# voila

***Release 0.2.10***

**The Voilà Development Team**

**May 28, 2021**

# CONTENTS

*From notebooks to standalone web applications and dashboards.*

Voilà allows you to convert a Jupyter Notebook into an interactive dashboard that allows you to share your work with others. It is secure and customizable, giving you control over what your readers experience.

For example, here's a dashboard created with Voilà. (You can try it interactively at the following Binder link)

For more information about Voilà, see the sections below.

CHAPTER

ONE

# INSTALLING VOILÀ

Voilà can be installed with the `mamba` or `conda` package manager

```
mamba install -c conda-forge voila
```

or from `PyPI`:

```
pip install voila
```

Once Voilà is installed, it can be used either as a Command-Line Interface, or as a Jupyter Server extension. See *Using Voilà* for information on how to use Voilà.

CHAPTER

TWO

# USING VOILÀ

Voilà can be used as a standalone application, or as a Jupyter server extension. This page describes how to do each. Before you begin, make sure that you follow the steps in *Installing Voilà*.

The following sections cover how to use Voilà.

## 2.1 As a standalone application

Voilà can be used to run, convert, and serve a Jupyter notebook as a standalone app. This can be done via the command-line, with the following pattern:

```
voila <path-to-notebook> <options>
```

For example, to render the `bqplot` example notebook as a standalone app, run

```
git clone https://github.com/voila-dashboards/voila
cd voila
voila notebooks/bqplot.ipynb
```

Voilà displays a message when your notebook-based application is live. By default, Voilà runs at `localhost:8866`.

To serve a **directory of Jupyter Notebooks**, navigate to the directory you'd like to serve, then simply run `voila`:

```
cd notebooks/
voila
```

The page served by Voilà will now contain a list of any notebooks in the directory. By clicking on one, you will trigger Voilà's conversion process. A new Jupyter kernel will be created for each notebook you click.

## 2.2 As a Jupyter server extension

You can also use Voilà from within a Jupyter server (e.g., after running `jupyter lab` or `jupyter notebook`).

**Note:** Voilà can also be used as a notebook server extension, both with the notebook server or with the jupyter_server.

To use Voilà within a pre-existing Jupyter server, first start the server, then go to the following URL:

```
<url-of-my-server>/voila
```

For example, if you typed `jupyter lab` and it was running at `http://localhost:8888/lab`, then Voilà would be accessed at `http://localhost:8888/voila`.

In this case, Voilà will serve the directory in which the Jupyter server was started.

## 2.3 How does Voilà work?

When Voilà is run on a notebook, the following steps occur:

1. Voilà runs the code in the notebook and collects the outputs
2. The notebook and its outputs are converted to HTML. By default, the notebook **code cells are hidden**.
3. This page is served either as a Tornado application, or via the Jupyter server.
4. When users access the page, the widgets on the page have access to the underlying Jupyter kernel.

## 2.4 The example notebooks

The notebooks directory contains a collection of Jupyter notebooks that can be rendered using Voilà:

- **basics.ipynb** - a notebook with interactions requiring a roundtrip to the kernel.
- **bqplot.ipynb** - uses custom Jupyter widgets such as bqplot.
- **dashboard.ipynb** - uses gridstack.js for the layout of each output.
- **gridspecLayout.ipynb** - uses GridspecLayout for the layout of different widgets.
- **interactive.ipynb** - makes use of ipywidget's @interact.
- **ipympl.ipynb** - contains custom interactive matplotlib figures using the ipympl widget.
- **ipyvolume.ipynb** - uses custom Jupyter widgets such as ipyvolume.
- **query-strings.ipynb** - uses HTTP query parameters to parametrize a notebook
- **xleaflet.ipynb** - a notebook that uses C++ kernel and interactive widgets

These examples demonstrate different interactive HTML widgets and can be used as inspiration for getting started with Voilà.

To **run the example notebooks**, a few additional libraries can be installed using:

```
conda install -c conda-forge ipywidgets ipyvolume bqplot scipy
```

Or alternatively:

```
conda env create
```

The examples can then be served with:

```
cd notebooks/
voila
```

## 2.5 Using third-party Widgets with Voilà

By default, Voilà doesn't serve Jupyter Widgets installed as a classic notebook extension (nbextension).

Instead, it fallbacks to fetching the files from a CDN. This might result in an error (404) in case the custom widget has not been published to `npm`, or when Voilà runs in an environment without an Internet connection.

