Red Hat OpenShift AI Cloud Service 1

Working with distributed workloads

Use distributed workloads for faster and more efficient data processing and model training

Red Hat OpenShift AI Cloud Service 1 Working with distributed workloads

Use distributed workloads for faster and more efficient data processing and model training

Legal Notice

Abstract

Distributed workloads enable data scientists to use multiple cluster nodes in parallel for faster and more efficient data processing and model training. The CodeFlare framework simplifies task orchestration and monitoring, and offers seamless integration for automated resource scaling and optimal node utilization with advanced GPU support.

Table of Contents

PREFACE

To train complex machine-learning models or process data more quickly, data scientists can use the distributed workloads feature to run their jobs on multiple OpenShift worker nodes in parallel. This approach significantly reduces the task completion time, and enables the use of larger datasets and more complex models.

CHAPTER 1. OVERVIEW OF DISTRIBUTED WORKLOADS

You can use the distributed workloads feature to queue, scale, and manage the resources required to run data science workloads across multiple nodes in an OpenShift cluster simultaneously. Typically, data science workloads include several types of artificial intelligence (AI) workloads, including machine learning (ML) and Python workloads.

Distributed workloads provide the following benefits:

- You can iterate faster and experiment more frequently because of the reduced processing time.

- You can use larger datasets, which can lead to more accurate models.

- You can use complex models that could not be trained on a single node.

- You can submit distributed workloads at any time, and the system then schedules the distributed workload when the required resources are available.

The distributed workloads infrastructure includes the following components:

CodeFlare Operator

Secures deployed Ray clusters and grants access to their URLs

CodeFlare SDK

Defines and controls the remote distributed compute jobs and infrastructure for any Python-based environment

KubeRay

Manages remote Ray clusters on OpenShift for running distributed compute workloads

Kueue

Manages quotas and how distributed workloads consume them, and manages the queueing of distributed workloads with respect to quotas

You can run distributed workloads from data science pipelines, from Jupyter notebooks, or from Microsoft Visual Studio Code files.

CHAPTER 2. CONFIGURING DISTRIBUTED WORKLOADS

To configure the distributed workloads feature for your data scientists to use in OpenShift AI, you must create the required Kueue resources, enable several components in the Red Hat OpenShift AI Add-on, and optionally configure the CodeFlare Operator.

2.1. CONFIGURING THE DISTRIBUTED WORKLOADS COMPONENTS

To configure the distributed workloads feature for your data scientists to use in OpenShift AI, you must enable several components.

Prerequisites

- You have logged in to OpenShift with the **cluster-admin** role.

- You have access to the data science cluster.

- You have installed Red Hat OpenShift AI.

- You have sufficient resources. In addition to the minimum OpenShift AI resources described in Installing and deploying OpenShift AI , you need 1.6 vCPU and 2 GiB memory to deploy the distributed workloads infrastructure.

- You have access to a Ray cluster image. For information about how to create a Ray cluster, see the Ray Clusters documentation .

> **NOTE**
>
> Mutual Transport Layer Security (mTLS) is enabled by default in the CodeFlare component in OpenShift AI. In the current OpenShift AI version, **submissionMode=K8sJobMode** is not supported in the Ray job specification, so the KubeRay Operator cannot create a submitter Kubernetes Job to submit the Ray job. Instead, users must configure the Ray job specification to set **submissionMode=HTTPMode** only, so that the KubeRay Operator sends a request to the RayCluster to create a Ray job.

- You have access to the data sets and models that the distributed workload uses.

- You have access to the Python dependencies for the distributed workload.

- You have removed any previously installed instances of the CodeFlare Operator, as described in the Knowledgebase solution How to migrate from a separately installed CodeFlare Operator in your data science cluster.

- If you want to use graphics processing units (GPUs), you have enabled GPU support in OpenShift AI. See Enabling GPU support in OpenShift AI .

