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: 2023-03-25

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

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Using_Images.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

以下のトピックを使用して、OpenShift Container Platform 3.11 ユーザーが利用できる、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 6

第2章 SOURCE-TO-IMAGE (S2I) 7
2.1. 概要 7
2.2. JAVA 7
2.2.1. 概要 7
2.2.2. バージョン 7
2.2.3. イメージ 7
2.2.4. ビルドプロセス 8
2.2.5. 設定 8
2.2.6. Java アプリケーションのビルドとデプロイ 9
2.2.7. ソースからのビルドとデプロイ 9
2.2.8. バイナリーアーティファクトからのビルドおよびデプロイ 9
2.2.9. Java 環境変数 10
2.2.10. 関連情報 21
2.3. .NET CORE 21
2.3.1. .NET Core を使用する利点 21
2.3.2. サポート対象バージョン 21
2.3.3. イメージ 21
2.3.4. ビルドプロセス 21
2.3.5. 環境変数 22
2.3.6. .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 24
2.4. NODE.JS 24
2.4.1. 概要 24
2.4.2. バージョン 24
2.4.3. イメージ 24
2.4.4. ビルドプロセス 25
2.4.5. 設定 25
2.4.6. ホットデプロイ 26
2.5. PERL 26
2.5.1. 概要 26
2.5.2. バージョン 26
2.5.3. イメージ 27
2.5.4. ビルドプロセス 27
2.5.5. 設定 27
2.5.6. ログへのアクセス 28
2.5.7. ホットデプロイ 28
2.6. PHP 29
2.6.1. 概要 29
2.6.2. バージョン 29
2.6.3. イメージ 29
2.6.4. ビルドプロセス 30
2.6.5. 設定 30
2.6.5.1. Apache 設定 32
2.6.6. ログへのアクセス 32
2.6.7. ホットデプロイ 32
2.7. PYTHON 33
2.7.1. 概要 33
2.7.2. バージョン 33
2.7.3. イメージ 33
2.7.4. ビルドプロセス 34

2.7.5. 設定 34
2.7.6. ホットデプロイ 35
2.8. RUBY 35
2.8.1. 概要 35
2.8.2. バージョン 36
2.8.3. イメージ 36
2.8.4. ビルドプロセス 36
2.8.5. 設定 37
2.8.6. ホットデプロイ 38
2.9. S2I イメージのカスタマイズ 39
2.9.1. 概要 39
2.9.2. イメージに埋め込まれたスクリプトの呼び出し 39

第3章 データベースイメージ 41
3.1. 概要 41
3.2. MYSQL 41
3.2.1. 概要 41
3.2.2. バージョン 41
3.2.3. イメージ 41
3.2.4. 設定と使用方法 42
3.2.4.1. データベースの初期化 42
3.2.4.2. コンテナーでの MySQL コマンドの実行 42
3.2.4.3. 環境変数 43
3.2.4.4. ボリュームマウントポイント 45
3.2.4.5. パスワードの変更 45
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 46
3.2.6. MySQL レプリケーションの使用 47
3.2.6.1. MySQL マスターのデプロイメント設定の作成 47
3.2.6.2. ヘッドレスサービスの作成 50
3.2.6.3. MySQL スレーブのスケーリング 51
3.2.7. トラブルシューティング 51
3.2.7.1. Linux ネイティブ AIO の障害 51
3.3. POSTGRESQL 52
3.3.1. 概要 52
3.3.2. バージョン 52
3.3.3. イメージ 52
3.3.4. 設定と使用方法 53
3.3.4.1. データベースの初期化 53
3.3.4.2. コンテナーでの PostgreSQL コマンドの実行 53
3.3.4.3. 環境変数 54
3.3.4.4. ボリュームマウントポイント 55
3.3.4.5. パスワードの変更 55
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 56
3.4. MONGODB 57
3.4.1. 概要 57
3.4.2. バージョン 57
3.4.3. イメージ 57
3.4.4. 設定と使用方法 57
3.4.4.1. データベースの初期化 57
3.4.4.2. コンテナーでの MongoDB コマンドの実行 58
3.4.4.3. 環境変数 58
3.4.4.4. ボリュームマウントポイント 59
3.4.4.5. パスワードの変更 60

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 61
3.4.6. MongoDB レプリケーション 61
3.4.6.1. 制限 62
3.4.6.2. サンプルテンプレートの使用 63
3.4.6.3. スケールアップ 64
3.4.6.4. スケールダウン 64
3.5. MARIADB 65
3.5.1. 概要 65
3.5.2. バージョン 65
3.5.3. イメージ 65
3.5.4. 設定と使用方法 65
3.5.4.1. データベースの初期化 65
3.5.4.2. コンテナーでの MariaDB コマンドの実行 66
3.5.4.3. 環境変数 66
3.5.4.4. ボリュームマウントポイント 68
3.5.4.5. パスワードの変更 69
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 70
3.5.6. トラブルシューティング 70
3.5.6.1. Linux ネイティブ AIO の障害 71

第4章 その他のイメージ 72
4.1. 概要 72
4.2. JENKINS 72
4.2.1. 概要 72
4.2.2. イメージ 72
4.2.3. 設定とカスタマイズ 72
4.2.3.1. 認証 72
4.2.3.1.1. OpenShift Container Platform OAuth 認証 73
4.2.3.1.2. Jenkins 標準認証 73
4.2.3.2. 環境変数 74
4.2.3.3. プロジェクト間のアクセス 76
4.2.3.4. ボリュームマウントポイント 76
4.2.3.5. S2I (Source-To-Image) による Jenkins イメージのカスタマイズ 77
4.2.3.6. Jenkins Kubernetes プラグインの設定 78
4.2.3.6.1. パーミッションに関する考慮事項 80
4.2.4. 使用法 80
4.2.4.1. テンプレートからの Jenkins サービスの作成 81
4.2.4.2. Jenkins Kubernetes プラグインの使用 81
4.2.4.3. メモリーの要件 83
4.2.5. Jenkins プラグイン 84
4.2.5.1. OpenShift Container Platform クライアントプラグイン 84
4.2.5.2. OpenShift Container Platform Pipeline プラグイン 84
4.2.5.3. OpenShift Container Platform 同期プラグイン 84
4.2.5.4. Kubernetes プラグイン 85
4.3. JENKINS エージェント 85
4.3.1. 概要 85
4.3.2. イメージ 86
4.3.3. 設定とカスタマイズ 86
4.3.3.1. 環境変数 86
4.3.4. 使用法 87
4.3.4.1. メモリーの要件 87
4.3.4.1.1. Gradle ビルド 88
4.3.5. エージェント Pod の保持 88

4.4. 他のコンテナーイメージ 89

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、その他のコンテナーイメージを確認します。

Red Hat の公式コンテナーイメージは、registry.redhat.io の Red Hat レジストリーで提供されています。OpenShift Container Platform がサポートする S2I、データベース、Jenkins イメージは Red Hat 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 イメージの **registry.redhat.io/openshift3/ose** などです。

xPaaS ミドルウェアイメージは、Red Hat レジストリーのそれぞれの製品リポジトリーで提供されていますが、接尾辞として **-openshift** が付いています。たとえば、JBoss EAP イメージの **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。関連するイメージを参照または検索してください。

重要

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

第2章 SOURCE-TO-IMAGE (S2I)

2.1. 概要

このトピックグループには、OpenShift Container Platform ユーザーが利用できる、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. JAVA

2.2.1. 概要

OpenShift Container Platform は、Java アプリケーションをビルドするための S2I ビルダーイメージを提供します。これらのビルダーイメージは、アプリケーションソースまたはバイナリーアーティファクトを取得し、ソースが提供された場合に Maven を使用してソースをビルドし、アーティファクトを必要な依存関係でアセンブルして、Java アプリケーションを含む新規のすぐに実行可能なイメージを作成します。結果として生成されるイメージは、OpenShift Container Platform で実行することも、Docker で直接実行することもできます。

ビルダーイメージは、メインクラスで実行される Maven ベースの Java スタンドアロンプロジェクトで使用することを目的としています。

2.2.2. バージョン

現在のバージョンの Java S2I ビルダーイメージは、OpenJDK 1.8、11、Jolokia 1.6.2、および Maven 3.6 をサポートします。

2.2.3. イメージ

RHEL 7 および RHEL 8 イメージは、Red Hat レジストリーから入手できます。

注記

registry.redhat.io では認証が必要です。registry.redhat.io の環境の設定方法に関する詳細は、Red Hat Container Registry Authentication を参照してください。

RHEL 7 ベースのイメージ

```
$ docker pull registry.redhat.io/redhat-openjdk-18/openjdk18-openshift
$ docker pull registry.redhat.io/openjdk/openjdk-11-rhel7
```

RHEL 8 ベースのイメージ

```
$ docker pull registry.redhat.io/ubi8/openjdk-8
$ docker pull registry.redhat.io/ubi8/openjdk-11
```

OpenShift Container Platform でこれらのイメージを使用するには、Red Hat レジストリー から直接アクセスするか、これらをご自身の 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で、イメージを参照する イメージストリーム を作成することができます。その後、OpenShift Container Platform リソースは イメージストリーム定義 を参照できます。

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、コンテナーにそのソースコードを実行用に準備させることにより、すぐに実行できる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概要は、S2I Build Process を参照してください。

2.2.5. 設定

デフォルトでは、Java S2I ビルダーイメージは Maven を使用して、以下のゴールおよびオプションでプロジェクトをビルドします。

```
mvn -e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga -Dfabric8.skip=true --batch-mode
-Djava.net.preferIPv4Stack=true -s /tmp/artifacts/configuration/settings.xml -
Dmaven.repo.local=/tmp/artifacts/m2  package
```

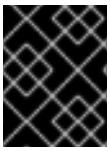
これらのデフォルトに基づいて、ビルダーイメージはプロジェクトをコンパイルし、テストを実行せずにすべての推移的な依存関係を出力ディレクトリーにコピーします。また、プロジェクトに **openshift** という名前のプロファイルがある場合、ビルド用にアクティブ化されます。

以下の環境変数を指定すると、これらのデフォルトのゴールおよびオプションを上書きできます。

変数名	説明
MAVEN_S2I_ARTIFACT_DIRS	ビルド出力用にスキャンするソースディレクトリーの相対パス。これは **DEPLOY_DIR** にコピーされます。デフォルトは **target** です。
JAVA_MAIN_CLASS	**java** の引数として使用する メインクラス。この環境変数が指定されると、**JAVA_APP_DIR** のすべての jar ファイルがクラスパスと **JAVA_LIB_DIR** に追加されます。
MAVEN_ARGS	**mvn** コマンドに渡される引数。この変数を定義すると、デフォルトの **-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package** が置き換えられます。
MAVEN_ARGS_APPEND	追加の Maven 引数。

これは、OpenJDK コンテナーの動作を設定するために使用できる環境変数のセレクションです。包括的な一覧は、「Java 環境変数」 を参照してください。

2.2.6. Java アプリケーションのビルドとデプロイ



重要

OpenJDK イメージストリーム を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。

同じ S2I ビルダーイメージを使用して、ソースまたはバイナリーアーティファクトから Java アプリケーションをビルドできます。

2.2.7. ソースからのビルドとデプロイ

Java S2I ビルダーイメージは、ソースリポジトリーに対して **oc new-app** を実行してアプリケーションをソースからビルドするために使用できます。

```
$ oc new-app registry.redhat.io/redhat-openjdk-18/openjdk18-openshift~https://github.com/jboss-openshift/openshift-quickstarts --context-dir=undertow-servlet
```

