OpenShift Container Platform 4.14

Jenkins

Jenkins

Last Updated: 2026-02-19

OpenShift Container Platform 4.14 Jenkins

Jenkins

Legal Notice

Abstract

Jenkins for OpenShift Container Platform

Table of Contents

CHAPTER 1. CONFIGURING JENKINS IMAGES

OpenShift Container Platform provides a container image for running Jenkins. This image provides a Jenkins server instance, which can be used to set up a basic flow for continuous testing, integration, and delivery.

The image is based on the Red Hat Universal Base Images (UBI).

OpenShift Container Platform follows the LTS release of Jenkins. OpenShift Container Platform provides an image that contains Jenkins 2.x.

The OpenShift Container Platform Jenkins images are available on Quay.io or registry.redhat.io.

For example:

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

To use these images, you can either access them directly from these registries or push them into your OpenShift Container Platform container image registry. Additionally, you can create an image stream that points to the image, either in your container image registry or at the external location. Your OpenShift Container Platform resources can then reference the image stream.

But for convenience, OpenShift Container Platform provides image streams in the **openshift** namespace for the core Jenkins image as well as the example Agent images provided for OpenShift Container Platform integration with Jenkins.

1.1. CONFIGURATION AND CUSTOMIZATION

You can manage Jenkins authentication in two ways:

- OpenShift Container Platform OAuth authentication provided by the OpenShift Container Platform Login plugin.
- Standard authentication provided by Jenkins.

1.1.1. OpenShift Container Platform OAuth authentication

OAuth authentication is activated by configuring options on the **Configure Global Security** panel in the Jenkins UI, or by setting the **OPENSHIFT_ENABLE_OAUTH** environment variable on the Jenkins **Deployment configuration** to anything other than **false**. This activates the OpenShift Container Platform Login plugin, which retrieves the configuration information from pod data or by interacting with the OpenShift Container Platform API server.

Valid credentials are controlled by the OpenShift Container Platform identity provider.

Jenkins supports both browser and non-browser access.

Valid users are automatically added to the Jenkins authorization matrix at log in, where OpenShift Container Platform roles dictate the specific Jenkins permissions that users have. The roles used by default are the predefined **admin**, **edit**, and **view**. The login plugin executes self-SAR requests against those roles in the project or namespace that Jenkins is running in.

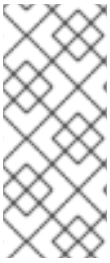
Users with the **admin** role have the traditional Jenkins administrative user permissions. Users with the **edit** or **view** role have progressively fewer permissions.

The default OpenShift Container Platform **admin**, **edit**, and **view** roles and the Jenkins permissions those roles are assigned in the Jenkins instance are configurable.

When running Jenkins in an OpenShift Container Platform pod, the login plugin looks for a config map named **openshift-jenkins-login-plugin-config** in the namespace that Jenkins is running in.

If this plugin finds and can read in that config map, you can define the role to Jenkins Permission mappings. Specifically:

- The login plugin treats the key and value pairs in the config map as Jenkins permission to OpenShift Container Platform role mappings.
- The key is the Jenkins permission group short ID and the Jenkins permission short ID, with those two separated by a hyphen character.
- If you want to add the **Overall Jenkins Administer** permission to an OpenShift Container Platform role, the key should be **Overall-Administer**.
- To get a sense of which permission groups and permissions IDs are available, go to the matrix authorization page in the Jenkins console and IDs for the groups and individual permissions in the table they provide.
- The value of the key and value pair is the list of OpenShift Container Platform roles the permission should apply to, with each role separated by a comma.
- If you want to add the **Overall Jenkins Administer** permission to both the default **admin** and **edit** roles, as well as a new Jenkins role you have created, the value for the key **Overall-Administer** would be **admin,edit,jenkins**.



NOTE

The **admin** user that is pre-populated in the OpenShift Container Platform Jenkins image with administrative privileges is not given those privileges when OpenShift Container Platform OAuth is used. To grant these permissions the OpenShift Container Platform cluster administrator must explicitly define that user in the OpenShift Container Platform identity provider and assigns the **admin** role to the user.

Jenkins users' permissions that are stored can be changed after the users are initially established. The OpenShift Container Platform Login plugin polls the OpenShift Container Platform API server for permissions and updates the permissions stored in Jenkins for each user with the permissions retrieved from OpenShift Container Platform. If the Jenkins UI is used to update permissions for a Jenkins user, the permission changes are overwritten the next time the plugin polls OpenShift Container Platform.

You can control how often the polling occurs with the **OPENSHIFT_PERMISSIONS_POLL_INTERVAL** environment variable. The default polling interval is five minutes.

The easiest way to create a new Jenkins service using OAuth authentication is to use a template.

1.1.2. Jenkins authentication

Jenkins authentication is used by default if the image is run directly, without using a template.

The first time Jenkins starts, the configuration is created along with the administrator user and password. The default user credentials are **admin** and **password**. Configure the default password by setting the **JENKINS_PASSWORD** environment variable when using, and only when using, standard

Jenkins authentication.

Procedure

- Create a Jenkins application that uses standard Jenkins authentication:

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

1.2. JENKINS ENVIRONMENT VARIABLES

The Jenkins server can be configured with the following environment variables:

Variable	Definition	Example values and settings
OPENSHIFT_ENABLE_OAUTH	Determines whether the OpenShift Container Platform Login plugin manages authentication when logging in to Jenkins. To enable, set to **true**.	Default: **false**
JENKINS_PASSWORD	The password for the **admin** user when using standard Jenkins authentication. Not applicable when **OPENSHIFT_ENABLE_OAUTH** is set to **true**.	Default: **password**
JAVA_MAX_HEAP_PARAM, **CONTAINER_HEAP_PERCENT**, **JENKINS_MAX_HEAP_UPPER_BOUND_MB**	These values control the maximum heap size of the Jenkins JVM. If **JAVA_MAX_HEAP_PARAM** is set, its value takes precedence. Otherwise, the maximum heap size is dynamically calculated as **CONTAINER_HEAP_PERCENT** of the container memory limit, optionally capped at **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB. By default, the maximum heap size of the Jenkins JVM is set to 50% of the container memory limit with no cap.	**JAVA_MAX_HEAP_PARAM** example setting: **-Xmx512m** **CONTAINER_HEAP_PERCENT** default: **0.5**, or 50% **JENKINS_MAX_HEAP_UPPER_BOUND_MB** example setting: **512 MiB**

