

Controlling User Access and Managing Resources

This chapter teaches you how to manage data science projects and controller access with users and groups within OpenShift AI. You will also learn about different administrative configuration options from within OpenShift AI.

Goals:

- Manage access to the OpenShift AI Dashboard
- Control access to data science projects with users and groups
- Configure automatic cleanup of idle workbenches
- Manage default workbench and model server sizes

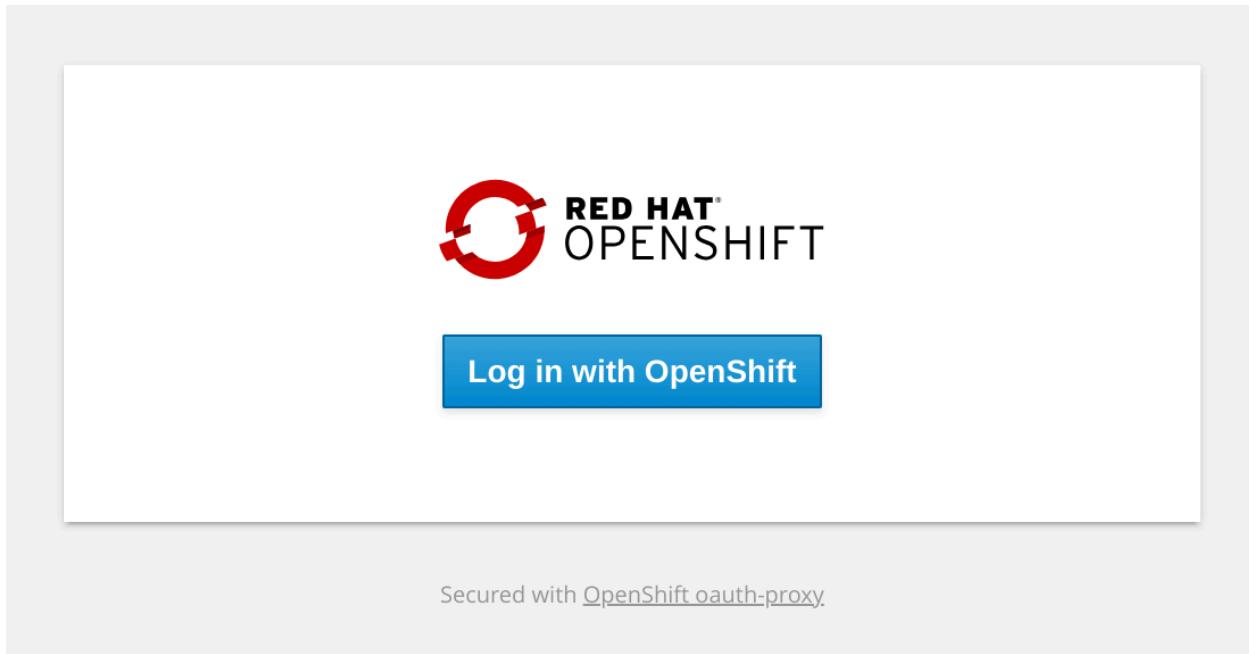
Users, Projects, and Permissions

As discussed in the previous section, Red Hat OpenShift AI allows users to install several different components to enable ML workloads. One of the components is the OpenShift AI Dashboard, which acts as the primary user interface to help guide users to working with, deploying, and managing ML solutions.

This section discusses the intricacies of managing users with OpenShift AI and how to modify the default permissions for a project.

Logging Into The OpenShift AI Dashboard

OpenShift AI utilizes OpenShift OAuth as the authentication method for all OpenShift AI related resources. When attempting to log into the OpenShift AI Dashboard or other UI components, users will be presented with a page prompting them to Log in with OpenShift.



By default, any user with the ability to login to OpenShift will be able to access the OpenShift AI Dashboard.

For assistance with configuring user access to OpenShift with various identity providers, please refer to the official [Preparing for users](#) OpenShift documentation.

Admin Access

By default, users with the `cluster-admin` role are automatically granted admin permissions in the OpenShift AI Dashboard. Additionally, a group called `rhods-admins` is automatically created, and any user added to that group will be granted admin permissions.

Admin access in the OpenShift AI Dashboard allows users to control various settings of OpenShift AI through the Dashboard UI, including:

- Configuring custom notebook images
- Configuring custom serving runtimes
- Configuring the idle notebook culler (automatic removal of inactive notebooks)
- Other notebook configuration settings

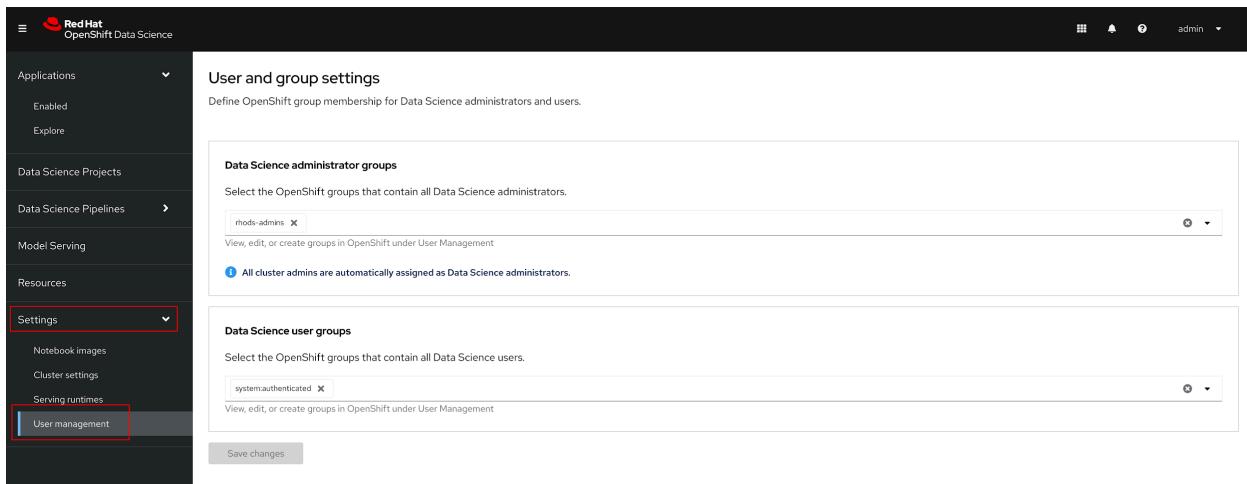
To add users to the `rhods-admins` group, run the following command:

Unset

```
oc adm groups add-users rhods-admins my-user1 my-user2
```

Updating Default User Permissions

The default permissions for the OpenShift AI Dashboard can be updated through the Settings User management section.



The admin group can be updated to another group other than the default `rhods-admins` group, such as one that is configured to sync with an external system. Additionally, admin permissions to the Dashboard are always granted to users assigned the `cluster-admin` ClusterRole.

It is highly recommended that dedicated admin users are configured for OpenShift AI and that organizations do not rely on the `cluster-admin` role for exclusive permissions to admin configurations of OpenShift AI. Dedicated Admin users should be added to the existing `rhods-admins` group, or another group which already contains the correct users should be configured in lieu of the `rhods-admins` group.

The normal Data Science user group can also be updated to change what users are able to sign into the OpenShift AI Dashboard. By default, the Dashboard is configured to allow all authenticated users with the `system:authenticated` role. A custom group can be configured to allow only specific users belonging to that group to access the Dashboard.

Updating the access in the User and group settings in the Dashboard will only impact a user's abilities to access the Dashboard, and will not impact any permissions granted by regular Kubernetes based RBAC.

For example, if the normal user group is updated to only allow specific users to access the Dashboard, and a user that is not part of that group has admin rights, they may still have the ability to create Data Science related objects such as Notebooks, Data Science Pipelines, or Model Servers in a namespace they have permission in using the associated k8s objects without the UI.

Managing Dashboard Permissions with GitOps

The User and group settings in the OpenShift AI Dashboard can also be managed using the OdhDashboardConfigs object called odh-dashboard-config located in the redhat-ods-applications namespace along with many of the other settings found in the Dashboard's Settings section:

Unset

```
oc get odhdashboardconfigs odh-dashboard-config -n  
redhat-ods-applications -o yaml
```

A truncated version of the OdhDashboardConfig object with the groupConfig settings can be found below:

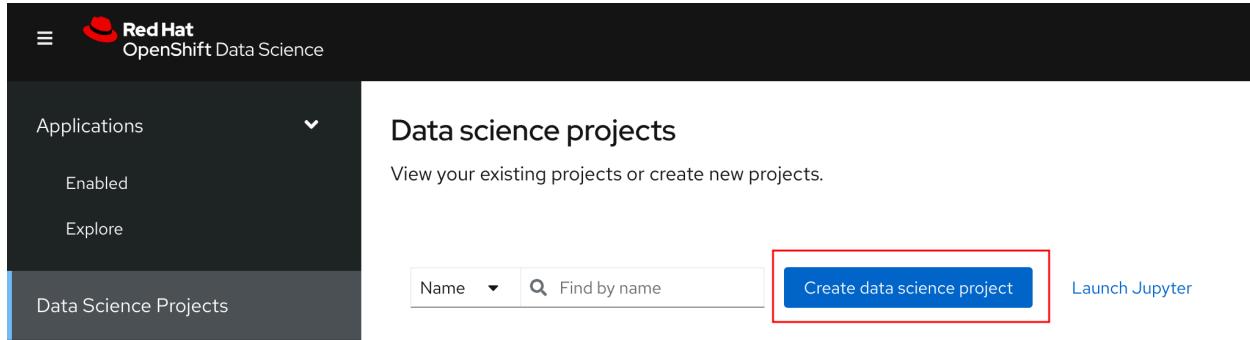
Unset

```
apiVersion: opendatahub.io/v1alpha  
kind: OdhDashboardConfig  
metadata:  
  name: odh-dashboard-config  
  namespace: redhat-ods-applications  
spec:  
  groupsConfig:  
    adminGroups: rhods-admins  
    allowedGroups: 'system:authenticated'
```

This object can be exported from the cluster and stored in a git repository to enable managing these settings with GitOps.

