Red Hat OpenShift Data Science 1

开发数据模型

了解开发和部署工作流，并在智能应用程序中部署数据模型

Last Updated: 2023-10-05

Red Hat OpenShift Data Science 1 开发数据模型

了解开发和部署工作流，并在智能应用程序中部署数据模型

法律通告

摘要

了解开发和部署工作流，并在智能应用程序中部署数据模型。

目录

第 1 章 开发数据模型概述

阅读本节，了解开发和部署使用 Red Hat OpenShift Data Science 创建的预测模型的应用程序所需的工作。

您的组织可能将这些责任拆分给多个不同的人员（如数据科学家，应用程序开发人员），也可能有单一角色完成。每个步骤都会指明适当的角色。

表 1.1. 按角色分配任务

应用程序开发人员	数据科学家	任务描述
✔		使用 OpenShift Data Science 应用模板，在 Git 中创建 Python S2I 项目。 • 方法 1：使用 GitHub 模板。 • 方法 2：使用 CookieCutter 项目生成器。
✔		配置用户对 Git 项目的访问权限，以便数据科学家可从存储库推送到和拉取。
此时，您可以开发模型以及同时使用它的应用程序。		
✔		使用项目存储库创建 OpenShift 应用。
✔		构建 OpenShift 应用以验证您的代码。
✔		使用 webhook 自动化构建流程。
	✔	启动 Jupyter，创建或导入一个笔记本。
	✔	将应用 Git 项目导入到 JupyterLab。
	✔	使用 JupyterLab 中的笔记本开发和测试您的模型。
	✔	在一个单独的 Python 文件中将您的模型保存为一个单独的 Python 功能。
	✔	更新 **requirements.txt** 文件，包括您的功能所需的依赖项。
	✔	测试笔记本服务器中的功能。
	✔	将您的更新推送回远程 Git 项目。
✔	✔	测试部署的应用端点。

第 2 章 为 OPENSHIFT DATA SCIENCE 创建 PYTHON S2I 应用

2.1. 从 GITHUB 模板为 OPENSHIFT DATA SCIENCE 创建 PYTHON S2I 应用

您可以使用红帽的 Python S2I 应用程序存储库作为 GitHub 中的模板，快速创建适合 Red Hat OpenShift Data Science 的应用。使用模板生成一个新的存储库，起名称、目录结构和文件与现有 Red Hat OpenShift Data Science 存储库相同。

先决条件

- 您有一个 GitHub 帐户。
- 您有凭证来访问包含您要使用的相关模板的 GitHub 存储库。

流程

1. 在 GitHub 上，进入到模板仓库的主页（红帽 Python S2I 应用存储库）。
2. 点 **Use this template**。
3. 可选：在 **Owner** 列表中，选择您要拥有该存储库的帐户。
4. 在 **Repository name** 字段中，输入新存储库的名称。
5. 可选：在 **Description** 字段中输入新存储库的描述。
6. 设置存储库的可见性级别。
   a. 为确保存储库对任何人可见，请保留 **Public** （公共）。默认情况下，存储库的可见性设置为 **Public**。
   b. 点 **Private** 来限制谁可以查看并提交到存储库。
7. 可选： 选择 **Include all** branch 复选框，将模板存储库的分支复制到您的新存储库中。
8. 点 **Create repository from template**。

验证

- 您从模板创建的存储库可以从您的 GitHub 帐户查看并可访问。

2.2. 使用 COOKIECUTTER 为 OPENSHIFT DATA SCIENCE 创建 PYTHON S2I 应用程序

您可以使用 Cookiecutter 快速创建适合 Red Hat OpenShift Data Science 的应用程序。Cookiecutter 是一个 Python 库，可为您的数据科学工作创建一个灵活、标准化的项目结构。您可以使用 Cookiecutter 来进一步自定义项目的存储库。例如，您可以修改存储库的目录结构，以满足您的项目的要求。

先决条件

- 启动并运行 Jupyter 服务器。
- 您有一个 GitHub 帐户。

- 您有凭证来访问包含您要使用的相关模板的 GitHub 存储库。

流程

1. 在 JupyterLab 界面中，点 **File → New → Terminal**。
2. 在终端中，运行 **pip install** 命令来安装 Cookiecutter。

```
pip install cookiecutter
```

