数を更新して一致させる必要があるものもあります。たとえば、フロントエンド Pod のより一般的な **DATABASE_USER** 変数などは、データベースユーザーのパスワードと一致する必要がある場合があります。必要とされる環境変数すべてにおいて、パスワードがアプリケーションごとに一致しているようにしてください。一致しない場合には、トリガーされた時点で、Pod の再デプロイメントが失敗する場合があります。

設定変更トリガーが設定されている場合には、環境変数を更新すると、データベースサーバーの再デプロイメントがトリガーされます。設定されていない場合には、新しいデプロイメントを手動で起動して、パスワードの変更を適用する必要があります。

新規パスワードが有効になっていることを確認するには、まず、実行中の MongoDB Pod に対してリモートシェルセッションを開きます。

```
$ oc rsh <pod>
```

Bash シェルから、データベースユーザーの新規パスワードを確認します。

```
bash-4.2$ mongo -u $MONGODB_USER -p <new_password> $MONGODB_DATABASE --
eval "db.version()"
```

パスワードが正しく変更された場合には、以下のような出力が表示されるはずです。

```
MongoDB shell version: 2.6.9
connecting to: sampledb
2.6.9
```

admin ユーザーの新規パスワードを確認するには、以下を実行します。

```
bash-4.2$ mongo -u admin -p <new_admin_password> admin --eval
"db.version()"
```

パスワードが正しく変更された場合には、以下のような出力が表示されるはずです。

```
MongoDB shell version: 2.6.9
connecting to: admin
2.6.9
```

3.4.5. テンプレートからのデータベースサービスの作成

OpenShift Container Platform には テンプレート が含まれており、新規データベースサービスの作成を簡素化します。テンプレートには、必須の環境変数をすべて定義するパラメーターフィールドがあり (ユーザー、パスワード、データベース名など)、自動生成されたパスワード値など、事前定義済みのデフォルト値が設定されます。また、テンプレートは デプロイメント設定 および サービス の両方を定義します。

MongoDB テンプレートは、クラスターの初期設定時にクラスター管理者により、デフォルトの **openshift** プロジェクトに登録しておく必要があります。詳細については、必要に応じて、「デフォルトのイメージストリームおよびテンプレートの読み込み」を参照してください。

利用可能なテンプレートは以下の 2 種類です。

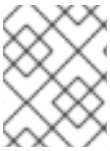
- **mongodb-ephemeral** は、データベースのコンテンツ用に一時ストレージを使用するので、開発またはテスト目的にのみ使用します。つまり、Pod が別の Pod に移動されたり、デプロイメント設定が更新され、再デプロイがトリガーされたりなど、データベース Pod が何らかの理由で再起動された場合には、データはすべて失われます。

- **mongodb-persistent** は、データベースのデータ用に永続ボリュームストアを使用するので、データは Pod が再起動されても残ります。永続ボリュームを使用する場合には、OpenShift Container Platform デプロイメントで定義された永続ボリュームプールが必要です。プールの設定に関するクラスター管理者向けの説明は、こちらを参照してください。

この説明に従い、テンプレートをインスタンス化できます。

サービスをインスタンス化したら、データベースにアクセスする予定のある別のコンポーネントのデプロイメント設定に、ユーザー名、パスワード、データベース名の環境変数をコピーできます。このコンポーネントは、定義したサービスを使用してこのデータベースにアクセスできます。

3.4.6. MongoDB レプリケーション



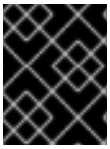
注記

現在、テクノロジープレビューとして、データベースイメージ用にクラスター化を有効にできますが、この機能は実稼働環境での使用を目的としていません。

Red Hat は、StatefulSet を使用した MongoDB のレプリケーション (クラスタリング) 用に、概念実証 テンプレートを提供します。GitHub からサンプルテンプレート を入手できます。

たとえば、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のサンプルをアップロードするには、以下を実行します。

```
$ oc create -f \
    https://raw.githubusercontent.com/sclorg/mongodb-
container/master/examples/petset/mongodb-petset-persistent.yaml
```



重要

以下のテンプレートサンプルでは、永続ストレージを使用します。このテンプレートを使用するには、クラスターで永続ボリュームが利用できるようにする必要があります。

OpenShift Container Platform は正常でない Pod (コンテナー) を自動的に再起動するので、レプリカセットのメンバーの 1 つまたは複数で、クラッシュまたは障害が発生すると、レプリカセットメンバーは再起動されます。

レプリカセットのメンバーがダウンまたは再起動している場合に考えられるシナリオは以下のいずれかです。

1. プライマリーメンバーがダウンしている:
   このような場合には、他の 2 つのメンバーが新しいプライマリーを選択します。新しいプライマリーが選択されるまで、読み取りには影響はありませんが、書き込みが失敗してしまいます。正常に選択された後には、書き込みおよび読み取りは通常通りに処理されます。

2. セカンダリーメンバーの 1 つがダウンしている:
   読み取りおよび書き込みには影響はありません。**oplogSize** 設定と書き込み速度によって、3 番目のメンバーがレプリカセットへの参加に失敗する可能性があるため、手動の介入によりデータベースのコピーをもう一度同期する必要があります。

3. 2 つのメンバーがダウンしている:
   3 つのメンバーで構成されるレプリカセットメンバーが他のメンバーに到達できない場合には、プライマリーロールが指定されていれば、そのロールが取り消されます。このような場合には、読み取りはセカンダリーメンバーが行い、書き込みに失敗します。他のメンバーが 1 つでも起動したらすぐに、新しいプライマリーメンバーが選択され、読み取りおよび書き込みが通常通りに処理されます。

4. 全メンバーがダウンしている:
   このように極端な場合は、読み取りおよび書き込み両方に失敗します。2 つ以上のメンバーが起動してくると、レプリカセットメンバーにプライマリーとセカンダリーメンバーが含まれるように選択が行われ、その後に読み取りと書き込みが通常通りに処理されます。

これが MongoDB の推奨のレプリケーションストラテジーです。

注記

実稼働環境の場合には、できるだけメンバー間の分離を確保する必要があります。StatefulSet Pod を異なるノードにスケジューリングするノード選択機能を 1 つまたは複数使用し、個別のボリュームでサポートされるストレージを提供することを推奨します。

3.4.6.1. 制限

- MongoDB 3.2 のみがサポートされます。
- スケールダウンする場合には、レプリカセットの設定は手動で更新する必要があります。
- ユーザーおよび管理者のパスワードの変更は手動のプロセスで行います。以下を実行する必要があります。
  - StatefulSet 設定の環境変数の値を更新する
  - データベースのパスワードを変更する
  - 順次 Pod をすべて再起動する

