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:

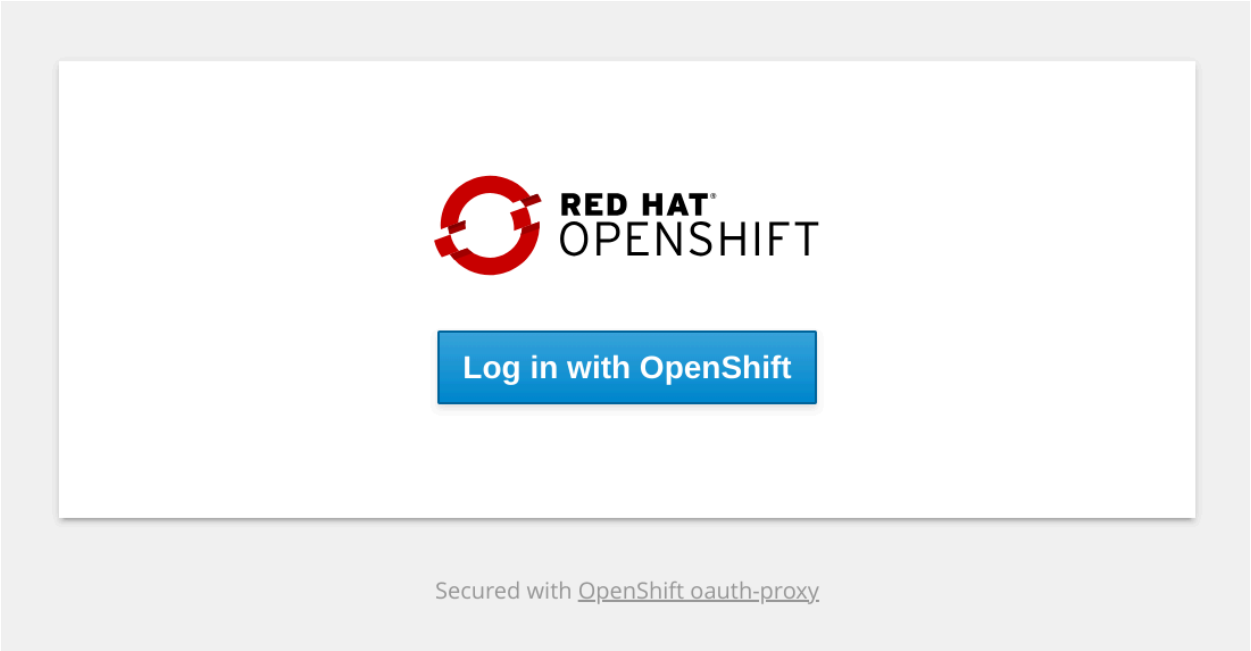
- Manage access to the OpenShift AI Dashboard
- Control access to data science projects with users and groups
- Configure automatic cleanup of idle workbenches
- Manage default workbench and model server sizes

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:

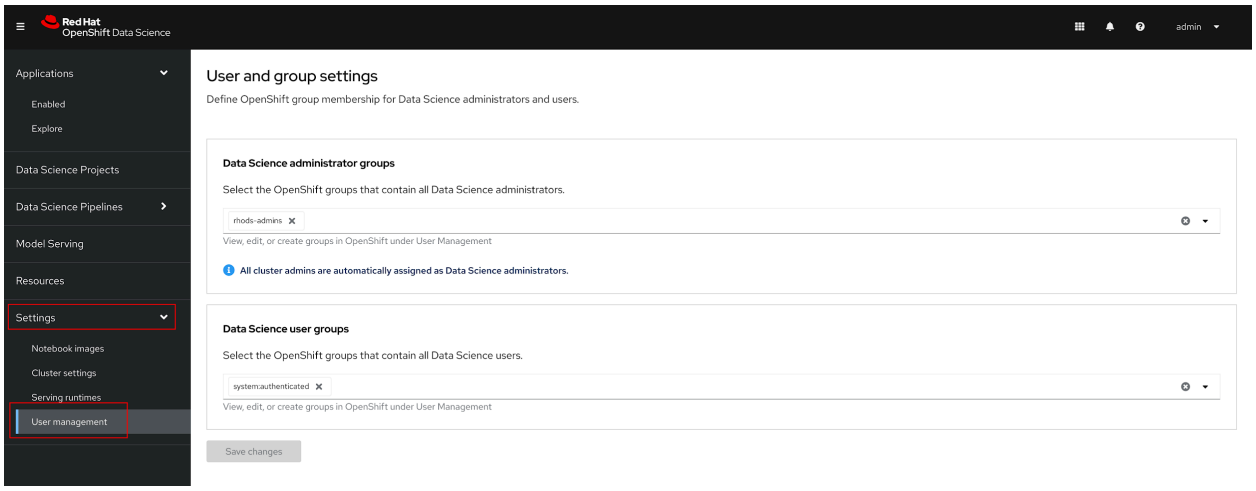
- Configuring custom notebook images
- Configuring custom serving runtimes
- Configuring the idle notebook culler (automatic removal of inactive notebooks)
- Other notebook configuration settings

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 users 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 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
```

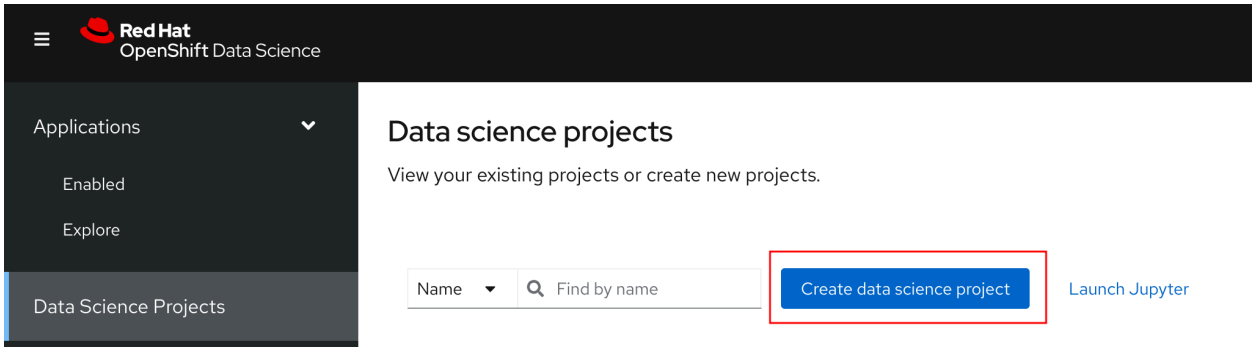
A truncated version of the OdhDashboardConfig object with the groupConfig settings can be found below:

```
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 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 vise 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.

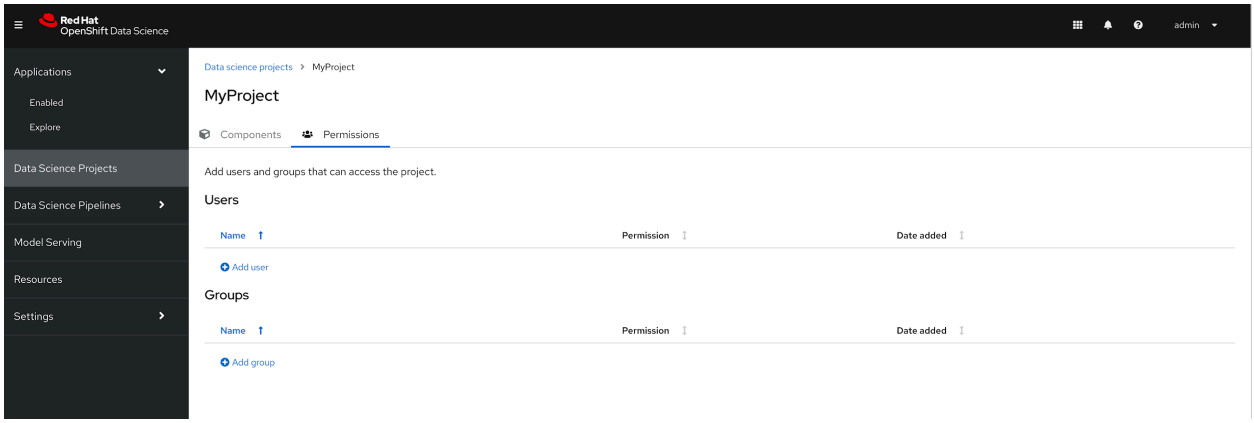
> For information on how to disable self-provisioner on OpenShift refer to the official Disabling project self-provisioning OpenShift documentation.

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.



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.

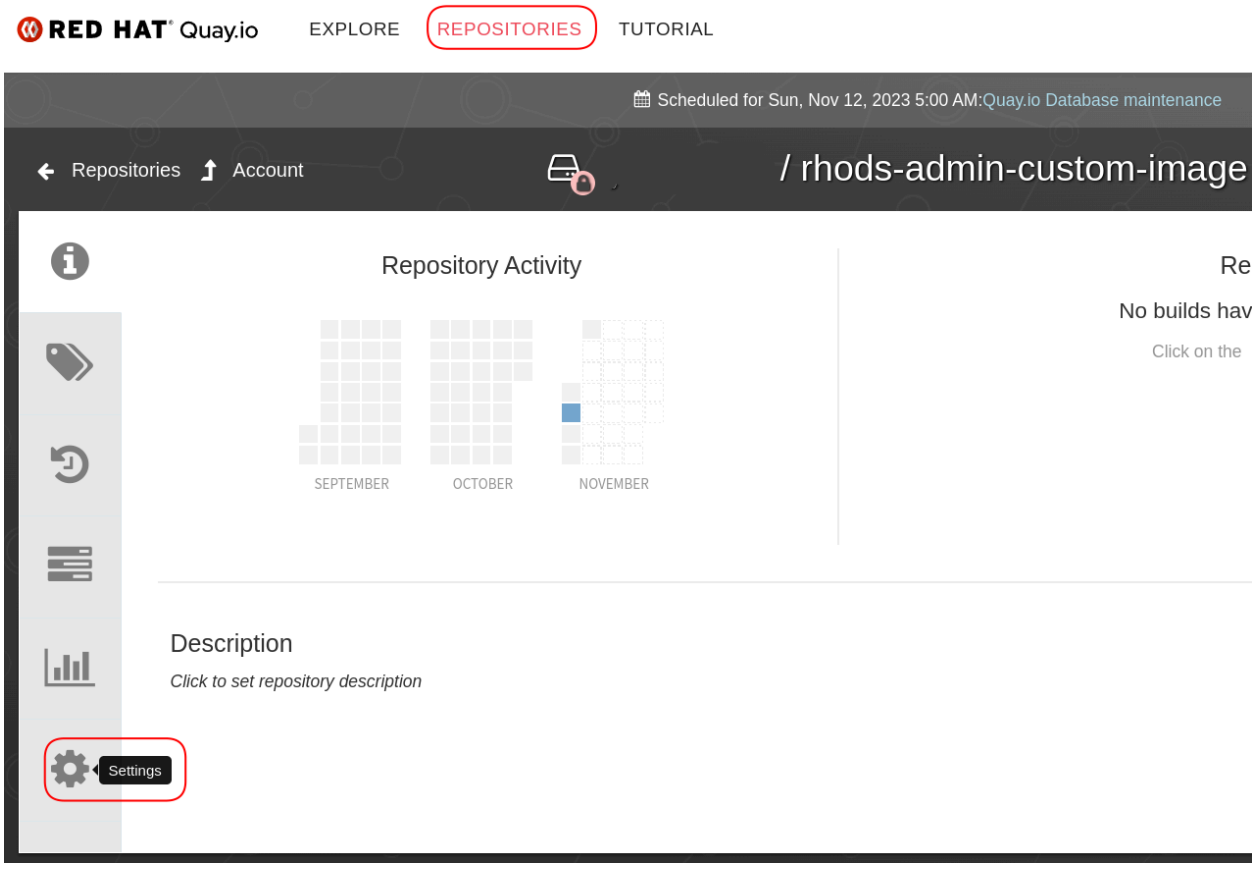
Importing Custom Notebook Images

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

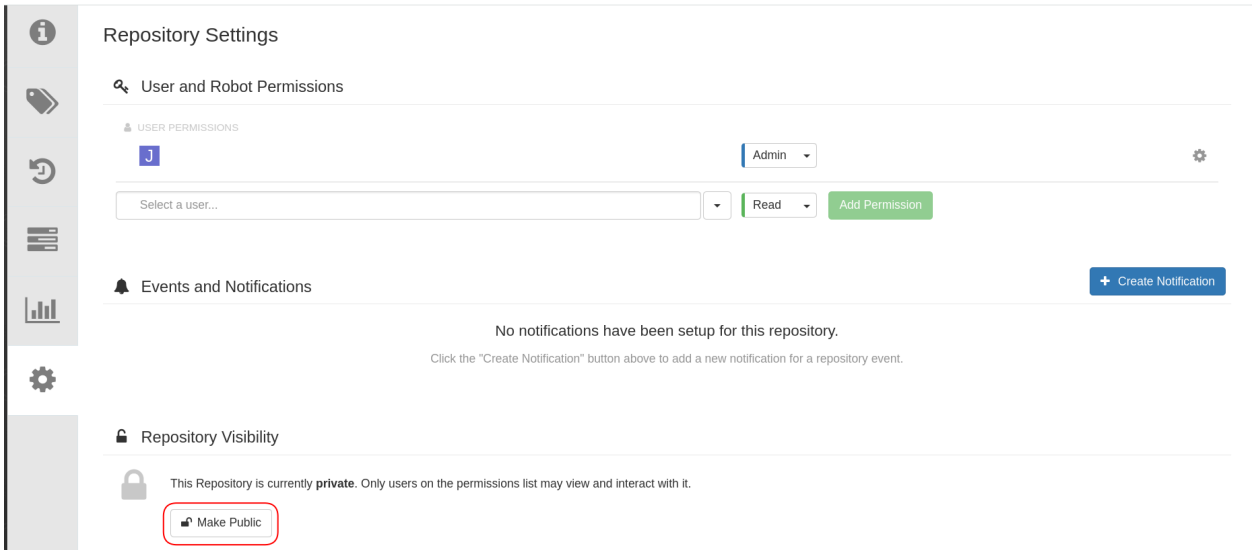
In this section we will go over importing a custom notebook through the OpenShift AI dashboard and test it to make sure our dependencies are included.

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.

