Custom Notebooks

Creating Custom Notebooks

Name: Roey Werner
Consultant.

Date: 18.03.2025

Red Hat

Contents

Red Hat

Introduction:

Custom notebook images are useful if you want to add libraries that you often use, or that you require at a specific version different than the one provided in the base images. It's also useful if you need to use OS packages or applications, which you cannot install on the fly in your running environment.

Image source and Pre-built images

In the opendatahub-io-contrib/workbench-images repository, you will find the source code as well as pre-built images for a lot of use cases. A few of the available images are:

- Base and CUDA-enabled images for different "lines" of OS: UBI8, UBI9, and Centos Stream 9.
- Jupyter images enhanced with:
    - specific libraries like OptaPy or Monai,
    - with integrated applications like Spark,
    - providing other IDEs like VSCode or RStudio
- VSCode
- RStudio

All those images are constantly and automatically updated and rebuilt for the latest patch and fixes, and new releases are available regularly to provide new versions of the libraries or the applications.

Red Hat

Building your own images

In the repository above, you will find many examples from the source code to help you understand how to create your own image. Here are a few rules, tips and examples to help you.

Rules

- On OpenShift, every container in a standard namespace (unless you modify security) runs with a user with a random user id (uid), and the group id (gid) 0. Therefore, all the folders that you want to write in, and all the files you want to modify (temporarily) in your image must be accessible by this user. The best practice is to set the ownership at 1001:0 (user "default", group "0").

```
Unset
RUN chgrp -R /some/directory && \
    chmod -R g=u /some/directory
```

- When launching a notebook from Applications->Enabled, the "personal" volume of a user is mounted at `/opt/app-root/src`. This is not configurable, so make sure to build your images with this default location for the data that you want persisted.

Red Hat

Install Python packages

- Start from a base image of your choice. Normally it's already running under user 1001, so no need to change it.
- Copy your pipfile.lock or your requirements.txt
- Install your packages

Example:

```
Unset
FROM BASE_IMAGE

# Copying custom packages
COPY Pipfile.lock ./

# Install packages and cleanup
# (all commands are chained to minimize layer size)
RUN echo "Installing softwares and packages" && \
    # Install Python packages \
    micropipenv install && \
    rm -f ./Pipfile.lock
    # Fix permissions to support pip in OpenShift environments \
    chmod -R g+w /opt/app-root/lib/python3.9/site-packages && \
    fix-permissions /opt/app-root -P

WORKDIR /opt/app-root/src

ENTRYPOINT ["start-notebook.sh"]
```

In this example, the fix-permissions script (present in all standard images and custom images from the opendatahub-contrib repo) fixes any bad ownership or rights that may be present.

Install an OS package

- If you have to install OS packages and Python packages, it's better to start with the OS.
- In your Containerfile/Dockerfile, switch to user 0, install your package(s), then switch back to user 1001. Example:

```
Unset
USER 0

RUN INSTALL_PKGS="java-11-openjdk java-11-openjdk-devel" && \
yum install -y --setopt=tsflags=nodocs $INSTALL_PKGS && \
yum -y clean all --enablerepo='*'

USER 1001
```

Tips and tricks

Enabling CodeReady Builder (CRB) and EPEL

CRB and EPEL are repositories providing packages absent from a standard RHEL or UBI installation. They are useful and required to be able to install specific software (RStudio, I'm looking at you...).

- Enabling EPEL on UBI9-based images (on UBI9 images CRB is now enabled by default.):

```
Unset
RUN yum install -y https://download.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
```

- Enabling CRB and EPEL on Centos Stream 9-based images:

```
Unset
RUN yum install -y yum-utils && \
  yum-config-manager --enable crb && \
  yum install -y https://download.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
```

Minimizing image size

A container image uses a "layered" filesystem. Every time you have in your file a COPY or a RUN command, a new layer is created. Nothing is ever deleted: removing a file is simply "masking" it in the next layer. Therefore you must be very careful when you create your Containerfile/Dockerfile.

- If you start from an image that is constantly updated, like ubi9/python-39 from the Red Hat Catalog, don't do a `yum update`. This will only fetch new metadata, update a few files that may not have any impact, and get you a bigger image.
- Rebuild your images often from scratch, but don't do a `yum update` on a previous version.
- Group your `RUN` commands as much as you can, add `&& \` at the end of each line to chain your commands.

- If you need to compile something for building an image, use the multi-stage builds approach. Build the library or application in an intermediate container image, then copy the result to your final image. Otherwise, all the build artefacts will persist in your image...