

---

# **docker-stacks Documentation**

***Release latest***

**Project Jupyter**

**Jan 09, 2019**



<b>1</b>	<b>Quick Start</b>	<b>3</b>
<b>2</b>	<b>Table of Contents</b>	<b>5</b>
2.1	Selecting an Image . . . . .	5
2.2	Running a Container . . . . .	8
2.3	Common Features . . . . .	11
2.4	Image Specifics . . . . .	14
2.5	Contributed Recipes . . . . .	18
2.6	Package Updates . . . . .	24
2.7	New Recipes . . . . .	24
2.8	Image Tests . . . . .	24
2.9	New Features . . . . .	25
2.10	Community Stacks . . . . .	26
2.11	Maintainer Playbook . . . . .	31



Jupyter Docker Stacks are a set of ready-to-run Docker images containing Jupyter applications and interactive computing tools. You can use a stack image to do any of the following (and more):

- Start a personal Jupyter Notebook server in a local Docker container
- Run JupyterLab servers for a team using JupyterHub
- Write your own project Dockerfile



# CHAPTER 1

---

## Quick Start

---

You can try a [recent build of the jupyter/base-notebook image on mybinder.org](#) by simply clicking the preceding link. Otherwise, the two 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.

The other pages in this documentation describe additional uses and features in detail.

**Example 1:** This command pulls the `jupyter/scipy-notebook` image tagged `17aba6048f44` 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:17aba6048f44
```

**Example 2:** This command pulls the `jupyter/datascience-notebook` image tagged `9b06df75e445` 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/?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:9b06df75e445
```





## 2.1 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.

### 2.1.1 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 `pandoc` for saving notebooks as PDFs)
- `Miniconda` Python 3.x in `/opt/conda`
- 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`
- [Pandoc](#) and [TeX Live](#) for notebook document conversion
- `git`, `emacs`, `jed`, `nano`, 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, including `ggplot2`, `dplyr`, `tidyr`, `readr`, `purrr`, `tibble`, `stringr`, `lubridate`, and `broom` from [conda-forge](#)
- `plyr`, `devtools`, `shiny`, `rmarkdown`, `forecast`, `rsqlite`, `reshape2`, `nycflights13`, `caret`, `rcurl`, and `randomforest` 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
- `pandas`, `numexpr`, `matplotlib`, `scipy`, `seaborn`, `scikit-learn`, `scikit-image`, `sympy`, `cython`, `patsy`, `statsmodel`, `cloudpickle`, `dill`, `numba`, `bokeh`, `sqlalchemy`, `hdf5`, `vincent`, `beautifulsoup`, `protobuf`, and `xlrd` packages
- `ipywidgets` for interactive visualizations 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](#) and [keras](#) machine learning libraries

## 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
- The [Julia](#) compiler and base environment
- [IJulia](#) to support Julia code in Jupyter notebooks
- [HDF5](#), [Gadfly](#), and [RDatasets](#) packages

## jupyter/pyspark-notebook

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

jupyter/pyspark-notebook includes Python support for Apache Spark, optionally on Mesos.

- Everything in jupyter/scipy-notebook and its ancestor images
- [Apache Spark](#) with Hadoop binaries
- [Mesos](#) client libraries

## 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, optionally on Mesos.

- Everything in jupyter/pyspark-notebook and its ancestor images
- [IRKernel](#) to support R code in Jupyter notebooks
- [Apache Toree](#) and [spylon-kernel](#) to support Scala code in Jupyter notebooks
- [ggplot2](#), [sparklyr](#), and [rcurl](#) packages

## 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

Pull requests to the `jupyter/docker-stacks` repository trigger builds of all images on Travis CI. These images are for testing purposes only and are not saved for use. When pull requests merge to master, all images rebuild on Docker Cloud and become available to `docker pull` from Docker Hub.

## Versioning

The `latest` tag in each Docker Hub repository tracks the master branch HEAD reference on GitHub. `latest` is a moving target, by definition, and will have backward-incompatible changes regularly.

