OpenShift Container Platform 4.12

Jenkins

Jenkins

Last Updated: 2025-11-12

OpenShift Container Platform 4.12 Jenkins

Jenkins

Abstract

OpenShift Container Platform 用の Jenkins

Table of Contents

第1章 JENKINS イメージの設定

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

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

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

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

以下に例を示します。

```
$ podman pull registry.redhat.io/ocp-tools-4/jenkins-rhel8:<image_tag>
```

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で、対象イメージを参照するイメージストリームを作成することもできます。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。

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

1.1. 設定とカスタマイズ

Jenkins 認証は、以下の 2 つの方法で管理できます。

- OpenShift Container Platform 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。

1.1.1. OpenShift Container Platform OAuth 認証

OAuth 認証をアクティブ化するには、Jenkins UI の **Configure Global Security** パネルでオプションを設定するか、Jenkins の **Deployment configuration** の **OPENSHIFT_ENABLE_OAUTH** 環境変数を **false** 以外に設定します。これにより、OpenShift Container Platform ログインプラグインが有効になり、Pod データからか、OpenShift Container Platform API サーバーと対話して設定情報を取得します。

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

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

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

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

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

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

ログインプラグインがその config map を検出して読み取ることができる場合は、Jenkins パーミッションマッピングにロールを定義できます。具体的には以下を実行します。

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

注記

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

保存される Jenkins ユーザーのパーミッションは、初回のユーザー作成後に変更できます。OpenShift Container Platform 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は、プラグインが次回に OpenShift Container Platform をポーリングするタイミングで、パーミッションの変更が上書きされます。

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

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

1.1.2. Jenkins 認証

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

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

手順

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

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    ocp-tools-4/jenkins-rhel8
```

1.2. JENKINS 環境変数

Jenkins サーバーは、以下の環境変数で設定できます。

変数	定義	値と設定の例
OPENSHIFT_ENABLE_OAUTH	Jenkins へのログイン時に OpenShift Container Platform ログインプラグインが認証を管理するかどうかを決定します。有効にするには、**true** に設定します。	デフォルト: **false**
JENKINS_PASSWORD	標準の Jenkins 認証を使用する際の **admin** ユーザーのパスワード。**OPENSHIFT_ENABLE_OAUTH** が **true** に設定されている場合は該当しません。	デフォルト: **password**

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

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

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

変数	定義	値と設定の例
AGENT_BASE_IMAGE	この値を設定すると、このイメージで提供されるサンプルの Kubernetes プラグイン Pod テンプレートで **jnlp** コンテナーに使用されるイメージが上書きされます。それ以外の場合は、**openshift** namespace の **jenkins-agent-base-rhel8:latest** イメージストリームタグが使用されます。	デフォルト: **image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest**
JAVA_BUILDER_IMAGE	この値を設定すると、このイメージで提供される **java-builder** サンプルの Kubernetes プラグイン Pod テンプレートで **java-builder** コンテナーに使用されるイメージが上書きされます。それ以外の場合は、**openshift** namespace の **java:latest** イメージストリームタグからのイメージが使用されます。	デフォルト: **image-registry.openshift-image-registry.svc:5000/openshift/java:latest**
JAVA_FIPS_OPTIONS	この値を設定すると、FIPS ノードで実行しているときに JVM がどのように動作するかを制御します。詳細については、FIPS モードで OpenJDK 11 を設定 するを参照してください。	デフォルト: **-Dcom.redhat.fips=false**

1.3. JENKINS へのプロジェクト間のアクセスの提供

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

手順

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

```
$ oc describe serviceaccount jenkins
```

出力例

```
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>
```

出力例

```
Name:       jenkins-token-uyswp
Labels:     <none>
Annotations:    kubernetes.io/service-account.name=jenkins,kubernetes.io/service-account.uid=32f5b661-2a8f-11e5-9528-3c970e3bf0b7
Type:   kubernetes.io/service-account-token
Data
====
ca.crt: 1066 bytes
token:  eyJhbGc..<content cut>....wRA
```

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

1.4. JENKINS のボリューム間のマウントポイント

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

- **/var/lib/jenkins** - 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。

1.5. SOURCE-TO-IMAGE による JENKINS イメージのカスタマイズ

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

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

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

plugins

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

plugins.txt

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