3.4.6.2. サンプルテンプレートの使用

事前作成されている永続ボリューム 3 つあり、永続ボリュームのプロビジョニングが設定されていることを前提とします。

1. MongoDB クラスターを作成する新規プロジェクトを作成します。

```
$ oc new-project mongodb-cluster-example
```

2. サンプルテンプレートを使用して新規アプリケーションを作成します。

```
$ oc new-app https://raw.githubusercontent.com/sclorg/mongodb-
container/master/examples/petset/mongodb-petset-persistent.yaml
```

このコマンドでは、3 つのレプリカセットメンバーを含む MongoDB クラスターが作成されました。

3. 新規の MongoDB Pod のステータスを確認します。

```
$ oc get pods
NAME        READY     STATUS    RESTARTS   AGE
mongodb-0   1/1       Running   0          50s
mongodb-1   1/1       Running   0          50s
mongodb-2   1/1       Running   0          49s
```

サンプルのテンプレートからクラスターを作成すると、3 つのメンバーを含むレプリカセットが作成されます。Pod が実行されると、以下のようにこれらの Pod でさまざまなアクションを実行できます。

- Pod の 1 つのログを確認します。

```
$ oc logs mongodb-0
```

- Pod にログインします。

```
$ oc rsh mongodb-0
sh-4.2$
```

- MongoDB インスタンスにログインします。

```
sh-4.2$ mongo $MONGODB_DATABASE -u $MONGODB_USER -p$MONGODB_PASSWORD
MongoDB shell version: 3.2.6
connecting to: sampledb
```

```
rs0:PRIMARY>
```

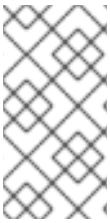
3.4.6.3. スケールアップ

MongoDB は、レプリカセット内に奇数の数のメンバーを指定することを推奨します。永続ボリュームが十分に存在し、動的ストレージプロビジョナーがある場合には、**oc scale** を使用してスケールアップを行います。

```
$ oc scale --replicas=5 statefulsets/mongodb

$ oc get pods
NAME        READY     STATUS    RESTARTS   AGE
mongodb-0   1/1       Running   0          9m
mongodb-1   1/1       Running   0          8m
mongodb-2   1/1       Running   0          8m
mongodb-3   1/1       Running   0          1m
mongodb-4   1/1       Running   0          57s
```

これにより、レプリカセットと接続する新規 Pod が作成され、設定が更新されます。



注記

oplogSize 設定よりもデータベースのサイズが大きい場合には、既存のデータベースは手動でスケールアップする必要があります。このような場合には、新規メンバーの初回同期を手動で行う必要があります。詳しい情報は、「Check the Size of the Oplog」および「MongoDB Replication」ドキュメントを参照してください。

3.4.6.4. スケールダウン

レプリカセットをスケールダウンするには、メンバー数を 5 つから 3 つ、または 3 つから 1 つのみに変更することができます。

前提条件 (ストレージの空き容量、既存のデータベースのサイズ、**oplogSize**) を満たす場合には、手動での介入なしにスケールアップができますが、スケールダウンは常に手動での介入が必要です。

スケールダウンを実行するには、以下を実行します。

1. **oc scale** コマンドを使用して、新しいレプリカ数を設定します。

   ```
   $ oc scale --replicas=3 statefulsets/mongodb
   ```

   新しいレプリカ数が以前の数の過半数を占める場合には、削除された Pod の 1 つに、プライマリーメンバーロールを指定されていた時のために、レプリカセットにより新しいプライマリーが選択される場合があります。たとえば、メンバーを 5 から 3 にスケールダウンする場合などです。

   また、少ない数にスケールダウンすると一時的に、レプリカセットに含まれるのがセカンダリーメンバーだけで、読み取り専用モードとなることがあります。たとえば、メンバーを 5 から 1 にスケールダウンする場合などです。

2. 存在しなくなったメンバーを削除するように、レプリカセットの設定を更新します。
   これは、レプリカ数の検査 (downward API 経由で公開) や StatefulSet から削除された Pod を判断する **PreStop** Pod フックを設定し、それ以外の理由で再起動されないようにする実装など、今後改善される可能性があります。

3. 無効になった Pod が使用するボリュームを消去します。

3.5. MARIADB

3.5.1. 概要

OpenShift Container Platform には、MariaDB の実行用のコンテナーイメージがあります。このイメージでは、設定ファイルで指定されるユーザー名、パスワード、データベース名の設定に基づいてデータベースサービスが提供されます。

3.5.2. バージョン

現時点では、OpenShift Container Platform は MariaDB のバージョン 10.0 および 10.1 をサポートします。

3.5.3. イメージ

これらのイメージには 2 つのフレーバーがあり、ニーズに合わせて選択できます。

- RHEL 7
- CentOS 7

RHEL 7 ベースのイメージ

RHEL 7 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/rhscl/mariadb-100-rhel7
$ docker pull registry.redhat.io/rhscl/mariadb-101-rhel7
```

CentOS 7 ベースのイメージ

これらのイメージは、Docker Hub で入手できます。

```
$ docker pull openshift/mariadb-100-centos7
$ docker pull centos/mariadb-101-centos7
```

これらのイメージを使用するには、イメージレジストリーから直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリーにプッシュしてください。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、OpenShift Container Platform リソースがこの ImageStream を参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して ImageStream の 定義例 があります。

3.5.4. 設定および用途

3.5.4.1. データベースの初期化

共有ボリュームを初めて使用する場合には、データベース、データベースの管理ユーザー、MariaDB root ユーザー (**MYSQL_ROOT_PASSWORD** 環境変数を指定した場合) が作成され、次に MariaDB デーモンが起動します。別のコンテナーにボリュームを再アタッチする場合には、データベース、データベースユーザー、管理者ユーザーは作成されず、MariaDB デーモンが起動します。

以下のコマンドは、新しいデータベースの Pod を作成し、さらにコンテナー内で MariaDB を実行します。

```
$ oc new-app \
    -e MYSQL_USER=<username> \
    -e MYSQL_PASSWORD=<password> \
    -e MYSQL_DATABASE=<database_name> \
    registry.redhat.io/rhscl/mariadb-101-rhel7
```

3.5.4.2. コンテナーでの MariaDB コマンドの実行

OpenShift Container Platform は Software Collections (SCL) を使用して、MariaDB をインストール、起動します。(デバッグ用に) 実行中のコンテナー内で MariaDB コマンドを実行する場合には bash を使用して呼び出す必要があります。

これを実行するには、まず、実行中の MariaDB Pod の名前を特定します。たとえば、現在のプロジェクトで Pod の一覧を表示できます。