Every image on Docker Hub also receives a 12-character tag which corresponds with the git commit SHA that triggered the image build. You can inspect the state of the `jupyter/docker-stacks` repository for that commit to review the definition of the image (e.g., images with tag `7c45ec67c8e7` were built from <https://github.com/jupyter/docker-stacks/tree/7c45ec67c8e7>).

You must refer to git-SHA image tags when stability and reproducibility are important in your work. (e.g. `FROM jupyter/scipy-notebook:7c45ec67c8e7, docker run -it --rm jupyter/scipy-notebook:7c45ec67c8e7`). You should only use `latest` when a one-off container instance is acceptable (e.g., you want to briefly try a new library in a notebook).

### 2.1.2 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.

*This list only have 1 example. You can be the next!*

- [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](#). Click here to launch .

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

## 2.2 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.

### 2.2.1 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 `2c80cf3537ca` 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:2c80cf3537ca
```

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=112bb073331f1460b73768c76dff2f87ac1d4ca7870d46a
[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=112bb073331f1460b73768c76dff2f87ac1d4ca7870d46a

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             NAMES
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 e5c5a7d3e52d 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:e5c5a7d3e52d
```

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.
```

(continues on next page)

(continued from previous page)

```
[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
```

## 2.2.2 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.

### 2.2.3 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.

### 2.2.4 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.

## 2.3 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.

### 2.3.1 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
```

### 2.3.2 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` and set the working directory `-w /home/$NB_USER`. This feature is useful when mounting host volumes with specific home folder.
- `-e NB_UID=1000` - Instructs the startup script to switch the numeric user ID of `$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.
- `-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.
- `-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`.

### 2.3.3 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.

## 2.3.4 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.

## 2.3.5 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 your 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`).

### 2.3.6 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 `conda` or `pip` to install new packages without any additional permissions.

```
# install a package into the default (python 3.x) environment
pip install some-package
conda install some-package
```

## 2.4 Image Specifics

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

### 2.4.1 Apache Spark

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.

#### In a Python Notebook

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

# do something to prove it works
rdd = sc.parallelize(range(1000))
rdd.takeSample(False, 5)
```

#### In a R Notebook

```
library(SparkR)

as <- sparkR.session("local[*]")
```

(continues on next page)

(continued from previous page)

```
# do something to prove it works
df <- as.DataFrame(iris)
head(filter(df, df$Petal_Width > 0.2))
```

### In a Spylon Kernel Scala Notebook

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[*]"
```

```
// Now run Scala code that uses the initialized SparkContext in sc
val rdd = sc.parallelize(0 to 999)
rdd.takeSample(false, 5)
```

### In an Apache Toree Scala Notebook

Apache Toree instantiates a local `SparkContext` for you in variable `sc` when the kernel starts.

```
val rdd = sc.parallelize(0 to 999)
rdd.takeSample(false, 5)
```

### Connecting to a Spark Cluster on Mesos

This configuration allows your compute cluster to scale with your data.

1. Deploy Spark on Mesos.
2. Configure each slave with the `--no-switch_user` flag or create the `$NB_USER` account on every slave node.
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.
4. Follow the language specific instructions below.

### In a Python Notebook

```
import os
# make sure pyspark tells workers to use python3 not 2 if both are installed
os.environ['PYSPARK_PYTHON'] = '/usr/bin/python3'

import pyspark
conf = pyspark.SparkConf()

# point to mesos master or zookeeper entry (e.g., zk://10.10.10.10:2181/mesos)
```

(continues on next page)

(continued from previous page)

```

conf.setMaster("mesos://10.10.10.10:5050")
# point to spark binary package in HDFS or on local filesystem on all slave
# nodes (e.g., file:///opt/spark/spark-2.2.0-bin-hadoop2.7.tgz)
conf.set("spark.executor.uri", "hdfs://10.10.10.10/spark/spark-2.2.0-bin-hadoop2.7.tgz
→")
# set other options as desired
conf.set("spark.executor.memory", "8g")
conf.set("spark.core.connection.ack.wait.timeout", "1200")

# create the context
sc = pyspark.SparkContext(conf=conf)

# do something to prove it works
rdd = sc.parallelize(range(100000000))
rdd.sumApprox(3)