Default Data Science Project Permissions

The OpenShift AI Dashboard permits users of the Dashboard to create new Data Science Projects when self-provisioning is enabled on the cluster:



The screenshot shows the OpenShift Data Science interface. On the left, there's a sidebar with 'Applications' expanded, showing 'Enabled' and 'Explore'. Below that is a 'Data Science Projects' section. The main area is titled 'Data science projects' and contains the text 'View your existing projects or create new projects.' At the bottom, there's a search bar with 'Name' and 'Find by name' fields, a red-bordered 'Create data science project' button, and a 'Launch Jupyter' link.

The terms Namespaces, OpenShift Projects, and Data Science Projects are often used synonymy. While there are technical differences between these objects, a project is essentially a namespace with some additional features provided by OpenShift. OpenShift will always create a corresponding project for every namespace, and vice versa.

Similarly, a Data Science Project is just an OpenShift Project that has a specific label, which enables it to be used with the Dashboard.

For information on how to disable self-provisioner on OpenShift refer to the official [Disabling project self-provisioning](#) OpenShift documentation.

When a user creates a new Data Science Project through the Dashboard, the Dashboard will create a corresponding OpenShift Project and Kubernetes Namespace. The user who created the Data Science Project will automatically be granted the OpenShift admin role for the Namespace.

OpenShift allows for both "cluster admins" who have the ability to manage all cluster and namespace scoped objects on a cluster, and "namespace admins" that have permission to manage all namespace scoped objects in a specific namespace.

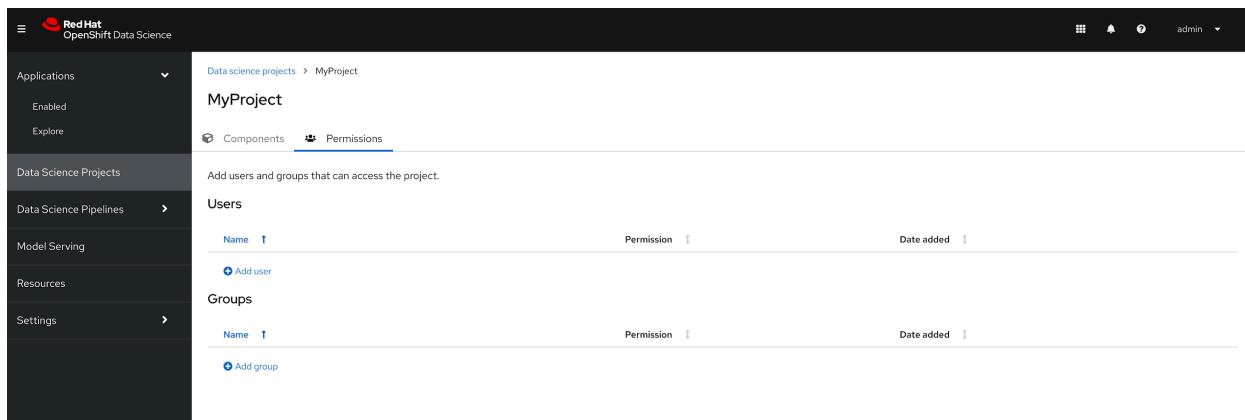
All users with the `cluster-admin` role have the ability to see all other users namespaces.

Managing Permissions on a Data Science Project

A Data Science Project is nothing more than a normal OpenShift Project with a few special labels. These labels are required for the project to appear in the Dashboard. As such, permissions to the project function the same as any other project/namespace in OpenShift.

Users and Groups permissions to a project are managed using Kubernetes Role Based Access and Control (RBAC). The Dashboard provides a user interface to easily assign additional users or groups permissions to a project.

An admin user on the project can add additional users or groups to a project by navigating to a specific project under the Data Science Projects menu, and selecting the Permissions tab.



The screenshot shows the Red Hat OpenShift Data Science dashboard. The left sidebar has a dark theme with white text. It includes sections for Applications (Enabled, Explore), Data Science Projects, Data Science Pipelines (with a right arrow), Model Serving, Resources, and Settings (with a right arrow). The 'Data Science Projects' section is currently selected. The main content area shows a breadcrumb path: Data science projects > MyProject. Below this, there are two tabs: 'Components' and 'Permissions', with 'Permissions' being the active tab. A note says 'Add users and groups that can access the project.' Below this are two tables: one for 'Users' and one for 'Groups'. Both tables have columns for 'Name', 'Permission', and 'Date added'. At the bottom of each table is a blue 'Add [User/Group]' button. The top right corner of the dashboard shows the user 'admin'.

From the permissions tab, project admin users are able to add additional users or groups and grant them the edit or admin role on the project.

The Dashboard Permissions user interface only displays users and groups that were granted permissions directly through the Dashboard. Any users or groups that have been granted permission to the Project by any traditional OpenShift role management such as creating RoleBindings on the project or who are granted higher level cluster permissions will not be displayed in the user interface.

Manually Creating Data Science Projects

When self-provisioner is disabled on a cluster, a cluster administrator will be required to manually create Data Science Projects for users.

As mentioned previously, a Data Science Project is a normal OpenShift project/namespace with a few special labels that allow to be managed by the RHOAI Dashboard, allowing it to be managed using any existing processes or tools your organization already utilizes to manage namespaces.

To manually create a Data Science Project from the cli, you can run the following commands to create a namespace and apply the necessary labels:

```
unset  
oc create namespace myproject  
oc label namespace myproject opendatahub.io/dashboard='true'  
modelmesh-enabled='true'
```

Alternatively, the following YAML object can be used to create the Data Science Project:

```
unset  
kind: Namespace  
apiVersion: v1  
metadata:  
  name: myproject  
  labels:  
    modelmesh-enabled: 'true'  
    opendatahub.io/dashboard: 'true'
```

Once the Data Science Project has been created, access to the project will need to be configured for the necessary user or group using either the cli, or a namespace RoleBinding. Refer to the OpenShift documentation for [Adding roles to users](#) for additional instructions.

Importing Custom Notebook Images

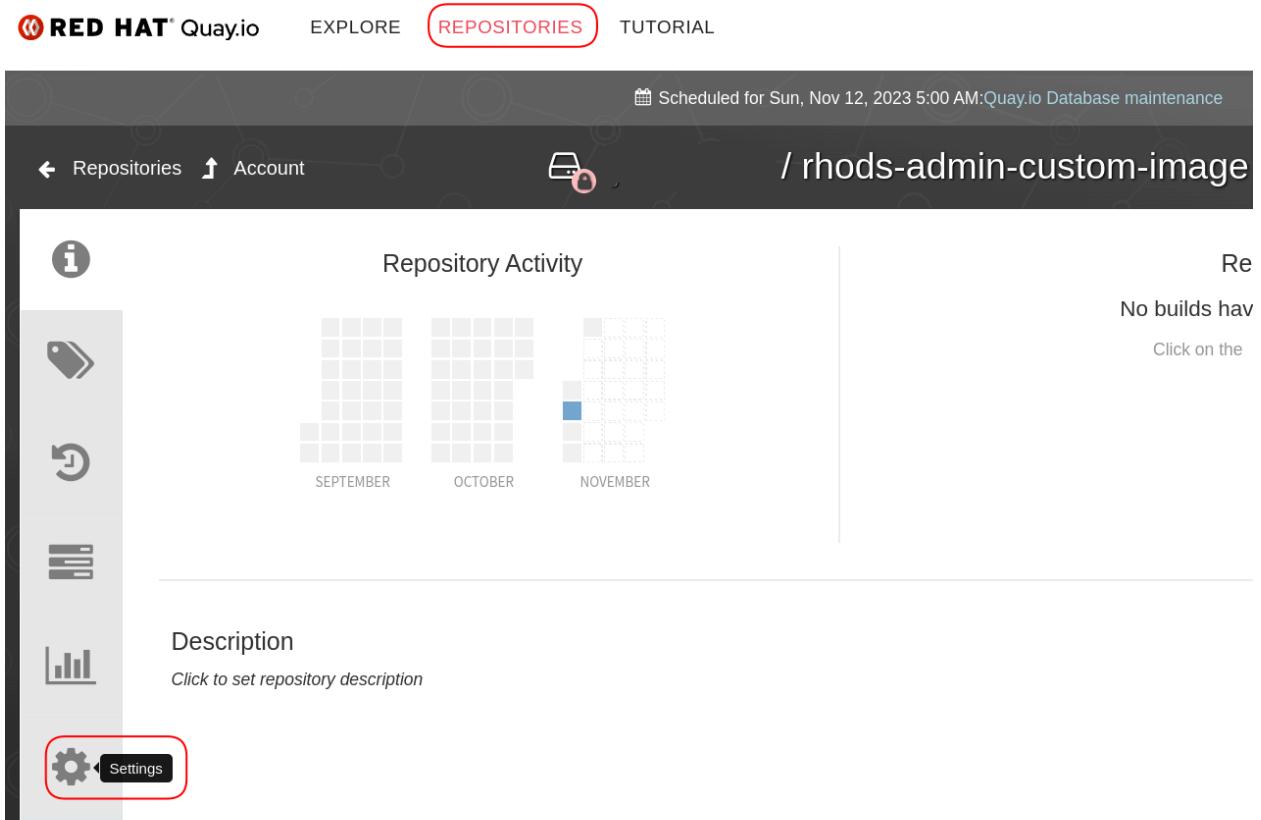
from:

<https://redhatquickcourses.github.io/rhods-admin/rhods-admin/1.33/chapter3/index.html>

In this section we will go over importing a custom notebook through the OpenShift AI dashboard and test it to make sure our dependencies are included.

