Red Hat OpenShift Data Science 1

Developing and deploying data models

Understand the development and deployment workflow and deploy your data models in intelligent applications

Last Updated: 2022-01-28

Red Hat OpenShift Data Science 1 Developing and deploying data models

Understand the development and deployment workflow and deploy your data models in intelligent applications

Legal Notice

Abstract

Understand the development and deployment workflow and deploy your data models in intelligent applications.

Table of Contents

PREFACE

This documentation is provided for the Field Trial release of Red Hat OpenShift Data Science.

See the following documents for service and life cycle information related to this Field Trial release:

- OpenShift Data Science Service Definition
- OpenShift Data Science Life Cycle

CHAPTER 1. OVERVIEW OF DEVELOPING AND DEPLOYING DATA MODELS

Read this section to understand the work required to develop and deploy an application that uses a predictive model created using Red Hat OpenShift Data Science.

Your organization might split responsibility for this process between several roles, such as a data scientist and an application developer, or this work might be done by a single role. An appropriate role is noted for each step.

Table 1.1. Development tasks by role

Application developer	Data scientist	Task description
✔		Create a Python S2I project in Git using an OpenShift Data Science application template. • Method 1: Use GitHub templates. • Method 2: Use the CookieCutter project generator.
✔		Configure user access to the Git project so that data scientists can push to and pull from the repository.
From this point, you can develop the model and the application that uses it simultaneously.		
✔		Create an OpenShift application using the project repository.
✔		Build the OpenShift application to verify your code.
✔		Automate the build process using webhooks.
	✔	Launch JupyterHub and either create or import a notebook.
	✔	Import the application Git project into JupyterLab.
	✔	Develop and test your model using notebooks in JupyterLab.
	✔	Save your model as an independent Python function in a separate Python file.
	✔	Update the **requirements.txt** file with dependencies your function requires.
	✔	Test the function on your notebook server.
	✔	Push your updates back to the remote Git project.
✔	✔	Test the deployed application endpoint.

CHAPTER 2. CREATING A PYTHON S2I APPLICATION FOR OPENSHIFT DATA SCIENCE

2.1. CREATING A PYTHON S2I APPLICATION FOR OPENSHIFT DATA SCIENCE FROM A GITHUB TEMPLATE

You can create an application suitable for Red Hat OpenShift Data Science quickly by using Red Hat's Python S2I application repository as a template in GitHub. Use the template to generate a new repository with the same format, directory structure and files as an existing Red Hat OpenShift Data Science repository.

Prerequisites

- You have a GitHub account.
- You have credentials to access the GitHub repository containing the relevant template that you want to use.

Procedure

1. On GitHub, navigate to the main page of the template repository.
2. Click **Use this template**.
3. Optional: From the **Owner** list, select the account that you want to own the repository.
4. In the **Repository name** field, enter a name for the new repository.
5. Optional: In the **Description** field, enter a description for the new repository.
6. Set the repository's visibility level.
    a. To ensure that the repository is visible to anyone, leave **Public** selected. By default, the repository's visibility is set to **Public**.
    b. Click **Private** to restrict who can see and commit to the repository.
7. Optional: Select the **Include all branches** check box to copy the template repository's branches to your new repository.
8. Click **Create repository from template**.

Verification

- The repository that you created from the template is visible and accessible from your GitHub account.

2.2. CREATING A PYTHON S2I APPLICATION FOR OPENSHIFT DATA SCIENCE USING COOKIECUTTER

You can create an application suitable for Red Hat OpenShift Data Science quickly by using Cookiecutter. Cookiecutter is a Python library that creates a flexible, standardized project structure for your data science work. You can use Cookiecutter to further customize your project's repository. For

example, you can modify the repository's directory structure to suit your project's requirements.

Prerequisites

- A launched and running JupyterHub server.
- You have a GitHub account.
- You have credentials to access the GitHub repository containing the template that you want to use.

Procedure

1. In the JupyterLab interface, click **File → New → Terminal**.
2. In the terminal, run the **pip install** command to install Cookiecutter.