```

## In a R Notebook

```

library(SparkR)

# Point to mesos master or zookeeper entry (e.g., zk://10.10.10.10:2181/mesos)
# Point to spark binary package in HDFS or on local filesystem on all slave
# nodes (e.g., file:///opt/spark/spark-2.2.0-bin-hadoop2.7.tgz) in sparkEnvir
# Set other options in sparkEnvir
sc <- sparkR.session("mesos://10.10.10.10:5050", sparkEnvir=list(
  spark.executor.uri="hdfs://10.10.10.10/spark/spark-2.2.0-bin-hadoop2.7.tgz",
  spark.executor.memory="8g"
))

# do something to prove it works
data(iris)
df <- as.DataFrame(iris)
head(filter(df, df$Petal_Width > 0.2))

```

## In a Spylon Kernel Scala Notebook

```

%%init_spark
# Configure the location of the mesos master and spark distribution on HDFS
launcher.master = "mesos://10.10.10.10:5050"
launcher.conf.spark.executor.uri=hdfs://10.10.10.10/spark/spark-2.2.0-bin-hadoop2.7.
→tgz

```

```

// Now run Scala code that uses the initialized SparkContext in sc
val rdd = sc.parallelize(0 to 999)
rdd.takeSample(false, 5)

```

## In an Apache Toree Scala Notebook

The Apache Toree kernel automatically creates a `SparkContext` when it starts based on configuration information from its command line arguments and environment variables. You can pass information about your Mesos cluster via

the SPARK\_OPTS environment variable when you spawn a container.

For instance, to pass information about a Mesos master, Spark binary location in HDFS, and an executor options, you could start the container like so:

```
docker run -d -p 8888:8888 -e SPARK_OPTS='--master=mesos://10.10.10.10:5050 \
--spark.executor.uri=hdfs://10.10.10.10/spark/spark-2.2.0-bin-hadoop2.7.tgz \
--spark.executor.memory=8g' jupyter/all-spark-notebook
```

Note that this is the same information expressed in a notebook in the Python case above. Once the kernel spec has your cluster information, you can test your cluster in an Apache Toree notebook like so:

```
// should print the value of --master in the kernel spec
println(sc.master)

// do something to prove it works
val rdd = sc.parallelize(0 to 99999999)
rdd.sum()
```

## 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.
4. The language specific instructions are almost same as mentioned above for Mesos, only the master url would now be something like `spark://10.10.10.10:7077`

## 2.4.2 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)
```

## 2.5 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.

### 2.5.1 Using `pip install` or `conda 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:9f9e5ca8fe5a
# Install in the default python3 environment
RUN pip install 'ggplot==0.6.8'
```

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:9f9e5ca8fe5a
# Install from requirements.txt file
COPY requirements.txt /tmp/
RUN pip install --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:9f9e5ca8fe5a
# Install from requirements.txt file
COPY requirements.txt /tmp/
RUN conda install --yes --file /tmp/requirements.txt && \
    fix-permissions $CONDA_DIR && \
    fix-permissions /home/$NB_USER
```

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

## 2.5.2 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 conda create --quiet --yes -p $CONDA_DIR/envs/python2 python=2.7 ipython_
↳ipykernel kernda && \
    conda clean -tipsy

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_USER
```

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

## 2.5.3 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 jupyter/datascience-notebook start.sh jupyter lab`

## 2.5.4 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>

## 2.5.5 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 conda install -c damianavila82 rise
```

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

## 2.5.6 xgboost

You need to install conda's gcc for Python xgboost to work properly. Otherwise, you'll get an exception about libgomp.so.1 missing GOMP\_4.0.