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
```

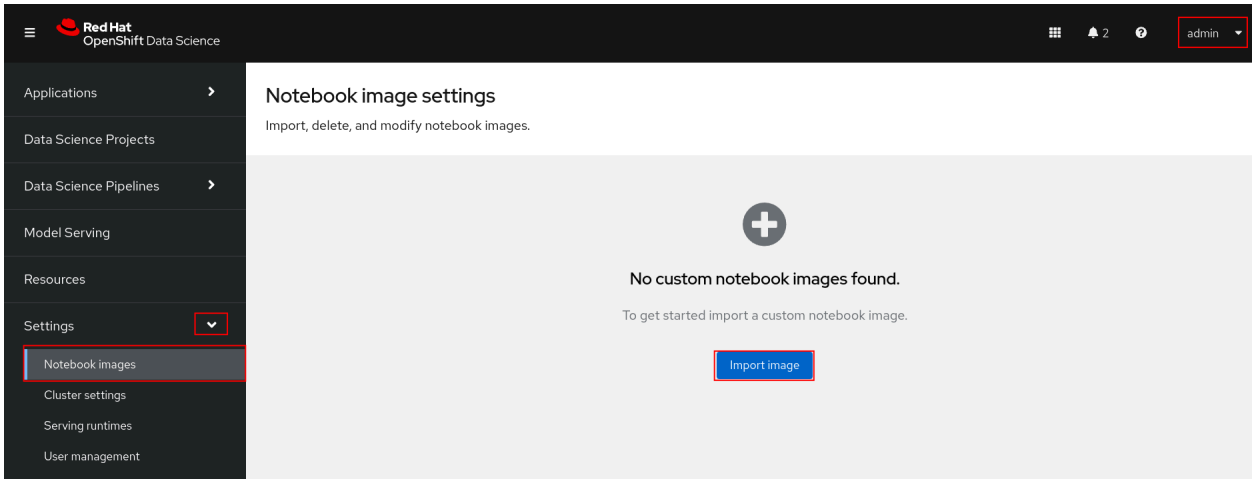
Finally, link the secret to the `default` service account as a pull secret:

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

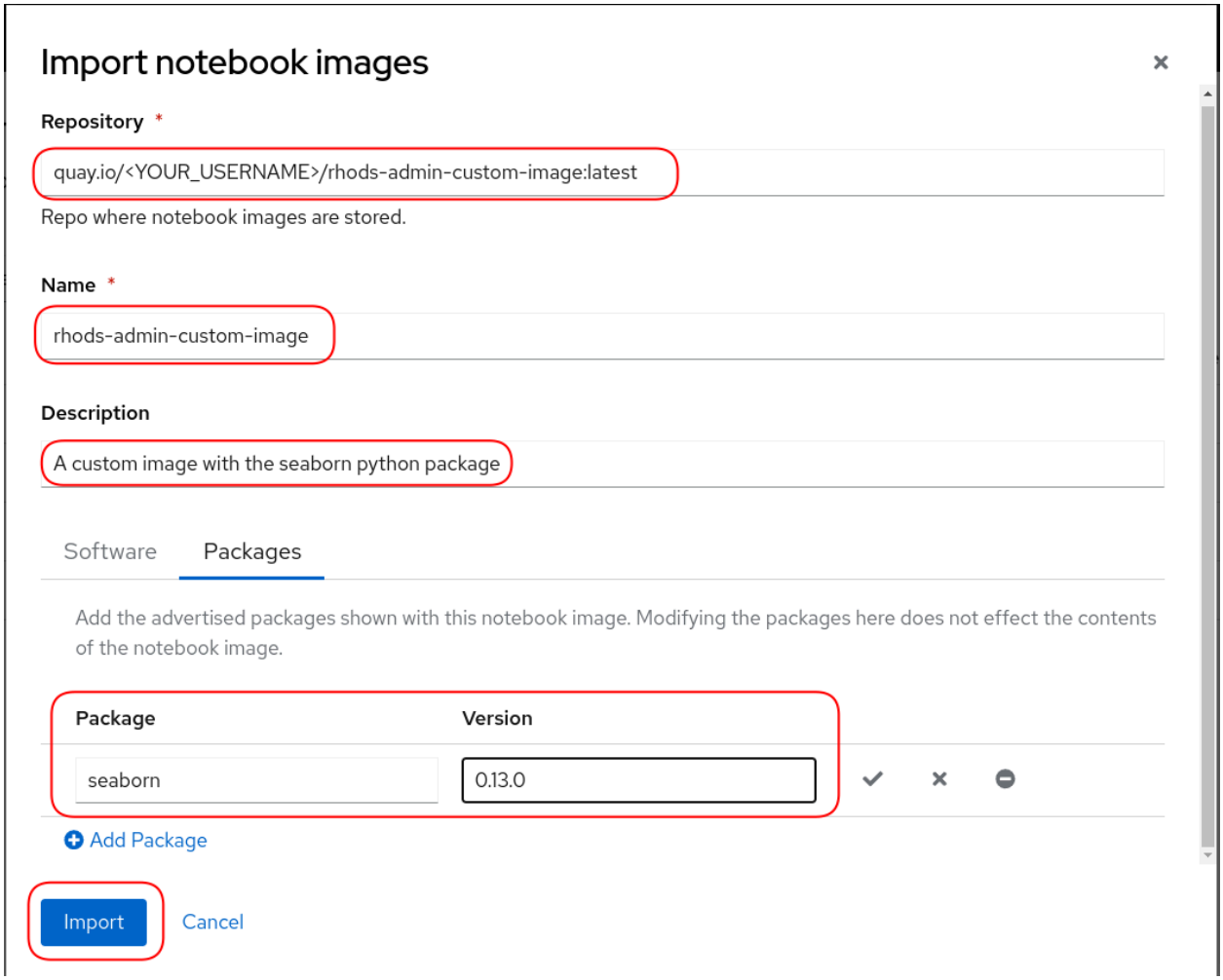
```
\
    -n redhat-ods-applications
```

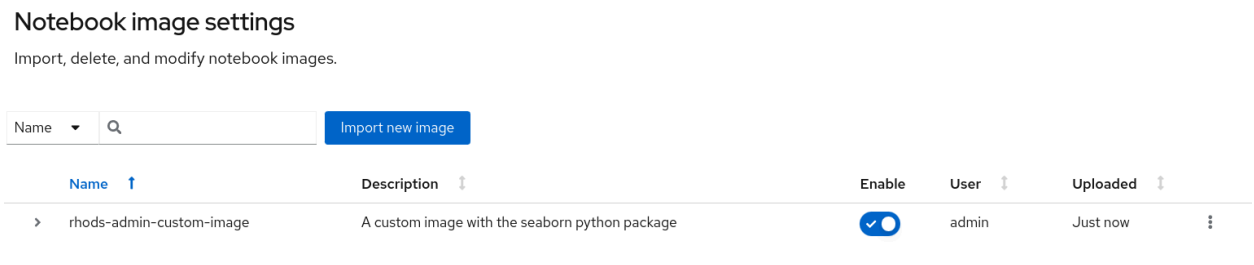
For more information, refer to Importing images and image streams from private registries.

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



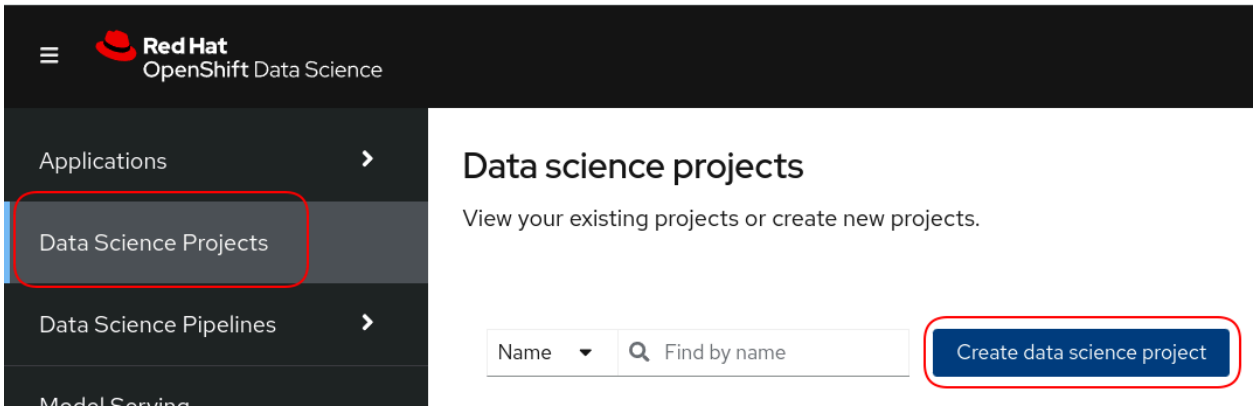
7.
8. Enter the path to the your repository and give the image a name. Optionally add a description and package information. Click the Import button.

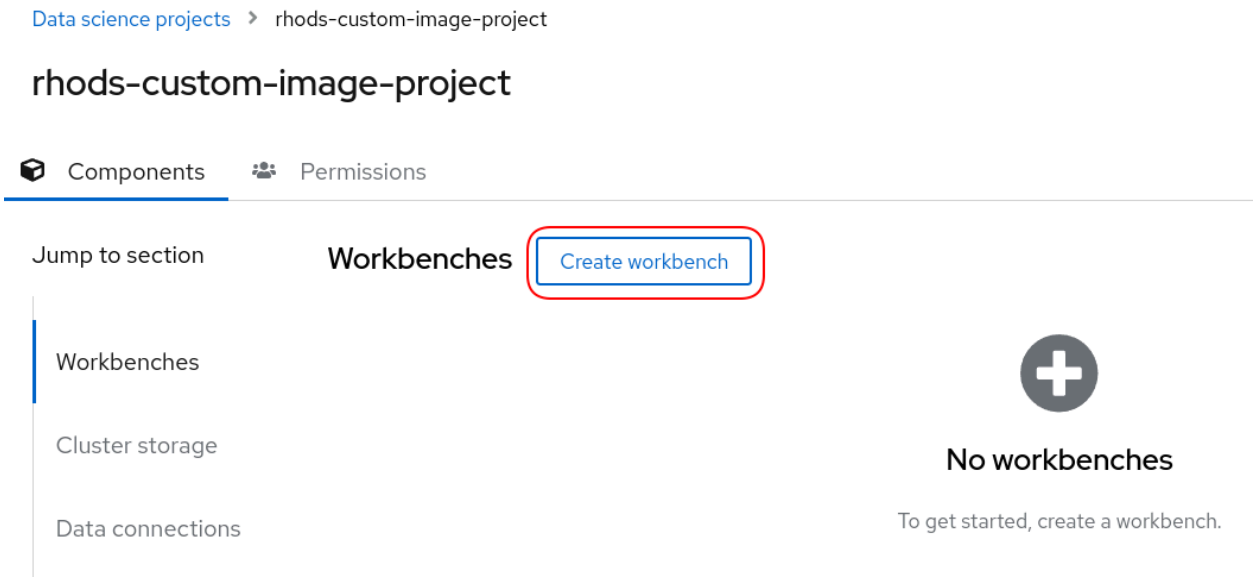
9. 

10. You should now see the image in the list and enabled.
11. 

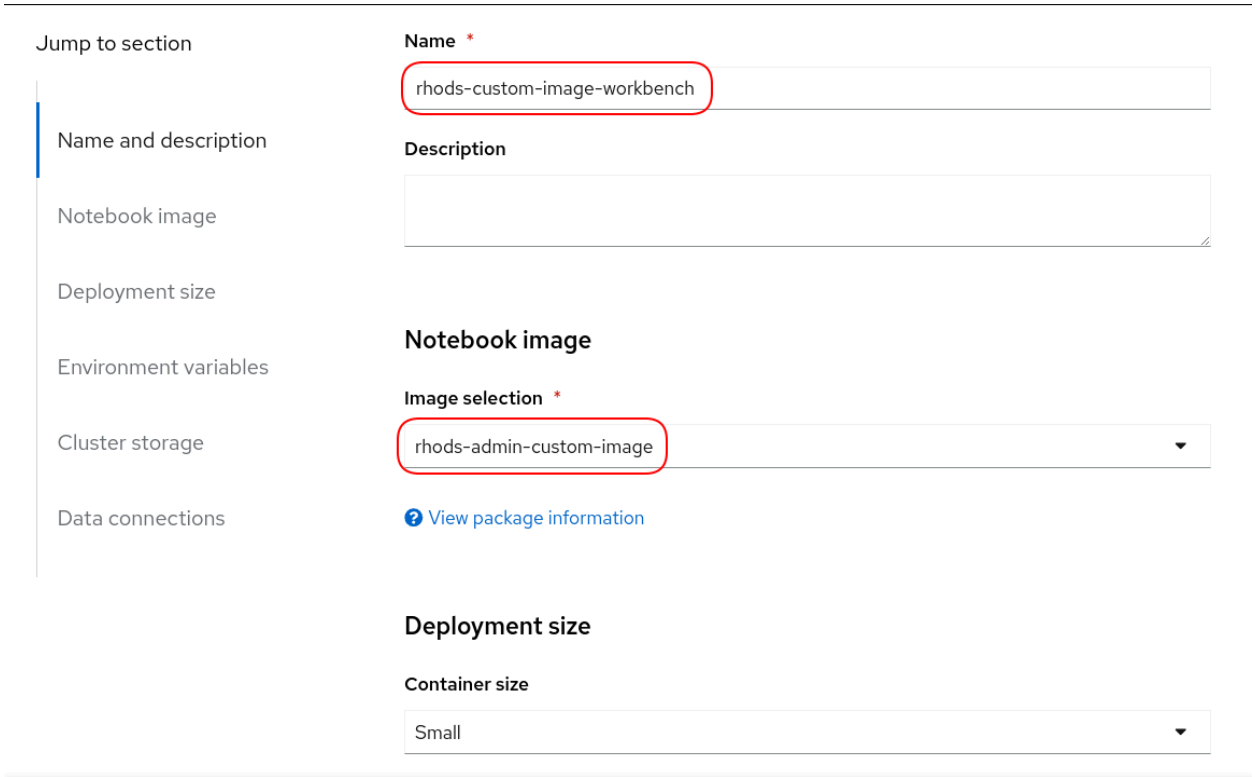
Test the Image in a Workbench

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.

2. 

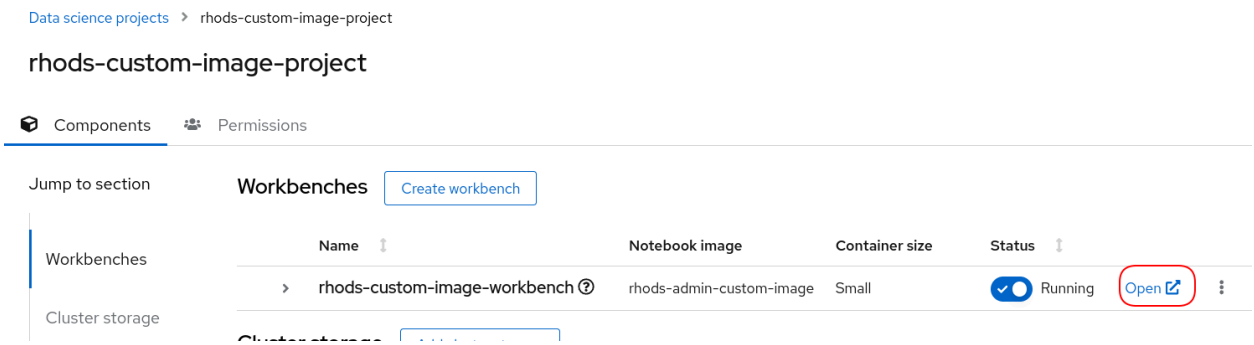
3. Give the project a name and resource name. Click the Create button.
4.
5. Click the Create button.
6. 

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.

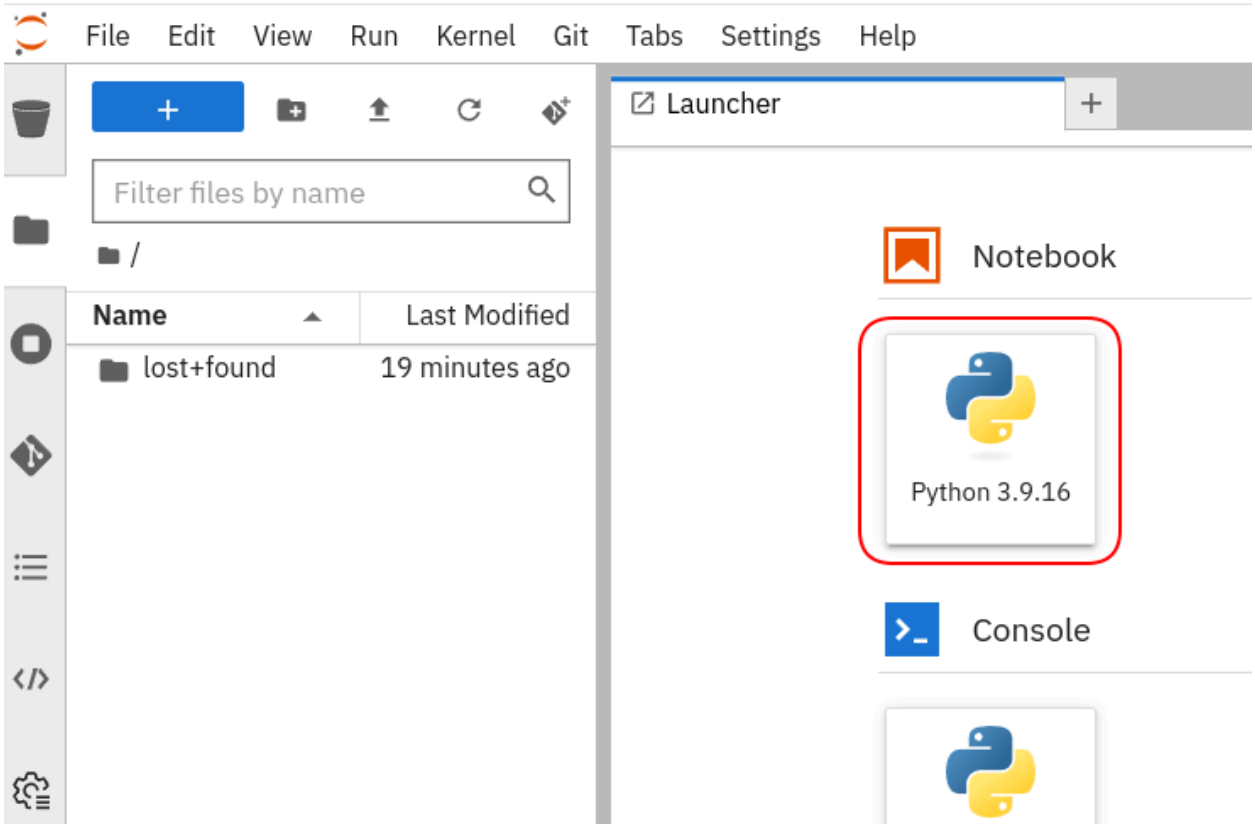


8. 

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

10. 

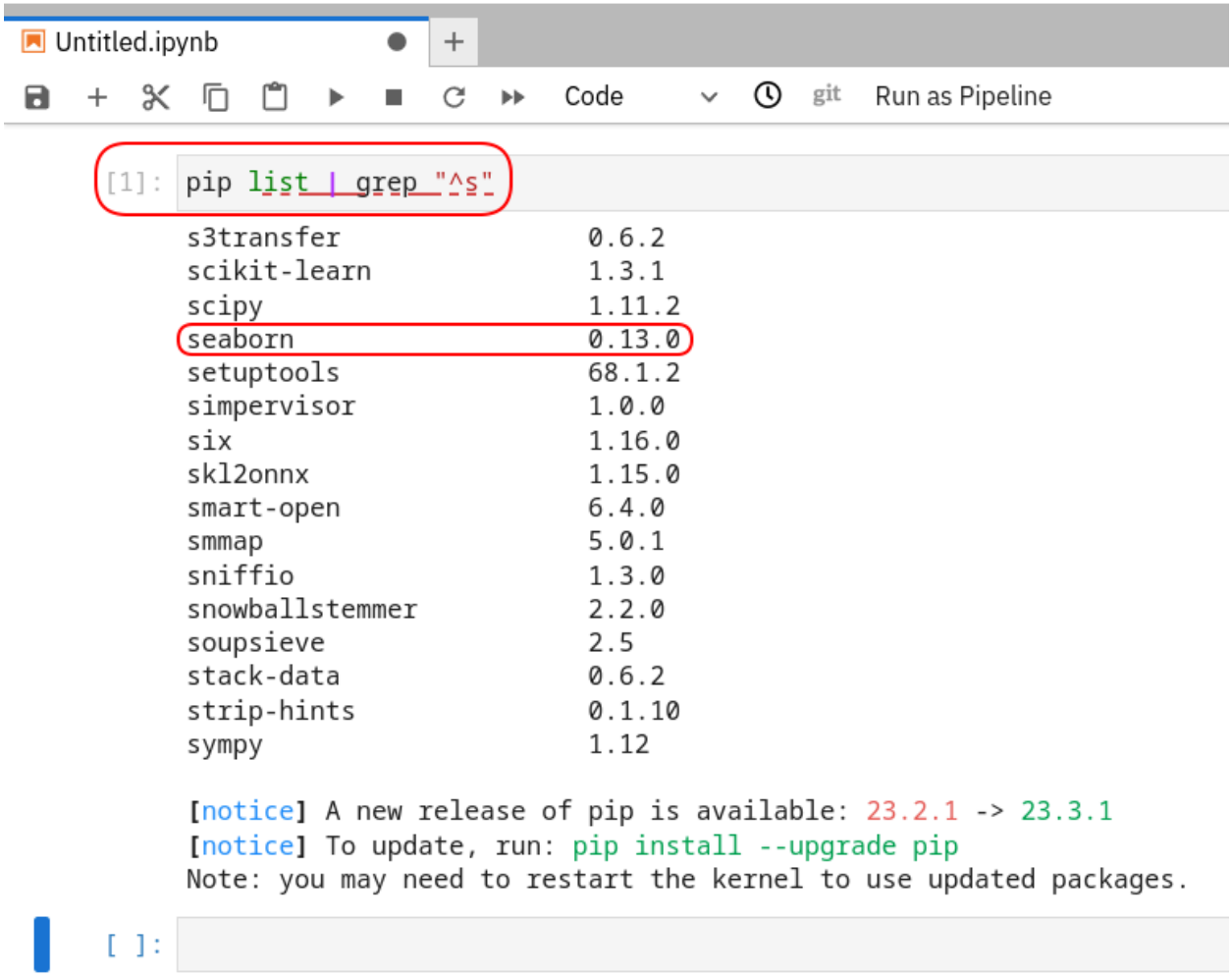
11. Login to the notebook and click the Python 3.9.16 notebook tile.

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.

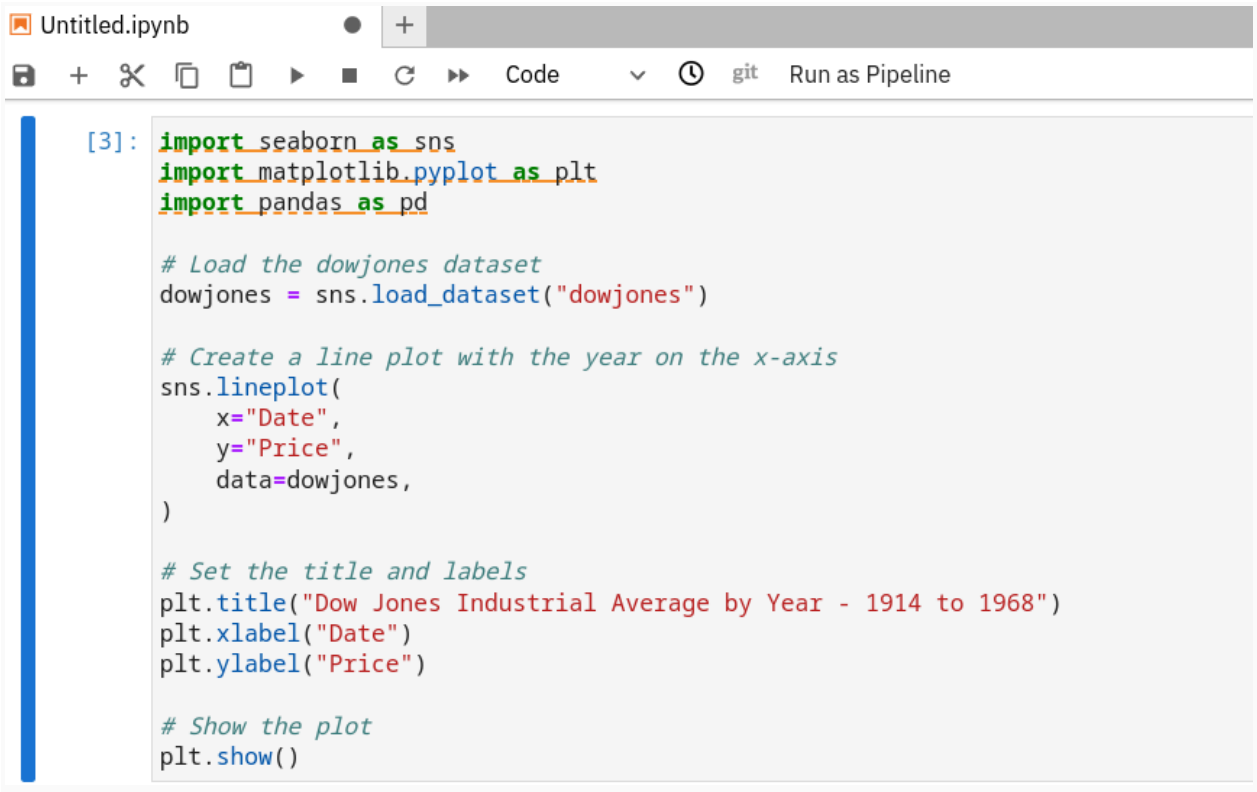


15.

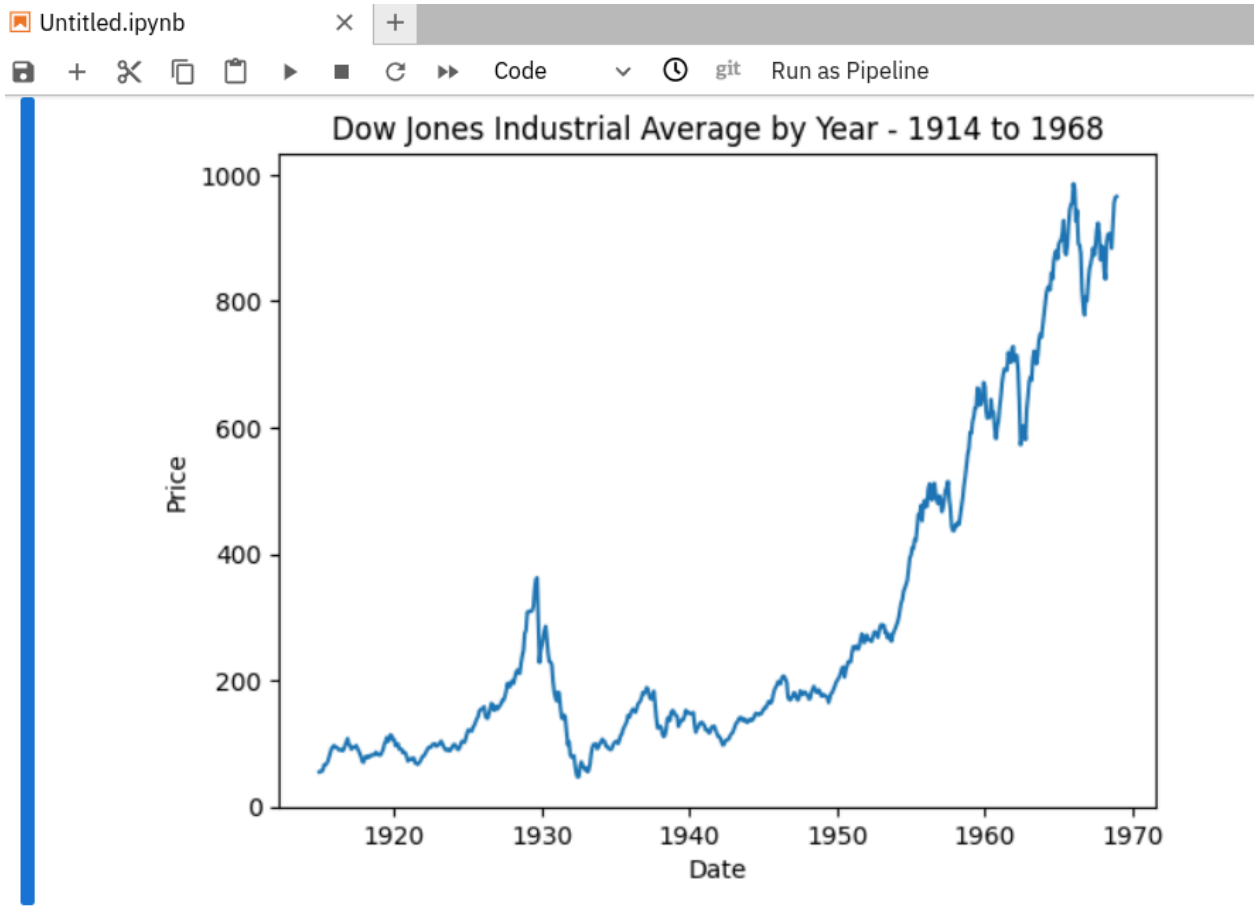
16. Now let's test out the seaborn package to make sure it works. Copy the following code to the cell below.

