Deploying Jenkins on OpenShift

Objectives

After completing this section, you should be able to:

* Deploy Jenkins on OpenShift using standard templates and image streams.
* Deploy a CI/CD development pipeline on Jenkins.
* Manage Jenkins projects and builds using the Jenkins web UI.
* Create API tokens for scripts that automate Jenkins tasks.

Introducing Red Hat Jenkins Images and Templates for OpenShift

Red Hat provides several supported Jenkins images as part of Red Hat OpenShift Container Platform, including:

**registry.redhat.io/openshift4/ose-jenkins**

A Jenkins master image that is preconfigured with a number of Jenkins plug-ins that integrate with OpenShift.

**registry.redhat.io/openshift4/ose-jenkins-agent-maven**

A Jenkins agent image that provides Java development tools.

**registry.redhat.io/openshift4/ose-jenkins-agent-nodejs**

A Jenkins agent image that provides Node.js development tools.

**registry.redhat.io/openshift4/ose-jenkins-agent-base**

A base image for building custom agent images. Use this base image to create a new Jenkins agent image if the previous agent images do not provide the tools that you need. All of the Red Hat Jenkins agent images also include the OpenShift CLI because it is required by the OpenShift Pipeline DSL plug-in.

Issues related to these images, such as security vulnerabilities, fall under the Service-Level Agreement (SLA) of the Red Hat OpenShift Container Platform if you deploy these images on OpenShift clusters. Other ways of deploying Jenkins, including using these same images under a local container run time, are supported on a best-effort basis.

OpenShift also provides predefined image streams for the master, Java agent, and Node.js agent in the openshift namespace.

The recommended way to deploy Jenkins on OpenShift is using the predefined templates in the openshift namespace, among them:

* jenkins-ephemeral deploys a Jenkins server using ephemeral storage for quick testing.
* jenkins-persistent deploys a Jenkins server using persistent storage for development and production environments.

There are also versions of these templates that deploy Jenkins integrated with OpenShift cluster Alerting and Metering. In the Red Hat OpenShift Container Platform 4.4 release, these monitoring templates have known issues with fixes expected in version 4.5.

Describing Jenkins Plug-ins For OpenShift

The Jenkins master image from Red Hat includes several plug-ins that allow Jenkins to work when integrated with an OpenShift cluster:

**Kubernetes plug-in**

Creates agents on-demand as pods.

**Kubernetes Credentials plug-in**

Authenticates to OpenShift using credentials from a service account.

**OpenShift Sync plug-in**

Creates Jenkins credentials, accounts, and agents from OpenShift image streams, configuration maps, service accounts, and secrets.

**OpenShift Jenkins Pipeline DSL**

Also known as *OpenShift Client plug-in*, provides imperative DSL statements that deploy applications and manage OpenShift resources.

These plug-ins are Open Source software, so nothing prevents you from adding these plug-ins to an external Jenkins instance, running outside of OpenShift. The Jenkins instance could manage both projects that target OpenShift, and also projects that target other deployment platforms.

However, remember that the Red Hat OpenShift Platform Service-Level Agreement (SLA) only covers scenarios where Jenkins is deployed on OpenShift using Red Hat container images for Jenkins. Support for Jenkins servers deployed outside of OpenShift, and for the OpenShift Jenkins plug-ins deployed outside of Red Hat Jenkins images, is best-effort only.

If you require additional Jenkins plug-ins that do not come preinstalled with Red Hat Jenkins container images, the product documentation describes how to use Source-to-Image (S2I) to build child Jenkins images that are fully supported, excepting issues related to the plug-ins that you add, which are given best-effort support.

Deploying Jenkins on OpenShift Using Standard Templates

All Jenkins templates provided with OpenShift create a deployment configuration to manage an Jenkins master pod. These templates do not deploy any agent pod. As required by Jenkins builds, these are created on demand and the Jenkins master takes the role of a controller for its agent pods.

