Red Hat OpenShift AI Self-Managed 2.10

Working with connected applications

Connect to applications from Red Hat OpenShift AI Self-Managed

Last Updated: 2024-06-25

Red Hat OpenShift AI Self-Managed 2.10 Working with connected applications

Connect to applications from Red Hat OpenShift AI Self-Managed

Abstract

Learn how to enable access to connected applications, remove unused applications from your dashboard, and access and use the Jupyter application that is enabled by default.

Table of Contents

PREFACE

You can extend Red Hat OpenShift AI capabilities by connecting to a wide range of open source and third-party applications, such as Starburst, Anaconda, and IBM watsonx.ai.

You can also remove unused applications from your OpenShift AI dashboard so that you can focus on the applications that you are most likely to use.

CHAPTER 1. VIEWING APPLICATIONS THAT ARE CONNECTED TO OPENSHIFT AI

You can view the available open source and third-party connected applications from the OpenShift AI dashboard.

Prerequisites

- You have logged in to OpenShift AI.

Procedure

1. From the OpenShift AI dashboard, select **Applications → Explore**.
   The **Explore** page displays applications that are available for use with OpenShift AI.

2. Click a tile for more information about the application or to access the **Enable** button.
   Note: The **Enable** button is visible only if an application does not require an OpenShift Operator installation.

Verification

You can access the **Explore** page and click on tiles.

CHAPTER 2. ENABLING APPLICATIONS THAT ARE CONNECTED TO OPENSHIFT AI

You must enable SaaS-based applications, such as Anaconda Professional Edition, before using them with Red Hat OpenShift AI. On-cluster applications are enabled automatically.

Typically, you can install, or enable applications connected to OpenShift AI using one of the following methods:

- Enabling the application from the **Explore** page on the OpenShift AI dashboard, as documented in the following procedure.
- Installing the Operator for the application from OperatorHub. OperatorHub is a web console for cluster administrators to discover and select Operators to install on their cluster. It is deployed by default in OpenShift Container Platform (Installing from OperatorHub using the web console).

  NOTE

  Deployments containing Operators installed from OperatorHub may not be fully supported by Red Hat.

- Installing the Operator for the application from Red Hat Marketplace (Install Operators).
- Installing the application as an Operator to your OpenShift Container Platform cluster (Adding Operators to a cluster).

For some applications (such as Jupyter), the API endpoint is available on the tile for the application on the **Enabled** page of OpenShift AI. Certain applications cannot be accessed directly from their tiles, for example, OpenVINO and Anaconda provide notebook images for use in Jupyter and do not provide an endpoint link from their tile. Additionally, it may be useful to store these endpoint URLs as environment variables for easy reference in a notebook environment.

Some independent software vendor (ISV) applications must be installed in specific namespaces. In these cases, the tile for the application in the OpenShift AI dashboard specifies the required namespace.

To help you get started quickly, you can access the application's learning resources and documentation on the **Resources** page, or on the **Enabled** page by clicking the relevant link on the tile for the application.

Prerequisites

- You have logged in to OpenShift AI.
- Your administrator has installed or configured the application on your OpenShift Container Platform cluster.

Procedure

1. On the OpenShift AI home page, click **Explore**.
2. On the **Explore** page, find the tile for the application that you want to enable.
3. Click **Enable** on the application tile.

4. If prompted, enter the application’s service key and then click **Connect**.

5. Click **Enable** to confirm that you want to enable the application.

Verification

- The application that you enabled appears on the **Enabled** page.

- The API endpoint is displayed on the tile for the application on the **Enabled** page.

CHAPTER 3. REMOVING DISABLED APPLICATIONS FROM THE DASHBOARD

After your administrator has disabled your unused applications, you can manually remove them from the Red Hat OpenShift AI dashboard. Disabling and removing unused applications allows you to focus on the applications that you are most likely to use.

Prerequisites

- You are logged in to Red Hat OpenShift AI.
- Your administrator has disabled the application that you want to remove, as described in Disabling applications connected to OpenShift AI .

Procedure

1. In the OpenShift AI interface, click **Enabled**.
   On the **Enabled** page, tiles for disabled applications are denoted with a **Disabled** label.
2. Click **Disabled** on the tile for the application that you want to remove.
3. Click the link to remove the application tile.

Verification

- The tile for the disabled application no longer appears on the **Enabled** page.

CHAPTER 4. USING THE JUPYTER APPLICATION

Red Hat OpenShift AI provides access to Jupyter as an enabled application for situations where, for example, you do not want users to have their own data science projects or you want to open a notebook that was developed outside of OpenShift AI and has no dependencies on other environments.

