
docker-stacks Documentation

Release latest

Project Jupyter

09 jan. 2022

1	Guia rápido	3
2	CPU Architectures	5
2.1	Caveats for arm64 images	5
3	Índice	7
3.1	Selecting an Image	7
3.2	Running a Container	11
3.3	Common Features	14
3.4	Image Specifics	18
3.5	Contributed Recipes	24
3.6	Project Issues	35
3.7	Package Updates	35
3.8	New Recipes	36
3.9	Doc Translations	36
3.10	Lint	36
3.11	Image Tests	38
3.12	New Features	38
3.13	Community Stacks	39
3.14	Maintainer Playbook	47

Jupyter Docker Stacks é um conjunto de imagens Docker prontas para uso contendo aplicações Jupyter e ferramentas interativas. Você pode usar uma pilha de imagens para fazer qualquer uma dessas coisas (e muito mais):

- Rodar um servidor Jupyter Notebook em um container Docker local
- Rodar um servidor JupyterLab para uma equipe usando JupyterHub
- Escrever seu próprio Dockerfile

You can try a [relatively recent build of the jupyter/base-notebook image on mybinder.org](#) by simply clicking the preceding link. Otherwise, three examples below may help you get started if you [have Docker installed](#), know [which Docker image](#) you want to use and want to launch a single Jupyter Notebook server in a container.

As próximas paginas desta documentação descrevem os usos e as funcionalidades adicionais com mais detalhes

Example 1: This command pulls the `jupyter/scipy-notebook` image tagged `33add21fab64` from Docker Hub if it is not already present on the local host. It then starts a container running a Jupyter Notebook server and exposes the server on host port 8888. The server logs appear in the terminal. Visiting `http://<hostname>:8888/?token=<token>` in a browser loads the Jupyter Notebook dashboard page, where `hostname` is the name of the computer running docker and `token` is the secret token printed in the console. The container remains intact for restart after the notebook server exits.:

```
docker run -p 8888:8888 jupyter/scipy-notebook:33add21fab64
```

Example 2: This command performs the same operations as **Example 1**, but it exposes the server on host port 10000 instead of port 8888. Visiting `http://<hostname>:10000/?token=<token>` in a browser loads Jupyter Notebook server, where `hostname` is the name of the computer running docker and `token` is the secret token printed in the console.:

```
docker run -p 10000:8888 jupyter/scipy-notebook:33add21fab64
```

Example 3: This command pulls the `jupyter/datascience-notebook` image tagged `33add21fab64` from Docker Hub if it is not already present on the local host. It then starts an *ephemeral* container running a Jupyter Notebook server and exposes the server on host port 10000. The command mounts the current working directory on the host as `/home/jovyan/work` in the container. The server logs appear in the terminal. Visiting `http://<hostname>:10000/lab?token=<token>` in a browser loads JupyterLab, where `hostname` is the name of the computer running docker and `token` is the secret token printed in the console. Docker destroys the container after notebook server exit, but any files written to `~/work` in the container remain intact on the host.:

```
docker run --rm -p 10000:8888 -e JUPYTER_ENABLE_LAB=yes -v "${PWD}":/home/jovyan/work \
↳ jupyter/datascience-notebook:33add21fab64
```


All published containers support amd64 (x86_64) and aarch64, except for datascience and tensorflow, which only support amd64 for now.

2.1 Caveats for arm64 images

- The manifests we publish in this projects wiki as well as the image tags for the multi platform images that also support arm, are all based on the amd64 version even though details about the installed packages versions could differ between architectures. For the status about this, see [#1401](#).
- Only the amd64 images are actively tested currently. For the status about this, see [#1402](#).

3.1 Selecting an Image

- *Core Stacks*
- *Image Relationships*
- *Community Stacks*

Using one of the Jupyter Docker Stacks requires two choices:

1. Which Docker image you wish to use
2. How you wish to start Docker containers from that image

This section provides details about the first.

3.1.1 Core Stacks

The Jupyter team maintains a set of Docker image definitions in the <https://github.com/jupyter/docker-stacks> GitHub repository. The following sections describe these images including their contents, relationships, and versioning strategy.

jupyter/base-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

`jupyter/base-notebook` is a small image supporting the *options common across all core stacks*. It is the basis for all other stacks.

- Minimally-functional Jupyter Notebook server (e.g., no LaTeX support for saving notebooks as PDFs)
- [Miniforge](#) Python 3.x in `/opt/conda` with two package managers
 - `conda`: “cross-platform, language-agnostic binary package manager”.

- `mamba`: “reimplementation of the conda package manager in C++”. We use this package manager by default when installing packages.

- `notebook`, `jupyterhub` and `jupyterlab` packages
- No preinstalled scientific computing packages
- Unprivileged user `jovyan` (`uid=1000`, configurable, see options) in group `users` (`gid=100`) with ownership over the `/home/jovyan` and `/opt/conda` paths
- `tini` as the container entrypoint and a `start-notebook.sh` script as the default command
- A `start-singleuser.sh` script useful for launching containers in JupyterHub
- A `start.sh` script useful for running alternative commands in the container (e.g. `ipython`, `jupyter kernelgateway`, `jupyter lab`)
- Options for a self-signed HTTPS certificate and passwordless `sudo`

jupyter/minimal-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

`jupyter/minimal-notebook` adds command line tools useful when working in Jupyter applications.

- Everything in `jupyter/base-notebook`
- [TeX Live](#) for notebook document conversion
- `git`, `vi` (actually `vim-tiny`), `nano` (actually `nano-tiny`), `tzdata`, and `unzip`

jupyter/r-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

`jupyter/r-notebook` includes popular packages from the R ecosystem.

- Everything in `jupyter/minimal-notebook` and its ancestor images
- The R interpreter and base environment
- `IRKernel` to support R code in Jupyter notebooks
- `tidyverse` packages from `conda-forge`
- `caret`, `crayon`, `devtools`, `forecast`, `hexbin`, `htmltools`, `htmlwidgets`, `nycflights13`, `randomforest`, `rcurl`, `rmarkdown`, `rodbc`, `rsqlite`, `shiny`, `tidymodels`, `unixodbc` packages from `conda-forge`

jupyter/scipy-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

`jupyter/scipy-notebook` includes popular packages from the scientific Python ecosystem.

- Everything in `jupyter/minimal-notebook` and its ancestor images
- `altair`, `beautifulsoup4`, `bokeh`, `bottleneck`, `cloudpickle`, `conda-forge::blas*=openblas`, `cython`, `dask`, `dill`, `h5py`, `matplotlib-base`, `numba`, `numexpr`, `pandas`, `patsy`, `protobuf`, `pytables`, `scikit-image`, `scikit-learn`, `scipy`, `seaborn`, `sqlalchemy`, `statsmodel`, `sympy`, `widetsnbextension`, `xlrd` packages
- `ipympl` and `ipywidgets` for interactive visualizations and plots in Python notebooks
- `Facets` for visualizing machine learning datasets

jupyter/tensorflow-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

jupyter/tensorflow-notebook includes popular Python deep learning libraries.

- Everything in jupyter/scipy-notebook and its ancestor images
- tensorflow machine learning library

jupyter/datascience-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

jupyter/datascience-notebook includes libraries for data analysis from the Julia, Python, and R communities.

- Everything in the jupyter/scipy-notebook and jupyter/r-notebook images, and their ancestor images
- rpy2 package
- The Julia compiler and base environment
- IJulia to support Julia code in Jupyter notebooks
- HDF5, Gadfly, RDatasets packages

jupyter/pyspark-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

jupyter/pyspark-notebook includes Python support for Apache Spark.

- Everything in jupyter/scipy-notebook and its ancestor images
- Apache Spark with Hadoop binaries
- pyarrow library

jupyter/all-spark-notebook

[Source on GitHub](#) | [Dockerfile commit history](#) | [Docker Hub image tags](#)

jupyter/all-spark-notebook includes Python, R, and Scala support for Apache Spark.

- Everything in jupyter/pyspark-notebook and its ancestor images
- IRKernel to support R code in Jupyter notebooks
- rcurl, sparklyr, ggplot2 packages
- spylon-kernel to support Scala code in Jupyter notebooks

Image Relationships

The following diagram depicts the build dependency tree of the core images. (i.e., the FROM statements in their Dockerfiles). Any given image inherits the complete content of all ancestor images pointing to it.

Builds

Every Monday and whenever a pull requests is merged, images are rebuilt and pushed to the public container registry.

Versioning via image tags

Whenever a docker image is pushed to the container registry, it is tagged with:

- a latest tag
- a 12-character git commit SHA like `b9f6ce795cfc`
- a date formatted like `2021-08-29`
- a set of software version tags like `python-3.9.6` and `lab-3.0.16`

For stability and reproducibility, you should either reference a date formatted tag from a date before the current date (in UTC time) or a git commit SHA older than the latest git commit SHA in the default branch of the jupyter/docker-stacks GitHub repository.

3.1.2 Community Stacks

The core stacks are just a tiny sample of what's possible when combining Jupyter with other technologies. We encourage members of the Jupyter community to create their own stacks based on the core images and link them below.

- [csharp-notebook](#) is a community Jupyter Docker Stack image. Try [C# in Jupyter Notebooks](#). The image includes more than 200 Jupyter Notebooks with example C# code and can readily be tried online via [mybinder.org](#). Try it on .
- [education-notebook](#) is a community Jupyter Docker Stack image. The image includes nbgrader and RISE on top of the datascience-notebook image. Try it on .
- **jamesdbrock/ihaskell-notebook**
[Source on GitHub](#) | [Dockerfile commit history](#) | [Github container registry](#)
[jamesdbrock/ihaskell-notebook](#) is based on [IHaskell](#). Includes popular packages and example notebooks.
Try it on
- [java-notebook](#) is a community Jupyter Docker Stack image. The image includes [JJava](#) kernel on top of the minimal-notebook image. Try it on .
- [sage-notebook](#) is a community Jupyter Docker Stack image with the [sagemath](#) kernel on top of the minimal-notebook image. Try it on .
- **GPU-Jupyter**: Leverage Jupyter Notebooks with the power of your NVIDIA GPU and perform GPU calculations using Tensorflow and Pytorch in collaborative notebooks. This is done by generating a Dockerfile, that consists of the [nvidia/cuda](#) base image, the well-maintained **docker-stacks** that is integrated as submodule and GPU-able libraries like **Tensorflow**, **Keras** and **PyTorch** on top of it.

- [PRP GPU Jupyter repo](#) and [Registry PRP](#) (Pacific Research Platform) maintained registry for jupyter stack based on NVIDIA CUDA-enabled image. Added the PRP image with Pytorch and some other python packages, and GUI Desktop notebook based on <https://github.com/jupyterhub/jupyter-remote-desktop-proxy>.
- [cgspatial-notebook](#) is a community Jupyter Docker Stack image. The image includes major geospatial Python & R libraries on top of the datascience-notebook image. Try it on
- [kotlin-notebook](#) is a community Jupyter Docker Stack image. The image includes [Kotlin kernel for Jupyter/IPython](#) on top of the base-notebook image. Try it on

See the [contributing guide](#) for information about how to create your own Jupyter Docker Stack.