- If you want to use self-signed certificates, you have added them to a central Certificate Authority (CA) bundle as described in Working with certificates. No additional configuration is necessary to use those certificates with distributed workloads. The centrally configured self-signed certificates are automatically available in the workload pods at the following mount points:

  - Cluster-wide CA bundle:

> /etc/pki/tls/certs/odh-trusted-ca-bundle.crt
> /etc/ssl/certs/odh-trusted-ca-bundle.crt

- Custom CA bundle:

  > /etc/pki/tls/certs/odh-ca-bundle.crt
  > /etc/ssl/certs/odh-ca-bundle.crt

Procedure

1. In the OpenShift console, click **Operators → Installed Operators**.

2. Search for the **Red Hat OpenShift AI** Operator, and then click the Operator name to open the Operator details page.

3. Click the **Data Science Cluster** tab.

4. Click the default instance name (for example, **default-dsc**) to open the instance details page.

5. Click the **YAML** tab to show the instance specifications.

6. Enable the required distributed workloads components. In the **spec:components** section, set the **managementState** field correctly for the required components. The list of required components depends on whether the distributed workload is run from a pipeline or notebook or both, as shown in the following table.

Table 2.1. Components required for distributed workloads

Component	Pipelines only	Notebooks only	Pipelines and notebooks
codeflare	**Managed**	**Managed**	**Managed**
dashboard	**Managed**	**Managed**	**Managed**
datasciencepipelines	**Managed**	**Removed**	**Managed**
kueue	**Managed**	**Managed**	**Managed**
ray	**Managed**	**Managed**	**Managed**
workbenches	**Removed**	**Managed**	**Managed**

7. Click **Save**. After a short time, the components with a **Managed** state are ready.

Verification

Check the status of the **codeflare-operator-manager**, **kuberay-operator**, and **kueue-controller-manager** pods, as follows:

1. In the OpenShift console, from the **Project** list, select **redhat-ods-applications**.

2. Click **Workloads → Deployments**.

3. Search for the **codeflare-operator-manager**, **kuberay-operator**, and **kueue-controller-manager** deployments. In each case, check the status as follows:

   a. Click the deployment name to open the deployment details page.

   b. Click the **Pods** tab.

   c. Check the pod status.
   When the status of the **codeflare-operator-manager-*<pod-id>***, **kuberay-operator-*<pod-id>***, and **kueue-controller-manager-*<pod-id>*** pods is **Running**, the pods are ready to use.

   d. To see more information about each pod, click the pod name to open the pod details page, and then click the **Logs** tab.

2.2. CONFIGURING QUOTA MANAGEMENT FOR DISTRIBUTED WORKLOADS

Configure quotas for distributed workloads on a cluster, so that you can share resources between several data science projects.

Prerequisites

- You have cluster administrator privileges for your OpenShift cluster.

- You have downloaded and installed the OpenShift command-line interface (CLI). See Installing the OpenShift CLI (Red Hat OpenShift Dedicated) or Installing the OpenShift CLI (Red Hat OpenShift Service on AWS).

- You have enabled the required distributed workloads components as described in Configuring the distributed workloads components.

- You have sufficient resources. In addition to the base OpenShift AI resources, you need 1.6 vCPU and 2 GiB memory to deploy the distributed workloads infrastructure.

- The resources are physically available in the cluster.

> **NOTE**
>
> OpenShift AI currently supports only a single cluster queue per cluster (that is, homogenous clusters), and only empty resource flavors. For more information about Kueue resources, see the Kueue documentation.