Variable	Definition	Example values and settings
JAVA_INITIAL_HEAP_PARAM, **CONTAINER_INITIAL_PERCENT**	These values control the initial heap size of the Jenkins JVM. If **JAVA_INITIAL_HEAP_PARAM** is set, its value takes precedence. Otherwise, the initial heap size is dynamically calculated as **CONTAINER_INITIAL_PERCENT** of the dynamically calculated maximum heap size. By default, the JVM sets the initial heap size.	**JAVA_INITIAL_HEAP_PARAM** example setting: **-Xms32m** **CONTAINER_INITIAL_PERCENT** example setting: **0.1**, or 10%
CONTAINER_CORE_LIMIT	If set, specifies an integer number of cores used for sizing numbers of internal JVM threads.	Example setting: **2**
JAVA_TOOL_OPTIONS	Specifies options to apply to all JVMs running in this container. It is not recommended to override this value.	Default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**
JAVA_GC_OPTS	Specifies Jenkins JVM garbage collection parameters. It is not recommended to override this value.	Default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**
JENKINS_JAVA_OVERRIDES	Specifies additional options for the Jenkins JVM. These options are appended to all other options, including the Java options above, and may be used to override any of them if necessary. Separate each additional option with a space; if any option contains space characters, escape them with a backslash.	Example settings: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**.
JENKINS_OPTS	Specifies arguments to Jenkins.	

Variable	Definition	Example values and settings
INSTALL_PLUGINS	Specifies additional Jenkins plugins to install when the container is first run or when **OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS** is set to **true**. Plugins are specified as a comma-delimited list of name:version pairs.	Example setting: **git:3.7.0,subversion:2.10.2**.
OPENSHIFT_PERMISSIONS_POLL_INTERVAL	Specifies the interval in milliseconds that the OpenShift Container Platform Login plugin polls OpenShift Container Platform for the permissions that are associated with each user that is defined in Jenkins.	Default: **300000** - 5 minutes
OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG	When running this image with an OpenShift Container Platform persistent volume (PV) for the Jenkins configuration directory, the transfer of configuration from the image to the PV is performed only the first time the image starts because the PV is assigned when the persistent volume claim (PVC) is created. If you create a custom image that extends this image and updates the configuration in the custom image after the initial startup, the configuration is not copied over unless you set this environment variable to **true**.	Default: **false**
OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS	When running this image with an OpenShift Container Platform PV for the Jenkins configuration directory, the transfer of plugins from the image to the PV is performed only the first time the image starts because the PV is assigned when the PVC is created. If you create a custom image that extends this image and updates plugins in the custom image after the initial startup, the plugins are not copied over unless you set this environment variable to **true**.	Default: **false**

Variable	Definition	Example values and settings
ENABLE_FATAL_ERROR_LOG_FILE	When running this image with an OpenShift Container Platform PVC for the Jenkins configuration directory, this environment variable allows the fatal error log file to persist when a fatal error occurs. The fatal error file is saved at **/var/lib/jenkins/logs**.	Default: **false**
AGENT_BASE_IMAGE	Setting this value overrides the image used for the **jnlp** container in the sample Kubernetes plugin pod templates provided with this image. Otherwise, the image from the **jenkins-agent-base-rhel8:latest** image stream tag in the **openshift** namespace is used.	Default: **image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest**
JAVA_BUILDER_IMAGE	Setting this value overrides the image used for the **java-builder** container in the **java-builder** sample Kubernetes plugin pod templates provided with this image. Otherwise, the image from the **java:latest** image stream tag in the **openshift** namespace is used.	Default: **image-registry.openshift-image-registry.svc:5000/openshift/java:latest**
JAVA_FIPS_OPTIONS	Setting this value controls how the JVM operates when running on a FIPS node. For more information, see Configure OpenJDK 11 in FIPS mode.	Default: **-Dcom.redhat.fips=false**

1.3. PROVIDING JENKINS CROSS PROJECT ACCESS

If you are going to run Jenkins somewhere other than your same project, you must provide an access token to Jenkins to access your project.

Procedure

1. Identify the secret for the service account that has appropriate permissions to access the project Jenkins must access:

```
$ oc describe serviceaccount jenkins
```

Example output

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

In this case the secret is named **jenkins-token-uyswp**.

2. Retrieve the token from the secret:

```
$ oc describe secret <secret name from above>
```

Example output

```
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
```

The token parameter contains the token value Jenkins requires to access the project.

1.4. JENKINS CROSS VOLUME MOUNT POINTS

The Jenkins image can be run with mounted volumes to enable persistent storage for the configuration:

- **/var/lib/jenkins** is the data directory where Jenkins stores configuration files, including job definitions.

1.5. CUSTOMIZING THE JENKINS IMAGE THROUGH SOURCE-TO-IMAGE

To customize the official OpenShift Container Platform Jenkins image, you can use the image as a source-to-image (S2I) builder.

You can use S2I to copy your custom Jenkins jobs definitions, add additional plugins, or replace the provided **config.xml** file with your own, custom, configuration.

To include your modifications in the Jenkins image, you must have a Git repository with the following directory structure:

plugins

This directory contains those binary Jenkins plugins you want to copy into Jenkins.

plugins.txt

This file lists the plugins you want to install using the following syntax:

```
pluginId:pluginVersion
```

configuration/jobs

This directory contains the Jenkins job definitions.

configuration/config.xml

This file contains your custom Jenkins configuration.

The contents of the **configuration/** directory is copied to the **/var/lib/jenkins/** directory, so you can also include additional files, such as **credentials.xml**, there.

Sample build configuration customizes the Jenkins image in OpenShift Container Platform