```
$ oc get pods
```

次に、Pod に対してリモートシェルセッションを開始します。

```
$ oc rsh <pod>
```

コンテナーに入ると、必要な SCL が自動的に有効になります。

Bash シェルから **mysql** コマンドを実行し、MariaDB の対話セッションを開始して通常の MariaDB 操作が実行できるようになりました。たとえば、データベースユーザーとして認証するには、以下を実行します。

```
bash-4.2$ mysql -u $MYSQL_USER -p$MYSQL_PASSWORD -h $HOSTNAME
$MYSQL_DATABASE
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.5.37 MySQL Community Server (GPL)
...
mysql>
```

完了したら、**quit** または **exit** を入力して MySQL セッションを終了します。

3.5.4.3. 環境変数

MariaDB ユーザー名、パスワード、データベース名は、以下の環境変数で設定する必要があります。

表3.7 MariaDB 環境変数

変数名	説明
MYSQL_USER	作成する MySQL アカウントのユーザー名
MYSQL_PASSWORD	ユーザーアカウントのパスワード

変数名	説明
MYSQL_DATABASE	データベース名
MYSQL_ROOT_PASSWORD	root ユーザーのパスワード (オプション)

警告

ユーザー名、パスワード、データベース名を指定する必要があります。この 3 つすべてを指定しない場合には、Pod は起動に失敗し、OpenShift Container Platform は Pod の再起動を継続的に試行します。

MariaDB 設定は、以下の環境変数で設定できます。

表3.8 MariaDB の他の設定

変数名	説明	デフォルト
MYSQL_LOWER_CASE_TABLE_NAMES	テーブル名の保存および比較方法を設定します。	0
MYSQL_MAX_CONNECTIONS	クライアントが同時に接続可能な最大数	151
MYSQL_MAX_ALLOWED_PACKET	生成された文字列/中間文字列または 1 つのパケットの最大サイズ	200M
MYSQL_FT_MIN_WORD_LEN	FULLTEXT インデックスに含める文字の最小長	4
MYSQL_FT_MAX_WORD_LEN	FULLTEXT インデックスに含める文字の最大長	20
MYSQL_AIO	ネイティブの AIO が壊れている場合に **innodb_use_native_aio** の設定値を制御します。	1
MYSQL_TABLE_OPEN_CACHE	全スレッド用に開くテーブル数	400

変数名	説明	デフォルト
MYSQL_KEY_BUFFER_SIZE	インデックスブロックに使用するバッファーサイズ	32M (または利用可能なメモリーの 10%)
MYSQL_SORT_BUFFER_SIZE	分類に使用するバッファーサイズ	256K
MYSQL_READ_BUFFER_SIZE	シーケンススキャンに使用するバッファーサイズ	8M (または利用可能メモリーの 5%)
MYSQL_INNODB_BUFFER_POOL_SIZE	InnoDB がテーブルやインデックスデータをキャッシュするバッファープールのサイズ	32M (または利用可能なメモリーの 50%)
MYSQL_INNODB_LOG_FILE_SIZE	ロググループにある各ログファイルのサイズ	8M (または利用可能メモリーの 15%)
MYSQL_INNODB_LOG_BUFFER_SIZE	InnoDB がディスクのログファイルへの書き込みに使用するバッファーサイズ	8M (または利用可能メモリーの 15%)
MYSQL_DEFAULTS_FILE	別の設定ファイルを参照します。	/etc/my.cnf
MYSQL_BINLOG_FORMAT	binlog 形式で設定します。サポートされる値は、**row** および **statement** です。	statement

3.5.4.4. ボリュームのマウントポイント

MariaDB イメージは、マウントしたボリュームで実行して、データベース用に永続ストレージを有効化できます。

- **/var/lib/mysql/data**: MySQL のデータディレクトリーは、MariaDB がデータベースファイルを保存する場所にあります。

注記

ホストからコンテナーにディレクトリーをマウントする場合には、マウントしたディレクトリーに適切なパーミッションが設定されていることを確認してください。また、ディレクトリーの所有者とグループが、コンテナー内で実行中のユーザー名と一致することを確認します。

3.5.4.5. パスワードの変更

パスワードはイメージ設定の一部であるため、データベースユーザー (**MYSQL_USER**) と **admin** ユーザーのパスワードを変更するための唯一のサポートされる方法とし、環境変数 **MYSQL_PASSWORD** と **MYSQL_ROOT_PASSWORD** をそれぞれ変更することができます。

現在のパスワードは、Pod またはデプロイメント設定を Web コンソールで表示するか、CLI で環境変数を表示して、確認できます。

```
$ oc set env pod <pod_name> --list
```

SQL ステートメントや、前述した環境変数以外の方法でデータベースのパスワードを変更すると、変数に保存されている値と、実際のパスワードが一致しなくなる可能性があります。データベースコンテナーが起動するたびに、パスワードは環境変数に保存されている値にリセットされます。

これらのパスワードを変更するには、**oc set env** コマンドを使用して、関連するデプロイメント設定の任意の環境変数 1 つまたは両方を更新します。たとえば、テンプレートからアプリケーションを作成する場合など、複数のデプロイメント設定がこれらの環境変数を使用する場合には、デプロイメント設定ごとに変数を更新し、パスワードがどこでも同期されるようにします。これは、すべて同じコマンドで実行できます。

```
$ oc set env dc <dc_name> [<dc_name_2> ...] \
  MYSQL_PASSWORD=<new_password> \
  MYSQL_ROOT_PASSWORD=<new_root_password>
```

重要

アプリケーションによっては、アプリケーションの他の場所にあるパスワードの他の環境変数を更新して一致させる必要があるものもあります。たとえば、フロントエンド Pod のより一般的な **DATABASE_USER** 変数などは、データベースユーザーのパスワードと一致する必要がある場合があります。必要とされる環境変数すべてにおいて、パスワードがアプリケーションごとに一致しているようにしてください。一致しない場合には、トリガーされた時点で、Pod の再デプロイメントが失敗する場合があります。

設定変更トリガーが設定されている場合には、環境変数を更新すると、データベースサーバーの再デプロイメントがトリガーされます。設定されていない場合には、新しいデプロイメントを手動で起動して、パスワードの変更を適用する必要があります。

新規パスワードが有効になっていることを確認するには、まず、実行中の MariaDB Pod へのリモートシェルセッションを開始します。