Import the Notebook Image

1. Before we import the image into OpenShif AI we need to set the quay repository we just created to public. In a browser login to quay.io and go to the rhods-admin-custom-image repository. Select the Settings gear icon.



- 2.
3. Scroll down until you see the Repository Visibility section. Click the Make Public button and click Ok to make the repository public.

The screenshot shows the 'Repository Settings' page for a GitHub repository. On the left is a sidebar with icons for repository info, tags, releases, issues, pull requests, and settings. The main area has a header 'Repository Settings'. Under 'User and Robot Permissions', there's a 'USER PERMISSIONS' table with one row for 'J' (John Doe) with 'Admin' selected. A dropdown menu shows 'Read' and a green 'Add Permission' button. Below this is an 'Events and Notifications' section with a message: 'No notifications have been setup for this repository.' and a 'Create Notification' button. At the bottom is a 'Repository Visibility' section with a lock icon, stating 'This Repository is currently private. Only users on the permissions list may view and interact with it.' and a red-bordered 'Make Public' button.

4.

5.

To use private container images, you can create a `dockerconfigjson` secret in the `redhat-ods-applications` project and assign this secret to the default service account in this project.

The secret can contain the registry credentials in dockerconfig JSON format, which, if you use Quay, you can download from the Account Settings page, by clicking Generate Encrypted Password and then downloading the credentials JSON file from the Docker Configuration option.

After downloading the credentials file, use this file to create a secret:

```
Unset
$ oc create secret generic quay-pull-secret \
--from-file=.dockerconfigjson=path-to-credentials-file.json \
--type=kubernetes.io/dockerconfigjson \
-n redhat-ods-applications
```

Finally, link the secret to the default service account as a pull secret:

```
Unset
$ oc secrets link default quay-pull-secret --for=pull
```

```
\-n redhat-ods-applications
```

For more information, refer to [Importing images and image streams from private registries](#).

6. Login to the OpenShift AI dashboard as the admin user. Expand Settings, click Notebook Images, and then click Import Image.

The screenshot shows the Red Hat OpenShift Data Science dashboard. The left sidebar has a dark theme with white text. It includes links for Applications, Data Science Projects, Data Science Pipelines, Model Serving, Resources, and Settings. Under Settings, there is a dropdown menu with options: Notebook images (which is highlighted with a red box), Cluster settings, Serving runtimes, and User management. The main content area is titled "Notebook image settings" with the subtitle "Import, delete, and modify notebook images." It displays a message "No custom notebook images found." with a large plus sign icon. Below this, it says "To get started import a custom notebook image." and features a blue "Import image" button.

- 7.
8. Enter the path to the your repository and give the image a name. Optionally add a description and package information. Click the Import button.

Import notebook images

Repository *
quay.io/<YOUR_USERNAME>/rhods-admin-custom-image:latest

Repo where notebook images are stored.

Name *
rhods-admin-custom-image

Description
A custom image with the seaborn python package

Software Packages

Add the advertised packages shown with this notebook image. Modifying the packages here does not effect the contents of the notebook image.

Package	Version
seaborn	0.13.0

[+ Add Package](#)

[Import](#) [Cancel](#)

9.

10. You should now see the image in the list and enabled.

Notebook image settings

Import, delete, and modify notebook images.

Name	Description	Enable	User	Uploaded	⋮
rhods-admin-custom-image	A custom image with the seaborn python package	<input checked="" type="checkbox"/>	admin	Just now	⋮

11.

Test the Image in a Workbench

1. Now we'll test out the image we just imported. Go to Data Science Projects and then click the Create data science project button.

The screenshot shows the Red Hat OpenShift Data Science interface. The top navigation bar has the Red Hat logo and the text "Red Hat OpenShift Data Science". Below the navigation bar, there is a sidebar with three main items: "Applications", "Data Science Projects", and "Data Science Pipelines". The "Data Science Projects" item is highlighted with a red box. To the right of the sidebar, the main content area is titled "Data science projects" with the sub-instruction "View your existing projects or create new projects.". At the bottom of this section is a search bar with "Name" and "Find by name" fields, and a blue "Create data science project" button, which is also highlighted with a red box.

- 2.
3. Give the project a name and resource name. Click the Create button.
- 4.
5. Click the Create button.

[Data science projects](#) > rhods-custom-image-project

rhods-custom-image-project

This screenshot shows the details of the "rhods-custom-image-project". At the top, there are two tabs: "Components" (which is active) and "Permissions". Below the tabs, there is a "Jump to section" dropdown and a "Workbenches" section. The "Workbenches" section includes a "Create workbench" button, which is highlighted with a red box. To the right of this section is a large circular icon with a plus sign and the text "No workbenches". Below the icon, it says "To get started, create a workbench.".

Components	Permissions
Jump to section	Workbenches
Workbenches	Create workbench
Cluster storage	No workbenches
Data connections	To get started, create a workbench.

- 6.
7. Give the workbench a name and select the rhods-admin-custom-image from the Image selection drop down list. Click the Create Workbench button.

Jump to section

Name and description	Name * rhods-custom-image-workbench
Notebook image	Description
Deployment size	Notebook image
Environment variables	Image selection *
Cluster storage	rhods-admin-custom-image
Data connections	View package information

Deployment size

Container size
Small

8.

[Create workbench](#) [Cancel](#)

9. Once the Status changes to Running click the Open link.

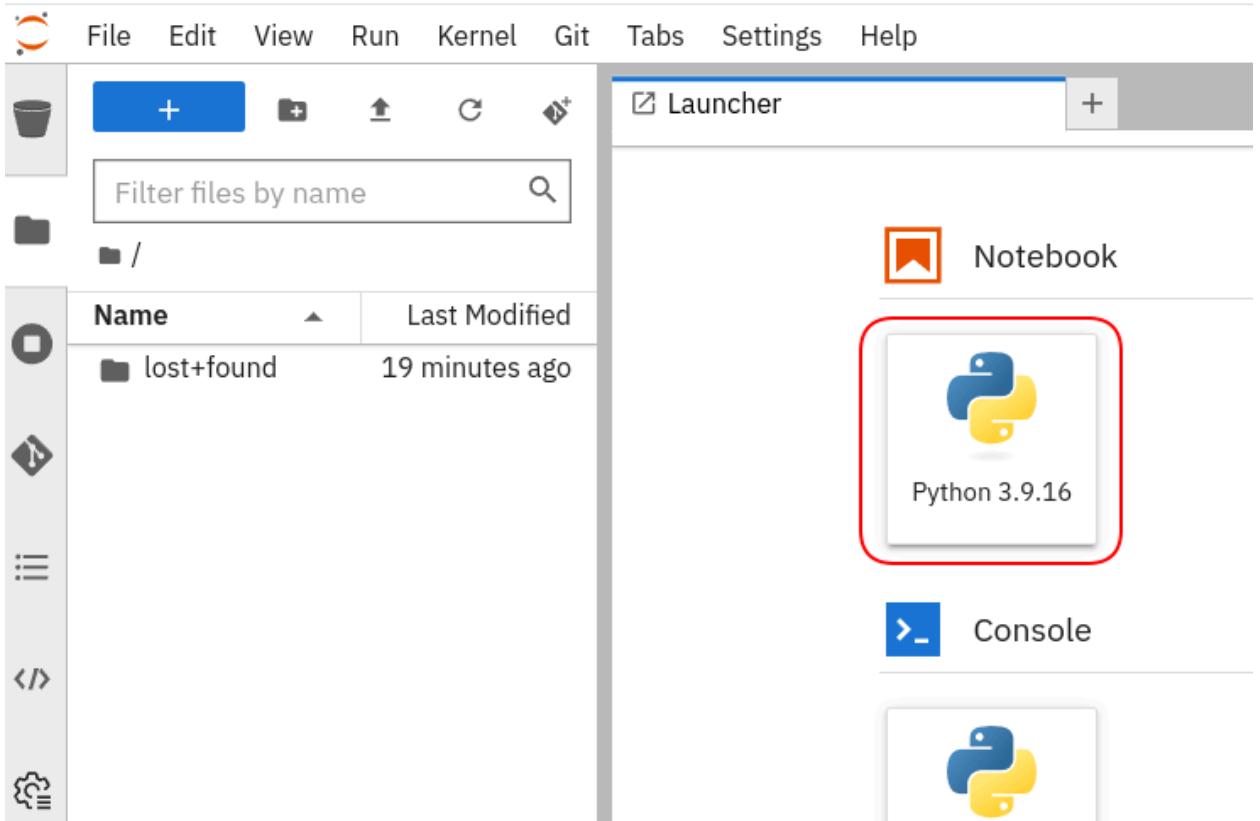
10.