To let the Voilà standalone app serve the nbextensions, use the `enable_nbextensions` flag as follows:

```
voila --enable_nbextensions=True
```

When using Voilà as a server extension:

```
jupyter notebook --VoilaConfiguration.enable_nbextensions=True
```

CHAPTER

THREE

# CUSTOMIZING VOILÀ

There are many ways you can customize Voilà to control the look and feel of the dashboards you create.

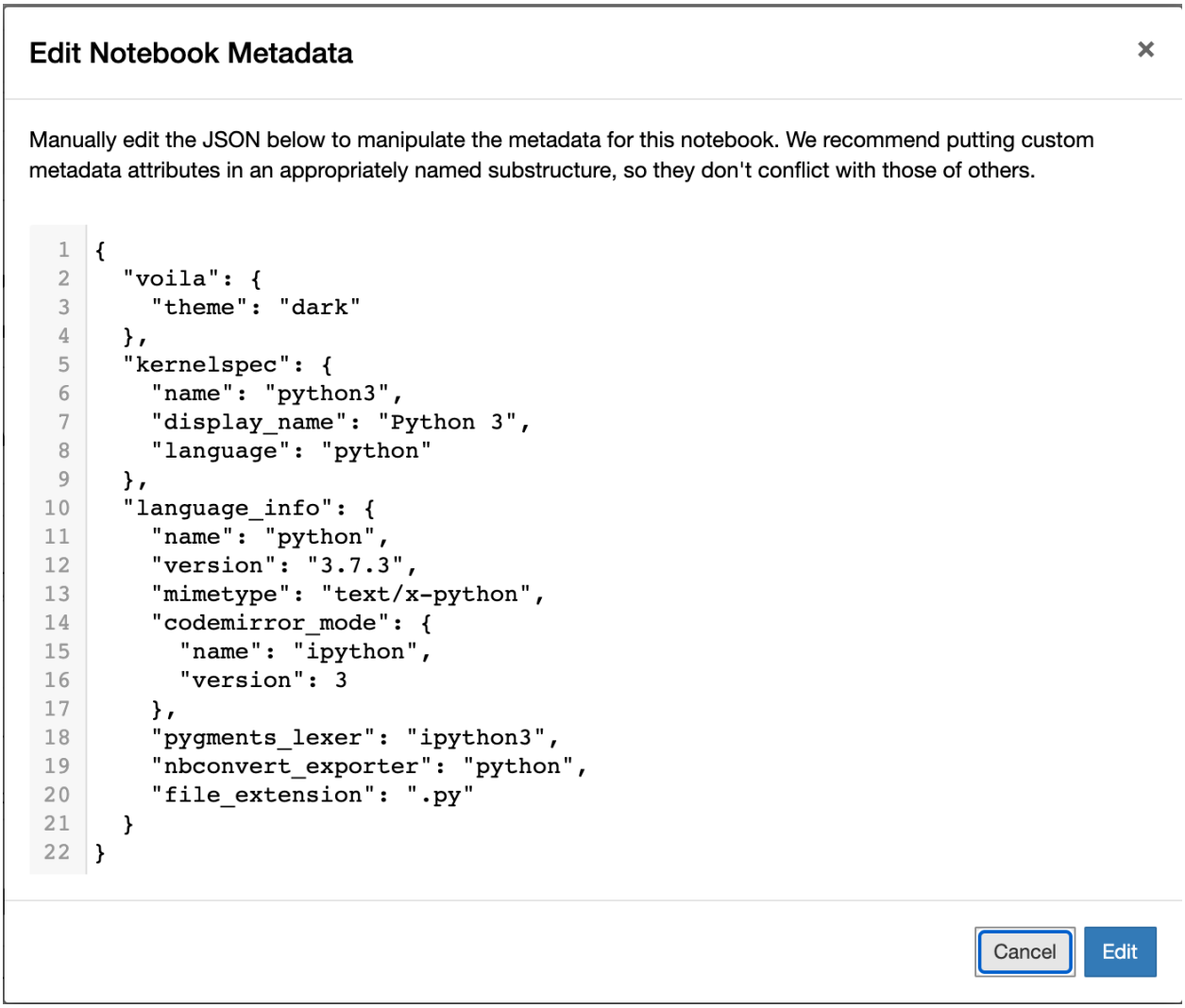
## 3.1 Changing the theme

By default, Voilà uses the **light** theme, but you can set the theme to **dark** by passing the following option:

```
voila <path-to-notebook> --theme=dark
```

Or by passing in the query parameter `voila-theme`, e.g. a URL like `http://localhost:8867/voila/render/query-strings.ipynb?voila-theme=dark`.