デフォルトでは、テストは実行されません。アプリケーションをビルドし、ビルドの一部としてテストを実行するには、以下のコマンドに示すようにデフォルトの **MAVEN_ARGS** を上書きします。

```
$ oc new-app registry.redhat.io/redhat-openjdk-18/openjdk18-openshift~<git_repo_URL> --context-dir=<context_dir> --build-env='MAVEN_ARGS=-e -Popenshift -Dcom.redhat.xpaas.repo.redhatga package'
```

Java プロジェクトが複数の Maven モジュールで設定される場合は、アーティファクトの出力ディレクトリーを明示的に指定すると便利です。Maven プロジェクトがアーティファクトを出力するディレクトリーを指定すると、S2I ビルドがこれらを取得できるようになります。

ビルドするモジュールおよびアーティファクトの出力ディレクトリーを指定するには、以下のコマンドを使用します。

```
$ oc new-app registry.redhat.io/redhat-openjdk-18/openjdk18-openshift~<git_repo_URL> --context-dir=<context_dir> --build-env='MAVEN_S2I_ARTIFACT_DIRS=relative/path/to/artifacts/dir' --build-env='MAVEN_ARGS=install -pl <groupId>:<artifactId> -am'
```

2.2.8. バイナリーアーティファクトからのビルドおよびデプロイ

Java S2I ビルダーイメージを使用して、提供するバイナリーアーティファクトを使用するアプリケーションを構築できます。

1. 新しいバイナリービルドを作成します。

   ```
   $ oc new-build --name=<application_name> registry.redhat.io/redhat-openjdk-18/openjdk18-openshift --binary=true
   ```

2. ビルドを開始し、ローカルマシンのバイナリーアーティファクトへのパスを指定します。

   ```
   $ oc start-build <application_name> --from-dir=/path/to/artifacts --follow
   ```

3. アプリケーションの作成

```
$ oc new-app <application_name>
```

2.2.9. Java 環境変数

以下の表は、OpenJDK コンテナーの動作を設定するために使用される Java 環境変数の包括的な一覧です。

表2.1 設定環境変数

変数名	説明	値の例
AB_JOLOKIA_CONFIG	設定された場合、Jolokia の リファレンスマニュアル で説明されているように、パスを含むこのファイルを Jolokia JVM エージェントプロパティーとして使用します。設定しない場合、マニュアルで定義された設定を使用して、**/opt/jolokia/etc/jolokia.properties** が作成されます。それ以外の場合、本書の残りの設定は無視されます。	**/opt/jolokia/custom.properties**
AB_JOLOKIA_DISCOVERY_ENABLED	Jolokia の検索を有効にします。デフォルトは **false** です。	**true**
AB_JOLOKIA_HOST	バインド先のホストアドレス。デフォルトは **0.0.0.0** です。	**127.0.0.1**
AB_JOLOKIA_ID	使用するエージェント ID。デフォルトはコンテナー ID の **$HOSTNAME** です。	**openjdk-app-1-xqlsj**
AB_JOLOKIA_OFF	設定されている場合には、Jolokia のアクティベートを無効にします。たとえば、空の値を echo します。Jolokia はデフォルトで有効になります。	**true**
AB_JOLOKIA_OPTS	エージェント設定に追加されるその他のオプション。**key=value,key=value,...** の形式で表示される必要があります。	**backlog=20**
AB_JOLOKIA_PASSWORD	Basic 認証のパスワード。デフォルトでは認証は無効になっています。	**mypassword**
AB_JOLOKIA_PORT	リッスンするポート。デフォルトは **8778** です。	**5432**

変数名	説明	値の例
AB_JOLOKIA_USER	Basic 認証のユーザー。デフォルトは **jolokia** です。	**myusername**
AB_PROMETHEUS_ENABLE	Prometheus エージェントの使用を有効にします。	**true**
AB_PROMETHEUS_JMX_EXPORTER_PORT	Prometheus JMX Exporter に使用するポート。	**9799**
CONTAINER_CORE_LIMIT	CFS Bandwidth Control で説明されているように、計算されたコア制限。	**2**
CONTAINER_MAX_MEMORY	コンテナーに指定されたメモリー制限。	**1024**
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	以前の GC 時間に対する現在の GC 時間の重み付け。	**90**
GC_CONTAINER_OPTIONS	使用する Java GC を指定します。この変数の値には、必要な GC を指定するのに必要な JRE コマンドラインインターフェイスオプションが含まれる必要があります。これは、**-XX:+UseParallelOldGC** のデフォルトを上書きします。	**-XX:+UseG1GC**
GC_MAX_HEAP_FREE_RATIO	縮小を回避するための GC 後のヒープ解放の最大パーセンテージ。	**40**
GC_MAX_METASPACE_SIZE	メタスペースの最大サイズ。	**100**
GC_METASPACE_SIZE	初期メタスペースのサイズ。	**20**
GC_MIN_HEAP_FREE_RATIO	拡大を回避するための GC 後のヒープ解放の最小パーセンテージ。	**20**
GC_TIME_RATIO	ガベージコレクションに費やした時間に対する、ガベージコレクション外で費やした時間 (アプリケーションの実行に費やした時間など) の比率を指定します。	**4**

変数名	説明	値の例
HTTPS_PROXY	https プロキシーの場所。これは **http_proxy** および **HTTP_PROXY** に優先され、Maven ビルドと Java ランタイムの両方に使用されます。	**myuser@127.0.0.1:8080**
HTTP_PROXY	http プロキシーの場所。これは、Maven ビルドと Java ランタイムの両方に使用されます。	**127.0.0.1:8080**
JAVA_APP_DIR	アプリケーションがあるディレクトリー。アプリケーションのすべてのパスは、このディレクトリーに相対的です。	**myapplication/`**
JAVA_ARGS	**java** アプリケーションに渡される引数。	-
JAVA_CLASSPATH	使用するクラスパス。指定しない場合、起動スクリプトは **JAVA_APP_DIR/classpath** ファイルを確認し、その内容をそのままクラスパスとして使用します。このファイルが存在しない場合は、app dir のすべての jar が追加されます (**classes:JAVA_APP_DIR/***)。	-
JAVA_DEBUG	設定されている場合、リモートデバッグがオンになります。デフォルトでは無効にされています。	**true**
JAVA_DEBUG_PORT	リモートデバッグに使用されるポート。デフォルトは **5005** です。	**8787**
JAVA_DIAGNOSTICS	この変数を設定して、問題が発生した場合に診断情報の一部を標準出力に取得します。デフォルトでは無効にされています。	**true**

変数名	説明	値の例
JAVA_INITIAL_MEM_RATIO	**JAVA_OPTS** に **-Xms** オプションが指定されていない場合に使用されます。これは、最大ヒープメモリーを基にデフォルトの初期ヒープメモリーを算出するために使用されます。コンテナーのメモリー制約のないコンテナーで使用されると、このオプションは効果がありません。メモリー制約がある場合、**-Xms** はここで設定されている **-Xmx** メモリーの比率に設定されます。デフォルトは **25** で、**-Xmx** の 25% が初期ヒープサイズとして使用されることを意味します。このメカニズムを省略するには、この値を **0** に設定します。この場合、**-Xms** オプションは追加されません。	**25**
JAVA_LIB_DIR	Java jar ファイルと、クラスパスを 1 行のクラスパス (コロン区切り) として、または 1 行ずつリストされた jar ファイルでクラスパスを保持するオプションの **classpath** ファイルを保持するディレクトリー。設定されていない場合は、**JAVA_LIB_DIR** は **JAVA_APP_DIR** と同じになります。	-
JAVA_MAIN_CLASS	**java** の引数として使用する メインクラス。この環境変数が指定されると、**JAVA_APP_DIR** のすべての jar ファイルがクラスパスと **JAVA_LIB_DIR** に追加されます。	**com.example.MainClass**
JAVA_MAX_INITIAL_MEM	**JAVA_OPTS** に **-Xms** オプションが指定されていない場合に使用されます。これは、初期ヒープメモリーの最大値を算出するために使用されます。コンテナーのメモリー制約のないコンテナーで使用されると、このオプションは効果がありません。メモリー制約がある場合、**-Xms** はここで設定されている値に制限されます。デフォルトは **4096** MB です。つまり、**-Xms** の計算された値は 4096 MB を超えることはありません。この変数の値は、MB で表されます。	**4096**

変数名	説明	値の例
JAVA_MAX_MEM_RATIO	**JAVA_OPTS** に **-Xmx** オプションが指定されていない場合に使用されます。これは、コンテナーの制限に基づいてデフォルトの最大ヒープメモリーを算出するために使用されます。コンテナーのメモリー制約のないコンテナーで使用されると、このオプションは効果がありません。メモリー制約がある場合、**-Xmx** は、ここで設定されているコンテナーの使用可能なメモリーの比率に設定されます。デフォルト値の **50** は、利用可能なメモリーの 50% が上限として使用されることを意味します。このメカニズムを省略するには、この値を **0** に設定します。この場合、**-Xmx** オプションは追加されません。	-
JAVA_OPTS	**java** コマンドに渡される JVM オプション。	**-verbose:class**
JAVA_OPTS_APPEND	**JAVA_OPTS** で生成されたオプションに追加されるユーザー指定の Java オプション。	**-Dsome.property=foo**
LOGGING_SCRIPT_DEBUG	スクリプトのデバッグを有効にするには **true** に設定します。**SCRIPT_DEBUG** を非推奨にします。	**true**
MAVEN_ARGS	Maven の呼び出し時に使用する引数。デフォルトの **package hawt-app:build -DskipTests -e** を置き換えます。**package** 実行フェーズにまだバインドされていない場合は、**hawt-app:build** ゴールを実行してください。実行しない場合は、起動スクリプトは機能しません。	**-e -Popenshift -DskipTests -Dcom.redhat.xpaas.repo.redhatga package**
MAVEN_ARGS_APPEND	追加の Maven 引数。	**-X -am -pl**

変数名	説明	値の例
MAVEN_CLEAR_REPO	設定すると、アーティファクトのビルド後に Maven リポジトリーが削除されます。これは、作成されたアプリケーションイメージを小さく保つのに役立ちますが、インクリメンタルビルドを阻止します。この変数は、**S2I_ENABLE_INCREMENTAL_BUILDS** で上書きされます。デフォルトは **false** です。	-
MAVEN_LOCAL_REPO	ローカルの Maven リポジトリーとして使用するディレクトリー。	**/home/jboss/.m2/repository**
MAVEN_MIRRORS	設定すると、マルチミラーサポートが有効になり、他の **MAVEN_MIRROR_*** 変数に接頭辞が付けられます。たとえば、**DEV_ONE_MAVEN_MIRROR_URL** および **QE_TWO_MAVEN_MIRROR_URL** のようになります。	**dev-one、qe-two**
MAVEN_MIRROR_URL	アーティファクトの取得に使用されるミラーのベース URL。	**http://10.0.0.1:8080/repository/internal/**
MAVEN_REPOS	設定すると、マルチリポジトリーサポートが有効になり、他の **MAVEN_REPO_*** 変数に接頭辞が付けられます。たとえば、**DEV_ONE_MAVEN_REPO_URL** および **QE_TWO_MAVEN_REPO_URL** のようになります。	**dev-one、qe-two**
MAVEN_S2I_ARTIFACT_DIRS	ビルド出力用にスキャンするソースディレクトリーの相対パス。これは **DEPLOY_DIR** にコピーされます。デフォルトは **target** です。	**target**
MAVEN_S2I_GOALS	Maven ビルドで実行されるゴールのスペース区切りリスト。例: **mvn $MAVEN_S2I_GOALS**デフォルトは **package** です。	**package install**
MAVEN_SETTINGS_XML	使用するカスタムの Maven settings.xml ファイルの場所。	**/home/jboss/.m2/settings.xml**