If you choose a persistent template to deploy Jenkins, then it also creates a persistent volume claim for the Jenkins configuration folder. If an external artifact server, such as a Nexus server or a container image registry, is unavailable, then you can use that same volume to store build artifacts.

If you run builds on the master node, then those builds reuse the volume to store workspace folders. If you run builds on agent pods, then those builds use ephemeral storage for workspace folders so that every build starts from a clean state. If you must save artifacts from builds running on agent pods, then your pipeline must send artifacts for storage on an external server or the master node.

Do not scale the Jenkins deployment configuration to multiple replica pods because Jenkins does not support multiple instances writing to the same configuration folder.

That is, High Availability (HA) for a Jenkins instance on OpenShift comes from Kubernetes ability to restart a pod managed in a deployment configuration, and from the storage provider’s ability to retain data in persistent volumes.

The standard templates define parameters and default values for persistent volume size and memory requirements, among others. Small Jenkins instances require no changes to these default parameter values and can be deployed by creating a project and then running:

**[user@demo ~]$ oc new-app --template jenkins-persistent**

The standard templates also create a service account named jenkins for all interactions with the OpenShift cluster. The Jenkins service account is assigned the edit role on its OpenShift project, so it is ready to deploy applications to the same project that runs the Jenkins instance.

If your Jenkins instance requires additional rights, then you must add more project or cluster roles to that service account, for example, to allow Jenkins to create resources in other projects.

Deployment Topologies for Jenkins on OpenShift

In theory, a single, large Jenkins instance could manage all projects in all OpenShift clusters for an organization. Some organizations prefer this kind of central CI/CD server that is strictly managed by a corporate IT operations team. This large Jenkins instance could hold credentials to authenticate to multiple OpenShift clusters.

Because such a large Jenkins instance may be hard to manage, many organizations run multiple Jenkins instances in a single cluster or across multiple OpenShift clusters.

It is also easier to manage permissions for these Jenkins instances if they are in different projects.

A common pattern is: each development group inside an organization manages its own Jenkins instance, and that instance does not manage projects from other groups.

Thus, the IT operations team is free from the burden of managing Jenkins instances for development teams. OpenShift role-based access permissions, combined with OpenShift resource limits and resource quotas, are sufficient to ensure that these development teams cannot break their clusters with their pipelines. Also, developers can grant access to Jenkins to projects that they own.

Organizations that use Jenkins to implement GitOps workflows can create one or more dedicated Jenkins instances with either cluster administrator privileges or custom roles that allow limited access to a few cluster operators. Organizations restrict access to these instances to cluster administrators.

Mapping OpenShift Roles to Jenkins User Permissions

Jenkins provides authentication and authorization mechanisms that can be mostly ignored when using images from Red Hat. These images automatically translate standard OpenShift project roles into Jenkins permissions as follows:

* admin: manage the Jenkins instance.
* edit: create and edit projects in the Jenkins instance.
* view: view status, logs, and artifacts of builds from the Jenkins instance.

When an OpenShift user logs in to Jenkins, the OpenShift Sync plug-in automatically defines a Jenkins user by concatenating the OpenShift user name and its roles. The same Jenkins plug-in manages translation of OpenShift role bindings to Jenkins permissions.

Creating a Pipeline Project Using the Jenkins Web UI

Older Red Hat OpenShift Container Platform releases supported embedding Jenkins pipelines inside a Build Configuration resource, using the pipeline strategy. That feature, and the integration of these pipeline build configurations with the OpenShift web console, allowed OpenShift users to avoid the Jenkins web UI for basic tasks such as creating a pipeline project and running builds.

Red Hat OpenShift Container Platform 4.3 deprecated the pipeline build strategy because build configurations are being replaced by a new build system and operator, integrated with the OpenShift Pipelines feature. Note that the Jenkins container images, templates, and plug-ins are not deprecated. Only the integration of Jenkins and OpenShift build configurations is deprecated.