3. 运行 **cookiecutter** 命令，从 Cookiecutter 存储库模板创建项目。

```
cookiecutter template-repository-URL
```

将 *template-repository-URL* 替换为模板存储库的 URL: **https://github.com/opendatahub-io/odh-s2i-project-cookiecutter**

4. 提示时，提供以下信息：
   a. 项目的名称。
   b. 存储库的名称。
   c. 项目的作者名称。
   d. 项目的描述。
   e. 您的开源许可证文件类型。
   Cookiecutter 模板存储库的内容显示在左侧栏中的 **File Browser** 中。
5. 在 GitHub 中创建存储库。
   a. 在 GitHub 主页右上角，点击 **+ → New repository**。
   这时将打开 **Create a new repository** 页面。
   b. 在 **Repository template** 字段中，选择要使用的模板。
   c. 可选： 选择 **Include all** branch 复选框，将模板存储库的分支复制到您的新存储库中。
   d. 在 **Owner** 字段中，选择存储库所有者的用户名。
   e. 在 **Repository** name 字段中，输入存储库的名称。
   f. 可选：在 **Description** 字段中输入存储库的描述。
6. 设置存储库的可见性级别。
   a. 为确保存储库对任何人可见，请保留 **Public** （公共）。默认情况下，存储库的可见性设置为 **Public**。
   b. 点 **Private** 以选择哪些用户可以查看存储库并提交到存储库。
   c. 点 **Create repository**。
7. 在您的 Jupyter 服务器上克隆存储库。
   a. 在 JupyterLab 界面中，点 **Git → Clone a Repository**。

此时会显示**克隆存储库**对话框。

b. 输入您要克隆的存储库的 URL。

c. 单击 **Clone**。
克隆的存储库位于左侧栏中的**文件浏览器**中。

d. 在文件**浏览器**中，将 Cookiecutter 创建的文件和目录移到您克隆的存储库。

8. 将您的更改推送到远程存储库。

a. 在左侧侧边栏中，点 **Git** ()。

b. 如果您有未跟踪的更改，请在 **Changes** 选项卡中将光标悬停在 **Untracked** 部分栏上，然后点  。

c. 如果您的文件包含更改，在 **Changes** 选项卡中，将光标悬停在 **Changed** 部分栏上，点  。

d. 在 **Required** 字段中，输入您的更改概述。

e. 在 **Description** 字段中，输入您的更改描述。

f. 点 **Commit**。

g. 在 JupyterLab 接口中，点击 **Git → Push to Remote** 将您的更改推送到远程存储库。
此时会打开**所需的 Git 凭证**。

h. 输入您的凭证来访问远程存储库。

i. 点击 **OK**。

验证

- 您可以访问从模板创建的远程存储库。
- 您可以查看推送在远程存储库中的更改。

第 3 章 配置用户对远程 GIT 存储库的访问权限

您的数据科学家和应用程序开发人员需要开发人员访问远程 Git 存储库，以便从存储库推送和拉取镜像。存储库所有者可以作为开发人员将这些用户添加到存储库中，以启用此访问权限。

对于 GitHub 仓库，请参阅 GitHub 文档：

- 对于个人存储库：邀请协作人员
- 对于机构仓库： 添加机构成员

第 4 章 从 GIT 存储库创建 OPENSHIFT 应用

您可以从 Git 存储库导入代码，并使用它来在 OpenShift Dedicated 上创建、构建和部署 Red Hat OpenShift Data Science 应用程序。

先决条件

- 已登陆到 OpenShift Dedicated Web 控制台。
- 处于 **Developer** 视角。
- 在项目中拥有适当的角色和权限，可在 OpenShift Dedicated 中创建应用程序和其他工作负载。
- 您已配置了 Git 存储库。
- 有导入 Git 存储库的权限。