変数名	説明	値の例
NO_PROXY	直接アクセスできるホスト、IP アドレス、またはドメインのコンマ区切りリスト。これは、Maven ビルドと Java ランタイムの両方に使用されます。	**foo.example.com,bar.example.com**
S2I_ARTIFACTS_DIR	アーティファクトのロケーションマウントは、インクリメンタルビルドで使用される **save-artifacts** スクリプトで永続化されます。これはエンドユーザーによって上書きされてはなりません。	**${S2I_DESTINATION_DIR}/artifacts}**
S2I_DESTINATION_DIR	**io.openshift.s2i.destination** ラベルで指定された S2I マウントの root ディレクトリー。これはエンドユーザーによって上書きされてはなりません。	**/tmp**
S2I_ENABLE_INCREMENTAL_BUILDS	将来のビルドで使用するために保存できるように、ソースおよび中間ビルドファイルを削除しないでください。デフォルトは **true** です。	**true**
S2I_IMAGE_SOURCE_MOUNTS	イメージに含める必要があるソースディレクトリー内の相対パスのコンマ区切りリスト。一覧にはワイルドカードを含めることができます。これは find を使用して展開されます。デフォルトでは、マウントされたディレクトリーの内容は、ソースフォルダーと同様に処理されます。ここでは、**S2I_SOURCE_CONFIGURATION_DIR**、**S2I_SOURCE_DATA_DIR**、および **S2I_SOURCE_DEPLOYMENTS_DIR** の内容がそれぞれのターゲットディレクトリーにコピーされます。または、**install.sh** ファイルがマウントポイントのルートにある場合は、代わりに実行されます。**CUSTOM_INSTALL_DIRECTORIES** を非推奨にします。	**extras/***

変数名	説明	値の例
S2I_SOURCE_CONFIGURATION_DIR	製品設定ディレクトリーにコピーされるアプリケーション設定ファイルを含むディレクトリーへの相対パス。**S2I_TARGET_CONFIGURATION_DIR** を参照してください。デフォルトは **configuration** です。	**configuration**
S2I_SOURCE_DATA_DIR	製品データディレクトリーにコピーされるアプリケーションデータファイルを含むディレクトリーへの相対パス。**S2I_TARGET_DATA_DIR** を参照してください。デフォルトは **data** です。	**data**
S2I_SOURCE_DEPLOYMENTS_DIR	製品デプロイメントディレクトリーにコピーされるバイナリーファイルを含むディレクトリーへの相対パス。**S2I_TARGET_DATA_DIR** を参照してください。デフォルトは **deployments** に設定されます。	**デプロイメント**
S2I_SOURCE_DIR	ビルドするソースコードのマウントの場所。これはエンドユーザーによって上書きされてはなりません。	**${S2I_DESTINATION_DIR}/src}**
S2I_TARGET_CONFIGURATION_DIR	**S2I_SOURCE_DIR/S2I_SOURCE_CONFIGURATION_DIR** にあるファイルがコピーされる絶対パス。	**/opt/eap/standalone/configuration**
S2I_TARGET_DATA_DIR	**S2I_SOURCE_DIR/S2I_SOURCE_DATA_DIR** にあるファイルがコピーされる絶対パス。	**/opt/eap/standalone/data**
S2I_TARGET_DEPLOYMENTS_DIR	**S2I_SOURCE_DIR/S2I_SOURCE_DEPLOYMENTS_DIR** にあるファイルがコピーされる絶対パス。また、これはビルド出力がコピーされるディレクトリーです。	**/deployments**

変数名	説明	値の例
http_proxy	http プロキシーの場所。これは **HTTP_PROXY** よりも優先され、Maven ビルドと Java ランタイムの両方に使用されます。	**http://127.0.0.1:8080**
https_proxy	https プロキシーの場所。これは、**HTTPS_PROXY**、**http_proxy**、および **HTTP_PROXY** よりも優先され、Maven ビルドと Java ランタイムの両方に使用されます。	**myuser:mypass@127.0.0.1:8080**
no_proxy	直接アクセスできるホスト、IP アドレス、またはドメインのコンマ区切りリスト。これは **NO_PROXY** よりも優先され、Maven ビルドと Java ランタイムの両方に使用されます。	***.example.com**
prefix_MAVEN_MIRROR_ID	指定されたミラーに使用する ID。省略する場合には、一意の ID が生成されます。	**internal-mirror**
prefix_MAVEN_MIRROR_OF	このエントリーでミラリングされるリポジトリー ID。デフォルト値は **external:** です。	-
prefix_MAVEN_MIRROR_URL	ミラーの URL。	**http://10.0.0.1:8080/repository/internal**
prefix_MAVEN_REPO_DIRECTORY_PERMISSIONS	Maven リポジトリーディレクトリーのパーミッション。	**775**
prefix_MAVEN_REPO_FILE_PERMISSIONS	Maven リポジトリーファイルのパーミッション。	**664**
prefix_MAVEN_REPO_HOST	完全に定義された URL を使用していない場合、Maven リポジトリーホストは、サービスにフォールバックします。	**repo.example.com**
prefix_MAVEN_REPO_ID	Maven リポジトリー ID。	**my-repo-id**
prefix_MAVEN_REPO_LAYOUT	Maven リポジトリーのレイアウト。	**default**
prefix_MAVEN_REPO_NAME	Maven リポジトリー名。	**my-repo-name**

変数名	説明	値の例
prefix_MAVEN_REPO_PASSPHRASE	Maven リポジトリーのパスフレーズ。	**maven1!**
prefix_MAVEN_REPO_PASSWORD	Maven リポジトリーのパスワード。	**maven1!**
prefix_MAVEN_REPO_PATH	完全に定義された URL を使用していない場合、Maven リポジトリーパスは、サービスにフォールバックします。	**/maven2/**
prefix_MAVEN_REPO_PORT	完全に定義された URL を使用していない場合、Maven リポジトリーのポートは、サービスにフォールバックします。	**8080**
prefix_MAVEN_REPO_PRIVATE_KEY	Maven リポジトリーのプライベートキー。	**${user.home}/.ssh/id_dsa**
prefix_MAVEN_REPO_PROTOCOL	完全に定義された URL を使用していない場合、Maven リポジトリーのプロトコルは、サービスにフォールバックします。	**http**
prefix_MAVEN_REPO_RELEASES_CHECKSUM_POLICY	Maven リポジトリーは、チェックサムポリシーをリリースします。	**warn**
prefix_MAVEN_REPO_RELEASES_ENABLED	Maven リポジトリーのリリースが有効。	**true**
prefix_MAVEN_REPO_RELEASES_UPDATE_POLICY	Maven リポジトリーリリース更新ポリシー。	**always**
prefix_MAVEN_REPO_SERVICE	**prefix_MAVEN_REPO_URL** が指定されていない場合に、ルックアップする Maven リポジトリーサービス。	**buscentr-myapp**
prefix_MAVEN_REPO_SNAPSHOTS_CHECKSUM_POLICY	Maven リポジトリースナップショットチェックサムポリシー。	**warn**
prefix_MAVEN_REPO_SNAPSHOTS_ENABLED	Maven リポジトリーのスナップショットが有効。	**true**
prefix_MAVEN_REPO_SNAPSHOTS_UPDATE_POLICY	Maven リポジトリースナップショット更新ポリシー。	**always**

変数名	説明	値の例
prefix_MAVEN_REPO_URL	Maven リポジトリーの完全に定義された URL。	**http://repo.example.com:8080/maven2/**
prefix_MAVEN_REPO_USERNAME	Maven リポジトリーのユーザー名。	**mavenUser**

表2.2 デフォルト値の設定環境変数

変数名	説明	値
AB_JOLOKIA_AUTH_OPENSHIFT	OpenShift TLS 通信のクライアント認証を切り替えます。このパラメーターの値は、提供されるクライアントの証明書に含まれる必要がある相対識別名になります。このパラメーターを有効にすると、Jolokia が自動的に https 通信モードに切り替わります。デフォルトの CA 証明書は **/var/run/secrets/kubernetes.io/serviceaccount/ca.crt** に設定されます。	**true**
AB_JOLOKIA_HTTPS	https との安全な通信をオンにします。デフォルトでは、**AB_JOLOKIA_OPTS** で **serverCert** 設定の指定がないと、自己署名サーバー証明書が生成されます。	**true**
AB_JOLOKIA_PASSWORD_RANDOM	**AB_JOLOKIA_PASSWORD** が無作為に生成されるかどうかを決定します。パスワードを無作為に生成する場合は **true** に設定します。生成された値は **/opt/jolokia/etc/jolokia.pw** に書き込まれます。	**true**
AB_PROMETHEUS_JMX_EXPORTER_CONFIG	Prometheus JMX Exporter に使用する設定へのパス。	**/opt/jboss/container/prometheus/etc/jmx-exporter-config.yaml**
S2I_SOURCE_DEPLOYMENTS_FILTER	デプロイメントのコピー時に適用される、スペースで区切られたフィルターの一覧。デフォルトは ***** に設定されます。	*****

2.2.10. 関連情報

- 追加情報と例は、Red Hat JBoss Middleware のドキュメントを参照してください。

2.3. .NET CORE

2.3.1. .NET Core を使用する利点

.NET Core は、自動メモリー管理と最新のプログラミング言語を備えた汎用開発プラットフォームです。これにより、ユーザーは高品質のアプリケーションを効率的に構築できます。.NET Core は、認定済みのコンテナーを介して Red Hat Enterprise Linux (RHEL 7) および OpenShift Container Platform で利用できます。

- マイクロサービスベースのアプローチに従う機能。一部のコンポーネントは .NET で構築され、他のコンポーネントは Java で構築されますが、すべてが Red Hat Enterprise Linux および OpenShift Container Platform でサポートされている共通プラットフォームで実行できます。
- 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。お客様は、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 内から .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します。

2.3.2. サポート対象バージョン

.NET Core Life Cycle には、現在サポートされている .NET Core のバージョンが記載されています。

2.3.3. イメージ

イメージは、Red Hat レジストリーから入手できます。

標準インストールを実行した場合は、**dotnet** イメージストリームが存在します。サポートされている最新バージョンを追加するには、.NET イメージストリーム を インストール します。

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、コンテナーにそのソースコードを実行用に準備させることにより、すぐに実行できる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概要は、S2I Build Process を参照してください。

2.3.5. 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。これらの変数を設定して、.NET Core アプリケーションのビルド動作を制御できます。