```
pluginId:pluginVersion
```

configuration/jobs

このディレクトリーには、Jenkins ジョブ定義が含まれます。

configuration/config.xml

このファイルには、カスタムの Jenkins 設定が含まれます。

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

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

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

1.6. JENKINS KUBERNETES プラグインの設定

OpenShift Jenkins イメージにはプリインストールされた Jenkins 用の Kubernetes プラグイン が含まれているため、Kubernetes と OpenShift Container Platform を使用して複数のコンテナーホストで Jenkins エージェントを動的にプロビジョニングできます。

Kubernetes プラグインを使用するために、OpenShift Container Platform は、Jenkins エージェントとしての使用に適した OpenShift Agent Base イメージを提供します。

重要

OpenShift Container Platform 4.11 は、Red Hat が OpenShift Container Platform ライフサイクル外でイメージを生成および更新できるように、OpenShift Jenkins および OpenShift Agent Base イメージを **registry.redhat.io** の **ocp-tools-4** リポジトリーに移動します。以前のバージョンでは、これらのイメージは OpenShift Container Platform インストールペイロードに使用され、**openshift4** リポジトリーは **registry.redhat.io** にありました。

OpenShift Jenkins Maven および NodeJS Agent イメージは、OpenShift Container Platform 4.11 ペイロードから削除されました。Red Hat はこれらのイメージを生成しなくなり、**registry.redhat.io** の **ocp-tools-4** リポジトリーから入手できなくなりました。Red Hat は、OpenShift Container Platform ライフサイクルポリシー に従って、重要なバグ修正またはセキュリティー CVE のためにこれらのイメージの 4.10 以前のバージョンを維持します。

詳細は、次の「関連情報」セクションの 「OpenShift Jenkins イメージに対する重要な変更」リンクを参照してください。

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、**Restrict where this project can be run** 設定で任意の Jenkins ジョブに適用できる各イメージのラベルが含まれています。ラベルが適用されている場合、ジョブはそれぞれのエージェントイメージを実行する OpenShift Container Platform Pod の下で実行されます。

重要

OpenShift Container Platform 4.10 以降では、Kubernetes プラグインを使用して Jenkins エージェントを実行するために推奨されるパターンは、**jnlp** および **sidecar** コンテナーの両方で Pod テンプレートを使用することです。**jnlp** コンテナーは、OpenShift Container Platform Jenkins Base エージェントイメージを使用して、ビルド用の別の Pod の起動を容易にします。**sidecar** コンテナーイメージには、起動した別の Pod 内の特定の言語でビルドするために必要なツールがあります。Red Hat Container Catalog の多くのコンテナーイメージは、**openshift** namespace にあるサンプルイメージストリームで参照されます。OpenShift Container Platform Jenkins イメージには、このアプローチを示すサイドカーコンテナーを含む **java-build** という名前の Pod テンプレートがあります。この Pod テンプレートは、**openshift** namespace の **Java** イメージストリームによって提供される最新の Java バージョンを使用します。

Jenkins イメージは、Kubernetes プラグイン向けの追加エージェントイメージの自動検出および自動設定も提供します。

OpenShift Container Platform 同期プラグインを使用すると、Jenkins の起動時に、Jenkins イメージが実行中のプロジェクト内、またはプラグインの設定にリストされているプロジェクト内で以下の項目を検索します。

- **role** ラベルが **jenkins-agent** に設定されたイメージストリーム。
- **role** アノテーションが **jenkins-agent** に設定されたイメージストリームタグ。
- **role** ラベルが **jenkins-agent** に設定された config map。

Jenkins イメージは、適切なラベルを持つイメージストリーム、または適切なアノテーションを持つイメージストリームタグを見つけると、対応する Kubernetes プラグイン設定を生成します。このようにして、イメージストリームによって提供されるコンテナーイメージを実行する Pod で実行するように

Jenkins ジョブを割り当てることができます。

イメージストリームまたはイメージストリームタグの名前とイメージ参照は、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、**agent-label** キーを使用してイメージストリームまたはイメージストリームタグオブジェクトにアノテーションを設定することで制御できます。これらを使用しない場合には、名前をラベルとして使用します。

注記

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

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

適切なラベルを持つ config map が見つかると、Jenkins イメージは、config map のキーと値のデータペイロードの任意の値に、Jenkins および Kubernetes プラグイン Pod テンプレートの設定形式と一致する Extensible Markup Language (XML) が含まれていると想定します。イメージストリームやイメージストリームタグに対する config map の主な利点の 1 つは、すべての Kubernetes プラグイン Pod テンプレートパラメーターを制御できることです。

jenkins-agent の config map の例:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-agent
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>
```

以下の例は、**openshift** namespace のイメージストリームを参照する 2 つのコンテナーを示しています。1 つのコンテナーが、Pod を Jenkins エージェントとして起動するための JNLP コントラクトを処理します。もう 1 つのコンテナーは、特定のコーディング言語でコードを構築するためのツールを備えたイメージを使用します。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-agent
data:
  template2: |-
        <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
          <inheritFrom></inheritFrom>
          <name>template2</name>
          <instanceCap>2147483647</instanceCap>
          <idleMinutes>0</idleMinutes>
          <label>template2</label>
          <serviceAccount>jenkins</serviceAccount>
          <nodeSelector></nodeSelector>
          <volumes/>
          <containers>
            <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
              <name>jnlp</name>
              <image>image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-
rhel8:latest</image>
              <privileged>false</privileged>
              <alwaysPullImage>true</alwaysPullImage>
              <workingDir>/home/jenkins/agent</workingDir>
              <command></command>
              <args>\$(JENKINS_SECRET) \$(JENKINS_NAME)</args>
              <ttyEnabled>false</ttyEnabled>
              <resourceRequestCpu></resourceRequestCpu>
              <resourceRequestMemory></resourceRequestMemory>
              <resourceLimitCpu></resourceLimitCpu>
              <resourceLimitMemory></resourceLimitMemory>
              <envVars/>
            </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
            <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
              <name>java</name>
              <image>image-registry.openshift-image-registry.svc:5000/openshift/java:latest</image>
              <privileged>false</privileged>
              <alwaysPullImage>true</alwaysPullImage>
              <workingDir>/home/jenkins/agent</workingDir>
```

```
        <command>cat</command>
        <args></args>
        <ttyEnabled>true</ttyEnabled>
        <resourceRequestCpu></resourceRequestCpu>
        <resourceRequestMemory></resourceRequestMemory>
        <resourceLimitCpu></resourceLimitCpu>
        <resourceLimitMemory></resourceLimitMemory>
        <envVars/>
      </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
    </containers>
    <envVars/>
    <annotations/>
    <imagePullSecrets/>
    <nodeProperties/>
  </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

注記

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

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

インストールされた後、OpenShift Container Platform 同期プラグインは、イメージストリーム、イメージストリームタグ、および config map に更新がないか、OpenShift Container Platform の API サーバーをモニタリングして、Kubernetes プラグインの設定を調整します。

以下のルールが適用されます。

- config map、イメージストリーム、またはイメージストリームタグからラベルまたはアノテーションを削除すると、既存の **PodTemplate** が Kubernetes プラグインの設定から削除されます。
- これらのオブジェクトが削除されると、対応する設定が Kubernetes プラグインから削除されます。
- 適切なラベルおよびアノテーションが付いた **ConfigMap**、**ImageStream**、または **ImageStreamTag** オブジェクトを作成するか、最初の作成後にラベルを追加すると、Kubernetes プラグイン設定で **PodTemplate** が作成されます。
- config map 形式による **PodTemplate** の場合、**PodTemplate** の config map データへの変更は、Kubernetes プラグイン設定の **PodTemplate** 設定に適用されます。この変更は、config map への変更と変更の間に Jenkins UI を介して **PodTemplate** に加えられた変更もオーバーライドします。

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、イメージが、エントリーポイントとしてエージェントを実行する必要があります。詳細は、公式の Jenkins ドキュメント を参照してください。

関連情報

- OpenShift Jenkins イメージに対する重要な変更

1.7. JENKINS パーミッション

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

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

OpenShift Container Platform で提供される Jenkins のテンプレートサンプルを使用する場合は、**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、Jenkins マスターと同じサービスアカウントを使用するように設定します。