```
%%bash
conda install -y gcc
pip install xgboost

import xgboost
```

## 2.5.7 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.

## 2.5.8 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>

## 2.5.9 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](https://github.com/jupyter/docker-stacks/issues/124) and [docker-stacks/pull/185](https://github.com/jupyter/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:5ded1de07260
RUN pip install jupyterhub==0.8.0b1
```

Credit: [MinRK](#)

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

## 2.5.10 Spark

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

### Using PySpark with AWS S3

```
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()
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](https://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 http://cdn-fastly.deb.debian.org/debian jessie-backports main' > /etc/
↳ apt/sources.list.d/jessie-backports.list && \
    apt-get -y update && \
    apt-get install --no-install-recommends -t jessie-backports -y openjdk-8-jdk && \
    rm /etc/apt/sources.list.d/jessie-backports.list && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/ && \
# Add hadoop binaries
    wget http://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 && \
    apt-get install --no-install-recommends -y build-essential python-dev libsasl2-
↳ 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
```

(continues on next page)

(continued from previous page)

```

mkdir -p /etc/hadoop/conf/ && \
chown $NB_USER:users /etc/hadoop/conf/

USER $NB_USER

# Install useful jupyter extensions and python libraries like :
# - Dashboards
# - PyDoop
# - PyHive
RUN pip install jupyter_dashboards faker && \
    jupyter dashboards quick-setup --sys-prefix && \
    pip2 install pyhive pydoop thrift sasl thrift_sasl faker

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 --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_USER

```

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

## 2.5.11 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:6d2a05346196 start.sh jupyter lab --LabApp.token=''
```

For jupyter classic:

```
docker run jupyter/base-notebook:6d2a05346196 start.sh jupyter notebook --NotebookApp.
↪token=''
```

## 2.5.12 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_USER

RUN pip install jupyter_contrib_nbextensions && \
    jupyter contrib nbextension install --user && \
    # can modify or enable additional extensions here
    jupyter nbextension enable spellchecker/main --user

```

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

## 2.6 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 [Travis CI](#), which graciously provides free build services for open source projects like this one. If you use `make`, call:

```
make image/somestack-notebook
```

1. [Submit a pull request](#) (PR) with your changes.
2. Watch for Travis to report a build success or failure for your PR on GitHub.
3. 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.

## 2.7 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.

## 2.8 Image Tests

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

### 2.8.1 How the Tests Work

Travis executes `make build-test-all` against every pull request submitted to the `jupyter/docker-stacks` repository. The `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.

### 2.8.2 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 image/somestack-notebook
make test/somestack-notebook
```

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

## 2.9 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.

### 2.9.1 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.

### 2.9.2 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?

### 2.9.3 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 [Travis CI](#), which graciously provides free build services for open source projects like this one. If you use `make`, call:

```
make image/somestack-notebook
```

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

## 2.10 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 Travis CI to build and test your image when users submit pull requests to your repository.
3. Configure Docker Cloud 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.

### 2.10.1 Creating a Project

First, install [cookiecutter](#) using `pip` or `conda`:

```
pip install cookiecutter    # or conda 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 Cloud / Hub. You must have access to manage this Docker Cloud org in order to push images here and setup 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]:
```

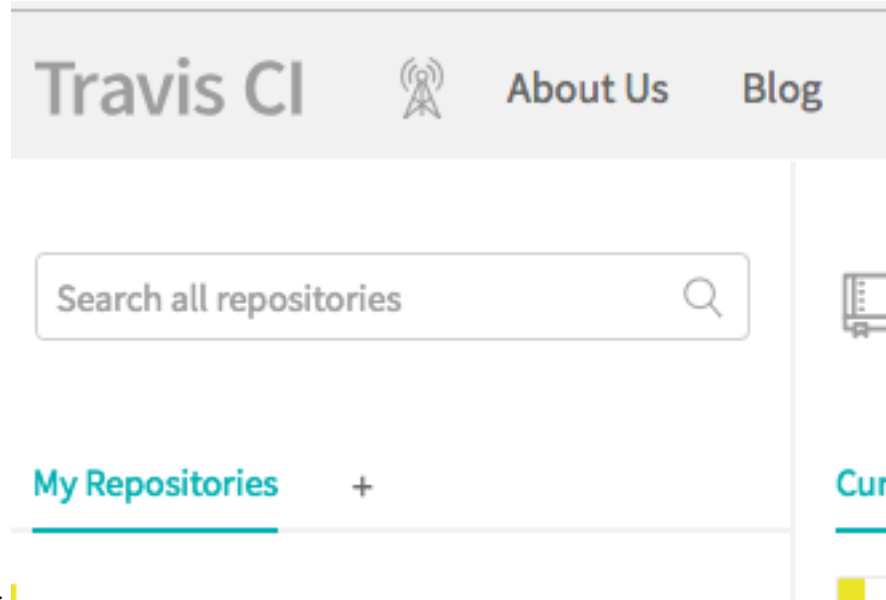
Initialize your project as a Git repository and push it to GitHub.


