OpenShift Container Platform 4.20

AI workloads

Running AI workloads on OpenShift Container Platform

Last Updated: 2026-01-20

OpenShift Container Platform 4.20 AI workloads

Running AI workloads on OpenShift Container Platform

Legal Notice

Abstract

This document provides information about running artificial intelligence (AI) workloads on an OpenShift Container Platform cluster. It includes details on how to enable large-scale AI training workloads to run reliably across nodes.

Table of Contents

CHAPTER 1. OVERVIEW OF AI WORKLOADS ON OPENSHIFT CONTAINER PLATFORM

OpenShift Container Platform provides a secure, scalable foundation for running artificial intelligence (AI) workloads across training, inference, and data science workflows.

1.1. OPERATORS FOR RUNNING AI WORKLOADS

You can use Operators to run artificial intelligence (AI) and machine learning (ML) workloads on OpenShift Container Platform. With Operators, you can build a customized environment that meets your specific AI/ML requirements while continuing to use OpenShift Container Platform as the core platform for your applications.

OpenShift Container Platform provides several Operators that can help you run AI workloads:

Red Hat build of Kueue

You can use Red Hat build of Kueue to provide structured queues and prioritization so that workloads are handled fairly and efficiently. Without proper prioritization, important jobs might be delayed while less critical jobs occupy resources.
For more information, see "Introduction to Red Hat build of Kueue".

Leader Worker Set Operator

You can use the Leader Worker Set Operator to enable large-scale AI inference workloads to run reliably across nodes with synchronization between leader and worker processes. Without proper coordination, large training runs might fail or stall.
For more information, see "Leader Worker Set Operator overview".

JobSet Operator (Technology Preview)

You can use the JobSet Operator to easily manage and run large-scale, coordinated workloads like high-performance computing (HPC) and AI training. The JobSet Operator can help you gain fast recovery and efficient resource use through features like multi-template job support and stable networking.
For more information, see "JobSet Operator overview".

Additional resources

- Introduction to Red Hat build of Kueue
- Leader Worker Set Operator overview
- JobSet Operator overview

CHAPTER 2. RED HAT BUILD OF KUEUE

2.1. INTRODUCTION TO RED HAT BUILD OF KUEUE

Red Hat build of Kueue is a Kubernetes-native system that manages access to resources for jobs. Red Hat build of Kueue can determine when a job waits, is admitted to start by creating pods, or should be *preempted*, meaning that active pods for that job are deleted.

NOTE

In the context of Red Hat build of Kueue, a job can be defined as a one-time or on-demand task that runs to completion.

Red Hat build of Kueue is based on the Kueue open source project.

Red Hat build of Kueue is compatible with environments that use heterogeneous, elastic resources. This means that the environment has many different resource types, and those resources are capable of dynamic scaling.

Red Hat build of Kueue does not replace any existing components in a Kubernetes cluster, but instead integrates with the existing Kubernetes API server, scheduler, and cluster autoscaler components.

Red Hat build of Kueue supports all-or-nothing semantics. This means that either an entire job with all of its components is admitted to the cluster, or the entire job is rejected if it does not fit on the cluster.

2.1.1. Personas

Different personas exist in a Red Hat build of Kueue workflow.

Batch administrators

Batch administrators manage the cluster infrastructure and establish quotas and queues.

Batch users

Batch users run jobs on the cluster. Examples of batch users might be researchers, AI/ML engineers, or data scientists.

Serving users

Serving users run jobs on the cluster. For example, to expose a trained AI/ML model for inference.

Platform developers

Platform developers integrate Red Hat build of Kueue with other software. They might also contribute to the Kueue open source project.

2.1.2. Workflow overview

The Red Hat build of Kueue workflow can be described at a high level as follows:

1. Batch administrators create and configure **ResourceFlavor**, **LocalQueue**, and **ClusterQueue** resources.
2. User personas create jobs on the cluster.
3. The Kubernetes API server validates and accepts job data.

4. Red Hat build of Kueue admits jobs based on configured options, such as order or quota. It injects affinity into the job by using resource flavors, and creates a **Workload** object that corresponds to each job.
5. The applicable controller for the job type creates pods.
6. The Kubernetes scheduler assigns pods to a node in the cluster.
7. The Kubernetes cluster autoscaler provisions more nodes as required.

2.2. RELEASE NOTES

Red Hat build of Kueue is released as an Operator that is supported on OpenShift Container Platform.

2.2.1. Compatible environments

Before you install Red Hat build of Kueue, review this section to ensure that your cluster meets the requirements.

2.2.1.1. Supported architectures

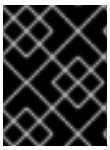
Red Hat build of Kueue version 1.1 and later is supported on the following architectures:

- ARM64
- 64-bit x86
- ppc64le (IBM Power®)
- s390x (IBM Z®)

2.2.1.2. Supported platforms

Red Hat build of Kueue version 1.1 and later is supported on the following platforms:

- OpenShift Container Platform
- Hosted control planes for OpenShift Container Platform



IMPORTANT

Currently, Red Hat build of Kueue is not supported on Red Hat build of MicroShift (MicroShift).

2.2.2. Release notes for Red Hat build of Kueue version 1.2

Red Hat build of Kueue version 1.2 is a generally available release that is supported on OpenShift Container Platform versions 4.18 and later. Red Hat build of Kueue version 1.2 uses Kueue version 0.14.

2.2.2.1. New features and enhancements

Monitoring of pending workloads

Red Hat build of Kueue version 1.2 provides the **VisibilityOnDemand** feature to monitor the pipeline of pending jobs in the cluster queue and the local queue, and help users to estimate when their jobs will start. For more information, see Monitoring pending workloads.

2.2.2.2. Fixed issues

Custom resources are not deleted properly when you uninstall Red Hat build of Kueue

After you uninstall the Red Hat Build of Kueue Operator using the **Delete all operand instances for this operator** option in the OpenShift Container Platform web console, Red Hat build of Kueue custom resources were attempted to be deleted. With this release, they are not considered for deletion.
(OCPBUGS-62254)

Documentation error in previous versions of Red Hat build of Kueue

In Creating a Kueue custom resource, the optional workload types **Pod**, **Deployment**, **StatefulSet** were omitted. They are now included.
(OCPBUGS-62877)

Red Hat build of Kueue metrics were not being exposed to Prometheus from version 1.1

Prometheus was not scraping metrics from the Operator's controller, even though the ServiceMonitor and RBAC resources were successfully created as part of the Operator installation. As a result, none of the Kueue metrics were available in the cluster monitoring stack.
The metrics service added during the installation was configured with an incorrect port reference, causing Prometheus to fail in scraping metrics from the Kueue endpoint. The port name has been updated with the correct port name.

(OCPBUGS-63441)

2.2.2.3. Known issues

Reconcile jobs only in opt-in namespaces

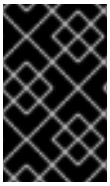
OpenShift Container Platform allows reconciliation of **Job** resources that have the **kueue.x-k8s.io/queue-name** label, even if these resources are in namespaces which are not configured to opt in to being managed by OpenShift Container Platform. This is inconsistent with the behavior for other core integrations like pods, deployments, and stateful sets, which are only reconciled if they are in namespaces which have been configured to opt in to being managed by OpenShift Container Platform by adding the **kueue.openshift.io/managed=true**.
(OCPBUGS-58205)

Kueue CR description reads as "Not available" in the OpenShift Container Platform web console

After installing Red Hat build of Kueue, in the **Operator details** view, the description for the **Kueue** CR reads as "Not available". This issue does not affect or degrade the Red Hat build of Kueue Operator functionality.
(OCPBUGS-62185)

2.2.3. Release notes for Red Hat build of Kueue version 1.1

Red Hat build of Kueue version 1.1 is a generally available release that is supported on OpenShift Container Platform versions 4.18 and later. Red Hat build of Kueue version 1.1 uses Kueue version 0.12.



IMPORTANT

If you have a previously installed version of Red Hat build of Kueue on your cluster, you must uninstall the Operator and manually install version 1.1. For information see Upgrading Red Hat build of Kueue .

2.2.3.1. New features and enhancements

Configure a default local queue

A default local queue serves as the local queue for newly created jobs that do not have the **kueue.x-k8s.io/queue-name** label. After you create a default local queue, any new jobs created in the namespace without a **kueue.x-k8s.io/queue-name** label automatically update to have the **kueue.x-k8s.io/queue-name: default** label.
(RFE-7615)

Multi-architecture and Hosted control planes support

With this release, Red Hat build of Kueue is supported on multiple different architectures, including ARM64, 64-bit x86, ppc64le (IBM Power®), and s390x (IBM Z®), as well as on Hosted control planes for OpenShift Container Platform.
(OCPSTRAT-2103)

(OCPSTRAT-2106)

2.2.3.2. Fixed issues

You can create a Kueue custom resource by using the OpenShift Container Platform web console

Before this update, if you tried to use the OpenShift Container Platform web console to create a **Kueue** custom resource (CR) by using the form view, the web console showed an error and the resource could not be created. With this release, the default namespace was removed from the **Kueue** CR template. As a result, you can use the OpenShift Container Platform web console to create a **Kueue** CR by using the form view.
(OCPBUGS-58118)

2.2.3.3. Known issues

Kueue CR description reads as "Not available" in the OpenShift Container Platform web console

After you install Red Hat build of Kueue, in the **Operator details** view, the description for the **Kueue** CR reads as "Not available". This issue does not affect or degrade the Red Hat build of Kueue Operator functionality.
(OCPBUGS-62185)

Custom resources are not deleted properly when you uninstall Red Hat build of Kueue

After you uninstall the Red Hat Build of Kueue Operator using the **Delete all operand instances for this operator** option in the OpenShift Container Platform web console, some Red Hat build of Kueue custom resources are not fully deleted. These resources can be viewed in the **Installed Operators** view with the status **Resource is being deleted**. As a workaround, you can manually delete the resource finalizers to remove them fully.
(OCPBUGS-62254)

2.2.4. Release notes for Red Hat build of Kueue version 1.0.1

Red Hat build of Kueue version 1.0.1 is a patch release that is supported on OpenShift Container Platform versions 4.18 and 4.19 on the 64-bit x86 architecture.