```
$ oc rsh <pod>
```

Bash シェルから、データベースユーザーの新規パスワードを確認します。

```
bash-4.2$ mysql -u $MYSQL_USER -p<new_password> -h $HOSTNAME
$MYSQL_DATABASE -te "SELECT * FROM (SELECT database()) db CROSS JOIN
(SELECT user()) u"
```

パスワードが正しく変更された場合には、以下のような表が表示されるはずです。

```
+------------+---------------------+
| database() | user()              |
+------------+---------------------+
| sampledb   | user0PG@172.17.42.1 |
+------------+---------------------+
```

root ユーザーの新規パスワードを確認するには、以下を実行します。

```
bash-4.2$ mysql -u root -p<new_root_password> -h $HOSTNAME $MYSQL_DATABASE
-te "SELECT * FROM (SELECT database()) db CROSS JOIN (SELECT user()) u"
```

パスワードが正しく変更された場合には、以下のような表が表示されるはずです。

```
+------------+------------------+
| database() | user()           |
+------------+------------------+
| sampledb   | root@172.17.42.1 |
+------------+------------------+
```

3.5.5. テンプレートからのデータベースサービスの作成

OpenShift Container Platform には テンプレート が含まれており、新規データベースサービスの作成を簡素化します。テンプレートには、必須の環境変数をすべて定義するパラメーターフィールドがあり (ユーザー、パスワード、データベース名など)、自動生成されたパスワード値など、事前定義済みのデフォルト値が設定されます。また、テンプレートは デプロイメント設定 および サービス の両方を定義します。

MariaDB テンプレートは、クラスターの初期設定時にクラスター管理者により、デフォルトの **openshift** プロジェクトに登録しておく必要があります。詳細については、必要に応じて、「デフォルトのイメージストリームおよびテンプレートの読み込み」を参照してください。

利用可能なテンプレートは以下の 2 種類です。

- **mariadb-ephemeral** は、データベースのコンテンツ用に一時ストレージを使用するので、開発またはテスト目的にのみ使用します。つまり、Pod が別の Pod に移動されたり、デプロイメント設定が更新され、再デプロイがトリガーされたりなど、データベース Pod が何らかの理由で再起動された場合には、データはすべて失われます。

- **mariadb-persistent** は、データベースのデータ用に永続ボリュームストアを使用するので、データは Pod が再起動されても残ります。永続ボリュームを使用する場合には、OpenShift Container Platform デプロイメントで定義された永続ボリュームプールが必要で

す。プールの設定に関するクラスター管理者向けの説明は、こちらを参照してください。

この説明に従い、テンプレートをインスタンス化できます。

サービスをインスタンス化したら、データベースにアクセスする予定のある別のコンポーネントのデプロイメント設定に、ユーザー名、パスワード、データベース名の環境変数をコピーできます。このコンポーネントは、定義したサービスを使用してこのデータベースにアクセスできます。

3.5.6. トラブルシューティング

以下のセクションでは、発生する可能性のある問題と、考えられる解決策を説明します。

3.5.6.1. Linux ネイティブの AIO の障害

現象

MySQL コンテナーが起動に失敗し、以下のようなログを出力します。

```
151113  5:06:56 InnoDB: Using Linux native AIO
151113  5:06:56  InnoDB: Warning: io_setup() failed with EAGAIN. Will make
5 attempts before giving up.
InnoDB: Warning: io_setup() attempt 1 failed.
InnoDB: Warning: io_setup() attempt 2 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 3 failed.
InnoDB: Warning: io_setup() attempt 4 failed.
Waiting for MySQL to start ...
InnoDB: Warning: io_setup() attempt 5 failed.
151113  5:06:59  InnoDB: Error: io_setup() failed with EAGAIN after 5
attempts.
InnoDB: You can disable Linux Native AIO by setting innodb_use_native_aio
= 0 in my.cnf
151113  5:06:59 InnoDB: Fatal error: cannot initialize AIO sub-system
151113  5:06:59 [ERROR] Plugin 'InnoDB' init function returned error.
151113  5:06:59 [ERROR] Plugin 'InnoDB' registration as a STORAGE ENGINE
failed.
151113  5:06:59 [ERROR] Unknown/unsupported storage engine: InnoDB
151113  5:06:59 [ERROR] Aborting
```

説明

MariaDB のストレージエンジンは、リソース制限が原因で、カーネルの AIO (非同期 I/O) 機能を使用できませんでした。

解決策

環境変数 **MYSQL_AIO** の値を **0** に設定して、AIO の使用を完全に停止します。今後のデプロイメントでは、この設定により MySQL の設定変数 **innodb_use_native_aio** の値が **0** に指定されます。

または **aio-max-nr** カーネルリソースを増やします。以下の例では、現在の **aio-max-nr** の値を検証して、この値を 2 倍にします。

```
$ sysctl fs.aio-max-nr
fs.aio-max-nr = 1048576
# sysctl -w fs.aio-max-nr=2097152
```

これはノードごとの解決策であるため、次にノードが再起動されるまで有効です。

第4章 他のイメージ

4.1. 概要

以下のトピックには、OpenShift Container Platform ユーザーに提供される、さまざまなコンテナーイメージに関する情報が含まれます。

4.2. JENKINS

4.2.1. 概要

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。

このイメージにはサンプルの Jenkins ジョブが含まれており、OpenShift Container Platform で定義した **BuildConfig** の新しいビルドをトリガーし、そのビルドの出力をテストします。ビルドに成功すると、この出力に再度タグ付けして、ビルドが実稼働環境での使用準備ができたことを示します。詳細情報は、README を参照してください。

OpenShift Container Platform は、Jenkins の LTS リリースに従います。OpenShift Container Platform には、Jenkins 2.x を含むイメージを提供します。Jenkins 1.x の別のイメージが以前は提供されていましたが、このイメージに対するメンテナンスは終了しました。

4.2.2. イメージ

OpenShift Container Platform Jenkins イメージのフレーバーは 2 種類あります。

RHEL 7 ベースのイメージ

RHEL 7 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/openshift3/jenkins-2-rhel7
```

CentOS 7 ベースのイメージ

このイメージは、Docker Hub で入手できます。

```
$ docker pull openshift/jenkins-2-centos7
```

これらのイメージを使用するには、イメージレジストリーから直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリーにプッシュしてください。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、OpenShift Container Platform リソースがこの ImageStream を参照できるようになります。提供されている全 OpenShift Container Platform イメージに対して ImageStream の 定義例 があります。