3.2 Running a Container

Using one of the Jupyter Docker Stacks requires two choices:

1. Which Docker image you wish to use
2. How you wish to start Docker containers from that image

This section provides details about the second.

3.2.1 Using the Docker CLI

You can launch a local Docker container from the Jupyter Docker Stacks using the [Docker command line interface](#). There are numerous ways to configure containers using the CLI. The following are some common patterns.

Example 1 This command pulls the `jupyter/scipy-notebook` image tagged `33add21fab64` from Docker Hub if it is not already present on the local host. It then starts a container running a Jupyter Notebook server and exposes the server on host port 8888. The server logs appear in the terminal and include a URL to the notebook server.

```
$ docker run -p 8888:8888 jupyter/scipy-notebook:33add21fab64

Executing the command: jupyter notebook
[I 15:33:00.567 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.
↳ local/share/jupyter/runtime/notebook_cookie_secret
[W 15:33:01.084 NotebookApp] WARNING: The notebook server is listening on all IP
↳ addresses and not using encryption. This is not recommended.
[I 15:33:01.150 NotebookApp] JupyterLab alpha preview extension loaded from /opt/conda/
↳ lib/python3.6/site-packages/jupyterlab
[I 15:33:01.150 NotebookApp] JupyterLab application directory is /opt/conda/share/
↳ jupyter/lab
[I 15:33:01.155 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 15:33:01.156 NotebookApp] 0 active kernels
[I 15:33:01.156 NotebookApp] The Jupyter Notebook is running at:
[I 15:33:01.157 NotebookApp] http://[all ip addresses on your system]:8888/?
↳ token=112bb073331f1460b73768c76dffb2f87ac1d4ca7870d46a
[I 15:33:01.157 NotebookApp] Use Control-C to stop this server and shut down all kernels.
↳ (twice to skip confirmation).
[C 15:33:01.160 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=112bb073331f1460b73768c76dffb2f87ac1d4ca7870d46a
```

Pressing Ctrl-C shuts down the notebook server but leaves the container intact on disk for later restart or permanent deletion using commands like the following:

```
# list containers
$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED      STATUS      PORTS
d67fe77f1a84        jupyter/base-notebook "tini -- start-noteb..." 44 seconds ago Exited (0) 39 seconds ago
cocky_mirzakhani

# start the stopped container
$ docker start -a d67fe77f1a84
Executing the command: jupyter notebook
[W 16:45:02.020 NotebookApp] WARNING: The notebook server is listening on all IP
addresses and not using encryption. This is not recommended.
...

# remove the stopped container
$ docker rm d67fe77f1a84
d67fe77f1a84
```

Example 2 This command pulls the `jupyter/r-notebook` image tagged `33add21fab64` from Docker Hub if it is not already present on the local host. It then starts a container running a Jupyter Notebook server and exposes the server on host port 10000. The server logs appear in the terminal and include a URL to the notebook server, but with the internal container port (8888) instead of the the correct host port (10000).

```
$ docker run --rm -p 10000:8888 -v "${PWD}":/home/jovyan/work jupyter/r-
notebook:33add21fab64

Executing the command: jupyter notebook
[I 19:31:09.573 NotebookApp] Writing notebook server cookie secret to /home/jovyan/.
local/share/jupyter/runtime/notebook_cookie_secret
[W 19:31:11.930 NotebookApp] WARNING: The notebook server is listening on all IP
addresses and not using encryption. This is not recommended.
[I 19:31:12.085 NotebookApp] JupyterLab alpha preview extension loaded from /opt/conda/
lib/python3.6/site-packages/jupyterlab
[I 19:31:12.086 NotebookApp] JupyterLab application directory is /opt/conda/share/
jupyter/lab
[I 19:31:12.117 NotebookApp] Serving notebooks from local directory: /home/jovyan
[I 19:31:12.117 NotebookApp] 0 active kernels
[I 19:31:12.118 NotebookApp] The Jupyter Notebook is running at:
[I 19:31:12.119 NotebookApp] http://[all ip addresses on your system]:8888/?
token=3b8dce890cb65570fb0d9c4a41ae067f7604873bd604f5ac
[I 19:31:12.120 NotebookApp] Use Control-C to stop this server and shut down all kernels.
(twice to skip confirmation).
[C 19:31:12.122 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=3b8dce890cb65570fb0d9c4a41ae067f7604873bd604f5ac
```

Pressing Ctrl-C shuts down the notebook server and immediately destroys the Docker container. Files written to `~/work` in the container remain touched. Any other changes made in the container are lost.

Example 3 This command pulls the `jupyter/all-spark-notebook` image currently tagged `latest` from Docker

Hub if an image tagged `latest` is not already present on the local host. It then starts a container named `notebook` running a JupyterLab server and exposes the server on a randomly selected port.

```
docker run -d -P --name notebook jupyter/all-spark-notebook
```

The assigned port and notebook server token are visible using other Docker commands.

```
# get the random host port assigned to the container port 8888
$ docker port notebook 8888
0.0.0.0:32769

# get the notebook token from the logs
$ docker logs --tail 3 notebook
Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
http://localhost:8888/?token=15914ca95f495075c0aa7d0e060f1a78b6d94f70ea373b00
```

Together, the URL to visit on the host machine to access the server in this case is <http://localhost:32769?token=15914ca95f495075c0aa7d0e060f1a78b6d94f70ea373b00>.

The container runs in the background until stopped and/or removed by additional Docker commands.

```
# stop the container
docker stop notebook
notebook

# remove the container permanently
docker rm notebook
notebook
```

3.2.2 Using Binder

[Binder](#) is a service that allows you to create and share custom computing environments for projects in version control. You can use any of the Jupyter Docker Stacks images as a basis for a Binder-compatible Dockerfile. See the [docker-stacks example](#) and [Using a Dockerfile](#) sections in the [Binder documentation](#) for instructions.

3.2.3 Using JupyterHub

You can configure JupyterHub to launcher Docker containers from the Jupyter Docker Stacks images. If you've been following the [Zero to JupyterHub with Kubernetes](#) guide, see the [Use an existing Docker image](#) section for details. If you have a custom JupyterHub deployment, see the [Picking or building a Docker image](#) instructions for the `dockerspawner` instead.

3.2.4 Using Other Tools and Services

You can use the Jupyter Docker Stacks with any Docker-compatible technology (e.g., [Docker Compose](#), [docker-py](#), your favorite cloud container service). See the documentation of the tool, library, or service for details about how to reference, configure, and launch containers from these images.

3.3 Common Features

A container launched from any Jupyter Docker Stacks image runs a Jupyter Notebook server by default. The container does so by executing a `start-notebook.sh` script. This script configures the internal container environment and then runs `jupyter notebook`, passing it any command line arguments received.

This page describes the options supported by the startup script as well as how to bypass it to run alternative commands.

3.3.1 Notebook Options

You can pass [Jupyter command line options](#) to the `start-notebook.sh` script when launching the container. For example, to secure the Notebook server with a custom password hashed using `IPython.lib.passwd()` instead of the default token, you can run the following:

```
docker run -d -p 8888:8888 jupyter/base-notebook start-notebook.sh --NotebookApp.  
↪password='sha1:74ba40f8a388:c913541b7ee99d15d5ed31d4226bf7838f83a50e'
```

For example, to set the base URL of the notebook server, you can run the following:

```
docker run -d -p 8888:8888 jupyter/base-notebook start-notebook.sh --NotebookApp.base_  
↪url=/some/path
```

3.3.2 Docker Options

You may instruct the `start-notebook.sh` script to customize the container environment before launching the notebook server. You do so by passing arguments to the `docker run` command.

- `-e NB_USER=jovyan` - Instructs the startup script to change the default container username from `jovyan` to the provided value. Causes the script to rename the `jovyan` user home folder. For this option to take effect, you must run the container with `--user root`, set the working directory `-w /home/${NB_USER}` and set the environment variable `-e CHOWN_HOME=yes` (see below for detail). This feature is useful when mounting host volumes with specific home folder.
- `-e NB_UID=1000` - Instructs the startup script to switch the numeric user ID of `${NB_USER}` to the given value. This feature is useful when mounting host volumes with specific owner permissions. For this option to take effect, you must run the container with `--user root`. (The startup script will `su ${NB_USER}` after adjusting the user ID.) You might consider using modern Docker options `--user` and `--group-add` instead. See the last bullet below for details.

- `-e NB_GID=100` - Instructs the startup script to change the primary group of `${NB_USER}` to `${NB_GID}` (the new group is added with a name of `${NB_GROUP}` if it is defined, otherwise the group is named `${NB_USER}`). This feature is useful when mounting host volumes with specific group permissions. For this option to take effect, you must run the container with `--user root`. (The startup script will `su ${NB_USER}` after adjusting the group ID.) You might consider using modern Docker options `--user` and `--group-add` instead. See the last bullet below for details. The user is added to supplemental group users (gid 100) in order to allow write access to the home directory and `/opt/conda`. If you override the user/group logic, ensure the user stays in group users if you want them to be able to modify files in the image.
- `-e NB_GROUP=<name>` - The name used for `${NB_GID}`, which defaults to `${NB_USER}`. This is only used if `${NB_GID}` is specified and completely optional: there is only cosmetic effect.
- `-e NB_UMASK=<umask>` - Configures Jupyter to use a different umask value from default, i.e. `022`. For example, if setting umask to `002`, new files will be readable and writable by group members instead of just writable by the owner. Wikipedia has a good article about [umask](#). Feel free to read it in order to choose the value that better fits your needs. Default value should fit most situations. Note that `NB_UMASK` when set only applies to the Jupyter process itself - you cannot use it to set a umask for additional files created during run-hooks e.g. via `pip` or `conda` - if you need to set a umask for these you must set `umask` for each command.
- `-e CHOWN_HOME=yes` - Instructs the startup script to change the `${NB_USER}` home directory owner and group to the current value of `${NB_UID}` and `${NB_GID}`. This change will take effect even if the user home directory is mounted from the host using `-v` as described below. The change is **not** applied recursively by default. You can change modify the `chown` behavior by setting `CHOWN_HOME_OPTS` (e.g., `-e CHOWN_HOME_OPTS='-R'`).
- `-e CHOWN_EXTRA="<some dir>,<some other dir>"` - Instructs the startup script to change the owner and group of each comma-separated container directory to the current value of `${NB_UID}` and `${NB_GID}`. The change is **not** applied recursively by default. You can change modify the `chown` behavior by setting `CHOWN_EXTRA_OPTS` (e.g., `-e CHOWN_EXTRA_OPTS='-R'`).
- `-e GRANT_SUDO=yes` - Instructs the startup script to grant the `NB_USER` user passwordless `sudo` capability. You do **not** need this option to allow the user to `conda` or `pip` install additional packages. This option is useful, however, when you wish to give `${NB_USER}` the ability to install OS packages with `apt` or modify other root-owned files in the container. For this option to take effect, you must run the container with `--user root`. (The `start-notebook.sh` script will `su ${NB_USER}` after adding `${NB_USER}` to `sudoers`.) **You should only enable sudo if you trust the user or if the container is running on an isolated host.**
- `-e GEN_CERT=yes` - Instructs the startup script to generates a self-signed SSL certificate and configure Jupyter Notebook to use it to accept encrypted HTTPS connections.
- `-e JUPYTER_ENABLE_LAB=yes` - Instructs the startup script to run `jupyter lab` instead of the default `jupyter notebook` command. Useful in container orchestration environments where setting environment variables is easier than change command line parameters.
- `-e RESTARTABLE=yes` - Runs Jupyter in a loop so that quitting Jupyter does not cause the container to exit. This may be useful when you need to install extensions that require restarting Jupyter.
- `-v /some/host/folder/for/work:/home/jovyan/work` - Mounts a host machine directory as folder in the container. Useful when you want to preserve notebooks and other work even after the container is destroyed. **You must grant the within-container notebook user or group (`NB_UID` or `NB_GID`) write access to the host directory (e.g., `sudo chown 1000 /some/host/folder/for/work`).**
- `--user 5000 --group-add users` - Launches the container with a specific user ID and adds that user to the `users` group so that it can modify files in the default home directory and `/opt/conda`. You can use these arguments as alternatives to setting `${NB_UID}` and `${NB_GID}`.
- `-e JUPYTER_ENV_VARS_TO_UNSET=ADMIN_SECRET_1,ADMIN_SECRET_2` - Unsets specified environment variables in the default startup script. The variables are unset after the hooks have executed but before the command provided to the startup script runs.