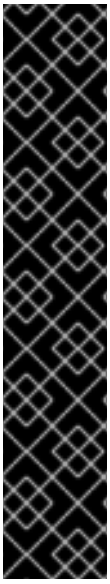
```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:                    1
    git:
      uri: https://github.com/custom/repository
    type: Git
  strategy:                  2
    sourceStrategy:
      from:
        kind: ImageStreamTag
     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

1. The **source** parameter defines the source Git repository with the layout described above.
2. The **strategy** parameter defines the original Jenkins image to use as a source image for the build.
3. The **output** parameter defines the resulting, customized Jenkins image that you can use in deployment configurations instead of the official Jenkins image.

1.6. CONFIGURING THE JENKINS KUBERNETES PLUGIN

The OpenShift Jenkins image includes the preinstalled Kubernetes plugin for Jenkins so that Jenkins agents can be dynamically provisioned on multiple container hosts using Kubernetes and OpenShift Container Platform.

To use the Kubernetes plugin, OpenShift Container Platform provides an OpenShift Agent Base image that is suitable for use as a Jenkins agent.



IMPORTANT

OpenShift Container Platform 4.11 moves the OpenShift Jenkins and OpenShift Agent Base images to the **ocp-tools-4** repository at **registry.redhat.io** so that Red Hat can produce and update the images outside the OpenShift Container Platform lifecycle. Previously, these images were in the OpenShift Container Platform install payload and the **openshift4** repository at **registry.redhat.io**.

The OpenShift Jenkins Maven and NodeJS Agent images were removed from the OpenShift Container Platform 4.11 payload. Red Hat no longer produces these images, and they are not available from the **ocp-tools-4** repository at **registry.redhat.io**. Red Hat maintains the 4.10 and earlier versions of these images for any significant bug fixes or security CVEs, following the OpenShift Container Platform lifecycle policy.

For more information, see the "Important changes to OpenShift Jenkins images" link in the following "Additional resources" section.

The Maven and Node.js agent images are automatically configured as Kubernetes pod template images within the OpenShift Container Platform Jenkins image configuration for the Kubernetes plugin. That configuration includes labels for each image that you can apply to any of your Jenkins jobs under their **Restrict where this project can be run** setting. If the label is applied, jobs run under an OpenShift Container Platform pod running the respective agent image.

IMPORTANT

In OpenShift Container Platform 4.10 and later, the recommended pattern for running Jenkins agents using the Kubernetes plugin is to use pod templates with both **jnlp** and **sidecar** containers. The **jnlp** container uses the OpenShift Container Platform Jenkins Base agent image to facilitate launching a separate pod for your build. The **sidecar** container image has the tools needed to build in a particular language within the separate pod that was launched. Many container images from the Red Hat Container Catalog are referenced in the sample image streams in the **openshift** namespace. The OpenShift Container Platform Jenkins image has a pod template named **java-build** with sidecar containers that demonstrate this approach. This pod template uses the latest Java version provided by the **java** image stream in the **openshift** namespace.

The Jenkins image also provides auto-discovery and auto-configuration of additional agent images for the Kubernetes plugin.

With the OpenShift Container Platform sync plugin, on Jenkins startup, the Jenkins image searches within the project it is running, or the projects listed in the plugin's configuration, for the following items:

- Image streams with the **role** label set to **jenkins-agent**.
- Image stream tags with the **role** annotation set to **jenkins-agent**.
- Config maps with the **role** label set to **jenkins-agent**.

When the Jenkins image finds an image stream with the appropriate label, or an image stream tag with the appropriate annotation, it generates the corresponding Kubernetes plugin configuration. This way, you can assign your Jenkins jobs to run in a pod running the container image provided by the image stream.

The name and image references of the image stream, or image stream tag, are mapped to the name and image fields in the Kubernetes plugin pod template. You can control the label field of the

Kubernetes plugin pod template by setting an annotation on the image stream, or image stream tag object, with the key **agent-label**. Otherwise, the name is used as the label.

NOTE

Do not log in to the Jenkins console and change the pod template configuration. If you do so after the pod template is created, and the OpenShift Container Platform Sync plugin detects that the image associated with the image stream or image stream tag has changed, it replaces the pod template and overwrites those configuration changes. You cannot merge a new configuration with the existing configuration.

Consider the config map approach if you have more complex configuration needs.

When it finds a config map with the appropriate label, the Jenkins image assumes that any values in the key-value data payload of the config map contain Extensible Markup Language (XML) consistent with the configuration format for Jenkins and the Kubernetes plugin pod templates. One key advantage of config maps over image streams and image stream tags is that you can control all the Kubernetes plugin pod template parameters.

Sample config map for jenkins-agent

```
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>
```

The following example shows two containers that reference image streams in the **openshift** namespace. One container handles the JNLP contract for launching Pods as Jenkins Agents. The other container uses an image with tools for building code in a particular coding language:

```
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>
```

NOTE

Do not log in to the Jenkins console and change the pod template configuration. If you do so after the pod template is created, and the OpenShift Container Platform Sync plugin detects that the image associated with the image stream or image stream tag has changed, it replaces the pod template and overwrites those configuration changes. You cannot merge a new configuration with the existing configuration.

Consider the config map approach if you have more complex configuration needs.

After it is installed, the OpenShift Container Platform Sync plugin monitors the API server of OpenShift Container Platform for updates to image streams, image stream tags, and config maps and adjusts the configuration of the Kubernetes plugin.

The following rules apply:

- Removing the label or annotation from the config map, image stream, or image stream tag deletes any existing **PodTemplate** from the configuration of the Kubernetes plugin.
- If those objects are removed, the corresponding configuration is removed from the Kubernetes plugin.
- If you create appropriately labeled or annotated **ConfigMap**, **ImageStream**, or **ImageStreamTag** objects, or add labels after their initial creation, this results in the creation of a **PodTemplate** in the Kubernetes-plugin configuration.
- In the case of the **PodTemplate** by config map form, changes to the config map data for the **PodTemplate** are applied to the **PodTemplate** settings in the Kubernetes plugin configuration. The changes also override any changes that were made to the **PodTemplate** through the Jenkins UI between changes to the config map.

To use a container image as a Jenkins agent, the image must run the agent as an entry point. For more details, see the official Jenkins documentation.

Additional resources

- Important changes to OpenShift Jenkins images

1.7. JENKINS PERMISSIONS

If in the config map the **<serviceAccount>** element of the pod template XML is the OpenShift Container Platform service account used for the resulting pod, the service account credentials are mounted into the pod. The permissions are associated with the service account and control which operations against the OpenShift Container Platform master are allowed from the pod.

Consider the following scenario with service accounts used for the pod, which is launched by the Kubernetes Plugin that runs in the OpenShift Container Platform Jenkins image.

If you use the example template for Jenkins that is provided by OpenShift Container Platform, the **jenkins** service account is defined with the **edit** role for the project Jenkins runs in, and the master Jenkins pod has that service account mounted.

The two default Maven and NodeJS pod templates that are injected into the Jenkins configuration are also set to use the same service account as the Jenkins master.

- Any pod templates that are automatically discovered by the OpenShift Container Platform sync plugin because their image streams or image stream tags have the required label or annotations are configured to use the Jenkins master service account as their service account.
- For the other ways you can provide a pod template definition into Jenkins and the Kubernetes plugin, you have to explicitly specify the service account to use. Those other ways include the Jenkins console, the **podTemplate** pipeline DSL that is provided by the Kubernetes plugin, or labeling a config map whose data is the XML configuration for a pod template.
- If you do not specify a value for the service account, the **default** service account is used.
- Ensure that whatever service account is used has the necessary permissions, roles, and so on defined within OpenShift Container Platform to manipulate whatever projects you choose to manipulate from the within the pod.

1.8. CREATING A JENKINS SERVICE FROM A TEMPLATE

Templates provide parameter fields to define all the environment variables with predefined default values. OpenShift Container Platform provides templates to make creating a new Jenkins service easy. The Jenkins templates should be registered in the default **openshift** project by your cluster administrator during the initial cluster setup.

The two available templates both define deployment configuration and a service. The templates differ in their storage strategy, which affects whether the Jenkins content persists across a pod restart.

NOTE

A pod might be restarted when it is moved to another node or when an update of the deployment configuration triggers a redeployment.

- **jenkins-ephemeral** uses ephemeral storage. On pod restart, all data is lost. This template is only useful for development or testing.
- **jenkins-persistent** uses a Persistent Volume (PV) store. Data survives a pod restart.

To use a PV store, the cluster administrator must define a PV pool in the OpenShift Container Platform deployment.

After you select which template you want, you must instantiate the template to be able to use Jenkins.

Procedure

1. Create a new Jenkins application using one of the following methods:
   - A PV:

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

- Or an **emptyDir** type volume where configuration does not persist across pod restarts:

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

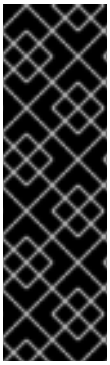
With both templates, you can run **oc describe** on them to see all the parameters available for overriding.

For example:

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

1.9. USING THE JENKINS KUBERNETES PLUGIN

In the following example, the **openshift-jee-sample BuildConfig** object causes a Jenkins Maven agent pod to be dynamically provisioned. The pod clones some Java source code, builds a WAR file, and causes a second **BuildConfig**, **openshift-jee-sample-docker** to run. The second **BuildConfig** layers the new WAR file into a container image.



IMPORTANT

OpenShift Container Platform 4.11 removed the OpenShift Jenkins Maven and NodeJS Agent images from its payload. Red Hat no longer produces these images, and they are not available from the **ocp-tools-4** repository at **registry.redhat.io**. Red Hat maintains the 4.10 and earlier versions of these images for any significant bug fixes or security CVEs, following the OpenShift Container Platform lifecycle policy.

For more information, see the "Important changes to OpenShift Jenkins images" link in the following "Additional resources" section.

Sample BuildConfig that uses the Jenkins Kubernetes plugin