注記

S2I ビルド設定または **.s2i/environment** ファイルで、ビルドの動作を制御する環境変数を設定して、これらを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 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**) 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ある必要があります。	**.**
DOTNET_ASSEMBLY_NAME	実行するアセンブリーを選択します。これには **.dll** 拡張子を **含めない** でください。これを、**csproj** (**PropertyGroup**/**AssemblyName**) で指定した出力アセンブリー名に設定します。	**csproj** ファイルの名前。
DOTNET_RESTORE_SOURCES	復元操作中に使用される **NuGet** パッケージソースのスペース区切り一覧を指定します。これにより、**NuGet.config** ファイルで指定されたすべてのソースが上書きされます。	
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 ツールのリストを指定します。特定バージョンの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または、単一のプロジェクトファイルを含むディレクトリーである必要があります。各項目に対して **dotnet test** が呼び出されます。	

変数名	説明	デフォルト
DOTNET_CONFIGURATION	**Debug** モードまたは **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** のいずれかである必要があります。	**Release**
DOTNET_VERBOSITY	dotnet build コマンドの詳細度を指定します。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。この変数は、msbuild の詳細値 (**q[uiet]**、**m[inimal]**、**n[ormal]**、**d[etailed]**、および **diag[nostic]**) のいずれかに設定できます。	
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カスタム NPM レジストリーミラーを使用してパッケージをダウンロードします。	
ASPNETCORE_URLS	この変数は **http://*:8080** に設定され、イメージにより公開されるポートを使用するように ASP.NET Core を設定します。これを変更することは **推奨されません**。	**http://*:8080**
DOTNET_RM_SRC	**true** に設定すると、ソースコードはイメージに含まれません。	
DOTNET_SSL_DIRS	信頼する追加の SSL 証明書を含むフォルダーと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、ビルド中に実行される各プロセスと、ビルド後にイメージで実行されるすべてのプロセス (ビルドされたアプリケーションを含む) によって信頼されます。項目は、(/ で始まる) 絶対パスまたはソースリポジトリーのパス (証明書など) にすることができます。	
DOTNET_RESTORE_DISABLE_PARALLEL	**true** に設定すると、複数のプロジェクトを並行して復元できなくなります。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**

変数名	説明	デフォルト
DOTNET_INCREMENTAL	**true** に設定すると、NuGet パッケージは保持され、インクリメンタルビルドに再利用できます。	**false**
DOTNET_PACK	**true** に設定すると、パブリッシュされたアプリケーションが含まれる **tar.gz** ファイルが **/opt/app-root/app.tar.gz** に作成されます。	

2.3.6. .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

最初に .NET イメージストリーム を 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** を実行することにより、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 dotnet:3.1~https://github.com/redhat-developer/s2i-dotnetcore-ex#dotnetcore-3.1 --context-dir app
```

2.4. NODE.JS

2.4.1. 概要

OpenShift Container Platform 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 イメージを提供します。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アセンブルし、Node.js アプリケーションを含む新規のイメージを作成します。結果として生成されるイメージは、OpenShift Container Platform またはコンテナーランタイムによって実行できます。

2.4.2. バージョン

現時点で、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。

2.4.3. イメージ

これらのイメージには、ニーズに応じて 2 つのフレーバーがあります

- 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、これらの イメージレジストリー から直接アクセスするか、またはこれらを OpenShift Container Platform コンテナーイメージレジストリー にプッシュします。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で、イメージを参照する イメージストリーム を作成することができます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されるすべての OpenShift Container Platform イメージの イメージストリームの定義例 を見つけることができます。

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、コンテナーにそのソースコードを実行用に準備させることにより、すぐに実行できる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。

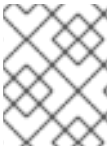
ビルドプロセスの詳細の概要は、S2I Build Process を参照してください。

2.4.5. 設定

Node.js イメージは、Node.js ランタイムの設定と動作を制御するために設定できる多くの環境変数をサポートしています。

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 で定義します。

また、新規アプリケーションを作成する 際、または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すること によって、既存のイメージで使用されるように環境変数を設定することもできます。



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2.4 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。すべての NPM パッケージは、ビルドプロセス時にミラーリンクからダウンロードされます。

2.4.6. 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、**DEV_MODE=true** 環境変数を使用してビルドされた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を作成する 際、または 既存のオブジェクトの環境変数を更新すること によって、新しい環境変数を設定できます。

警告

DEV_MODE=true 環境変数は、開発時またはデバッグ時にのみ使用してください。実稼働環境でこの設定を使用することは推奨されません。

実行中の Pod のソースコードを変更するには、コンテナーへの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。

2.5. PERL

2.5.1. 概要

OpenShift Container Platform 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 イメージを提供します。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アセンブルし、Perl アプリケーションを含む新規のイメージを作成します。結果として生成されるイメージは、OpenShift Container Platform またはコンテナーランタイムによって実行できます。

2.5.2. バージョン

現時点で、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。

2.5.3. イメージ

イメージには、ニーズに応じて 2 つのフレーバーがあります

- 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、これらの イメージレジストリー から直接アクセスするか、またはこれらを OpenShift Container Platform コンテナーイメージレジストリー にプッシュします。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で、イメージを参照する イメージストリーム を作成することができます。その後、OpenShift Container Platform リソースは ImageStream を参照できます。提供されるすべての OpenShift Container Platform イメージの イメージストリームの定義例 を見つけることができます。

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、コンテナーにそのソースコードを実行用に準備させることにより、すぐに実行できる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概要は、S2I Build Process を参照してください。

2.5.5. 設定

Perl イメージは、Perl ランタイムの設定と動作を制御するために設定できる多くの環境変数をサポートしています。

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 で定義します。

また、新規アプリケーションを作成する際、または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すること によって、既存のイメージで使用されるように環境変数を設定することもできます。

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2.5 Perl 環境変数

変数名	説明
ENABLE_CPAN_TEST	**true** に設定すると、この変数はすべての 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 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。
PERL_APACHE2_RELOAD	変更された Perl モジュールの自動リロードを有効にするには、これを **true** に設定します。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 によって処理される同時リクエストの数。デフォルトは 256 ですが、メモリーが制限されている場合は自動的に下げられます。

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ます。このファイルは、**oc rsh** コマンドを使用してコンテナーにアクセスすることで表示できます。

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。このイメージでホットデプロイメントを有効にするには、**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のみ使用してください。実稼働環境でこの設定をオンにすることは推奨していません。

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**に設定されます。

2.6. PHP

2.6.1. 概要

OpenShift Container Platform 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 イメージを提供します。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アセンブルし、PHP アプリケーションを含む新規のイメージを作成します。結果として生成されるイメージは、OpenShift Container Platform またはコンテナーランタイムによって実行できます。

2.6.2. 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。

2.6.3. イメージ

これらのイメージには、ニーズに応じて 2 つのフレーバーがあります

- 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、これらの イメージレジストリー から直接アクセスするか、またはこれらを OpenShift Container Platform コンテナーイメージレジストリー にプッシュします。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で、イメージを参照する イメージストリーム を作成することができます。その後、OpenShift Container Platform リソースがイメージストリームを参照できます。

提供されるすべての OpenShift Container Platform イメージの イメージストリームの定義例 を見つけることができます。

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、コンテナーにそのソースコードを実行用に準備させることにより、すぐに実行できる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概要は、S2I Build Process を参照してください。

2.6.5. 設定

PHP イメージは、PHP ランタイムの設定と動作を制御するために設定できる多くの環境変数をサポートしています。

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 で定義します。

また、新規アプリケーションを作成する 際、または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すること によって、既存のイメージで使用されるように環境変数を設定することもできます。

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2.6 PHP 環境変数

変数名	説明	デフォルト
ERROR_REPORTING	アクションを実行するエラー、警告、および通知を PHP に通知します。	E_ALL & ~E_NOTICE

変数名	説明	デフォルト
DISPLAY_ERRORS	PHP がエラー、通知、および警告を出力するかどうか、および出力する場所を制御します。	ON
DISPLAY_STARTUP_ERRORS	PHP の起動シーケンス中に発生する表示エラーは、表示エラーとは別に処理されます。	OFF
TRACK_ERRORS	最後のエラー/警告メッセージを **$php_errormsg** (ブール値) に保存します。	OFF
HTML_ERRORS	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 root を定義するパス (例: /public)	/

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

表2.7 追加の PHP 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チェックする頻度 (秒単位)。0 の場合、OPcache はすべてのリクエストで更新をチェックします。	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2.8 追加の PHP 設定

変数名	説明
PHPRC	php.ini ファイルへのパスを設定。

変数名	説明
PHP_INI_SCAN_DIR	追加の .ini 設定ファイルをスキャンするパス。

カスタムの composer リポジトリーミラー URL を使用して、デフォルトの 'packagist.org' の代わりにパッケージをダウンロードできます。

表2.9 Composer 環境変数

変数名	説明
COMPOSER_MIRROR	ビルドプロセス中に必要なパッケージをダウンロードするために、カスタム Composer リポジトリーミラー URL を使用するようにこの変数を設定します。注記: これは、**composer.json** に記載されているパッケージにのみ影響します。

2.6.5.1. Apache 設定

アプリケーションの **DocumentRoot** がソースディレクトリー **/opt/openshift/src** にネストされている場合は、独自の **.htaccess** ファイルを指定して、デフォルトの Apache の動作を上書きし、アプリケーションの要求の処理方法を指定できます。**.htaccess** ファイルは、アプリケーションソースの root に置く必要があります。

2.6.6. 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ます。このファイルは、**oc rsh** コマンドを使用してコンテナーにアクセスすることで表示できます。

2.6.7. 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、**OPCACHE_REVALIDATE_FREQ=0** 環境変数を使用して、ビルドされた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のみ使用してください。実稼働環境でこの設定をオンにすることは推奨していません。

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**に設定されます。

2.7. PYTHON

2.7.1. 概要

OpenShift Container Platform 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 イメージを提供します。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アセンブルし、Python アプリケーションを含む新規のイメージを作成します。結果として生成されるイメージは、OpenShift Container Platform またはコンテナーランタイムによって実行できます。

2.7.2. 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。

2.7.3. イメージ

これらのイメージには、ニーズに応じて 2 つのフレーバーがあります

- 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、これらの イメージレジストリー から直接アクセスするか、またはこれらを OpenShift Container Platform コンテナーイメージレジストリー にプッシュします。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で、イメージを参照する イメージストリーム を作成することができます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されるすべての OpenShift Container Platform イメージの イメージストリームの定義例 を見つけることができます。

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、コンテナーにそのソースコードを実行用に準備させることにより、すぐに実行できる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概要は、S2I Build Process を参照してください。

2.7.5. 設定

Python イメージは多数の環境変数をサポートし、環境変数を設定することで Python 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 で定義します。

また、新規アプリケーションを作成する 際、または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すること によって、既存のイメージで使用されるように環境変数を設定することもできます。

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2.10 Python 環境変数

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。この変数は、デフォルトで **app.py** に設定されます。
APP_MODULE	この変数は WSGI callable を指定します。これは、**$(MODULE_NAME):$(VARIABLE_NAME)** のパターンに準拠し、モジュール名はドットのフルパスで、変数名は指定された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** を使用する場合は、モジュール名を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。利用可能な setup-test-app のサンプルがあります。
APP_CONFIG	この変数は、gunicorn 設定 を使用して、有効な Python ファイルへのパスを指定します。

変数名	説明
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。Django プロジェクトにのみ影響します。
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。Django プロジェクトにのみ影響します。
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します。
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数 4 に設定されます。

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。Django を使用する場合、ホットデプロイメントはそのままで機能します。

Gunicorn を使用しながらホットデプロイメントを有効にするには、**リロード オプション** を **true** に設定して、リポジトリー内に Gunicorn 設定ファイルがあることを確認します。**APP_CONFIG** 環境変数を使用して、設定ファイルを指定します。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のみ使用してください。実稼働環境でこの設定をオンにすることは推奨していません。

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**に設定されます。

2.8. RUBY

2.8.1. 概要

OpenShift Container Platform 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

イメージを提供します。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アセンブルし、Ruby アプリケーションを含む新規のイメージを作成します。結果として生成されるイメージは、OpenShift Container Platform またはコンテナーランタイムによって実行できます。

2.8.2. 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。

2.8.3. イメージ

これらのイメージには、ニーズに応じて 2 つのフレーバーがあります

- 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、これらの イメージレジストリー から直接アクセスするか、またはこれらを OpenShift Container Platform コンテナーイメージレジストリー にプッシュします。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で、イメージを参照する イメージストリーム を作成することができます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されるすべての OpenShift Container Platform イメージの イメージストリームの定義例 を見つけることができます。

2.8.4. 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、コンテナーにそのソースコードを実行用に準備させることにより、すぐに実行できる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概要は、S2I Build Process を参照してください。

2.8.5. 設定

Ruby イメージは多数の環境変数をサポートし、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 で定義します。

また、新規アプリケーションを作成する 際、または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すること によって、既存のイメージで使用されるように環境変数を設定することもできます。

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2.11 Ruby 環境変数

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 (**production**、**development**、または **test**) を指定します。各レベルには、ロギングの詳細度、エラーページ、および ruby gem のインストールに関して異なる動作があります。アプリケーションアセットは、**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 (**production**、**development**、または **test**) を指定します。各レベルには、ロギングの詳細度、エラーページ、および ruby gem のインストールに関して異なる動作があります。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。この変数は、デフォルトで **${RACK_ENV}** に設定されます。
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。したがって、アセットがすでにコンパイルされている場合は、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示します。

変数名	説明
PUMA_WORKERS	この変数は、Puma の クラスター化されたモード 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、デフォルトの動作は **PUMA_WORKERS** を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します。
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。注記: この環境変数は、Ruby 2.2+ イメージでのみ利用可能です。

2.8.6. 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って異なります。

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、実行中の Pod に渡された **RAILS_ENV=development** 環境変数を使用して、ビルドされた 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の場合は、 **oc set env** コマンドを使用できます。