11. Login to the notebook and click the Python 3.9.16 notebook tile.

Data science projects > rhods-custom-image-project

rhods-custom-image-project

Components	Permissions										
Jump to section	Workbenches Create workbench										
Workbenches	<table border="1"> <thead> <tr> <th>Name</th> <th>Notebook image</th> <th>Container size</th> <th>Status</th> <th>⋮</th> </tr> </thead> <tbody> <tr> <td>rhods-custom-image-workbench</td> <td>rhods-admin-custom-image</td> <td>Small</td> <td><input checked="" type="checkbox"/> Running</td> <td>Open</td> </tr> </tbody> </table>	Name	Notebook image	Container size	Status	⋮	rhods-custom-image-workbench	rhods-admin-custom-image	Small	<input checked="" type="checkbox"/> Running	Open
Name	Notebook image	Container size	Status	⋮							
rhods-custom-image-workbench	rhods-admin-custom-image	Small	<input checked="" type="checkbox"/> Running	Open							
Cluster storage											



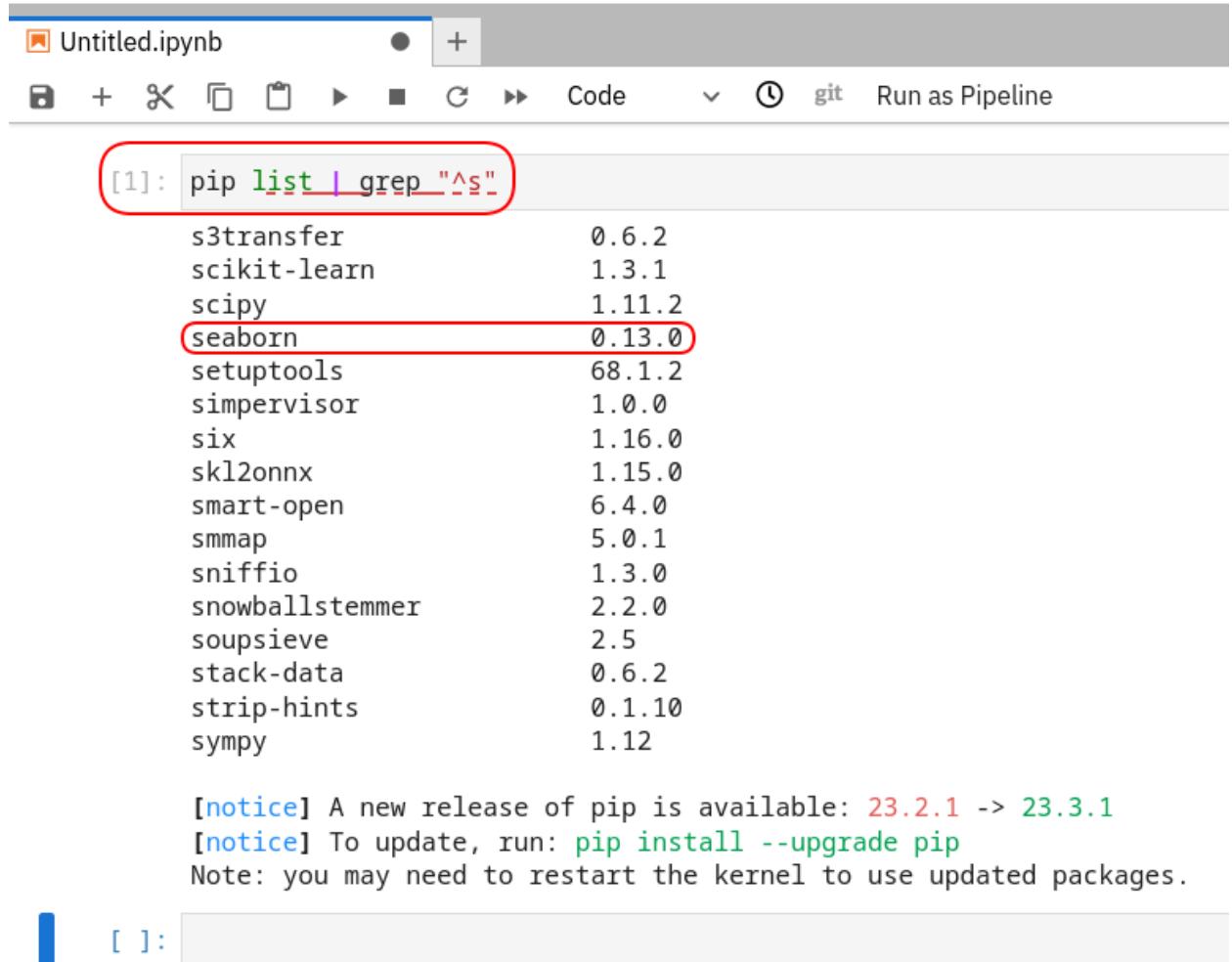
12.

13. In the first cell of the notebook add the following

```
Unset
```

```
pip list | grep '^s"
```

14. Press Shift + Enter on your keyboard to execute the cell. This will find all of the dependencies that start with "s". Note that seaborn is an installed dependency.



A screenshot of a Jupyter Notebook interface. The title bar says "Untitled.ipynb". Below it is a toolbar with icons for file operations and a "Code" button. The main area shows a code cell with the command `pip list | grep '^s_'`. The output lists various Python packages and their versions. The package `seaborn` is highlighted with a red box. The entire output block has a red box around it. At the bottom of the output, there are three notices about pip updates.

```
[1]: pip list | grep '^s_'

s3transfer          0.6.2
scikit-learn        1.3.1
scipy               1.11.2
seaborn              0.13.0
setuptools          68.1.2
simpervisor         1.0.0
six                 1.16.0
skl2onnx             1.15.0
smart-open           6.4.0
smmap                5.0.1
sniffio              1.3.0
snowballstemmer      2.2.0
soupsieve            2.5
stack-data           0.6.2
strip-hints          0.1.10
sympy                1.12

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: pip install --upgrade pip
Note: you may need to restart the kernel to use updated packages.
```

15.

16. Now let's test out the seaborn package to make sure it works. Copy the following code to the cell below.

Python

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load the dowjones dataset
dowjones = sns.load_dataset("dowjones")

# Create a line plot with the year on the x-axis
sns.lineplot(
    x="Date",
    y="Price",
```

```

    data=dowjones,
)

# Set the title and labels
plt.title("Dow Jones Industrial Average by Year - 1914 to 1968")
plt.xlabel("Date")
plt.ylabel("Price")

# Show the plot
plt.show()

```

The screenshot shows a Jupyter Notebook interface with a single code cell selected. The cell contains the following Python code:

```

[3]: import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Load the dowjones dataset
dowjones = sns.load_dataset("dowjones")

# Create a line plot with the year on the x-axis
sns.lineplot(
    x="Date",
    y="Price",
    data=dowjones,
)

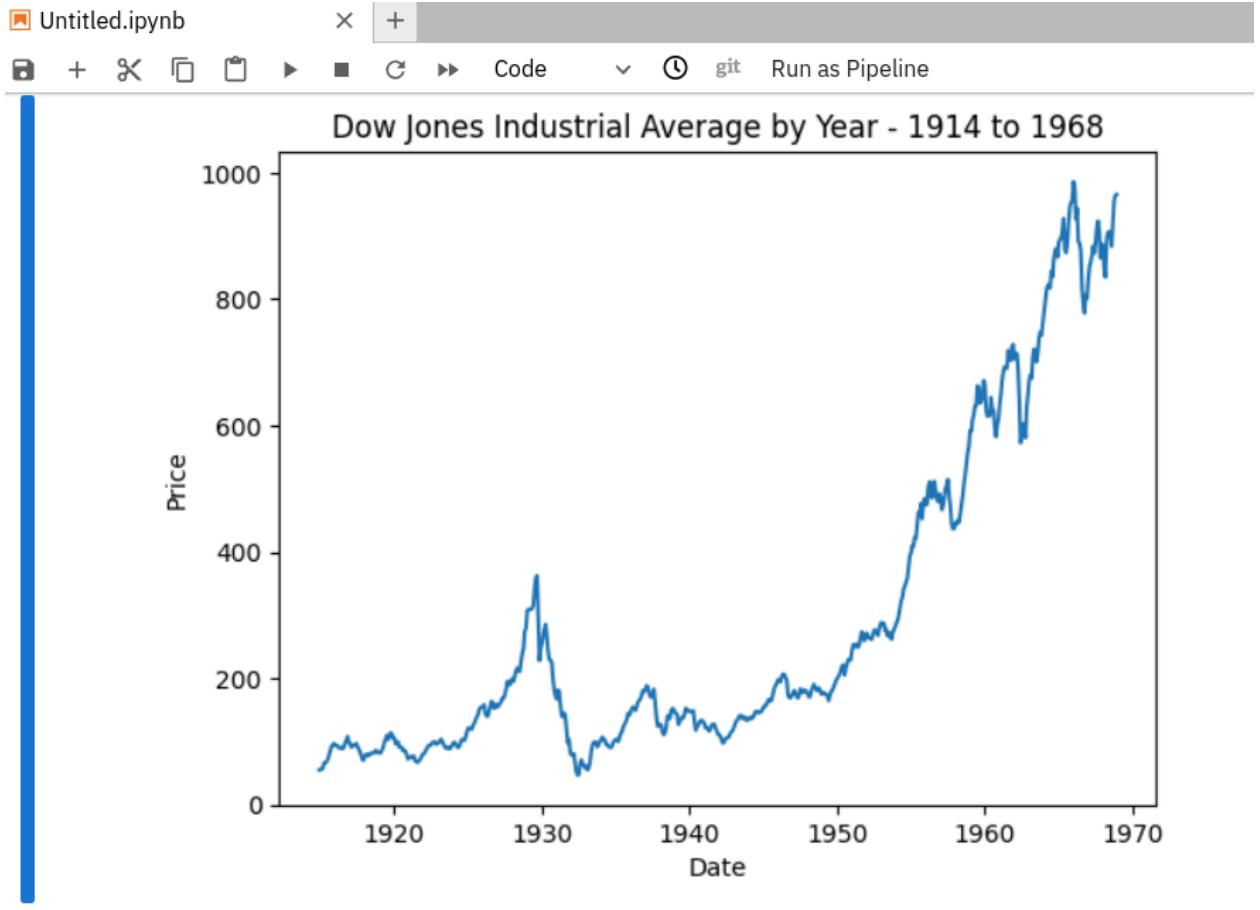
# Set the title and labels
plt.title("Dow Jones Industrial Average by Year - 1914 to 1968")
plt.xlabel("Date")
plt.ylabel("Price")