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

1.8. テンプレートからの JENKINS サービスの作成

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

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

注記

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

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

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

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

手順

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

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

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

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

両方のテンプレートで、それらに対して **oc describe** を実行して、オーバーライドに使用できるすべてのパラメーターを確認できます。

以下に例を示します。

```
$ oc describe jenkins-ephemeral
```

1.9. JENKINS KUBERNETES プラグインの使用

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

> **重要**
>
> OpenShift Container Platform 4.11 は、そのペイロードから OpenShift Jenkins Maven および NodeJS Agent イメージを削除しました。Red Hat はこれらのイメージを生成しなくなり、**registry.redhat.io** の **ocp-tools-4** リポジトリーから入手できなくなりました。Red Hat は、OpenShift Container Platform ライフサイクルポリシー に従って、重要なバグ修正またはセキュリティー CVE のためにこれらのイメージの 4.10 以前のバージョンを維持します。
>
> 詳細は、次の「関連情報」セクションの 「OpenShift Jenkins イメージに対する重要な変更」リンクを参照してください。

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

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: image.openshift.io/v1
  metadata:
    name: openshift-jee-sample
```

```
- kind: BuildConfig
  apiVersion: build.openshift.io/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: build.openshift.io/v1
  metadata:
    name: openshift-jee-sample
  spec:
    strategy:
      type: JenkinsPipeline
      jenkinsPipelineStrategy:
        jenkinsfile: |-
          node("maven") {
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
    triggers:
    - type: ConfigChange
```

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

Jenkins Kubernetes プラグインを使用し、メモリー制限と環境変数を指定するサンプル BuildConfig

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", (1)
                    cloud: "openshift", (2)
                    inheritFrom: "maven", (3)
                    containers: [
```

```
          containerTemplate(name: "jnlp", ④
                   image: "openshift/jenkins-agent-maven-35-centos7:v3.10", ⑤
                   resourceRequestMemory: "512Mi", ⑥
                   resourceLimitMemory: "512Mi", ⑦
                   envVars: [
            envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25") ⑧
          ])
        ]) {
        node("mypod") { ⑨
          sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
          sh "mvn -B -Popenshift package"
          sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
        }
      }
triggers:
- type: ConfigChange
```

1. **mypod** という名前の新しい Pod テンプレートが動的に定義されます。この新しい Pod テンプレート名はノードのスタンザで参照されます。
2. **cloud** の値は **openshift** に設定する必要があります。
3. 新しい Pod テンプレートは、既存の Pod テンプレートから設定を継承できます。この場合、OpenShift Container Platform で事前定義された Maven Pod テンプレートから継承されます。
4. この例では、既存のコンテナーの値を上書きし、名前で指定する必要があります。OpenShift Container Platform に含まれる Jenkins エージェントイメージはすべて、コンテナー名として **jnlp** を使用します。
5. 再びコンテナーイメージ名を指定します。これは既知の問題です。
6. **512 Mi** のメモリー要求を指定します。
7. **512 Mi** のメモリー制限を指定します。
8. 環境変数 **CONTAINER_HEAP_PERCENT** に値 **0.25** を指定します。
9. ノードスタンザは、定義された Pod テンプレート名を参照します。

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

アップストリームの Jenkins では少し前に、パイプラインとインラインで **podTemplate** パイプライン DSL を定義するための YAML 宣言型フォーマットが導入されました。OpenShift Container Platform の Jenkins イメージで定義されているサンプル **java-builder** Pod テンプレートを使用したこのフォーマットの例は以下のようになります。

```
def nodeLabel = 'java-buidler'

pipeline {
  agent {
    kubernetes {
      cloud 'openshift'
      label nodeLabel
```

```
      yaml """
apiVersion: v1
kind: Pod
metadata:
  labels:
    worker: ${nodeLabel}
spec:
  containers:
  - name: jnlp
    image: image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest
    args: ['\$(JENKINS_SECRET)', '\$(JENKINS_NAME)']
  - name: java
    image: image-registry.openshift-image-registry.svc:5000/openshift/java:latest
    command:
    - cat
    tty: true
"""
    }
  }

  options {
    timeout(time: 20, unit: 'MINUTES')
  }

  stages {
    stage('Build App') {
      steps {
        container("java") {
          sh "mvn --version"
        }
      }
    }
  }
}
```

関連情報

- OpenShift Jenkins イメージに対する重要な変更

1.10. JENKINS メモリーの要件

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

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

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

Jenkins Kubernetes プラグインによって作成されるエージェントコンテナーで、メモリー要求および制限の値を指定することが推奨されます。管理者ユーザーは、Jenkins 設定を使用して、エージェントの

イメージごとにデフォルト値を設定できます。メモリー要求および制限パラメーターは、コンテナーごとに上書きすることもできます。

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

1.11. 関連情報

- Red Hat Universal Base Images (UBI) の詳細は、ベースイメージのオプション を参照してください。
- OpenShift Jenkins イメージに対する重要な変更

第2章 JENKINS エージェント

OpenShift Container Platform は、Jenkins エージェントとして使用するベースイメージを提供します。

Jenkins エージェントのベースイメージは次のことを行います。

- 必須のツール (ヘッドレス Java、Jenkins JNLP クライアント) と有用なツール (**git**、**tar**、**zip**、**nss** など) の両方を取り入れます。
- JNLP エージェントをエントリーポイントとして確立します。
- Jenkins ジョブ内からコマンドラインの操作を呼び出す **oc** クライアントツールが含まれます。
- Red Hat Enterprise Linux (RHEL) および **localdev** イメージの両方の Dockerfile を提供します。

重要

OpenShift Container Platform リリースバージョンに適したバージョンのエージェントイメージを使用してください。OpenShift Container Platform バージョンと互換性のない **oc** クライアントバージョンを埋め込むと、予期しない動作が発生する可能性があります。

OpenShift Container Platform Jenkins イメージは、Jenkins Kubernetes プラグインでエージェントイメージを使用する方法を示すために、次のサンプル **java-builder** Pod テンプレートも定義します。

java-builder Pod テンプレートは 2 つのコンテナーを使用します。* OpenShift Container Platform Base エージェントイメージを使用し、Jenkins エージェントを開始および停止するための JNLP コントラクトを処理する **jnlp** コンテナー。* **java** OpenShift Container Platform サンプル ImageStream を使用する **java** コンテナー。これには、コードをビルドするための Maven バイナリー **mvn** を含むさまざまな Java バイナリーが含まれています。