   ```
   pip install cookiecutter
   ```

3. Run the **cookiecutter** command to create a project from a Cookiecutter repository template.

   ```
   cookiecutter template-repository-url
   ```

   Replace *template-repository-url* with the template repository's URL. For example, https://github.com/cookiecutter-template/template.

4. When prompted, provide the following information:
   a. A name for your project.
   b. A name for your repository.
   c. A name for the project's author.
   d. A description for your project.
   e. Your open source license file type.
   The contents of the Cookiecutter template repository appear in the **File Browser** in the left sidebar.
5. Create a repository in GitHub.
   a. In the upper-right corner of the GitHub home page, click **+ → New repository**.
   The **Create a new repository** page opens.
   b. In the **Repository template** field, select the template that you want to use.
   c. Optional: Select the **Include all branches** check box to copy the template repository's branches to your new repository.
   d. In the **Owner** field, select the repository owner's user name.
   e. In the **Repository** name field, enter a name for the repository.
   f. Optional: In the **Description** field, enter a description of the repository.

6. Set the repository's visibility level.
    a. To ensure that the repository is visible to anyone, leave **Public** selected. By default, the repository's visibility is set to **Public**.
    b. Click **Private** to choose who can see and commit to the repository.
    c. Click **Create repository**.
7. Clone the repository on your JupyterHub server.
    a. In the JupyterLab interface, click **Git → Clone a Repository**.
    The **Clone a repo** dialog appears.
    b. Enter the URL of the repository that you want to clone.
    c. Click **Clone**.
    The cloned repository appears in the **File Browser** in the left sidebar.
    d. In the **File Browser**, move the files and directories created by Cookiecutter to the repository that you cloned.
8. Push your changes to the remote repository.
    a. In the left sidebar, click **Git** ().
    b. If you have untracked changes, in the **Changes** tab, hover the cursor over the **Untracked** section bar and click  .
    c. If you have files that contain changes, in the **Changes tab**, hover the cursor over the **Changed** section bar and click  .
    d. In the **Required** field, enter a summary of your changes.
    e. In the **Description** field, enter a description of your changes.
    f. Click **Commit**.
    g. In the JupyterLab interface, click **Git → Push to Remote** to push your changes to the remote repository.
    The **Git credentials required** dialog opens.
    h. Enter your credentials to access the remote repository.
    i. Click **OK**.

Verification

- You can access the remote repository that you created from the template.
- You can see the changes that you pushed in the remote repository.

CHAPTER 3. CONFIGURING USER ACCESS TO THE REMOTE GIT REPOSITORY

Your data scientists and application developers need developer access to the remote Git repository in order to push and pull data from the repository. The repository owner can add these users to the repository as developers to enable this access.

For **GitHub** repositories, refer to the GitHub documentation:

- For personal repositories: Inviting collaborators
- For organization repositories: Adding organization members

CHAPTER 4. CREATING AN OPENSHIFT APPLICATION FROM A GIT REPOSITORY

You can import code from a Git repository and use it to create, build, and deploy a Red Hat OpenShift Data Science application on OpenShift Dedicated.

Prerequisites

- You have logged in to the OpenShift Dedicated web console.
- You are in the **Developer** perspective.
- You have the appropriate roles and permissions in a project to create applications and other workloads in OpenShift Dedicated.
- You have a configured Git repository.
- You have permissions for importing the Git repository.

Procedure

1. In OpenShift Dedicated, select the project to create the application in, or create a new project for the application.
2. In the **+Add** view, click **From Git** to see the **Import from Git** form.
3. In the **Git** section, enter the Git repository URL for the codebase you want to use to create an application.
4. Optional: Click **Show Advanced Git Options** to add details such as:
    - **Git Reference** to point to code in a specific branch, tag, or commit to be used to build the application.
    - **Context Dir** to specify the subdirectory for the application source code you want to use to build the application.
    - **Source Secret** to create a **Secret Name** with credentials for pulling your source code from a private repository.
5. In the **Builder** section the appropriate builder image is detected and selected by default.
6. In the **General** section:
    a. In the **Application** field, enter a unique name for the application grouping. This must be unique in the project.
    b. The **Name** field is populated automatically based on the git repository URL. This is used to identify the resources created for this application.
7. In the **Resources** section, select **Deployment Config**, to create an OpenShift style application.
8. In the **Advanced Options** section:
    a. The **Create a route to the application** checkbox is selected by default so that you can access your application using a publicly available URL.