Jenkins users are advised to either use the Jenkins web UI to manage pipeline projects and builds or to add plug-ins, such as the Configuration as Code (CasC) plug-in.

To access the Jenkins web interface, use the host name of the single route created by the Jenkins template. Open the host name in a web browser to display the OpenShift log in page. After successfully logging in, the Jenkins Dashboard displays.

The **New Item** link in the Jenkins Dashboard page allows you to create many project types and other Jenkins configuration items, such as folders to group projects and projects of different types.

The two most common project types are:

**Pipeline**

Runs a pipeline taking as input a single branch from a version control system repository.

**Multibranch pipeline**

Automatically creates new projects when new branches are detected in a version control system repository. All these projects share the same pipeline definition that must be flexible enough to avoid conflicts between builds in different branches.

Managing Pipeline Builds Using the Jenkins Web UI

The Jenkins web UI allows creating, changing, and deleting projects. It also allows starting, stopping, deleting, and inspecting builds.

A Jenkins instance also provides an alternative Ajax-based web UI called *Blue Ocean* for monitoring pipelines and managing builds. Although Blue Ocean is considered more developer-friendly, it does not have capabilities to manage Jenkins servers. The standard Jenkins web UI is a traditional web application and is required to administer a Jenkins server.

The main elements of the Jenkins web UI are:

* The initial page, which is called the Jenkins **Dashboard**. From there you can navigate to items (projects and folders).
* A bread crumb trail that shows your context within the UI. The **Jenkins** link always returns you to the Jenkins **Dashboard**.
* A side bar menu with links. These links change according to the context within the UI: Dashboard, folder, project, or build.

|  |
| --- |
|  |

Figure 4.4: Jenkins web UI bread crumb and side bar menu

The body of the Jenkins Dashboard page shows current folders and projects. If you click a folder, then you see its nested projects and folders. If you click a project, then you see a list of builds for that project.

If you click a build of a pipeline project, then you see the **Stage View** of that build; each stage is colored green or red depending on its success (green) or failed (red) status. You can click each stage to see its logs, or click **Console output** to see the complete logs of the entire build.

Multibranch pipeline projects are actually folders with autogenerated pipeline projects for each branch, but project settings are available only from the Multibranch pipeline folder.

Generating Jenkins API Tokens

To use either the Jenkins API or the Jenkins CLI, you require an API token. That token replaces the password in an HTTP BASIC authentication header, for example:

**[user@demo ~] curl --user '*jenkins-user*:*token*' \**

**> https://*jenkins-host*/*resource-path***

Jenkins users must keep the token value safe because it never expires. It is not possible to retrieve the token value after it is generated. Jenkins associates each token with a name so that if you suspect that a token is compromised, then you can delete it.

To generate the token, enter the Jenkins user configuration page by clicking the user name close to the **log out** link, and then clicking **Configure**. From this page you can also list and delete tokens previously generated for your Jenkins user.

**REFERENCES**

For more information about the Jenkins master container image, refer to the *Configuring Jenkins images* section in the *Using images* chapter in the Red Hat OpenShift Container Platform 4.4 *Images* documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.4/html-single/images/index#images-other-jenkins>

For more information about the Jenkins agent container images, refer to the *Jenkins agent* section in the *Using images* chapter in the Red Hat OpenShift Container Platform 4.4 *Images* documentation at <https://access.redhat.com/documentation/en-us/openshift_container_platform/4.4/html-single/images/index#images-other-jenkins-agent>

[Kubernetes plug-in for Jenkins](https://github.com/jenkinsci/kubernetes-plugin)

[Kubernetes Credentials plug-in for Jenkins](https://github.com/jenkinsci/kubernetes-credentials-plugin)

[OpenShift Sync plug-in for Jenkins](https://github.com/jenkinsci/openshift-sync-plugin)

[OpenShift Jenkins Pipeline (DSL) plug-in](https://github.com/jenkinsci/openshift-client-plugin)