```
$ 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。これらの gem には以下が含まれます。

- 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。

お使いのバージョンの S2I **run** スクリプト でアプリケーションイメージをビルドした後に、**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のみ使用してください。実稼働環境でこの設定をオンにすることは推奨していません。

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**に設定されます。

2.9. S2I イメージのカスタマイズ

2.9.1. 概要

S2I ビルダーイメージには通常、**assemble** および **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、すべてのユーザーに対して適切ではない場合があります。このトピックでは、デフォルトのスクリプトを含む S2I ビルダーの動作をカスタマイズする方法をいくつか取り上げます。

2.9.2. イメージに埋め込まれたスクリプトの呼び出し

通常、ビルダーイメージは、最も一般的なユースケースを含む、独自のバージョンの S2I スクリプトを提供します。これらのスクリプトで各自のニーズが満たされない場合、S2I 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する方法を提供します。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。スクリプトの置き換えは許容できる場合もありますが、場合によっては、イメージに含まれるスクリプトのロジックを保持しつつ、スクリプトの前後にコマンドをいくつか実行することをお勧めします。このような場合には、カスタムロジックを実行し、イメージ内のデフォルトのスクリプトに追加の作業を委任するラッパースクリプトを作成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別するには、**io.openshift.s2i.scripts-url** ラベルの値を確認します。以下のように **docker inspect** を使用します。

```
$ 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検査し、スクリプトが **/usr/libexec/s2i** ディレクトリーにあることを確認できます。

これに基づいて、呼び出しをラップして、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 run 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

このトピックグループには、OpenShift Container Platform ユーザーが利用でき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ています。

注記

データベースイメージのクラスターリングを有効にする設定はサンプルとして提供され、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 は、MySQL を実行するためのコンテナーイメージを提供します。このイメージは、設定を介して提供されるユーザー名、パスワード、および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を提供できます。

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

このイメージには、ニーズに応じて 2 つのフレーバーがあります

- 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、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュできます。さらに、コンテナーイメージレジストリーまたは外部の場所に、対象イメージを参照する ImageStream を作成するこ

ともできます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されているすべての OpenShift Container Platform イメージの サンプルImageStream 定義を確認することができます。

3.2.4. 設定と使用方法

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** 環境変数を指定する場合) が作成されます。その後、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必要がある場合は、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 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 への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 **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。コンテナー内からのローカル接続は常にパスワードなしで許可されます。
MYSQL_SERVICE_HOST	Kubernetes が自動作成したサービスホスト変数。
MYSQL_SERVICE_PORT	Kubernetes が自動作成したサービスポート変数。

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。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	テーブル名の保存方法と比較方法を設定します。	0
MYSQL_MAX_CONNECTIONS	同時クライアント接続の最大許容数	151
MYSQL_MAX_ALLOWED_PACKET	1 つのパケットまたは生成/中間文字列の最大サイズ。	200M

変数名	説明	デフォルト
MYSQL_FT_MIN_WORD_LEN	FULLTEXT インデックスに含まれる単語の最小長。	4
MYSQL_FT_MAX_WORD_LEN	FULLTEXT インデックスに含まれる単語の最大長。	20
MYSQL_AIO	ネイティブ AIO が破損した場合に **innodb_use_native_aio** 設定値を制御します。	1
MYSQL_TABLE_OPEN_CACHE	すべてのスレッドのオープンテーブルの数	400
MYSQL_KEY_BUFFER_SIZE	インデックスブロックに使用されるバッファーのサイズ	32m (または利用可能なメモリーの 10%)
MYSQL_SORT_BUFFER_SIZE	ソートに使用されるバッファーサイズ	256K
MYSQL_READ_BUFFER_SIZE	連続スキャンに使用されるバッファーのサイズ	8M (または利用可能なメモリーの 5%)
MYSQL_INNODB_BUFFER_POOL_SIZE	InnoDB がテーブルおよび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	ロググループ内の各ログファイルのサイズ	8M (または使用可能なメモリーの 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のサイズ	8M (または使用可能なメモリーの 15%)

一部のメモリー関連パラメーターには、2 つのデフォルト値があります。コンテナーに メモリー制限 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に基づいて動的に計算されます。

3.2.4.4. ボリュームマウントポイント

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**) および **root** ユーザーのパスワードを変更するためにサポートされる唯一の方法は、環境変数 **MYSQL_PASSWORD** および **MYSQL_ROOT_PASSWORD** をそれぞれ変更することです。

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 で環境変数を一覧表示して表示できます。

```
$ 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、この設定が解除されると、**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、常にリモートアクセスのある一般ユーザー **MYSQL_USER** には影響を与えません。これは、**localhost**にパスワードなしでいつでもログインできる**root**ユーザーによるローカルアクセスにも影響しません。

SQL ステートメントまたは前述の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、変数に格納されている値と実際のパスワードが一致しなくなります。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の 1 つまたは両方を更新します。たとえば、テンプレートから作成されたアプリケーションの場合など、複数のデプロイメント設定でこれらの環境変数を使用する場合は、パスワードがすべての場所で同期されるように、各デプロイメント設定の変数を更新する必要があります。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部分にパスワード用の他の環境変数があり、それらも一致するように更新する必要があります。たとえば、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フロントエンド Pod に、より一般的な **DATABASE_USER** 変数が存在する可能性があります。アプリケーションごとに必要なすべての環境変数に対してパスワードが同期していることを確認してください。同期していないと、トリガーされたときに Pod の再デプロイに失敗する可能性があります。

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。それ以外の場合には、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。

新規の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 への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 は、新しいデータベースサービスの作成を容易にするための テンプレート を提供します。テンプレートは、パスワード値の自動生成を含む事前定義されたデフォルトで、すべての必須環境変数 (ユーザー、パスワード、データベース名など) を定義するためのパラメーターフィールドを提供します。また、デプロイメント設定 および サービス の両方も定義します。

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って、デフォルトの **openshift** プロジェクトに登録されている必要があります。詳細は、必要に応じて、Loading the Default Image Streams and Templates を参照してください。

利用可能なテンプレートは 2 つあります。

- **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 が別のノードに移動されたり、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、データがすべて失われることを意味します。

- **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には、OpenShift Container Platform デプロイメントで永続ボリュームプールを定義する必要があります。プールの設定に関するクラスター管理者の手順は、Persistent Storage Using NFS にあります。

これらの 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。そのコンポーネントは、定義されたサービスを介してデータベースにアクセスできます。

3.2.6. MySQL レプリケーションの使用

注記

データベースイメージのクラスターリングを有効にする設定はサンプルとして提供され、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 (クラスターリング) 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 を取得できます。

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、サンプルテンプレートで定義されているオブジェクトの詳細と、それらがどのように連携してマスター/スレーブのレプリケーションを実装する MySQL サーバーのクラスターを開始するかについて説明します。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、レプリケーションコントローラー を定義するサンプルテンプレートで デプロイメント設定 を定義します。MySQL のマスター/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** サーバーを定義し、2 つ目は 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** フィールドを **run-mysqld-master** に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 レプリケーションには、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。この目的のために、次の環境変数がテンプレートで定義されています。

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"
```

この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すべての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、**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ます。これにより、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 (multi-master) 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をスケーリングできません。

MySQL マスター で作成されたデータベースを複製するには、デプロイメント設定がテンプレートで定義されます。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省略し、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。このサービスも、サンプルテンプレートで定義されます。

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。このレプリケーションコントローラー は、アカウントのリソース容量まで、両方向に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、レプリケーション用に登録するために、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** という名前のヘッドレスサービスを定義します。このサービスは、レプリケーションのみに使用されるのではなく、クライアントは MySQL ホストとして **mysql-master:3306** にクエリーも送信できます。

ヘッドレスサービスを含めるには、サービス定義の **clusterIP** パラメーターを **None** に設定します。次に、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 IP アドレスの一覧を取得できます。

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"
```