If you do not want to expose your application on a public route, clear the checkbox.

b. Optional: Click **Routing** to display advanced routing options.

   i. Customize the hostname for the route.

   ii. Specify the path the router watches.

   iii. Select the target port for traffic on the route.

   iv. Configure transport security for the route.

c. Optional: Click **Build configuration** to display advanced build configuration options, including any environment variables that your model requires to build.

d. Optional: Click **Deployment configuration** to display advanced deployment configuration options, including any environment variables that your model requires in its deployment environment.
For example, if your application integrates OpenShift Streams for Apache Kafka, add your Kafka environment variables here.

e. Optional: Click **Scaling** to define the number of pods or application instances to deploy initially.

f. Optional: Click **Resource Limit** to set the amount of **CPU** and **Memory** resources a container is guaranteed or allowed to use when running.

g. Optional: Click **Labels** to add custom labels to your application.

9. Click **Create** to create the application and see its build status in the **Topology** view.

Verification

- You can view your application in the **Topology** view.
- Click the application and check the **Resources** tab of the application details pane. Look for a success message under **Builds**, for example, **Build #1 is complete**.

Additional resources

- Creating applications using the Developer perspective
- Accessing the web console
- About the Developer perspective in the web console
- Default cluster roles

CHAPTER 5. BUILDING AN OPENSHIFT APPLICATION USING THE WEB CONSOLE

You can manually tell OpenShift Dedicated to build an existing OpenShift application with the **Start Build** button in the OpenShift Dedicated web console.

Prerequisites

- You have Developer access to OpenShift Dedicated.
- You have created an OpenShift Dedicated application.

Procedure

1. In OpenShift Dedicated, set the **Project** dropdown to your application project.
2. Click **Topology**.
3. Click on the application to see the application details pane.
4. Click the **Start build** button.

Additional resources

- Performing basic builds
- Automatically rebuilding updated applications using webhooks

CHAPTER 6. AUTOMATICALLY REBUILDING UPDATED APPLICATIONS USING WEBHOOKS

You can configure an OpenShift application to automatically rebuild and redeploy whenever updates are made to the Git repository that contains the application code. This ensures that the latest working version of your application is always available.

Prerequisites

- An OpenShift application created using a GitHub repository as a source.
- Permissions to change webhook settings in the GitHub repository.

Procedure

1. In OpenShift Dedicated, change into the **Developer** perspective and set the **Project** dropdown to the appropriate project.
2. Click **Topology** and click on your application to view the application details pane.
3. Under **Builds**, click the name of the build configuration, marked with **BC**, to view the build configuration page.
4. Under **Webhooks**, locate the entry for GitHub and click **Copy URL with Secret**
5. Navigate to your project page in GitHub and click **Settings**.
6. Click **Webhooks → Add webhook**.
7. Enter the following details on the **Add webhook** page:
    a. Paste the copied URL with secret into the **Payload URL** field.
    b. Set **Content type** to **application/json**.
    c. Leave all other options as the default.
    d. Click **Add webhook**.

Verification

- Make an update to your application code and verify that the application rebuilds and deploys correctly.

Additional resources

- Triggering and modifying builds
- Creating webhooks

CHAPTER 7. CREATING OR IMPORTING A NOTEBOOK IN JUPYTERHUB

7.1. CREATING A NEW NOTEBOOK

You can create a new Jupyter notebook from an existing notebook container image to access its resources and properties. The JupyterHub Spawner 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 Data Science.
- Ensure that you have logged in to JupyterHub and launched your notebook server.
- 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.

7.2. IMPORTING AN EXISTING NOTEBOOK FILE FROM A GIT REPOSITORY 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 JupyterHub 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.

CHAPTER 8. SAVING YOUR MODEL AS AN INDEPENDENT PYTHON FUNCTION

Turn your data model into an independent Python function so that you can run it outside your notebook server environment and use it in intelligent applications.