- `-e NOTEBOOK_ARGS="--log-level='DEBUG' --dev-mode"` - Adds custom options to launch `jupyter lab` or `jupyter notebook`. This way any option, supported by `jupyter` could be used by the user.

3.3.3 Startup Hooks

You can further customize the container environment by adding shell scripts (`*.sh`) to be sourced or executables (`chmod +x`) to be run to the paths below:

- `/usr/local/bin/start-notebook.d/` - handled before any of the standard options noted above are applied
- `/usr/local/bin/before-notebook.d/` - handled after all of the standard options noted above are applied and just before the notebook server launches

See the `run-hooks` function in the `jupyter/base-notebook start.sh` script for execution details.

3.3.4 SSL Certificates

You may mount SSL key and certificate files into a container and configure Jupyter Notebook to use them to accept HTTPS connections. For example, to mount a host folder containing a `notebook.key` and `notebook.crt` and use them, you might run the following:

```
docker run -d -p 8888:8888 \
  -v /some/host/folder:/etc/ssl/notebook \
  jupyter/base-notebook start-notebook.sh \
  --NotebookApp.keyfile=/etc/ssl/notebook/notebook.key
  --NotebookApp.certfile=/etc/ssl/notebook/notebook.crt
```

Alternatively, you may mount a single PEM file containing both the key and certificate. For example:

```
docker run -d -p 8888:8888 \
  -v /some/host/folder/notebook.pem:/etc/ssl/notebook.pem \
  jupyter/base-notebook start-notebook.sh \
  --NotebookApp.certfile=/etc/ssl/notebook.pem
```

In either case, Jupyter Notebook expects the key and certificate to be a base64 encoded text file. The certificate file or PEM may contain one or more certificates (e.g., server, intermediate, and root).

For additional information about using SSL, see the following:

- The [docker-stacks/examples](#) for information about how to use [Let's Encrypt](#) certificates when you run these stacks on a publicly visible domain.
- The `jupyter_notebook_config.py` file for how this Docker image generates a self-signed certificate.
- The [Jupyter Notebook documentation](#) for best practices about securing a public notebook server in general.

3.3.5 Alternative Commands

start.sh

The `start-notebook.sh` script actually inherits most of its option handling capability from a more generic `start.sh` script. The `start.sh` script supports all of the features described above, but allows you to specify an arbitrary command to execute. For example, to run the text-based `ipython` console in a container, do the following:

```
docker run -it --rm jupyter/base-notebook start.sh ipython
```

Or, to run JupyterLab instead of the classic notebook, run the following:

```
docker run -it --rm -p 8888:8888 jupyter/base-notebook start.sh jupyter lab
```

This script is particularly useful when you derive a new Dockerfile from this image and install additional Jupyter applications with subcommands like `jupyter console`, `jupyter kernelgateway`, etc.

Others

You can bypass the provided scripts and specify an arbitrary start command. If you do, keep in mind that features supported by the `start.sh` script and its kin will not function (e.g., `GRANT_SUDO`).

3.3.6 Conda Environments

The default Python 3.x [Conda environment](#) resides in `/opt/conda`. The `/opt/conda/bin` directory is part of the default jovyan user's `PATH`. That directory is also whitelisted for use in `sudo` commands by the `start.sh` script.

The jovyan user has full read/write access to the `/opt/conda` directory. You can use either `pip`, `conda` or `mamba` to install new packages without any additional permissions.

```
# install a package into the default (python 3.x) environment and cleanup after the
↳ installation
mamba install --quiet --yes some-package && \
    mamba clean --all -f -y && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"

pip install --quiet --no-cache-dir some-package && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"

conda install --quiet --yes some-package && \
    conda clean --all -f -y && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"
```

Using alternative channels

Conda is configured by default to use only the `conda-forge` channel. However, alternative channels can be used either one shot by overwriting the default channel in the installation command or by configuring mamba to use different channels. The examples below show how to use the `anaconda default channels` instead of `conda-forge` to install packages.

```
# using defaults channels to install a package
mamba install --channel defaults humanize
# configure conda to add default channels at the top of the list
conda config --system --prepend channels defaults
# install a package
mamba install --quiet --yes humanize && \
    mamba clean --all -f -y && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"
```

3.4 Image Specifics

This page provides details about features specific to one or more images.

3.4.1 Apache Spark™

Specific Docker Image Options

- `-p 4040:4040` - The `jupyter/pyspark-notebook` and `jupyter/all-spark-notebook` images open `SparkUI` (`Spark Monitoring and Instrumentation UI`) at default port `4040`, this option map `4040` port inside docker container to `4040` port on host machine. Note every new spark context that is created is put onto an incrementing port (ie. `4040`, `4041`, `4042`, etc.), and it might be necessary to open multiple ports. For example: `docker run -d -p 8888:8888 -p 4040:4040 -p 4041:4041 jupyter/pyspark-notebook`.

IPython low-level output capture and forward

Spark images (`pyspark-notebook` and `all-spark-notebook`) have been configured to disable IPython low-level output capture and forward system-wide. The rationale behind this choice is that Spark logs can be verbose, especially at startup when Ivy is used to load additional jars. Those logs are still available but only in the container's logs.

If you want to make them appear in the notebook, you can overwrite the configuration in a user level IPython kernel profile. To do that you have to uncomment the following line in your `~/.ipython/profile_default/ipython_kernel_config.py` and restart the kernel.

```
c. IPKernelApp.capture_fd_output = True
```

If you have no IPython profile you can initiate a fresh one by running the following command.

```
ipython profile create
# [ProfileCreate] Generating default config file: '/home/jovyan/.ipython/profile_default/
↪ipython_config.py'
# [ProfileCreate] Generating default config file: '/home/jovyan/.ipython/profile_default/
↪ipython_kernel_config.py'
```

- Spark distribution is defined by the combination of the Spark and the Hadoop version and verified by the package checksum, see [Download Apache Spark](#) and the [archive repo](#) for more information.
 - `spark_version`: The Spark version to install (3.0.0).
 - `hadoop_version`: The Hadoop version (3.2).
 - `spark_checksum`: The package checksum (BFE4540...).
- Spark can run with different OpenJDK versions.
 - `openjdk_version`: The version of (JRE headless) the OpenJDK distribution (11), see [Ubuntu packages](#).

```
# From the root of the project
# Build the image with different arguments
docker build --rm --force-rm \
    -t jupyter/pyspark-notebook:spark-2.4.7 ./pyspark-notebook \
    --build-arg spark_version=2.4.7 \
    --build-arg hadoop_version=2.7 \
    --build-arg spark_
checksum=0F5455672045F6110B030CE343C049855B7BA86C0ECB5E39A075FF9D093C7F648DA55DED12E72FFE65D84C32DCD5
\
    --build-arg openjdk_version=8

# Check the newly built image
docker run -it --rm jupyter/pyspark-notebook:spark-2.4.7 pyspark --version

# Welcome to
#
#      ____          _
#     /  __/___   ___  _____/  __/
#    _\  \/_  _\_/_  _\  ___/_\  \
#   /____/ .__/\_,_/_/_/_/_/_/_  version 2.4.7
#       /\
#
# Using Scala version 2.11.12, OpenJDK 64-Bit Server VM, 1.8.0_275
```

Usage Examples

The `jupyter/pyspark-notebook` and `jupyter/all-spark-notebook` images support the use of [Apache Spark](#) in Python, R, and Scala notebooks. The following sections provide some examples of how to get started using them.

Using Spark Local Mode

Spark **local mode** is useful for experimentation on small data when you do not have a Spark cluster available.

Local Mode in Python

In a Python notebook.

```
from pyspark.sql import SparkSession

# Spark session & context
spark = SparkSession.builder.master('local').getOrCreate()
sc = spark.sparkContext

# Sum of the first 100 whole numbers
rdd = sc.parallelize(range(100 + 1))
rdd.sum()
# 5050
```

Local Mode in R

In a R notebook with `SparkR`.

```
library(SparkR)

# Spark session & context
sc <- sparkR.session("local")

# Sum of the first 100 whole numbers
sdf <- createDataFrame(list(1:100))
dapplyCollect(sdf,
              function(x)
                { x <- sum(x) }
              )
# 5050
```

In a R notebook with `sparklyr`.

```
library(sparklyr)

# Spark configuration
conf <- spark_config()
# Set the catalog implementation in-memory
conf$spark.sql.catalogImplementation <- "in-memory"
```

(continua na próxima página)

(continuação da página anterior)

```
# Spark session & context
sc <- spark_connect(master = "local", config = conf)

# Sum of the first 100 whole numbers
sdf_len(sc, 100, repartition = 1) %>%
  spark_apply(function(e) sum(e))
# 5050
```

Local Mode in Scala

Spylon kernel instantiates a SparkContext for you in variable `sc` after you configure Spark options in a `%%init_spark` magic cell.

```
%%init_spark
# Configure Spark to use a local master
launcher.master = "local"
```

```
// Sum of the first 100 whole numbers
val rdd = sc.parallelize(0 to 100)
rdd.sum()
// 5050
```

Connecting to a Spark Cluster in Standalone Mode

Connection to Spark Cluster on **Standalone Mode** requires the following set of steps:

1. Verify that the docker image (check the Dockerfile) and the Spark Cluster which is being deployed, run the same version of Spark.
2. Deploy Spark in Standalone Mode.
3. Run the Docker container with `--net=host` in a location that is network addressable by all of your Spark workers. (This is a Spark networking requirement.)
 - NOTE: When using `--net=host`, you must also use the flags `--pid=host -e TINI_SUBREAPER=true`. See <https://github.com/jupyter/docker-stacks/issues/64> for details.

Note: In the following examples we are using the Spark master URL `spark://master:7077` that shall be replaced by the URL of the Spark master.