流程

1. 在 OpenShift Dedicated 中，选择要在其中创建应用程序的项目，或为应用程序创建一个新项目。
2. 在 **+Add** 视图中，点 **From Git** 以查看 **Import from Git** 表单。
3. 在 **Git** 部分中，输入您要用来创建应用程序的代码库的 Git 存储库 URL。
4. 可选：点击 **Show Advanced Git Options** 来添加详情，例如：
   - **git Reference**，指向特定的分支、标签或提交中的代码，以用于构建应用程序。
   - **Context Dir**，指定要用来构建应用程序的应用程序源代码的子目录。
   - **Source Secret**，创建一个具有用来从私有存储库拉取源代码的凭证的 **Secret Name**。
5. 在 **Builder** 部分中，会检测到适当的构建器镜像，并被默认选择。
6. 在 **General** 部分中：
   a. 在 **Application** 字段中输入应用程序组别的唯一名称。在项目中这必需是唯一的。
   b. **Name** 字段根据 Git 存储库 URL 自动填充。这用于识别为此应用创建的资源。
7. 在 **Resources** 部分中，选择 **Deployment Config**，以创建 OpenShift 风格的应用程序。
8. 在 **Advanced Options** 部分中：
   a. 默认选择 **Create a route to the application** 复选框，以便您可以使用公开的 URL 访问应用程序。
   如果您不想在公共路由上公开应用程序，请清除该复选框。
   b. 可选：点 **Routing** 以显示高级路由选项。
      i. 自定义路由的主机名。
      ii. 指定路由器监控的路径。
      iii. 为路由上的流量选择目标端口。

iv. 配置路由的传输安全性。

c. 可选：点 **Build configuration** 以显示高级构建配置选项，包括您的模型需要构建的任何环境变量。

d. 可选：点 **Deployment configuration** 来显示高级部署配置选项，包括模型在其部署环境中需要的任何环境变量。

e. 可选：点 **Scaling** 以定义初始部署的 pod 或应用程序实例的数量。

f. 可选：点 **Resource Limit** 设置容器保证或允许在运行时使用的 **CPU** 和 **Memory** 资源量。

g. 可选：点 **Labels** 为应用程序添加自定义标签。

9. 点击 **Create**，以创建应用程序并在 **Topology** 视图中查看其构建状态。

验证

- 您可以在 **Topology** 视图中查看应用程序。
- 点应用程序，再检查应用程序详情窗格的 **Resources** 选项卡。在 **Builds** 中查找成功消息，例如 **Build #1 is complete**。

其他资源

- 使用 Developer 视角创建应用程序
- 访问Web控制台
- 关于 web 控制台中的开发者视角
- 默认集群角色

第 5 章 使用 WEB 控制台构建 OPENSHIFT 应用程序

您可以手动告知 OpenShift Dedicated 使用 OpenShift Dedicated Web 控制台中的 **Start Build** 按钮构建现有的 OpenShift 应用。

先决条件

- 有开发人员对 OpenShift Dedicated 的访问权限。
- 您已创建了 OpenShift Dedicated 应用。

流程

1. 在 OpenShift Dedicated 中，将**项目** 下拉菜单设置为应用程序项目。
2. 点 **Topology**。
3. 点击应用程序查看应用程序详情窗格。
4. 点 **Start build** 按钮。

其他资源

- 执行基本构建
- 使用 Webhook 自动重建更新的应用程序

第 6 章 使用 WEBHOOK 自动重建更新的应用程序

您可以将 OpenShift 应用配置为在对包含应用代码的 Git 存储库进行更新时自动重建和重新部署。这样可确保应用程序的最新工作版本始终可用。

先决条件

- 使用 GitHub 存储库作为来源创建的 OpenShift 应用。
- 更改 GitHub 仓库中的 Webhook 设置的权限。

流程

1. 在 OpenShift Dedicated 中，切换到 **Developer** 视角，并将项目下拉菜单设置为适当的项目。
2. 点 **Topology** 并点应用程序来查看应用程序详情面板。
3. 在 **Builds** 下，单击标记为 **BC** 的构建配置的名称，以查看构建配置页面。
4. 在 **Webhooks** 下，找到 GitHub 的条目，再单击 **Copy URL with Secret**。
5. 导航到 GitHub 中的项目页面，再单击 **Settings**。
6. 点 **Webhooks → Add webhook**。
7. 在 **Add webhook** 页中输入以下信息：
   a. 将复制的 URL 使用 secret 粘贴到 **Payload URL** 字段中。
   b. 将 **Content type** 设置为 **application/json**。
   c. 所有其他选项保留默认值。
   d. 点击 **Add webhook**。