```
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
```

It is also possible to override the specification of the dynamically created Jenkins agent pod. The following is a modification to the preceding example, which overrides the container memory and specifies an environment variable.

Sample BuildConfig that uses the Jenkins Kubernetes plugin, specifying memory limit and environment variable

```
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. A new pod template called **mypod** is defined dynamically. The new pod template name is referenced in the node stanza.
2. The **cloud** value must be set to **openshift**.
3. The new pod template can inherit its configuration from an existing pod template. In this case, inherited from the Maven pod template that is pre-defined by OpenShift Container Platform.
4. This example overrides values in the pre-existing container, and must be specified by name. All Jenkins agent images shipped with OpenShift Container Platform use the Container name **jnlp**.
5. Specify the container image name again. This is a known issue.
6. A memory request of **512 Mi** is specified.
7. A memory limit of **512 Mi** is specified.
8. An environment variable **CONTAINER_HEAP_PERCENT**, with value **0.25**, is specified.
9. The node stanza references the name of the defined pod template.

By default, the pod is deleted when the build completes. This behavior can be modified with the plugin or within a pipeline Jenkinsfile.

Upstream Jenkins has more recently introduced a YAML declarative format for defining a **podTemplate** pipeline DSL in-line with your pipelines. An example of this format, using the sample **java-builder** pod template that is defined in the OpenShift Container Platform Jenkins image:

```
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"
        }
      }
    }
  }
}
```

Additional resources

- Important changes to OpenShift Jenkins images

1.10. JENKINS MEMORY REQUIREMENTS

When deployed by the provided Jenkins Ephemeral or Jenkins Persistent templates, the default memory limit is **1 Gi**.

By default, all other process that run in the Jenkins container cannot use more than a total of **512 MiB** of memory. If they require more memory, the container halts. It is therefore highly recommended that pipelines run external commands in an agent container wherever possible.

And if **Project** quotas allow for it, see recommendations from the Jenkins documentation on what a Jenkins master should have from a memory perspective. Those recommendations proscribe to allocate even more memory for the Jenkins master.

It is recommended to specify memory request and limit values on agent containers created by the Jenkins Kubernetes plugin. Admin users can set default values on a per-agent image basis through the Jenkins configuration. The memory request and limit parameters can also be overridden on a per-container basis.

You can increase the amount of memory available to Jenkins by overriding the **MEMORY_LIMIT** parameter when instantiating the Jenkins Ephemeral or Jenkins Persistent template.

1.11. ADDITIONAL RESOURCES

- See Base image options for more information about the Red Hat Universal Base Images (UBI).
- Important changes to OpenShift Jenkins images

CHAPTER 2. JENKINS AGENT

OpenShift Container Platform provides a base image for use as a Jenkins agent.

The Base image for Jenkins agents does the following:

- Pulls in both the required tools, headless Java, the Jenkins JNLP client, and the useful ones, including **git**, **tar**, **zip**, and **nss**, among others.
- Establishes the JNLP agent as the entry point.
- Includes the **oc** client tool for invoking command-line operations from within Jenkins jobs.
- Provides Dockerfiles for both Red Hat Enterprise Linux (RHEL) and **localdev** images.

IMPORTANT

Use a version of the agent image that is appropriate for your OpenShift Container Platform release version. Embedding an **oc** client version that is not compatible with the OpenShift Container Platform version can cause unexpected behavior.

The OpenShift Container Platform Jenkins image also defines the following sample **java-builder** pod template to illustrate how you can use the agent image with the Jenkins Kubernetes plugin.

The **java-builder** pod template employs two containers: * A **jnlp** container that uses the OpenShift Container Platform Base agent image and handles the JNLP contract for starting and stopping Jenkins agents. * A **java** container that uses the **java** OpenShift Container Platform Sample ImageStream, which contains the various Java binaries, including the Maven binary **mvn**, for building code.

2.1. JENKINS AGENT IMAGES

The OpenShift Container Platform Jenkins agent images are available on Quay.io or registry.redhat.io.

Jenkins images are available through the Red Hat Registry:

```
$ 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>
```

To use these images, you can either access them directly from Quay.io or registry.redhat.io or push them into your OpenShift Container Platform container image registry.