Procedure

1. In a terminal window, if you are not already logged in to your OpenShift cluster as a cluster administrator, log in to the OpenShift CLI as shown in the following example:

   ```
   $ oc login <openshift_cluster_url> -u <admin_username> -p <password>
   ```

2. Create an empty Kueue resource flavor, as follows:

   a. Create a file called **default_flavor.yaml** and populate it with the following content:

   Empty Kueue resource flavor

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

b. Apply the configuration to create the **default-flavor** object:

```
$ oc apply -f default_flavor.yaml
```

3. Create a cluster queue to manage the empty Kueue resource flavor, as follows:

a. Create a file called **cluster_queue.yaml** and populate it with the following content:

Example cluster queue

```
apiVersion: kueue.x-k8s.io/v1beta1
kind: ClusterQueue
metadata:
  name: "cluster-queue"
spec:
  namespaceSelector: {}  # match all.
  resourceGroups:
  - coveredResources: ["cpu", "memory", "nvidia.com/gpu"]
    flavors:
    - name: "default-flavor"
      resources:
      - name: "cpu"
        nominalQuota: 9
      - name: "memory"
        nominalQuota: 36Gi
      - name: "nvidia.com/gpu"
        nominalQuota: 5
```

b. Replace the example quota values (9 CPUs, 36 GiB memory, and 5 NVIDIA GPUs) with the appropriate values for your cluster queue. The cluster queue will start a distributed workload only if the total required resources are within these quota limits.
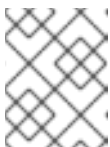
> **NOTE**
>
> In this release of OpenShift AI, the only accelerators supported for distributed workloads are NVIDIA GPUs.

c. Apply the configuration to create the **cluster-queue** object:

```
$ oc apply -f cluster_queue.yaml
```

4. Create a local queue that points to your cluster queue, as follows:

a. Create a file called **local_queue.yaml** and populate it with the following content:

Example local queue

```
apiVersion: kueue.x-k8s.io/v1beta1
```

```
kind: LocalQueue
metadata:
  namespace: test
  name: local-queue-test
  annotations:
    kueue.x-k8s.io/default-queue: 'true'
spec:
  clusterQueue: cluster-queue
```

The **kueue.x-k8s.io/default-queue: 'true'** annotation defines this queue as the default queue. Distributed workloads are submitted to this queue if no **local_queue** value is specified in the **ClusterConfiguration** section of the data science pipeline or Jupyter notebook or Microsoft Visual Studio Code file.

b. Update the **namespace** value to specify the same namespace as in the **ClusterConfiguration** section that creates the Ray cluster.

c. Optional: Update the **name** value accordingly.

d. Apply the configuration to create the local-queue object:

```
$ oc apply -f local_queue.yaml
```

The cluster queue allocates the resources to run distributed workloads in the local queue.

Verification

Check the status of the local queue in a project, as follows:

```
$ oc get -n <project-name> localqueues
```

Additional resources

- [Kueue documentation](#)

2.3. CONFIGURING THE CODEFLARE OPERATOR

If you want to change the default configuration of the CodeFlare Operator for distributed workloads in OpenShift AI, you can edit the associated config map.

Prerequisites

- You have logged in to OpenShift with the **cluster-admin** role.

- You have enabled the required distributed workloads components as described in [Configuring the distributed workloads components](#).

Procedure

1. In the OpenShift console, click **Workloads** → **ConfigMaps**.

2. From the **Project** list, select **redhat-ods-applications**.

3. Search for the **codeflare-operator-config** config map, and click the config map name to open the **ConfigMap details** page.

4. Click the **YAML** tab to show the config map specifications.

5. In the **data:config.yaml:kuberay** section, you can edit the following entries:

   ingressDomain

   This configuration option is null (**ingressDomain: ""**) by default. Do not change this option unless the Ingress Controller is not running on OpenShift. OpenShift AI uses this value to generate the dashboard and client routes for every Ray Cluster, as shown in the following examples:

   Example dashboard and client routes

   > ray-dashboard-*<clustername>-<namespace>.<your.ingress.domain>*
   > ray-client-*<clustername>-<namespace>.<your.ingress.domain>*

   mTLSEnabled

   This configuration option is enabled (**mTLSEnabled: true**) by default. When this option is enabled, the Ray Cluster pods create certificates that are used for mutual Transport Layer Security (mTLS), a form of mutual authentication, between Ray Cluster nodes. When this option is enabled, Ray clients cannot connect to the Ray head node unless they download the generated certificates from the **ca-secret-_<cluster_name>_** secret, generate the necessary certificates for mTLS communication, and then set the required Ray environment variables. Users must then re-initialize the Ray clients to apply the changes. The CodeFlare SDK provides the following functions to simplify the authentication process for Ray clients:

   Example Ray client authentication code

   > from codeflare_sdk import generate_cert
   >
   > generate_cert.generate_tls_cert(cluster.config.name, cluster.config.namespace)
   > generate_cert.export_env(cluster.config.name, cluster.config.namespace)
   >
   > ray.init(cluster.cluster_uri())

   rayDashboardOauthEnabled

   This configuration option is enabled (**rayDashboardOAuthEnabled: true**) by default. When this option is enabled, OpenShift AI places an OpenShift OAuth proxy in front of the Ray Cluster head node. Users must then authenticate by using their OpenShift cluster login credentials when accessing the Ray Dashboard through the browser. If users want to access the Ray Dashboard in another way (for example, by using the Ray **JobSubmissionClient** class), they must set an authorization header as part of their request, as shown in the following example:

   Example authorization header

   > {Authorization: "Bearer *<your-openshift-token>*"}

6. To save your changes, click **Save**.

7. To apply your changes, delete the pod:

   a. Click **Workloads → Pods**.

b. Find the **codeflare-operator-manager-<pod-id>** pod.

c. Click the options menu (⋮) for that pod, and then click **Delete Pod**. The pod restarts with your changes applied.

Verification

Check the status of the **codeflare-operator-manager** pod, as follows:

1. In the OpenShift console, click **Workloads → Deployments**.

2. Search for the **codeflare-operator-manager** deployment, and then click the deployment name to open the deployment details page.

3. Click the **Pods** tab. When the status of the **codeflare-operator-manager-<pod-id>** pod is **Running**, the pod is ready to use. To see more information about the pod, click the pod name to open the pod details page, and then click the **Logs** tab.

CHAPTER 3. RUNNING DISTRIBUTED WORKLOADS

In OpenShift AI, you can run a distributed workload from a notebook or from a pipeline. You can also run distributed workloads in a disconnected environment if you have access to all of the required software.

3.1. RUNNING DISTRIBUTED DATA SCIENCE WORKLOADS FROM NOTEBOOKS

To run a distributed data science workload from a notebook, you must first provide the link to your Ray cluster image.

Prerequisites

- You have access to a data science cluster that is configured to run distributed workloads as described in Configuring distributed workloads.

- You have created the required Kueue resources as described in Configuring quota management for distributed workloads.

- Optional: You have defined a *default* local queue for the Ray cluster by creating a **LocalQueue** resource and adding the following annotation to its configuration details, as described in Configuring quota management for distributed workloads:

    > "kueue.x-k8s.io/default-queue": "true"
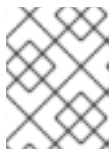
    NOTE

    If you do not create a default local queue, you must specify a local queue in each notebook.

- You have created a data science project that contains a workbench that is running one of the default notebook images, for example, the **Standard Data Science** notebook. See the table in Notebook images for data scientists for a complete list of default notebook images.

- You have launched your notebook server and logged in to your notebook editor. The examples in this procedure refer to the JupyterLab integrated development environment (IDE).

Procedure

1. Download the demo notebooks provided by the CodeFlare SDK. The demo notebooks provide guidelines for how to use the CodeFlare stack in your own notebooks.
   To access the demo notebooks, clone the **codeflare-sdk** repository as follows:

   a. In the JupyterLab interface, click **Git > Clone a Repository**.

   b. In the "Clone a repo" dialog, enter **https://github.com/project-codeflare/codeflare-sdk.git** and then click **Clone**. The **codeflare-sdk** repository is listed in the left navigation pane.