2.1. JENKINS エージェントイメージ

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

Jenkins イメージは、Red Hat レジストリーから入手できます。

```
$ docker pull registry.redhat.io/ocp-tools-4/jenkins-rhel8:<image_tag>
```

```
$ docker pull registry.redhat.io/ocp-tools-4/jenkins-agent-base-rhel8:<image_tag>
```

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

2.2. JENKINS エージェントの環境変数

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

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

変数	定義	値と設定の例
JAVA_GC_OPTS	Jenkins JVM ガーベッジコレクションのパラメーターを指定します。この値の上書きは推奨されません。	デフォルト: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**
JENKINS_JAVA_OVERRIDES	Jenkins JVM の追加オプションを指定します。これらのオプションは、上記の Java オプションを含む、その他すべてのオプションに追加され、必要に応じてそれらのいずれかを上書きするために使用できます。追加オプションがある場合は、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**
USE_JAVA_VERSION	コンテナーでエージェントを実行するために使用する Java バージョンのバージョンを指定します。コンテナーの基本イメージには、**java-11** と **java-1.8.0** の 2 つのバージョンの Java がインストールされています。コンテナーの基本イメージを拡張する場合は、関連付けられた接尾辞を使用して、Java の任意の代替バージョンを指定できます。	デフォルト値は **java-11** です。 設定例: **java-1.8.0**

2.3. JENKINS エージェントのメモリー要件

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

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

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

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

2.4. JENKINS エージェントの GRADLE ビルド

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

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

- **gradle.properties** ファイルに **org.gradle.daemon=false** を追加して、有効期間の長い (long-lived) Gradle デーモンを無効にするようにします。
- **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、プロセス以外で Java がコンパイルされないようにします.
- **build.gradle** ファイルで **test { maxParallelForks = 1 }** が設定されていることを確認して、複数の追加テストプロセスを無効にします。
- **GRADLE_OPTS**、**JAVA_OPTS**、または **JAVA_TOOL_OPTIONS** 環境変数で、Gradle JVM メモリーパラメーターを上書きします。
- **build.gradle** の **maxHeapSize** および **jvmArgs** 設定を定義するか、**-Dorg.gradle.jvmargs** コマンドライン引数を使用して、Gradle テスト JVM に最大ヒープサイズと JVM の引数を設定します。

2.5. JENKINS エージェント POD の保持

Jenkins エージェント Pod は、ビルドが完了するか、停止された後にデフォルトで削除されます。この動作は、Kubernetes プラグイン 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。

第3章 JENKINS から OPENSHIFT PIPELINES または TEKTON への移行

CI/CD ワークフローを Jenkins から Red Hat OpenShift Pipelines に移行できます。これは、Tekton プロジェクトに基づくクラウドネイティブの CI/CD エクスペリエンスです。

3.1. JENKINS と OPENSHIFT PIPELINES のコンセプトの比較

Jenkins および OpenShift Pipelines で使用される以下の同等の用語を確認および比較できます。

3.1.1. Jenkins の用語

Jenkins は、共有ライブラリーおよびプラグインを使用して拡張可能な宣言型およびスクリプト化されたパイプラインを提供します。Jenkins における基本的な用語は以下のとおりです。

- **Pipeline**: Groovy 構文を使用してアプリケーションをビルドし、テストし、デプロイするプロセスをすべて自動化します。
- **ノード**: スクリプト化されたパイプラインのオーケストレーションまたは実行できるマシン。
- **ステージ**: パイプラインで実行されるタスクの概念的に異なるサブセット。プラグインまたはユーザーインターフェイスは、このブロックを使用してタスクの状態または進捗を表示します。
- **ステップ**: コマンドまたはスクリプトを使用して、実行する正確なアクションを指定する単一タスク。

3.1.2. OpenShift Pipelines の用語

OpenShift Pipelines は、宣言型パイプラインに YAML 構文を使用し、タスクで構成されます。OpenShift Pipelines の基本的な用語は次のとおりです。

- **パイプライン**: 一連のタスク、並行したタスク、またはその両方。
- **タスク**: コマンド、バイナリー、またはスクリプトとしてのステップシーケンス。
- **PipelineRun**: 1 つ以上のタスクを使用したパイプラインの実行。
- **TaskRun**: 1 つ以上のステップを使用したタスクの実行。

> **注記**
>
> パラメーターやワークスペースなどの入力のセットを使用して PipelineRun または TaskRun を開始し、実行結果を出力およびアーティファクトのセットで開始できます。

- **Workspace**: OpenShift Pipelines では、ワークスペースは次の目的に役立つ概念的なブロックです。
  - 入力、出力、およびビルドアーティファクトのストレージ。
  - タスク間でデータを共有する一般的な領域。

    - シークレットに保持される認証情報のマウントポイント、config map に保持される設定、および組織が共有される共通のツール。

> **注記**
>
> Jenkins には、OpenShift Pipelines ワークスペースに直接相当するものはありません。コントロールノードは、クローン作成したコードリポジトリー、ビルド履歴、およびアーティファクトを格納するため、ワークスペースと考えることができます。ジョブが別のノードに割り当てられると、クローン化されたコードと生成されたアーティファクトはそのノードに保存されますが、コントロールノードはビルド履歴を維持します。

3.1.3. 概念のマッピング

Jenkins と OpenShift Pipelines のビルディングブロックは同等ではなく、特定の比較では技術的に正確なマッピングは提供されません。Jenkins と OpenShift Pipelines の以下の用語と概念は、一般的に相互に関連しています。

表3.1 Jenkins と OpenShift Pipelines - 基本的な比較

Jenkins	OpenShift Pipelines
パイプライン	パイプラインおよび PipelineRun
ステージ	タスク
Step	タスクのステップ

3.2. JENKINS から OPENSHIFT PIPELINES へのサンプルパイプラインの移行

以下の同等の例を使用して、Jenkins から OpenShift Pipelines へのパイプラインのビルド、テスト、およびデプロイを支援できます。

3.2.1. Jenkins パイプライン

ビルド、テスト、デプロイ用に Groovy で記述された Jenkins パイプラインを検討します。

```
pipeline {
   agent any
   stages {
       stage('Build') {
           steps {
               sh 'make'
           }
       }
       stage('Test'){
           steps {
               sh 'make check'
               junit 'reports/**/*.xml'
           }
       }
```

```
        stage('Deploy') {
            steps {
                sh 'make publish'
            }
        }
    }
}
```