```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()
```

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:

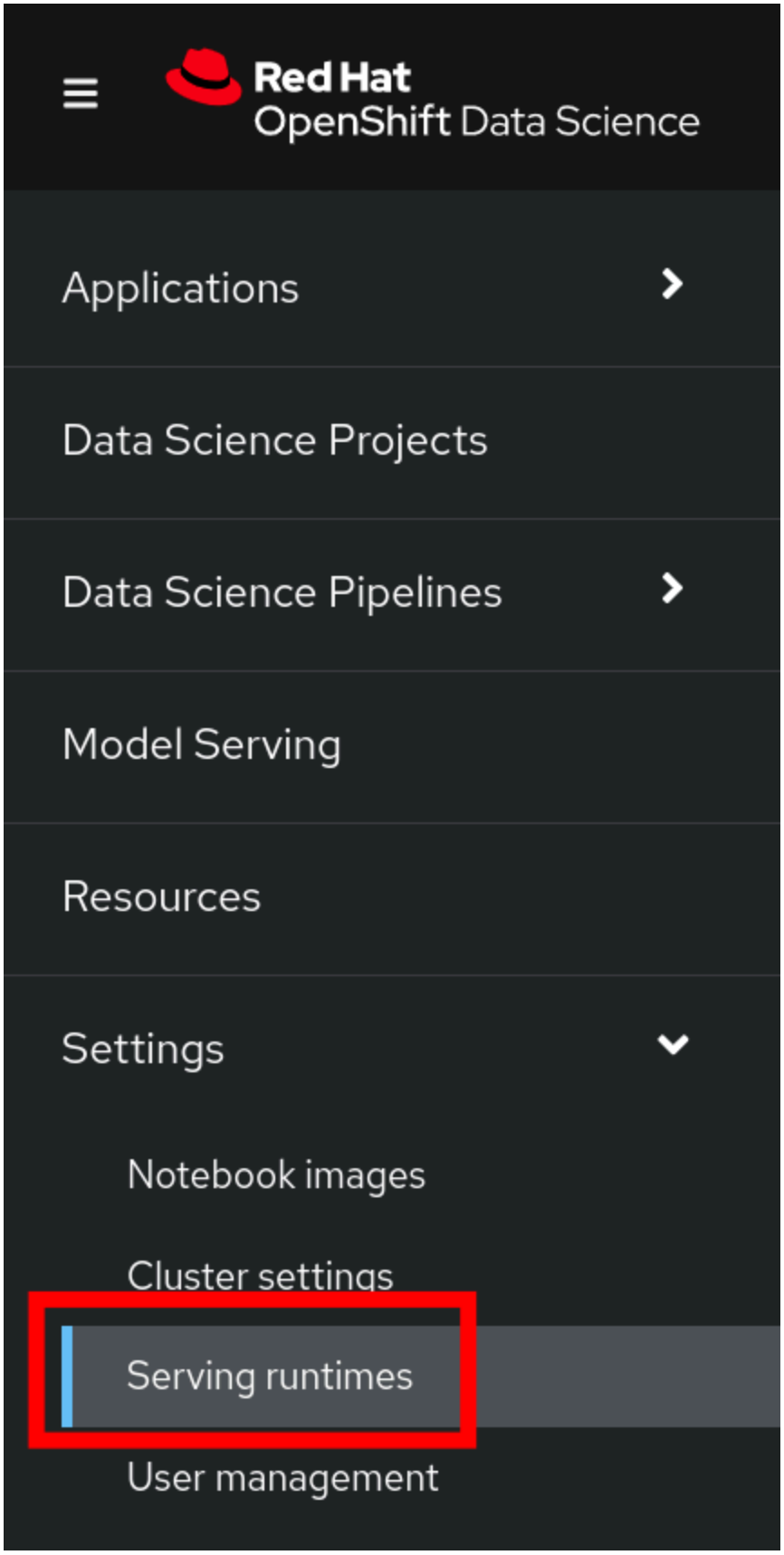
- An s3 bucket with a model in format onnx
- A Data Science project with the name iris-project
- A data connection to S3 with the name iris-data-connection

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).

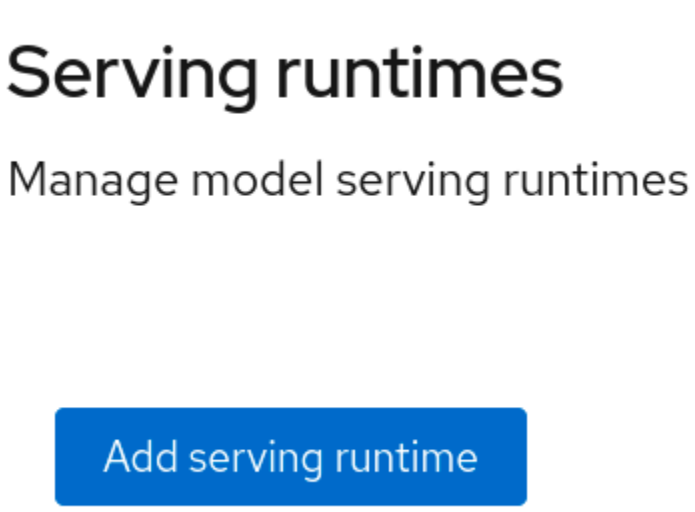
> While RHOAI supports the ability to add your own runtime, it does not support the runtimes themselves. Therefore, it is up to you to configure, adjust and maintain your custom runtimes.

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. 
Red Hat
OpenShift Data Science
Applications
Data Science Projects
Data Science Pipelines
Model Serving
Resources
Settings
Notebook images
Cluster settings
Serving runtimes
User management

4. Click on the Add Serving Runtime button:



5.
6. Click on Start from scratch and in the window that opens up, paste the following YAML:

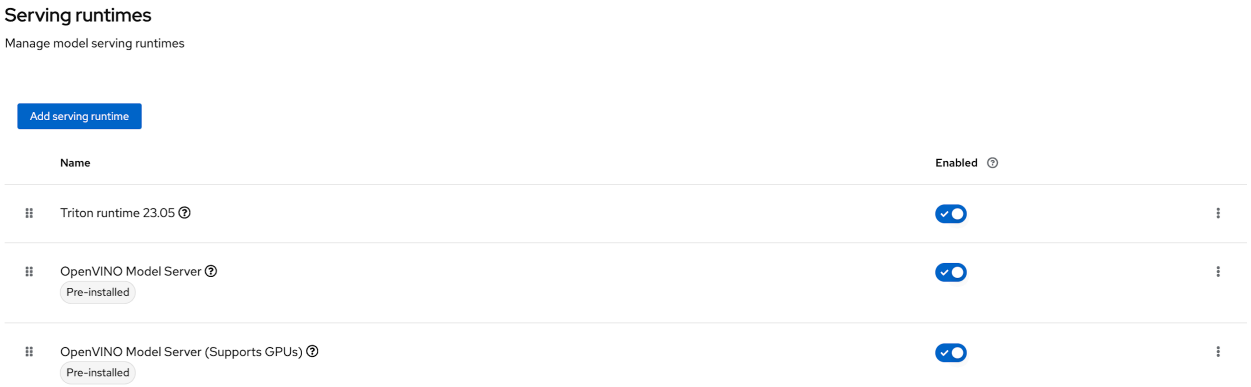
```
Unset
apiVersion: serving.kserve.io/v1alpha1
kind: ServingRuntime
metadata:
 name: triton-23.05-20230804
 labels:
  name: triton-23.05-20230804
 annotations:
  maxLoadingConcurrency: "2"
  openshift.io/display-name: "Triton runtime 23.05"
spec:
 supportedModelFormats:
  - name: keras
    version: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