The theme can also set in the notebook metadata, under `metadata/voila/theme` by editing the notebook file manually, or using the metadata editor in for instance the classical notebook




System administrators who want to disable changing the theme, can pass `--VoilaConfiguration.allow_theme_override=NO` or ``--VoilaConfiguration.allow_theme_override=NOTEBOOK`` to disable changing the theme completely, or only allow it from the notebook metadata.

Currently, Voilà supports only **light** and **dark** themes.

**Note:** Changing the theme from the notebook metadata may change in the future if this features moves to nbconvert.

## 3.2 Controlling the nbconvert template

Voilà uses **nbconvert** to convert your Jupyter Notebook into an HTML dashboard. nbconvert has a rich templating system that allows you to customize the way in which your Jupyter Notebook is converted into HTML.

By default, Voilà will render the HTML from your notebook in the same linear fashion that the notebook follows. If you'd like to use a different layout, this can be controlled by creating a new nbconvert template, registering it with Voilà, and calling it from the command-line like so:

```
voila <path-to-notebook> --template=<name-of-template>
```

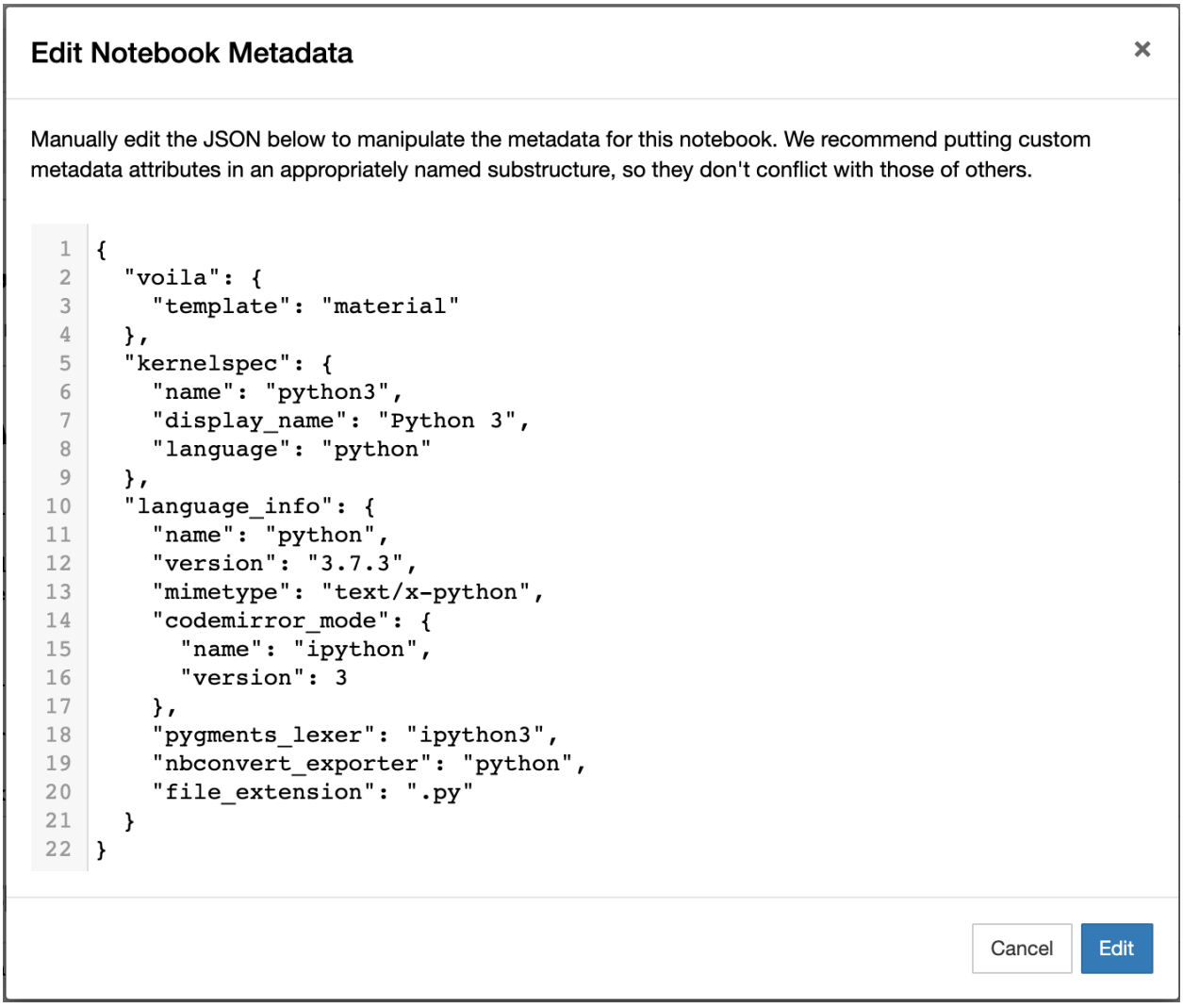
For example, Voilà includes one other template that uses a Javascript library and an alternate  layout in order to let the user drag and drop cells.