2. Run a distributed workload job as shown in the following example:

   a. In the JupyterLab interface, in the left navigation pane, double-click **codeflare-sdk**.

   b. Double-click **demo-notebooks**, and then double-click **guided-demos**.

c. Update each example demo notebook as follows:

- If not already specified, update the **import** section to import the **generate_cert** component:

    Updated import section

    ```
    from codeflare_sdk import generate_cert
    ```

- Replace the default namespace value with the name of your data science project.

- Replace the link to the example community image with a link to your Ray cluster image.

- Ensure that the following Ray cluster authentication code is included after the Ray cluster creation section.

    Ray cluster authentication code

    ```
    generate_cert.generate_tls_cert(cluster.config.name, cluster.config.namespace)
    generate_cert.export_env(cluster.config.name, cluster.config.namespace)
    ```

    NOTE

    Mutual Transport Layer Security (mTLS) is enabled by default in the CodeFlare component in OpenShift AI. You must include the Ray cluster authentication code to enable the Ray client that runs within a notebook to connect to a secure Ray cluster that has mTLS enabled.

- If you have not configured a default local queue by including the **kueue.x-k8s.io/default-queue: 'true'** annotation as described in Configuring quota management for distributed workloads, update the **ClusterConfiguration** section to specify the local queue for the Ray cluster, as shown in the following example:

    Example local queue assignment

    ```
    local_queue="your_local_queue_name"
    ```

d. Run the notebooks.

Verification

The notebooks run to completion without errors. In the notebooks, the output from the **cluster.status()** function or **cluster.details()** function indicates that the Ray cluster is **Active**.

3.2. RUNNING DISTRIBUTED DATA SCIENCE WORKLOADS FROM DATA SCIENCE PIPELINES

To run a distributed data science workload from a data science pipeline, you must first update the pipeline to include a link to your Ray cluster image.

Prerequisites

- You have logged in to OpenShift with the **cluster-admin** role.

- You have access to a data science cluster that is configured to run distributed workloads as described in Configuring distributed workloads.

- You have created the required Kueue resources as described in Configuring quota management for distributed workloads.

- Optional: You have defined a *default* local queue for the Ray cluster by creating a **LocalQueue** resource and adding the following annotation to the configuration details for that **LocalQueue** resource, as described in Configuring quota management for distributed workloads:

  > "kueue.x-k8s.io/default-queue": "true"

  > **NOTE**
  >
  > If you do not create a default local queue, you must specify a local queue in each pipeline.

- You have access to S3-compatible object storage.

- You have logged in to Red Hat OpenShift AI.

- You have created a data science project.

Procedure

1. Create a data connection to connect the object storage to your data science project, as described in Adding a data connection to your data science project .

2. Configure a pipeline server to use the data connection, as described in Configuring a pipeline server.

3. Create the data science pipeline as follows:

   a. Install the **kfp** Python package, which is required for all pipelines:

      > `$ pip install kfp`

   b. Install any other dependencies that are required for your pipeline.

   c. Build your data science pipeline in Python code.
      For example, create a file named **compile_example.py** with the following content:

      ```
      from kfp import dsl


      @dsl.component(
          base_image="registry.redhat.io/ubi8/python-39:latest",
          packages_to_install=['codeflare-sdk']
      )