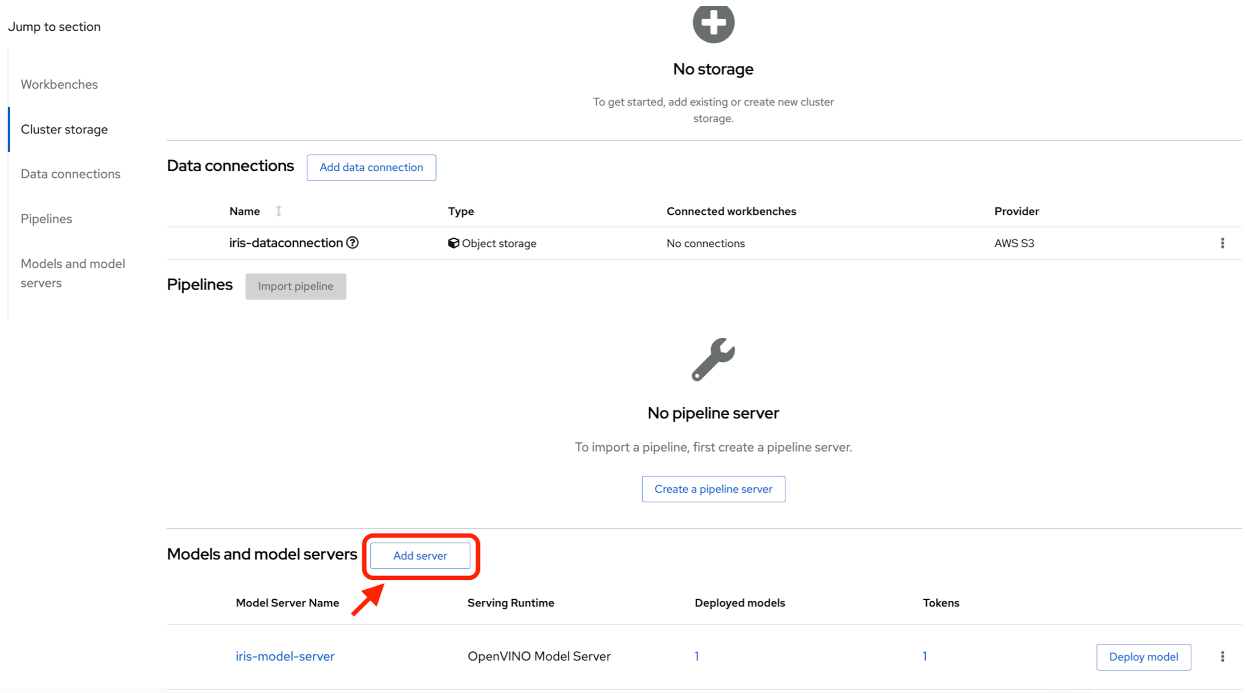
7. After clicking the Add button at the bottom of the input area, we are able to see the new Runtime in the list. We can re-order the list as needed (the order chosen here is the order in which the users will see these choices)



8.

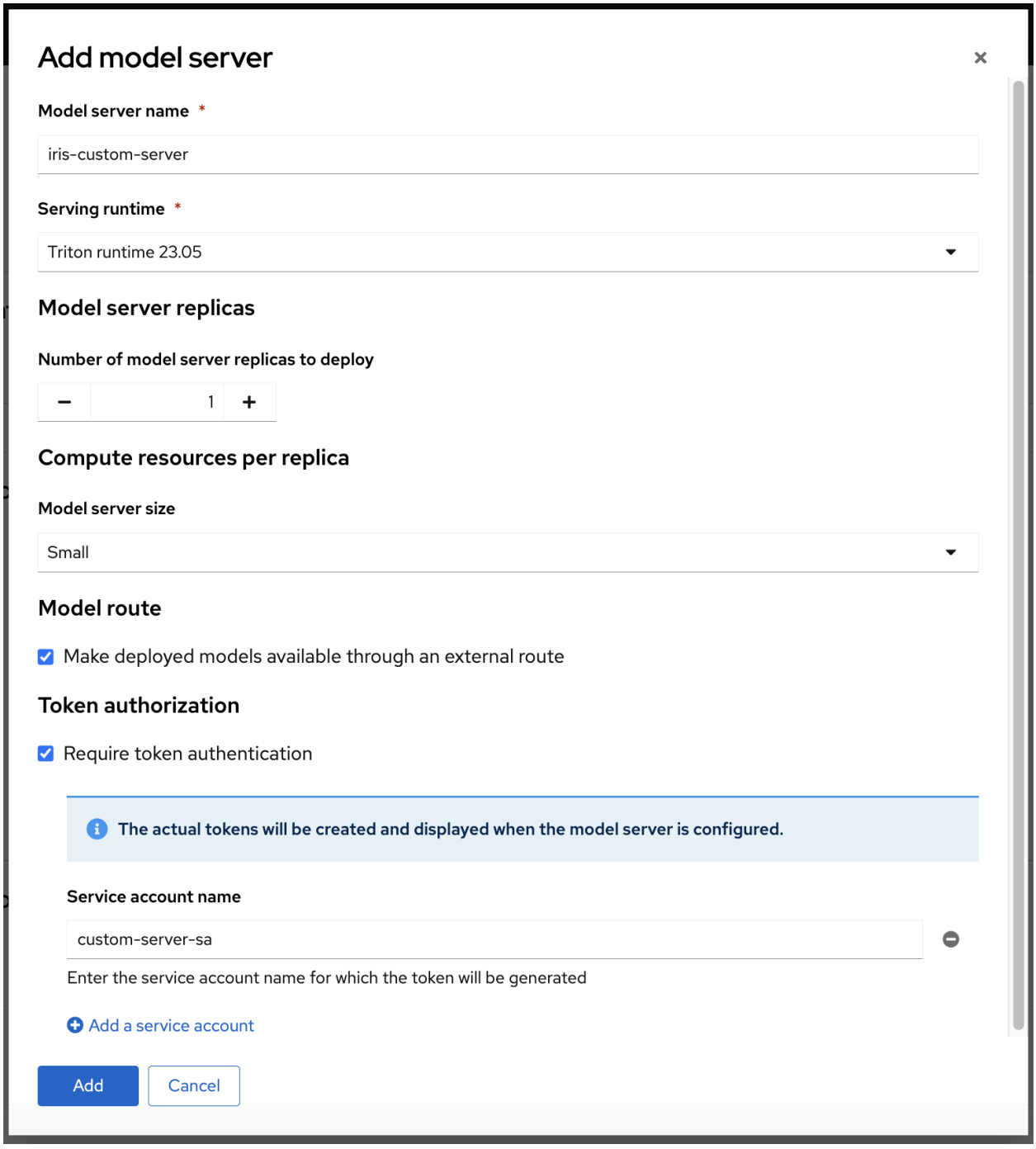
Creating The Model Server

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

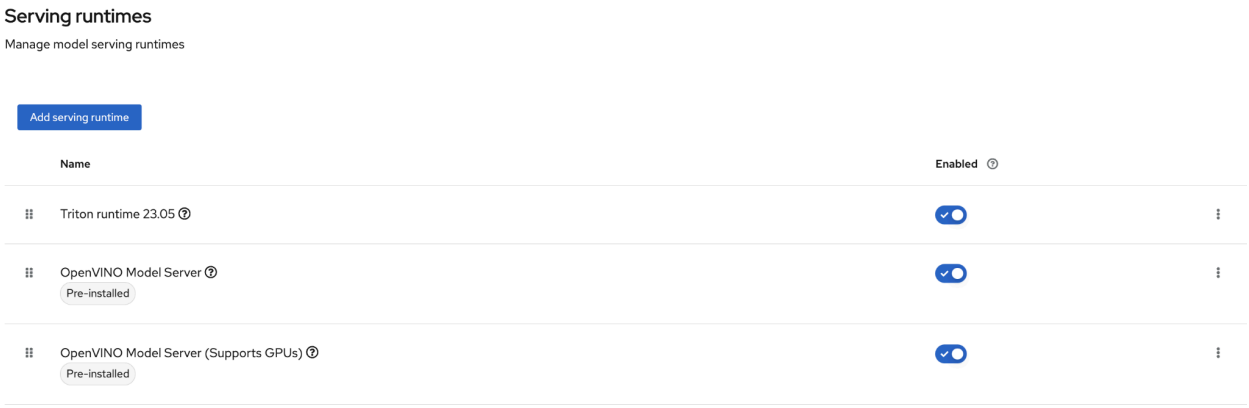


2.

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.