# Show the plot
plt.show()

```

17.

18. Click Shift + Enter to execute the cell. You should see the graph below. You can now close the notebook. Don't worry about saving it.



19.

Congratulations! You've successfully built, imported, and tested a custom image in OpenShift AI.

Creating a Custom Model Serving Runtime

from:

<https://redhatquickcourses.github.io/rhods-deploy/rhods-deploy/1.33/chapter1/section3.html>

A model-serving runtime provides integration with a specified model server and the model frameworks that it supports. By default, Red Hat OpenShift AI includes the OpenVINO Model Server runtime. However, if this runtime doesn't meet your needs (it doesn't support a particular model framework, for example), you might want to add your own, custom runtimes.

As an administrator, you can use the OpenShift AI interface to add and enable custom model-serving runtimes. You can then choose from your enabled runtimes when you create a new model server.

Prerequisite

In order to run this exercise, be sure to have handy the model we created in the previous section, that is:

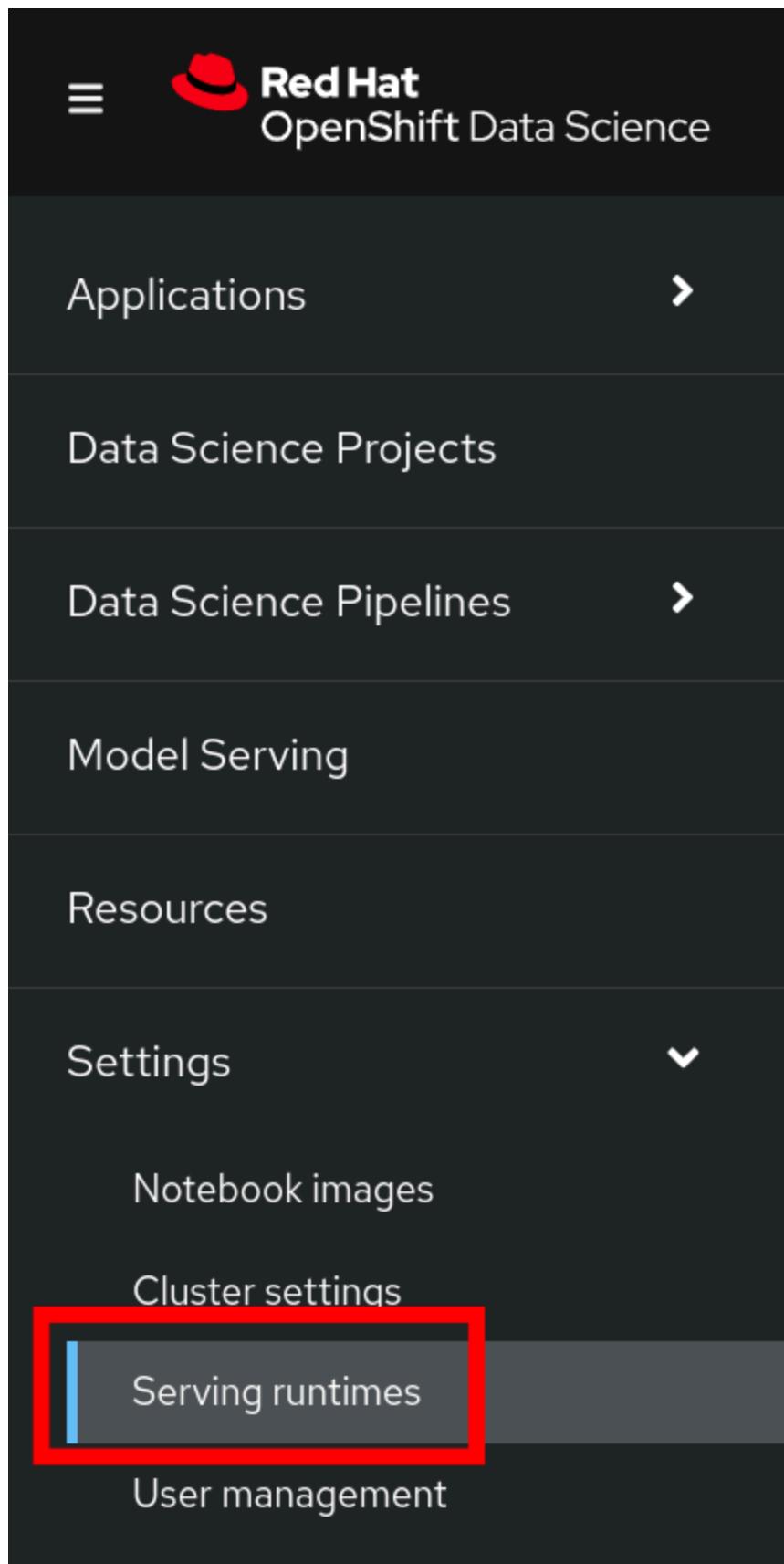
- An s3 bucket with a model in format onnx
- A Data Science project with the name iris-project
- A data connection to S3 with the name iris-data-connection

This exercise will guide you through the broad steps necessary to deploy a custom Serving Runtime in order to serve a model using the Triton Runtime (NVIDIA Triton Inference Server).

While RHOAI supports the ability to add your own runtime, it does not support the runtimes themselves. Therefore, it is up to you to configure, adjust and maintain your custom runtimes.

Adding The Custom Runtime

1. Log in to RHOAI with a user who is part of the RHOAI admin group
2. Navigate to the Settings menu, then Serving Runtimes



3.

4. Click on the Add Serving Runtime button:
-

Serving runtimes

Manage model serving runtimes

Add serving runtime

5.

6. Click on Start from scratch and in the window that opens up, paste the following YAML:

```
Unset
apiVersion: serving.kserve.io/v1alpha1
kind: ServingRuntime
metadata:
  name: triton-23.05-20230804
  labels:
    name: triton-23.05-20230804
  annotations:
    maxLoadingConcurrency: "2"
    openshift.io/display-name: "Triton runtime 23.05"
spec:
  supportedModelFormats:
    - name: keras
      version: "2"
      autoSelect: true
    - name: onnx
      version: "1"
      autoSelect: true
    - name: pytorch
```

```
version: "1"
autoSelect: true
- name: tensorflow
  version: "1"
  autoSelect: true
- name: tensorflow
  version: "2"
  autoSelect: true
- name: tensorrt
  version: "7"
  autoSelect: true

protocolVersions:
- grpc-v2
multiModel: true

grpcEndpoint: "port:8085"
grpcDataEndpoint: "port:8001"

volumes:
- name: shm
  emptyDir:
    medium: Memory
    sizeLimit: 2Gi
containers:
- name: triton
  image: nvcr.io/nvidia/tritonserver:23.05-py3
  command: [/bin/sh]
  args:
  - --c
  - 'mkdir -p /models/_triton_models;
    chmod 777 /models/_triton_models;
    exec tritonserver
    "--model-repository=/models/_triton_models"
    "--model-control-mode=explicit"
    "--strict-model-config=false"
```

```
--strict-readiness=false"
"--allow-http=true"
"--allow-sagemaker=false"
'

volumeMounts:
- name: shm
  mountPath: /dev/shm
resources:
requests:
cpu: 500m
memory: 1Gi
limits:
cpu: "5"
memory: 1Gi
livenessProbe:
# the server is listening only on 127.0.0.1, so an httpGet probe
sent
# from the kublet running on the node cannot connect to the server
# (not even with the Host header or host field)
# exec a curl call to have the request originate from localhost in
the
# container
exec:
command:
- curl
- --fail
- --silent
- --show-error
- --max-time
- "9"
- http://localhost:8000/v2/health/live
initialDelaySeconds: 5
periodSeconds: 30
timeoutSeconds: 10
builtInAdapter:
serverType: triton
```