4.2.3. 設定およびカスタマイズ

4.2.3.1. 認証

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

4.2.3.1.1. OpenShift Container Platform OAuth 認証

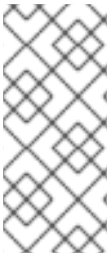
OAuth 認証 は、Jenkins UI の**Configure Global Security** パネルで設定するか、Jenkins **デプロイメント設定** の **OPENSHIFT_ENABLE_OAUTH** 環境変数を **false** 以外に設定して、有効化します。これにより、OpenShift ログインプラグインが有効になり、Pod データからまたは、OpenShift Container Platform API サーバーと対話して設定情報を取得します。

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。たとえば、**Allow All** がデフォルトのアイデンティティープロバイダーの場合には、ユーザー名とパスワードの両方に、空でなければどのような文字列でも指定できます。

Jenkins は ブラウザー および ブラウザー以外 のアクセスをサポートします。

OpenShift Container Platform **ロール** で、ユーザーに割り当てられる固有の Jenkins パーミッションが指定されている場合には、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。

admin ロールが割り当てられたユーザーは、従来の Jenkins 管理ユーザー権限が割り当てられます。**edit** または **view** ロールを持つユーザーのパーミッションは徐々に少なくなります。Jenkins パーミッションと OpenShift ロールのマッピングに関する具体的な情報については、Jenkins image source repository README を参照してください。



注記

OpenShift Container Platform OAuth を使用する場合には、OpenShift Container Platform クラスター管理者が明示的に 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定義し、**admin** ロールを割り当てない限り、OpenShift Container Platform Jenkins イメージ内で管理者権限で事前に生成された **admin** ユーザーには、これらの権限は割り当てられません。

Jenkins のユーザーパーミッションは、最初にユーザー作成してから変更できます。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 認証を使用して最も簡単に Jenkins サービスを作成する方法は、以下の説明のようにテンプレートを使用する方法です。

4.2.3.1.2. Jenkins 標準認証

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は、**admin** および **password** です。標準の Jenkins 認証を使用する場合 (のみ)、デフォルトのパスワードは、**JENKINS_PASSWORD** 環境変数で設定します。

標準の Jenkins 認証を使用して、新しい Jenkins アプリケーションを作成するには以下を実行します。

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    openshift/jenkins-2-centos7
```

4.2.3.2. 環境変数

Jenkins サーバーは、以下の環境変数で設定できます。

- **OPENSHIFT_ENABLE_OAUTH** (デフォルト: **false**)
  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。

- **JENKINS_PASSWORD** (デフォルト: **password**)
  標準の Jenkins 認証を使用時の **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。

- **OPENSHIFT_JENKINS_JVM_ARCH**
  x86_64 または **i386** に設定して、Jenkins のホストに使用する JVM を上書きします。メモリー効率を確保するため、メモリー制限が 2 GiB 以下のコンテナーで実行される場合には、Jenkins イメージはデフォルトで 32 ビットの JVM を動的に使用します。

- **JAVA_MAX_HEAP_PARAM**
  CONTAINER_HEAP_PERCENT (デフォルト: **0.5** または 50%)
  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には (設定例: **-Xmx512m**)、この値が優先されます。設定されていない場合には、最大ヒープサイズは動的に、コンテナーメモリー制限の **CONTAINER_HEAP_PERCENT**% (設定例: **0.5** または 50%) として計算され、オプションで **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB (設定例: **512**) を上限とします。

  デフォルトでは Jenkins JVM の最大ヒープサイズは、上限なしでコンテナーメモリー制限の 50% に設定されます。

- **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には (設定例: **-Xmx32m**)、この値が優先されます。設定されていない場合には、初期ヒープサイズは動的に、コンテナーメモリー制限の **CONTAINER_INITIAL_PERCENT**% (設定例: **0.1** または 10%) として計算されます。

  デフォルトでは、初期のヒープサイズは JVM に依存します。

- **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** (デフォルト: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**)
  対象のコンテナーで実行中のすべての JVM が従う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。

- **JAVA_GC_OPTS** (デフォルト: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -**

XX:AdaptiveSizePolicyWeight=90)
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。

- **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 (以下参照) に、追加の Jenkins プラグインを指定します。プラグインは、名前:バージョンのペアをコンマ区切りの一覧して指定します。設定例: **git:3.7.0,subversion:2.10.2**

- **OPENSHIFT_PERMISSIONS_POLL_INTERVAL** (デフォルト: **300000** - 5 分)
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取得するために OpenShift Container Platform をポーリングする頻度をミリ秒単位で指定します。

- **OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG** (デフォルト: **false**)
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に、永続ボリューム要求の作成により永続ボリュームが割り当てられるので、イメージから永続ボリュームに設定が移行されるのは、イメージの初回起動時だけです。このイメージを拡張するカスタムイメージを作成して、初回起動後にそのカスタムイメージの設定を更新する場合には、デフォルトで、この環境変数を **true** に設定していない限りコピーされません。

- **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** (デフォルト: **false**)
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に、永続ボリューム要求の作成により永続ボリュームが割り当てられるので、イメージから永続ボリュームにプラグインが移行されるのは、イメージの初回起動時だけです。このイメージを拡張するカスタムイメージを作成して、初回起動後にそのカスタムイメージの設定を更新する場合には、デフォルトで、この環境変数を **true** に設定していない限りコピーされません。

- **ENABLE_FATAL_ERROR_LOG_FILE** (default: **false**)
Jenkins 設定ディレクトリー用に OpenShift Container Platform の Persistent Claim (永続要求) 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ても、致命的なエラーのログファイルが永続することを許可します。致命的なエラーのファイルは **/var/lib/jenkins/logs** に保存されます。

- **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。デフォルトの NodeJS エージェント Pod は、Jenkins イメージの CentOS または RHEL バージョンのいずれかを使用しているかによって **docker.io/openshift/jenkins-agent-nodejs-8-centos7** または **registry.redhat.io/openshift3/jenkins-agent-nodejs-8-rhel7** を使用します。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。

- **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。デフォルトの maven エージェント Pod は、Jenkins イメージの CentOS または RHEL バージョンのいずれかを使用しているかによって **docker.io/openshift/jenkins-agent-maven-35-centos7** または **registry.redhat.io/openshift3/jenkins-agent-maven-35-rhel7** を使用します。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。

4.2.3.3. プロジェクト間のアクセス

同じプロジェクト内のデプロイメントとしてではなく、別の場所で 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。

1. Jenkins がアクセスするのに必要なプロジェクトへの適切なパーミッションが指定されているサービスアカウントのシークレットを特定します。

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

   今回の場合は、対象のシークレットは **jenkins-token-uyswp** という名前です。

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> # for example,
jenkins-token-uyswp
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
```