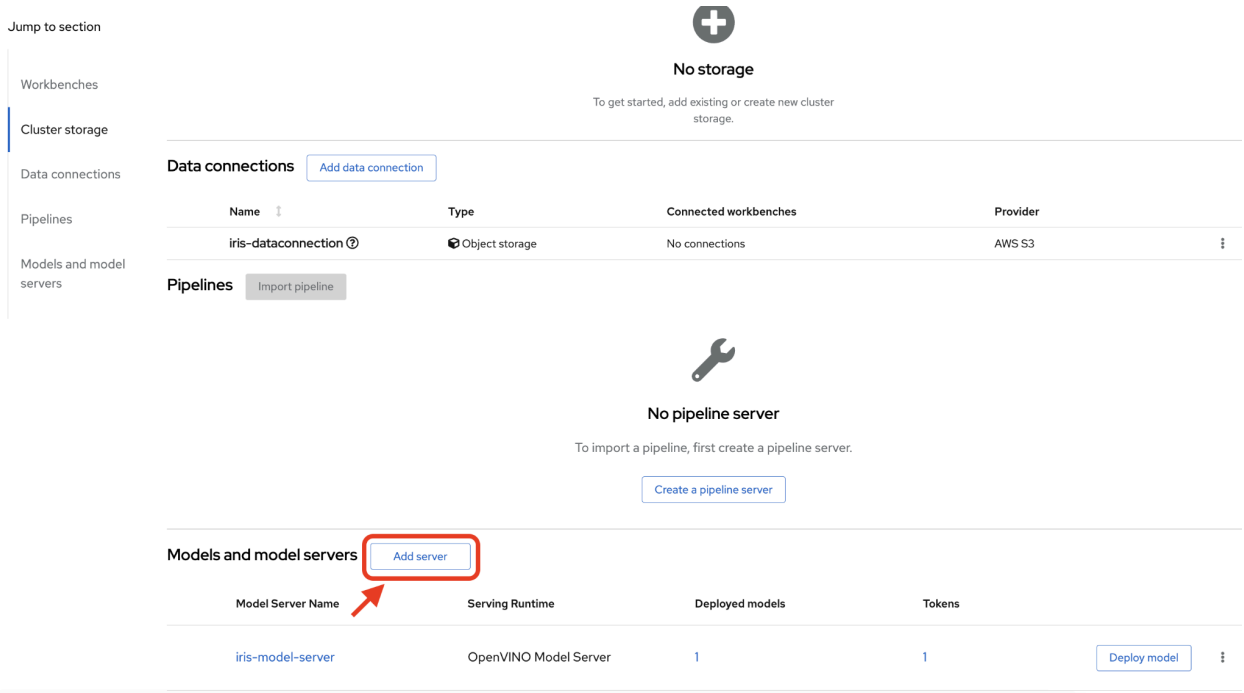
4. 

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.

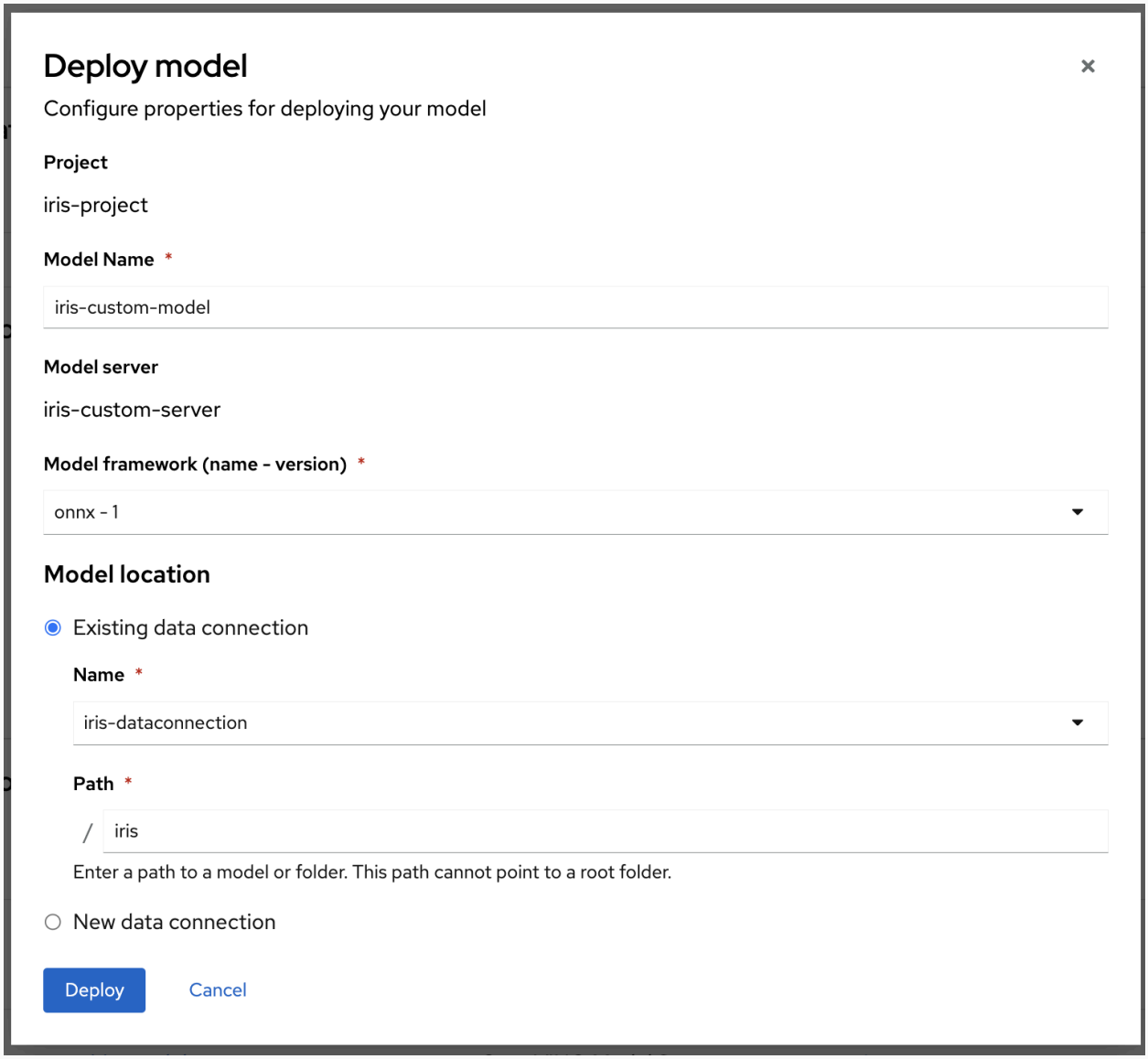
6. 

Deploy The Model

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

2. 

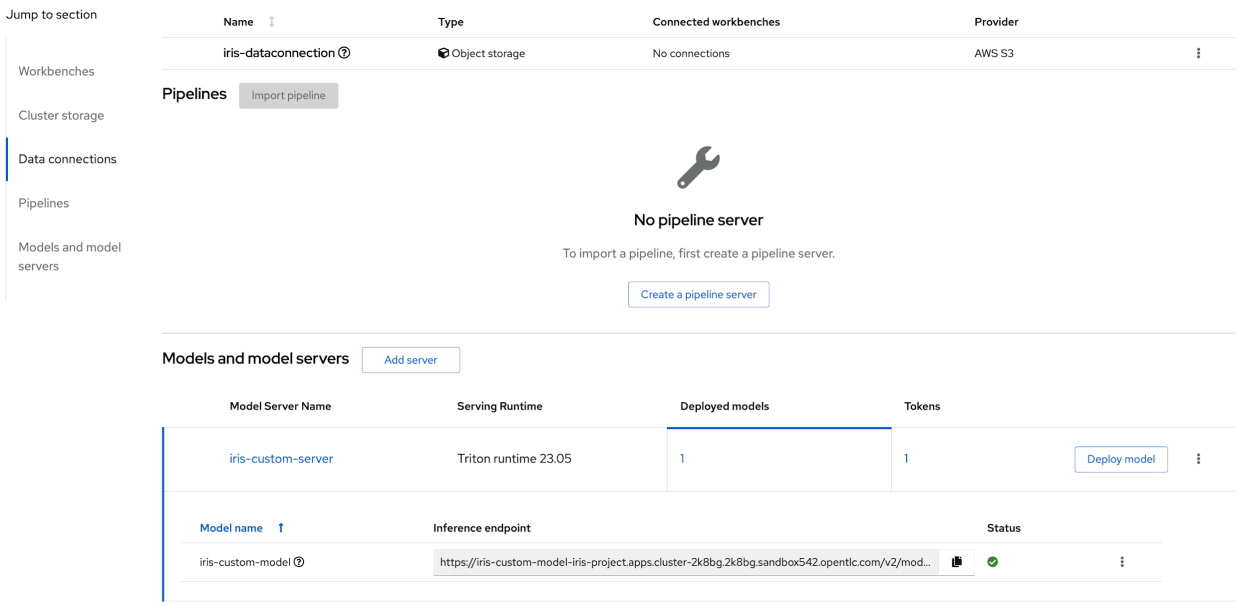
3. Fill up the Deploy Model form:
    - Model name: `iris-custom-model`
    - Model framework: `onnx - 1`
    - Model location data connection: `iris-data-connection`
    - Model location path: `iris`

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.

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=https://$(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],"da
ta":[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)

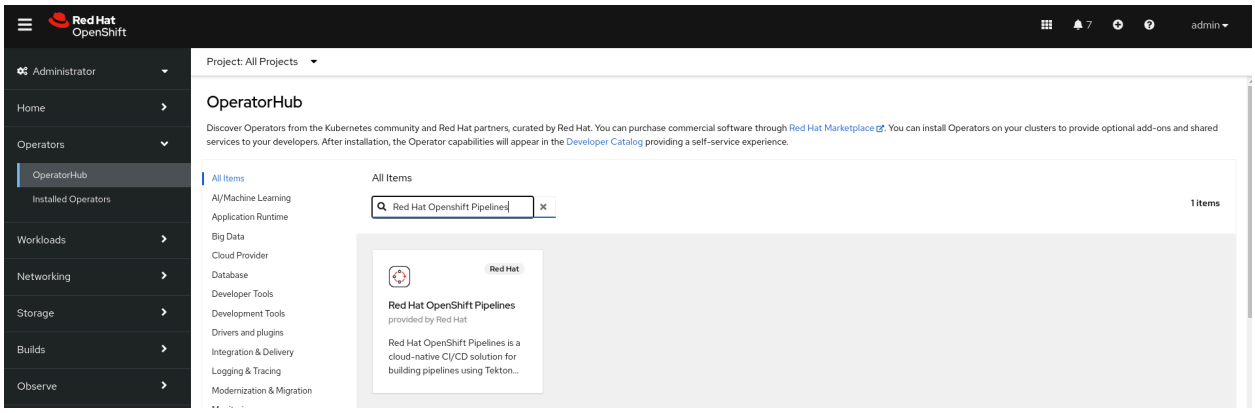
Installing Dependencies Using the Web Console

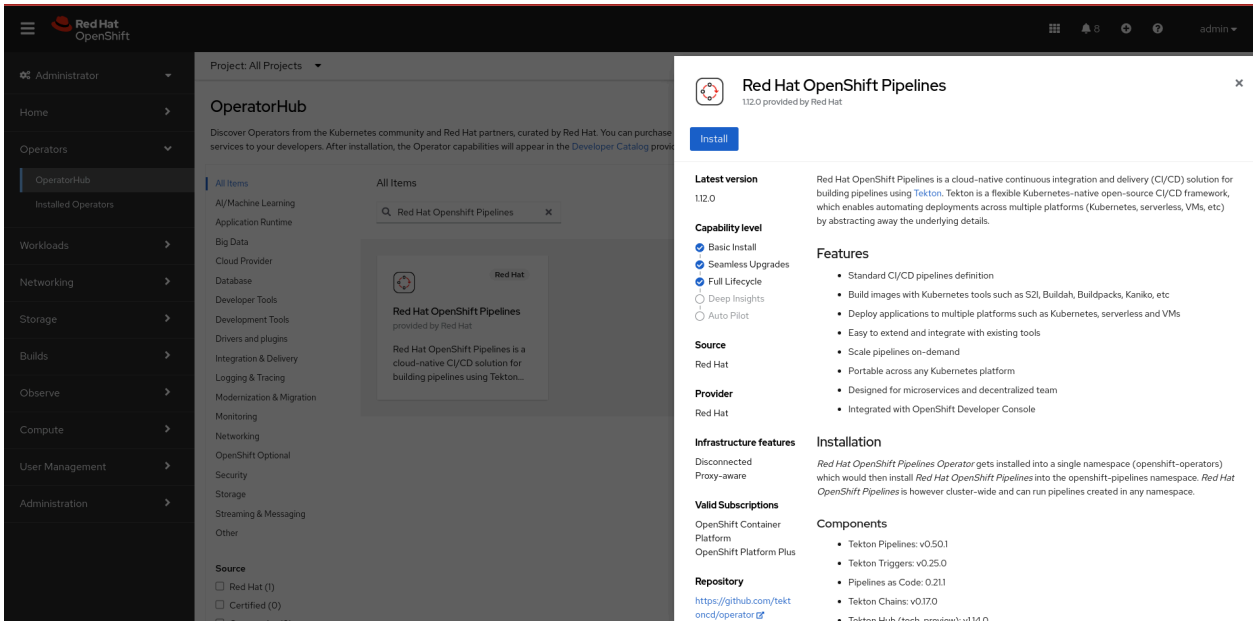
As described in the General Information about Installation section you may need to install other operators depending on the components and features of OpenShift AI you want to use. This section will discuss installing and configuring those components.
It is generally recommended to install any dependent 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

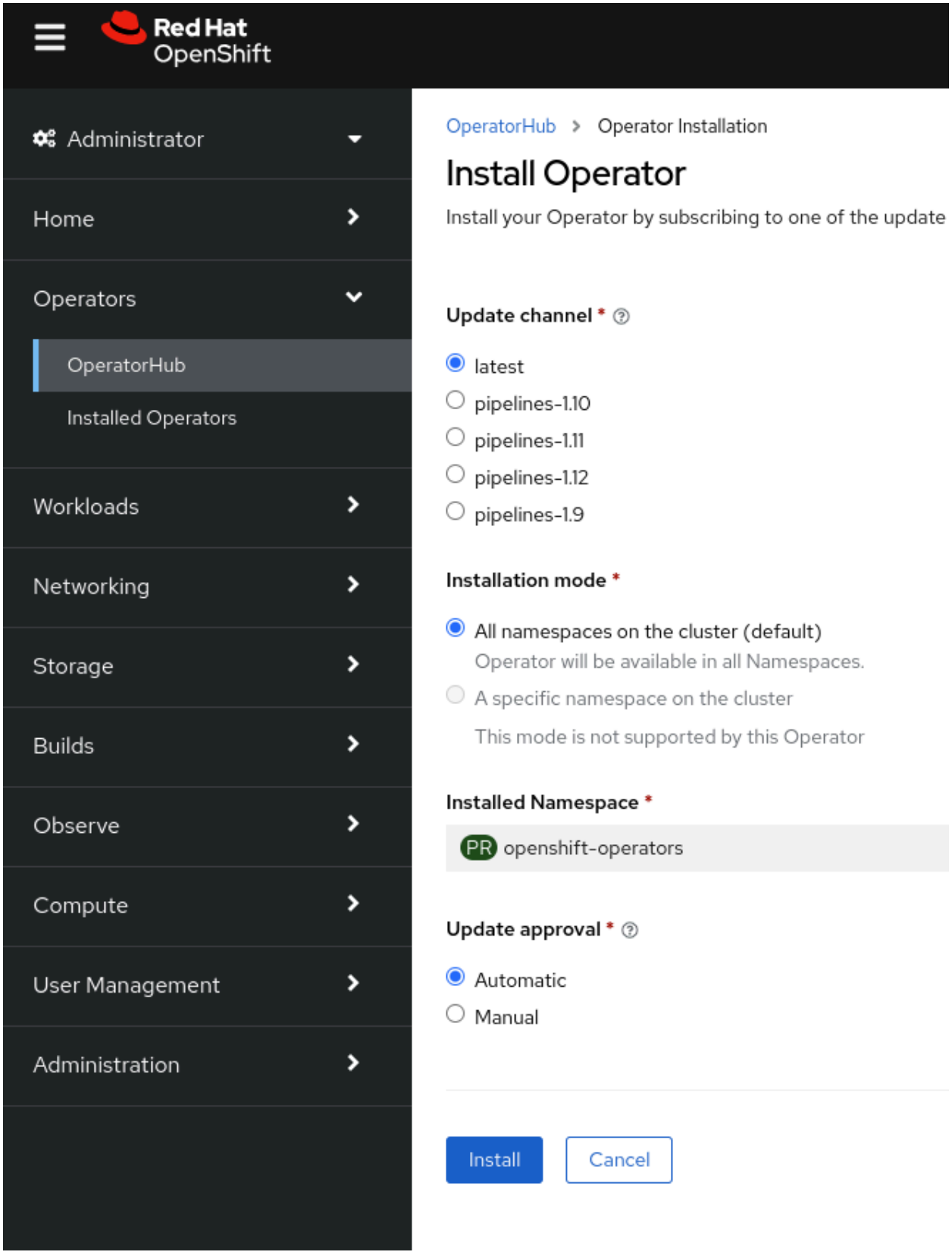
The Data Science Pipelines component utilizes Red Hat OpenShift Pipelines as an execution engine for all pipeline runs, and is required to be installed to take advantage of the Data Science Pipelines component.
The following section discusses installing the Red Hat OpenShift Pipelines operator.