```
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
```

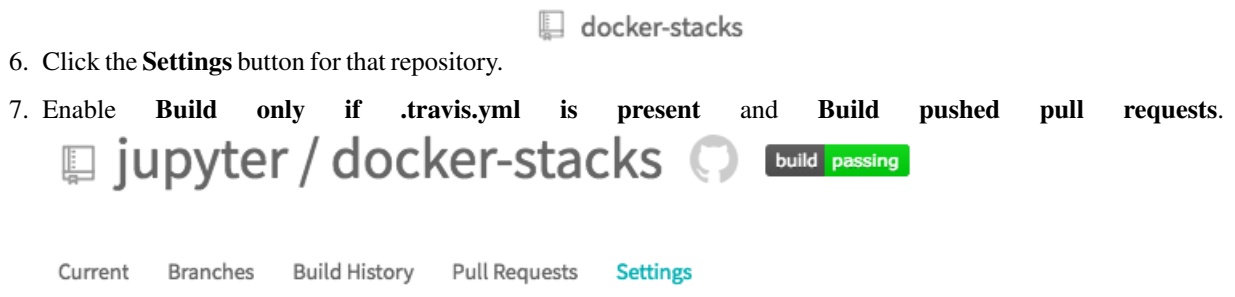
## 2.10.2 Configuring Travis

Next, link your GitHub project to Travis CI to build your Docker image whenever you or someone else submits a pull request.

1. Visit <https://docs.travis-ci.com/user/getting-started/#To-get-started-with-Travis-CI> and follow the instructions to add the Travis CI application to your GitHub account.
2. Visit <https://travis-ci.org>.



3. Click the + symbol at the top of the left sidebar. 
4. Locate your project repository either in your primary user account or in one of the organizations to which you belong.
5. Click the toggle to enable builds for the project repository.



### General

- |  |  |
|--|--|
| <input checked="" type="checkbox"/> Build only if .travis.yml is present   | <input type="checkbox"/> Build pushed branches                  |
| <input type="checkbox"/> Limit concurrent jobs  | <input checked="" type="checkbox"/> Build pushed pull requests  |

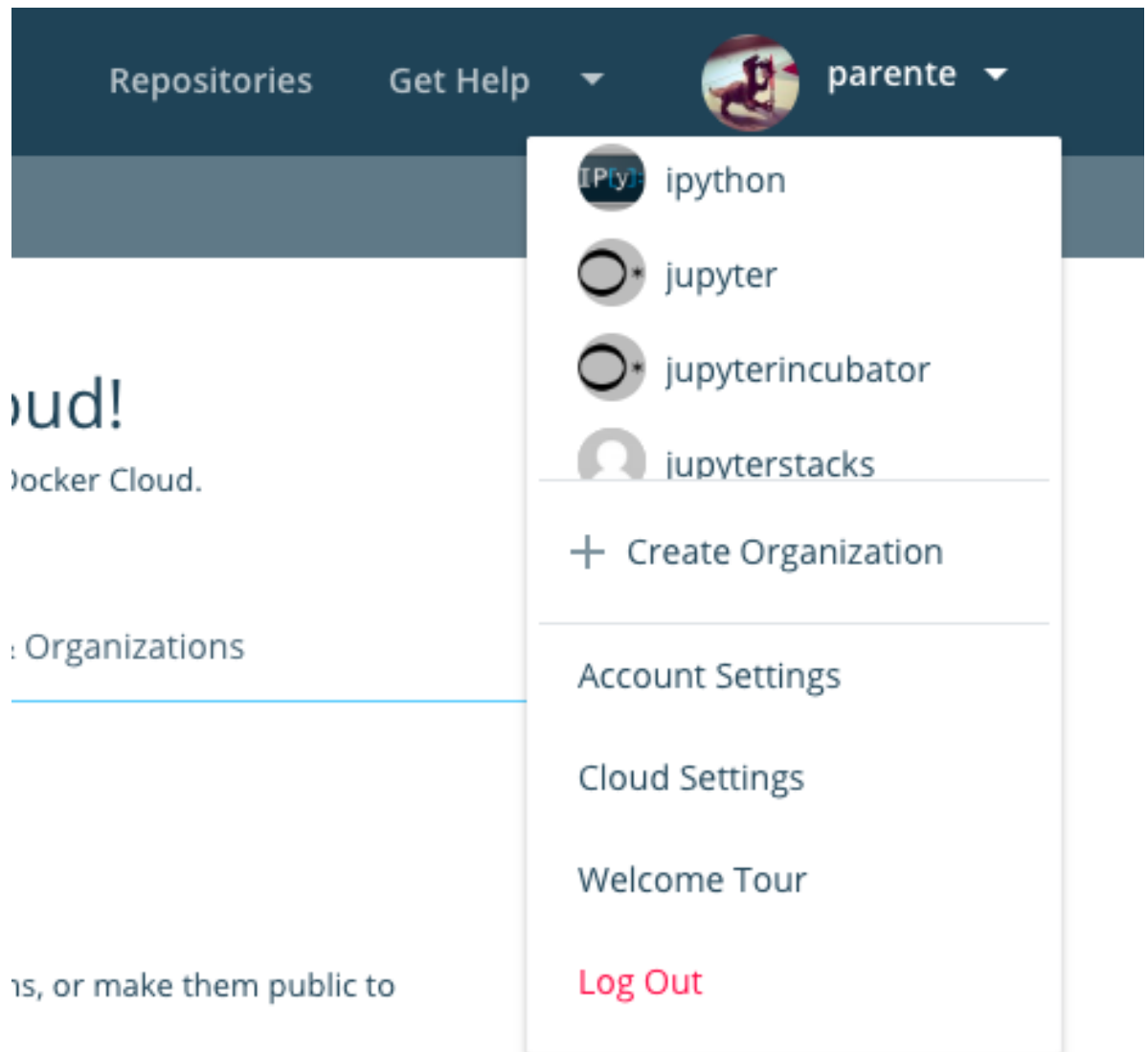
8. Disable **Build pushed branches**.

## 2.10.3 Configuring Docker Cloud

Now, configure Docker Cloud 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://cloud.docker.com/> and login.
2. Select the account or organization matching the one you entered when prompted with `stack_org` by the cook-





iecutter.

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 cookiecut-

## 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

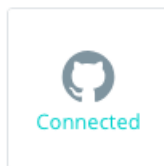


#### Private

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](#)