Note that the preferred way to access Jupyter on OpenShift AI is through a data science project, as described in Creating a workbench . The advantages to using an OpenShift AI data science project and creating a workbench that includes Jupyter, is that your project organizes your data science work in one place and adds functionality such as data connections so that you can access data and save your models and pipelines for automating your ML workflow.

4.1. STARTING A JUPYTER NOTEBOOK SERVER

Jupyter is based on a server-client architecture. The Jupyter notebook server runs in a container on the Red Hat OpenShift cluster. The client is the JupyterLab interface that opens in your web browser on your local computer. However, all of the commands that you enter in JupyterLab are executed by the notebook server. This architecture allows you to interact through your local computer in a browser environment, while all processing occurs on the cluster. The cluster provides the benefits of larger available resources and security because the data being processed never leaves the cluster.

From the Jupyter application tile, you can start a Jupyter notebook server. If you require extra power for use with large datasets, you can assign accelerators to your notebook server to optimize performance.

Prerequisites

- You are logged in to Red Hat OpenShift AI.
- You are starting Jupyter for the first time or you stopped your notebook server.
- You know the names and values you want to use for any environment variables in your notebook server environment, for example, **AWS_SECRET_ACCESS_KEY**.
- If you want to work with a large data set, work with your administrator to proactively increase the storage capacity of your notebook server. If applicable, also consider assigning accelerators to your notebook server.

Procedure

1. In the left navigation pane, click **Applications > Enabled**.
2. Locate the **Jupyter** tile on the **Enabled applications** page.
3. Click **Launch application**.
   If you see an **Access permission needed** message, you are not in the default user group or the default administrator group for OpenShift AI. Ask your administrator to add you to the correct group by using Adding users.

   If you have not previously authorized the **jupyter-nb-<username>** service account to access your account, the **Authorize Access** page appears prompting you to provide authorization. Inspect the permissions selected by default, and click the **Allow selected permissions** button.

   If your credentials are accepted, the **Notebook server control panel** opens displaying the **Start a notebook server** page.

4. Start a notebook server.

    a. In the **Notebook image** section, select the notebook image to use for your server.
    Different notebook images have different packages installed by default. Click the help icon (?) next to a notebook image name to view a list of its included packages.

    b. If the notebook image contains multiple versions, select the version of the notebook image from the **Versions** section.

    > **NOTE**
    >
    > When a new version of a notebook image is released, the previous version remains available and supported on the cluster. This gives you time to migrate your work to the latest version of the notebook image.

    c. From the **Container size** list, select a suitable container size for your server.

    d. Optional: From the **Accelerator** list, select an accelerator.

    e. If you selected an accelerator in the preceding step, specify the number of accelerators to use.

    > **IMPORTANT**
    >
    > Using accelerators is only supported with specific notebook images. For GPUs, only the PyTorch, TensorFlow, and CUDA notebook images are supported. For Habana Gaudi devices, only the HabanaAI notebook image is supported. In addition, you can only specify the number of accelerators required for your notebook server if accelerators are enabled on your cluster. To learn how to enable GPU support, see Enabling GPU support in OpenShift AI.

    f. Optional: Select and specify values for any new **Environment variables**.
    The interface stores these variables so that you only need to enter them once. Example variable names for common environment variables are automatically provided for frequently integrated environments and frameworks, such as Amazon Web Services (AWS).

    > **IMPORTANT**
    >
    > Select the **Secret** checkbox for variables with sensitive values that must remain private, such as passwords.

    g. Optional: Check **Start server in current tab**

    h. Click **Start server**.
    The **Starting server** progress indicator appears. Click **Expand event log** to view additional information about the server creation process. Depending on the deployment size and resources you requested, starting the server can take up to several minutes. Only click **Cancel** if you want to cancel the server creation.

    After the server starts, you see one of the following behaviors:

    - If you selected **Start server in current tab**in the preceding step, the JupyterLab interface opens in the current tab of your web browser.

- If you did not select **Start server in current tab** in the preceding step, the **Starting server** dialog box prompts you to open the server in a new browser tab or in the current browser tab.

Verification

- The JupyterLab interface opens.

Troubleshooting

- If you see the "Unable to load notebook server configuration options" error message, contact your administrator so that they can review the logs associated with your Jupyter pod and determine further details about the problem.

4.2. CREATING AND IMPORTING NOTEBOOKS

You can create a blank notebook or import a notebook from several different sources.

4.2.1. Creating a new notebook

You can create a new Jupyter notebook from an existing notebook container image to access its resources and properties. The **Notebook server control panel** contains a list of available container images that you can run as a single-user notebook server.