3.2.2. OpenShift Pipelines パイプライン

前の Jenkins パイプラインと同等のパイプラインを OpenShift Pipelines で作成するには、次の 3 つのタスクを作成します。

build タスクの YAML 定義ファイルの例

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-build
spec:
  workspaces:
  - name: source
  steps:
  - image: my-ci-image
    command: ["make"]
    workingDir: $(workspaces.source.path)
```

test タスクの YAML 定義ファイルの例

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-test
spec:
  workspaces:
  - name: source
  steps:
  - image: my-ci-image
    command: ["make check"]
    workingDir: $(workspaces.source.path)
  - image: junit-report-image
    script: |
      #!/usr/bin/env bash
      junit-report reports/**/*.xml
    workingDir: $(workspaces.source.path)
```

deploy タスクの YAML 定義ファイルの例

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myprojectd-deploy
spec:
  workspaces:
```

```
  - name: source
  steps:
  - image: my-deploy-image
    command: ["make deploy"]
    workingDir: $(workspaces.source.path)
```

3 つのタスクを順番に組み合わせて、OpenShift Pipelines でパイプラインを形成できます。

例: ビルド、テスト、およびデプロイのための OpenShift Pipelines

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: myproject-pipeline
spec:
  workspaces:
  - name: shared-dir
  tasks:
  - name: build
    taskRef:
      name: myproject-build
    workspaces:
    - name: source
      workspace: shared-dir
  - name: test
    taskRef:
      name: myproject-test
    workspaces:
    - name: source
      workspace: shared-dir
  - name: deploy
    taskRef:
      name: myproject-deploy
    workspaces:
    - name: source
      workspace: shared-dir
```

3.3. JENKINS プラグインから TEKTON HUB タスクへの移行

プラグイン を使用して、Jenkins の機能を拡張できます。OpenShift Pipelines で同様の拡張性を実現するには、Tekton Hub から利用可能なタスクのいずれかを使用します。

たとえば、Jenkins の git plugin に対応する Tekton Hub の git-clone タスクを考えてみます。

例: Tekton Hub からの git-clone タスク

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
 name: demo-pipeline
spec:
 params:
   - name: repo_url
   - name: revision
 workspaces:
```

```
    - name: source
  tasks:
    - name: fetch-from-git
      taskRef:
        name: git-clone
      params:
        - name: url
          value: $(params.repo_url)
        - name: revision
          value: $(params.revision)
      workspaces:
      - name: output
        workspace: source
```

3.4. カスタムタスクとスクリプトを使用した OPENSHIFT PIPELINES 機能の拡張

OpenShift Pipelines では、Tekton Hub で適切なタスクが見つからない場合、またはタスクをより細かく制御する必要がある場合は、カスタムタスクとスクリプトを作成して OpenShift Pipelines の機能を拡張できます。

例: maven test コマンドを実行するカスタムタスク

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-test
spec:
  workspaces:
  - name: source
  steps:
  - image: my-maven-image
    command: ["mvn test"]
    workingDir: $(workspaces.source.path)
```

例: パスを指定してカスタムシェルスクリプトを実行する

```
...
steps:
  image: ubuntu
  script: |
      #!/usr/bin/env bash
      /workspace/my-script.sh
...
```

例: カスタム Python スクリプトを YAML ファイルに書き込んで実行する

```
...
steps:
  image: python
  script: |
```

```
    #!/usr/bin/env python3
    print("hello from python!")
...
```

3.5. JENKINS と OPENSHIFT PIPELINES の実行モデルの比較

Jenkins と OpenShift Pipelines は同様の機能を提供しますが、アーキテクチャーと実行が異なります。

表3.2 Jenkins と OpenShift Pipelines の実行モデルの比較

Jenkins	OpenShift Pipelines
Jenkins にはコントローラーノードがあります。Jenkins は、パイプラインとステップを一元的に実行するか、他のノードで実行しているジョブのオーケストレーションを行います。	OpenShift Pipelines はサーバーレスで分散されており、実行のための central 依存関係はありません。
コンテナーは、パイプラインを介して Jenkins コントローラーノードによって起動されます。	OpenShift Pipelines は、'コンテナーファースト' アプローチを採用しています。このアプローチでは、すべてのステップが Pod 内のコンテナーとして実行されます (Jenkins のノードに相当)。
プラグインを使用することで拡張性が実現されます。	拡張性は、Tekton Hub のタスクを使用するか、カスタムタスクおよびスクリプトを作成して実行します。

3.6. 一般的な使用例の例

Jenkins と OpenShift Pipelines はどちらも、次のような一般的な CI/CD ユースケース向けの機能を提供します。

- Apache Maven を使用したイメージのコンパイル、ビルド、およびデプロイ
- プラグインを使用してコア機能の拡張
- 共有可能なライブラリーおよびカスタムスクリプトの再利用

3.6.1. Jenkins および OpenShift Pipelines での Maven パイプラインの実行

Jenkins ワークフローと OpenShift Pipelines ワークフローの両方で Maven を使用して、イメージのコンパイル、ビルド、およびデプロイを行うことができます。既存の Jenkins ワークフローを OpenShift Pipelines にマッピングするには、以下の例を検討してください。

例: Jenkins の Maven を使用して、イメージをコンパイルおよびビルドし、OpenShift にデプロイする

```
#!/usr/bin/groovy
node('maven') {
   stage 'Checkout'
   checkout scm

   stage 'Build'
```

```
    sh 'cd helloworld && mvn clean'
    sh 'cd helloworld && mvn compile'

    stage 'Run Unit Tests'
    sh 'cd helloworld && mvn test'

    stage 'Package'
    sh 'cd helloworld && mvn package'

    stage 'Archive artifact'
    sh 'mkdir -p artifacts/deployments && cp helloworld/target/*.war artifacts/deployments'
    archive 'helloworld/target/*.war'

    stage 'Create Image'
    sh 'oc login https://kubernetes.default -u admin -p admin --insecure-skip-tls-verify=true'
    sh 'oc new-project helloworldproject'
    sh 'oc project helloworldproject'
    sh 'oc process -f helloworld/jboss-eap70-binary-build.json | oc create -f -'
    sh 'oc start-build eap-helloworld-app --from-dir=artifacts/'

    stage 'Deploy'
    sh 'oc new-app helloworld/jboss-eap70-deploy.json' }
```

