Contents:

1 What is apparate? 3
2 Installation 5
3 Quickstart 7
4 Table of Contents 9
   4.1 Getting Started ................................................. 9
   4.2 Apparate .............................................................. 9
   4.3 Tutorials .............................................................. 11
   4.4 Contributing .......................................................... 12
   4.5 Contributor Covenant Code of Conduct ......................... 14
5 Indices and tables 17
Make your libraries magically appear in Databricks.
What is apparate?

Apparate is a tool to manage libraries in Databricks in an automated fashion. It allows you to move away from the point-and-click interface for your development work and for deploying production-level libraries for use in scheduled Databricks jobs. To learn more, check out Why did we build this?.
Installation

Apparate is hosted on PyPi, so to get the latest version simply install via pip:

```
pip install apparatus
```

You can also install from source, by cloning the git repository https://github.com/ShopRunner/apparate.git and installing via easy_install:

```
git clone https://github.com/ShopRunner/apparate.git
cd apparatus
easy_install .
```
To get started, first run `apparate configure` and answer the questions.

Then you are ready to upload libraries to Databricks, using the `apparate upload` and `apparate upload_and_update` commands.

Please see *Getting Started* for an introduction to the package, and *Apparate* for specifics on available options.
4.1 Getting Started

4.1.1 Why did we build this?

When our team started setting up CI/CD for the various packages we maintain, we encountered some difficulties integrating Jenkins with Databricks.

We write a lot of Python + PySpark packages in our data science work, and we often deploy these as batch jobs run on a schedule using Databricks. However, each time we merged in a new change to one of these libraries we would have to manually create an egg, upload it using the Databricks GUI, go find all the jobs that used the library, and update each one to point to the new job. As our team and set of libraries and jobs grew, this became unsustainable (not to mention a big break from the CI/CD philosophy . . . ).

As we set out to automate this using Databrick’s library API, we realized that this task required using two versions of the API and many dependant API calls. Instead of trying to recreate that logic in each Jenkinsfile, we wrote apparate. Now you can enjoy the magic as well!

To get started, check out Installation or Quickstart.

To learn more about how to use apparate, check out Tutorials or Apparate.

To help improve apparate, check out Contributing.

4.2 Apparate

The apparate cli is your point of contact for managing continuous delivery of Python packages for use in Databricks.

4.2.1 Configure

To get started, configure your Databricks account information. You’ll need your Databricks account connection info, and you will also be asked to name a production folder. To learn more about how these values will be used and where
to find this information, check out the Getting Started page.

When you’re ready to go, run `apparate configure`.

```bash
$ apparate configure --help
Usage: apparate configure [OPTIONS]

  Configure information about Databricks account and default behavior.

  Configuration is stored in a `.apparatecfg` file. A config file must exist
  before this package can be used, and can be supplied either directly as a
text file or generated using this configuration tool.

Options:
  --help  Show this message and exit.
```

Now you’re all set to start using apparate! The two main commands available in apparate are `upload` and
`upload_and_update`.

4.2.2 Upload

`upload` can be used anytime by anyone and promises not break anything. It simply uploads an egg file, and will
throw an error if a file with the same name already exists.

If you’ve set up your `.apparatecfg` file using the `configure` command, you only need to provide a path to the
`.egg` file, but can also override the default api token and destination folder if desired.

If you try to upload a library to Databricks that already exists there with the same version, a warning will be printed
instructing the user to update the version if a change has been made. Without a version change the new library will not
be uploaded.

This command will print out a message letting you know the name of the egg that was uploaded.

```bash
$ apparate upload --help
Usage: apparate upload [OPTIONS]

  The egg that the provided path points to will be uploaded to Databricks.

Options:
  -p, --path TEXT  path to egg file with name as output from setuptools
                   (e.g. dist/new_library-1.0.0-py3.6.egg) [required]
  -t, --token TEXT  Databricks API key - optional, read from `.apparatecfg`
                   if not provided
  -f, --folder TEXT  Databricks folder to upload to (e.g.
                     '/Users/my_email@fake_organization.com') - optional, read
                     from `.apparatecfg` if not provided
  --help           Show this message and exit.
```

4.2.3 Upload and Update

`upload_and_update` requires a token with admin-level permissions. It does have the capacity to delete libraries,
but if used in a CI/CD system will not cause any issues. For advice on how to set this up, check out the Getting Started
page.

Used with default settings, `upload_and_update` will start by uploading the `.egg` file. It will then go find all jobs
that use the same major version of the library and update them to point to the new version. Finally, it will clean up
outdated versions in the production library. No libraries in any other folders will ever be deleted.
If you’re nervous about deleting files, you can always use the `--no-cleanup` flag and no files will be deleted or overwritten. If you’re confident in your CI/CD system, however, leaving the cleanup variable set to `True` will keep your production folder tidy, with only the most current version of each major release of each library.

This command will print out a message letting you know (1) the name of the egg that was uploaded, (2) the list of jobs currently using the same major version of this library, (3) the list of jobs updated - this should match number 2, and (4) any old versions removed - if you haven’t used the `--no-cleanup` flag.