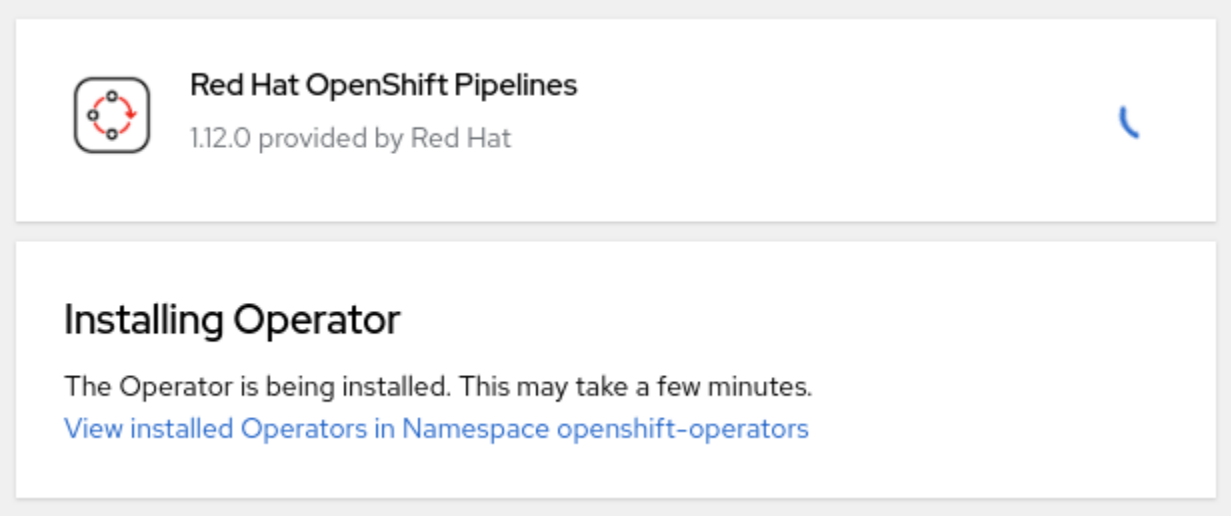
Demo: Installation of the Red Hat OpenShift Pipelines operator

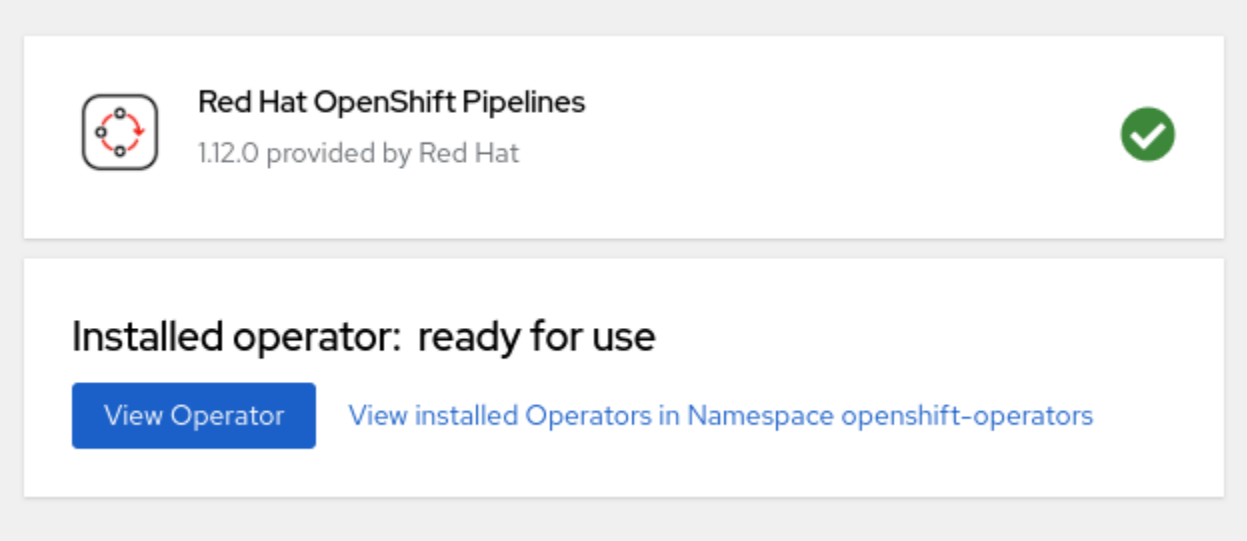
1. Login to Red Hat OpenShift using a user which has the *cluster-admin* role assigned.
2. Navigate to Operators → OperatorHub and search for Red Hat OpenShift Pipelines
3. 

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.
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.

7. 
Red Hat OpenShift
Administrator
Home
Operators
OperatorHub
Installed Operators
Workloads
Networking
Storage
Builds
Observe
Compute
User Management
Administration
OperatorHub > Operator Installation
Install Operator
Install your Operator by subscribing to one of the update
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

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.
9. 

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

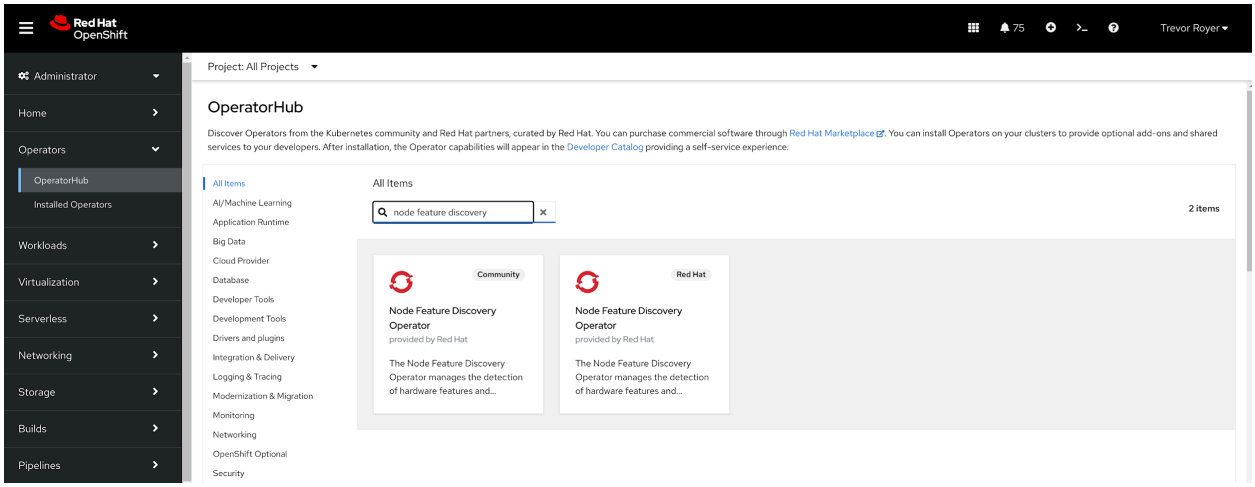
Installation of GPU Dependencies

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.

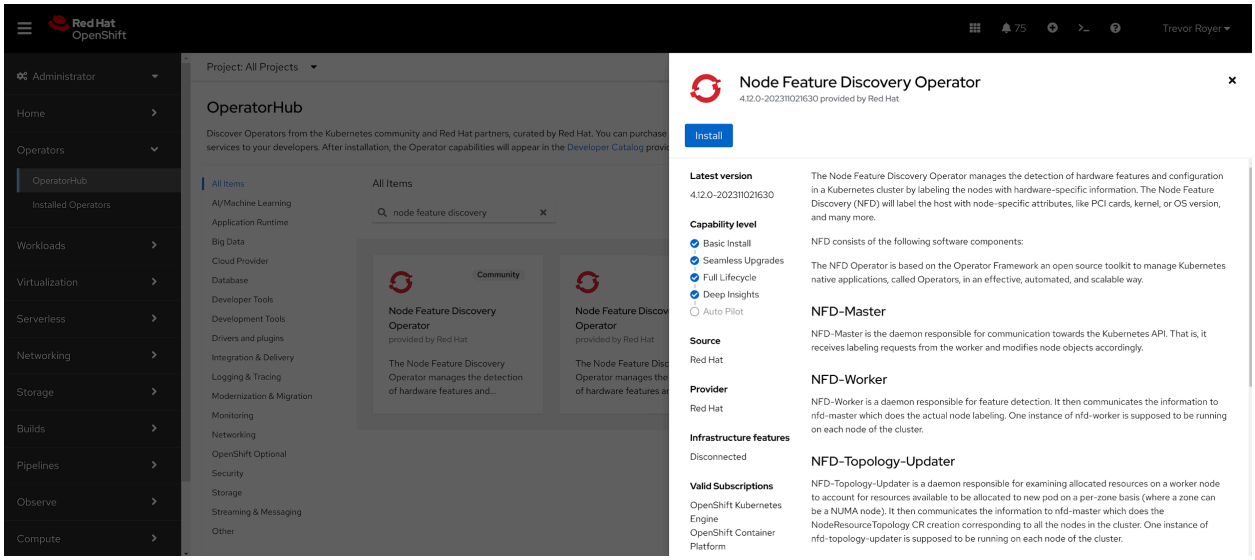
> To view the list of GPU models supported by the NVIDIA GPU Operator refer to the Supported NVIDIA GPUs and Systems docs.

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
3. 

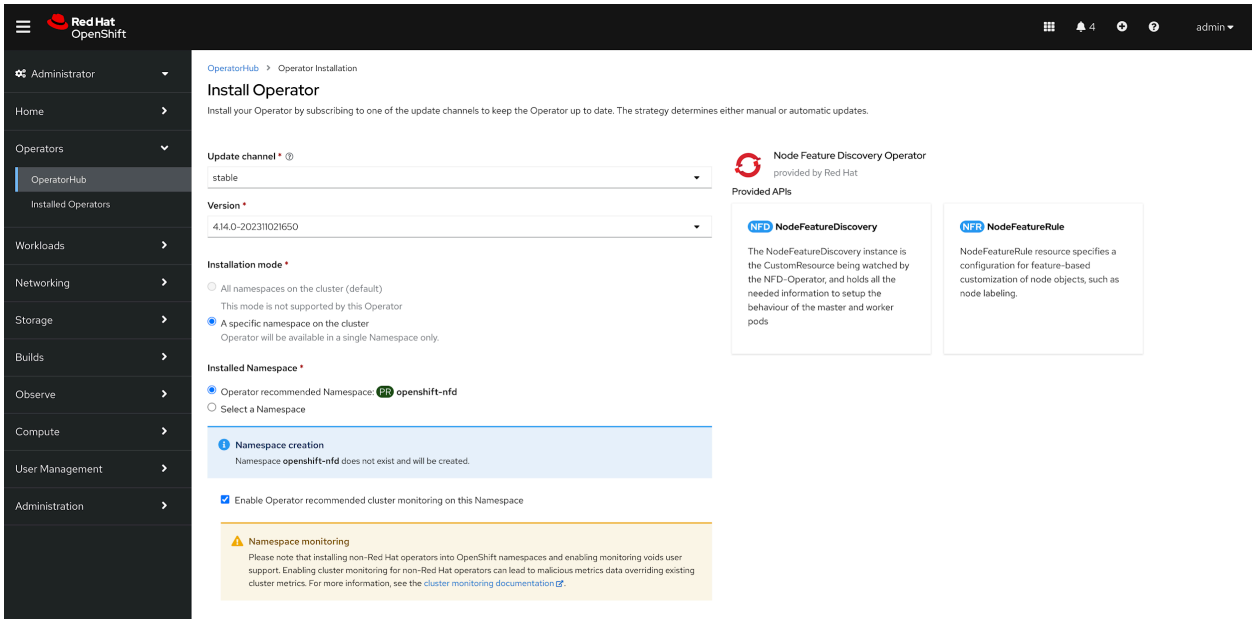
4. Two options for the Node Feature Discovery operator will be available. Click on the one with Red Hat in the top right hand corner and in the pop up window click on Install to open the operator's installation view.

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

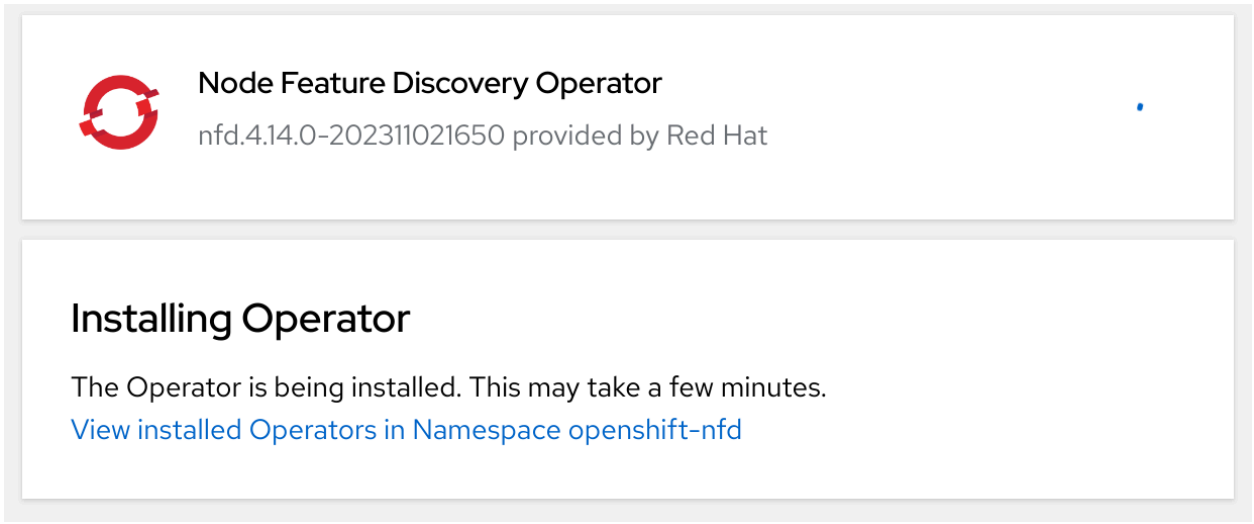
5. 