For example, to use the gridstack template, use the command:

```
voila <path-to-notebook> --template=gridstack
```

Or by passing in the query parameter `voila-template`, e.g. a URL like `http://localhost:8867/voila/render/query-strings.ipynb?voila-template=material` (Note that this requires installing voila-material).

The template can also set in the notebook metadata, under `metadata/voila/template` by editing the notebook file manually, or using the metadata editor in for instance the classical notebook




System administrators who want to disable changing the theme, can pass `--VoilaConfiguration.allow_template_override=NO` or ``--VoilaConfiguration.allow_template_override=NOTEBOOK` to disable changing the theme completely, or only allow it from the notebook metadata.

**Note:** Changing the template from the notebook metadata may change in the future if this features moves to nbconvert.

## 3.3 Creating your own template

You can create your own nbconvert template for use with Voilà. This allows you to control the look and feel of your dashboard.

In order to create your own template, first familiarize yourself with **Jinja**, **HTML**, and **CSS**. Each of these is used in creating custom templates. For more information, see the nbconvert templates documentation. For one example, check out the nbconvert basic HTML template.

A few example voila/nbconvert template projects are:

- https://github.com/voila-dashboards/voila-gridstack
- https://github.com/voila-dashboards/voila-material
- https://github.com/voila-dashboards/voila-vuetify

### 3.3.1 Where are Voilà templates located?

All Voilà templates are stored as folders with particular configuration/template files inside. These folders can exist in the standard Jupyter configuration locations, in a folder called `voila/templates`. For example:

```
~/.local/share/jupyter/voila/templates
~/path/to/env/dev/share/jupyter/voila/templates
/usr/local/share/jupyter/voila/templates
/usr/share/jupyter/voila/templates
```

Voilà will search these locations for a folder, one per template, where the folder name defines the template name.

### 3.3.2 The Voilà template structure

Within each template folder, you can provide your own nbconvert templates, static files, and HTML templates (for pages such as a 404 error). For example, here is the folder structure of the base Voilà template (called "default"):

```
tree path/to/env/share/jupyter/voila/templates/default/
├── nbconvert_templates
│   ├── base.tpl
│   └── voila.tpl
└── templates
    ├── 404.html
    ├── error.html
    ├── page.html
    └── tree.html
```

**To customize the nbconvert template**, store it in a folder called `templatename/nbconvert_templates/voila.tpl`. In the case of the default template, we also provide a `base.tpl` that our custom template uses as a base. The name `voila.tpl` is special - you cannot name your custom nbconvert something else.

**To customize the HTML page templates**, store them in a folder called `templatename/templates/<name>.html`. These are files that Voilà can serve as standalone HTML (for example, the `tree.html` template defines how folders/files are displayed in `localhost:8866/voila/tree`). You can override the defaults by providing your own HTML files of the same name.

**To configure your Voilà template**, you should add a `config.json` file to the root of your template folder.

### 3.3.3 An example custom template

To show how to create your own custom template, let's create our own nbconvert template. We'll have two goals:

1. Add an <h1> header displaying "Our awesome template" to the Voilà dashboard.
2. Add a custom 404.html page that displays an image.

First, we'll create a folder in `~/.local/share/jupyter/voila/templates` called `mytemplate`:

```
mkdir ~/.local/share/jupyter/voila/templates/mytemplate
cd ~/.local/share/jupyter/voila/templates/mytemplate
```

Next, we'll copy over the base template files for Voilà, which we'll modify:

```
cp -r path/to/env/share/jupyter/voila/templates/default/nbconvert_templates ./
cp -r path/to/env/share/jupyter/voila/templates/default/templates ./
```

We should now have a folder structure like this:

```
tree .
├── nbconvert_templates
│   ├── base.tpl
│   └── voila.tpl
└── templates
    ├── 404.html
    ├── error.html
    ├── page.html
    └── tree.html
```

Now, we'll edit `nbconvert_templates/voila.tpl` to include a custom H1 header.

As well as `templates/tree.html` to include an image.

Finally, we can tell Voilà to use this custom template the next time we use it on a Jupyter notebook by using the name of the folder in the `--template` parameter:

```
voila mynotebook.ipynb --template=mytemplate
```

The result should be a Voilà dashboard with your custom modifications made!