```
runtimeManagementPort: 8001  
memBufferBytes: 134217728  
modelLoadingTimeoutMillis: 90000
```

- After clicking the Add button at the bottom of the input area, we are able to see the new Runtime in the list. We can re-order the list as needed (the order chosen here is the order in which the users will see these choices)

Serving runtimes

Manage model serving runtimes

Name	Enabled	⋮
Triton runtime 23.05 ⓘ	<input checked="" type="checkbox"/>	⋮
OpenVINO Model Server ⓘ Pre-installed	<input checked="" type="checkbox"/>	⋮
OpenVINO Model Server (Supports GPUs) ⓘ Pre-installed	<input checked="" type="checkbox"/>	⋮

- 8.

Creating The Model Server

- Using the iris-project created in the previous section, scroll to the Models and model servers section, and select the Add server button

The screenshot shows the ModelDB interface with the following sections:

- Storage:** Shows "No storage". A callout says: "To get started, add existing or create new cluster storage."
- Data connections:** Shows "Data connections" with an "Add data connection" button. A table lists one connection: "iris-dataconnection" (Type: Object storage, Provider: AWS S3).
- Pipelines:** Shows "Pipelines" with an "Import pipeline" button.
- Model Servers:** Shows "Models and model servers" with an "Add server" button highlighted by a red arrow. A table lists one server: "iris-model-server" (Serving Runtime: OpenVINO Model Server, Deployed models: 1, Tokens: 1). A "Deploy model" button is also present.

2.

3. Create the model server with the following values:

- Server name: **iris-custom-server**.
- Serving runtime: **Triton runtime 23.05**. This is the newly added runtime.
- Activate the external route and the authentication. Use **custom-server-sa** as the service account name.

Add model server

Model server name *
iris-custom-server

Serving runtime *
Triton runtime 23.05

Model server replicas

Number of model server replicas to deploy
1

Compute resources per replica

Model server size
Small

Model route

Make deployed models available through an external route

Token authorization

Require token authentication

Service account name
custom-server-sa

Enter the service account name for which the token will be generated

[+ Add a service account](#)

Add **Cancel**

The actual tokens will be created and displayed when the model server is configured.

- 4.
5. After clicking the Add button at the bottom of the form, we are able to see our iris-custom-server model server, created with the Triton runtime 23.05 serving runtime.

Serving runtimes

Manage model serving runtimes

Add serving runtime

Name	Enabled	⋮
Triton runtime 23.05 ⓘ	Enabled ⓘ	⋮
OpenVINO Model Server ⓘ Pre-installed	Enabled ⓘ	⋮
OpenVINO Model Server (Supports GPUs) ⓘ Pre-installed	Enabled ⓘ	⋮

6.

Deploy The Model

1. Use the Deploy Model button at the right of the row with the iris-custom-server model server

Jump to section

No storage
To get started, add existing or create new cluster storage.

Data connections [Add data connection](#)

Name	Type	Connected workbenches	Provider	⋮
iris-dataconnection ⓘ	Object storage	No connections	AWS S3	⋮

Pipelines [Import pipeline](#)

No pipeline server
To import a pipeline, first create a pipeline server.

[Create a pipeline server](#)

Models and model servers [Add server](#)

Model Server Name	Serving Runtime	Deployed models	Tokens	⋮
iris-model-server	OpenVINO Model Server	1	1	Deploy model ⋮

2.

3. Fill up the Deploy Model form:

- Model name: **iris-custom-model**
- Model framework: **onnx - 1**
- Model location data connection: **iris-data-connection**
- Model location path: **iris**

Deploy model

Configure properties for deploying your model

Project
iris-project

Model Name *
iris-custom-model

Model server
iris-custom-server

Model framework (name - version) *
onnx - 1

Model location

Existing data connection

Name *
iris-dataconnection

Path *
/ iris

Enter a path to a model or folder. This path cannot point to a root folder.

New data connection

Deploy **Cancel**

4.

5.

Notice the model name, in this exercise we are naming it **iris-custom-model**, we can't use the *iris-model* name anymore. You can be creative and name it differently, just mind your selection when running the inference service with the APIs.

6.

After clicking the Deploy button at the bottom of the form, we see the model added to our Model Server row, wait for the green checkmark to appear.

The screenshot shows the ModelDB interface. On the left, there's a sidebar with 'Jump to section' and links for Workbenches, Pipelines (selected), Cluster storage, Data connections, Pipelines, and Models and model servers. The main area has two sections:

- Pipelines**: Shows a single pipeline named "iris-dataconnection" (Object storage type, AWS S3 provider). It says "No pipeline server" and has a "Create a pipeline server" button.
- Models and model servers**: Shows a table with one entry: "iris-custom-server" (Triton runtime 23.05). It has columns for Model Server Name, Serving Runtime, Deployed models (1), Tokens (1), and a "Deploy model" button.

7.

Test The Model With CURL

Now that the model is ready to use, we can make an inference using the REST API

1. Assign the route to an environment variable in your local machine, so that we can use it in our curl commands.

Unset

```
$ export IRIS_ROUTE=$(oc get routes -n iris-project | grep iris-custom-model | awk '{print $2}')
```

2. Assign an authentication token to an environment variable in your local machine.

Unset

```
$ export TOKEN=$(oc whoami -t)
```

3. Request an inference with the REST API.

Unset

```
$ curl -H "Authorization: Bearer $TOKEN" \
$IRIS_ROUTE/v2/models/iris-custom-model/infer -X POST \
```

```
--data '{"inputs": [{"name": "X", "shape": [1, 4], "datatype": "FP32", "data": [3, 4, 3, 2]}]}'
```

4. The result received from the inference service looks like the following:

```
Unset
```

```
{"model_name": "iris-custom-model__isvc-9cc7f4ebab", "model_version": "1", "outputs": [{"name": "label", "datatype": "INT64", "shape": [1, 1], "data": [1]}, {"name": "scores", "datatype": "FP32", "shape": [1, 3], "data": [4.851966, 3.1275778, 3.4580243]}]}
```

GPUs and accelerators

Installation of GPU dependencies:

(from

https://redhatquickcourses.github.io/rhods-admin/rhods-admin/1.33/chapter1/dependencies-install-web-console.html#_installation_of_gpu_dependencies)

Installing Dependencies Using the Web Console

As described in the [General Information about Installation](#) section you may need to install other operators depending on the components and features of OpenShift AI you want to use. This section will discuss installing and configuring those components.

It is generally recommended to install any dependent operators prior to installing the Red Hat OpenShift AI operator.

[Red Hat OpenShift Pipelines Operator](#)

The Red Hat OpenShift Pipelines Operator is required if you want to install the Data Science Pipelines component.

[NVIDIA GPU Operator](#)

The NVIDIA GPU Operator is required for GPU support in Red Hat OpenShift AI.

[Node Feature Discovery Operator](#)

The Node Feature Discovery Operator is a prerequisite for the NVIDIA GPU Operator. This section will discuss the process for installing the dependent operators using the OpenShift Web Console.

Installation of Data Science Pipelines Dependencies

The Data Science Pipelines component utilizes Red Hat OpenShift Pipelines as an execution engine for all pipeline runs, and is required to be installed to take advantage of the Data Science Pipelines component.

The following section discusses installing the Red Hat OpenShift Pipelines operator.

Demo: Installation of the Red Hat OpenShift Pipelines operator

1. Login to Red Hat OpenShift using a user which has the *cluster-admin* role assigned.
2. Navigate to Operators → OperatorHub and search for Red Hat OpenShift Pipelines

The screenshot shows the Red Hat OpenShift web interface. The left sidebar is titled 'Administrator' and includes sections for Home, Operators (selected), Workloads, Networking, Storage, Builds, and Observe. Under Operators, there is a sub-section for 'OperatorHub' with 'Installed Operators'. The main content area is titled 'OperatorHub' and displays a search bar with the query 'Red Hat OpenShift Pipelines'. A modal window is open, showing a card for 'Red Hat OpenShift Pipelines' provided by Red Hat. The card includes a logo, the name, a brief description, and a link to the developer catalog.

- 3.
4. Click on the Red Hat OpenShift Pipelines operator and in the pop up window click on Install to open the operator's installation view.

The screenshot shows the Red Hat OpenShift web interface with the 'OperatorHub' section selected. The search bar at the top contains 'Red Hat OpenShift Pipelines'. On the right, a detailed view of the 'Red Hat OpenShift Pipelines' operator is shown. The 'Install' button is highlighted in blue. The page provides information about the latest version (1.2.0), capability levels (Basic Install, Seamless Upgrades, Full Lifecycle), source (Red Hat), provider (Red Hat), infrastructure features (Disconnected, Proxy-aware), valid subscriptions (OpenShift Container Platform, OpenShift Platform Plus), repository (https://github.com/tekton/ondc/operator), components (Tekton Pipelines v0.50.1, Tekton Triggers v0.21.0, Pipelines-as-Code v0.17.0, Tekton Chains v0.17.0), and an installation note about the operator getting installed into a single namespace.

- 5.
6. In the installation view choose the Update channel and the Update approval parameters. You can accept the default values. The Installation mode and the Installed namespace parameters are fixed.

The screenshot shows the Red Hat OpenShift web interface. On the left is a sidebar with navigation links: Administrator, Home, Operators (selected), OperatorHub (highlighted), Installed Operators, Workloads, Networking, Storage, Builds, Observe, Compute, User Management, and Administration. The main content area is titled "Install Operator" and says "Install your Operator by subscribing to one of the update channels". It includes fields for "Update channel *": latest (selected), pipelines-1.10, pipelines-1.11, pipelines-1.12, and pipelines-1.9. It also includes fields for "Installation mode *": All namespaces on the cluster (default) and A specific namespace on the cluster (disabled). The "Installed Namespace *" field is set to "openshift-operators". The "Update approval *" field has "Automatic" selected. At the bottom are "Install" and "Cancel" buttons.

OperatorHub > Operator Installation

Install Operator

Install your Operator by subscribing to one of the update channels

Update channel * ?

latest

pipelines-1.10

pipelines-1.11

pipelines-1.12

pipelines-1.9

Installation mode *

All namespaces on the cluster (default)
Operator will be available in all Namespaces.

A specific namespace on the cluster
This mode is not supported by this Operator

Installed Namespace *

PR openshift-operators

Update approval * ?

Automatic

Manual

Install Cancel

7.

8. Click on the Install button at the bottom of to view the to proceed with the installation. A window showing the installation progress will pop up.

The screenshot shows a window titled "Red Hat OpenShift Pipelines" version 1.12.0 provided by Red Hat. Below the title, it says "Installing Operator". It displays a message: "The Operator is being installed. This may take a few minutes." and a link "View installed Operators in Namespace openshift-operators".

- 9.
10. When the installation finishes the operator is ready to be used by Red Hat OpenShift AI.

The screenshot shows a window titled "Red Hat OpenShift Pipelines" version 1.12.0 provided by Red Hat. It has a green checkmark icon in the top right corner. Below the title, it says "Installed operator: ready for use" and features two buttons: "View Operator" and "View installed Operators in Namespace openshift-operators".

- 11.
- Red Hat OpenShift Pipelines is now successfully installed.

For assistance installing OpenShift Pipelines from YAML or via ArgoCD, refer to examples found in the [redhat-cop/gitops-catalog/openshift-pipelines-operator](<https://github.com/redhat-cop/gitops-catalog/tree/main/openshift-pipelines-operator>) GitHub repo.

Installation of GPU Dependencies

Red Hat OpenShift AI makes it easy to expose GPUs to end users to help accelerate training and serving machine learning models.

Currently, Red Hat OpenShift AI supports accelerated compute with NVIDIA GPUs using the NVIDIA GPU Operator which relies on the Node Feature Discovery operator as a dependency.

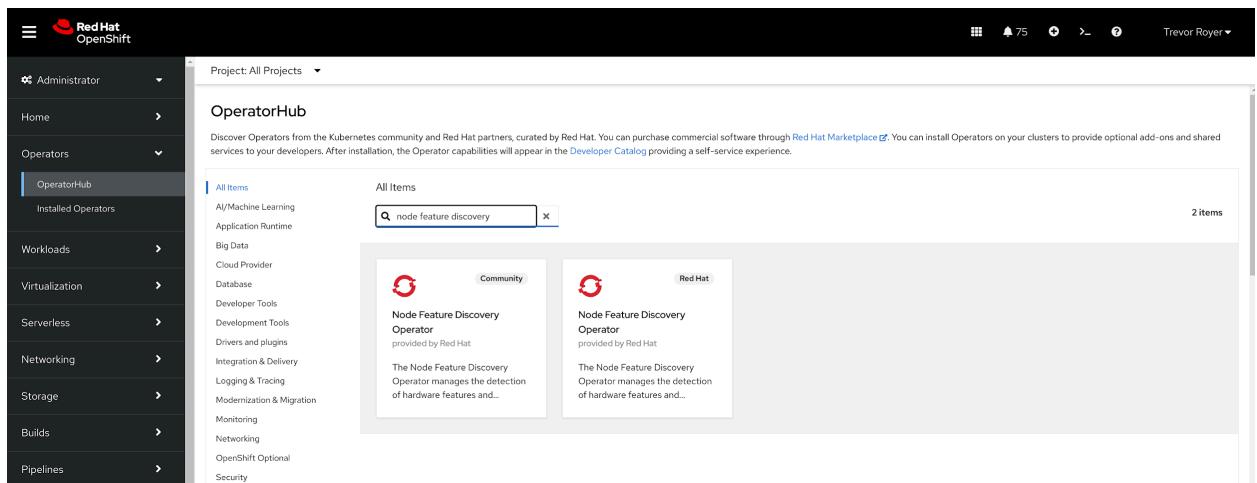
The following section will discuss the installation and a basic configuration of both NVIDIA GPU Operator and the Node Feature Discovery operator.

Node Feature Discovery and the NVIDIA GPU Operator can both be installed in a cluster that does not have a node with a GPU. This can be helpful when you plan to add GPUs at a later date. If a GPU is not present in the cluster the Dashboard will not present the user an option to deploy using a GPU.

To view the list of GPU models supported by the NVIDIA GPU Operator refer to the [Supported NVIDIA GPUs and Systems](#) docs.

Demo: Installation of the Node Feature Discovery operator

1. Login to Red Hat OpenShift using a user which has the *cluster-admin* role assigned.
2. Navigate to Operators → OperatorHub and search for Node Feature Discovery



The screenshot shows the Red Hat OpenShift OperatorHub interface. The left sidebar shows navigation options like Home, Operators (selected), Workloads, Virtualization, Serverless, Networking, Storage, Builds, and Pipelines. The main area is titled 'OperatorHub' and displays search results for 'node feature discovery'. It shows two items: one from the 'Community' and one from 'Red Hat'. Both items have a red circular icon with a white arrow. The 'Community' item is described as 'provided by Red Hat' and the 'Red Hat' item is described as 'provided by Red Hat'. A search bar at the top right contains the query 'node feature discovery'.

- 3.
4. Two options for the Node Feature Discovery operator will be available. Click on the one with Red Hat in the top right hand corner and in the pop up window click on Install to open the operator's installation view.

Make sure you select Node Feature Discovery from Red Hat not the Community version.

5.

- In the installation view check the box to Enable Operator recommended cluster monitoring on this Namespace and the Update approval parameters if desired. Leave the Update channel, Version, and the Installed Namespace parameters as the default options.

Some of these options may vary slightly depending on your version of OpenShift. Please refer to the official Node Feature Discovery Documentation for your version of OpenShift for the recommended settings.

7.

- Click on the Install button at the bottom of the page to proceed with the installation. A window showing the installation progress will pop up.



Node Feature Discovery Operator

nfd.4.14.0-202311021650 provided by Red Hat

Installing Operator

The Operator is being installed. This may take a few minutes.

[View installed Operators in Namespace openshift-nfd](#)

9.

10. When the installation finishes the operator to be configured. Click the button to View Operator.



Node Feature Discovery Operator

nfd.4.14.0-202311021650 provided by Red Hat



Installed operator: ready for use

[View Operator](#)

[View installed Operators in Namespace openshift-nfd](#)

11.

12. Click the Create instance button for the NodeFeatureDiscovery object.

Project: openshift-nfd ▾

Installed Operators > Operator details



Node Feature Discovery Operator
4.14.0-202311021650 provided by Red Hat

Details YAML Subscription Events All instances NodeFeatureDiscovery NodeFeatureRule

Provided APIs

NFD NodeFeatureDiscovery

The NodeFeatureDiscovery instance is the CustomResource being watched by the NFD-Operator, and holds all the needed information to setup the behaviour of the master and worker pods

[+ Create instance](#)

NFR NodeFeatureRule

NodeFeatureRule resource specifies a configuration for feature-based customization of node objects, such as node labeling.

[+ Create instance](#)

13.

14. Leave the default options for NodeFeatureDiscovery selected, and click the Create button.

Project: openshift-nfd ▾

Create NodeFeatureDiscovery

Create by completing the form. Default values may be provided by the Operator authors.

Configure via: Form view YAML view

! Note: Some fields may not be represented in this form view. Please select "YAML view" for full control.

Name *

nfd-instance

Labels

app=frontend

provided by Red Hat

The NodeFeatureDiscovery instance is the CustomResource being watched by the NFD-Operator, and holds all the needed information to setup the behaviour of the master and worker pods

extraLabelNs

ExtraLabelNs defines the list of allowed extra label namespaces. By default, only allow labels in the default `feature.node.kubernetes.io` label namespace

instance

Instance name. Used to separate annotation namespaces for multiple parallel deployments.

labelWhiteList

LabelWhiteList defines a regular expression for filtering feature labels based on their name. Each label must match against the given regular expression in order to be published.

15.

16. A new set of pods should appear in the Workloads → Pods section managed by the nfd-worker DaemonSet. Node Feature Discovery will now be able to automatically detect information about the nodes in the cluster and apply labels to those nodes.

17.

Name	Status	Ready	Restarts	Owner	Memory	CPU	Created
nfd-controller-manager-777944c5f-7pjx8	Running	2/2	0	nfd-controller-manager-777944c5f	93.4 MiB	0.003 cores	Nov 20, 2023, 10:46 AM
nfd-master-5bd9f8855f-jhcc	Running	1/1	0	nfd-master-5bd9f8855f	-	-	Nov 20, 2023, 10:56 AM
nfd-worker-99p7t	Running	1/1	0	nfd-worker	-	-	Nov 20, 2023, 10:56 AM
nfd-worker-bwfxv	Running	1/1	0	nfd-worker	-	-	Nov 20, 2023, 10:56 AM
nfd-worker-km7c5	Running	1/1	0	nfd-worker	-	-	Nov 20, 2023, 10:56 AM

For assistance installing the Node Feature Discovery Operator from YAML or via ArgoCD, refer to examples found in the [redhat-cop/gitops-catalog/nfd](<https://github.com/redhat-cop/gitops-catalog/tree/main/nfd>) GitHub repo.

Node Feature Discovery is now successfully installed and configured.

Demo: Installation of the NVIDIA GPU Operator

1. Login to Red Hat OpenShift using a user which has the *cluster-admin* role assigned.
2. Navigate to Operators → OperatorHub and search for NVIDIA GPU Operator

3.

All Items

All Items

Q. nvidia gpu

NVIDIA GPU Operator

provided by NVIDIA Corporation

Automate the management and monitoring of NVIDIA GPUs.

4. Click the NVIDIA GPU Operator tile and in the pop up window click on Install to open the operator's installation view.

NVIDIA GPU Operator
23.9.0 provided by NVIDIA Corporation

Latest version
23.9.0

Capability level

- Basic Install
- Seamless Upgrades
- Full Lifecycle
- Deep Insights
- Auto Pilot

Source
Certified

Provider
NVIDIA Corporation

Infrastructure features
Disconnected

Repository
<http://github.com/NVIDIA/gpu-operator>

Container image
nvcr.io/nvidia/gpu-operator@sha256:3d76a362ca957abc31a22a1c13a3a3d0dc

- 5.
6. In the installation view update the Update channel and Update approval parameters if desired. Leave the Installation mode and the Installed namespace parameters as the default options.

Update channel *

- stable
- v1.0
- v1.1
- v2.9
- v23.3
- v23.6
- v23.9**

Installation mode *

- All namespaces on the cluster (default)
This mode is not supported by this Operator
- A specific namespace on the cluster**
Operator will be available in a single Namespace only.

Installed Namespace *

ClusterPolicy
ClusterPolicy allows you to configure the GPU Operator

NVIDIA Driver
NVIDIA Driver allows you to deploy the NVIDIA driver

Namespace creation
Namespace nvidia-gpu-operator does not exist and will be created.

Select a Namespace

Update approval *

- Automatic
- Manual

- 7.
8. Click on the Install button at the bottom of the page to proceed with the installation. A window showing the installation progress will pop up.

 NVIDIA GPU Operator
23.9.0 provided by NVIDIA Corporation

Installing Operator

The Operator is being installed. This may take a few minutes.
[View installed Operators in Namespace nvidia-gpu-operator](#)

9.

10. When the installation finishes the operator to be configured. Click the button to View Operator.

 NVIDIA GPU Operator
23.9.0 provided by NVIDIA Corporation



Installed operator - ready for use

[View Operator](#) [View installed Operators in Namespace nvidia-gpu-operator](#)

11.

12. Click the Create instance button for the ClusterPolicy object.

Project: nvidia-gpu-operator ▾

Installed Operators > Operator details

 NVIDIA GPU Operator
23.9.0 provided by NVIDIA Corporation

[Details](#) [YAML](#) [Subscription](#) [Events](#) [All instances](#) [ClusterPolicy](#) [NVIDIAIDriver](#)

Provided APIs

CP ClusterPolicy
ClusterPolicy allows you to configure the GPU Operator
[Create instance](#)

NVID NVIDIAIDriver
NVIDIAIDriver allows you to deploy the NVIDIA driver
[Create instance](#)

13.

14. Leave the default options for ClusterPolicy selected, and click the Create button.

Project: nvidia-gpu-operator

Create ClusterPolicy

Create by completing the form. Default values may be provided by the Operator authors.

Configure via: Form view YAML view

Note: Some fields may not be represented in this form view. Please select "YAML view" for full control.

Name *

Labels

GPU Operator config *

GPU Operator config

NVIDIA GPU/vGPU Driver config *

NVIDIA GPU/vGPU Driver config

NVIDIA DCGM Exporter config *

NVIDIA DCGM Exporter config

NVIDIA Device Plugin config *

NVIDIA Device Plugin config

15.

16. After the gpu-cluster-policy ClusterPolicy is created, the NVIDIA GPU Operator will update the status of the ClusterPolicy to State: ready.

Project: nvidia-gpu-operator

Installed Operators > Operator details

NVIDIA GPU Operator
23.9.0 provided by NVIDIA Corporation

Actions

Details YAML Subscription Events All instances **ClusterPolicy** NVIDIAIDriver

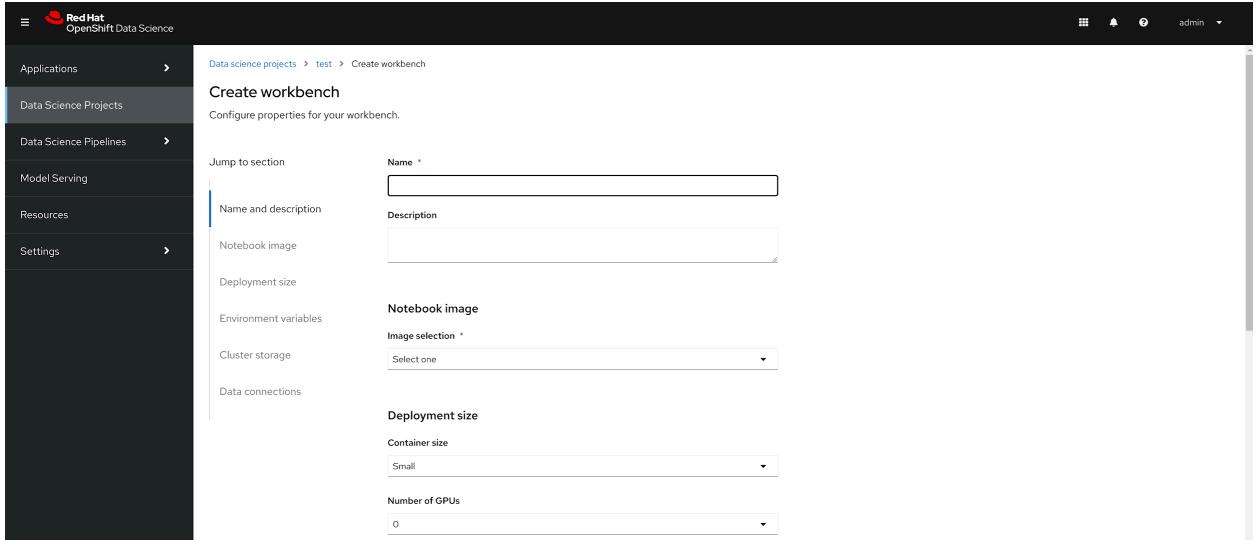
ClusterPolicies

Create ClusterPolicy

Name	Kind	Status	Labels	Last updated	⋮
gpu-cluster-policy	ClusterPolicy	State: ready	No labels	Nov 20, 2023, 11:19 AM	⋮

17.

18. After the Red Hat OpenShift AI operator has been installed and configured, users will be able to see an option for "Number of GPUs" when creating a new workbench.



19.

The Dashboard may initially show "All GPUs are currently in use, try again later." when Red Hat OpenShift AI is first installed. It may take a few minutes after Red Hat OpenShift AI is installed before the GPUs are initially detected.

The NVIDIA GPU Operator supports many advanced use cases such as Multi-Instance GPU (MIG) and Time Slicing that are configurable using the ClusterPolicy. For information about advanced GPU configuration capabilities, refer to the official [NVIDIA Documentation](#).

For assistance installing the NVIDIA GPU Operator from YAML or via ArgoCD, refer to examples found in the [redhat-cop/gitops-catalog/gpu-operator-certified](<https://github.com/redhat-cop/gitops-catalog/tree/main/gpu-operator-certified>) GitHub repo.

Managing Workbench and Model Server Sizes

When launching Workbenches or Model Servers from the Dashboard, users are presented with several default sizes they can select from. The default options may not suit your organizations needs