Red Hat build of Kueue version 1.0.1 uses Kueue version 0.11.

2.2.4.1. Bug fixes in Red Hat build of Kueue version 1.0.1

- Previously, leader election for Red Hat build of Kueue was not configured to tolerate disruption, which resulted in frequent crashing. With this release, the leader election values for Red Hat build of Kueue have been updated to match the durations recommended for OpenShift Container Platform. (OCPBUGS-58496)
- Previously, the **ReadyReplicas** count was not set in the reconciler, which meant that the Red Hat build of Kueue Operator status would report that there were no replicas ready. With this release, the **ReadyReplicas** count is based on the number of ready replicas for the deployment, which ensures that the Operator shows as ready in the OpenShift Container Platform console when the **kueue-controller-manager** pods are ready. (OCPBUGS-59261)
- Previously, when the **Kueue** custom resource (CR) was deleted from the **openshift-kueue-operator** namespace, the **kueue-manager-config** config map was not deleted automatically and could remain in the namespace. With this release, the **kueue-manager-config** config map, **kueue-webhook-server-cert** secret, and **metrics-server-cert** secret are deleted automatically when the **Kueue** CR is deleted. (OCPBUGS-57960)

2.2.5. Release notes for Red Hat build of Kueue version 1.0

Red Hat build of Kueue version 1.0 is a generally available release that is supported on OpenShift Container Platform versions 4.18 and 4.19 on the 64-bit x86 architecture. Red Hat build of Kueue version 1.0 uses Kueue version 0.11.

2.2.5.1. New features and enhancements

Role-based access control (RBAC)

Role-based access control (RBAC) enables you to control which types of users can create which types of Red Hat build of Kueue resources.

Configure resource quotas

Configuring resource quotas by creating cluster queues, resource flavors, and local queues enables you to control the amount of resources used by user-submitted jobs and workloads.

Control job and workload management

Labeling namespaces and configuring label policies enable you to control which jobs and workloads are managed by Red Hat build of Kueue.

Share borrowable resources between queues

Configuring cohorts, fair sharing, and gang scheduling settings enable you to share unused, borrowable resources between queues.

2.2.5.2. Known issues

Jobs in all namespaces are reconciled if they have the kueue.x-k8s.io/queue-name label

Red Hat build of Kueue uses the **managedJobsNamespaceSelector** configuration field, so that administrators can configure which namespaces opt in to be managed by Red Hat build of Kueue.

Because namespaces must be manually configured to opt in to being managed by Red Hat build of Kueue, resources in system or third-party namespaces are not impacted or managed by Red Hat build of Kueue.
The behavior in Red Hat build of Kueue 1.0 allows reconciliation of **Job** resources that have the **kueue.x-k8s.io/queue-name** label, even if these resources are in namespaces that are not configured to opt in to being managed by Red Hat build of Kueue. This is inconsistent with the behavior for other core integrations like pods, deployments, and stateful sets, which are only reconciled if they are in namespaces that have been configured to opt in to being managed by Red Hat build of Kueue.

(OCPBUGS-58205)

You cannot create a Kueue custom resource by using the OpenShift Container Platform web console

If you try to use the OpenShift Container Platform web console to create a **Kueue** custom resource (CR) by using the form view, the web console shows an error and the resource cannot be created. As a workaround, use the YAML view to create a **Kueue** CR instead.
(OCPBUGS-58118)

2.3. INSTALLING RED HAT BUILD OF KUEUE

You can install Red Hat build of Kueue by using the Red Hat Build of Kueue Operator in OperatorHub.

2.3.1. Compatible environments

Before you install Red Hat build of Kueue, review this section to ensure that your cluster meets the requirements.

2.3.1.1. Supported architectures

Red Hat build of Kueue version 1.1 and later is supported on the following architectures:

- ARM64
- 64-bit x86
- ppc64le (IBM Power®)
- s390x (IBM Z®)

2.3.1.2. Supported platforms

Red Hat build of Kueue version 1.1 and later is supported on the following platforms:

- OpenShift Container Platform
- Hosted control planes for OpenShift Container Platform

IMPORTANT

Currently, Red Hat build of Kueue is not supported on Red Hat build of MicroShift (MicroShift).

2.3.2. Installing the Red Hat Build of Kueue Operator

You can install the Red Hat Build of Kueue Operator on a OpenShift Container Platform cluster by using the OperatorHub in the web console.

Prerequisites

- You have administrator permissions on a OpenShift Container Platform cluster.
- You have access to the OpenShift Container Platform web console.
- You have installed and configured the cert-manager Operator for Red Hat OpenShift for your cluster.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Choose **Red Hat Build of Kueue Operator**from the list of available Operators, and click **Install**.

Verification

- Go to **Operators → Installed Operators** and confirm that the **Red Hat Build of Kueue Operator** is listed with **Status** as **Succeeded**.

Additional resources

- Installing the cert-manager Operator for Red Hat OpenShift

2.3.3. Upgrading Red Hat build of Kueue

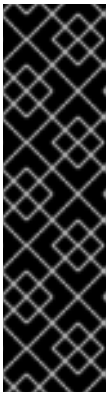
If you have previously installed Red Hat build of Kueue, you must manually upgrade your deployment to the latest version to use the latest bug fixes and feature enhancements.

Prerequisites

- You have installed a previous version of Red Hat build of Kueue.
- You are logged in to the OpenShift Container Platform web console with cluster administrator permissions.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → Installed Operators**, then select **Red Hat build of Kueue** from the list.
2. From the **Actions** drop-down menu, select **Uninstall Operator**.
3. The **Uninstall Operator?** dialog box opens. Click **Uninstall**.



IMPORTANT

Selecting the **Delete all operand instances for this operator**checkbox before clicking **Uninstall** deletes all existing resources from the cluster, including:

- The **Kueue** CR
- Any cluster queues, local queues, or resource flavors that you have created

Leave this box unchecked when upgrading your cluster to retain your created resources.

4. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
5. Choose **Red Hat Build of Kueue Operator**from the list of available Operators, and click **Install**.

Verification

1. Go to **Operators → Installed Operators**.
2. Confirm that the **Red Hat Build of Kueue Operator**is listed with **Status** as **Succeeded**.
3. Confirm that the version shown under the Operator name in the list is the latest version.

2.3.4. Creating a Kueue custom resource

After you have installed the Red Hat Build of Kueue Operator, you must create a **Kueue** custom resource (CR) to configure your installation.

Prerequisites

Ensure that you have completed the following prerequisites:

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have cluster administrator permissions and the **kueue-batch-admin-role** role.
- You have access to the OpenShift Container Platform web console.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → Installed Operators**.
2. In the **Provided APIs** table column, click **Kueue**. This takes you to the **Kueue** tab of the **Operator details** page.
3. Click **Create Kueue**. This takes you to the **Create Kueue** YAML view.
4. Enter the details for your **Kueue** CR.

Example Kueue CR

```
apiVersion: kueue.openshift.io/v1
kind: Kueue
metadata:
  labels:
```

```
    app.kubernetes.io/name: kueue-operator
    app.kubernetes.io/managed-by: kustomize
  name: cluster (1)
  namespace: openshift-kueue-operator
spec:
  managementState: Managed
  config:
    integrations:
      frameworks: (2)
      - BatchJob
    preemption:
      preemptionPolicy: Classical (3)
# ...
```

(1) The name of the **Kueue** CR must be **cluster**.

(2) If you want to configure Red Hat build of Kueue for use with other workload types, add those types here. The default configuration is **BatchJob**. Additional types are **Pod**, **Deployment**, and **StatefulSet**.

(3) Optional: If you want to configure fair sharing for Red Hat build of Kueue, set the **preemptionPolicy** value to **FairSharing**. The default setting in the **Kueue** CR is **Classical** preemption.

5. Click **Create**.

Verification

- After you create the **Kueue** CR, the web console brings you to the **Operator details** page, where you can see the CR in the list of **Kueues**.
- Optional: If you have the OpenShift CLI (**oc**) installed, you can run the following command and observe the output to confirm that your **Kueue** CR has been created successfully:

```
$ oc get kueue
```

Example output

```
NAME      AGE
cluster   4m
```

2.3.5. Labeling namespaces to allow Red Hat build of Kueue to manage jobs

The Red Hat build of Kueue Operator uses an opt-in webhook mechanism to ensure that policies are only enforced for the jobs and namespaces that it is expected to target.

You must label the namespaces where you want Red Hat build of Kueue to manage jobs with the **kueue.openshift.io/managed=true** label.

Prerequisites

- You have cluster administrator permissions.

- The Red Hat build of Kueue Operator is installed on your cluster, and you have created a **Kueue** custom resource (CR).
- You have installed the OpenShift CLI (**oc**).

Procedure

- Add the **kueue.openshift.io/managed=true** label to a namespace by running the following command:

```
$ oc label namespace <namespace> kueue.openshift.io/managed=true
```

When you add this label, you instruct the Red Hat build of Kueue Operator that the namespace is managed by its webhook admission controllers. As a result, any Red Hat build of Kueue resources within that namespace are properly validated and mutated.

2.4. INSTALLING RED HAT BUILD OF KUEUE IN A DISCONNECTED ENVIRONMENT

Before you can install Red Hat build of Kueue on a disconnected OpenShift Container Platform cluster, you must enable Operator Lifecycle Manager (OLM) in disconnected environments by completing the following steps:

- Disable the default remote OperatorHub sources for OLM.
- Use a workstation with full internet access to create and push local mirrors of the OperatorHub content to a mirror registry.
- Configure OLM to install and manage Operators from local sources on the mirror registry instead of the default remote sources.

After enabling OLM in a disconnected environment, you can continue to use your unrestricted workstation to keep your local OperatorHub sources updated as newer versions of Operators are released.