Standalone Mode in Python

The **same Python version** needs to be used on the notebook (where the driver is located) and on the Spark workers. The python version used at driver and worker side can be adjusted by setting the environment variables `PYSPARK_PYTHON` and / or `PYSPARK_DRIVER_PYTHON`, see [Spark Configuration](#) for more information.

```
from pyspark.sql import SparkSession

# Spark session & context
spark = SparkSession.builder.master('spark://master:7077').getOrCreate()
```

(continua na próxima página)

(continuação da página anterior)

```
sc = spark.sparkContext

# Sum of the first 100 whole numbers
rdd = sc.parallelize(range(100 + 1))
rdd.sum()
# 5050
```

Standalone Mode in R

In a R notebook with SparkR.

```
library(SparkR)

# Spark session & context
sc <- sparkR.session("spark://master:7077")

# Sum of the first 100 whole numbers
sdf <- createDataFrame(list(1:100))
dapplyCollect(sdf,
               function(x)
                 { x <- sum(x) }
               )
# 5050
```

In a R notebook with sparklyr.

```
library(sparklyr)

# Spark session & context
# Spark configuration
conf <- spark_config()
# Set the catalog implementation in-memory
conf$spark.sql.catalogImplementation <- "in-memory"
sc <- spark_connect(master = "spark://master:7077", config = conf)

# Sum of the first 100 whole numbers
sdf_len(sc, 100, repartition = 1) %>%
  spark_apply(function(e) sum(e))
# 5050
```

Standalone Mode in Scala

Spylon kernel instantiates a SparkContext for you in variable `sc` after you configure Spark options in a `%%init_spark` magic cell.

```
%%init_spark
# Configure Spark to use a local master
launcher.master = "spark://master:7077"
```

```
// Sum of the first 100 whole numbers
val rdd = sc.parallelize(0 to 100)
rdd.sum()
// 5050
```

Define Spark Dependencies

Spark dependencies can be declared thanks to the `spark.jars.packages` property (see [Spark Configuration](#) for more information).

They can be defined as a comma-separated list of Maven coordinates at the creation of the Spark session.

```
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder.appName("elasticsearch")
    .config(
        "spark.jars.packages",
        "org.elasticsearch:elasticsearch-spark-30_2.12:7.13.0"
    )
    .getOrCreate()
)
```

Dependencies can also be defined in the `spark-defaults.conf`. However, it has to be done by `root` so it should only be considered to build custom images.

```
USER root
RUN echo "spark.jars.packages org.elasticsearch:elasticsearch-spark-30_2.12:7.13.0" >> "$
↳ {SPARK_HOME}/conf/spark-defaults.conf"
USER ${NB_UID}
```

Jars will be downloaded dynamically at the creation of the Spark session and stored by default in `${HOME}/.ivy2/jars` (can be changed by setting `spark.jars.ivy`).

Note: This example is given for [Elasticsearch](#).

3.4.2 Tensorflow

The `jupyter/tensorflow-notebook` image supports the use of [Tensorflow](#) in single machine or distributed mode.

Single Machine Mode

```
import tensorflow as tf

hello = tf.Variable('Hello World!')

sess = tf.Session()
init = tf.global_variables_initializer()

sess.run(init)
sess.run(hello)
```

Distributed Mode

```
import tensorflow as tf

hello = tf.Variable('Hello Distributed World!')

server = tf.train.Server.create_local_server()
sess = tf.Session(server.target)
init = tf.global_variables_initializer()

sess.run(init)
sess.run(hello)
```

3.5 Contributed Recipes

Users sometimes share interesting ways of using the Jupyter Docker Stacks. We encourage users to *contribute these recipes* to the documentation in case they prove useful to other members of the community by submitting a pull request to docs/using/recipes.md. The sections below capture this knowledge.

3.5.1 Using sudo within a container

Password authentication is disabled for the NB_USER (e.g., jovyan). This choice was made to avoid distributing images with a weak default password that users ~might~ will forget to change before running a container on a publicly accessible host.

You can grant the within-container NB_USER passwordless sudo access by adding `-e GRANT_SUDO=yes` and `--user root` to your Docker command line or appropriate container orchestrator config.

For example:

```
docker run -it -e GRANT_SUDO=yes --user root jupyter/minimal-notebook
```

You should only enable sudo if you trust the user and/or if the container is running on an isolated host. See [Docker security documentation](#) for more information about running containers as root.

3.5.2 Using mamba install or pip install in a Child Docker image

Create a new Dockerfile like the one shown below.

```
# Start from a core stack version
FROM jupyter/datascience-notebook:33add21fab64
# Install in the default python3 environment
RUN pip install --quiet --no-cache-dir 'flake8==3.9.2' && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"
```

Then build a new image.

```
docker build --rm -t jupyter/my-datascience-notebook .
```

To use a requirements.txt file, first create your requirements.txt file with the listing of packages desired. Next, create a new Dockerfile like the one shown below.

```
# Start from a core stack version
FROM jupyter/datascience-notebook:33add21fab64
# Install from requirements.txt file
COPY --chown=${NB_UID}:${NB_GID} requirements.txt /tmp/
RUN pip install --quiet --no-cache-dir --requirement /tmp/requirements.txt && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"
```

For conda, the Dockerfile is similar:

```
# Start from a core stack version
FROM jupyter/datascience-notebook:33add21fab64
# Install from requirements.txt file
COPY --chown=${NB_UID}:${NB_GID} requirements.txt /tmp/
RUN mamba install --yes --file /tmp/requirements.txt && \
    mamba clean --all -f -y && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"
```

Ref: [docker-stacks/commit/79169618d571506304934a7b29039085e77db78c](https://github.com/jupyter/docker-stacks/commit/79169618d571506304934a7b29039085e77db78c)

3.5.3 Add a Python 2.x environment

Python 2.x was removed from all images on August 10th, 2017, starting in tag cc9feab481f7. You can add a Python 2.x environment by defining your own Dockerfile inheriting from one of the images like so:

```
# Choose your desired base image
FROM jupyter/scipy-notebook:latest

# Create a Python 2.x environment using conda including at least the ipython kernel
# and the kernda utility. Add any additional packages you want available for use
# in a Python 2 notebook to the first line here (e.g., pandas, matplotlib, etc.)
RUN mamba create --quiet --yes -p "${CONDA_DIR}/envs/python2" python=2.7 ipython_
↪ ipykernel kernda && \
    mamba clean --all -f -y

USER root

# Create a global kernelspec in the image and modify it so that it properly activates
# the python2 conda environment.
RUN "${CONDA_DIR}/envs/python2/bin/python" -m ipykernel install && \
    "${CONDA_DIR}/envs/python2/bin/kernda" -o -y /usr/local/share/jupyter/kernels/
↪ python2/kernel.json

USER ${NB_UID}
```

Ref: <https://github.com/jupyter/docker-stacks/issues/440>

3.5.4 Add a Python 3.x environment

The default version of Python that ships with conda/ubuntu may not be the version you want. To add a conda environment with a different version and make it accessible to Jupyter, the instructions are very similar to Python 2.x but are slightly simpler (no need to switch to root):

```
# Choose your desired base image
FROM jupyter/minimal-notebook:latest

# name your environment and choose python 3.x version
ARG conda_env=python36
ARG py_ver=3.6

# you can add additional libraries you want mamba to install by listing them below the
↪ first line and ending with "&& \"
RUN mamba create --quiet --yes -p "${CONDA_DIR}/envs/${conda_env}" python=${py_ver} \
↪ ipython ipykernel && \
    mamba clean --all -f -y

# alternatively, you can comment out the lines above and uncomment those below
# if you'd prefer to use a YAML file present in the docker build context

# COPY --chown=${NB_UID}:${NB_GID} environment.yml "/home/${NB_USER}/tmp/"
# RUN cd "/home/${NB_USER}/tmp/" && \
#     mamba env create -p "${CONDA_DIR}/envs/${conda_env}" -f environment.yml && \
#     mamba clean --all -f -y

# create Python 3.x environment and link it to jupyter
RUN "${CONDA_DIR}/envs/${conda_env}/bin/python" -m ipykernel install --user --name="$
↪ {conda_env}" && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"

# any additional pip installs can be added by uncommenting the following line
# RUN "${CONDA_DIR}/envs/${conda_env}/bin/pip" install

# prepend conda environment to path
ENV PATH "${CONDA_DIR}/envs/${conda_env}/bin:${PATH}"

# if you want this environment to be the default one, uncomment the following line:
# ENV CONDA_DEFAULT_ENV ${conda_env}
```

3.5.5 Run JupyterLab

JupyterLab is preinstalled as a notebook extension starting in tag `c33a7dc0eece`.

Run jupyterlab using a command such as `docker run -it --rm -p 8888:8888 -e JUPYTER_ENABLE_LAB=yes jupyter/datascience-notebook`

3.5.6 Dask JupyterLab Extension

Dask JupyterLab Extension provides a JupyterLab extension to manage Dask clusters, as well as embed Dask's dashboard plots directly into JupyterLab panes. Create the Dockerfile as:

```
# Start from a core stack version
FROM jupyter/scipy-notebook:latest

# Install the Dask dashboard
RUN pip install --quiet --no-cache-dir dask-labextension && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"

# Dask Scheduler & Bokeh ports
EXPOSE 8787
EXPOSE 8786

ENTRYPOINT ["jupyter", "lab", "--ip=0.0.0.0", "--allow-root"]
```

And build the image as:

```
docker build -t jupyter/scipy-dasklabextension:latest .
```

Once built, run using the command:

```
docker run -it --rm -p 8888:8888 -p 8787:8787 jupyter/scipy-dasklabextension:latest
```

Ref: <https://github.com/jupyter/docker-stacks/issues/999>

3.5.7 Let's Encrypt a Notebook server

See the README for the simple automation here <https://github.com/jupyter/docker-stacks/tree/master/examples/make-deploy> which includes steps for requesting and renewing a Let's Encrypt certificate.

Ref: <https://github.com/jupyter/docker-stacks/issues/78>

3.5.8 Slideshows with Jupyter and RISE

RISE allows via extension to create live slideshows of your notebooks, with no conversion, adding javascript Reveal.js:

```
# Add Live slideshows with RISE
RUN mamba install --quiet --yes -c damianavila82 rise && \
    mamba clean --all -f -y && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"
```

Credit: Paolo D. based on [docker-stacks/issues/43](https://github.com/jupyter/docker-stacks/issues/43)

3.5.9 xgboost

You need to install conda-forge's gcc for Python xgboost to work properly. Otherwise, you'll get an exception about libgomp.so.1 missing GOMP_4.0.

```
mamba install --quiet --yes gcc && \
  mamba clean --all -f -y && \
  fix-permissions "${CONDA_DIR}" && \
  fix-permissions "/home/${NB_USER}"

pip install --quiet --no-cache-dir xgboost && \
  fix-permissions "${CONDA_DIR}" && \
  fix-permissions "/home/${NB_USER}"

# run "import xgboost" in python
```

3.5.10 Running behind a nginx proxy

Sometimes it is useful to run the Jupyter instance behind a nginx proxy, for instance:

- you would prefer to access the notebook at a server URL with a path (<https://example.com/jupyter>) rather than a port (<https://example.com:8888>)
- you may have many different services in addition to Jupyter running on the same server, and want to nginx to help improve server performance in manage the connections

Here is a [quick example NGINX configuration](#) to get started. You'll need a server, a .crt and .key file for your server, and docker & docker-compose installed. Then just download the files at that gist and run `docker-compose up -d` to test it out. Customize the `nginx.conf` file to set the desired paths and add other services.