### 3.3.4 Voilà template cookiecutter

There is a Voilà template cookiecutter available to give you a running start. This cookiecutter contains some docker configuration for live reloading of your template changes to make development easier. Please refer to the cookiecutter repo for more information on how to use the Voilà template cookiecutter.

## 3.4 Adding your own static files

If you create your own theme, you may also want to define and use your own static files, such as CSS and Javascript. To use your own static files, follow these steps:

1. Create a folder along with your template (e.g., `mytemplate/static/`).
2. Put your static files in this template.
3. In your template file (e.g. `voila.tpl`), link these static files with the following path:

```
{{resources.base_url}}voila/static/<path-to-static-files>
```

4. When you call `voila`, configure the static folder by using the `--static` kwarg, or by configuring `--VoilaConfiguration.static_root`.

Any folders / files that are inside the folder given with this configuration will be copied to `{{resources.base_url}}voila/static/`.

For example, if you had a CSS file called `custom.css` in `static/css`, you would link it in your template like so:

```
<link rel="stylesheet" type="text/css" href="{{resources.base_url}}voila/static/css/
↪custom.css"></link>
```

## 3.5 Configure Voilà for the Jupyter Server

Several pieces of `voila`'s functionality can be controlled when it is run. This can be done either as a part of the standalone CLI, or with the Jupyter Server. To configure `voila` when run by the Jupyter Server, use the following pattern when invoking the command that runs Jupyter (e.g., Jupyter Lab or Jupyter Notebook):

```
<jupyter-command> --VoilaConfiguration.<config-key>=<config-value>
```

For example, to control the template used by `voila` from within a Jupyter Lab session, use the following command when starting the server:

```
jupyter lab --VoilaConfiguration.template=distill
```

When users run `voila` by hitting the `voila/` endpoint, this configuration will be used.

## 3.6 Serving static files

Unlike JupyterLab or the classic notebook server, `voila` does not serve all files that are present in the directory of the notebook. Only files that match one of the whitelists and none of the blacklist regular expression are served by Voilà:

```
voila mydir --VoilaConfiguration.file_whitelist="['.*']" \
  --VoilaConfiguration.file_blacklist="['private.*', '.*\.(ipynb)']"
```

Which will serve all files, except anything starting with private, or notebook files:

```
voila mydir --VoilaConfiguration.file_whitelist="['.*\.(png|jpg|gif|svg|mp4|avi|ogg)']"
```

Will serve many media files, and also never serve notebook files (which is the default blacklist).

## 3.7 Run scripts

Voilà can run text (or script) files, by configuring how a file extension maps to a kernel language:

```
voila mydir --VoilaConfiguration.extension_language_mapping='{".py": "python", ".jl":
↪"julia"}'
```

Voilà will find a kernel that matches the language specified, but can also be configured to use a specific kernel for each language:

```
voila mydir --VoilaConfiguration.extension_language_mapping='{".py": "python", ".jl":
↪"julia"}'\
  --VoilaConfiguration.language_kernel_mapping='{"python": "xpython"}'
```

In this case it will use the xeus-python. kernel to run *.py* files.

Note that the script will be executed as notebook with a single cell, meaning that only the last expression will be printed as output. Use the Jupyter display mechanism to output any text or rich output such as Jupyter widgets. For Python this would be a call to *IPython.display.display*.

Using Jupytext is another way to support script files. After installing jupytext, Voilà will see script files as if they are notebooks, and requires no extra configuration.

## 3.8 Cull idle kernels

Voilà starts a new Jupyter kernel every time a notebook is rendered to the user. In some situations, this can lead to a higher memory consumption.

The Jupyter Server exposes several options that can be used to terminate kernels that are not active anymore. They can be configured using the Voilà standalone app:

```
voila --MappingKernelManager.cull_interval=60 --MappingKernelManager.cull_idle_
↪timeout=120
```

The server will periodically check for idle kernels, in this example every 60 seconds, and cull them if they have been idle for more than 120 seconds.

The same parameters apply when using Voilà as a server extension:

```
jupyter notebook --MappingKernelManager.cull_interval=60 --MappingKernelManager.cull_
↪idle_timeout=120
```

There is also the `MappingKernelManager.cull_busy` and `MappingKernelManager.cull_connected` options to cull busy kernels and kernels with an active connection.

For more information about these options, check out the Jupyter Server documentation.

## 3.9 Hiding output and code cells based on cell tags

Voilà uses nbconvert under the hood to render the notebooks so we can benefit from some of its advanced functionalities to hide code and output cells based on cell tags.