例: OpenShift Pipelines の Maven を使用して、イメージをコンパイルおよびビルドし、OpenShift にデプロイする

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: maven-pipeline
spec:
  workspaces:
    - name: shared-workspace
    - name: maven-settings
    - name: kubeconfig-dir
      optional: true
  params:
    - name: repo-url
    - name: revision
    - name: context-path
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      workspaces:
        - name: output
          workspace: shared-workspace
      params:
        - name: url
          value: "$(params.repo-url)"
        - name: subdirectory
          value: ""
        - name: deleteExisting
          value: "true"
        - name: revision
          value: $(params.revision)
```

```
- name: mvn-build
  taskRef:
    name: maven
  runAfter:
    - fetch-repo
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "$(params.context-path)"
    - name: GOALS
      value: ["-DskipTests", "clean", "compile"]
- name: mvn-tests
  taskRef:
    name: maven
  runAfter:
    - mvn-build
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "$(params.context-path)"
    - name: GOALS
      value: ["test"]
- name: mvn-package
  taskRef:
    name: maven
  runAfter:
    - mvn-tests
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "$(params.context-path)"
    - name: GOALS
      value: ["package"]
- name: create-image-and-deploy
  taskRef:
    name: openshift-client
  runAfter:
    - mvn-package
  workspaces:
    - name: manifest-dir
      workspace: shared-workspace
    - name: kubeconfig-dir
      workspace: kubeconfig-dir
  params:
```