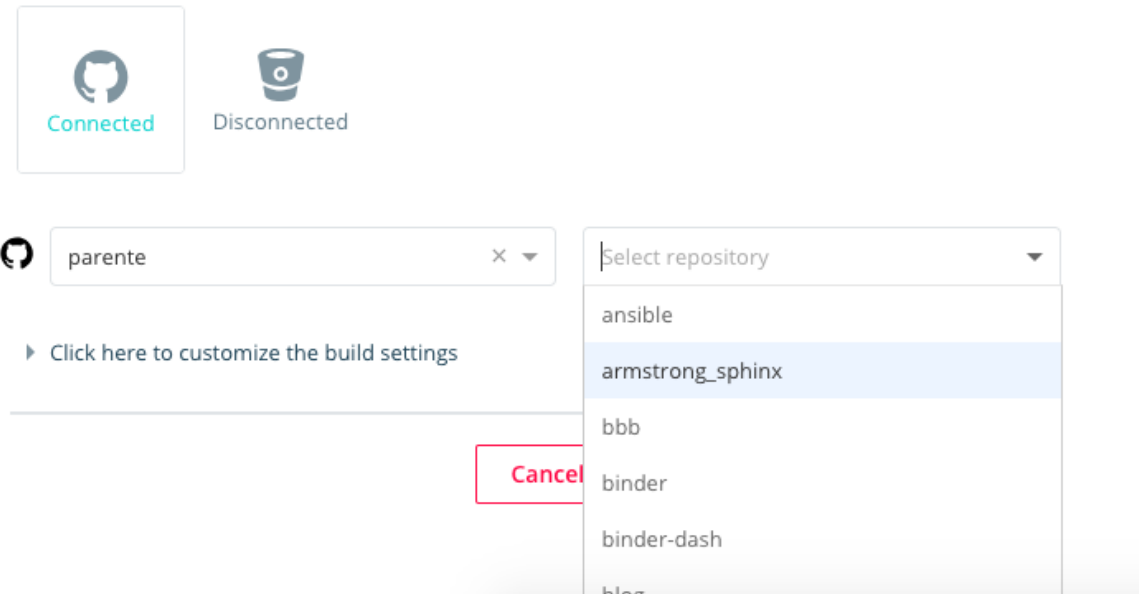


ter.

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.

## 2.10.4 Defining Your Image

Make edits 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 properly on Travis before merging to master. Refer to Docker Cloud for builds of your master branch that you can `docker pull`.

## 2.10.5 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 ths [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.

## 2.11 Maintainer Playbook

### 2.11.1 Merging Pull Requests

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

1. Make sure Travis is green for a PR.
2. Merge the PR.
3. Monitor the Docker Cloud build status for each of the stacks, starting with [jupyter/base-notebook](#) and ending with [jupyter/all-spark-notebook](#). See the [stack hierarchy diagram](#) for the current, complete build order.
4. Manually click the retry button next to any build that fails to resume that build and any dependent builds.
5. Try to avoid merging another PR to master until all outstanding builds complete. There's no way at present to propagate the git SHA to build through the Docker Cloud build trigger API. Every build trigger works off of master HEAD.

### 2.11.2 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.

### 2.11.3 Adding a New Core Image to Docker Cloud

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 Travis CI.
3. Create a new repository in the `jupyter` org on Docker Cloud named after the stack folder in the git repo.
4. Grant the `stacks` team permission to write to the repo.
5. Click *Builds* and then *Configure Automated Builds* for the repository.