```
apiVersion: opendatahub.io/v1alpha
kind: OdhDashboardConfig
metadata:
  annotations:
    internal.config.kubernetes.io/previousKinds:
      OdhDashboardConfig
    internal.config.kubernetes.io/previousNames:
      odh-dashboard-config
    internal.config.kubernetes.io/previousNamespaces: default
  name: odh-dashboard-config
  namespace: redhat-ods-applications
  labels:
    app.kubernetes.io/part-of: rhods-dashboard
    app.opendatahub.io/rhods-dashboard: 'true'
spec:
  dashboardConfig:
    modelMetricsNamespace: ''
    enablement: true
    disableProjects: false
    disableSupport: false
    disablePipelines: false
    disableProjectSharing: false
    disableModelServing: false
    disableCustomServingRuntimes: false
    disableISVBadges: false
    disableUserManagement: false
    disableInfo: false
    disableClusterManager: false
    disableBYONImageStream: false
```

```
    disableTracking: false
  groupsConfig:
    adminGroups: rhods-admins
    allowedGroups: 'system:authenticated'
  modelServerSizes:
    - name: Small
      resources:
        limits:
          cpu: '2'
          memory: 8Gi
        requests:
          cpu: '1'
          memory: 4Gi
    - name: Medium
      resources:
        limits:
          cpu: '8'
          memory: 10Gi
        requests:
          cpu: '4'
          memory: 8Gi
    - name: Large
      resources:
        limits:
          cpu: '10'
          memory: 20Gi
        requests:
          cpu: '6'
          memory: 16Gi
  notebookController:
    enabled: true
  notebookNamespace: rhods-notebooks
  notebookTolerationSettings:
    enabled: false
```

```
key: NotebooksOnly
pvcSize: 20Gi
notebookSizes:
- name: Small
  resources:
    limits:
      cpu: '2'
      memory: 8Gi
    requests:
      cpu: '1'
      memory: 8Gi
- name: Medium
  resources:
    limits:
      cpu: '6'
      memory: 24Gi
    requests:
      cpu: '3'
      memory: 24Gi
- name: Large
  resources:
    limits:
      cpu: '14'
      memory: 56Gi
    requests:
      cpu: '7'
      memory: 56Gi
- name: X Large
  resources:
    limits:
      cpu: '30'
      memory: 120Gi
    requests:
      cpu: '15'
```

```
memory: 120Gi  
templateOrder: []
```

The name of the OdhDashboardConfig, which must match
`odh-dashboard-config`

The namespaces where the Dashboard is installed

The default model server sizes, provided as a list

The default workbench sizes, provided by a list

Users will most often select the largest option available to them, despite what their actual needs are. Customizing these options based on the kinds of use cases your organization typically tackles can help to reduce the overall consumption of resources on a cluster.

OpenShift provides the ability to restrict how many resources, including CPU, memory, and GPUs, are consumed by each project on a cluster with a LimitRange. To learn more about these capabilities refer to the [Restrict resource consumption with limit ranges](#) documentation.