To hide the cell output for every cell in your notebook that has been tagged (how to tag) with "hide" in Voilà:

```
voila --TagRemovePreprocessor.remove_all_outputs_tags='{"hide"}' your_notebook.ipynb
```

To hide both the code cell and the output cell (if any) for every cell that has been tagged with "hide":

```
voila --TagRemovePreprocessor.remove_cell_tags='{"hide"}' your_notebook.ipynb
```

You can use any tag you want but be sure to use the same tag name in the Voilà command. And please note that this functionality will only hide the cells in Voilà but will not prevent them from being executed.

## 3.10 Cell execution timeouts

By default, Voilà does not have an execution timeout, meaning there is no limit for how long it takes for Voilà to execute and render your notebook. If you have potentially long-running cells, you may wish to set a cell execution timeout so that users of your dashboard will get an error if it takes longer than expected to execute the notebook. For example:

```
voila --VoilaExecutor.timeout=30 your_notebook.ipynb
```

With this setting, if any cell takes longer than 30 seconds to run, a `TimeoutError` will be raised. You can further customize this behavior using the `VoilaExecutor.timeout_func` and `VoilaExecutor.interrupt_on_timeout` options.

CHAPTER

FOUR

# DEPLOYING VOILÀ

The deployment docs are split up in two parts. First there is the general section, which should always be followed. Then there is a cloud service provider specific section of which one provider should be chosen.

If you are not sure where to deploy your app, we suggest Binder or Heroku. You can test deploying and serving your app without having to enter any credit card details, and with very little prior experience of deployments.

## 4.1 Setup an example project

1. Create a project directory of notebooks you wish to display. For this tutorial we will clone Voilà and treat the notebooks folder as our project root.

```
git clone git@github.com:voila-dashboards/voila.git
cd voila/notebooks/
```

2. Add a requirements.txt file to the project directory. This file should contain all the Python dependencies your Voilà app needs to run. For this tutorial we will copy the contents of the environment.yml of Voilà. We omit xleaflet and xeus-cling because these require extra work that is beyond the scope of this guide.

```
bqplot
ipympl
ipyvolume
scipy
voila
```

## 4.2 Cloud Service Providers

### 4.2.1 Deployment on Binder

Binder is one of the most accessible ways to deploy Voilà applications. The service is available at mybinder.org and is increasingly being used for reproducible research, making it an excellent fit for deploying Voilà applications.

1. Make sure the repository is publicly available (on GitHub, Gitlab or as a gist).
2. Follow this guide to prepare the repository. For simple deployments, steps listed in *Setup an example project* will be sufficient.

**Note:** Binder also supports `environment.yml` files and `conda` environments.

3. Go to mybinder.org and enter the URL of the repository.
4. In `Path to a notebook file`, select `URL` and use the Voilà endpoint: `voila/render/path/to/notebook.ipynb`
5. Click `Launch`.
6. Binder will trigger a new build if this is the first launch (or if there has been new changes since the last build). This might take a few minutes to complete. If an image is already available, the server will be able to start within a few seconds.

#### Customizing Voilà on Binder

To specify different options (such as the theme and template), create a `jupyter_config.json` file at the root of the repository with the following content:

```
{
  "VoilaConfiguration": {
    "theme": "dark",
    "template": "gridstack"
  }
}
```

An example can be found in the voila-demo repository.

### 4.2.2 Deployment on Heroku

Heroku.com is an attractive option if you want to try out deployment for free. You have limited computing hours, however the app will also automatically shutdown if it is idle.

The general steps for deployment at Heroku can be found here. High level instructions, specific to Voilà can be found below:

1. Follow the steps of the official documentation to install the heroku cli and login on your machine.
2. Add a file named runtime.txt to the project directory with the following content:

   ```
   python-3.7.3
   ```

3. Add a file named Procfile to the project directory with the following content if you want to show all notebooks:

   ```
   web: voila --port=$PORT --no-browser
   ```

   Or the following if you only want to show one notebook:

   ```
   web: voila --port=$PORT --no-browser your_notebook.ipynb
   ```

4. Initialize your git repo and commit your code. At minimum you need to commit your notebooks, requirements.txt, runtime.txt, and the Procfile.

   ```
   git init
   git add <your-files>
   git commit -m "my message"
   ```

5. Create an Heroku instance and push the code.

```
heroku create
git push heroku master
```

6. Open your web app

```
heroku open
```

To resolve issues, it is useful to see the logs of your application. You can do this by running:

```
heroku logs --tail
```

### 4.2.3 Deployment on Google App Engine

You can deploy on Google App Engine in a "flexible" environment. This means that the underlying machine will always run. This is more expensive than a "standard" environment, which is similar to Heroku's free option. However, Google App Engine's "standard" environment does not support websockets, which is a requirement for voila.