2.2. JENKINS AGENT ENVIRONMENT VARIABLES

Each Jenkins agent container can be configured with the following environment variables.

Variable	Definition	Example values and settings
JAVA_MAX_HEAP_PARAM, **CONTAINER_HEAP_PERCENT**, **JENKINS_MAX_HEAP_UPPER_BOUND_MB**	These values control the maximum heap size of the Jenkins JVM. If **JAVA_MAX_HEAP_PARAM** is set, its value takes precedence. Otherwise, the maximum heap size is dynamically calculated as **CONTAINER_HEAP_PERCENT** of the container memory limit, optionally capped at **JENKINS_MAX_HEAP_UPPER_BOUND_MB** MiB. By default, the maximum heap size of the Jenkins JVM is set to 50% of the container memory limit with no cap.	**JAVA_MAX_HEAP_PARAM** example setting: **-Xmx512m** **CONTAINER_HEAP_PERCENT** default: **0.5**, or 50% **JENKINS_MAX_HEAP_UPPER_BOUND_MB** example setting: **512 MiB**
JAVA_INITIAL_HEAP_PARAM, **CONTAINER_INITIAL_PERCENT**	These values control the initial heap size of the Jenkins JVM. If **JAVA_INITIAL_HEAP_PARAM** is set, its value takes precedence. Otherwise, the initial heap size is dynamically calculated as **CONTAINER_INITIAL_PERCENT** of the dynamically calculated maximum heap size. By default, the JVM sets the initial heap size.	**JAVA_INITIAL_HEAP_PARAM** example setting: **-Xms32m** **CONTAINER_INITIAL_PERCENT** example setting: **0.1**, or 10%
CONTAINER_CORE_LIMIT	If set, specifies an integer number of cores used for sizing numbers of internal JVM threads.	Example setting: **2**
JAVA_TOOL_OPTIONS	Specifies options to apply to all JVMs running in this container. It is not recommended to override this value.	Default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**

Variable	Definition	Example values and settings
JAVA_GC_OPTS	Specifies Jenkins JVM garbage collection parameters. It is not recommended to override this value.	Default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**
JENKINS_JAVA_OVERRIDES	Specifies additional options for the Jenkins JVM. These options are appended to all other options, including the Java options above, and can be used to override any of them, if necessary. Separate each additional option with a space and if any option contains space characters, escape them with a backslash.	Example settings: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**
USE_JAVA_VERSION	Specifies the version of Java version to use to run the agent in its container. The container base image has two versions of java installed: **java-11** and **java-1.8.0**. If you extend the container base image, you can specify any alternative version of java using its associated suffix.	The default value is **java-11**. Example setting: **java-1.8.0**

2.3. JENKINS AGENT MEMORY REQUIREMENTS

A JVM is used in all Jenkins agents to host the Jenkins JNLP agent as well as to run any Java applications such as **javac**, Maven, or Gradle.

By default, the Jenkins JNLP agent JVM uses 50% of the container memory limit for its heap. This value can be modified by the **CONTAINER_HEAP_PERCENT** environment variable. It can also be capped at an upper limit or overridden entirely.

By default, any other processes run in the Jenkins agent container, such as shell scripts or **oc** commands run from pipelines, cannot use more than the remaining 50% memory limit without provoking an OOM kill.

By default, each further JVM process that runs in a Jenkins agent container uses up to 25% of the container memory limit for its heap. It might be necessary to tune this limit for many build workloads.

2.4. JENKINS AGENT GRADLE BUILDS

Hosting Gradle builds in the Jenkins agent on OpenShift Container Platform presents additional complications because in addition to the Jenkins JNLP agent and Gradle JVMs, Gradle spawns a third JVM to run tests if they are specified.

The following settings are suggested as a starting point for running Gradle builds in a memory constrained Jenkins agent on OpenShift Container Platform. You can modify these settings as required.

- Ensure the long-lived Gradle daemon is disabled by adding **org.gradle.daemon=false** to the **gradle.properties** file.
- Disable parallel build execution by ensuring **org.gradle.parallel=true** is not set in the **gradle.properties** file and that **--parallel** is not set as a command-line argument.
- To prevent Java compilations running out-of-process, set **java { options.fork = false }** in the **build.gradle** file.
- Disable multiple additional test processes by ensuring **test { maxParallelForks = 1 }** is set in the **build.gradle** file.
- Override the Gradle JVM memory parameters by the **GRADLE_OPTS**, **JAVA_OPTS** or **JAVA_TOOL_OPTIONS** environment variables.
- Set the maximum heap size and JVM arguments for any Gradle test JVM by defining the **maxHeapSize** and **jvmArgs** settings in **build.gradle**, or through the **-Dorg.gradle.jvmargs** command-line argument.

2.5. JENKINS AGENT POD RETENTION

Jenkins agent pods, are deleted by default after the build completes or is stopped. This behavior can be changed by the Kubernetes plugin pod retention setting. Pod retention can be set for all Jenkins builds, with overrides for each pod template. The following behaviors are supported:

- **Always** keeps the build pod regardless of build result.
- **Default** uses the plugin value, which is the pod template only.
- **Never** always deletes the pod.
- **On Failure** keeps the pod if it fails during the build.

You can override pod retention in the pipeline Jenkinsfile:

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

1 Allowed values for **podRetention** are **never()**, **onFailure()**, **always()**, and **default()**.

WARNING

Pods that are kept might continue to run and count against resource quotas.

CHAPTER 3. MIGRATING FROM JENKINS TO OPENSHIFT PIPELINES OR TEKTON

You can migrate your CI/CD workflows from Jenkins to Red Hat OpenShift Pipelines, a cloud-native CI/CD experience based on the Tekton project.

3.1. COMPARISON OF JENKINS AND OPENSHIFT PIPELINES CONCEPTS

You can review and compare the following equivalent terms used in Jenkins and OpenShift Pipelines.

3.1.1. Jenkins terminology

Jenkins offers declarative and scripted pipelines that are extensible using shared libraries and plugins. Some basic terms in Jenkins are as follows:

- **Pipeline**: Automates the entire process of building, testing, and deploying applications by using Groovy syntax.
- **Node**: A machine capable of either orchestrating or executing a scripted pipeline.
- **Stage**: A conceptually distinct subset of tasks performed in a pipeline. Plugins or user interfaces often use this block to display the status or progress of tasks.
- **Step**: A single task that specifies the exact action to be taken, either by using a command or a script.