      def ray_fn(openshift_server: str, openshift_token: str): 1
          import ray
          from codeflare_sdk import Cluster, ClusterConfiguration, TokenAuthentication,
      ```

```
generate_cert

  auth = TokenAuthentication( 2
    token=openshift_token, server=openshift_server
  )
  auth_return = auth.login()


  cluster = Cluster( 3
    ClusterConfiguration(
      name="raytest",
      num_workers=1,
      head_cpus="500m",
      min_memory=1,
      max_memory=1,
      num_gpus=0,
      image="quay.io/project-codeflare/ray:latest-py39-cu118", 4
      local_queue="local_queue_name", 5
    )
  )


  print(cluster.status())
  cluster.up() 6
  cluster.wait_ready() 7
  print(cluster.status())
  print(cluster.details())


  ray_dashboard_uri = cluster.cluster_dashboard_uri()
  ray_cluster_uri = cluster.cluster_uri()
  print(ray_dashboard_uri, ray_cluster_uri)

  # Enable Ray client to connect to secure Ray cluster that has mTLS enabled
  generate_cert.generate_tls_cert(cluster.config.name, cluster.config.namespace) 8
  generate_cert.export_env(cluster.config.name, cluster.config.namespace)


  ray.init(address=ray_cluster_uri)
  print("Ray cluster is up and running: ", ray.is_initialized())


  @ray.remote
  def train_fn(): 9
    # complex training function
    return 100


  result = ray.get(train_fn.remote())
  assert 100 == result
  ray.shutdown()
  cluster.down() 10
  auth.logout()
  return result
```

```
@dsl.pipeline( 11
    name="Ray Simple Example",
    description="Ray Simple Example",
)


def ray_integration(openshift_server: str, openshift_token: str): 12
    ray_fn(openshift_server=openshift_server, openshift_token=openshift_token)


if __name__ == '__main__':
    from kfp.compiler import Compiler
    Compiler().compile(ray_integration, 'compiled-example.yaml')
```

1   Imports from the CodeFlare SDK the packages that define the cluster functions

2   Authenticates with the cluster by using values that you specify when creating the pipeline run; if your cluster uses self-signed certificates, include **ca-cert-path=<path>** in the **TokenAuthentication** parameter list, where **<path>** is the path to the cluster-wide Certificate Authority (CA) bundle that contains the self-signed certificates

3   Specifies the Ray cluster configuration: replace these example values with the values for your Ray cluster

4   Specifies the location of the Ray cluster image: if using a disconnected environment, replace the default value with the location for your environment

5   Specifies the local queue to which the Ray cluster will be submitted: you can omit this line if you configured a default local queue

6   Creates a Ray cluster using the specified image and configuration

7   Waits for the Ray cluster to be ready before proceeding

8   Enables the Ray client to connect to a secure Ray cluster that has mutual Transport Layer Security (mTLS) enabled; mTLS is enabled by default in the CodeFlare component in OpenShift AI

9   Replace the example details in this section with the details for your workload

10   Removes the Ray cluster when your workload is finished

11   Replace the example name and description with the values for your workload

12   Compiles the Python code and saves the output in a YAML file

d.   Compile the Python file (in this example, the **compile_example.py** file):

```
$ python compile_example.py
```

This command creates a YAML file (in this example, **compiled-example.yaml**), which you can import in the next step.

4. Import your data science pipeline, as described in Importing a data science pipeline .

5. Schedule the pipeline run, as described in Scheduling a pipeline run.

6. When the pipeline run is complete, confirm that it is included in the list of triggered pipeline runs, as described in Viewing the details of a pipeline run.

Verification

The YAML file is created and the pipeline run completes without errors.

You can view the run details, as described in Viewing the details of a pipeline run.

Additional resources

- Working with data science pipelines

- Ray Clusters documentation

CHAPTER 4. MONITORING DISTRIBUTED WORKLOADS

In OpenShift AI, you can view project metrics for distributed workloads, and view the status of all distributed workloads in the selected project. You can use these metrics to monitor the resources used by distributed workloads, assess whether project resources are allocated correctly, track the progress of distributed workloads, and identify corrective action when necessary.

4.1. VIEWING PROJECT METRICS FOR DISTRIBUTED WORKLOADS

In OpenShift AI, you can view the following project metrics for distributed workloads:

- **CPU** – The number of CPU cores that are currently being used by all distributed workloads in the selected project.

- **Memory** – The amount of memory in gibibytes (GiB) that is currently being used by all distributed workloads in the selected project.

You can use these metrics to monitor the resources used by the distributed workloads, and assess whether project resources are allocated correctly.

Prerequisites

- You have installed Red Hat OpenShift AI.

- On the OpenShift cluster where OpenShift AI is installed, user workload monitoring is enabled.

- You have logged in to OpenShift AI.

- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **rhoai-users** or **rhoai-admins**) in OpenShift.