トークンフィールドには、Jenkins がプロジェクトへのアクセスに必要とするトークンの値が含まれます。

4.2.3.4. ボリューム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化できます。

- **/var/lib/jenkins**: これは、Jenkins がジョブの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。

4.2.3.5. Source-To-Image での 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、以下の 2 つのオプションがあります。

- Docker のレイヤリングを使用する
- ここに記載されているように Source-To-Image としてイメージを使用する

S2I を使用して、カスタムの Jenkins ジョブ定義、追加のプラグインをコピーしたり、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
このファイルには、インストールするプラグインを記載します。

```
pluginId:pluginVersion
```

configuration/jobs
このディレクトリーには、Jenkins ジョブ定義を含めます。

configuration/config.xml
このファイルには、カスタムの Jenkins 設定を含めます。

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。

以下は、OpenShift Container Platform で Jenkins イメージをカスタマイズするビルド設定例です。

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

1. **source** フィールドでは、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** フィールドでは、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 フィールドは、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る、カスタマイズして作成された Jenkins イメージを定義します。

4.2.3.6. 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 があり、Kubernetes および OpenShift Container Platform を使用して、Jenkins エージェントを動的に複数のコンテナーホストで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を 5 つ (**Base**、**Maven**、および **Node.js**) 提供します。詳しい情報は、「Jenkins エージェント」を参照してください。

注記

jenkins-slave-maven-* および jenkins-slave-nodejs-* イメージには、v3.10 リリースサイクルで deprecated (非推奨) のマークが付けられます。イメージは、ユーザーがアプリケーションを新規の jenkins-agent-maven-* および jenkins-agent-nodejs-* イメージに移行できるようにしばらく残されます。

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、自動的に Kubernetes Pod テンプレートイメージとして構成されます。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" に設定されている Jenkins ジョブのいずれかに適用できます。ラベルが適用されると、適切なエージェントイメージを実行する OpenShift Container Platform Pod で指定のジョブが実行されます。

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を自動検出および自動設定します。OpenShift 同期プラグイン では、Jenkins の起動時に Jenkins イメージが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記載されているプロジェクト内で、以下がないか検索します。

- 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
- ラベル **role** が **jenkins-slave** に設定されている ConfigMap

適切なラベルまたは、適切なアノテーションが付いたイメージストリームタグが見つかると、適切な Kubernetes プラグイン設定が生成され、イメージストリーム提供のコンテナ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イメージ参照および名前が、Kubernetes プラグインの Pod テンプレートにある名前およびイメージフィールドにマッピングされます。Kubernetes プラグインの Pod テンプレートのラベルフィールドは、イメージストリームにアノテーションを設定するか、イメージストリームタグオブジェクトに **slave-label** キーを設定して制御できます。これらを使用しない場合には、名前をラベルとして使用します。

適切なラベルが指定された ConfigMap が見つかった場合には、ConfigMap のキーおよび値のデータペイロードに 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 が含まれることを前提とします。ConfigMap を使用時に注意すべき主な違いは、イメージストリームまたはイメージストリームタグではなく、Kubernetes プラグインの Pod テンプレートの各種フィールドすべてを制御できます。

以下は ConfigMap の例です。

```
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>
```

起動後に 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節します。

特に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てしまいます。
- 同様に、これらのオブジェクトが削除されると、該当の設定が Kubernetes プラグインから削除されます。
- それとは逆に、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** が作成されてしまいます。

- **ConfigMap** フォームを使用した **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** の変更は上書きされます。

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、スレーブエージェントをエントリーポイントとして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。

4.2.3.6.1. パーミッションの留意事項

以前の ConfigMap の例では、Pod テンプレート XML の **<serviceAccount>** 要素は、作成される Pod で使用する OpenShift Container Platform サービスアカウントとなっています。Pod にマウントされたサービスアカウントの認証情報と、そのサービスアカウントに関連付けられたパーミッションで、どの OpenShift Container Platform マスターが Pod からアクセスできるかを制御します。

OpenShift Container Platform Jenkins イメージで実行される Kubernetes プラグインで起動される Pod で使用されるサービスアカウントでは、以下を考慮してください。

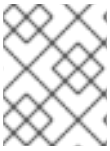
- OpenShift Container Platform で提供される Jenkins のテンプレートサンプルを使用する場合には、**jenkins** サービスアカウントが、Jenkins が実行されているプロジェクトの **edit** ロールで定義され、マスター Jenkins Pod にサービスアカウントがマウントされます。
- Jenkins 設定に挿入される、2 つのデフォルトの Maven および NodeJS Pod テンプレートは、マスターと同じサービスアカウンを使用するように設定されます。
- イメージストリームやイメージストリームタグに必須のラベルやアノテーションが指定されている結果として、OpenShift Sync プラグイン が自動検出した Pod テンプレートには、マスターのサービスアカウントがそのサービスアカウントとして設定されます。
- 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ですが、使用するサービスアカウントを明示的に指定します。
- 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 で定義して、Pod から操作することにした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。

4.2.4. 使用法

4.2.4.1. 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全環境変数 (パスワード) 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。クラスター管理者は、初期のクラスター設定時に、Jenkins テンプレートをデフォルトの **openshift** プロジェクトに登録しておく必要があります。詳細は、必要に応じて、「デフォルトのイメージストリームとテンプレートの読み込み」を参照してください。

デプロイメント設定およびサービス。



注記

Pod は、別のノードに移動時、または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-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。

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します。

4.2.4.2. Jenkins Kubernetes プラグインの使用

新規 Jenkins サービスの作成

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。この Pod は Java ソースのクローン作成、WAR ファイルのビルドを行い、次に 2 番目の BuildConfig (openshift-jee-sample-docker) を実行して、新規作成した WAR ファイルをコンテナーイメージに階層を作成します。