3.1.2. OpenShift Pipelines terminology

OpenShift Pipelines uses YAML syntax for declarative pipelines and consists of tasks. Some basic terms in OpenShift Pipelines are as follows:

- **Pipeline**: A set of tasks in a series, in parallel, or both.
- **Task**: A sequence of steps as commands, binaries, or scripts.
- **PipelineRun**: Execution of a pipeline with one or more tasks.
- **TaskRun**: Execution of a task with one or more steps.

NOTE

You can initiate a PipelineRun or a TaskRun with a set of inputs such as parameters and workspaces, and the execution results in a set of outputs and artifacts.

- **Workspace**: In OpenShift Pipelines, workspaces are conceptual blocks that serve the following purposes:
  - Storage of inputs, outputs, and build artifacts.
  - Common space to share data among tasks.

- Mount points for credentials held in secrets, configurations held in config maps, and common tools shared by an organization.

NOTE

In Jenkins, there is no direct equivalent of OpenShift Pipelines workspaces. You can think of the control node as a workspace, as it stores the cloned code repository, build history, and artifacts. When a job is assigned to a different node, the cloned code and the generated artifacts are stored in that node, but the control node maintains the build history.

3.1.3. Mapping of concepts

The building blocks of Jenkins and OpenShift Pipelines are not equivalent, and a specific comparison does not provide a technically accurate mapping. The following terms and concepts in Jenkins and OpenShift Pipelines correlate in general:

Table 3.1. Jenkins and OpenShift Pipelines - basic comparison

Jenkins	OpenShift Pipelines
Pipeline	Pipeline and PipelineRun
Stage	Task
Step	A step in a task

3.2. MIGRATING A SAMPLE PIPELINE FROM JENKINS TO OPENSHIFT PIPELINES

You can use the following equivalent examples to help migrate your build, test, and deploy pipelines from Jenkins to OpenShift Pipelines.

3.2.1. Jenkins pipeline

Consider a Jenkins pipeline written in Groovy for building, testing, and deploying:

```
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 pipeline

To create a pipeline in OpenShift Pipelines that is equivalent to the preceding Jenkins pipeline, you create the following three tasks:

Example **build** task YAML definition file

```
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)
```

Example **test** task YAML definition file

```
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)
```

Example **deploy** task YAML definition file

```
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)
```

You can combine the three tasks sequentially to form a pipeline in OpenShift Pipelines:

Example: OpenShift Pipelines pipeline for building, testing, and deployment

```
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. MIGRATING FROM JENKINS PLUGINS TO TEKTON HUB TASKS

You can extend the capability of Jenkins by using plugins. To achieve similar extensibility in OpenShift Pipelines, use any of the tasks available from Tekton Hub.

For example, consider the git-clone task in Tekton Hub, which corresponds to the git plugin for Jenkins.

Example: git-clone task from Tekton Hub

```
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. EXTENDING OPENSHIFT PIPELINES CAPABILITIES USING CUSTOM TASKS AND SCRIPTS

In OpenShift Pipelines, if you do not find the right task in Tekton Hub, or need greater control over tasks, you can create custom tasks and scripts to extend the capabilities of OpenShift Pipelines.

Example: A custom task for running the `maven test` command

```
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)
```

Example: Run a custom shell script by providing its path

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

Example: Run a custom Python script by writing it in the YAML file

```
...
steps:
  image: python
  script: |
      #!/usr/bin/env python3
      print(“hello from python!”)
...
```

3.5. COMPARISON OF JENKINS AND OPENSHIFT PIPELINES EXECUTION MODELS

Jenkins and OpenShift Pipelines offer similar functions but are different in architecture and execution.

Table 3.2. Comparison of execution models in Jenkins and OpenShift Pipelines

Jenkins	OpenShift Pipelines
Jenkins has a controller node. Jenkins runs pipelines and steps centrally, or orchestrates jobs running in other nodes.	OpenShift Pipelines is serverless and distributed, and there is no central dependency for execution.
Containers are launched by the Jenkins controller node through the pipeline.	OpenShift Pipelines adopts a 'container-first' approach, where every step runs as a container in a pod (equivalent to nodes in Jenkins).
Extensibility is achieved by using plugins.	Extensibility is achieved by using tasks in Tekton Hub or by creating custom tasks and scripts.

3.6. EXAMPLES OF COMMON USE CASES

Both Jenkins and OpenShift Pipelines offer capabilities for common CI/CD use cases, such as:

- Compiling, building, and deploying images using Apache Maven
- Extending the core capabilities by using plugins
- Reusing shareable libraries and custom scripts

3.6.1. Running a Maven pipeline in Jenkins and OpenShift Pipelines

You can use Maven in both Jenkins and OpenShift Pipelines workflows for compiling, building, and deploying images. To map your existing Jenkins workflow to OpenShift Pipelines, consider the following examples:

Example: Compile and build an image and deploy it to OpenShift using Maven in Jenkins

```
#!/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' }
```

Example: Compile and build an image and deploy it to OpenShift using Maven in OpenShift Pipelines.

```
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. Extending the core capabilities of Jenkins and OpenShift Pipelines by using plugins

Jenkins has the advantage of a large ecosystem of numerous plugins developed over the years by its extensive user base. You can search and browse the plugins in the Jenkins Plugin Index.

OpenShift Pipelines also has many tasks developed and contributed by the community and enterprise users. A publicly available catalog of reusable OpenShift Pipelines tasks are available in the Tekton Hub.

In addition, OpenShift Pipelines incorporates many of the plugins of the Jenkins ecosystem within its core capabilities. For example, authorization is a critical function in both Jenkins and OpenShift Pipelines. While Jenkins ensures authorization using the Role-based Authorization Strategy plugin, OpenShift Pipelines uses OpenShift's built-in Role-based Access Control system.

3.6.3. Sharing reusable code in Jenkins and OpenShift Pipelines

Jenkins shared libraries provide reusable code for parts of Jenkins pipelines. The libraries are shared between Jenkinsfiles to create highly modular pipelines without code repetition.

Although there is no direct equivalent of Jenkins shared libraries in OpenShift Pipelines, you can achieve similar workflows by using tasks from the Tekton Hub in combination with custom tasks and scripts.

3.7. ADDITIONAL RESOURCES

- Understanding OpenShift Pipelines
- Role-based Access Control

CHAPTER 4. IMPORTANT CHANGES TO OPENSHIFT JENKINS IMAGES

IMPORTANT

Openshift Jenkins receives periodic updates from Jenkins LTS releases and associated plugins. These updates may include bug fixes, security vulnerability patches, and occasionally new features. However, Red Hat does not plan to introduce further enhancements or major changes to the functionality or contents of the OpenShift Jenkins container image, other than updates to its dependent plugins.