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

3.2.6.3. MySQL スレーブのスケーリング

クラスター内の 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>
```

これは、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** サービスへの問い合わせを試み、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、スレーブ上のすべての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これを削除します。

3.2.7. トラブルシューティング

本セクションでは、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 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。後続のデプロイメントでは、この設定により MySQL 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り、次のノードが再起動するまで続きます。

3.3. POSTGRESQL

3.3.1. 概要

OpenShift Container Platform には、PostgreSQL 実行用のコンテナーイメージを提供します。このイメージは、設定を介して提供されるユーザー名、パスワード、および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を提供できます。

3.3.2. バージョン

現時点で、OpenShift Container Platform は PostgreSQL のバージョン 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、ニーズに応じて 2 つのフレーバーがあります

- 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、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュできます。さらに、コンテナーイメージレジストリーまたは外部の場所に、対象イメージを参照する ImageStream を作成するこ

ともできます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されているすべての OpenShift Container Platform イメージの サンプルImageStream 定義を確認することができます。

3.3.4. 設定と使用方法

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は、データベース、データベースの管理ユーザー、PostgreSQL postgres ユーザー (**POSTGRESQL_ADMIN_PASSWORD** 環境変数を指定する場合) が作成されます。その後、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は、データベース、データベースユーザー、管理ユーザーは作成されず、PostgreSQL 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 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 をインストールし、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必要がある場合は、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 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 への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 操作を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	作成される PostgreSQL アカウントのユーザー名このユーザーには、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のパスワード。これが設定されていない場合は、**postgres** アカウントにリモートからログインできません。コンテナー内からのローカル接続は常にパスワードなしで許可されます。

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。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	許可されるクライアント接続の最大数。	100
POSTGRESQL_MAX_PREPARED_TRANSACTIONS	"prepared" 状態にあるトランザクションの最大数を設定します。prepared 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の PostgreSQL 専用のメモリー量。	32M
POSTGRESQL_EFFECTIVE_CACHE_SIZE	オペレーティングシステム別、および PostgreSQL 自体で、ディスクキャッシュに利用可能なメモリー量の推定。	128M

変数名	説明	デフォルト

3.3.4.4. ボリュームマウントポイント

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** 管理ユーザーのパスワードを変更するためにサポートされている唯一の方法は、環境変数 **POSTGRESQL_PASSWORD** および **POSTGRESQL_ADMIN_PASSWORD** をそれぞれ変更することです。

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 で環境変数を一覧表示して表示できます。

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

SQL ステートメントまたは前述の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、変数に格納されている値と実際のパスワードが一致しなくなります。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の 1 つまたは両方を更新します。たとえば、テンプレートから作成されたアプリケーションの場合など、複数のデプロイメント設定でこれらの環境変数を使用する場合は、パスワードがすべての場所で同期されるように、各デプロイメント設定の変数を更新する必要があります。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部分にパスワード用の他の環境変数があり、それらも一致するように更新する必要があります。たとえば、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フロントエンド Pod に、より一般的な **DATABASE_USER** 変数が存在する可能性があります。アプリケーションごとに必要なすべての環境変数に対してパスワードが同期していることを確認してください。同期していないと、トリガーされたときに Pod の再デプロイに失敗する可能性があります。

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。それ以外の場合には、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。

新規の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 への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** 管理ユーザーの新規の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 は、新しいデータベースサービスの作成を容易にするための テンプレート を提供します。テンプレートは、パスワード値の自動生成を含む事前定義されたデフォルトで、すべての必須環境変数 (ユーザー、パスワード、データベース名など) を定義するためのパラメーターフィールドを提供します。また、デプロイメント設定 および サービス の両方も定義します。

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って、デフォルトの **openshift** プロジェクトに登録されている必要があります。詳細は、必要に応じて、Loading the Default Image Streams and Templates を参照してください。

利用可能なテンプレートは 2 つあります。

- **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 が別のノードに移動されたり、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、データがすべて失われることを意味します。

- **PostgreSQL-persistent** は、データベース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には、OpenShift Container Platform デプロイメントで永続ボリュームプールを定義する必要があります。プールの設定に関するクラスター管理者の手順は、Persistent Storage Using NFS にあります。

これらの 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。そのコンポーネントは、定義されたサービスを介して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 は、MongoDB を実行するためのコンテナーイメージを提供します。このイメージは、設定を介して提供されるユーザー名、パスワード、および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を提供できます。

3.4.2. バージョン

現時点で、OpenShift Container Platform 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、ニーズに応じて 2 つのフレーバーがあります

- 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、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュできます。さらに、コンテナーイメージレジストリーまたは外部の場所に、対象イメージを参照する ImageStream を作成することもできます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されているすべての OpenShift Container Platform イメージの サンプルImageStream 定義を確認することができます。

3.4.4. 設定と使用方法

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ます。その後、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、管理ユーザーは作成されず、MongoDB 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必要がある場合は、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 への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** 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	作成される 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。4 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。

注記

管理ユーザー名は admin に設定され、**MONGODB_ADMIN_PASSWORD** 環境変数を設定してパスワードを指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 がより小さなデフォルトのデータファイルサイズを使用するように設定します。	**true**
MONGODB_QUIET	出力量の制限を試みる quiet モードで MongoDB を実行します。	**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おり、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**) および **admin** ユーザーのパスワードを変更するためにサポートされている唯一の方法は、環境変数 **MONGODB_PASSWORD** と **MONGODB_ADMIN_PASSWORD** をそれぞれ変更することです。

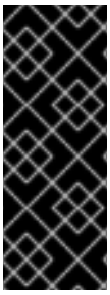
現在のパスワードは、Pod またはデプロイメント設定を Web コンソールで表示するか、CLI で環境変数を一覧表示して表示できます。

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

MongoDB で直接データベース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の 1 つまたは両方を更新します。たとえば、テンプレートから作成されたアプリケーションの場合など、複数のデプロイメント設定でこれらの環境変数を使用する場合は、パスワードがすべての場所で同期されるように、各デプロイメント設定の変数を更新する必要があります。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部分にパスワード用の他の環境変数があり、それらも一致するように更新する必要があります。たとえば、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フロントエンド Pod に、より一般的な **DATABASE_USER** 変数が存在する可能性があります。アプリケーションごとに必要なすべての環境変数に対してパスワードが同期していることを確認してください。同期していないと、トリガーされたときに Pod の再デプロイに失敗する可能性があります。

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。それ以外の場合には、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。

新規の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 への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 は、新しいデータベースサービスの作成を容易にするための テンプレート を提供します。テンプレートは、パスワード値の自動生成を含む事前定義されたデフォルトで、すべての必須環境変数 (ユーザー、パスワード、データベース名など) を定義するためのパラメーターフィールドを提供します。また、デプロイメント設定 および サービス の両方も定義します。

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って、デフォルトの **openshift** プロジェクトに登録されている必要があります。詳細は、必要に応じて、Loading the Default Image Streams and Templates を参照してください。

利用可能なテンプレートは 2 つあります。

- **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 が別のノードに移動されたり、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、データがすべて失われることを意味します。

- **mongodb-persistent** は、データベースデータ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には、OpenShift Container Platform デプロイメントで永続ボリュームプールを定義する必要があります。プールの設定に関するクラスター管理者の手順は、Persistent Storage Using NFS にあります。

これらの 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。そのコンポーネントは、定義されたサービスを介して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

データベースイメージのクラスターリングを有効にする設定はサンプルとして提供され、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 (クラスターリング) 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 を取得できます。

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 つ以上がクラッシュまたは失敗すると、レプリカセットメンバーは再起動されます。

レプリカセットメンバーがダウンまたは再起動している間は、以下のいずれかのシナリオになります。

1. プライマリーメンバーがダウンしている。
   この場合、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。それまでは、読み取りには影響はありませんが、書き込みは失敗します。正常に選択された後は、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は、3 つ目のメンバーがレプリカセットへの再度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. 任意の 2 つのメンバーがダウンしている。
   3 つのメンバーで設定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。この場合、読み取りはセカンダリーメンバーが行う可能性があり、書き込みは失敗します。もう 1 つのメンバーが再度起動したらすぐに、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. すべてのメンバーがダウンしている。
   このように極端なケースでは、読み取りおよび書き込みの両方が失敗します。2 つ以上のメンバーが再度起動すると、レプリカセット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。

これは MongoDB に推奨されるレプリケーションストラテジーです。

注記

実稼働環境では、できるだけメンバー間の分離を維持する必要があります。StatefulSet Pod を異なるノードにスケジュールするために 1 つ以上のノード選択機能を使用し、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が推奨されます。

3.4.6.1. 制限

- MongoDB 3.2 のみがサポートされます。
- スケールダウンの場合は、レプリカセットの設定を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。以下が必要です。
  - StatefulSet 設定の環境変数の値の更新
  - データベースのパスワードの変更
  - すべての Pod を次々に再起動します。

3.4.6.2. サンプルテンプレートの使用

事前に作成されている永続ボリュームが既に 3 つあるか、または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しい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
```

このコマンドにより、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 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になります。Pod が実行されたら、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>
```

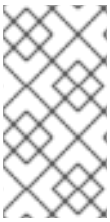
3.4.6.3. スケールアップ

MongoDB は、レプリカセット内のメンバーの数を奇数にすることを推奨しています。利用可能な永続ボリュームが十分に存在するか、動的ストレージプロビジョナーがある場合には、**oc scale** コマンド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、レプリカセットに接続し、その設定を更新する新規の Pod が作成されます。



注記

データベースのサイズが **oplogSize** 設定よりも大きい場合は、既存のデータベースのスケールアップは手動で実行する必要があります。このような場合は、新規メンバーの初回同期を手動で行う必要があります。詳細は、Check the Size of the Oplog および MongoDB Replication の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 (ストレージの可用性、既存のデータベースのサイズ、**oplogSize**) を満たす場合には、スケールアップを手動で実行することはありませんが、スケールダウンは常に手動で実行する必要があります。

スケールダウンの方法:

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、削除された Pod の 1 つにプライマリーメンバーロールがあると、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ことがあります。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。

   または、小さい数値にスケールダウンすると、レプリカセットが一時的にセカンダリーメンバーのみを持ち、読み取り専用モードになります。たとえば、メンバーを 5 から 1 のみ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、将来改善される可能性があります。可能な実装は、(ダウンワード API を介して公開される) レプリカの数を検査し、Pod が StatefulSet から削除されていることを判別する **PreStop** Pod フックが設定されていて、他の理由で再起動されていないことです。

3. 廃止された Pod で使用されたボリュームを削除する。

3.5. MARIADB

3.5.1. 概要

OpenShift Container Platform は、MariaDB を実行するためのコンテナーイメージを提供します。このイメージは、設定ファイルで提供されるユーザー名、パスワード、および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を提供できます。

3.5.2. バージョン

現時点で、OpenShift Container Platform は MariaDB のバージョン 10.0 および 10.1 を提供しています。

3.5.3. イメージ

これらのイメージには、ニーズに応じて 2 つのフレーバーがあります

- 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、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュできます。さらに、コンテナーイメージレジストリーまたは外部の場所に、対象イメージを参照する ImageStream を作成することもできます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されているすべての OpenShift Container Platform イメージの サンプルImageStream 定義を確認することができます。

3.5.4. 設定と使用方法

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** 環境変数を指定する場合) が作成されます。その後、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、管理ユーザーは作成されず、MariaDB 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 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 をインストールし、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必要がある場合は、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 への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 **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	作成される 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
MYSQL_DATABASE	データベース名

変数名	説明
MYSQL_ROOT_PASSWORD	root ユーザーのパスワード (任意)

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。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	テーブル名の保存方法と比較方法を設定します。	0
MYSQL_MAX_CONNECTIONS	同時クライアント接続の最大許容数	151
MYSQL_MAX_ALLOWED_PACKET	1 つのパケットまたは生成/中間文字列の最大サイズ。	200M
MYSQL_FT_MIN_WORD_LEN	FULLTEXT インデックスに含まれる単語の最小長。	4
MYSQL_FT_MAX_WORD_LEN	FULLTEXT インデックスに含まれる単語の最大長。	20
MYSQL_AIO	ネイティブ AIO が破損した場合に **innodb_use_native_aio** 設定値を制御します。	1
MYSQL_TABLE_OPEN_CACHE	すべてのスレッドのオープンテーブルの数	400
MYSQL_KEY_BUFFER_SIZE	インデックスブロックに使用されるバッファーのサイズ	32M (または利用可能なメモリーの 10%)

変数名	説明	デフォルト
MYSQL_SORT_BUFFER_SIZE	ソートに使用されるバッファーサイズ	256K
MYSQL_READ_BUFFER_SIZE	連続スキャンに使用されるバッファーのサイズ	8M (または利用可能メモリーの5%)
MYSQL_INNODB_BUFFER_POOL_SIZE	InnoDB がテーブルおよび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	ロググループ内の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のサイズ	8M (または利用可能メモリーの15%)
MYSQL_DEFAULTS_FILE	代替の設定ファイルを参照します。	/etc/my.cnf
MYSQL_BINLOG_FORMAT	set は、binlog 形式を設定します。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です。

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は、マウントされ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**) および **admin** ユーザーのパスワードを変更するためにサポートされる唯一の方法は、環境変数 **MYSQL_PASSWORD** および **MYSQL_ROOT_PASSWORD** をそれぞれ変更することです。

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 で環境変数を一覧表示して表示できます。

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

SQL ステートメントまたは前述の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、変数に格納されている値と実際のパスワードが一致しなくなります。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の 1 つまたは両方を更新します。たとえば、テンプレートから作成されたアプリケーションの場合など、複数のデプロイメント設定でこれらの環境変数を使用する場合は、パスワードがすべての場所で同期されるように、各デプロイメント設定の変数を更新する必要があります。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部分にパスワード用の他の環境変数があり、それらも一致するように更新する必要があります。たとえば、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フロントエンド Pod に、より一般的な **DATABASE_USER** 変数が存在する可能性があります。アプリケーションごとに必要なすべての環境変数に対してパスワードが同期していることを確認してください。同期していないと、トリガーされたときに Pod の再デプロイに失敗する可能性があります。

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。それ以外の場合には、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 |
+------------+------------------+
```