同様のタスクを実行する、より詳細にわたる例は こちらで確認できます。

例4.1 Jenkins Kubernetes プラグインを使用した BuildConfig の例

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

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

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

例4.2 Jenkins Kubernetes プラグインを使用した BuildConfig の例 (メモリー制限および環境変数の指定)

```
kind: BuildConfig
apiVersion: v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", 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」と呼ばれる新規 Pod テンプレートがその場で定義されます。この新しい Pod テンプレート名は、以下のノード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。今回の例では、OpenShift Container Platform で事前定義された「maven」Pod テンプレートを継承します。

4 既存のコンテナーの値を上書きするので、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」を使用します。

5 コンテナーイメージは、再度指定する必要があります。これは既知の問題です。

6 512Mi のメモリー要求を指定します。

7 512Mi のメモリー制限を指定します。

8 環境変数 CONTAINER_HEAP_PERCENT に「0.25」の値を指定します。

9 ノードスタンザは、上記で新たに定義した 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、プラグインを使用するか、またはパイプライン Jenkinsfile 内で変更できます。詳細は、「エージェント Pod の保持」を参照してください。

Kubernetes プラグインの設定に関する詳細は、Kubernetes プラグインのドキュメントを参照してください。

4.2.4.3. 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 512MiB です。

Jenkins が使用する JVM のチューニングに関する参考情報は、「OpenShift Container Platform での OpenJDK のサイジング」を参照してください。

メモリー効率に関して、メモリー制限が 2 GiB 以下に指定されたコンテナーで実行中の場合には、デフォルトで Jenkins イメージが動的に 32 ビットの JVMを使用します。この動作は、**OPENSHIFT_JENKINS_JVM_ARCH** 環境変数で上書きできます。

デフォルトでは、Jenkins 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。また、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。詳しい情報は、「環境変数」を参照してください。

デフォルトでは、パイプラインからローカルで実行されるシェルスクリプトや **oc** コマンドなど、Jenkins コンテナーで実行される他の全プロセスでは、OOM 終了を呼び出さずに、残りの 256MiB 以上のメモリーを組み合わせて使用できる可能性が低い点を考慮してください。そのため、パイプラインは、できる限りエージェントコンテナーで外部のコマンドを実行することを強く推奨します。

Jenkins Kubernetes プラグインを作成した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を推奨します。管理者は、Jenkins 設定を使用して、デフォルトはエージェントのイメージごとに設定できます。メモリー要求および制限は、上記 のようにコンテナーベースでも上書きできます。

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

4.2.5. Jenkins プラグイン

以下のプラグインは、Jenkins と OpenShift Container Platform の統合用に提供されています。これらのプラグインは、デフォルトで Jenkins イメージに含まれています。

4.2.5.1. OpenShift Container Platform Client プラグイン

OpenShift Container Platform クライアントプラグインは、OpenShift Container Platform と強力な対話を行うために、信頼性があり、簡潔で包括的な、流れるような (fluent) Jenkins Pipeline 構文を提供することが目的です。このプラグインは **oc** バイナリーを活用しますが、このバイナリーは、スクリプトを実行するノードで利用できるようにしておく必要があります。

このプラグインは、Jenkins イメージで完全にサポートされており、イメージに含まれています。このプラグインでは以下が提供されます。

- Jenkins Pipeline で使用するための流れるような構文
- **oc** で利用可能なオプションでの使用および公開
- Jenkins の認証情報およびクラスターとの統合
- 従来の Jenkins Freestyle ジョブに対する継続的なサポート

詳しい情報は、「OpenShift Pipeline ビルドのチュートリアル」およびプラグインの README を参照してください。

4.2.5.2. OpenShift Container Platform Pipeline プラグイン

OpenShift Container Platform Pipeline プラグインは、Jenkins と OpenShift Container Platform が統合する前のプラグインで、OpenShift Container Platform クライアントプラグインよりも機能が少なくなっています。これは非推奨になっていますが、引き続き v3.11 までの OpenShift Container Platform バージョンで機能します。これより後のバージョンでは、Jenkins Pipeline から直接 **oc** バイナリーを使用するか、または OpenShift Container Platform クライアントプラグインを使用します。

詳しい情報は、プラグインの README を参照してください。

4.2.5.3. OpenShift Container Platform Sync プラグイン

OpenShift Container Platform Pipeline ビルドストラテジーが Jenkins と OpenShift Container Platform をスムーズに統合できるように、OpenShift Sync プラグイン は OpenShift Container Platform の API サーバーをモニタリングして、Pipeline ストラテジーを採用する **BuildConfigs** と **Builds** に更新がないか確認し、Jenkins Pipeline プロジェクトを作成するか (**BuildConfig** の作成時) または、作成されたプロジェクトでジョブを開始します (**Build** の起動時)。

「Jenkins Kubernetes プラグインの設定」で記載されているように、このプラグインは、OpenShift Container Platform に定義済みで具体的に引用された **ImageStream**、**ImageStreamTag** または **ConfigMap** オブジェクトをベースに、Kubernetes プラグインの **PodTemplate** 設定を作成できます。

このプラグインは、**credential.sync.jenkins.openshift.io** のラベルキーと、**true** のラベルの値が指定された **Secret** オブジェクトを受け入れて Jenkins 認証情報を構築し、Jenkins 認証情報階層内のデフォルトのグローバルドメインに配置できるようになりました。認証情報の ID は、**Secret** が定義されている namespace、ハイフン (**-**)、その後の **Secret** の名前で構成されます。

PodTemplates の **ConfigMaps** の処理と同様に、OpenShift Container Platform で定義された

Secret オブジェクトは、マスター設定とみなされます。OpenShift Container Platform のオブジェクトのその後の更新は、Jenkins 認証情報に適用されます (その間に認証情報に加えられた変更を上書きします)。

credential.sync.jenkins.openshift.io プロパティーを削除したり、このプロパティーを **true** 以外に設定したり、OpenShift Container Platform から **Secret** を削除したりすると、Jenkins の関連の認証情報が削除されます。

シークレットのタイプは、以下のように Jenkins 認証情報タイプにマッピングされます。

- Opaque タイプの **Secret** オブジェクトの場合には、プラグインは **data** セクション内で **username** および **password** を検索し、Jenkins UsernamePasswordCredentials 認証情報を構築します。OpenShift Container Platform では、**password** フィールドには、実際のパスワードまたはユーザーの一意のトークンを指定できることを忘れないでください。これらが指定されていない場合には、**ssh-privatekey** フィールドを検索し、Jenkins BasicSSHUserPrivateKey の認証情報を作成します。
- **kubernetes.io/basic-auth** タイプの `Secret` オブジェクトの場合は、プラグインは Jenkins UsernamePasswordCredentials の認証情報を作成します。
- **kubernetes.io/ssh-auth** タイプの **Secret** オブジェクトの場合には、プラグインは Jenkins BasicSSHUserPrivateKey 認証情報を作成します。

4.2.5.4. Kubernetes プラグイン

Kubernetes プラグインを使用して、クラスターの Pod として Jenkins エージェントを実行します。Kubernetes プラグインの自動設定については、「Jenkins Kubernetes プラグインの使用」で説明しています。

4.3. JENKINS エージェント

4.3.1. 概要

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 (**Base**、**Maven** および **Node.js**) 提供します。

最初は、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) および役立つツール (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。
- CentOS および RHEL イメージ両方に対して 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には、CentOS および RHEL 両方用の Dockerfiles が含まれており、新規エージェントイメージをビルドする場合に参照できます。また、**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、イメージについて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

使用する OpenShift Container Platform 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、継承するようにしてください。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には、予期せぬ動作が発生する可能性があります。詳細情報は、「「versioning policy」を参照してください。

4.3.2. イメージ

OpenShift Container Platform Jenkins エージェントイメージのフレーバーは 2 種類あります。

RHEL 7 ベースのイメージ

RHEL 7 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/openshift3/jenkins-slave-base-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-slave-maven-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-slave-nodejs-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-agent-maven-35-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-agent-nodejs-8-rhel7
```