6. Select `jupyter/docker-stacks` as the source repository.
7. Choose *Build on Docker Cloud's infrastructure using a Small node* unless you have reason to believe a bigger host is required.
8. Update the *Build Context* in the default build rule to be `/<name-of-the-stack>`.
9. Toggle *Autobuild* to disabled unless the stack is a new root stack (e.g., like `jupyter/base-notebook`).
10. If the new stack depends on the build of another stack in the hierarchy:
  - (a) Hit *Save* and then click *Configure Automated Builds*.
  - (b) At the very bottom, add a build trigger named *Stack hierarchy trigger*.
  - (c) Copy the build trigger URL.
  - (d) Visit the parent repository *Builds* page and click *Configure Automated Builds*.
  - (e) Add the URL you copied to the `NEXT_BUILD_TRIGGERS` environment variable comma separated list of URLs, creating that environment variable if it does not already exist.
  - (f) Hit *Save*.
11. If the new stack should trigger other dependent builds:
  - (a) Add an environment variable named `NEXT_BUILD_TRIGGERS`.
  - (b) Copy the build trigger URLs from the dependent builds into the `NEXT_BUILD_TRIGGERS` comma separated list of URLs.

(c) Hit *Save*.

12. Adjust other *NEXT\_BUILD\_TRIGGERS* values as needed so that the build order matches that in the stack hierarchy diagram.

### 2.11.4 Adding a New Maintainer Account

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

### 2.11.5 Pushing a Build Manually

If automated builds on Docker Cloud have 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/Cloud credentials.
4. Run `make retry/release-all`.