```
- name: SCRIPT
  value: |
    cd "$(params.context-path)"
    mkdir -p ./artifacts/deployments && cp ./target/*.war ./artifacts/deployments
    oc new-project helloworldproject
    oc project helloworldproject
    oc process -f jboss-eap70-binary-build.json | oc create -f -
    oc start-build eap-helloworld-app --from-dir=artifacts/
    oc new-app jboss-eap70-deploy.json
```

3.6.2. プラグインを使用して Jenkins および OpenShift Pipelines のコア機能を拡張する

Jenkins には、その広範なユーザーベースによって長年にわたって開発された多数のプラグインの大規模なエコシステムという利点があります。Jenkins プラグインインデックス でプラグインを検索および参照できます。

OpenShift Pipelines には、コミュニティーおよびエンタープライズユーザーによって開発および提供された多くのタスクもあります。再利用可能な OpenShift Pipelines タスクの公開されているカタログは、Tekton Hub で入手できます。

さらに、OpenShift Pipelines は、Jenkins エコシステムのプラグインの多くをコア機能に組み込んでいます。たとえば、承認は Jenkins と OpenShift Pipelines の両方で重要な機能です。Jenkins は Role-based Authorization Strategy プラグインを使用して認可を保証しますが、OpenShift Pipelines は OpenShift のビルトインロールベースアクセス制御システムを使用します。

3.6.3. Jenkins および OpenShift Pipelines での再利用可能なコードの共有

Jenkins 共有ライブラリー は、Jenkins パイプラインの一部に再利用可能なコードを提供します。ライブラリーは、Jenkinsfiles 間で共有され、コードの繰り返しなしに、高度にモジュール化されたパイプラインを作成します。

OpenShift Pipelines には Jenkins 共有ライブラリーの直接の機能は存在しませんが、カスタムタスクやスクリプトと組み合わせて Tekton Hub のタスクを使用して同様のワークフローを実行できます。

3.7. 関連情報

- OpenShift Pipelines について
- ロールベースのアクセス制御

第4章 OPENSHIFT JENKINS イメージに対する重要な変更

重要

OpenShift Jenkins は Jenkins LTS リリースおよび関連するプラグインから定期的な更新を受け取ります。これらの更新には、バグ修正、セキュリティー脆弱性パッチ、および時折新機能が含まれる場合があります。ただし、Red Hat は、依存するプラグインへの更新以外に、OpenShift Jenkins コンテナーイメージの機能または内容にさらなる拡張機能や大幅な変更を導入する予定はありません。

また、以下の 3 つの Red Hat が管理する Jenkins プラグインがメンテナンスモードになりました。重大なバグ修正のみが対応され、新しい機能拡張や機能開発は予定されていません。

- Jenkins クライアントプラグイン
- Jenkins ログインプラグイン
- Jenkins 同期プラグイン

OpenShift Container Platform 4.11 は、OpenShift Jenkins および OpenShift Agent Base イメージを **registry.redhat.io** の **ocp-tools-4** リポジトリーに移動します。また、ペイロードから OpenShift Jenkins Maven および NodeJS Agent イメージを削除します。

- OpenShift Container Platform 4.11 は、Red Hat が OpenShift Container Platform ライフサイクル外でイメージを生成および更新できるように、OpenShift Jenkins および OpenShift Agent Base イメージを **registry.redhat.io** の **ocp-tools-4** リポジトリーに移動します。以前のバージョンでは、これらのイメージは OpenShift Container Platform インストールペイロードに使用され、**openshift4** リポジトリーは **registry.redhat.io** にありました。

- OpenShift Container Platform 4.10 は、OpenShift Jenkins Maven および NodeJS Agent イメージを非推奨にしました。OpenShift Container Platform 4.11 は、これらのイメージをペイロードから削除します。Red Hat はこれらのイメージを生成しなくなり、**registry.redhat.io** の **ocp-tools-4** リポジトリーから入手できなくなりました。Red Hat は、 OpenShift Container Platform ライフサイクルポリシー に従って、重要なバグ修正またはセキュリティー CVE のためにこれらのイメージの 4.10 以前のバージョンを維持します。

これらの変更は、Jenkins Kubernetes プラグインで複数のコンテナー Pod テンプレート を使用するという OpenShift Container Platform 4.10 の推奨事項に対応したものです。

4.1. OPENSHIFT JENKINS イメージの再配置

OpenShift Container Platform 4.11 では、特定の OpenShift Jenkins イメージの場所と可用性が大幅に変更されています。さらに、これらのイメージをいつ、どのように更新するかを設定できます。

OpenShift Jenkins イメージの変わらない点

- Cluster Samples Operator は、OpenShift Jenkins イメージを操作するための **ImageStream** および **Template** オブジェクトを管理します。

- デフォルトでは、Jenkins Pod テンプレートの Jenkins **DeploymentConfig** オブジェクトは、Jenkins イメージが変更になると、再デプロイをトリガーします。デフォルトでは、このイメージは、Samples Operator ペイロードの **ImageStream** YAML ファイルの **openshift** namespace にある Jenkins イメージストリームの **jenkins:2** イメージストリームタグによって参照されます。

- OpenShift Container Platform 4.10 以前から 4.11 にアップグレードする場合、非推奨の **maven** および **nodejs** Pod テンプレートはデフォルトのイメージ設定のままです。
- OpenShift Container Platform 4.10 以前から 4.11 にアップグレードする場合、**jenkins-agent-maven** および **jenkins-agent-nodejs** イメージストリームは引き続きクラスターに存在します。これらのイメージストリームを維持するには、次のセクション「**openshift** namespace の **jenkins-agent-maven** および **jenkins-agent-nodejs** イメージストリームはどうなりますか?」を参照してください。

OpenShift Jenkins イメージのサポートマトリックスの変更点は何ですか?

registry.redhat.io レジストリーの **ocp-tools-4** リポジトリーにある新しい各イメージは、OpenShift Container Platform の複数のバージョンをサポートします。Red Hat がこれらの新しいイメージの 1 つを更新すると、すべてのバージョンで同時にイメージが利用可能になります。このような提供の仕組みは、Red Hat がセキュリティーアドバイザリーへの対応としてイメージを更新する際にきわめて有効です。この変更は、まず OpenShift Container Platform 4.11 以降に適用されます。この変更は、最終的に OpenShift Container Platform 4.9 以降に適用される予定です。

以前は、各 Jenkins イメージがサポートするバージョンは、OpenShift Container Platform の 1 つのバージョンだけでした。場合によっては、Red Hat は徐々に各イメージを順次更新することもありました。

OpenShift Jenkins および Jenkins Agent Base ImageStream および ImageStreamTag オブジェクトにはどのような追加機能がありますか?

ペイロード内のイメージストリームから非ペイロードイメージを参照するイメージストリームに移動することで、OpenShift Container Platform は追加のイメージストリームタグを定義できます。Red Hat は、OpenShift Container Platform 4.10 以前に存在する既存の **"value": "jenkins:2"** および **"value": "image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest"** イメージストリームタグに合わせて、一連の新しいイメージストリームタグを作成しました。これらの新規イメージストリームタグは、Jenkins 関連のイメージストリームのメンテナンス方法を改善するための要求の一部に対応します。

新規イメージストリームタグに以下を実行します。

ocp-upgrade-redeploy

OpenShift Container Platform のアップグレード時に Jenkins イメージを更新するには、Jenkins デプロイメント設定でこのイメージストリームタグを使用します。このイメージストリームタグは、**jenkins** イメージストリームの既存の **2** のイメージストリームタグと **jenkins-agent-base-rhel8** イメージストリームの **latest** イメージストリームタグに対応します。これは 1 つの SHA またはイメージダイジェストのみに固有のイメージタグを使用します。Jenkins のセキュリティーアドバイザリーなどに応じて **ocp-tools-4** イメージを変更する際に、Red Hat Engineering チームが Cluster Samples Operator のペイロードを更新します。

user-maintained-upgrade-redeploy

OpenShift Container Platform をアップグレードした後に Jenkins を手動で再デプロイするには、Jenkins デプロイメント設定でこのイメージストリームタグを使用します。このイメージストリームタグは、利用可能な最も具体的なイメージバージョンインジケーターを使用します。Jenkins を再デプロイするときは、**$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift** コマンドを実行します。このコマンドを発行すると、OpenShift Container Platform **ImageStream** コントローラーは **registry.redhat.io** イメージレジストリーにアクセスし、その Jenkins

ImageStreamTag オブジェクトの OpenShift イメージレジストリーのスロットに更新されたイメージを保存します。それ以外の場合は、このコマンドを実行しないと、Jenkins デプロイ設定によって再デプロイがトリガーされません。

scheduled-upgrade-redeploy

Jenkins イメージの最新バージョンがリリースされたときに自動的に再デプロイするには、Jenkins デプロイ設定でこのイメージストリームタグを使用します。このイメージストリームタグは、バッキングイメージの変更をチェックする OpenShift Container Platform イメージストリームコントローラーのイメージストリームタグ機能の定期的なインポートを使用します。たとえば、最近の Jenkins セキュリティーアドバイザリーが原因でイメージが変更になると、OpenShift Container Platform は Jenkins デプロイメント設定の再デプロイメントをトリガーします。次の「関連情報」の「イメージストリームタグの定期的なインポートの設定」を参照してください。

openshift namespace の jenkins-agent-maven および jenkins-agent-nodejs イメージストリームはどうなりますか?

OpenShift Container Platform の OpenShift Jenkins Maven および NodeJS エージェントイメージは、4.10 で非推奨になり、4.11 で OpenShift Container Platform インストールペイロードから削除されました。それらには、**ocp-tools-4** リポジトリーで定義された代替手段がありません。ただし、この問題はサイドカーパターンを使用することで回避できます。次の「関連情報」セクションに記載されている「Jenkins エージェント」のトピックを参照してください。

ただし、Cluster Samples Operator は、以前のリリースで作成された **jenkins-agent-maven** および **jenkins-agent-nodejs** イメージストリームを削除しません。これらは、**registry.redhat.io** 上のそれぞれの OpenShift Container Platform ペイロードイメージのタグを指しています。したがって、次のコマンドを実行して、これらのイメージの更新をプルできます。