CentOS 7 ベースのイメージ

これらのイメージは、Docker Hub で入手できます。

```
$ docker pull openshift/jenkins-slave-base-centos7
$ docker pull openshift/jenkins-slave-maven-centos7
$ docker pull openshift/jenkins-slave-nodejs-centos7
$ docker pull openshift/jenkins-agent-maven-35-centos7
$ docker pull openshift/jenkins-agent-nodejs-8-centos7
```

これらのイメージを使用するには、イメージレジストリーから直接アクセスするか、ご自身の OpenShift Container Platform コンテナーイメージレジストリーにプッシュしてください。

4.3.3. 設定およびカスタマイズ

4.3.3.1. 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。

- **OPENSHIFT_JENKINS_JVM_ARCH**
  x86_64 または **i386** に設定して、Jenkins エージェントのホストに使用する JVM を上書きします。メモリー効率を確保するため、メモリー制限が 2 GiB のコンテナーで実行される場合には、Jenkins エージェントイメージはデフォルトで 32 ビットの JVM を動的に使用します。

- **JAVA_MAX_HEAP_PARAM**
  CONTAINER_HEAP_PERCENT (デフォルト: **0.5**、50%)
  JNLP_MAX_HEAP_UPPER_BOUND_MB
  これらの値は Jenkins エージェントの 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には (設定例: **-Xmx512m**)、この値が優先されます。設定されていない場合には、最大ヒープサイズは動的に、コンテナーメモリー制限

の **CONTAINER_HEAP_PERCENT**% (設定例: **0.5**、50%) として計算され、オプションで **JNLP_MAX_HEAP_UPPER_BOUND_MB** MiB (設定例: **512**) を上限とします。

デフォルトでは Jenkins エージェントの 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_INITIAL_HEAP_PARAM**
CONTAINER_INITIAL_PERCENT
これらの値は Jenkins エージェントの 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には (設定例: **-Xmx32m**)、この値が優先されます。設定されていない場合には、初期ヒープサイズは動的に、コンテナーメモリー制限の **CONTAINER_INITIAL_PERCENT**% (設定例: **0.1**、10%) として計算されます。

  デフォルトでは、初期のヒープサイズは JVM に依存します。

- **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** (デフォルト: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**)
対象のコンテナーで実行中のすべての JVM が従う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。

- **JAVA_GC_OPTS** (デフォルト: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**)
Jenkins エージェントの 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。

- **JNLP_JAVA_OVERRIDES**
Jenkins エージェントの 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**

4.3.4. 使用法

4.3.4.1. メモリーの要件

Jenkins JNLP エージェントのホストや、Java アプリケーション (例: **javac**、Maven または Gradle) の実行に、Jenkins すべてにおいて JVM を使用します。Jenkins エージェントが使用する JVM のチューニングに関する参考情報は、「OpenShift Container Platform での OpenJDK のサイジング」を参照してください。

メモリー効率に関して、メモリー制限が 2 GiB 以下指定されたコンテナーで実行中の場合には、デフォルトで Jenkins イメージが動的に 32 ビットの JVMを使用します。この動作は、**OPENSHIFT_JENKINS_JVM_ARCH** 環境変数で上書きできます。JVM の選択は、デフォルトでエージェントコンテナー内の Jenkins JNLP エージェントおよび、他の Java プロセスの両方に適用されます。

デフォルトでは、Jenkins JVM エージェント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。また、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。詳しい情報は、「環境変数」を参照してください。

デフォルトでは、パイプラインから実行されるシェルスクリプトや **oc** コマンドなど、Jenkins エージェントコンテナーで実行される他の全プロセスでは、OOM kill を呼び出さずに、メモリー制限の残り 50% を超えたメモリーを使用できる可能性が低い点を考慮してください。

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 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。これは、多くのビルドワークロード用にチューニングする必要がある場合があります。詳細情報は、「OpenShift Container Platform での OpenJDK のサイジング」を参照してください。

Jenkins エージェントコンテナーのメモリー要求や制限の指定に関する情報は、Jenkins ドキュメントを参照してください。

4.3.4.1.1. Gradle ビルド

OpenShift 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、Gradle が 3 番目の JVM を起動してテストを実行するので、複雑性が増します。

OpenShift での JVM のチューニングに関する参考情報は、「OpenShift Container Platform での OpenJDK のサイジング」を参照してください。

以下の設定は、OpenShift 上のメモリーに制約がある Jenkins エージェントで 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、後で設定を緩和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、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** が指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。
- build.gradle ファイルでの **java { options.fork = false }** を設定して、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。
- OpenShift Container Platform での OpenJDK のサイジングに従い、GRADLE_OPTS, JAVA_OPTS または JAVA_TOOL_OPTIONS の環境変数で、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 の設定、または **-Dorg.gradle.jvmargs** コマンドラインの引数で、Gradle テスト JVM に最大ヒープサイズおよび JVM の引数を設定します。

4.3.5. エージェント Pod の保持

Jenkins エージェント Pod (スレーブ Pod としても知られている) はビルドが完了するか、または中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、引き続き実行され、リソースクォータに対してカウントされます。

4.4. 他のコンテナーイメージ

Red Hat Container Catalog に含まれていないコンテナーイメージを使用する場合には、Docker Hub にある他の任意のコンテナーイメージを OpenShift Container Platform インスタンスで使用できます。

任意の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する場合の OpenShift Container Platform 固有のガイドラインについては、『イメージの作成』ガイドの「任意の ID のサポート」を参照してください。

重要

サポートの詳細については、「OpenShift Enterprise サポートポリシー」で定義されている製品サポートの対象範囲を参照してください。

「システムおよび環境要件」のセキュリティー警告も参照してください。

第5章 XPAAS ミドルウェアイメージ

5.1. 概要

Red Hat は、OpenShift Container Platform 用に設計されたミドルウェア製品のホストとして使用できるように、コンテナー化された xPaaS イメージを提供します。これらのイメージに関するドキュメントは、Red Hat カスタマーポータル で参照できます。