验证

- 更新应用程序代码，并验证应用程序重新构建并部署正确。

其他资源

- 触发和修改构建
- 创建 Webhook

第 7 章 在 JUPYTER 中创建或导入笔记本

7.1. 创建新笔记本

您可以从现有笔记本容器镜像创建一个新的 Jupyter 笔记本，以访问其资源和属性。启动一个笔记本服务器页面包含可用容器镜像列表，您可以作为单用户笔记本服务器运行。

先决条件

- 确保您已登陆到 Red Hat OpenShift Data Science。
- 确保您已启动了笔记本服务器并登录到 Jupyter。
- 笔记本镜像存在于 registry、镜像流中，并可访问。

流程

1. 点 **File** → **New** → **Notebook**。
2. 如果出现提示，请从列表中为您的笔记本选择一个内核。
   如果要使用内核，点 **Select**。如果您不想使用内核，点 **No Kernel**。

验证

- 检查笔记本文件是否在 JupyterLab 接口中看到。

7.2. 使用 JUPYTERLAB 从 GIT 存储库上传现有的笔记本文件

您可以使用 JupyterLab 用户界面将 Git 存储库克隆到工作区，以继续您的工作或从外部项目集成文件。

先决条件

- 启动并运行 Jupyter 服务器。
- 要克隆的 Git 存储库的读取访问权限。

流程

1. 复制 Git 存储库的 HTTPS URL。
   - 在 GitHub 中，点击 **zfcp Code** → **HTTPS**，然后点击 Clipboard 按钮。
   - 在 GitLab 上，单击 **Clone**，再单击 **Clone with HTTPS** 下的 Clipboard 按钮。
2. 在 JupyterLab 界面中，点 **Git Clone** 按钮()。

   您还可以在菜单中点击 **Git** → **Clone a repository**，或者点击 Git 图标()并点击 **Clone a repository** 按钮。

   此时会显示*克隆存储库*对话框。
3. 输入包含笔记本的存储库的 HTTPS URL。
4. 点 **CLONE**。

5. 若有提示，请输入您的 Git 存储库的用户名和密码。

验证

- 检查存储库的内容是否在 JupyterLab 中的文件浏览器中看到，或者在终端中运行 **ls** 命令，以验证存储库是否显示为一个目录。

第 8 章 将您的模型保存为独立的 PYTHON 功能

将您的数据模型转换成独立的 Python 功能，以便您可以在笔记本服务器环境外部运行并在智能应用程序中使用它。

先决条件

- 您可以访问 JupyterLab 接口。
- 您已在 Jupyter 笔记本中开发了预测模型。
- 您的 Jupyter 笔记本保存在从 Red Hat OpenShift Data Science sample S2I 应用程序存储库中创建的 Git 存储库中。

流程

1. 在 JupyterLab 中，创建一个新的 **prediction.py** 文件。
2. 编辑 **prediction.py** 文件，以根据 Jupyter 笔记本中的预测模型定义一个 **predict** 功能。
   - 仅包括进行预测所需的代码。例如，您不需要导入仅与 Jupyter 笔记本中呈现图表相关的库。
   - 如果需要新的软件包来运行您的预测，请更新 **requirements.txt** 文件的内容，并运行 **pip install -r requirements.txt** 以安装新软件包。
3. 通过在新的笔记本单元中调用功能，测试您可以从笔记本中运行独立的 Python 功能，例如：

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

验证

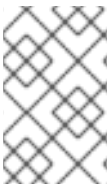
- **predict** 功能可以正常工作，并在从笔记本单元调用时返回预期的输出。

其他资源

- 在笔记本服务器中安装 Python 软件包

8.1. 在笔记本服务器中安装 PYTHON 软件包

您可以通过将软件包和版本添加到 **requirements.txt** 文件中，安装不是默认笔记本服务器镜像一部分的 Python 软件包，然后在笔记本单元中运行 **pip install** 命令。



注意

您还可以直接安装软件包，但红帽建议您使用 **requirements.txt** 文件，以便在不同的笔记本中轻松重复使用该文件中声明的软件包。此外，在使用 S2I 构建来部署模型时，使用 **requirements.txt** 文件也很有用。

先决条件

- 登录 Jupyter 并打开笔记本。

流程

1. 使用以下方法之一创建新文本文件：
   - 单击 **+** 以打开新启动程序，然后单击**文本文件**。
   - 点 **File → New → Text File**。
2. 将文本文件重命名为 **requirements.txt**。
   a. 在文件的名称上单击鼠标右键，然后单击**重命名文本**。此时会打开 **Rename File** 对话框。
   b. 在 **New Name** 字段中输入 **requirements.txt**，然后单击 **Rename**。
3. 将要安装的软件包添加到 **requirements.txt** 文件中。

```
altair
```