3.5.11 Host volume mounts and notebook errors

If you are mounting a host directory as `/home/jovyan/work` in your container and you receive permission errors or connection errors when you create a notebook, be sure that the `jovyan` user (UID=1000 by default) has read/write access to the directory on the host. Alternatively, specify the UID of the `jovyan` user on container startup using the `-e NB_UID` option described in the [Common Features, Docker Options](#) section

Ref: <https://github.com/jupyter/docker-stacks/issues/199>

3.5.12 Manpage installation

Most containers, including our Ubuntu base image, ship without manpages installed to save space. You can use the following dockerfile to inherit from one of our images to enable manpages:

```
# Choose your desired base image
ARG BASE_CONTAINER=jupyter/datascience-notebook:latest
FROM $BASE_CONTAINER

USER root

# `/etc/dpkg/dpkg.cfg.d/excludes` contains several `path-exclude`s, including man pages
# Remove it, then install man, install docs
```

(continua na próxima página)

(continuação da página anterior)

```

RUN rm /etc/dpkg/dpkg.cfg.d/excludes && \
    apt-get update --yes && \
    dpkg -l | grep ^ii | cut -d' ' -f3 | xargs apt-get install --yes --no-install-
    ↳ recommends --reinstall man && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

USER ${NB_UID}

```

Adding the documentation on top of an existing singleuser image wastes a lot of space and requires reinstalling every system package, which can take additional time and bandwidth; the `datascience-notebook` image has been shown to grow by almost 3GB when adding manpages in this way. Enabling manpages in the base Ubuntu layer prevents this container bloat. Just use previous Dockerfile with original ubuntu image as your base container:

```

# Ubuntu 20.04 (focal) from 2020-04-23
# https://github.com/docker-library/official-images/commit/
    ↳ 4475094895093bcc29055409494cce1e11b52f94
ARG BASE_CONTAINER=ubuntu:focal-
    ↳ 20200423@sha256:238e696992ba9913d24cfc3727034985abd136e08ee3067982401acdc30cbf3f

```

For Ubuntu 18.04 (bionic) and earlier, you may also require to workaround for a mandb bug, which was fixed in mandb >= 2.8.6.1:

```

# https://git.savannah.gnu.org/cgit/man-db.git/commit/?
    ↳ id=8197d7824f814c5d4b992b4c8730b5b0f7ec589a
# https://launchpadlibrarian.net/435841763/man-db_2.8.5-2_2.8.6-1.diff.gz

RUN echo "MANPATH_MAP ${CONDA_DIR}/bin ${CONDA_DIR}/man" >> /etc/manpath.config && \
    echo "MANPATH_MAP ${CONDA_DIR}/bin ${CONDA_DIR}/share/man" >> /etc/manpath.config && \
    ↳ \
    mandb

```

Be sure to check the current base image in `base-notebook` before building.

3.5.13 JupyterHub

We also have contributed recipes for using JupyterHub.

Use JupyterHub's dockerspawner

In most cases for use with DockerSpawner, given any image that already has a notebook stack set up, you would only need to add:

1. install the `jupyterhub-singleuser` script (for the right Python)
2. change the command to launch the single-user server

Swapping out the FROM line in the `jupyterhub/singleuser` Dockerfile should be enough for most cases.

Credit: Justin Tyberg, quanghoc, and Min RK based on [docker-stacks/issues/124](#) and [docker-stacks/pull/185](#)

Containers with a specific version of JupyterHub

To use a specific version of JupyterHub, the version of `jupyterhub` in your image should match the version in the Hub itself.

```
FROM jupyter/base-notebook:33add21fab64
RUN pip install --quiet --no-cache-dir jupyterhub==1.4.1 && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"
```

Credit: MinRK

Ref: <https://github.com/jupyter/docker-stacks/issues/177>

3.5.14 Spark

A few suggestions have been made regarding using Docker Stacks with spark.

Using PySpark with AWS S3

Using Spark session for hadoop 2.7.3

```
import os
# !ls /usr/local/spark/jars/hadoop* # to figure out what version of hadoop
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages org.apache.hadoop:hadoop-aws:2.7.3 \
↳pyspark-shell'

import pyspark
myAccessKey = input()
mySecretKey = input()

spark = pyspark.sql.SparkSession.builder \
    .master("local[*]") \
    .config("spark.hadoop.fs.s3a.access.key", myAccessKey) \
    .config("spark.hadoop.fs.s3a.secret.key", mySecretKey) \
    .getOrCreate()

df = spark.read.parquet("s3://myBucket/myKey")
```

Using Spark context for hadoop 2.6.0

```
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages com.amazonaws:aws-java-sdk:1.10.34,org.
↳apache.hadoop:hadoop-aws:2.6.0 pyspark-shell'

import pyspark
sc = pyspark.SparkContext("local[*]")

from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

hadoopConf = sc._jsc.hadoopConfiguration()
myAccessKey = input()
```

(continua na próxima página)

(continuação da página anterior)

```
mySecretKey = input()
hadoopConf.set("fs.s3.impl", "org.apache.hadoop.fs.s3native.NativeS3FileSystem")
hadoopConf.set("fs.s3.awsAccessKeyId", myAccessKey)
hadoopConf.set("fs.s3.awsSecretAccessKey", mySecretKey)

df = sqlContext.read.parquet("s3://myBucket/myKey")
```

Ref: <https://github.com/jupyter/docker-stacks/issues/127>

Using Local Spark JARs

```
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--jars /home/jovyan/spark-streaming-kafka-assembly_
↳ 2.10-1.6.1.jar pyspark-shell'
import pyspark
from pyspark.streaming.kafka import KafkaUtils
from pyspark.streaming import StreamingContext
sc = pyspark.SparkContext()
ssc = StreamingContext(sc,1)
broker = "<my_broker_ip>"
directKafkaStream = KafkaUtils.createDirectStream(ssc, ["test1"], {"metadata.broker.list
↳ ": broker})
directKafkaStream.pprint()
ssc.start()
```

Ref: <https://github.com/jupyter/docker-stacks/issues/154>

Using spark-packages.org

If you'd like to use packages from spark-packages.org, see <https://gist.github.com/parente/c95fdaba5a9a066efaab> for an example of how to specify the package identifier in the environment before creating a SparkContext.

Ref: <https://github.com/jupyter/docker-stacks/issues/43>

Use jupyter/all-spark-notebooks with an existing Spark/YARN cluster

```
FROM jupyter/all-spark-notebook

# Set env vars for pydoop
ENV HADOOP_HOME /usr/local/hadoop-2.7.3
ENV JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64
ENV HADOOP_CONF_HOME /usr/local/hadoop-2.7.3/etc/hadoop
ENV HADOOP_CONF_DIR /usr/local/hadoop-2.7.3/etc/hadoop

USER root
# Add proper open-jdk-8 not just the jre, needed for pydoop
RUN echo 'deb https://cdn-fastly.deb.debian.org/debian jessie-backports main' > /etc/apt/
↳ sources.list.d/jessie-backports.list && \
    apt-get update --yes && \
    apt-get install --yes --no-install-recommends -t jessie-backports openjdk-8-jdk && \
```

(continua na próxima página)

```

rm /etc/apt/sources.list.d/jessie-backports.list && \
apt-get clean && rm -rf /var/lib/apt/lists/* && \
# Add hadoop binaries
wget https://mirrors.ukfast.co.uk/sites/ftp.apache.org/hadoop/common/hadoop-2.7.3/
↪hadoop-2.7.3.tar.gz && \
tar -xvf hadoop-2.7.3.tar.gz -C /usr/local && \
chown -R "${NB_USER}:users" /usr/local/hadoop-2.7.3 && \
rm -f hadoop-2.7.3.tar.gz && \
# Install os dependencies required for pydoop, pyhive
apt-get update --yes && \
apt-get install --yes --no-install-recommends build-essential python-dev libsass2-
↪dev && \
apt-get clean && rm -rf /var/lib/apt/lists/* && \
# Remove the example hadoop configs and replace
# with those for our cluster.
# Alternatively this could be mounted as a volume
rm -f /usr/local/hadoop-2.7.3/etc/hadoop/*

# Download this from ambari / cloudera manager and copy here
COPY example-hadoop-conf/ /usr/local/hadoop-2.7.3/etc/hadoop/

# Spark-Submit doesn't work unless I set the following
RUN echo "spark.driver.extraJavaOptions -Dhdp.version=2.5.3.0-37" >> /usr/local/spark/
↪conf/spark-defaults.conf && \
echo "spark.yarn.am.extraJavaOptions -Dhdp.version=2.5.3.0-37" >> /usr/local/spark/
↪conf/spark-defaults.conf && \
echo "spark.master=yarn" >> /usr/local/spark/conf/spark-defaults.conf && \
echo "spark.hadoop.yarn.timeline-service.enabled=false" >> /usr/local/spark/conf/
↪spark-defaults.conf && \
chown -R "${NB_USER}:users" /usr/local/spark/conf/spark-defaults.conf && \
# Create an alternative HADOOP_CONF_HOME so we can mount as a volume and repoint
# using ENV var if needed
mkdir -p /etc/hadoop/conf/ && \
chown "${NB_USER}:users" /etc/hadoop/conf/

USER ${NB_UID}

# Install useful jupyter extensions and python libraries like :
# - Dashboards
# - PyDoop
# - PyHive
RUN pip install --quiet --no-cache-dir jupyter_dashboards faker && \
jupyter dashboards quick-setup --sys-prefix && \
pip2 install --quiet --no-cache-dir pyhive pydoop thrift sasl thrift_sasl faker && \
fix-permissions "${CONDA_DIR}" && \
fix-permissions "/home/${NB_USER}"

USER root
# Ensure we overwrite the kernel config so that toree connects to cluster
RUN jupyter toree install --sys-prefix --spark_opts="\
--master yarn
--deploy-mode client

```

(continuação da página anterior)

```

--driver-memory 512m
--executor-memory 512m
--executor-cores 1
--driver-java-options
-Dhdp.version=2.5.3.0-37
--conf spark.hadoop.yarn.timeline-service.enabled=false
"
USER ${NB_UID}

```

Credit: [britishbadger](https://github.com/britishbadger) from [docker-stacks/issues/369](https://github.com/docker-stacks/issues/369)

3.5.15 Run Jupyter Notebook/Lab inside an already secured environment (i.e., with no token)

(Adapted from [issue 728](#))

The default security is very good. There are use cases, encouraged by containers, where the jupyter container and the system it runs within, lie inside the security boundary. In these use cases it is convenient to launch the server without a password or token. In this case, you should use the `start.sh` script to launch the server with no token:

For jupyterlab:

```
docker run jupyter/base-notebook:33add21fab64 start.sh jupyter lab --LabApp.token='
```

For jupyter classic:

```
docker run jupyter/base-notebook:33add21fab64 start.sh jupyter notebook --NotebookApp.
↪ token='
```