3.5.5. 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 は、新しいデータベースサービスの作成を容易にするための テンプレート を提供します。テンプレートは、パスワード値の自動生成を含む事前定義されたデフォルトで、すべての必須環境変数 (ユーザー、パスワード、データベース名など) を定義するためのパラメーターフィールドを提供します。また、デプロイメント設定 および サービス の両方も定義します。

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って、デフォルトの **openshift** プロジェクトに登録されている必要があります。詳細は、必要に応じて、Loading the Default Image Streams and Templates を参照してください。

利用可能なテンプレートは 2 つあります。

- **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 が別のノードに移動されたり、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、データがすべて失われることを意味します。

- **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には、OpenShift Container Platform デプロイメントで永続ボリュームプールを定義する必要があります。プールの設定に関するクラスター管理者の手順は、Persistent Storage Using NFS にあります。

これらの 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。そのコンポーネントは、定義されたサービスを使用して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

本セクションでは、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 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。後続のデプロイメントでは、この設定により MySQL 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り、次のノードが再起動するまで続きます。

第4章 その他のイメージ

4.1. 概要

このトピックグループには、OpenShift ContainerPlatform ユーザーが利用できる他のコンテナーイメージに関する情報が含まれています。

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 ジョブも含まれています。これは、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て、ビルドが成功すると出力にタグを付け直し、ビルドの実稼働環境の準備ができたことを示します。詳細は、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 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、現在は維持されていません。

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、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュできます。さらに、コンテナーイメージレジストリーまたは外部の場所に、対象イメージを参照する ImageStream を作成することもできます。その後、OpenShift Container Platform リソースが ImageStream を参照できます。提供されているすべての OpenShift Container Platform イメージの サンプルImageStream 定義を確認することができます。

4.2.3. 設定とカスタマイズ

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

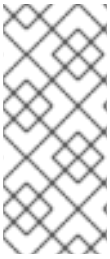
OAuth 認証 は、Jenkins UI の **Configure Global Security** パネルを設定するか、Jenkins **Deployment Config** の **OPENSHIFT_ENABLE_OAUTH** 環境変数を **false** 以外に設定して、有効にします。これにより、OpenShift ログインプラグインが有効になり、Pod データから、または OpenShift Container Platform API サーバーと対話して、設定情報を取得します。

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、空でない文字列を指定できます。

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

有効なユーザーは、ログイン時に自動的に Jenkins 承認マトリックスに追加されます。ここで、OpenShift Container Platform **Roles** は、ユーザーに割り当てられる固有の Jenkins パーミッションを指定します。

admin ロールを持つユーザーには、従来の Jenkins 管理ユーザーパーミッションがあります。 **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。OpenShift ロールや Jenkins パーミッションのマッピングに関する詳細は、Jenkins image source repository README を参照してください。



注記

OpenShift Container Platform クラスター管理者が OpenShift Container Platform ID プロバイダーでそのユーザーを明示的に定義し、**admin** ロールをユーザーに割り当てない限り、OpenShift Container Platform OAuth が使用される場合、管理者特権を持つ OpenShift Container Platform Jenkins イメージに事前入力された **admin** ユーザーには、それらの特権は与えられません。

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

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

OAuth 認証を使用して新しい Jenkins サービスを作成する最も簡単な方法は、以下で説明するように テンプレートを使用 することです。

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情報は、**admin** および **password** です。標準の Jenkins 認証を使用する場合のみ、**JENKINS_PASSWORD** 環境変数を設定して、デフォルトのパスワードを設定します。

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 認証を使用する際の **admin** ユーザーのパスワード。**OPENSHIFT_ENABLE_OAUTH** が **true** に設定されている場合には該当しません。

- **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。

- **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。それ以外の場合は、最大ヒープサイズはコンテナーメモリー制限の **CONTAINER_HEAP_PERCENT**% (設定例: **0.5**、または 50%) として動的に計算され、オプションで **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB (設定例: **512**) に制限されます。

  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: **-Xms32m**)、この値が優先されます。それ以外の場合は、初期ヒープサイズは、動的に計算された最大ヒープサイズの **CONTAINER_INITIAL_PERCENT**% (設定例: **0.1**、または 10%) として動的に計算される場合があります。

  デフォルトでは、最初のヒープサイズは JVM に任されています。

- **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**)
  こ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に、インストールする追加の Jenkins プラグインを指定します (以下を参照)。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 で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用に 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、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になります。このイメージを拡張し、初回起動後にカスタムイメージの設定を更新するカスタムイメージを作成する場合、この環境変数を **true** に設定しない限り、デフォルトでこれはコピーされません。

- **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、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になります。このイメージを拡張し、初回起動後にカスタムイメージのプラグインを更新するカスタムイメージを作成する場合、この環境変数を **true** に設定しない限り、デフォルトでこれらはコピーされません。

- **ENABLE_FATAL_ERROR_LOG_FILE** (デフォルト: **false**)
  Jenkins 設定ディレクトリー用に OpenShift Container Platform の永続ボリューム要求を使用してこのイメージを実行する場合、この環境変数は、致命的なエラーが生じる際に、致命的なエラーのログファイルが永続することを可能にします。致命的なエラーのファイルは **/var/lib/jenkins/logs** に保存されます。

- **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 バージョンのどちらを実行しているかに応じて、**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 バージョンのどちらを実行しているかに応じて、**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。

- **JENKINS_UC_INSECURE**
  Jenkins Update Center リポジトリーが無効な SSL 証明書を使用する場合に、Jenkins プラグインのダウンロードが許可されるかどうかを決定します。これは、不明な CA を持つ自己署名証明書を使用するセルフホストリポジトリーが使用されるか、またはエンタープライズプロキシーが中間者傍受を実行する場合に、当てはまる可能性があります。この変数は、Jenkins イメージのビルド時に発生する可能性のあるプラグインのダウンロードや、Jenkins イメージの拡張機能がビルドされる場合に適用されます。また、Jenkins イメージを実行し、いずれかのオプションを使用して追加のプラグイン (plugins.txt を含む S2I または INSTALL_PLUGINS 環境変数) をダウンロードするときにも適用されます。この変数を有効にするには true に設定します。

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 がアクセスする必要のあるプロジェクトにアクセスするための適切な権限を持つ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
```

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

2. シークレットからトークンを取得します。

```
$ oc describe secret <secret name from above> # 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なトークンの値が含まれます。

4.2.3.4. ボリュームマウントポイント

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

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

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

正式な OpenShift Container Platform Jenkins イメージをカスタマイズするには、以下の 2 つのオプションがあります。

- Docker の階層化を使用します。
- ここで説明するように、イメージを Source-To-Image ビルダーとして使用します。

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

Jenkins イメージに変更を追加するには、以下のディレクトリー構造を持つ 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一覧表示します。

```
pluginId:pluginVersion
```

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

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

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

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

```
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:                    (1)
    git:
      uri: https://github.com/custom/repository
    type: Git
  strategy:                  (2)
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jenkins:latest
        namespace: openshift
    type: Source
  output:                    (3)
```

```
to:
  kind: ImageStreamTag
  name: custom-jenkins:latest
```

1. **source** フィールドは、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 が含まれ、Kubernetes および OpenShift Container Platform を使用して、Jenkins エージェントを複数のコンテナーホストで動的にプロビジョニングできるようにします。

OpenShift Container Platform は、Kubernetes プラグインを使用するために、Jenkins エージェントとして使用するのに適したイメージを 5 つ (**Base**、**Maven**、および **Node.js** イメージ) 提供します。詳細は、Jenkins Agents を参照してください。

注記

jenkins-slave-maven-* および jenkins-slave-nodejs-* イメージは、v3.10 リリースサイクル中に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、Kubernetes Pod テンプレートイメージとして自動的に設定されます。この設定にはイメージごとのラベルが含まれており、"Restrict where this project can be run" の設定にある Jenkins ジョブのいずれかに適用できます。ラベルが適用されると、指定されたジョブの実行は、それぞれのエージェントイメージを実行している OpenShift Container Platform Pod の下で実行されます。

Jenkins イメージは、Kubernetes プラグインの追加のエージェントイメージの自動検出および自動設定を実行します。OpenShift Sync プラグイン では、Jenkins イメージは Jenkins の起動時に、実行中のプロジェクトまたはプラグインの設定に具体的に一覧表示されているプロジェクト内で以下を検索します。

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

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

イメージストリームまたはイメージストリームタグの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 が検出される場合は、ConfigMap のキーと値のデータペイロードの値に、Jenkins および Kubernetes プラグインの Pod テンプレートの設定形式に準拠する XML が含まれることが想定されます。ConfigMap を使用する時に注意すべき主な差別化要因は、イメージストリームまたはイメージストリームタグの代わりに、Kubernetes プラグインの Pod テンプレートのさまざまなフィールドをすべて制御できることです。