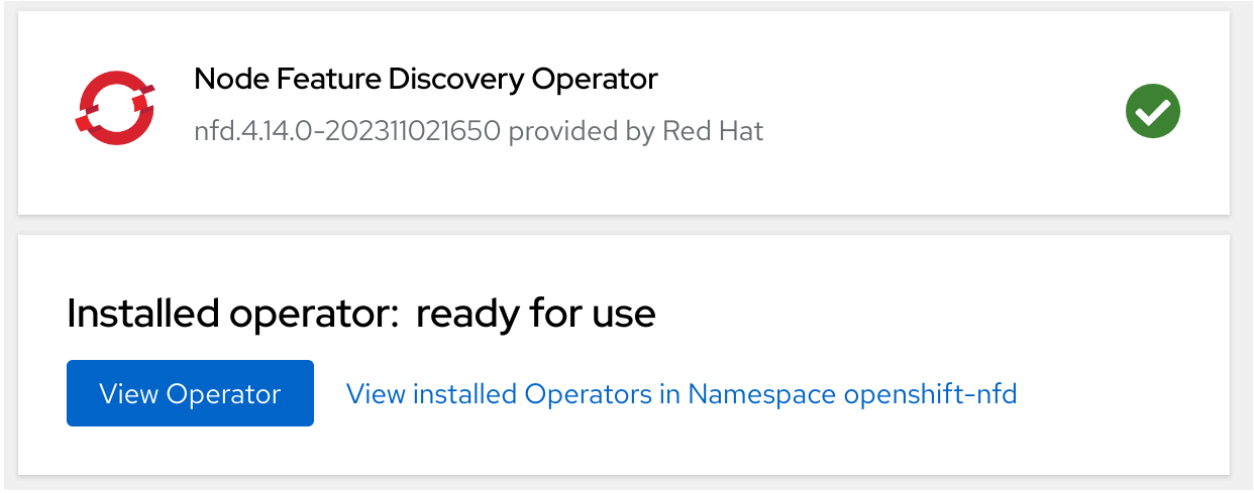
6. In the installation view check the box to Enable Operator recommended cluster monitoring on this Namespace and the Update approval parameters if desired. Leave the Update channel, Version, and the Installed Namespace parameters as the default options.

> 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.

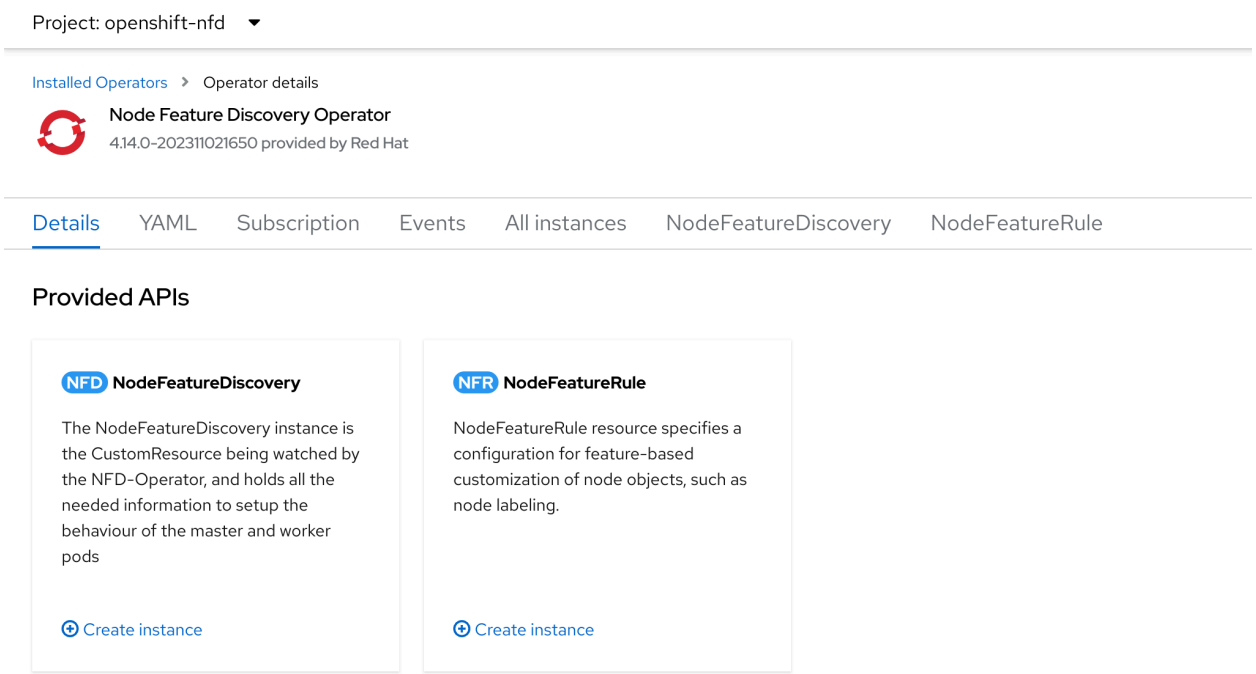
7. 

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.

9. 

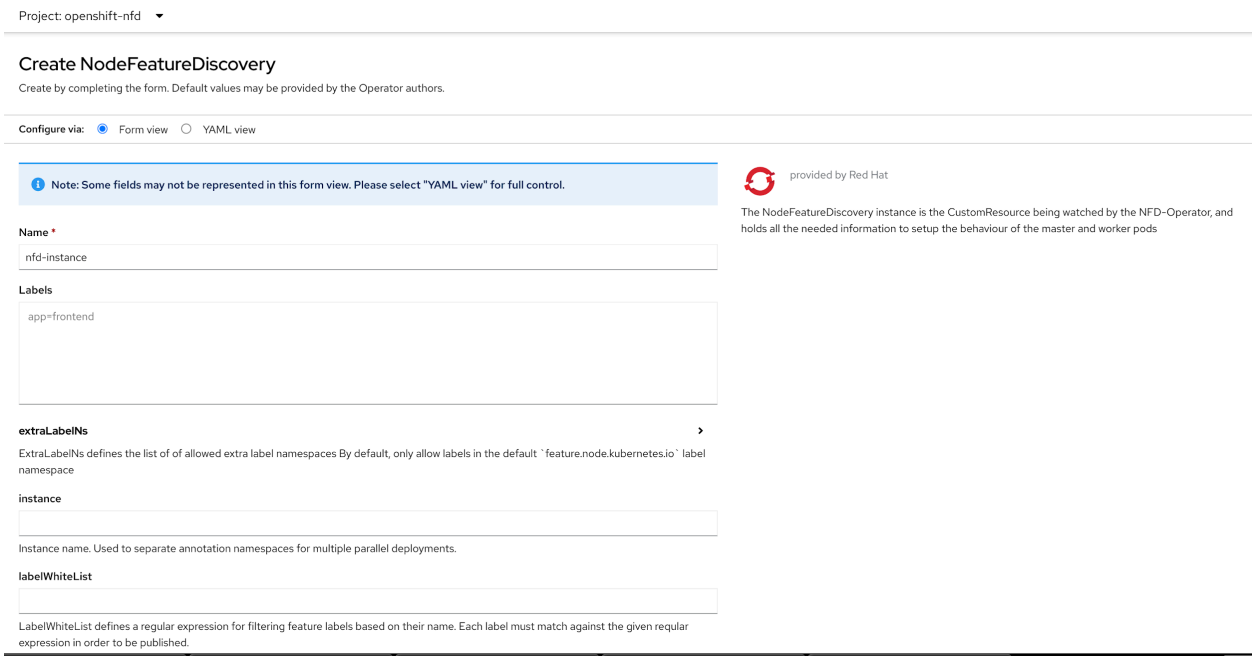
10. When the installation finishes the operator to be configured. Click the button to View Operator.
11. 

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



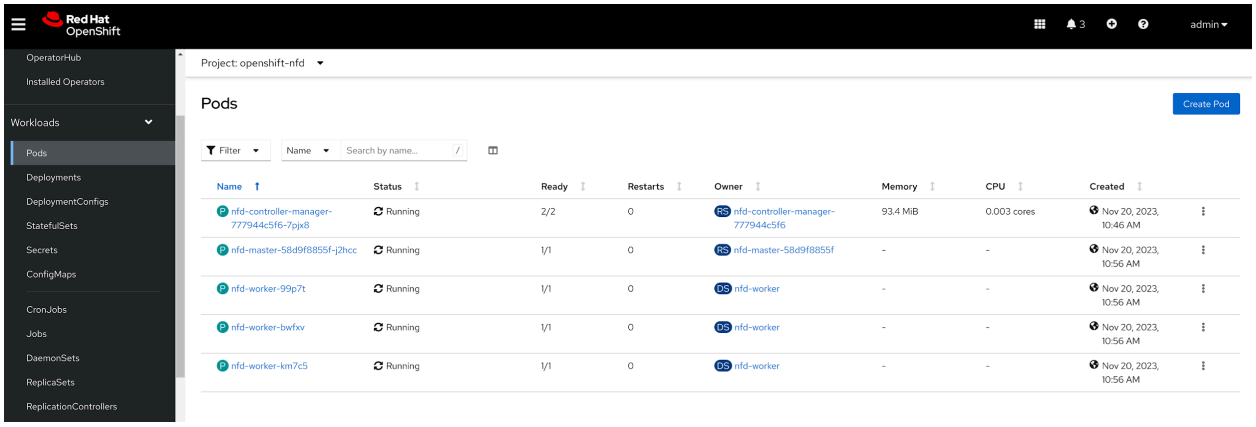
13.

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



15.

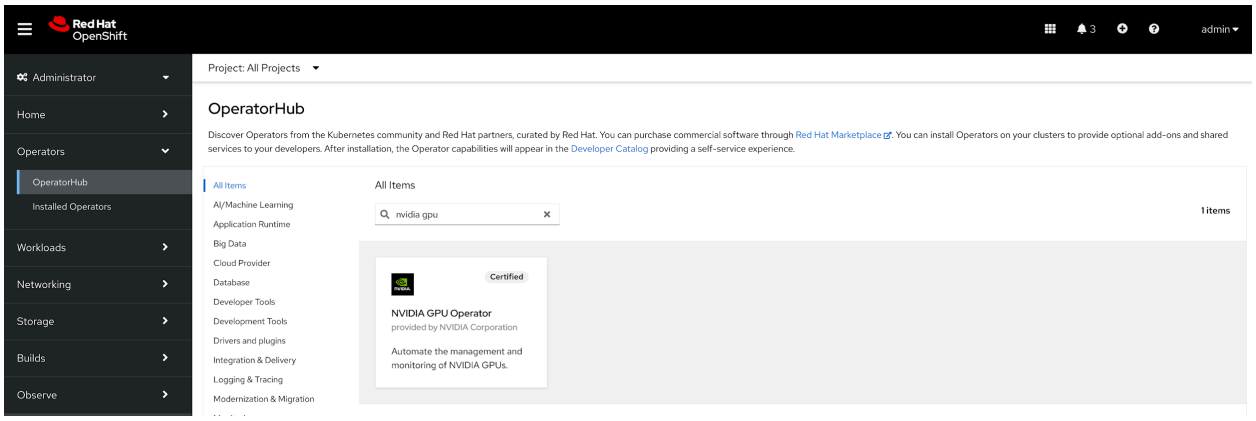
16. A new set of pods should appear in the Workloads → Pods section managed by the nfd-worker DaemonSet. Node Feature Discovery will now be able to automatically detect information about the nodes in the cluster and apply labels to those nodes.

17. 

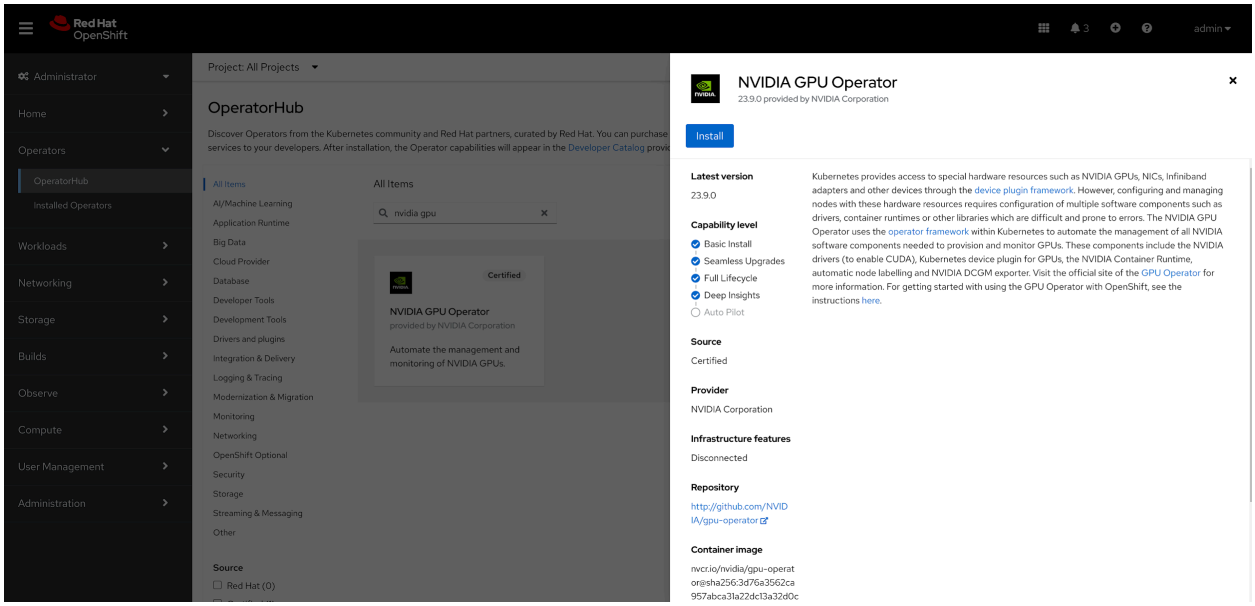
> For assistance installing the Node Feature Discovery Operator from YAML or via ArgoCD, refer to examples found in the [redhat-cop/gitops-catalog/nfd](https://github.com/redhat-cop/gitops-catalog/tree/main/nfd) GitHub repo.

Node Feature Discovery is now successfully installed and configured.