3.5.16 Enable nbextension spellchecker for markdown (or any other nbextension)

NB: this works for classic notebooks only

```

# Update with your base image of choice
FROM jupyter/minimal-notebook:latest

USER ${NB_UID}

RUN pip install --quiet --no-cache-dir jupyter_contrib_nbextensions && \
    jupyter contrib nbextension install --user && \
    # can modify or enable additional extensions here
    jupyter nbextension enable spellchecker/main --user && \
    fix-permissions "${CONDA_DIR}" && \
    fix-permissions "/home/${NB_USER}"

```

Ref: <https://github.com/jupyter/docker-stacks/issues/675>

3.5.17 Enable Delta Lake in Spark notebooks

Please note that the [Delta Lake](#) packages are only available for Spark version > 3.0. By adding the properties to `spark-defaults.conf`, the user no longer needs to enable Delta support in each notebook.

```
FROM jupyter/pyspark-notebook:latest

ARG DELTA_CORE_VERSION="1.0.0"
RUN pip install --quiet --no-cache-dir delta-spark==${DELTA_CORE_VERSION} && \
    fix-permissions "${HOME}" && \
    fix-permissions "${CONDA_DIR}"

USER root

RUN echo 'spark.sql.extensions io.delta.sql.DeltaSparkSessionExtension' >> "${SPARK_HOME}
↪/conf/spark-defaults.conf" && \
    echo 'spark.sql.catalog.spark_catalog org.apache.spark.sql.delta.catalog.DeltaCatalog
↪' >> "${SPARK_HOME}/conf/spark-defaults.conf"

USER ${NB_UID}

# Trigger download of delta lake files
RUN echo "from pyspark.sql import SparkSession" > /tmp/init-delta.py && \
    echo "from delta import *" >> /tmp/init-delta.py && \
    echo "spark = configure_spark_with_delta_pip(SparkSession.builder).getOrCreate()" >> ↪
↪/tmp/init-delta.py && \
    python /tmp/init-delta.py && \
    rm /tmp/init-delta.py
```

3.5.18 Add Custom Font in Scipy notebook

The example below is a Dockerfile to load Source Han Sans with normal weight which is usually used for web.

```
FROM jupyter/scipy-notebook:latest

RUN PYV=$(ls "${CONDA_DIR}/lib" | grep ^python) && \
    MPL_DATA="${CONDA_DIR}/lib/${PYV}/site-packages/matplotlib/mpl-data" && \
    wget --quiet -P "${MPL_DATA}/fonts/ttf/" https://mirrors.cloud.tencent.com/adobe-
↪fonts/source-han-sans/SubsetOTF/CN/SourceHanSansCN-Normal.otf && \
    sed -i 's/#font.family/font.family/g' "${MPL_DATA}/matplotlibrc" && \
    sed -i 's/#font.sans-serif:/font.sans-serif: Source Han Sans CN,/g' "${MPL_DATA}/
↪matplotlibrc" && \
    sed -i 's/#axes.unicode_minus: True/axes.unicode_minus: False/g' "${MPL_DATA}/
↪matplotlibrc" && \
    rm -rf "/home/${NB_USER}/.cache/matplotlib" && \
    python -c 'import matplotlib.font_manager;print("font loaded: ",("Source Han Sans CN
↪" in [f.name for f in matplotlib.font_manager.fontManager.ttflist]))'
```

3.6 Project Issues

We appreciate your taking the time to report an issue you encountered using the Jupyter Docker Stacks. Please review the following guidelines when reporting your problem.

- If you believe you’ve found a security vulnerability in any of the Jupyter projects included in Jupyter Docker Stacks images, please report it to security@ipython.org, not in the issue trackers on GitHub. If you prefer to encrypt your security reports, you can use [this PGP public key](#).
- If you think your problem is unique to the Jupyter Docker Stacks images, please search the [jupyter/docker-stacks issue tracker](#) to see if someone else has already reported the same problem. If not, please open a [new issue](#) and provide all of the information requested in the issue template.
- If the issue you’re seeing is with one of the open source libraries included in the Docker images and is reproducible outside the images, please file a bug with the appropriate open source project.
- If you have a general question about how to use the Jupyter Docker Stacks in your environment, in conjunction with other tools, with customizations, and so on, please post your question on the [Jupyter Discourse site](#).

3.7 Package Updates

We actively seek pull requests which update packages already included in the project Dockerfiles. This is a great way for first-time contributors to participate in developing the Jupyter Docker Stacks.

Please follow the process below to update a package version:

1. Locate the Dockerfile containing the library you wish to update (e.g., [base-notebook/Dockerfile](#), [scipy-notebook/Dockerfile](#))
2. Adjust the version number for the package. We prefer to pin the major and minor version number of packages so as to minimize rebuild side-effects when users submit pull requests (PRs). For example, you’ll find the Jupyter Notebook package, `notebook`, installed using `conda` with `notebook=5.4.*`.
3. Please build the image locally before submitting a pull request. Building the image locally shortens the debugging cycle by taking some load off GitHub Actions, which graciously provide free build services for open source projects like this one. If you use `make`, call:

```
make build/somestack-notebook
```

4. [Submit a pull request](#) (PR) with your changes.
5. Watch for GitHub to report a build success or failure for your PR on GitHub.
6. Discuss changes with the maintainers and address any build issues. Version conflicts are the most common problem. You may need to upgrade additional packages to fix build failures.

3.7.1 Notes

In order to help identifying packages that can be updated you can use the following helper tool. It will list all the packages installed in the Dockerfile that can be updated – dependencies are filtered to focus only on requested packages.

```
$ make check-outdated/base-notebook
```

```
# INFO      test_outdated:test_outdated.py:80 3/8 (38%) packages could be updated
```

(continua na próxima página)

```
# INFO      test_outdated:test_outdated.py:82
# Package    Current    Newest
# -----
# conda      4.7.12    4.8.2
# jupyterlab 1.2.5     2.0.0
# python     3.7.4     3.8.2
```

3.8 New Recipes

We welcome contributions of *recipes*, short examples of using, configuring, or extending the Docker Stacks, for inclusion in the documentation site. Follow the process below to add a new recipe:

1. Open the `docs/using/recipes.md` source file.
2. Add a second-level Markdown heading naming your recipe at the bottom of the file (e.g., `## Add the RISE extension`)
3. Write the body of your recipe under the heading, including whatever command line, Dockerfile, links, etc. you need.
4. [Submit a pull request](#) (PR) with your changes. Maintainers will respond and work with you to address any formatting or content issues.

3.9 Doc Translations

We are delighted when members of the Jupyter community want to help translate these documentation pages to other languages. If you're interested, please visit links below to join our team on [Transifex](#) and to start creating, reviewing, and updating translations of the Jupyter Docker Stacks documentation.

1. Follow the steps documented on the [Getting Started as a Translator](#) page.
2. Look for *jupyter-docker-stacks* when prompted to choose a translation team. Alternatively, visit <https://www.transifex.com/project-jupyter/jupyter-docker-stacks-1> after creating your account and request to join the project.
3. See [Translating with the Web Editor](#) in the Transifex documentation.

3.10 Lint

In order to enforce some rules **linters** are used in this project. Linters can be run either during the **development phase** (by the developer) and during **integration phase** (by GitHub Actions). To integrate and enforce this process in the project lifecycle we are using **git hooks** through [pre-commit](#).

3.10.1 Pre-commit hook

Pre-commit hook installation

pre-commit is a Python package that needs to be installed. This can be achieved by using the generic task used to install all Python development dependencies.

```
# Install all development dependencies for the project
$ make dev-env
# It can also be installed directly
$ pip install pre-commit
```

Then the git hooks scripts configured for the project in `.pre-commit-config.yaml` need to be installed in the local git repository.

```
make pre-commit-install
```

Run

Now pre-commit (and so configured hooks) will run automatically on `git commit` on each changed file. However it is also possible to trigger it against all files.

- Note: Hadolint pre-commit uses docker to run, so docker should be running while running this command.

```
make pre-commit-all
```

3.10.2 Image Lint

To comply with [Docker best practices](#), we are using the [Hadolint](#) tool to analyse each Dockerfile .

Ignoring Rules

Sometimes it is necessary to ignore [some rules](#). The following rules are ignored by default for all images in the `.hadolint.yaml` file.

- [DL3006](#): We use a specific policy to manage image tags.
 - `base-notebook FROM` clause is fixed but based on an argument (`ARG`).
 - Building downstream images from (`FROM`) the latest is done on purpose.
- [DL3008](#): System packages are always updated (`apt-get`) to the latest version.

For other rules, the preferred way to do it is to flag ignored rules in the Dockerfile.

It is also possible to ignore rules by using a special comment directly above the Dockerfile instruction you want to make an exception for. Ignore rule comments look like `# hadolint ignore=DL3001,SC1081`. For example:

```
FROM ubuntu

# hadolint ignore=DL3003,SC1035
RUN cd /tmp && echo "hello!"
```

3.11 Image Tests

We greatly appreciate pull requests that extend the automated tests that vet the basic functionality of the Docker images.

3.11.1 How the Tests Work

GitHub executes `make build-test-all` against pull requests submitted to the `jupyter/docker-stacks` repository. This `make` command builds every docker image. After building each image, the `make` command executes `pytest` to run both image-specific tests like those in `base-notebook/test/` and common tests defined in `test/`. Both kinds of tests make use of global `pytest fixtures` defined in the `conftest.py` file at the root of the projects.

3.11.2 Contributing New Tests

Please follow the process below to add new tests:

1. If the test should run against every image built, add your test code to one of the modules in `test/` or create a new module.
2. If your test should run against a single image, add your test code to one of the modules in `some-notebook/test/` or create a new module.
3. Build one or more images you intend to test and run the tests locally. If you use `make`, call:

```
make build/somestack-notebook
make test/somestack-notebook
```

4. [Submit a pull request \(PR\)](#) with your changes.
5. Watch for GitHub to report a build success or failure for your PR on GitHub.
6. Discuss changes with the maintainers and address any issues running the tests on GitHub.

3.12 New Features

Thank you for contributing to the Jupyter Docker Stacks! We review pull requests of new features (e.g., new packages, new scripts, new flags) to balance the value of the images to the Jupyter community with the cost of maintaining the images over time.

3.12.1 Suggesting a New Feature

Please follow the process below to suggest a new feature for inclusion in one of the core stacks:

1. [Open a GitHub issue](#) describing the feature you'd like to contribute.
2. Discuss with the maintainers whether the addition makes sense in [one of the core stacks](#), as a *recipe in the documentation*, as a *community stack*, or as something else entirely.

3.12.2 Selection Criteria

Roughly speaking, we evaluate new features based on the following criteria:

- **Usefulness to Jupyter users:** Is the feature generally applicable across domains? Does it work with Jupyter Notebook, JupyterLab, JupyterHub, etc.?
- **Fit with the image purpose:** Does the feature match the theme of the stack in which it will be added? Would it fit better in a new, community stack?
- **Complexity of build / runtime configuration:** How many lines of code does the feature require in one of the Dockerfiles or startup scripts? Does it require new scripts entirely? Do users need to adjust how they use the images?
- **Impact on image metrics:** How many bytes does the feature and its dependencies add to the image(s)? How many minutes do they add to the build time?
- **Ability to support the addition:** Can existing maintainers answer user questions and address future build issues? Are the contributors interested in helping with long-term maintenance? Can we write tests to ensure the feature continues to work over time?