以下は ConfigMap の例になります。

```
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、OpenShift Sync プラグイン は、OpenShift Container Platform の API サーバーを監視して、**ImageStreams**、**ImageStreamTags**、および **ConfigMaps** の更新を確認し、Kubernetes プラグインの設定を調整します。

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ます。
- 同様に、これらのオブジェクトが削除されると、対応する設定が Kubernetes プラグインから削除されます。
- 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** が作成されます。
- **ConfigMap** フォームを介した **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** の変更が上書きされます。

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

4.2.3.6.1. パーミッションに関する考慮事項

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は、サービスアカウントに関連付けられたパーミッションとともに、OpenShift ContainerPlatform マスターに対するどの操作を Pod から許可するかを制御します。

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 が実行するプロジェクトの **edit** ロールで定義され、マスター Jenkins Pod にサービスアカウントがマウントされます。
- Jenkins 設定に挿入される 2 つのデフォルトの Maven および NodeJS Pod テンプレートも、マスターと同じサービスアカウントを使用するように設定されます。
- 必要なラベルまたはアノテーションを持つイメージストリームまたはイメージストリームタグの結果として、OpenShift Sync プラグイン によって自動検出された Pod テンプレートには、サービスアカウントがマスターのサービスアカウントに設定されています。
- Pod テンプレートの定義を Jenkins と Kubernetes プラグインに渡す他の方法として、使用するサービスアカウントを明示的に指定する必要があります。
- 他の方法には、Jenkins コンソール、Kubernetes プラグインで提供される **podTemplate** パイプライン DSL、または Pod テンプレートの XML 設定をデータとする ConfigMap のラベル付けなどが含まれます。
- サービスアカウントの値を指定しない場合には、**default** のサービスアカウントが使用されます。
- 使用するサービスアカウントが何であっても、必要なパーミッション、ロールなどを OpenShift Container Platform 内で定義して、Pod 内から操作することを選択したプロジェクトを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 にはパラメーターフィールドがあり、事前定義されたデフォルト値ですべての環境変数を定義します。OpenShift Container Platform では、新規の Jenkins サービスを簡単に作成できるようにテンプレートが提供されています。Jenkins テンプレートは、クラスターの初期設定時にクラスター管理者によって、デフォルトの **openshift** プロジェクトに登録されている必要があります。詳細は、必要に応じて、Loading the Default Image Streams and Templates 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、別のノードへの移動時や、デプロイメント設定の更新で再デプロイメントがトリガーされた時に、再起動される可能性があります。

- **jenkins-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。

Jenkins を使用可能にするには、テンプレートを インスタンス化 する必要があります。

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。Pod は Java ソースのクローンを作成し、WAR ファイルをビルドしてから、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化します。

同様のゴールを達成するより完全なサンプルは、こちら から入手できます。

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上記の例を変更したものです。

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" と呼ばれる新規の Pod テンプレートがオンザフライで定義されます。新しい Pod テンプレート名は、以下のノードのスタンザで参照されます。
2. "cloud" 値は "openshift" に設定する必要があります。
3. 新しい Pod テンプレートは、既存の Pod テンプレートから設定を継承できます。この場合、OpenShift Container Platform で事前定義されている "maven" Pod テンプレートから継承します。
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 に同梱される 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 に値 "0.25" を指定します。
9. ノードスタンザは、上記で新たに定義された Pod テンプレート名を参照します。

デフォルトで、Pod はビルドの完了時に削除されます。この動作は、プラグインを使用するか、またはパイプライン Jenkinsfile 内で変更できます。詳細は、Agent Pod Retention 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、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背景情報は、Sizing OpenJDK on OpenShift Container Platform を参照してください。

メモリー効率のために、メモリー制限が 2GiB 未満のコンテナーで実行されている場合、デフォルトで Jenkins イメージは 32 ビット JVM を動的に使用します。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、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。詳細は、Environment Variables 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すべてのプロセスでは、OOM kill を引き起こさずに、残りの 256MiB メモリーの合計を超えるメモリーを使用できる可能性はありません。そのため、パイプラインは可能な限り、エージェントコンテナーで外部コマンドを実行することを強くお勧めします。

Jenkins Kubernetes プラグインによって作成されるエージェントコンテナーで、メモリー要求および制

限の値を指定することが推奨されます。管理者は、Jenkins 設定を使用して、エージェントごとのイメージに基づいてデフォルトを設定できます。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 の一時テンプレートまたは永続テンプレートをインスタンス化する際に **MEMORY_LIMIT** パラメーターを上書きします。

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。これは、デフォルトで Jenkins イメージで使用できます。

4.2.5.1. OpenShift Container Platform クライアントプラグイン

OpenShift Container Platform クライアントプラグインは、OpenShift Container Platform との高度な対話を実現するために、読み取り可能かつ簡潔で、包括的で Fluent (流れるような) な Jenkins Pipeline 構文を提供することを目指しています。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。

このプラグインは完全にサポートされ、Jenkins イメージに含まれています。以下を提供します。

- Jenkins Pipeline で使用するための Fluent-style 構文
- **oc** で利用可能なオプションの使用および公開
- Jenkins 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

詳細は、OpenShift Pipeline Builds tutorial および the plug-in's 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 の以前の統合で、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。OpenShift Container Platform の後のバージョンについては、**oc** バイナリーを Jenkins Pipeline から直接使用するか、または OpenShift Container Platform クライアントプラグイン を使用します。

詳細は、the plug-in's README を参照してください。

4.2.5.3. OpenShift Container Platform 同期プラグイン

Jenkins と OpenShiftContainerPlatform を統合するための OpenShiftContainer Platform Pipeline ビルドストラテジー を容易にするために、OpenShift Sync プラグイン は、OpenShift Container Platform の API サーバーを監視して、Pipeline ストラテジーを採用する **BuildConfigs** および **Builds** への更新を確認し、Jenkins Pipeline プロジェクトを作成するか (**BuildConfig** の作成時)、作成されるプロジェクトでジョブを開始します (**Build** の開始時)。

Jenkins Kubernetes プラグインの設定 で説明されているように、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** オブジェクトを取得し、Jenkins 認証情報の階層内のデフォルトのグローバルドメイン

に配置される Jenkins 認証情報を構築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 (**-**)、そして **Secret** の名前で設定されます。

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 のオブジェクトへのその後の更新は、Jenkins 認証情報に適用されます (暫定的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 で関連する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。これらがない場合は、**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、Using the Jenkins Kubernetes Plug-in 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** イメージ) 提供します。

1 番目は、Jenkins エージェントの ベースイメージ です。

- 必須のツール (ヘッドレス Java、Jenkins JNLP クライアント) と有用なツール (git、tar、zip、nss など) の両方を取り入れます。

- JNLP エージェントをエントリーポイントとして確立します。

- Jenkins ジョブ内からコマンドラインの操作を呼び出す **oc** クライアントツールが含まれます。

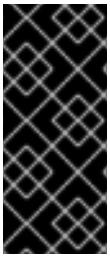
- CentOS イメージおよび RHEL イメージの両方に Dockerfile を提供します。

ベースイメージを拡張するイメージがさらに 2 つ提供されます。

- Maven v3.5 イメージ

- Node.js v10 イメージ および Node.js v12 イメージ

Maven および Node.js Jenkins エージェントイメージは、新しいエージェントイメージをビルドするときに参照できる CentOS と RHEL の両方の Dockerfile を提供します。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

使用している OpenShift Container Platform のバージョンに適したエージェントイメージバージョンを使用して拡張します。エージェントイメージに埋め込まれた **oc** クライアントバージョンが、OpenShift Container Platform バージョンと互換性がない場合、予期しない動作が発生する可能性があります。詳細は、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-10-rhel7
$ docker pull registry.redhat.io/openshift3/jenkins-agent-nodejs-12-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-10-centos7
$ docker pull openshift/jenkins-agent-nodejs-12-centos7
```

これらのイメージを使用するには、これらのレジストリーから直接アクセスするか、これらを OpenShift Container Platform コンテナーイメージレジストリーにプッシュできます。

4.3.3. 設定とカスタマイズ

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。メモリー効率のために、メモリー制限が 2GiB 未満のコンテナーで実行されている場合、デフォルトで Jenkins エージェントイメージは 32 ビット JVM を動的に使用します。

- **JAVA_MAX_HEAP_PARAM**
  CONTAINER_HEAP_PERCENT (デフォルト: **0.1**、10%)
  JNLP_MAX_HEAP_UPPER_BOUND_MB
  これらの値は Jenkins エージェント 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。それ以外の場合は、最大ヒープサイズはコンテナーメモリー制限の **CONTAINER_HEAP_PERCENT**% (設定例: **0.5**、つまり 50%) として動的に計算され、オプションで **JNLP_MAX_HEAP_UPPER_BOUND_MB** MiB (設定例: **512**) に制限されます。

  デフォルトでは、Jenkins エージェント 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 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: **-Xms32m**)、この値が優先されます。それ以外の場合は、初期ヒープサイズは、動的に計算された最大ヒープサイズの **CONTAINER_INITIAL_PERCENT**% (設定例: **0.1**、つまり 10%) として動的に計算される場合があります。

  デフォルトでは、最初のヒープサイズは JVM に任されています。

- **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**)
  こ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 JVM ガベージコレクションのパラメーターを指定します。この値の上書きは推奨していません。

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

JVM はすべての Jenkins エージェントで使用され、Jenkins JNLP エージェントをホストし、Java アプリケーション (**javac**、Maven または Gradle など) を実行します。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、Sizing OpenJDK on OpenShift Container Platform を参照してください。

メモリー効率のために、メモリー制限が 2GiB 未満のコンテナーで実行されている場合、デフォルトで Jenkins イメージは 32 ビット JVM を動的に使用します。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 JenkinsJNLP エージェントとエージェントコンテナー内の他の Java プロセスの両方に適用されます。

デフォルトで、Jenkins JNLP エージェントの JVM は、ヒープにコンテナーメモリー制限の 50% を使用します。この値は、**CONTAINER_HEAP_PERCENT** の環境変数で変更可能です。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。詳細は、Environment Variables 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すべてのプロセスでは、OOM kill を引き起こさずに、残りの 50% のメモリー制限を超えるメモリーを使用することはできません。

デフォルトでは、Jenkins エージェントコンテナーで実行される他の各 JVM プロセスは、最大でコンテナーメモリー制限の 25% をヒープに使用します。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。詳細は、Sizing OpenJDK on OpenShift Container Platform 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、テストが指定されている場合に Gradle が 3 番目の JVM を起動してテストを実行するので、さらに複雑になります。

OpenShift での JVM のチューニングに関する参考情報は、Sizing OpenJDK on OpenShift Container Platform を参照してください。

以下の設定は、OpenShift でメモリーに制約がある Jenkins エージェントの Gradle ビルドを実行する場合の開始点として推奨されます。必要に応じて、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、有効期間の長い gradle デーモンが無効になっていることを確認します。
- **org.gradle.parallel=true** が gradle.properties ファイルに設定されておらず、**--parallel** がコマンドライン引数として設定されていないことを確認して、並列ビルドの実行を無効にします。
- **java { options.fork = false }** を build.gradle ファイルに設定して、Java コンパイルがプロセス以外で実行されるのを防ぎます。
- 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。
- Sizing OpenJDK on OpenShift Container Platform 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。
- build.gradle の maxHeapSize および jvmArgs 設定によって、または **-Dorg.gradle.jvmargs** コマンドライン引数を使用して、Gradle テスト JVM の最大ヒープサイズおよび JVM 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 (別名: スレーブ Pod) は、ビルドの完了後、またはビルドの停止後にデフォルトで削除されます。この動作は、Kubernetes プラグインの **Pod Retention** (Pod の保持) 設定で変更できます。Pod の保持は、すべての Jenkins ビルドについて各 Pod テンプレートの上書きで設定できます。以下の動作がサポートされます。

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

Pod の保持はパイプライン Jenkinsfile で上書きできます。

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

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

警告

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

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必要がある場合は、他の任意のコンテナーイメージを OpenShift Container Platform インスタンス (Docker Hub にあるものなど) 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、Creating Images ガイドの Support Arbitrary User IDs を参照してください。

重要

サポート性の詳細については、OpenShift Container Platform Support Policy 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。

System and Environment Requirements のセキュリティー警告も参照してください。