Additionally, the following three Red Hat-maintained Jenkins plugins are now in maintenance mode. Only critical bug fixes will be addressed, and no new enhancements or feature development are planned.

- Jenkins Client Plugin
- Jenkins Login Plugin
- Jenkins Sync Plugin

OpenShift Container Platform 4.11 moves the OpenShift Jenkins and OpenShift Agent Base images to the **ocp-tools-4** repository at **registry.redhat.io**. It also removes the OpenShift Jenkins Maven and NodeJS Agent images from its payload:

- OpenShift Container Platform 4.11 moves the OpenShift Jenkins and OpenShift Agent Base images to the **ocp-tools-4** repository at **registry.redhat.io** so that Red Hat can produce and update the images outside the OpenShift Container Platform lifecycle. Previously, these images were in the OpenShift Container Platform install payload and the **openshift4** repository at **registry.redhat.io**.
- OpenShift Container Platform 4.10 deprecated the OpenShift Jenkins Maven and NodeJS Agent images. OpenShift Container Platform 4.11 removes these images from its payload. Red Hat no longer produces these images, and they are not available from the **ocp-tools-4** repository at **registry.redhat.io**. Red Hat maintains the 4.10 and earlier versions of these images for any significant bug fixes or security CVEs, following the OpenShift Container Platform lifecycle policy.

These changes support the OpenShift Container Platform 4.10 recommendation to use multiple container Pod Templates with the Jenkins Kubernetes Plugin.

4.1. RELOCATION OF OPENSHIFT JENKINS IMAGES

OpenShift Container Platform 4.11 makes significant changes to the location and availability of specific OpenShift Jenkins images. Additionally, you can configure when and how to update these images.

What stays the same with the OpenShift Jenkins images?

- The Cluster Samples Operator manages the **ImageStream** and **Template** objects for operating the OpenShift Jenkins images.
- By default, the Jenkins **DeploymentConfig** object from the Jenkins pod template triggers a redeployment when the Jenkins image changes. By default, this image is referenced by the

jenkins:2 image stream tag of Jenkins image stream in the **openshift** namespace in the **ImageStream** YAML file in the Samples Operator payload.

- If you upgrade from OpenShift Container Platform 4.10 and earlier to 4.11, the deprecated **maven** and **nodejs** pod templates are still in the default image configuration.

- If you upgrade from OpenShift Container Platform 4.10 and earlier to 4.11, the **jenkins-agent-maven** and **jenkins-agent-nodejs** image streams still exist in your cluster. To maintain these image streams, see the following section, "What happens with the **jenkins-agent-maven** and **jenkins-agent-nodejs** image streams in the **openshift** namespace?"

What changes in the support matrix of the OpenShift Jenkins image?

Each new image in the **ocp-tools-4** repository in the **registry.redhat.io** registry supports multiple versions of OpenShift Container Platform. When Red Hat updates one of these new images, it is simultaneously available for all versions. This availability is ideal when Red Hat updates an image in response to a security advisory. Initially, this change applies to OpenShift Container Platform 4.11 and later. It is planned that this change will eventually apply to OpenShift Container Platform 4.9 and later.

Previously, each Jenkins image supported only one version of OpenShift Container Platform and Red Hat might update those images sequentially over time.

What additions are there with the OpenShift Jenkins and Jenkins Agent Base ImageStream and ImageStreamTag objects?

By moving from an in-payload image stream to an image stream that references non-payload images, OpenShift Container Platform can define additional image stream tags. Red Hat has created a series of new image stream tags to go along with the existing **"value": "jenkins:2"** and **"value": "image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest"** image stream tags present in OpenShift Container Platform 4.10 and earlier. These new image stream tags address some requests to improve how the Jenkins-related image streams are maintained.

About the new image stream tags:

ocp-upgrade-redeploy

To update your Jenkins image when you upgrade OpenShift Container Platform, use this image stream tag in your Jenkins deployment configuration. This image stream tag corresponds to the existing **2** image stream tag of the **jenkins** image stream and the **latest** image stream tag of the **jenkins-agent-base-rhel8** image stream. It employs an image tag specific to only one SHA or image digest. When the **ocp-tools-4** image changes, such as for Jenkins security advisories, Red Hat Engineering updates the Cluster Samples Operator payload.

user-maintained-upgrade-redeploy

To manually redeploy Jenkins after you upgrade OpenShift Container Platform, use this image stream tag in your Jenkins deployment configuration. This image stream tag uses the least specific image version indicator available. When you redeploy Jenkins, run the following command: **$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift**. When you issue this command, the OpenShift Container Platform **ImageStream** controller accesses the **registry.redhat.io** image registry and stores any updated images in the OpenShift image registry's slot for that Jenkins **ImageStreamTag** object. Otherwise, if you do not run this command, your Jenkins deployment configuration does not trigger a redeployment.

scheduled-upgrade-redeploy

To automatically redeploy the latest version of the Jenkins image when it is released, use this image stream tag in your Jenkins deployment configuration. This image stream tag uses the periodic importing of image stream tags feature of the OpenShift Container Platform image stream

controller, which checks for changes in the backing image. If the image changes, for example, due to a recent Jenkins security advisory, OpenShift Container Platform triggers a redeployment of your Jenkins deployment configuration. See "Configuring periodic importing of image stream tags" in the following "Additional resources."

What happens with the jenkins-agent-maven and jenkins-agent-nodejs image streams in the openshift namespace?

The OpenShift Jenkins Maven and NodeJS Agent images for OpenShift Container Platform were deprecated in 4.10, and are removed from the OpenShift Container Platform install payload in 4.11. They do not have alternatives defined in the **ocp-tools-4** repository. However, you can work around this by using the sidecar pattern described in the "Jenkins agent" topic mentioned in the following "Additional resources" section.

However, the Cluster Samples Operator does not delete the **jenkins-agent-maven** and **jenkins-agent-nodejs** image streams created by prior releases, which point to the tags of the respective OpenShift Container Platform payload images on **registry.redhat.io**. Therefore, you can pull updates to these images by running the following commands:

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

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

What OpenShift Container Platform architectures and versions does OpenShift Jenkins support?

Jenkins supports the following architectures across OpenShift Container Platform releases:

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

However, for OpenShift Container Platform Extended Update Support (EUS) releases, only the **amd64** architecture is officially supported. As a result, OpenShift Jenkins images are shipped exclusively for **amd64** on these releases. This is because the OpenShift Container Platform platform itself supports only the **amd64** architecture for EUS releases. For more information, see Support Matrix for OpenShift Jenkins releases.