Prerequisites

- You have access to the JupyterLab interface.
- You have developed a prediction model in a Jupyter notebook.
- Your Jupyter notebook is saved in a Git repository that was created from the Red Hat OpenShift Data Science sample S2I application repositories.

Procedure

1. In JupyterLab, create a new **prediction.py** file.
2. Edit the **prediction.py** file to define a **predict** function based on the prediction model in your Jupyter notebook.
   - Include only the code required to make the prediction. For example, you do not need to import libraries that only related to rendering plots in your Jupyter notebook.
   - If any new packages are required to run your prediction, update the contents of the **requirements.txt** file and run **pip install -r requirements.txt** to install the new packages.
3. Test that you can run the independent Python function from your notebook by calling the function in a new notebook cell, for example:

```
from prediction import predict
 predict(data)
```

Verification

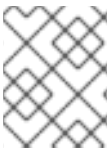
- The **predict** function runs correctly and returns the expected output when called from the notebook cell.

Additional resources

- Installing Python packages on your notebook server

8.1. INSTALLING PYTHON PACKAGES ON YOUR NOTEBOOK SERVER

You can install Python packages that are not part of the default notebook server image 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 it is easier to deploy your model later.

Prerequisites

- Log in to JupyterHub 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
```

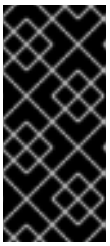
   You can specify the exact version to install by using the **==** (equal to) operator, for example:

```
altair==4.1.0
```

   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 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.

CHAPTER 9. TESTING YOUR PYTHON FUNCTION USING THE SAMPLE FLASK APPLICATION

You should test that your independent Python function works as expected before it goes into your application.

Prerequisites

- You have created an application from one of the S2I sample repositories by following the instructions in one of the following sections:
  - Creating a Python S2I application for OpenShift Data Science from a GitHub template
  - Creating a Python S2I application for OpenShift Data Science using Cookiecutter
- You have created a separate function for your model by following the instructions in Saving your mnodel as an independent Python function .

Procedure

1. In JupyterLab, open the **run_flask.ipynb** notebook file.
2. Run all cells in the notebook by clicking **Cell → Run All**.
   This starts the Flask application.

Verification

- Open a terminal in JupyterLab by clicking **File → New → Terminal** and run the following command:

```
curl -X POST -H "Content-Type: application/json" --data '{"data" : "hello world"}'
http://localhost:5000/prediction
```

Alternatively, enter the following in a new notebook cell and run the cell.

```
!curl -X POST -H "Content-Type: application/json" --data '{"data" : "hello world"}'
http://localhost:5000/prediction
```

If no changes have been made to the sample app, you see a response in the browser similar to **{"prediction" : "not implemented"}**.

CHAPTER 10. 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 JupterLab 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.

CHAPTER 11. TESTING THE DEPLOYED APPLICATION ENDPOINT FOR YOUR PREDICTION FUNCTION

After you deploy your application, you can test that your prediction function works properly at the deployed endpoint.

Prerequisites

- Your application is built and deployed with your prediction function included.
- You know the web address for the application containing your prediction function.

Procedure

1. Open a terminal in JupyterLab by clicking **File → New → Terminal**.
2. Run the following command, replacing **<application-url>** with the web address for the application, for example, **http://myapp-myproject.apps.mycluster.abc1.s1.devshift.org**.

   ```
   curl -X POST -H "Content-Type: application/json" --data {"data" : "hello world"} <application-url>/prediction
   ```

   For example:

   ```
   curl -X POST -H "Content-Type: application/json" --data {"data" : "hello world"} http://myapp-myproject.apps.mycluster.abc1.s1.devshift.org/prediction
   ```

   Alternatively, enter **!** followed by the same command in a new notebook cell and run the cell.

   ```
   !curl -X POST -H "Content-Type: application/json" --data {"data" : "hello world"} <application-url>/prediction
   ```

Verification

- The endpoint is working if you receive a response from the application, such as **{"prediction" : "not implemented"}**.