3.12.3 Submitting a Pull Request

If there's agreement that the feature belongs in one or more of the core stacks:

1. Implement the feature in a local clone of the `jupyter/docker-stacks` project.
2. Please build the image locally before submitting a pull request Building the image locally shortens the debugging cycle by taking some load off GitHub Actions, which graciously provide free build services for open source projects like this one. If you use `make`, call:

```
make build/somestack-notebook
```

3. [Submit a pull request](#)(PR) with your changes.
4. Watch for GitHub to report a build success or failure for your PR on GitHub.
5. Discuss changes with the maintainers and address any build issues.

3.13 Community Stacks

We love to see the community create and share new Jupyter Docker images. We've put together a [cookiecutter project](#) and the documentation below to help you get started defining, building, and sharing your Jupyter environments in Docker. Following these steps will:

1. Setup a project on GitHub containing a Dockerfile based on either the `jupyter/base-notebook` or `jupyter/minimal-notebook` image.
2. Configure GitHub Actions to build and test your image when users submit pull requests to your repository.
3. Configure Docker Hub to build and host your images for others to use.
4. Update the [list of community stacks](#) in this documentation to include your image.

This approach mirrors how we build and share the core stack images. Feel free to follow it or pave your own path using alternative services and build tools.

3.13.1 Creating a Project

First, install `cookiecutter` using `pip` or `conda`:

```
pip install cookiecutter # or mamba install cookiecutter
```

Run the `cookiecutter` command pointing to the `jupyter/cookiecutter-docker-stacks` project on GitHub.

```
cookiecutter https://github.com/jupyter/cookiecutter-docker-stacks.git
```

Enter a name for your new stack image. This will serve as both the git repository name and the part of the Docker image name after the slash.

```
stack_name [my-jupyter-stack]:
```

Enter the user or organization name under which this stack will reside on Docker Hub. You must have access to manage this Docker Hub organization to push images here and set up automated builds.

```
stack_org [my-project]:
```

Select an image from the `jupyter/docker-stacks` project that will serve as the base for your new image.

```
stack_base_image [jupyter/base-notebook]:
```

Enter a longer description of the stack for your README.

```
stack_description [my-jupyter-stack is a community maintained Jupyter Docker Stack,
↪ image]:
```

Initialize your project as a Git repository and push it to GitHub.

```
cd <stack_name you chose>

git init
git add .
git commit -m 'Seed repo'
git remote add origin <url from github>
git push -u origin master
```

3.13.2 Configuring GitHub actions

The `cookiecutter` template comes with a `.github/workflows/docker.yml` file, which allows you to use GitHub actions to build your Docker image whenever you or someone else submits a pull request.

1. By default the `.github/workflows/docker.yml` file has the following triggers configuration:

```
on:
  pull_request:
    paths-ignore:
      - "*.md"
  push:
    branches:
      - main
      - master
```

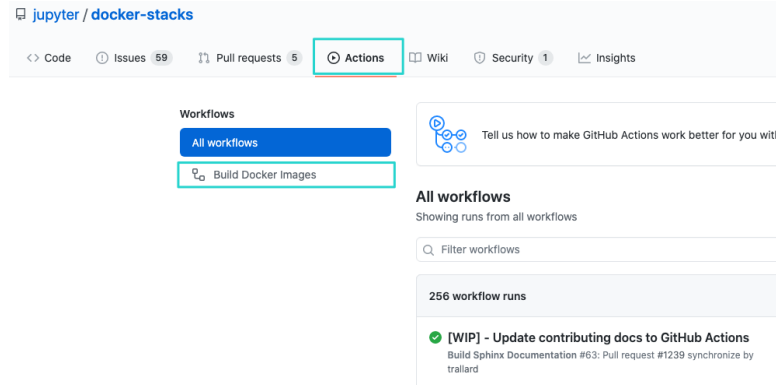
(continua na próxima página)

(continuação da página anterior)

```
paths-ignore:
- "**.md"
```

This will trigger the CI pipeline whenever you push to your `main` or `master` branch and when any Pull Requests are made to your repository. For more details on this configuration, visit the [GitHub actions documentation on triggers](#).

2. Commit your changes and push to GitHub.



3. Head back to your repository and click on the **Actions** tab.
From there, you can click on the workflows on the left-hand side of the screen.
4. In the next screen, you will be able to see information about the workflow run and duration. If you click again on the button with the workflow name, you will see the logs for the workflow steps.

✓ Merge pull request #1261 from moschlar/patch-1 Build Docker Images #193

Summary

Jobs

✓ Build Docker Images

Triggered via push 9 days ago

romainx pushed · 4d9c9bd · master

Status

Success

Total duration

37m 52s

Artifacts

—

docker.yml

on: push

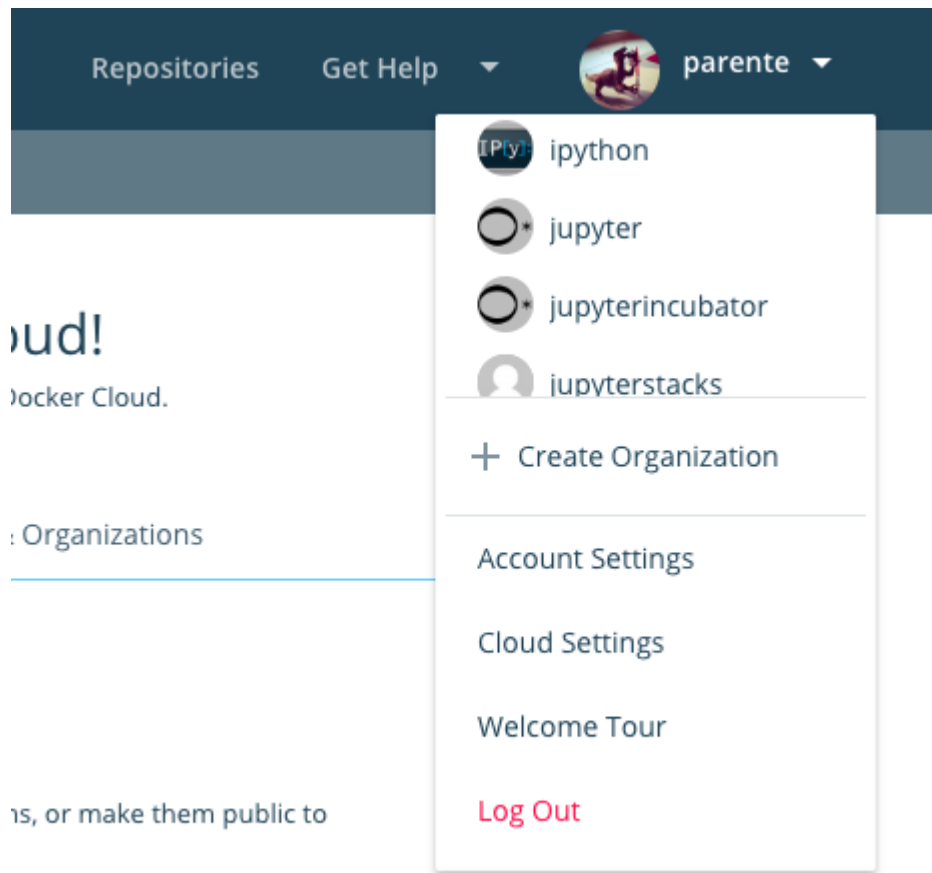
✓ Build Docker Images

37m 39s

3.13.3 Configuring Docker Hub

Now, configure Docker Hub to build your stack image and push it to Docker Hub repository whenever you merge a GitHub pull request to the master branch of your project.

1. Visit <https://hub.docker.com/> and log in.
2. Select the account or organization matching the one you entered when prompted with `stack_org` by the cooki-



ecutter.

3. Scroll to the bottom of the page and click **Create repository**.
4. Enter the name of the image matching the one you entered when prompted with `stack_name` by the cookiecutter.

Create Repository

parente / my-stack

My specialized Jupyter stack

Visibility

Using 0 of 1 private repositories. [Get more](#)



Public

Public repositories appear in Docker Store search results

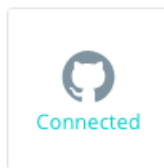


Private

Only you can see private repositories

Build Settings *(optional)*

Autobuild triggers a new build with every **git push** to your source code repository [Learn more](#)