Prerequisites

- Ensure that you have logged in to Red Hat OpenShift AI.
- Ensure that you have launched your notebook server and logged in to JupyterLab.
- The notebook image exists in a registry, image stream, and is accessible.

Procedure

1. Click **File → New → Notebook**.
2. If prompted, select a kernel for your notebook from the list.
   If you want to use a kernel, click **Select**. If you do not want to use a kernel, click **No Kernel**.

Verification

- Check that the notebook file is visible in the JupyterLab interface.

4.2.2. Uploading an existing notebook file from local storage

You can load an existing notebook from local storage into JupyterLab to continue work, or adapt a project for a new use case.

Prerequisites

- Credentials for logging in to JupyterLab.
- A launched and running Jupyter notebook server.

- A notebook file exists in your local storage.

Procedure

1. In the **File Browser** in the left sidebar of the JupyterLab interface, click **Upload Files** ( ).
2. Locate and select the notebook file and click **Open**.
   The file is displayed in the **File Browser**.

Verification

- The notebook file displays in the **File Browser** in the left sidebar of the JupyterLab interface.
- You can open the notebook file in JupyterLab.

4.3. COLLABORATING ON NOTEBOOKS BY USING GIT

If your notebooks or other files are stored in Git version control, you can import them from a Git repository onto your notebook server to work with them in JupyterLab. When you are ready, you can push your changes back to the Git repository so that others can review or use your models.

4.3.1. Uploading an existing notebook file from a Git repository by using JupyterLab

You can use the JupyterLab user interface to clone a Git repository into your workspace to continue your work or integrate files from an external project.

Prerequisites

- A launched and running Jupyter notebook server.
- Read access for the Git repository you want to clone.

Procedure

1. Copy the HTTPS URL for the Git repository.
   - On GitHub, click **Code → HTTPS** and click the Clipboard button.
   - On GitLab, click **Clone** and click the Clipboard button under **Clone with HTTPS**.
2. In the JupyterLab interface, click the **Git Clone** button ( ).

   You can also click **Git → Clone a repository** in the menu, or click the Git icon ( ) and click the **Clone a repository** button.

   The *Clone a repo* dialog appears.
3. Enter the HTTPS URL of the repository that contains your notebook.
4. Click **CLONE**.
5. If prompted, enter your username and password for the Git repository.

Verification

- Check that the contents of the repository are visible in the file browser in JupyterLab, or run the **ls** command in the terminal to verify that the repository is shown as a directory.

4.3.2. Uploading an existing notebook file from a Git repository by using the command line interface

You can use the command line interface to clone a Git repository into your workspace to continue your work or integrate files from an external project.

Prerequisites

- A launched and running Jupyter notebook server.

Procedure

1. Copy the HTTPS URL for the Git repository.
   - On GitHub, click ▮ **Code → HTTPS** and click the Clipboard button.
   - On GitLab, click **Clone** and click the Clipboard button under **Clone with HTTPS**.
2. In JupyterLab, click **File → New → Terminal** to open a terminal window.
3. Enter the **git clone** command.

   ```
   git clone <git-clone-URL>
   ```

   Replace `*<git-clone-URL>*` with the HTTPS URL, for example:

   ```
   [1234567890@jupyter-nb-jdoe ~]$ git clone https://github.com/example/myrepo.git
   Cloning into myrepo...
   remote: Enumerating objects: 11, done.
   remote: Counting objects: 100% (11/11), done.
   remote: Compressing objects: 100% (10/10), done.
   remote: Total 2821 (delta 1), reused 5 (delta 1), pack-reused 2810
   Receiving objects: 100% (2821/2821), 39.17 MiB | 23.89 MiB/s, done.
   Resolving deltas: 100% (1416/1416), done.
   ```

Verification

- Check that the contents of the repository are visible in the file browser in JupyterLab, or run the **ls** command in the terminal to verify that the repository is shown as a directory.

4.3.3. Updating your project with changes from a remote Git repository

You can pull changes made by other users into your data science project from a remote Git repository.

Prerequisites

- You have configured the remote Git repository.
- You have already imported the Git repository into JupyterLab, and the contents of the repository are visible in the file browser in JupyterLab.

- You have permissions to pull files from the remote Git repository to your local repository.
- You have credentials for logging in to Jupyter.
- You have a launched and running Jupyter server.

Procedure

1. In the JupyterLab interface, click the **Git** button ( ).
2. Click the **Pull latest changes** button ( ).

Verification

- You can view the changes pulled from the remote repository in the **History** tab of the Git pane.