In the same way as `upload`, if you try to upload a library to Databricks that already exists there with the same version, a warning will be printed instructing the user to update the version if a change has been made. Without a version change the new library will not be uploaded.

```bash
$ apparate upload_and_update --help
Usage: apparate upload_and_update [OPTIONS]

The egg that the provided path points to will be uploaded to Databricks. All jobs which use the same major version of the library will be updated to use the new version, and all version of this library in the production folder with the same major version and a lower minor version will be deleted.

Unlike `upload`, `upload_and_update` does not ask for a folder because it relies on the production folder specified in the config. This is to protect against accidentally updating jobs to versions of a library still in testing/development.

All egg names already in Databricks must be properly formatted with versions of the form `<name>-0.0.0`.

Options:
- `-p, --path TEXT` path to egg file with name as output from `setuptools` (e.g. `dist/new_library-1.0.0-py3.6.egg`) `[required]`
- `-t, --token TEXT` Databricks API key with admin permissions on all jobs using library - optional, read from `~.apparatecfg` if not provided
- `--cleanup / --no-cleanup` if cleanup, remove outdated files from production folder; if no-cleanup, remove nothing [default: `True`]
- `--help` Show this message and exit.

For more info about usage, check out the `Tutorials`.

### 4.3 Tutorials

#### 4.3.1 For testing and development

When developing libraries, we often found it frustrating to frequently re-upload a library that was changing daily as we worked out a new feature. With apparate, this workflow is much simpler.

When you are ready to test out changes to your library, start by deleting the current version. (Unfortunately moving or renaming the old version is insufficient, and it must be fully deleted AND removed from the trash folder before the cluster will recognize the new copy). Next restart your cluster, so it wipes the old version from its imports.

Create a new egg file from your package. In the top level of your package, run:
python setup.py bdist_egg

Upload the library to your preferred development folder:

```bash
apparate upload -p ./dist/my_library-1.0.1-py3.6.egg -f /Users/my_email@fake_organization.com/dev_folder
```

Finally, attach the new library to your cluster, and you’re ready to test away!

### 4.3.2 For production libraries

While useful for testing libraries, the real reason we wrote this package involved frustrations we encountered building out our continuous integration/continuous deployment infrastructure. If you are using a CI/CD setup with tools such as Jenkins or Travis, apparate works in these tools to cleanly integrate your Python packages with production jobs in Databricks. As we use Jenkins here at ShopRunner to manage CI/CD, I will continue with that example, but this should work in any similar tool.

First, you will need a Databricks token with admin permission accessible in Jenkins, here represented by the environment variable `TOKEN`. You also need to set up the `.apparatecfg` file. While the `apparate configure` tool makes this easy to do locally, in an automated setup it’s often easier to provide the file directly, using a command like:

```bash
echo """"[DEFAULT]
host = https://my-organization.cloud.databricks.com
token = ${TOKEN}
prod_folder = /Shared/production_libraries"""" > ~/.apparatecfg
```

A standard Jenkinsfile for one of our Python packages will run a linting tool, run unittests, push the egg to our artifact store, and then use apparate to push the egg to Databricks. This final steps works as follows:

```bash
apparate upload_and_update -p `ls dist/*.egg`
```

The `ls dist/*.egg` lists the egg files in the `dist` subfolder (which should just be the egg you want to upload).

We’ve also found it useful to redirect the printed statements to a Slack channel, so we get notifications when jobs are updated. This makes it easy to diagnose which library version caused problems if jobs ever fail.

For more details on options available with these two commands, check out [Apparate](#).

### 4.4 Contributing

#### 4.4.1 How to Contribute

We welcome contributions in the form of issues or pull requests!

We want this to be a place where all are welcome to discuss and contribute, so please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms. Find the code of conduct [below](#) or in the `code_of_conduct.md` file on GitHub.

If you have a problem using apparate or see a possible improvement, open an issue in the GitHub issue tracker. Please be as specific as you can.

If you see an open issue you’d like to be fixed, take a stab at it and open a PR!
Steps for making a pull request:

1. Fork the project from GitHub
2. Clone the forked repo to your local disk:
   ```shell
git clone https://github.com/<your_github_user_name>/apparate.git
```
3. Create a new branch:
   ```shell
git checkout -b my_awesome_new_feature
```
4. Install requirements (virtualenvs always recommended!):
   ```shell
   pip install -r requirements-dev.txt
   ```
5. Write some awesome useful code
6. Update unitests, docs, and CHANGELOG - to view docs locally:
   ```bash
cd docs/
make docs
open _build/html/index.html
```
7. Double-check that unitests pass and the linter doesn’t complain:
   ```bash
   pytest
   flake8 apparate tests
   ```
8. Submit a PR! Please submit your PR against the WIP branch - someone will review your code, then when it is ready merge it into WIP. Once there tests and linting will be run automatically, and once that is verified we will merge WIP into master.

Note: several of the tests rely on the .apparatecfg file, so make sure to run `apparate configure` before running tests. If you want to run tests using a different token than is in your .apparatecfg file, you can also pass in the values directly, as shown in the second example. Values passed as options will override those in the config.

To run unitests with using defaults in .apparatecfg:

```bash
pytest
```

To run unitests with a different token:

```bash
pytest --token abc123
```

Warning: tests in `test_token_permissions` make actual API calls. They only make read calls, but do require an internet connection. To run only tests that are isolated, use:

```bash
pytest --deselect tests/test_token_permissions.py
```

This package follows PEP8 standards, uses numpy-type docstrings, and should be tested in python3.
4.5 Contributor Covenant Code of Conduct

4.5.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

4.5.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others’ private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

4.5.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

4.5.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.
4.5.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at htorrence@shoprunner.com. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project’s leadership.

4.5.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html
CHAPTER 5

Indices and tables

• genindex
• search