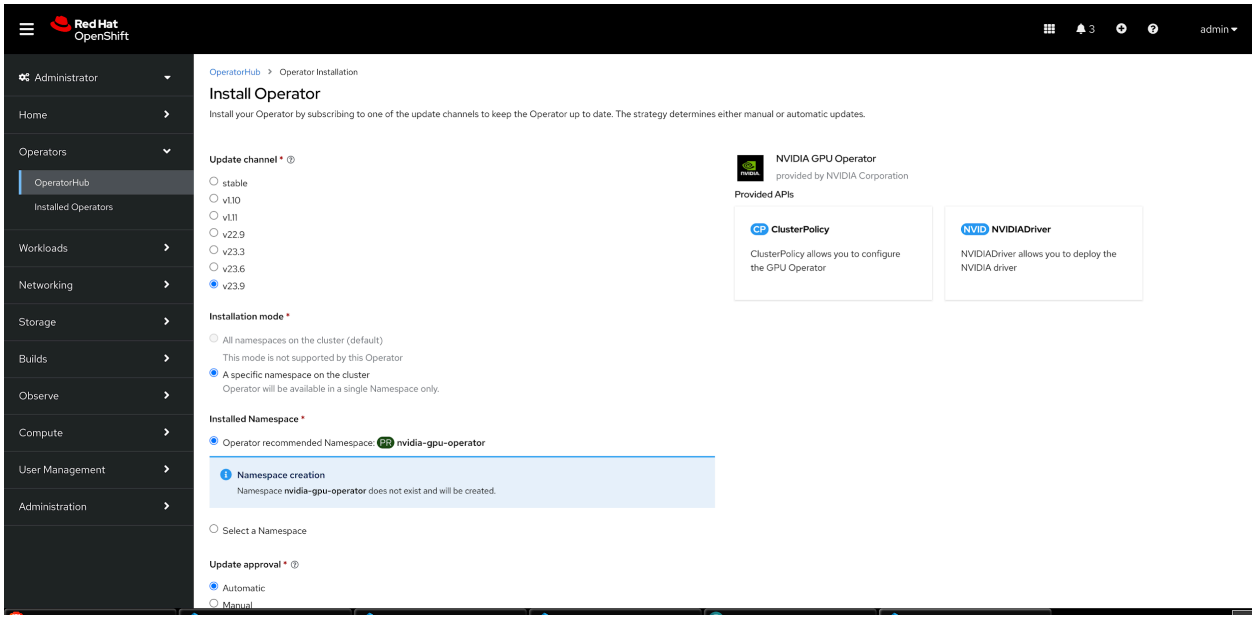
Demo: Installation of the NVIDIA GPU Operator

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. 

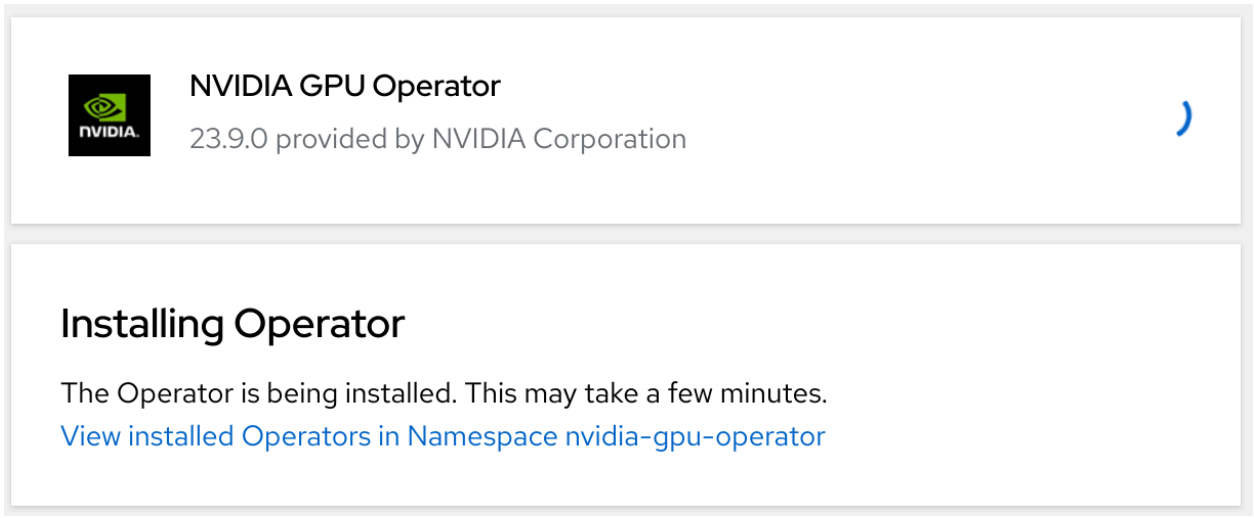
4. Click the NVIDIA GPU Operator tile and in the pop up window click on Install to open the operator's installation view.

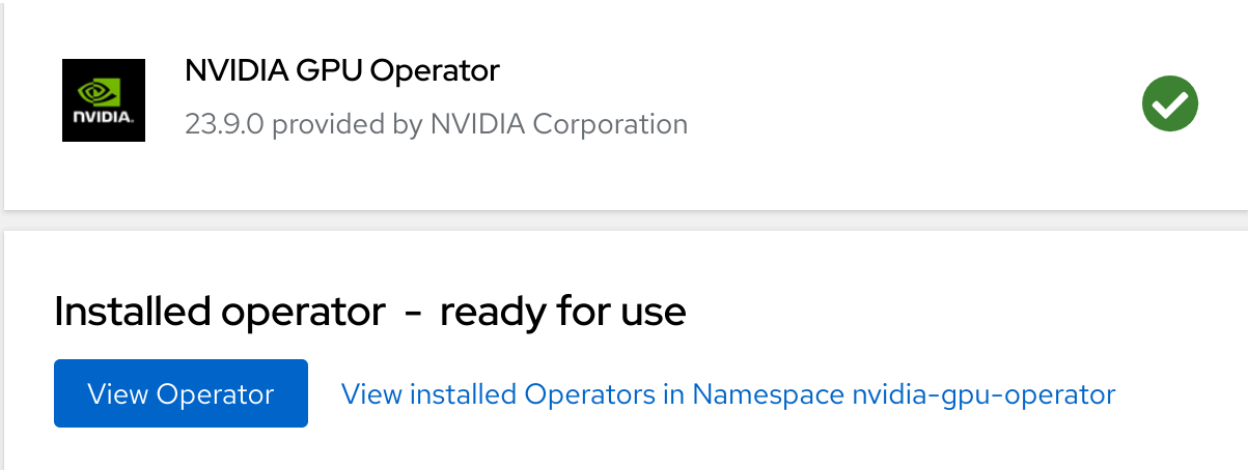
5. 

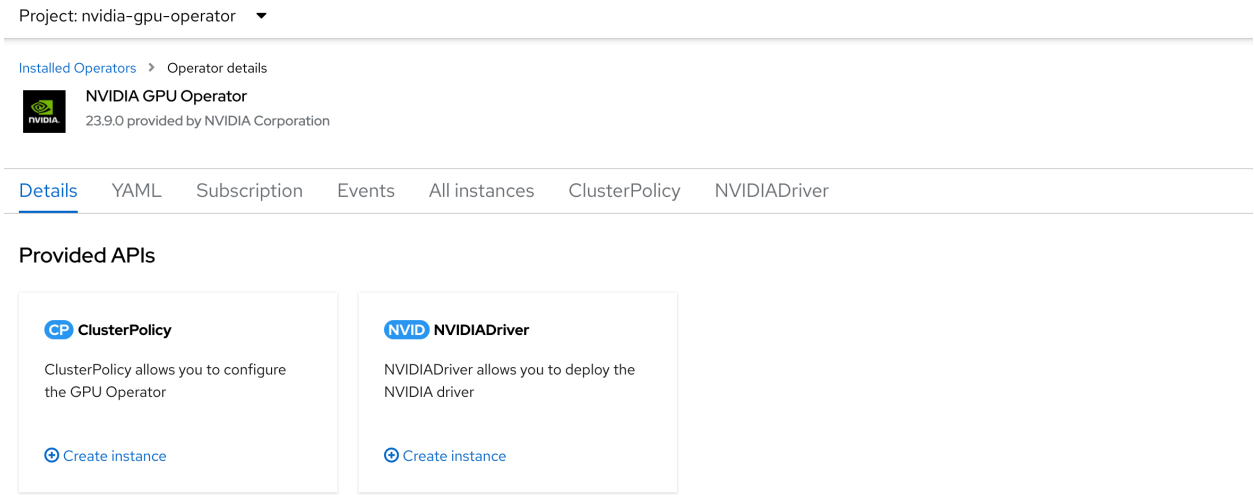
6. In the installation view update the Update channel and Update approval parameters if desired. Leave the Installation mode and the Installed namespace parameters as the default options.

7. 

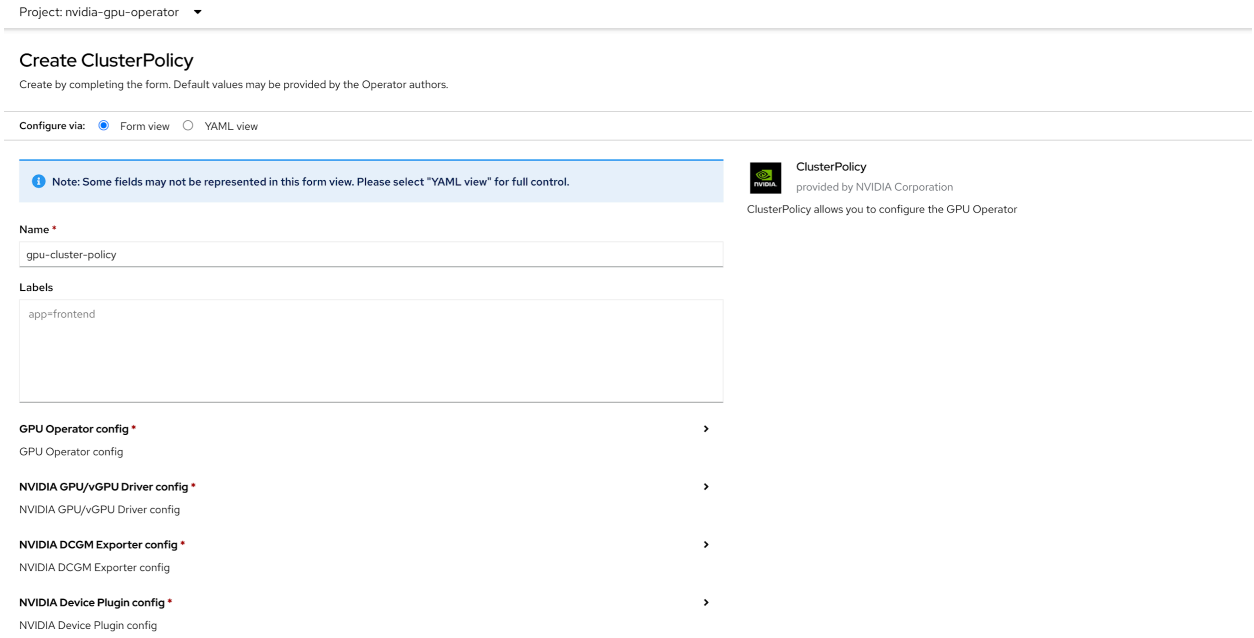
8. Click on the Install button at the bottom of the page to proceed with the installation. A window showing the installation progress will pop up.

9. 

10. When the installation finishes the operator to be configured. Click the button to View Operator.
11. 

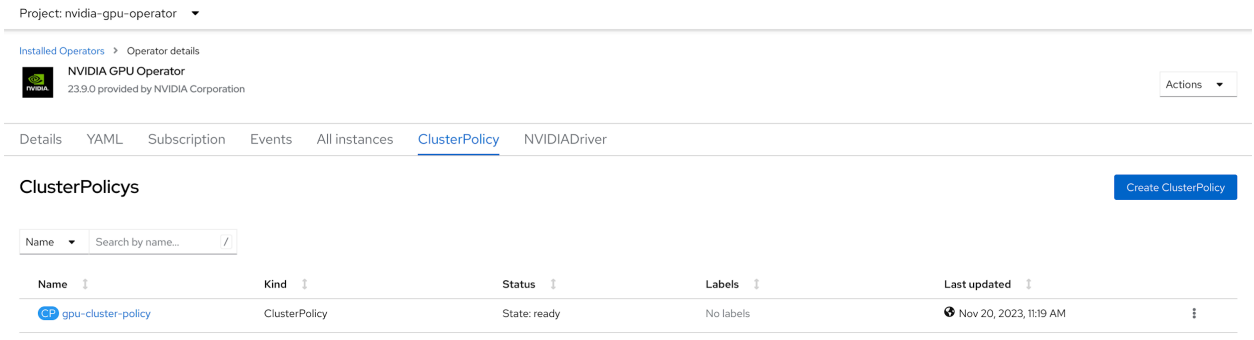
12. Click the Create instance button for the ClusterPolicy object.
13. 

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



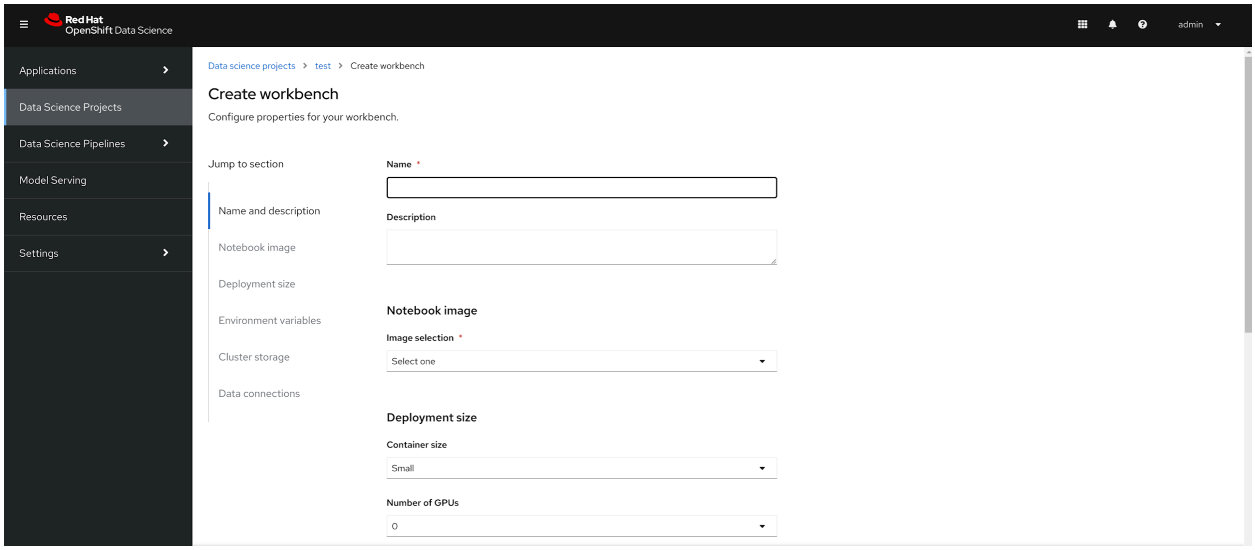
15.

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



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.