For full documentation on completing these steps, see the OpenShift Container Platform documentation on Using Operator Lifecycle Manager in disconnected environments .

2.4.1. Compatible environments

Before you install Red Hat build of Kueue, review this section to ensure that your cluster meets the requirements.

2.4.1.1. Supported architectures

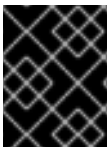
Red Hat build of Kueue version 1.1 and later is supported on the following architectures:

- ARM64
- 64-bit x86
- ppc64le (IBM Power®)
- s390x (IBM Z®)

2.4.1.2. Supported platforms

Red Hat build of Kueue version 1.1 and later is supported on the following platforms:

- OpenShift Container Platform
- Hosted control planes for OpenShift Container Platform



IMPORTANT

Currently, Red Hat build of Kueue is not supported on Red Hat build of MicroShift (MicroShift).

2.4.2. Installing the Red Hat Build of Kueue Operator

You can install the Red Hat Build of Kueue Operator on a OpenShift Container Platform cluster by using the OperatorHub in the web console.

Prerequisites

- You have administrator permissions on a OpenShift Container Platform cluster.
- You have access to the OpenShift Container Platform web console.
- You have installed and configured the cert-manager Operator for Red Hat OpenShift for your cluster.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Choose **Red Hat Build of Kueue Operator**from the list of available Operators, and click **Install**.

Verification

- Go to **Operators → Installed Operators** and confirm that the **Red Hat Build of Kueue Operator** is listed with **Status** as **Succeeded**.

Additional resources

- Installing the cert-manager Operator for Red Hat OpenShift

2.4.3. Upgrading Red Hat build of Kueue

If you have previously installed Red Hat build of Kueue, you must manually upgrade your deployment to the latest version to use the latest bug fixes and feature enhancements.

Prerequisites

- You have installed a previous version of Red Hat build of Kueue.
- You are logged in to the OpenShift Container Platform web console with cluster administrator permissions.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → Installed Operators**, then select **Red Hat build of Kueue** from the list.
2. From the **Actions** drop-down menu, select **Uninstall Operator**.
3. The **Uninstall Operator?** dialog box opens. Click **Uninstall**.

IMPORTANT

Selecting the **Delete all operand instances for this operator** checkbox before clicking **Uninstall** deletes all existing resources from the cluster, including:

- The **Kueue** CR
- Any cluster queues, local queues, or resource flavors that you have created

Leave this box unchecked when upgrading your cluster to retain your created resources.

4. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
5. Choose **Red Hat Build of Kueue Operator** from the list of available Operators, and click **Install**.

Verification

1. Go to **Operators → Installed Operators**.
2. Confirm that the **Red Hat Build of Kueue Operator** is listed with **Status** as **Succeeded**.
3. Confirm that the version shown under the Operator name in the list is the latest version.

2.4.4. Creating a Kueue custom resource

After you have installed the Red Hat Build of Kueue Operator, you must create a **Kueue** custom resource (CR) to configure your installation.

Prerequisites

Ensure that you have completed the following prerequisites:

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have cluster administrator permissions and the **kueue-batch-admin-role** role.
- You have access to the OpenShift Container Platform web console.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → Installed Operators**.
2. In the **Provided APIs** table column, click **Kueue**. This takes you to the **Kueue** tab of the **Operator details** page.
3. Click **Create Kueue**. This takes you to the **Create Kueue** YAML view.

4. Enter the details for your **Kueue** CR.

Example Kueue CR

```
apiVersion: kueue.openshift.io/v1
kind: Kueue
metadata:
  labels:
    app.kubernetes.io/name: kueue-operator
    app.kubernetes.io/managed-by: kustomize
  name: cluster (1)
  namespace: openshift-kueue-operator
spec:
  managementState: Managed
  config:
    integrations:
      frameworks: (2)
      - BatchJob
    preemption:
      preemptionPolicy: Classical (3)
# ...
```

(1) The name of the **Kueue** CR must be **cluster**.

(2) If you want to configure Red Hat build of Kueue for use with other workload types, add those types here. The default configuration is **BatchJob**. Additional types are **Pod**, **Deployment**, and **StatefulSet**.

(3) Optional: If you want to configure fair sharing for Red Hat build of Kueue, set the **preemptionPolicy** value to **FairSharing**. The default setting in the **Kueue** CR is **Classical** preemption.

5. Click **Create**.

Verification

- After you create the **Kueue** CR, the web console brings you to the **Operator details** page, where you can see the CR in the list of **Kueues**.
- Optional: If you have the OpenShift CLI (**oc**) installed, you can run the following command and observe the output to confirm that your **Kueue** CR has been created successfully:

```
$ oc get kueue
```

Example output

```
NAME      AGE
cluster   4m
```

2.4.5. Labeling namespaces to allow Red Hat build of Kueue to manage jobs

The Red Hat build of Kueue Operator uses an opt-in webhook mechanism to ensure that policies are only enforced for the jobs and namespaces that it is expected to target.

You must label the namespaces where you want Red Hat build of Kueue to manage jobs with the **kueue.openshift.io/managed=true** label.

Prerequisites

- You have cluster administrator permissions.
- The Red Hat build of Kueue Operator is installed on your cluster, and you have created a **Kueue** custom resource (CR).
- You have installed the OpenShift CLI (**oc**).

Procedure

- Add the **kueue.openshift.io/managed=true** label to a namespace by running the following command:

```
$ oc label namespace <namespace> kueue.openshift.io/managed=true
```

When you add this label, you instruct the Red Hat build of Kueue Operator that the namespace is managed by its webhook admission controllers. As a result, any Red Hat build of Kueue resources within that namespace are properly validated and mutated.

2.5. CONFIGURING ROLE-BASED PERMISSIONS

The following procedures provide information about how you can configure role-based access control (RBAC) for your Red Hat build of Kueue deployment. These RBAC permissions determine which types of users can create which types of Red Hat build of Kueue objects.

2.5.1. Cluster roles

The Red Hat build of Kueue Operator deploys **kueue-batch-admin-role** and **kueue-batch-user-role** cluster roles by default.

kueue-batch-admin-role

This cluster role includes the permissions to manage cluster queues, local queues, workloads, and resource flavors.

kueue-batch-user-role

This cluster role includes the permissions to manage jobs and to view local queues and workloads.

2.5.2. Configuring permissions for batch administrators

You can configure permissions for batch administrators by binding the **kueue-batch-admin-role** cluster role to a user or group of users.

Prerequisites

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have cluster administrator permissions.

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **ClusterRoleBinding** object as a YAML file:

   #### Example ClusterRoleBinding object

   ```
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: kueue-admins 1
   subjects: 2
   - kind: User
     name: admin@example.com
     apiGroup: rbac.authorization.k8s.io
   roleRef: 3
     kind: ClusterRole
     name: kueue-batch-admin-role
     apiGroup: rbac.authorization.k8s.io
   ```

   1 Provide a name for the **ClusterRoleBinding** object.

   2 Add details about which user or group of users you want to provide user permissions for.

   3 Add details about the **kueue-batch-admin-role** cluster role.

2. Apply the **ClusterRoleBinding** object:

   ```
   $ oc apply -f <filename>.yaml
   ```

Verification

- You can verify that the **ClusterRoleBinding** object was applied correctly by running the following command and verifying that the output contains the correct information for the **kueue-batch-admin-role** cluster role:

  ```
  $ oc describe clusterrolebinding.rbac
  ```

  #### Example output

  ```
  ...
  Name:         kueue-batch-admin-role
  Labels:       app.kubernetes.io/name=kueue
  Annotations:  <none>
  Role:
    Kind:  ClusterRole
    Name:  kueue-batch-admin-role
  Subjects:
    Kind           Name                    Namespace
  ```

```
----          ----                  ---------
User          admin@example.com     admin-namespace
...
```

2.5.3. Configuring permissions for users

You can configure permissions for Red Hat build of Kueue users by binding the **kueue-batch-user-role** cluster role to a user or group of users.

Prerequisites

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have cluster administrator permissions.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **RoleBinding** object as a YAML file:

   Example ClusterRoleBinding object

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kueue-users (1)
  namespace: user-namespace (2)
subjects: (3)
- kind: Group
  name: team-a@example.com
  apiGroup: rbac.authorization.k8s.io
roleRef: (4)
  kind: ClusterRole
  name: kueue-batch-user-role
  apiGroup: rbac.authorization.k8s.io
```

   (1) Provide a name for the **RoleBinding** object.

   (2) Add details about which namespace the **RoleBinding** object applies to.

   (3) Add details about which user or group of users you want to provide user permissions for.

   (4) Add details about the **kueue-batch-user-role** cluster role.

2. Apply the **RoleBinding** object:

```
$ oc apply -f <filename>.yaml
```

Verification

- You can verify that the **RoleBinding** object was applied correctly by running the following command and verifying that the output contains the correct information for the **kueue-batch-user-role** cluster role:

```
$ oc describe rolebinding.rbac
```

Example output

```
...
Name:         kueue-users
Labels:       app.kubernetes.io/name=kueue
Annotations:  <none>
Role:
  Kind:  ClusterRole
  Name:  kueue-batch-user-role
Subjects:
  Kind           Name                    Namespace
  ----           ----                    ---------
  Group          team-a@example.com      user-namespace
...
```

2.5.4. Additional resources

- Using RBAC to define and apply permissions
- Glossary of common terms for OpenShift Container Platform authentication and authorization

2.6. CONFIGURING QUOTAS

As an administrator, you can use Red Hat build of Kueue to configure quotas to optimize resource allocation and system throughput for user workloads. You can configure quotas for compute resources such as CPU, memory, pods, and GPU.

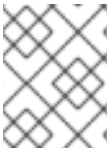
You can configure quotas in Red Hat build of Kueue by completing the following steps:

1. Configure a cluster queue.
2. Configure a resource flavor.
3. Configure a local queue.

Users can then submit their workloads to the local queue.

2.6.1. Configuring a cluster queue

A cluster queue is a cluster-scoped resource, represented by a **ClusterQueue** object, that governs a pool of resources such as CPU, memory, and pods. Cluster queues can be used to define usage limits, quotas for resource flavors, order of consumption, and fair sharing rules.



NOTE

The cluster queue is not ready for use until a **ResourceFlavor** object has also been configured.

Prerequisites

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have cluster administrator permissions or the **kueue-batch-admin-role** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **ClusterQueue** object as a YAML file:

Example of a basic ClusterQueue object using a single resource flavor