   您可以使用 **==** (等于) 操作符指定要安装的确切版本，例如：

```
altair==4.1.0
```

注意

红帽建议指定准确的软件包版本，以便随着时间的推移增强您的笔记本服务器的稳定性。在环境行为中，新的软件包版本可能会带来不必要的更改或意外更改。

   要同时安装多个软件包，请将每个软件包放在单独的行中。

4. 使用笔记本电脑单元将 **requirements.txt** 中的软件包安装到您的服务器中。
   a. 在笔记本中创建新单元并输入以下命令：

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

   b. 按 Shift 和 Enter 运行单元。

重要

这会在笔记本电脑服务器上安装软件包，但您仍必须在代码单元中运行 **import** 指令，以便在您的代码中使用软件包。

```
import altair
```

验证

- 确认 **requirements.txt** 中的软件包显示在笔记本电脑服务器上安装的软件包列表中。详情请参阅笔记本服务器上安装的 Python 软件包。

第 9 章 使用示例 FLASK 应用程序测试您的 PYTHON 功能

您使用应用程序前，您应该测试独立 Python 功能是否可以按预期工作。

先决条件

- 您已按照以下部分之一中的说明从 S2I 示例存储库创建了应用程序：
  - 从 GitHub 模板为 OpenShift Data Science 创建 Python S2I 应用
  - 使用 Cookiecutter 为 OpenShift Data Science 创建 Python S2I 应用程序
- 您已按照保存模型作为一个独立的 Python 功能中的说明为您的模型创建了一个独立的功能。

流程

1. 在 JupyterLab 中，打开 **run_flask.ipynb** 笔记本文件。
2. 点 **Cell → Run All**，在笔记本中运行所有单元。
   这将启动 Flask 应用程序。

验证

- 点 **File → New → Terminal** 在 JupyterLab 中打开一个终端，再运行以下命令：

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

或者，在新的笔记本单元中输入以下内容并运行单元：

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

如果没有对示例应用程序进行任何更改，您会在浏览器中看到类似 **{"prediction" : "not implemented"}** 的信息。

第 10 章 将项目更改推送到 GIT 存储库

要在生产环境中构建和部署您的应用，请将您的工作上传到远程 Git 存储库。

先决条件

- 您已在 JupyterLab 界面中打开了笔记本。
- 您已将相关的 Git 存储库添加到笔记本服务器中。
- 有权限将更改推送到相关的 Git 存储库。
- 已安装 Git 版本控制扩展。

流程

1. 点 **File → Save All** 保存任何未保存的更改。
2. 点 Git 图标()在 JupyterLab 界面中打开 Git 窗格。
3. 确认更改的文件显示在 **Changed** 下。
   如果您的更改的文件出现在 **Untracked** 中，请点击 **Git → Simple Staging** 以启用简化的 Git 进程。
4. 提交您的更改。
   a. 确保 **Changed** 下的所有文件都有一个蓝色复选标记。
   b. 在 **Summary** 字段中输入您所做的更改的简短描述。
   c. 点 **Commit**。
5. 点 **Git → Push to Remote** 将您的更改推送到远程存储库。
6. 出现提示时，输入您的 Git 凭据，再单击 **OK**。

验证

- 您最近推送的更改在远程 Git 存储库中可见。

第 11 章 为您的预测功能测试部署的应用程序端点

部署应用程序后，您可以测试您的预测功能是否在部署的端点上正常工作。

先决条件

- 您的应用程序是构建和部署的，且包含您的预测功能。
- 您知道包含预测功能的应用程序的 Web 地址。

流程

1. 点 **File → New → Terminal**，在 JupyterLab 中打开一个终端。
2. 运行以下命令，将 **<application-url>** 替换为应用程序的 web 地址，例如 **http://myapp-myproject.apps.mycluster.abc1.s1.devshift.org**。

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

   例如：

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

   或者，在新的笔记本单元中输入 **!** 后接同一命令，并运行单元格。

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

验证

- 如果您从应用程序收到响应，如 **{"predictions" : "not implemented"}**，则端点可以正常工作。