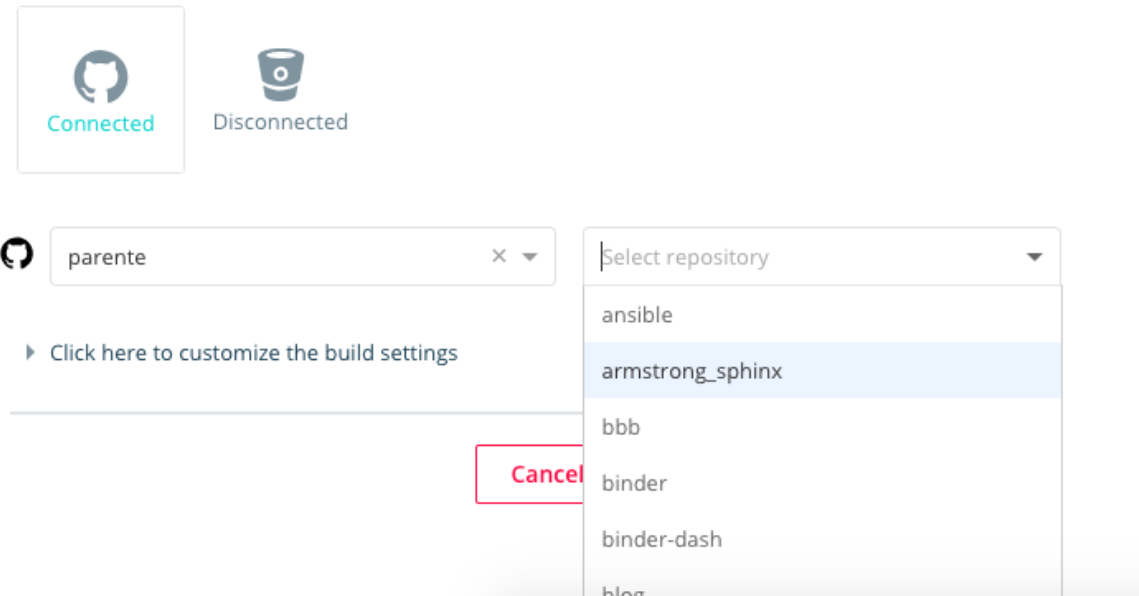


Disconnected

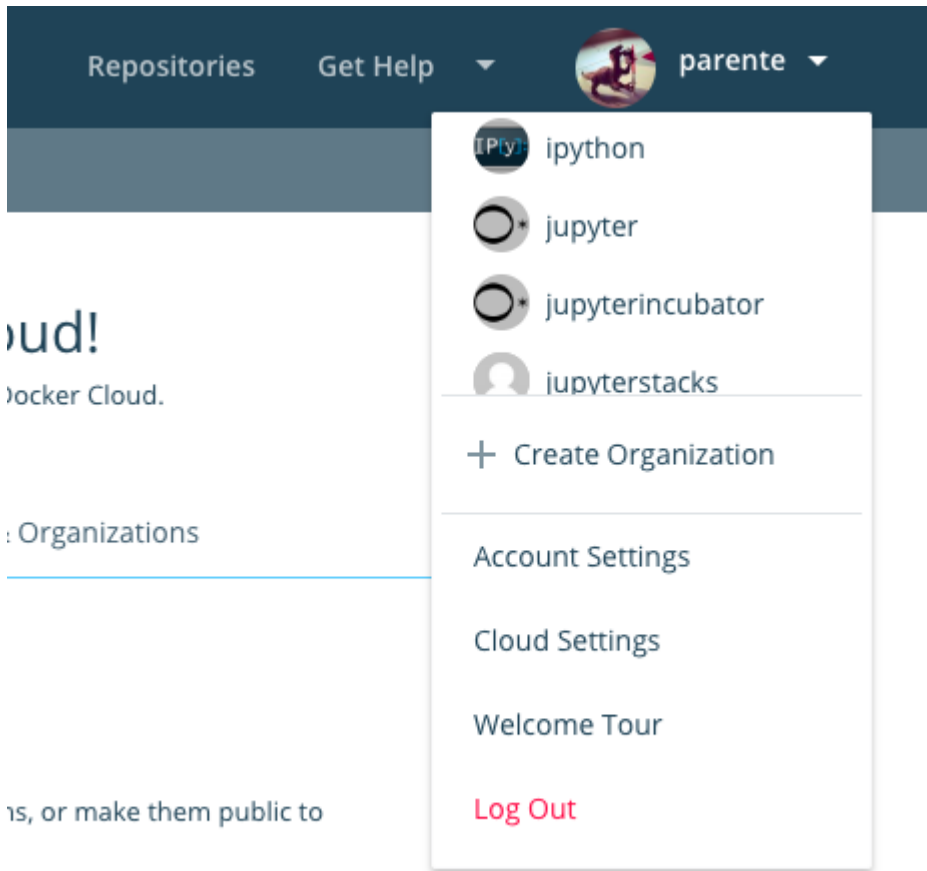
5. Enter a description for your image.
6. Click **GitHub** under the **Build Settings** and follow the prompts to connect your account if it is not already connected.
7. Select the GitHub organization and repository containing your image definition from the dropdowns.

Build Settings *(optional)*

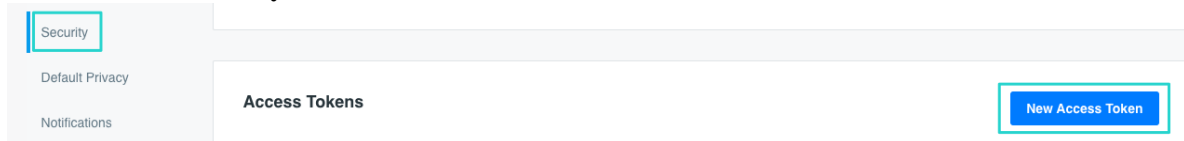
Autobuild triggers a new build with every **git push** to your source code repository [Learn more](#)



8. Click the **Create and Build** button.
9. Click on your avatar on the top-right corner and select Account settings.



10. Click on **Security** and then click on the **New Access Token** button.



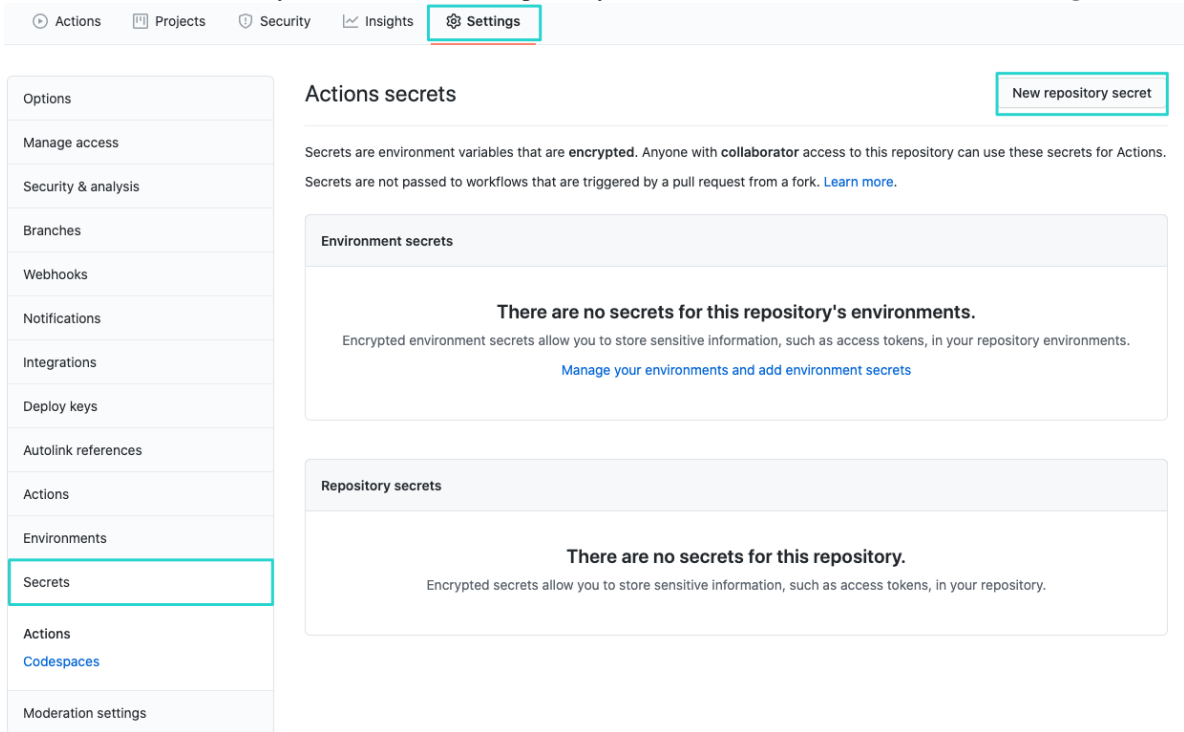
New Access Token

A personal access token is similar to a password except you can use it to access to each one at any time. [Learn more](#)

my-jupyter-docker-token

11. Enter a meaningful name for your token and click on **Create**



12. Copy the personal access token displayed on the next screen. **Note that you will not be able to see it again after you close the pop-up window.**
13. Head back to your GitHub repository and click on the **Settings** tab.



14. Click on the **Secrets** section and then on the **New repository secret** button on the top right corner (see image above).
15. Create a **DOCKERHUB_TOKEN** secret and paste the Personal Access Token from DockerHub in the **value** field. [Actions secrets / New secret](#)

eld.

16. Repeat the above step but creating a **DOCKERHUB_USERNAME** and replacing the *value* field with your DockerHub username. Once you have completed these steps, your repository secrets section should look something like this:

Repository secrets		
 DOCKERHUB_TOKEN	Updated now	<button>Update</button> <button>Remove</button>
 DOCKERHUB_USERNAME	Updated 3 minutes ago	<button>Update</button> <button>Remove</button>

3.13.4 Defining Your Image

Make edits to the Dockerfile in your project to add third-party libraries and configure Jupyter applications. Refer to the Dockerfiles for the core stacks (e.g., [jupyter/datascience-notebook](#)) to get a feel for what's possible and best practices.

Submit pull requests to your project repository on GitHub. Ensure your image builds correctly on GitHub actions before merging to master or main. Refer to Docker Hub to build your master or main branch that you can `docker pull`.

3.13.5 Sharing Your Image

Finally, if you'd like to add a link to your project to this documentation site, please do the following:

1. Clone the [jupyter/docker-stacks](#) GitHub repository.
2. Open the `docs/using/selecting.md` source file and locate the **Community Stacks** section.
3. Add a bullet with a link to your project and a short description of what your Docker image contains.
4. [Submit a pull request](#)(PR) with your changes. Maintainers will respond and work with you to address any formatting or content issues.

3.14 Maintainer Playbook

3.14.1 Merging Pull Requests

To build new images and publish them to the Docker Hub registry, do the following:

1. Make sure GitHub Actions status checks pass for the PR.
2. Merge the PR.
3. Monitor the merge commit GitHub Actions status. **Note:** we think, GitHub Actions are quite reliable, so please, investigate, if some error occurs. The process of building docker images in PRs is exactly the same after merging to master, except there is an additional push step.
4. Try to avoid merging another PR to master until all pending builds complete. This way you will know which commit might have broken the build and also have correct tags for moving tags (like `python` version).

3.14.2 Updating the Ubuntu Base Image

When there's a security fix in the Ubuntu base image or after some time passes, it's a good idea to update the pinned SHA in the [jupyter/base-notebook Dockerfile](#). Submit it as a regular PR and go through the build process. Expect the build to take a while to complete: every image layer will rebuild.

3.14.3 Adding a New Core Image to Docker Hub

When there's a new stack definition, do the following before merging the PR with the new stack:

1. Ensure the PR includes an update to the stack overview diagram [in the documentation](#). The image links to the [blockdiag source](#) used to create it.
2. Ensure the PR updates the [Makefile](#) which is used to build the stacks in order on GitHub Actions.
3. Ensure necessary tags / manifests are added for the new image in the [tagging](#) folder.
4. Create a new repository in the [jupyter org](#) on Docker Hub named after the stack folder in the git repo.
5. Grant the [stacks team](#) permission to write to the repo.

3.14.4 Adding a New Maintainer Account

1. Visit <https://hub.docker.com/app/jupyter/team/stacks/users>
2. Add the maintainer's Docker Hub username.
3. Visit <https://github.com/orgs/jupyter/teams/docker-image-maintainers/members>
4. Add the maintainer's GitHub username.

3.14.5 Pushing a Build Manually

If automated build in Github Actions has got you down, do the following to push a build manually:

1. Clone this repository.
2. Check out the git SHA you want to build and publish.
3. `docker login` with your Docker Hub credentials.
4. Run `make push-all`.

3.14.6 Enabling a New Doc Language Translation

First enable translation on Transifex:

1. Visit <https://www.transifex.com/project-jupyter/jupyter-docker-stacks-1/languages/>.
2. Click *Edit Languages* in the top right.
3. Select the language from the dropdown.
4. Click *Apply*.

Then setup a subproject on ReadTheDocs for the language:

1. Visit <https://readthedocs.org/dashboard/import/manual/>.
2. Enter `jupyter-docker-stacks-language_abbreviation` for the project name.

3. Enter <https://github.com/jupyter/docker-stacks> for the URL.
4. Check *Edit advanced project options*.
5. Click *Next*.
6. Select the *Language* from the dropdown on the next screen.
7. Click *Finish*.

Finally link the new language subproject to the top level doc project:

1. Visit <https://readthedocs.org/dashboard/jupyter-docker-stacks/translations/>.
2. Select the subproject you created from the *Project* dropdown.
3. Click *Add*.