```
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: cluster-queue
spec:
  namespaceSelector: {} (1)
  resourceGroups:
  - coveredResources: ["cpu", "memory", "pods", "foo.com/gpu"] (2)
    flavors:
    - name: "default-flavor" (3)
      resources: (4)
      - name: "cpu"
        nominalQuota: 9
      - name: "memory"
        nominalQuota: 36Gi
      - name: "pods"
        nominalQuota: 5
      - name: "foo.com/gpu"
        nominalQuota: 100
```

(1) Defines which namespaces can use the resources governed by this cluster queue. An empty **namespaceSelector** as shown in the example means that all namespaces can use these resources.

(2) Defines the resource types governed by the cluster queue. This example **ClusterQueue** object governs CPU, memory, pod, and GPU resources.

(3) Defines the resource flavor that is applied to the resource types listed. In this example, the **default-flavor** resource flavor is applied to CPU, memory, pod, and GPU resources.

(4) Defines the resource requirements for admitting jobs. This example cluster queue only admits jobs if the following conditions are met:

- The sum of the CPU requests is less than or equal to 9.
- The sum of the memory requests is less than or equal to 36Gi.
- The total number of pods is less than or equal to 5.
- The sum of the GPU requests is less than or equal to 100.

2. Apply the **ClusterQueue** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

Next steps

The cluster queue is not ready for use until a **ResourceFlavor** object has also been configured.

2.6.2. Configuring a resource flavor

After you have configured a **ClusterQueue** object, you can configure a **ResourceFlavor** object.

Resources in a cluster are typically not homogeneous. If the resources in your cluster are homogeneous, you can use an empty **ResourceFlavor** instead of adding labels to custom resource flavors.

You can use a custom **ResourceFlavor** object to represent different resource variations that are associated with cluster nodes through labels, taints, and tolerations. You can then associate workloads with specific node types to enable fine-grained resource management.

Prerequisites

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have cluster administrator permissions or the **kueue-batch-admin-role** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **ResourceFlavor** object as a YAML file:

Example of an empty ResourceFlavor object

```
apiVersion: kueue.x-k8s.io/v1beta1
kind: ResourceFlavor
metadata:
  name: default-flavor
```

Example of a custom ResourceFlavor object

```
apiVersion: kueue.x-k8s.io/v1beta1
kind: ResourceFlavor
metadata:
  name: "x86"
spec:
  nodeLabels:
    cpu-arch: x86
```

2. Apply the **ResourceFlavor** object by running the following command:

```
$ oc apply -f <filename>.yaml
```

2.6.3. Configuring a local queue

A local queue is a namespaced object, represented by a **LocalQueue** object, that groups closely related workloads that belong to a single namespace.

As an administrator, you can configure a **LocalQueue** object to point to a cluster queue. This allocates resources from the cluster queue to workloads in the namespace specified in the **LocalQueue** object.

Prerequisites

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have cluster administrator permissions or the **kueue-batch-admin-role** role.
- You have installed the OpenShift CLI (**oc**).
- You have created a **ClusterQueue** object.

Procedure

1. Create a **LocalQueue** object as a YAML file:

   Example of a basic LocalQueue object

   ```
   apiVersion: kueue.x-k8s.io/v1beta1
   kind: LocalQueue
   metadata:
     namespace: team-namespace
     name: user-queue
   spec:
     clusterQueue: cluster-queue
   ```

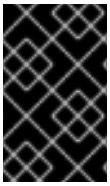
2. Apply the **LocalQueue** object by running the following command:

   ```
   $ oc apply -f <filename>.yaml
   ```

2.6.4. Configuring a default local queue

As a cluster administrator, you can improve quota enforcement in your cluster by managing all jobs in selected namespaces without needing to explicitly label each job. You can do this by creating a default local queue.

A default local queue serves as the local queue for newly created jobs that do not have the **kueue.x-k8s.io/queue-name** label. After you create a default local queue, any new jobs created in the namespace without a **kueue.x-k8s.io/queue-name** label automatically update to have the **kueue.x-k8s.io/queue-name: default** label.



IMPORTANT

Preexisting jobs in a namespace are not affected when you create a default local queue. If jobs already exist in the namespace before you create the default local queue, you must label those jobs explicitly to assign them to a queue.

Prerequisites

- You have installed Red Hat build of Kueue version 1.1 on your cluster.
- You have cluster administrator permissions or the **kueue-batch-admin-role** role.
- You have installed the OpenShift CLI (**oc**).
- You have created a **ClusterQueue** object.

Procedure

1. Create a **LocalQueue** object named **default** as a YAML file:

   Example of a default LocalQueue object

   ```
   apiVersion: kueue.x-k8s.io/v1beta1
   kind: LocalQueue
   metadata:
     namespace: team-namespace
     name: default
   spec:
     clusterQueue: cluster-queue
   ```

2. Apply the **LocalQueue** object by running the following command:

   ```
   $ oc apply -f <filename>.yaml
   ```

Verification

1. Create a job in the same namespace as the default local queue.
2. Observe that the job updates with the **kueue.x-k8s.io/queue-name: default** label.

2.6.5. Additional resources

- RBAC permissions
- Kubernetes documentation about cluster queues

2.7. MANAGING JOBS AND WORKLOADS

Red Hat build of Kueue does not directly manipulate jobs that are created by users. Instead, Kueue manages **Workload** objects that represent the resource requirements of a job. Red Hat build of Kueue automatically creates a workload for each job, and syncs any decisions and statuses between the two objects.

2.7.1. Labeling namespaces to allow Red Hat build of Kueue to manage jobs

The Red Hat build of Kueue Operator uses an opt-in webhook mechanism to ensure that policies are only enforced for the jobs and namespaces that it is expected to target.

You must label the namespaces where you want Red Hat build of Kueue to manage jobs with the **kueue.openshift.io/managed=true** label.

Prerequisites

- You have cluster administrator permissions.
- The Red Hat build of Kueue Operator is installed on your cluster, and you have created a **Kueue** custom resource (CR).
- You have installed the OpenShift CLI (**oc**).

Procedure

- Add the **kueue.openshift.io/managed=true** label to a namespace by running the following command:

```
$ oc label namespace <namespace> kueue.openshift.io/managed=true
```

When you add this label, you instruct the Red Hat build of Kueue Operator that the namespace is managed by its webhook admission controllers. As a result, any Red Hat build of Kueue resources within that namespace are properly validated and mutated.

2.7.2. Configuring label policies for jobs

The **spec.config.workloadManagement.labelPolicy** spec in the **Kueue** custom resource (CR) is an optional field that controls how Red Hat build of Kueue decides whether to manage or ignore different jobs. The allowed values are **QueueName**, **None** and empty (**""**).

If the **labelPolicy** setting is omitted or empty (**""**), the default policy is that Red Hat build of Kueue manages jobs that have a **kueue.x-k8s.io/queue-name** label, and ignores jobs that do not have the **kueue.x-k8s.io/queue-name** label. This is the same workflow as if the **labelPolicy** is set to **QueueName**.

If the **labelPolicy** setting is set to **None**, jobs are managed by Red Hat build of Kueue even if they do not have the **kueue.x-k8s.io/queue-name** label.

Example workloadManagement spec configuration

```
apiVersion: kueue.openshift.io/v1
kind: Kueue
metadata:
  labels:
    app.kubernetes.io/name: kueue-operator
    app.kubernetes.io/managed-by: kustomize
  name: cluster
  namespace: openshift-kueue-operator
spec:
  config:
    workloadManagement:
      labelPolicy: QueueName
# ...
```

Example user-created Job object containing the kueue.x-k8s.io/queue-name label

```
apiVersion: batch/v1
kind: Job
metadata:
  generateName: sample-job-
  namespace: my-namespace
  labels:
    kueue.x-k8s.io/queue-name: user-queue
spec:
# ...
```

2.8. MONITORING PENDING WORKLOADS

Red Hat build of Kueue provides the **VisibilityOnDemand** feature to monitor pending workloads. A workload is an application that runs to completion. It can be composed by one or multiple pods that, loosely or tightly coupled, as a whole, complete a task. A workload is the unit of admission in Red Hat build of Kueue.

The **VisibilityOnDemand** feature provides the ability for batch administrators to monitor the pipeline of pending jobs in the cluster queue and the local queue and batch users just for local queue, and help users to estimate when their jobs will start.

You can regulate inbound requests and high request volumes, and provide user permissions for viewing the pending workloads.

2.8.1. API Priority and Fairness

Red Hat build of Kueue uses Kubernetes API Priority and Fairness (APF) To help manage pending workloads. APF is a flow control mechanism that allows you to define API-level policies to regulate inbound requests to the API server. It protects the API server from being overwhelmed by unexpectedly high request volume, while protecting critical traffic from the throttling effect on best-effort workloads.

Additional resources

- API Priority and Fairness

2.8.2. Providing user permissions

You can configure role-based access control (RBAC) objects for the users of your Red Hat build of Kueue deployment. These objects determine which types of users can create which types of Red Hat build of Kueue objects.

You need to provide permissions to the users that require access to the specific APIs.

- If the user needs access to the pending workloads from the **ClusterQueue** resource, a **ClusterRoleBinding** schema needs to be created referencing the ClusterRole **kueue-batch-admin-role**.

- If the user needs access to the pending workloads from the **LocalQueue** resource, a **RoleBinding** schema needs to be created referencing the ClusterRole **kueue-batch-user-role**.

Additional resources

- Configuring role-based permissions

2.8.3. Monitoring pending workloads on demand

To test the monitoring of pending workloads, you must correctly configure both the **ClusterQueue** and the **LocalQueue** resources. After that, you can create jobs on that **LocalQueue**. Kueue manages the workload object created from the job so, when a job is submitted and saturates the **ClusterQueue**, its corresponding workloads can be seen in the list of pending workloads.

Prerequisites

- You have cluster administrator permissions.
- The Red Hat build of Kueue Operator is installed on your cluster, and you have created a **Kueue** custom resource (CR).
- You have installed the OpenShift CLI (**oc**).
- The OpenShift CLI (**oc**) has communication with your cluster.

The following procedure tells you how to install and test workload monitoring.

Procedure

1. Create the assets by running the following command:

```
cat <<EOF| oc create -f -
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: ResourceFlavor
metadata:
  name: "default-flavor"
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: "cluster-queue"
spec:
  namespaceSelector: {} # match all.
  resourceGroups:
  - coveredResources: ["cpu", "memory"]
    flavors:
    - name: "default-flavor"
      resources:
      - name: "cpu"
        nominalQuota: 9
      - name: "memory"
        nominalQuota: 36Gi
---
apiVersion: kueue.x-k8s.io/v1beta1
kind: LocalQueue
metadata:
  namespace: "default"
  name: "user-queue"
spec:
  clusterQueue: "cluster-queue"
---
EOF
```

2. Create the following file with the job manifest:

```
cat >> job.yaml << EOF
apiVersion: batch/v1
kind: Job
metadata:
  generateName: sample-job-
  namespace: default
  labels:
    kueue.x-k8s.io/queue-name: user-queue
spec:
  parallelism: 3
  completions: 3
  suspend: true
  template:
    spec:
      containers:
      - name: <example-job>
        image: registry.k8s.io/e2e-test-images/agnhost:2.53
        command: [ "/bin/sh" ]
        args: [ "-c", "sleep 60" ]
        resources:
          requests:
            cpu: "1"
            memory: "200Mi"
      restartPolicy: Never
EOF
```

3. Label the default namespace to be managed by Kueue by running the following command:

```
$ oc label namespace default kueue.openshift.io/managed=true
```

4. Create the six jobs by running the following command:

```
for i in {1..6}; do oc create -f job.yaml; done
```

In this example, three of the jobs saturate the **ClusterQueue** resource and the other three jobs should be pending.

2.8.3.1. Viewing pending workloads in ClusterQueue

To view all pending workloads at the cluster level, administrators can use the **ClusterQueue** object visibility endpoint of Kueue's visibility API. This endpoint returns a list of all workloads currently waiting for admission by that **ClusterQueue** resource.

Procedure

1. To view pending workloads in **ClusterQueue** run the following command:

```
$ oc get --raw "/apis/visibility.kueue.x-k8s.io/v1beta1/clusterqueues/cluster-queue/pendingworkloads"
```

Example output


```
{
  "kind": "PendingWorkloadsSummary",
  "apiVersion": "visibility.kueue.x-k8s.io/v1beta1",
  "metadata": {
    "creationTimestamp": null
  },
  "items": [
    {
      "metadata": {
        "name": "job-sample-job-jrjfr-8d56e",
        "namespace": "default",
        "creationTimestamp": "2023-12-05T15:42:03Z",
        "ownerReferences": [
          {
            "apiVersion": "batch/v1",
            "kind": "Job",
            "name": "sample-job-jrjfr",
            "uid": "5863cf0e-b0e7-43bf-a445-f41fa1abedfa"
          }
        ]
      },
      "priority": 0,
      "localQueueName": "user-queue",
      "positionInClusterQueue": 0,
      "positionInLocalQueue": 0
    },
    {
      "metadata": {
        "name": "job-sample-job-jg9dw-5f1a3",
        "namespace": "default",
        "creationTimestamp": "2023-12-05T15:42:03Z",
        "ownerReferences": [
          {
            "apiVersion": "batch/v1",
            "kind": "Job",
            "name": "sample-job-jg9dw",
            "uid": "fd5d1796-f61d-402f-a4c8-cbda646e2676"
          }
        ]
      },
      "priority": 0,
      "localQueueName": "user-queue",
      "positionInClusterQueue": 1,
      "positionInLocalQueue": 1
    },
    {
      "metadata": {
        "name": "job-sample-job-t9b8m-4e770",
        "namespace": "default",
        "creationTimestamp": "2023-12-05T15:42:03Z",
        "ownerReferences": [
          {
            "apiVersion": "batch/v1",
            "kind": "Job",
```

```
            "name": "sample-job-t9b8m",
            "uid": "64c26c73-6334-4d13-a1a8-38d99196baa5"
          }
        ]
      },
      "priority": 0,
      "localQueueName": "user-queue",
      "positionInClusterQueue": 2,
      "positionInLocalQueue": 2
    }
  ]
}
```

You can pass the following optional query parameters:

limit <integer>

1000 is the default. Specifies the maximum number of pending workloads that should be fetched.

offset <integer>

0 is the default. Specifies the position of the first pending workload that should be fetched, starting from 0.

2. To view only one pending workload starting from position 0 in **ClusterQueue** run the following command:

```
$ oc get --raw "/apis/visibility.kueue.x-k8s.io/v1beta1/clusterqueues/cluster-queue/pendingworkloads?limit=1&offset=0"
```

2.8.3.2. Viewing pending workloads in LocalQueue

To view the pending workloads submitted by a specific tenant within their namespace, users can query the **LocalQueue** resource visibility endpoint of Kueue's visibility API. This provides an ordered list of their jobs waiting in that queue.

Procedure

1. To view pending workloads in **LocalQueue** run the following command:

```
$ oc get --raw /apis/visibility.kueue.x-k8s.io/v1beta1/namespaces/default/localqueues/user-queue/pendingworkloads
```

Example output

```
{
  "kind": "PendingWorkloadsSummary",
  "apiVersion": "visibility.kueue.x-k8s.io/v1beta1",
  "metadata": {
    "creationTimestamp": null
  },
  "items": [
    {
      "metadata": {
        "name": "job-sample-job-jrjfr-8d56e",
```

```
        "namespace": "default",
        "creationTimestamp": "2023-12-05T15:42:03Z",
        "ownerReferences": [
          {
            "apiVersion": "batch/v1",
            "kind": "Job",
            "name": "sample-job-jrjfr",
            "uid": "5863cf0e-b0e7-43bf-a445-f41fa1abedfa"
          }
        ]
      },
      "priority": 0,
      "localQueueName": "user-queue",
      "positionInClusterQueue": 0,
      "positionInLocalQueue": 0
    },
    {
      "metadata": {
        "name": "job-sample-job-jg9dw-5f1a3",
        "namespace": "default",
        "creationTimestamp": "2023-12-05T15:42:03Z",
        "ownerReferences": [
          {
            "apiVersion": "batch/v1",
            "kind": "Job",
            "name": "sample-job-jg9dw",
            "uid": "fd5d1796-f61d-402f-a4c8-cbda646e2676"
          }
        ]
      },
      "priority": 0,
      "localQueueName": "user-queue",
      "positionInClusterQueue": 1,
      "positionInLocalQueue": 1
    },
    {
      "metadata": {
        "name": "job-sample-job-t9b8m-4e770",
        "namespace": "default",
        "creationTimestamp": "2023-12-05T15:42:03Z",
        "ownerReferences": [
          {
            "apiVersion": "batch/v1",
            "kind": "Job",
            "name": "sample-job-t9b8m",
            "uid": "64c26c73-6334-4d13-a1a8-38d99196baa5"
          }
        ]
      },
      "priority": 0,
      "localQueueName": "user-queue",
      "positionInClusterQueue": 2,
      "positionInLocalQueue": 2
    }
  ]
}
```

You can pass the following optional query parameters:

limit <integer>

1000 is the default. Specifies the maximum number of pending workloads that should be fetched.

offset <integer>

0 is the default. Specifies the position of the first pending workload that should be fetched, starting from 0.

2. To view only one pending workload starting from position 0 in LocalQueue run the following command:

```
$ oc get --raw "/apis/visibility.kueue.x-k8s.io/v1beta1/namespaces/default/localqueues/user-queue/pendingworkloads?limit=1&offset=0"
```

2.8.4. Modifying monitoring settings

Modify the monitoring settings according to your organization's requirements to ensure users can access and view the pending workloads in a timely and reliable manner.

This procedure tells you how to modify the resource flow control for the Red Hat build of Kueue **VisibilityOnDemand** feature. Modifications directly impact the system's ability to handle concurrent requests for job visibility information.

Procedure

1. Edit the **PriorityLevelConfiguration** asset for **VisibilityOnDemand** on **Kueue** by running the following command:

```
$ oc edit prioritylevelconfiguration kueue-visibility
```

2. Modify the **nominalConcurrencyShares** field in the **PriorityLevelConfiguration** asset by setting the value for **kueue.openshift.io/allow-nominal-concurrency-shares-update** to **true**. The possible values you can specify for **nominalConcurrencyShares** are **0**, **2** (the default) until **5**. If you specify a value that is not acceptable (the value **1** or any value above **5**), the default value **2**, is enforced.

   See the following example:

```
apiVersion: flowcontrol.apiserver.k8s.io/v1
kind: PriorityLevelConfiguration
metadata:
  name: kueue-visibility
  annotations:
    kueue.openshift.io/allow-nominal-concurrency-shares-update: "false"
spec:
  limited:
    borrowingLimitPercent: 0
    lendablePercent: 90
    limitResponse:
      queuing:
        handSize: 4
        queueLengthLimit: 50
```

```
      queues: 16
    type: Queue
  nominalConcurrencyShares: 2
type: Limited
```

The default value for **kueue.openshift.io/allow-nominal-concurrency-shares-update** is **false**. If you change the value of **nominalConcurrencyShares** to any value other than **2**, then you must first change the value of **kueue.openshift.io/allow-nominal-concurrency-shares-update** to **true**. Otherwise, the value you assign for **nominalConcurrencyShares** will not take effect.

3. Verify the value is kept by running the following command:

```
$ oc get prioritylevelconfiguration kueue-visibility
```

2.9. USING COHORTS

You can use cohorts to group cluster queues and determine which cluster queues are able to share borrowable resources with each other. Borrowable resources are defined as the unused nominal quota of all the cluster queues in a cohort.

Using cohorts can help to optimize resource utilization by preventing under-utilization and enabling fair sharing configurations. Cohorts can also help to simplify resource management and allocation between teams, because you can group cluster queues for related workloads or for each team. You can also use cohorts to set resource quotas at a group level to define the limits for resources that a group of cluster queues can consume.

2.9.1. Configuring cohorts within a cluster queue spec

You can add a cluster queue to a cohort by specifying the name of the cohort in the **.spec.cohort** field of the **ClusterQueue** object, as shown in the following example:

```
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: cluster-queue
spec:
# ...
  cohort: example-cohort
# ...
```

All cluster queues that have a matching **spec.cohort** are part of the same cohort.

If the **spec.cohort** field is omitted, the cluster queue does not belong to any cohort and cannot access borrowable resources.

2.10. CONFIGURING FAIR SHARING

Fair sharing is a preemption strategy that is used to achieve an equal or weighted share of borrowable resources between the tenants of a cohort. Borrowable resources are the unused nominal quota of all the cluster queues in a cohort.

You can configure fair sharing by setting the **preemptionPolicy** value in the **Kueue** custom resource (CR) to **FairSharing**.

2.10.1. Cluster queue weights

After you have enabled fair sharing, you must set share values for each cluster queue before fair sharing can take place. Share values are represented as the **weight** value in a **ClusterQueue** object.

Share values are important because they allow administrators to prioritize specific job types or teams. Critical applications or high-priority teams can be configured with a weighted value so that they receive a proportionally larger share of the available resources. Configuring weights ensures that unused resources are distributed according to defined organizational or project priorities rather than on a first-come, first-served basis.

The **weight** value, or share value, defines a comparative advantage for the cluster queue when competing for borrowable resources. Generally, Red Hat build of Kueue admits jobs with a lower share value first. Jobs with a higher share value are more likely to be preempted before those with lower share values.

Example cluster queue with a fair sharing weight configured

```
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: cluster-queue
spec:
  namespaceSelector: {}
  resourceGroups:
  - coveredResources: ["cpu"]
    flavors:
    - name: default-flavor
      resources:
      - name: cpu
        nominalQuota: 9
  cohort: example-cohort
  fairSharing:
    weight: 2
```

2.10.1.1. Zero weight

A **weight** value of **0** represents an infinite share value. This means that the cluster queue is always at a disadvantage compared to others, and its workloads are always the first to be preempted when fair sharing is enabled.

2.10.2. Additional resources

- Creating a **Kueue** custom resource

2.11. GANG SCHEDULING

Gang scheduling ensures that a group or *gang* of related jobs only start when all required resources are available. Red Hat build of Kueue enables gang scheduling by suspending jobs until the OpenShift Container Platform cluster can guarantee the capacity to start and execute all of the related jobs in the gang together. This is also known as *all-or-nothing* scheduling.

Gang scheduling is important if you are working with expensive, limited resources, such as GPUs. Gang scheduling can prevent jobs from claiming but not using GPUs, which can improve GPU utilization and can reduce running costs. Gang scheduling can also help to prevent issues like resource segmentation and deadlocking.

2.11.1. Configuring gang scheduling

As a cluster administrator, you can configure gang scheduling by modifying the **gangScheduling** spec in the **Kueue** custom resource (CR).

Example Kueue CR with gang scheduling configured

```
apiVersion: kueue.openshift.io/v1
kind: Kueue
metadata:
  name: cluster
  labels:
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/name: kueue-operator
  namespace: openshift-kueue-operator
spec:
  config:
    gangScheduling:
      policy: ByWorkload (1)
      byWorkload:
        admission: Parallel (2)
# ...
```

(1) You can set the **policy** value to enable or disable gang scheduling. The possible values are **ByWorkload**, **None**, or empty (**""**).

ByWorkload
: When the **policy** value is set to **ByWorkload**, each job is processed and considered for admission as a single unit. If the job does not become ready within the specified time, the entire job is evicted and retried at a later time.

None
: When the **policy** value is set to **None**, gang scheduling is disabled.

Empty ("")
: When the **policy** value is empty or set to **""**, the Red Hat build of Kueue Operator determines settings for gang scheduling. Currently, gang scheduling is disabled by default.

(2) If the **policy** value is set to **ByWorkload**, you must configure job admission settings. The possible values for the **admission** spec are **Parallel**, **Sequential**, or empty (**""**).

Parallel
: When the **admission** value is set to **Parallel**, pods from any job can be admitted at any time. This can cause a deadlock, where jobs are in contention for cluster capacity. When a deadlock occurs, the successful scheduling of pods from another job can prevent the scheduling of pods from the current job.

Sequential
: When the **admission** value is set to **Sequential**, only pods from the currently processing job are admitted. After all of the pods from the current job have been admitted and are ready, Red Hat build of Kueue processes the next job. Sequential processing can slow down

admission when the cluster has sufficient capacity for multiple jobs, but provides a higher likelihood that all of the pods for a job are scheduled together successfully.

Empty ("")

When the **admission** value is empty or set to **""**, the Red Hat build of Kueue Operator determines job admission settings. Currently, the **admission** value is set to **Parallel** by default.

2.11.2. Additional resources

- Creating a Kueue custom resource

2.12. RUNNING JOBS WITH QUOTA LIMITS

You can run Kubernetes jobs with Red Hat build of Kueue enabled to manage resource allocation within defined quota limits. This can help to ensure predictable resource availability, cluster stability, and optimized performance.

2.12.1. Identifying available local queues

Before you can submit a job to a queue, you must find the name of the local queue.

Prerequisites

- A cluster administrator has installed and configured Red Hat build of Kueue on your OpenShift Container Platform cluster.
- A cluster administrator has assigned you the **kueue-batch-user-role** cluster role.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Run the following command to list available local queues in your namespace:

```
$ oc -n <namespace> get localqueues
```

Example output

```
NAME         CLUSTERQUEUE    PENDING WORKLOADS
user-queue   cluster-queue   3
```

2.12.2. Defining a job to run with Red Hat build of Kueue

When you are defining a job to run with Red Hat build of Kueue, ensure that it meets the following criteria:

- Specify the local queue to submit the job to, by using the **kueue.x-k8s.io/queue-name** label.
- Include the resource requests for each job pod.

Red Hat build of Kueue suspends the job, and then starts it when resources are available. Red Hat build of Kueue creates a corresponding workload, represented as a **Workload** object with a name that matches the job.

Prerequisites

- A cluster administrator has installed and configured Red Hat build of Kueue on your OpenShift Container Platform cluster.
- A cluster administrator has assigned you the **kueue-batch-user-role** cluster role.
- You have installed the OpenShift CLI (**oc**).
- You have identified the name of the local queue that you want to submit jobs to.

Procedure

1. Create a **Job** object.

Example job

```
apiVersion: batch/v1
kind: Job (1)
metadata:
  generateName: sample-job- (2)
  namespace: my-namespace
  labels:
    kueue.x-k8s.io/queue-name: user-queue (3)
spec:
  parallelism: 3
  completions: 3
  template:
    spec:
      containers:
      - name: dummy-job
        image: registry.k8s.io/e2e-test-images/agnhost:2.53
        args: ["entrypoint-tester", "hello", "world"]
        resources: (4)
          requests:
            cpu: 1
            memory: "200Mi"
      restartPolicy: Never
```

(1) Defines the resource type as a **Job** object, which represents a batch computation task.

(2) Provides a prefix for generating a unique name for the job.

(3) Identifies the queue to send the job to.

(4) Defines the resource requests for each pod.

2. Run the job by running the following command:

```
$ oc create -f <filename>.yaml
```

Verification

- Verify that pods are running for the job you have created, by running the following command and observing the output:

```
$ oc get job <job-name>
```

Example output

```
NAME              STATUS      COMPLETIONS   DURATION   AGE
sample-job-sk42x  Suspended   0/1                      2m12s
```

- Verify that a workload has been created in your namespace for the job, by running the following command and observing the output:

```
$ oc -n <namespace> get workloads
```

Example output

```
NAME                        QUEUE        RESERVED IN   ADMITTED   FINISHED   AGE
job-sample-job-sk42x-77c03  user-queue                              3m8s
```

2.13. GETTING SUPPORT

If you experience difficulty with a procedure described in this documentation, or with Red Hat build of Kueue in general, visit the Red Hat Customer Portal.

From the Customer Portal, you can:

- Search or browse through the Red Hat Knowledgebase of articles and solutions relating to Red Hat products.
- Submit a support case to Red Hat Support.
- Access other product documentation.

2.13.1. About the Red Hat Knowledgebase

The Red Hat Knowledgebase provides rich content aimed at helping you make the most of Red Hat's products and technologies. The Red Hat Knowledgebase consists of articles, product documentation, and videos outlining best practices on installing, configuring, and using Red Hat products. In addition, you can search for solutions to known issues, each providing concise root cause descriptions and remedial steps.

2.13.2. Collecting data for Red Hat Support

You can use the **oc adm must-gather** CLI command to collect the information about your Red Hat build of Kueue instance that is most likely needed for debugging issues, including:

- Red Hat build of Kueue custom resources, such as workloads, cluster queues, local queues, resource flavors, admission checks, and their corresponding cluster resource definitions (CRDs)
- Services

- Endpoints
- Webhook configurations
- Logs from the **openshift-kueue-operator** namespace and **kueue-controller-manager** pods

Collected data is written into a new directory named **must-gather/** in the current working directory by default.

Prerequisites

- The Red Hat build of Kueue Operator is installed on your cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.
2. Collect **must-gather** data by running the following command:

```
$ oc adm must-gather \
  --image=registry.redhat.io/kueue/kueue-must-gather-rhel9:<version>
```

   Where **<version>** is your current version of Red Hat build of Kueue.

3. Create a compressed file from the **must-gather** directory that was just created in your working directory. Make sure you provide the date and cluster ID for the unique **must-gather** data. For more information about how to find the cluster ID, see How to find the cluster-id or name on OpenShift cluster.
4. Attach the compressed file to your support case on the the **Customer Support** page of the Red Hat Customer Portal.

2.13.3. Additional resources

- Support overview

CHAPTER 3. LEADER WORKER SET OPERATOR

3.1. LEADER WORKER SET OPERATOR OVERVIEW

Use the Leader Worker Set Operator to manage multi-node AI/ML inference deployments efficiently. The Leader Worker Set Operator treats groups of pods as one unit to simplify scaling, recovery, and updates for large workloads.

Using large language models (LLMs) for AI/ML inference often requires significant compute resources, and workloads typically must be sharded across multiple nodes. This can make deployments complex, creating challenges around scaling, recovery from failures, and efficient pod placement.

The Leader Worker Set Operator simplifies these multi-node deployments by treating a group of pods as a single, coordinated unit. It manages the lifecycle of each pod in the group, scales the entire group together, and performs updates and failure recovery at the group level to ensure consistency.

3.1.1. About the Leader Worker Set Operator

Use the Leader Worker Set Operator to deploy groups of pods as a single, manageable unit. This helps you to deploy large AI/ML inference workloads, such as sharded large language models (LLMs).

The Leader Worker Set Operator is based on the LeaderWorkerSet open source project. **LeaderWorkerSet** is a custom Kubernetes API that can be used to deploy a group of pods as a unit. This is useful for artificial intelligence (AI) and machine learning (ML) inference workloads, where large language models (LLMs) are sharded across multiple nodes.

With the **LeaderWorkerSet** API, pods are grouped into units consisting of one leader and multiple workers, all managed together as a single entity. Each pod in a group has a unique pod identity. Pods within a group are created in parallel and share identical lifecycle stages. Rollouts, rolling updates, and pod failure restarts are performed as a group.

In the **LeaderWorkerSet** configuration, you define the size of the groups and the number of group replicas. If necessary, you can define separate templates for leader and worker pods, allowing for role-specific customization. You can also configure topology-aware placement, so that pods in the same group are co-located in the same topology.

IMPORTANT

Before you install the Leader Worker Set Operator, you must install the cert-manager Operator for Red Hat OpenShift because it is required to configure services and manage metrics collection.

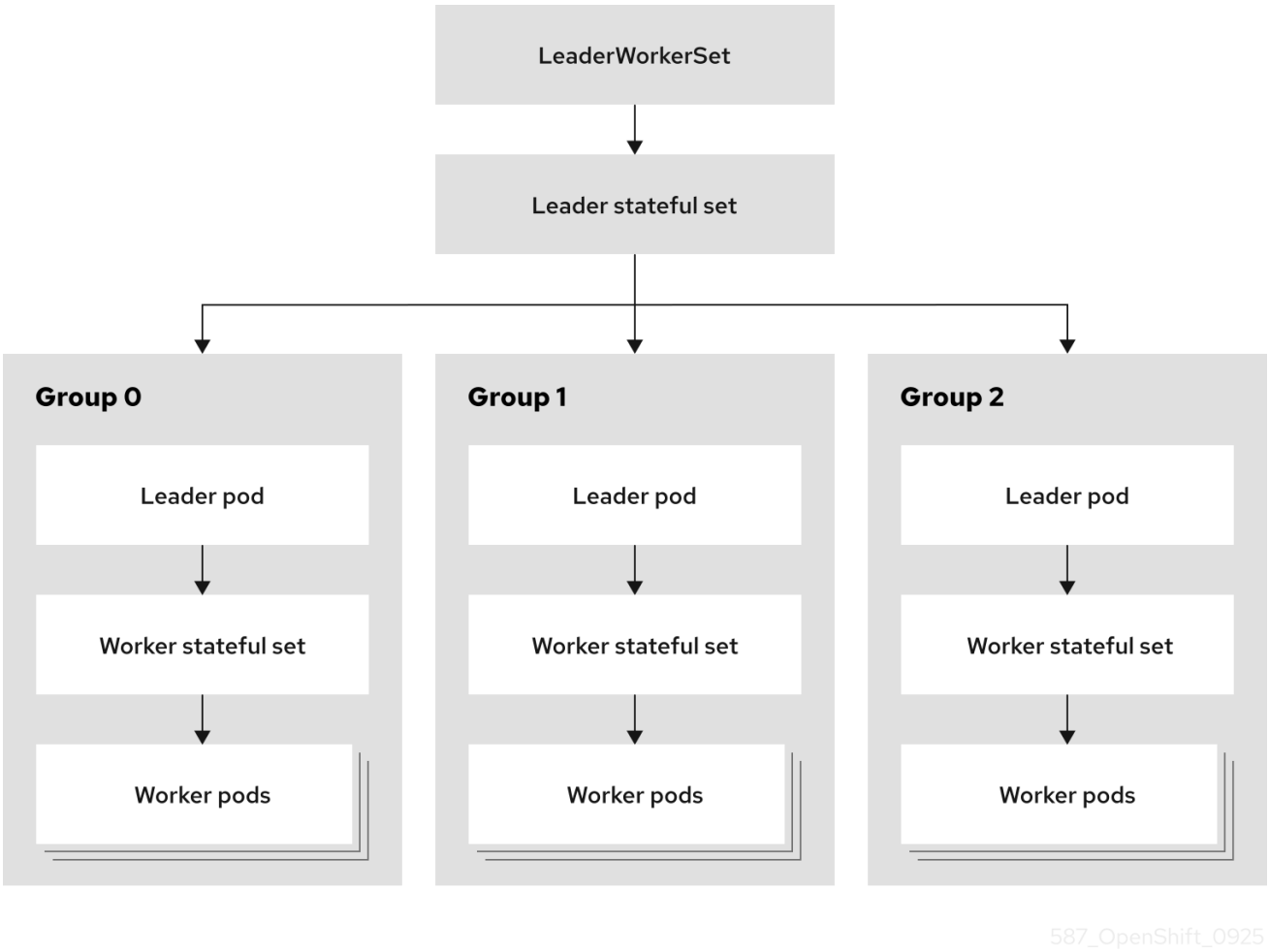
Monitoring for the Leader Worker Set Operator is provided by default with OpenShift Container Platform through Prometheus.

3.1.1.1. LeaderWorkerSet architecture

Review the LeaderWorkerSet architecture to learn how the **LeaderWorkerSet** API organizes groups of pods into a single unit, with one pod as the leader and the rest as the workers, to coordinate distributed workloads.

The following diagram describes the LeaderWorkerSet architecture:

Figure 3.1. Leader worker set architecture



The **LeaderWorkerSet** API uses a leader stateful set to manage the deployment and lifecycle of the groups of pods. For each replica defined, a leader-worker group is created.

Each leader-worker group contains a leader pod and a worker stateful set. The worker stateful set is owned by the leader pod and manages the set of worker pods associated with that leader pod. The specified size defines the total number of pods in each leader-worker group, with the leader pod included in that number.

3.1.2. Additional resources

- LeaderWorkerSet documentation (Kubernetes)

3.2. LEADER WORKER SET OPERATOR RELEASE NOTES

Review the Leader Worker Set Operator release notes to track its development and learn what is new and changed with each release.

You can use the Leader Worker Set Operator to manage distributed inference workloads and process large-scale inference requests efficiently.

These release notes track the development of the Leader Worker Set Operator.

For more information, see About the Leader Worker Set Operator.

3.2.1. Release notes for Leader Worker Set Operator 1.0.0

Review the release notes for Leader Worker Set Operator 1.0.0 to learn what is new and updated with this release.

Issued: 18 September 2025

The following advisories are available for the Leader Worker Set Operator 1.0.0:

- RHBA-2025:13974
- RHBA-2025:13574

3.2.1.1. New features and enhancements

- This is the initial release of the Leader Worker Set Operator.

3.3. MANAGING DISTRIBUTED WORKLOADS WITH THE LEADER WORKER SET OPERATOR

You can use the Leader Worker Set Operator to manage distributed inference workloads and process large-scale inference requests efficiently.

3.3.1. Installing the Leader Worker Set Operator

You can install the Leader Worker Set Operator through the OpenShift Container Platform web console to begin managing distributed AI workloads.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have installed the cert-manager Operator for Red Hat OpenShift.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Verify that the cert-manager Operator for Red Hat OpenShift is installed.
3. Install the Leader Worker Set Operator.
    a. Navigate to **Ecosystem → Software Catalog**.
    b. Enter **Leader Worker Set Operator** into the filter box.
    c. Select the **Leader Worker Set Operator** and click **Install**.
    d. On the **Install Operator** page:
        i. The **Update channel** is set to **stable-v1.0**, which installs the latest stable release of Leader Worker Set Operator 1.0.
        ii. Under **Installation mode**, select **A specific namespace on the cluster**

iii. Under **Installed Namespace**, select **Operator recommended Namespace: openshift-lws-operator**.

iv. Under **Update approval**, select one of the following update strategies:

- The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.
- The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.

v. Click **Install**.

4. Create the custom resource (CR) for the Leader Worker Set Operator:

a. Navigate to **Installed Operators → Leader Worker Set Operator**.

b. Under **Provided APIs**, click **Create instance** in the **LeaderWorkerSetOperator** pane.

c. Click **Create**.

3.3.2. Deploying a leader worker set

You can use the Leader Worker Set Operator to deploy a leader worker set to assist with managing distributed workloads across nodes.

Prerequisites

- You have installed the Leader Worker Set Operator.

Procedure

1. Create a new project by running the following command:

```
$ oc new-project my-namespace
```

2. Create a file named **leader-worker-set.yaml**