When Red Hat updates Jenkins container images, are they available for all OpenShift Container Platform versions simultaneously?

Yes, Jenkins container images are updated on a quarterly basis, and the updates are made available for all supported Jenkins images across all supported OpenShift Container Platform releases.

How long are the released Jenkins images supported?

Red Hat supports only the latest Long-Term Support (LTS) version of the Jenkins core, as provided in our latest container images. We do not support multiple core versions. Our policy is to align with the latest Jenkins LTS version released by the upstream community.

Does the Jenkins release align with OpenShift Container Platform versions?

Yes. Our goal is to maintain platform alignment. This means that Jenkins controller and agent images are built and tested for each supported OpenShift Container Platform releases.

Are Jenkins release timelines aligned with OpenShift Container Platform release cycles?

Jenkins is no longer part of the OpenShift Container Platform core payload. Releases are managed separately. However, our intent is to publish updated OpenShift Jenkins images for newly released OpenShift Container Platform releases within a few weeks of the OpenShift Container Platform GA release.

Does Red Hat follow the upstream Jenkins lifecycle and LTS versions?

Yes. We align with the Jenkins upstream lifecycle and follow the LTS version. Red Hat typically ships OpenShift Jenkins image updates quarterly unless a critical fix requires an out-of-cycle release.

To verify the current Jenkins LTS version: - Navigate to the Jenkins Catalog → Packages section - Search for “Jenkins” - The result will show two packages, one of which is the Jenkins LTS package.

What is not supported?

- Jenkins versions older than the current OpenShift Jenkins LTS are not supported.
- Running Jenkins outside of OpenShift Container Platform is not supported.
- Multiple core versions of Jenkins are not supported. Plugins bundled with our OpenShift Jenkins images follow the same versioning across all supported OpenShift Container Platform releases.

4.2. CUSTOMIZING THE JENKINS IMAGE STREAM TAG

To override the default upgrade behavior and control how the Jenkins image is upgraded, you set the image stream tag value that your Jenkins deployment configurations use.

The default upgrade behavior is the behavior that existed when the Jenkins image was part of the install payload. The image stream tag names, **2** and **ocp-upgrade-redeploy**, in the **jenkins-rhel.json** image stream file use SHA-specific image references. Therefore, when those tags are updated with a new SHA, the OpenShift Container Platform image change controller automatically redeploys the Jenkins deployment configuration from the associated templates, such as **jenkins-ephemeral.json** or **jenkins-persistent.json**.

For new deployments, to override that default value, you change the value of the **JENKINS_IMAGE_STREAM_TAG** in the **jenkins-ephemeral.json** Jenkins template. For example, replace the **2** in **"value": "jenkins:2"** with one of the following image stream tags:

- **ocp-upgrade-redeploy**, the default value, updates your Jenkins image when you upgrade OpenShift Container Platform.
- **user-maintained-upgrade-redeploy** requires you to manually redeploy Jenkins by running **$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift** after upgrading OpenShift Container Platform.
- **scheduled-upgrade-redeploy** periodically checks the given **<image>:<tag>** combination for changes and upgrades the image when it changes. The image change controller pulls the changed image and redeploys the Jenkins deployment configuration provisioned by the templates. For more information about this scheduled import policy, see the "Adding tags to image streams" in the following "Additional resources."

NOTE

To override the current upgrade value for existing deployments, change the values of the environment variables that correspond to those template parameters.

Prerequisites

- You are running OpenShift Jenkins on OpenShift Container Platform 4.14.
- You know the namespace where OpenShift Jenkins is deployed.

Procedure

- Set the image stream tag value, replacing **<namespace>** with namespace where OpenShift Jenkins is deployed and **<image_stream_tag>** with an image stream tag:

  Example

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

  TIP

  Alternatively, to edit the Jenkins deployment configuration YAML, enter **$ oc edit dc/jenkins -n <namespace>** and update the **value: 'jenkins:<image_stream_tag>'** line.

4.3. ABOUT THE OPENSHIFT CLI TOOL IN OPENSHIFT JENKINS IMAGES

If your Jenkins pipelines require a **specific** OpenShift CLI (**oc**) version for compatibility or reproducibility, you must explicitly configure the desired client version in the Jenkins pipeline DSL.

Starting with 4.12, the OpenShift Jenkins container images ship with the **latest available version** of the **oc** CLI tool bundled inside the image, rather than the client version matching the cluster release.

If your Jenkins pipelines require a **specific oc** client version for compatibility or reproducibility, you must explicitly configure the desired client version in the Jenkins pipeline DSL.

4.4. OPENSHIFT JENKINS RELEASE COMPARED TO BUNDLED oc CLIENT VERSION TABLE

Use the following table to understand which version of OpenShift CLI (**oc**) is shipped with your OpenShift Jenkins images.

OpenShift Jenkins release	Default **oc** version	Bundled **oc** version
4.12	4.13	4.12, 4.13

OpenShift Jenkins release	Default **oc** version	Bundled **oc** version
4.13	4.13	4.12, 4.13
4.14	4.15	4.14, 4.15
4.15	4.15	4.14, 4.15
4.16	4.20	4.16, 4.17, 4.18, 4.19, 4.20
4.17	4.20	4.16, 4.17, 4.18, 4.19, 4.20
4.18	4.20	4.16, 4.17, 4.18, 4.19, 4.20
4.19	4.20	4.16, 4.17, 4.18, 4.19, 4.20
4.20	4.20	4.16, 4.17, 4.18, 4.19, 4.20

4.5. SPECIFYING A FIXED oc CLIENT VERSION FOR OPENSHIFT JENKINS IMAGES

You can ensure that your Jenkins pipeline uses your specified **oc** client version with the Jenkins container image by configuring the version you require.

Procedure

- Define the **oc** tool version explicitly in the pipeline configuration to use a specific OpenShift client version in a Jenkins pipeline as shown in the following example:

Example pipeline configuration:

```
pipeline {
    agent any

    tools {
        oc 'oc-4.14'
    }

    stages {
        stage('Version') {
            steps {
                sh 'oc version'
            }
        }
    }
}
```

4.6. ADDITIONAL RESOURCES

- Adding tags to image streams
- Configuring periodic importing of image stream tags
- Jenkins agent
- Certified **jenkins** images
- Certified **jenkins-agent-base** images
- Certified **jenkins-agent-maven** images
- Certified **jenkins-agent-nodejs** images