4.3.4. Pushing project changes to a Git repository

To build and deploy your application in a production environment, upload your work to a remote Git repository.

Prerequisites

- You have opened a notebook in the JupyterLab interface.
- You have already added the relevant Git repository to your notebook server.
- You have permission to push changes to the relevant Git repository.
- You have installed the Git version control extension.

Procedure

1. Click **File → Save All** to save any unsaved changes.
2. Click the Git icon ( ) to open the Git pane in the JupyterLab interface.
3. Confirm that your changed files appear under **Changed**.
   If your changed files appear under **Untracked**, click **Git → Simple Staging** to enable a simplified Git process.
4. Commit your changes.
   a. Ensure that all files under **Changed** have a blue checkmark beside them.
   b. In the **Summary** field, enter a brief description of the changes you made.
   c. Click **Commit**.
5. Click **Git → Push to Remote** to push your changes to the remote repository.
6. When prompted, enter your Git credentials and click **OK**.

Verification

- Your most recently pushed changes are visible in the remote Git repository.

4.4. MANAGING PYTHON PACKAGES

In JupyterLab, you can view the Python packages that are installed on your notebook image and install additional packages.

4.4.1. Viewing Python packages installed on your notebook server

You can check which Python packages are installed on your notebook server and which version of the package you have by running the **pip** tool in a notebook cell.

Prerequisites

- Log in to JupyterLab and open a notebook.

Procedure

1. Enter the following in a new cell in your notebook:

```
!pip list
```

2. Run the cell.

Verification

- The output shows an alphabetical list of all installed Python packages and their versions. For example, if you use this command immediately after creating a notebook server that uses the **Minimal** image, the first packages shown are similar to the following:

```
Package                          Version
-------------------------------- ----------
aiohttp                          3.7.3
alembic                          1.5.2
appdirs                          1.4.4
argo-workflows                   3.6.1
argon2-cffi                      20.1.0
async-generator                  1.10
async-timeout                    3.0.1
attrdict                         2.0.1
attrs                            20.3.0
backcall                         0.2.0
```

Additional resources

- Installing Python packages on your notebook server

4.4.2. Installing Python packages on your notebook server

You can install Python packages that are not part of the default notebook server by adding the package and the version to a **requirements.txt** file and then running the **pip install** command in a notebook cell.

NOTE

You can also install packages directly, but Red Hat recommends using a **requirements.txt** file so that the packages stated in the file can be easily re-used across different notebooks. In addition, using a **requirements.txt** file is also useful when using a S2I build to deploy a model.

Prerequisites

- Log in to JupyterLab and open a notebook.

Procedure

1. Create a new text file using one of the following methods:
   - Click **+** to open a new launcher and click **Text file**.
   - Click **File → New → Text File**.
2. Rename the text file to **requirements.txt**.
   a. Right-click on the name of the file and click **Rename Text**. The **Rename File** dialog opens.
   b. Enter **requirements.txt** in the **New Name** field and click **Rename**.
3. Add the packages to install to the **requirements.txt** file.

```
altair
```

You can specify the exact version to install by using the **==** (equal to) operator, for example:

```
altair==4.1.0
```

NOTE

Red Hat recommends specifying exact package versions to enhance the stability of your notebook server over time. New package versions can introduce undesirable or unexpected changes in your environment's behavior.

To install multiple packages at the same time, place each package on a separate line.

4. Install the packages in **requirements.txt** to your server using a notebook cell.
   a. Create a new cell in your notebook and enter the following command:

```
!pip install -r requirements.txt
```

   b. Run the cell by pressing Shift and Enter.

IMPORTANT

This command installs the package on your notebook server, but you must still run the **import** directive in a code cell to use the package in your code.

```
import altair
```

Verification

- Confirm that the packages in **requirements.txt** appear in the list of packages installed on the notebook server. See Viewing Python packages installed on your notebook server for details.

4.5. UPDATING NOTEBOOK SERVER SETTINGS BY RESTARTING YOUR SERVER

You can update the settings on your notebook server by stopping and relaunching the notebook server. For example, if your server runs out of memory, you can restart the server to make the container size larger.

Prerequisites

- A running notebook server.
- Log in to JupyterLab.

Procedure

1. Click **File → Hub Control Panel**.
   The **Notebook server control panel** opens.
2. Click the **Stop notebook server** button.
   The **Stop server** dialog opens.
3. Click **Stop server** to confirm your decision.
   The **Start a notebook server** page opens.
4. Update the relevant notebook server settings and click **Start server**.

Verification

- The notebook server starts and contains your updated settings.