- Your data science project contains distributed workloads.

Procedure

1. In the OpenShift AI left navigation pane, click **Distributed Workloads Metrics**.

2. From the **Project** list, select the project that contains the distributed workloads that you want to monitor.

3. Click the **Project metrics** tab.

4. Optional: From the **Refresh interval** list, select a value to specify how frequently the graphs on the metrics page are refreshed to show the latest data.
   You can select one of these values: **15 seconds**, **30 seconds**, **1 minute**, **5 minutes**, **15 minutes**, **30 minutes**, **1 hour**, **2 hours**, or **1 day**.

5. In the **Requested resources** section, review the **CPU** and **Memory** graphs to identify the resources requested by distributed workloads as follows:

   - Requested by the selected project

   - Requested by all projects, including the selected project and projects that you cannot access

   - Total shared quota for all projects, as provided by the cluster queue

For each resource type (**CPU** and **Memory**), subtract the **Requested by all projects** value from the **Total shared quota** value to calculate how much of that resource quota has not been requested and is available for all projects.

6. Scroll down to the **Top resource-consuming distributed workloads** section to review the following graphs:

   - Top 5 distributed workloads that are consuming the most CPU resources

   - Top 5 distributed workloads that are consuming the most memory

   You can also identify how much CPU or memory is used in each case.

7. Scroll down to view the **Distributed workload resource metrics** table, which lists all of the distributed workloads in the selected project, and indicates the current resource usage and the status of each distributed workload.
   In each table entry, progress bars indicate how much of the requested CPU and memory is currently being used by this distributed workload. To see numeric values for the actual usage and requested usage for CPU (measured in cores) and memory (measured in GiB), hover the cursor over each progress bar. Compare the actual usage with the requested usage to assess the distributed workload configuration. If necessary, reconfigure the distributed workload to reduce or increase the requested resources.

Verification

On the **Project metrics** tab, the graphs and table provide resource-usage data for the distributed workloads in the selected project.

4.2. VIEWING THE STATUS OF DISTRIBUTED WORKLOADS

In OpenShift AI, you can view the status of all distributed workloads in the selected project. You can track the progress of the distributed workloads, and identify corrective action when necessary.

Prerequisites

- You have installed Red Hat OpenShift AI.

- On the OpenShift cluster where OpenShift AI is installed, user workload monitoring is enabled.

- You have logged in to OpenShift AI.

- If you are using specialized OpenShift AI groups, you are part of the user group or admin group (for example, **rhoai-users** or **rhoai-admins**) in OpenShift.

- Your data science project contains distributed workloads.

Procedure

1. In the OpenShift AI left navigation pane, click **Distributed Workloads Metrics**.

2. From the **Project** list, select the project that contains the distributed workloads that you want to monitor.

3. Click the **Distributed workload status** tab.

4. Optional: From the **Refresh interval** list, select a value to specify how frequently the graphs on the metrics page are refreshed to show the latest data.
   You can select one of these values: **15 seconds**, **30 seconds**, **1 minute**, **5 minutes**, **15 minutes**, **30 minutes**, **1 hour**, **2 hours**, or **1 day**.

5. In the **Status overview** section, review a summary of the status of all distributed workloads in the selected project.
   The status can be **Pending**, **Inadmissible**, **Admitted**, **Running**, **Evicted**, **Succeeded**, or **Failed**.

6. Scroll down to view the **Distributed workloads** table, which lists all of the distributed workloads in the selected project. The table provides the priority, status, creation date, and latest message for each distributed workload.
   The latest message provides more information about the current status of the distributed workload. Review the latest message to identify any corrective action needed. For example, a distributed workload might be **Inadmissible** because the requested resources exceed the available resources. In such cases, you can either reconfigure the distributed workload to reduce the requested resources, or reconfigure the cluster queue for the project to increase the resource quota.

Verification

On the **Distributed workload status** tab, the graph provides a summarized view of the status of all distributed workloads in the selected project, and the table provides more details about the status of each distributed workload.