The general steps for deployment at Google App Engine can be found here. High level instructions specific to Voilà can be found below:

1. Follow the "Before you begin steps" from the official documentation to create your account, project and App Engine app.
2. Add an app.yaml file to the project directory with the following content:

```
runtime: python
env: flex
runtime_config:
  python_version: 3
entrypoint: voila --port=$PORT --no-browser
```

3. Edit the last line if you want to show only one notebook

```
entrypoint: voila --port=$PORT --no-browser your_notebook.ipynb
```

4. Deploy your app

```
gcloud app deploy
```

5. Open your app

```
gcloud app browse
```

## 4.3 Running Voilà on a private server

### 4.3.1 Prerequisites

- A server running Ubuntu 18.04 (or later) with root access.
- Ability to SSH into the server and run commands from the prompt.
- The public IP address of the server.
- A domain name pointing to the IP address of the server.

### 4.3.2 Steps

1. SSH into the server:

```
ssh ubuntu@<ip-address>
```

2. Install nginx:

```
sudo apt install nginx
```

3. To check that `nginx` is correctly installed:

```
sudo systemctl status nginx
```

4. Create the file `/etc/nginx/sites-enabled/yourdomain.com` with the following content:

```
server {
    listen 80;
    server_name yourdomain.com;
    proxy_buffering off;
    location / {
            proxy_pass http://localhost:8866;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_read_timeout 86400;
    }

    client_max_body_size 100M;
    error_log /var/log/nginx/error.log;
}
```

5. Enable and start the `nginx` service:

```
sudo systemctl enable nginx.service
sudo systemctl start nginx.service
```

6. Install pip:

```
sudo apt update && sudo apt install python3-pip
```

7. Follow the instructions in *Setup an example project*, and install the dependencies:

```
sudo python3 -m pip install -r requirements.txt
```

8. Create a new systemd service for running Voilà in `/usr/lib/systemd/system/voila.service`. The service will ensure Voilà is automatically restarted on startup:

```
[Unit]
Description=Voila
```

(continues on next page)

(continued from previous page)

```
[Service]
Type=simple
PIDFile=/run/voila.pid
ExecStart=voila --no-browser voila/notebooks/basics.ipynb
User=ubuntu
WorkingDirectory=/home/ubuntu/
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

In this example Voilà is started with `voila --no-browser voila/notebooks/basics.ipynb` to serve a single notebook. You can edit the command to change this behavior and the notebooks Voilà is serving.

9. Enable and start the `voila` service:

```
sudo systemctl enable voila.service
sudo systemctl start voila.service
```

**Note:** To check the logs for Voilà:

```
journalctl -u voila.service
```

10. Now go to `yourdomain.com` to access the Voilà application.

### 4.3.3 Enable HTTPS with Let's Encrypt

1. Install `certbot`:

```
sudo add-apt-repository ppa:certbot/certbot
sudo apt update
sudo apt install python-certbot-nginx
```

2. Obtain the certificates from Let's Encrypt. The `--nginx` flag will edit the nginx configuration automatically:

```
sudo certbot --nginx -d yourdomain.com
```

3. `/etc/nginx/sites-enabled/yourdomain.com` should now contain a few more entries:

```
$ cat /etc/nginx/sites-enabled/yourdomain.com

...
listen 443 ssl; # managed by Certbot
ssl_certificate /etc/letsencrypt/live/yourdomain.com/fullchain.pem; #␣
↪managed by Certbot
ssl_certificate_key /etc/letsencrypt/live/yourdomain.com/privkey.pem; #␣
↪managed by Certbot
include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
...
```

4. Visit https://yourdomain.com to access the Voilà applications over HTTPS.
5. To automatically renew the certificates (they expire after 90 days), open the `crontab` file:

```
crontab -e
```

And add the following line:

```
0 12 * * * /usr/bin/certbot renew --quiet
```

For more information, you can also follow the guide on the nginx blog.

## 4.4 Sharing Voilà applications with ngrok

ngrok is a useful tool to expose local servers to the public internet over secure tunnels. It can be used to share Voilà applications served by a local instance of Voilà.

The main use case for using Voilà with ngrok is to quickly share a notebook as an interactive application without having to deploy to external hosting.

> **Warning:** Don't forget to exercise caution before exposing local apps and data to the public over the internet.
>
> While Voilà does not permit arbitrary code execution, be aware that sensitive information could be exposed, depending on the content and the logic of the notebook.
>
> It's good practice to keep the ngrok tunnel connection short-lived, and limit its use to quick sharing purposes.

### 4.4.1 Setup ngrok

To setup ngrok, follow the Download and setup ngrok guide.

### 4.4.2 Sharing Voilà applications

1. Start Voilà locally: `voila --no-browser my_notebook.ipynb`
2. In a new terminal window, start ngrok: `ngrok http 8866`
3. Copy the link from the ngrok terminal window. The links looks like the following: https://8bb6fded.ngrok.io/
4. Send the link
5. When using the ngrok link, the requests will be forwared to your local instance of Voilà.

CHAPTER

FIVE

# CONTRIBUTING TO VOILÀ

Voilà is a subproject of Project Jupyter and subject to the Jupyter governance and Code of conduct.

## 5.1 General Guidelines

For general documentation about contributing to Jupyter projects, see the Project Jupyter Contributor Documentation.

## 5.2 Community

The Voilà team organizes public video meetings. The schedule for future meetings and minutes of past meetings can be found on our team compass

## 5.3 Setting up a development environment

First, you need to fork the project. Then setup your environment:

```
# create a new conda environment
conda create -n voila -c conda-forge notebook nodejs
conda activate voila

# download voila from your GitHub fork
git clone https://github.com/<your-github-username>/voila.git

# install JS dependencies and build js assets
cd voila/js
npm install
cd ..

# install Voilà in editable mode
python -m pip install -e .
```

## 5.4 Run Voilà

To start Voilà, run:

```
voila
```

or

```
python -m voila
```

This will open a new browser tab at [http://localhost:8866/{]}(http://localhost:8866/).

When making changes to the frontend side of Voilà, open a new terminal window and run:

```
cd packages/voila/
npm run watch
```

Then reload the browser tab.

**Note**: the notebooks directory contains some examples that can be run with Voilà. Checkout the instructions in the user guide for details on how to run them.

## 5.5 Extensions

### 5.5.1 Server extension

To manually enable the classic notebook server extension:

```
jupyter serverextension enable voila --sys-prefix
```

For Jupyter Server:

```
jupyter server extension enable voila.server_extension --sys-prefix
```

This makes Voilà available as a server extension: http://localhost:8888/voila/tree.

### 5.5.2 Notebook extension

To install the notebook extension:

```
jupyter nbextension install voila --sys-prefix --py
jupyter nbextension enable voila --sys-prefix --py
```

### 5.5.3 JupyterLab extension

Node.js is required and can be installed with conda:

```
conda install -c conda-forge nodejs
```

The JupyterLab extension requires the server extension to be enabled. This can be done by running:

```
jupyter serverextension enable voila --sys-prefix
```

You can verify if the server extension is enabled by running:

```
jupyter serverextension list
```

If you use Jupyter Server:

```
jupyter server extension enable voila --sys-prefix
```

You can verify if the server extension is enabled by running:

```
jupyter server extension list
```

The JupyterLab extension is developed as a prebuilt extension using the new distribution system added in JupyterLab 3.0. To setup the development environment:

```
# install the package in development mode
python -m pip install -e .

# link your development version of the extension with JupyterLab
jupyter labextension develop . --overwrite

# build the lab extension
jlpm run build --scope @voila-dashboards/jupyterlab-preview

# it is also possible to start in watch mode to pick up changes automatically
jlpm run watch
```

### 5.5.4 Frontend Packages

The Voilà repository consists of several packages such as the Voilà frontend and the JupyterLab extension.

It follows a monorepo structure and uses `lerna` to streamline the workflow.

To build all the frontend packages at once, run the following commands:

```
# install dependencies
jlpm

# build the packages
jlpm run build
```

This will run the `build` script in each of the packages.

Using this structure, packages can easily be linted and follow the same code style and conventions used in other Jupyter projects. To lint the packages:

```
# install dependencies
jlpm

# run ESLint
jlpm run eslint

# run prettier
jlpm run prettier
```

## 5.6 Tests

Install the test dependencies

```
python -m pip install -e ".[test]"
```

Enable the Jupyter server extension:

```
jupyter server extension enable voila.server_extension --sys-prefix
```

Running the tests locally also requires the *test_template* to be installed:

```
python -m pip install ./tests/test_template
```

Finally, to run the tests:

```
python -m pytest
```

## 5.7 Editing templates

The default template files are located in the folder *share/jupyter/voila/templates/default*. They are automatically picked up when running Voilà in development mode.

After editing the templates, reload the browser tab to see the changes.