```
apiVersion: leaderworkerset.x-k8s.io/v1
kind: LeaderWorkerSet
metadata:
  generation: 1
  name: my-lws
  namespace: my-namespace
spec:
  leaderWorkerTemplate:
    leaderTemplate:
      metadata: {}
      spec:
        containers:
        - image: nginxinc/nginx-unprivileged:1.27
          name: leader
          resources: {}
    restartPolicy: RecreateGroupOnPodRestart
    size: 3
    workerTemplate:
```

```
    metadata: {}
    spec:
      containers:
      - image: nginxinc/nginx-unprivileged:1.27
        name: worker
        ports:
        - containerPort: 8080
          protocol: TCP
        resources: {}
  networkConfig:
    subdomainPolicy: Shared
  replicas: 2
  rolloutStrategy:
    rollingUpdateConfiguration:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  startupPolicy: LeaderCreated
```

where:

metadata.name

Specifies the name of the leader worker set resource.

metadata.namespace

Specifies the namespace for the leader worker set to run in.

spec.leaderWorkerTemplate.leaderTemplate

Specifies the pod template for the leader pods.

spec.leaderWorkerTemplate.restartPolicy

Specifies the restart policy for when pod failures occur. Allowed values are **RecreateGroupOnPodRestart** to restart the whole group or **None** to not restart the group.

spec.leaderWorkerTemplate.size

Specifies the number of pods to create for each group, including the leader pod. For example, a value of **3** creates 1 leader pod and 2 worker pods. The default value is **1**.

spec.leaderWorkerTemplate.workerTemplate

Specifies the pod template for the worker pods.

spec.networkConfig.subdomainPolicy

Specifies the policy to use when creating the headless service. Allowed values are **UniquePerReplica** or **Shared**. The default value is **Shared**.

spec.replicas

Specifies the number of replicas, or leader-worker groups. The default value is **1**.

spec.rolloutStrategy.rollingUpdateConfiguration.maxSurge

Specifies the maximum number of replicas that can be scheduled above the **replicas** value during rolling updates. The value can be specified as an integer or a percentage.

For more information on all available fields to configure, see LeaderWorkerSet API upstream documentation.

3. Apply the leader worker set configuration by running the following command:

```
$ oc apply -f leader-worker-set.yaml
```

Verification

1. Verify that pods were created by running the following command:

```
$ oc get pods -n my-namespace
```

Example output

```
NAME       READY   STATUS    RESTARTS   AGE
my-lws-0    1/1     Running   0          4s
my-lws-0-1  1/1     Running   0          3s
my-lws-0-2  1/1     Running   0          3s
my-lws-1    1/1     Running   0          7s
my-lws-1-1  1/1     Running   0          6s
my-lws-1-2  1/1     Running   0          6s
```

- **my-lws-0** is the leader pod for the first group.
- **my-lws-1** is the leader pod for the second group.

2. Review the stateful sets by running the following command:

```
$ oc get statefulsets
```

Example output

```
NAME      READY   AGE
my-lws    4/4     111s
my-lws-0  2/2     57s
my-lws-1  2/2     60s
```

- **my-lws** is the leader stateful set for all leader-worker groups.
- **my-lws-0** is the worker stateful set for the first group.
- **my-lws-1** is the worker stateful set for the second group.

3.3.3. Additional resources

- LeaderWorkerSet API (Kubernetes)

3.4. UNINSTALLING THE LEADER WORKER SET OPERATOR

If you no longer need the Leader Worker Set Operator in your cluster, you can uninstall the Operator and remove its related resources.

3.4.1. Uninstalling the Leader Worker Set Operator

You can use the web console to uninstall the Leader Worker Set Operator if you no longer need the Operator in your cluster.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have installed the Leader Worker Set Operator.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Navigate to **Operators → Installed Operators**.
3. Select **openshift-lws-operator** from the **Project** dropdown list.
4. Delete the **LeaderWorkerSetOperator** instance.
   a. Click **Leader Worker Set Operator** and select the **LeaderWorkerSetOperator** tab.
   b. Click the Options menu  next to the **cluster** entry and select **Delete LeaderWorkerSetOperator**.
   c. In the confirmation dialog, click **Delete**.
5. Uninstall the Leader Worker Set Operator.
   a. Navigate to **Operators → Installed Operators**.
   b. Click the Options menu  next to the **Leader Worker Set Operator** entry and click **Uninstall Operator**.
   c. In the confirmation dialog, click **Uninstall**.

3.4.2. Uninstalling Leader Worker Set Operator resources

Optionally, remove custom resources (CRs) and the associated namespace after the Leader Worker Set Operator is uninstalled. This cleans up all remaining Leader Worker Set artifacts.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have uninstalled the Leader Worker Set Operator.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Remove CRDs that were created when the Leader Worker Set Operator was installed:

a. Navigate to **Administration → CustomResourceDefinitions**.

b. Enter **LeaderWorkerSetOperator** in the **Name** field to filter the CRDs.

c. Click the Options menu  next to the **LeaderWorkerSetOperator** CRD and select **Delete CustomResourceDefinition**.

d. In the confirmation dialog, click **Delete**.

3. Delete the **openshift-lws-operator** namespace.

a. Navigate to **Administration → Namespaces**.

b. Enter **openshift-lws-operator** into the filter box.

c. Click the Options menu  next to the **openshift-lws-operator** entry and select **Delete Namespace**.

d. In the confirmation dialog, enter **openshift-lws-operator** and click **Delete**.

CHAPTER 4. JOBSET OPERATOR

4.1. JOBSET OPERATOR OVERVIEW

Use the JobSet Operator on OpenShift Container Platform to manage and run large-scale, coordinated workloads like high-performance computing (HPC) and AI training. Features like multi-template job support and stable networking can help you recover quickly and use resources efficiently.

IMPORTANT

JobSet Operator is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

4.1.1. About the JobSet Operator

Use the JobSet Operator on OpenShift Container Platform to manage large, distributed, and coordinated computing workloads, such as high-performance computing (HPC) or artificial intelligence (AI) training, and gain automatic stability, coordination, and failure recovery.

The JobSet Operator is based on the JobSet open source project.

JobSet Operator is designed to manage a group of jobs as a single, coordinated unit. This is especially useful for fields like HPC and training massive AI models where you need a team of machines to run for hours or days.

You can use the JobSet Operator to solve problems that are too big or too complex for a standard OpenShift Container Platform job. The JobSet Operator provides coordination, stability, and recovery.

The JobSet Operator automatically sets up stable headless service to get an IP address so workers can find and communicate with each other, even after a failure and restart. It also provides automatic failure recovery. If one small part of a large training job fails, the Operator can be configured to restart the entire group of workers from a saved checkpoint. This saves time and computing costs.

The JobSet Operator offers startup control, allowing you to define a specific startup sequence to ensure dependencies are met. For example, making sure the leader is running before any workers attempt to connect.

JobSet Operator makes managing large, distributed, and coordinated computing tasks on OpenShift Container Platform easier, turning many individual components into one resilient and manageable system.

4.1.2. Additional resources

- JobSet documentation (Kubernetes)

4.2. INSTALLING THE JOBSET OPERATOR

Install the JobSet Operator on OpenShift Container Platform to enable management of large-scale, coordinated computing workloads, giving your applications a unified API and failure recovery.

IMPORTANT

JobSet Operator is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

4.2.1. Installing the JobSet Operator

Install the JobSet Operator on OpenShift Container Platform using the web console to begin managing large-scale, coordinated computing workloads.

Prerequisites

- You have access to the cluster with **cluster-admin** privileges.
- You have access to the OpenShift Container Platform web console.
- You have installed the cert-manager Operator for Red Hat OpenShift.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Verify that the cert-manager Operator for Red Hat OpenShift is installed.
3. Install the JobSet Operator.
    a. Navigate to **Ecosystem → Software Catalog**.
    b. Search for and select the **openshift-operators** project.
    c. Enter **JobSet Operator** into the filter box.
    d. Select the **JobSet Operator** and click **Install**.
    e. On the **Install Operator** page:
        i. The **Update channel** is set to **tech-preview-v0.1**, which installs the latest stable release of JobSet Operator 0.1.
        ii. Under **Installation mode**, select **A specific namespace on the cluster**.
        iii. Under **Installed Namespace**, select **Operator recommended Namespace: openshift-jobset-operator**.
        iv. Under **Update approval**, select one of the following update strategies:

- The **Automatic** strategy allows Operator Lifecycle Manager (OLM) to automatically update the Operator when a new version is available.
- The **Manual** strategy requires a user with appropriate credentials to approve the Operator update.

v. Click **Install**.

4. Create the custom resource (CR) for the JobSet Operator:

a. Navigate to **Installed Operators → JobSet Operator**.

b. Under **Provided APIs**, click **Create instance** in the **JobSetOperator** pane.

c. Set the name to **cluster**.

d. Set the **managementState** to **Managed**.

e. Click **Create**.

Verification

- Check that the JobSet Operator and operand pods are running by entering the following command:

```
$ oc get pod -n openshift-jobset-operator
```

Example output

```
NAME                                      READY   STATUS    RESTARTS   AGE
jobset-controller-manager-5595547fb-b4g2x   1/1     Running   0          48s
jobset-operator-596cb848c6-q2dmp            1/1     Running   0          2m33s
```

4.3. JOBSET OPERATOR RELEASE NOTES

Track the development, features, and fixes for the JobSet Operator, which manages coordinated, large-scale computing workloads on OpenShift Container Platform.

IMPORTANT

JobSet Operator is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

For more information, see About the JobSet Operator.

4.3.1. Release notes for JobSet Operator 0.1.0

Review the new features and advisories for the initial Technology Preview release of JobSet Operator 0.1.0.

Issued: 4 November 2025

The following advisories are available for the JobSet Operator 0.1.0:

- RHBA-2025:19431

4.3.1.1. New features and enhancements

- This is the initial Technology Preview release of the JobSet Operator.