```
$ oc import-image jenkins-agent-nodejs -n openshift
```

```
$ oc import-image jenkins-agent-maven -n openshift
```

OpenShift Jenkins はどの OpenShift Container Platform アーキテクチャーとバージョンをサポートしていますか?

Jenkins は、すべての OpenShift Container Platform リリースで次のアーキテクチャーをサポートしています。

- **amd64**
- **arm64**
- **ppc64le**
- **s390x**

ただし、OpenShift Container Platform の Extended Update Support (EUS) リリースでは、**amd64** アーキテクチャーのみが公式にサポートされています。そのため、EUS リリースでは OpenShift Jenkins イメージは **amd64** 専用として提供されます。これは、EUS リリースの場合、OpenShift Container Platform プラットフォーム自体がサポートしているアーキテクチャーが **amd64** だけであるためです。詳細は、Support Matrix for OpenShift Jenkins releases を参照してください。

Red Hat が Jenkins コンテナーイメージを更新したときに、すべての OpenShift Container Platform バージョンで同時にイメージを利用できるようになりますか?

はい、Jenkins コンテナーイメージは四半期ごとに更新され、サポートされているすべての OpenShift Container Platform リリースのすべての Jenkins イメージで更新が利用できるようになります。

リリースされた Jenkins イメージはどのくらいの期間サポートされますか?

Red Hat は、最新のコンテナーイメージで提供される Jenkins コアの最新の長期サポート (LTS) バージョンのみをサポートします。複数のコアバージョンはサポートしていません。当社では、アップストリームコミュニティーによってリリースされた最新の Jenkins LTS バージョンに合わせることを方針としています。

Jenkins のリリースは OpenShift Container Platform のバージョンと対応していますか?

はい。当社はプラットフォームとの対応関係を維持することを目標としています。そのため、Jenkins コントローラーエージェントのイメージは、サポート対象の OpenShift Container Platform リリースごとにビルドおよびテストされます。

Jenkins のリリース予定は OpenShift Container Platform のリリースサイクルと同じですか?

Jenkins は OpenShift Container Platform のコアペイロードの一部ではなくなりました。リリースは別途管理されています。ただし、当社では、OpenShift Container Platform GA リリースから数週間以内に、新しくリリースされた OpenShift Container Platform 用の新しい OpenShift Jenkins イメージを公開することを目指しています。

Red Hat はアップストリームの Jenkins のライフサイクルと LTS バージョンに準拠していますか?

はい。当社はアップストリームの Jenkins のライフサイクルと LTS バージョンに準拠しています。Red Hat は、重要な修正のためにサイクル外のリリースが必要な場合を除き、通常、OpenShift Jenkins イメージの更新を四半期ごとに提供します。

現在の Jenkins LTS バージョンを確認するには、Jenkins Catalog → Packages セクションに移動し、“Jenkins” を検索してください。結果に 2 つのパッケージが表示され、そのうちの 1 つは Jenkins LTS パッケージです。

サポートされていないものは何ですか?

- 現在の OpenShift Jenkins LTS よりも古いバージョンの Jenkins はサポートされていません。
- OpenShift Container Platform 外部での Jenkins の実行はサポートされていません。
- Jenkins の複数のコアバージョンはサポートされていません。OpenShift Jenkins イメージにバンドルされているプラグインには、サポート対象のすべての OpenShift Container Platform リリースで、同じバージョン管理が適用されます。

4.2. JENKINS イメージストリームタグのカスタマイズ

デフォルトのアップグレード動作をオーバーライドし、Jenkins イメージのアップグレード方法を制御するには、Jenkins デプロイメント設定で使用するイメージストリームタグの値を設定します。

デフォルトのアップグレード動作は、Jenkins イメージがインストールペイロードの一部であったときに存在した動作です。**jenkins-rhel.json** イメージストリームファイル内のイメージストリームタグ名 **2** および **ocp-upgrade-redeploy** は、SHA 固有のイメージ参照を使用します。したがって、これらのタグが新しい SHA で更新されると、OpenShift Container Platform イメージ変更コントローラーは、関連するテンプレート (**jenkins-ephemeral.json** や **jenkins-persistent.json** など) から Jenkins デプロイメント設定を自動的に再デプロイします。

新しいデプロイメントの場合、そのデフォルト値をオーバーライドするには、**jenkins-ephemeral.json** Jenkins テンプレートの **JENKINS_IMAGE_STREAM_TAG** の値を変更します。たとえば、**"value": "jenkins:2"** の **2** を次のイメージストリームタグのいずれかに置き換えます。

- デフォルト値 の **ocp-upgrade-redeploy** は、OpenShift Container Platform をアップグレードするときに Jenkins イメージを更新します。
- **user-maintained-upgrade-redeploy** では、OpenShift Container Platform のアップグレード後に **$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift** を実行して、Jenkins を手動で再デプロイする必要があります。
- **scheduled-upgrade-redeploy** は、指定された **<image>:<tag>** の組み合わせの変更を定期的にチェックし、変更があった場合はイメージをアップグレードします。イメージ変更コントローラーは、変更されたイメージをプルし、テンプレートによってプロビジョニングされた Jenkins デプロイ設定を再デプロイします。この定期インポートポリシーの詳細は、次の「関連情報」の「イメージストリームへのタグの追加」を参照してください。

> **注記**
>
> 既存のデプロイメントの現在のアップグレード値をオーバーライドするには、それらのテンプレートパラメーターに対応する環境変数の値を変更します。

前提条件

- OpenShift Container Platform 4.12 で OpenShift Jenkins を実行している。
- OpenShift Jenkins がデプロイされている namespace を知ってる。

手順

- **<namespace>** を OpenShift Jenkins がデプロイされている namespace に置き換え、**<image_stream_tag>** をイメージストリームタグに置き換えて、イメージストリームタグの値を設定します。

  例

  ```
  $ oc patch dc jenkins -p '{"spec":{"triggers":[{"type":"ImageChange","imageChangeParams":
  {"automatic":true,"containerNames":["jenkins"],"from":
  {"kind":"ImageStreamTag","namespace":"<namespace>","name":"jenkins:
  <image_stream_tag>"}}}]}}'
  ```

  ヒント

  または、Jenkins デプロイメント設定 YAML を編集するには、**$ oc edit dc/jenkins -n <namespace>** を入力し、**value: 'jenkins:<image_stream_tag>'** 行を更新します。

4.3. 関連情報

- イメージストリームへのタグの追加
- イメージストリームタグの定期的なインポートの設定
- Jenkins エージェント
- 認定済み **jenkins** イメージ
- 認定済み **jenkins-agent-base** イメージ

- 認定済み **jenkins-agent-maven** イメージ
- 認定済み **jenkins-agent-nodejs** イメージ