Orchest

Release 0.1.0

Jul 07, 2020

Contents:

Website – GitHub – Community forum

Orchest is an open source, cloud native, development environment built for data science. Orchest enables you to develop, train and run your models on the cloud without any knowledge of cloud infrastructure.

CHAPTER 1

Orchest overview

Orchest is an open source, cloud native, development environment build for data science. Orchest enables you to develop, train and run your models on the cloud without any knowledge of cloud infrastructure.

1.1 The Orchest platform

Orchest is not here to reinvent the wheel when it comes to your favorite editor, Orchest is a web based platform that works on top of your filesystem (similar to JupyterLab) allowing you to use your editor of choice. With Orchest you get to focus on visually building and iterating on your pipelining ideas.

A pipeline in Orchest can be thought of as a graph consisting of executable files, e.g. notebooks or scripts, within their own isolated environment (powered by containerization). Users get a visual pipeline editor to describe the execution order of individual steps that represent those executable files. After coding your scripts, Orchest allows you to select and run any subset of the steps whilst keeping in mind the defined execution order of the pipeline.

Orchest essentially provides your with a development environment for your data science efforts without taking away the tools you know and love.

1.2 What can I use Orchest for?

With Orchest, you get to build pipelines where each step has its own isolated environment allowing you to focus on a specific task, may it be: data engineering, model building or more low level things such as data transforms.

With Orchest you get to:

- Visually construct pipelines.
- Code your data science efforts in your editor of choice.
- Modularize, i.e. split up, your (monolithic) notebooks.
- Run any selection of pipeline steps.

- Select specific notebook cells to skip when running a pipeline through the pre-installed celltags extension of JupyterLab.

What Orchest does for you:

- Provide you with an interactive pipeline editing view.
- Manage your dependencies and environments.
- Run your pipelines based on the defined execution order.
- Pass data between your steps.

1.3 Orchest roadmap

Orchest is just beginning to take shape. In the near future you can expect the following features:

- Scheduled experiments by parametrizing your pipeline steps.
- Managed hosted version to easily try out the Orchest platform.
- Integration to load in your existing projects from GitHub. Note that you can already setup Orchest for an existing project on your filesystem.
- Alternatives for on disk data passing, e.g. in-memory powered by Apache Arrow.

CHAPTER 2

Installation

Orchest can be run locally on Windows, macOS and Linux.

2.1 Docker access

The run scripts (orchest.sh/orchest.bat) will mount the Docker socket to the `orchest-ctl` container to manage the local Docker containers necessary for running Orchest. In addition, the Docker socket is necessary for the dynamic spawning of containers that occurs when running individual pipeline steps.

2.2 Requirements

- Docker (tested on 19.03.9)

2.3 Installation steps

Linux/macOS

```
git clone https://github.com/orchest/orchest.git
cd orchest
./orchest.sh start
```

Windows

```
git clone https://github.com/orchest/orchest.git
cd orchest
orchest start
```

Note! On Windows, make sure to give Docker permission to mount the directory in which you cloned Orchest. For more details check the Windows Docker documentation (*Docker settings* > *Resources* > *File sharing* > Add directory that contains Orchest).

CHAPTER 3

Quickstart

3.1 General workflow in Orchest

1. Build your pipeline. Visually build your pipeline: create, drag and connect steps.
2. Write your code. Use your editor of choice or use the integrated JupyterLab environment in Orchest.
3. Run your pipeline. Happy with your implementation? Go back to the pipeline view and run your pipeline.
4. Check your output (either inside your notebook or the logs view for scripts).

3.2 Your first project

You can follow the next example.

Launch Orchest:

```
./orchest.sh start
```

On Windows:

```
orchest.bat start
```

Build your pipeline. Create two steps and connect them.

Directly edit your pipeline steps.

Run your pipeline and see the results come in.

CHAPTER 4

How Orchest works

Orchest is powered by your filesystem. Upon launching, Orchest will mount a directory called the *userdir*. Its default location is `orchest/orchest/userdir/`. Inside this directory it will store the following files *for each pipeline*:

- Your scripts that make up the pipeline, for example `.ipynb` files.
- The Orchest *Data passing SDK* stores step outputs in the `.data` directory to pass data between pipeline steps.
- Logs are stored in `.logs` to show STDOUT output from scripts in the pipeline view.
- An autogenerated *pipeline.json* file that defines the properties of the pipeline and its steps. This includes: execution order, names, images, etc. Orchest needs this pipeline definition file to work.

Orchest runs as a collection of Docker containers and only stores a global configuration file. The location for this config is `~/.config/orchest/config.json` for Unix based systems and `%UserProfile%\.orchest\config.json` for Windows.

4.1 Installing additional packages

Orchest runs all your pipeline step code scripts (.ipynb, .py, .R, .sh) in containers. The default images are based on the and come with a number of .

We plan on supporting custom images and/or container commits, to avoid having to reinstall packages each time a pipeline step is run.

4.1.1 Installing additional Python packages

Execute commands inside the scripts to install the package before use.

For Jupyter notebooks you can run the following code in a cell:

```
!conda install <package name>
```

or for the `pip` packages run:

```
!pip install <package name>
```

Or directly from within Python (i.e. for Python scripts):

```
from pip._internal import main as pip

pip(['install', '--user', '<package name>'])
```

4.1.2 Installing additional R packages

R packages can be installed with the regular command:

```
install.packages("<package name>")
```

CHAPTER 5

Data passing SDK

Full documentation of the Orchest SDK can be found here: orchest-sdk.

Orchest SDK for data passing between pipeline steps in the Orchest platform. The SDK manages the destination and source of the data, leaving the user only with the decision what data to send (because receiving automatically retrieves all the sent data).

The destination and source of the data are inferred through the defined pipeline definition in the platform (the *pipeline.json* file).

We plan to also support other popular programming languages such as R.

5.1 Python SDK

Python package to pass data between pipeline steps in the Orchest platform.

5.1.1 Installation

Note: The SDK comes pre-installed when using the Orchest platform.

Currently the recommended method for installing the Orchest SDK is through the GitHub repository using `pip`

```
# To get the latest release you can substitute "master" for "develop".
pip install git+https://github.com/orchest/orchest-sdk.git@master#subdirectory=python
```

5.1.2 Code example

Example of passing data, where the pipeline (defined inside the `pipeline.json`) is *Step 1 -> Step 2*.

```
"""Step 1"""
from orchest import transfer

data = [1, 2, 3]

# Output the data so that Step 2 can retrieve it.
transfer.output(data)
```

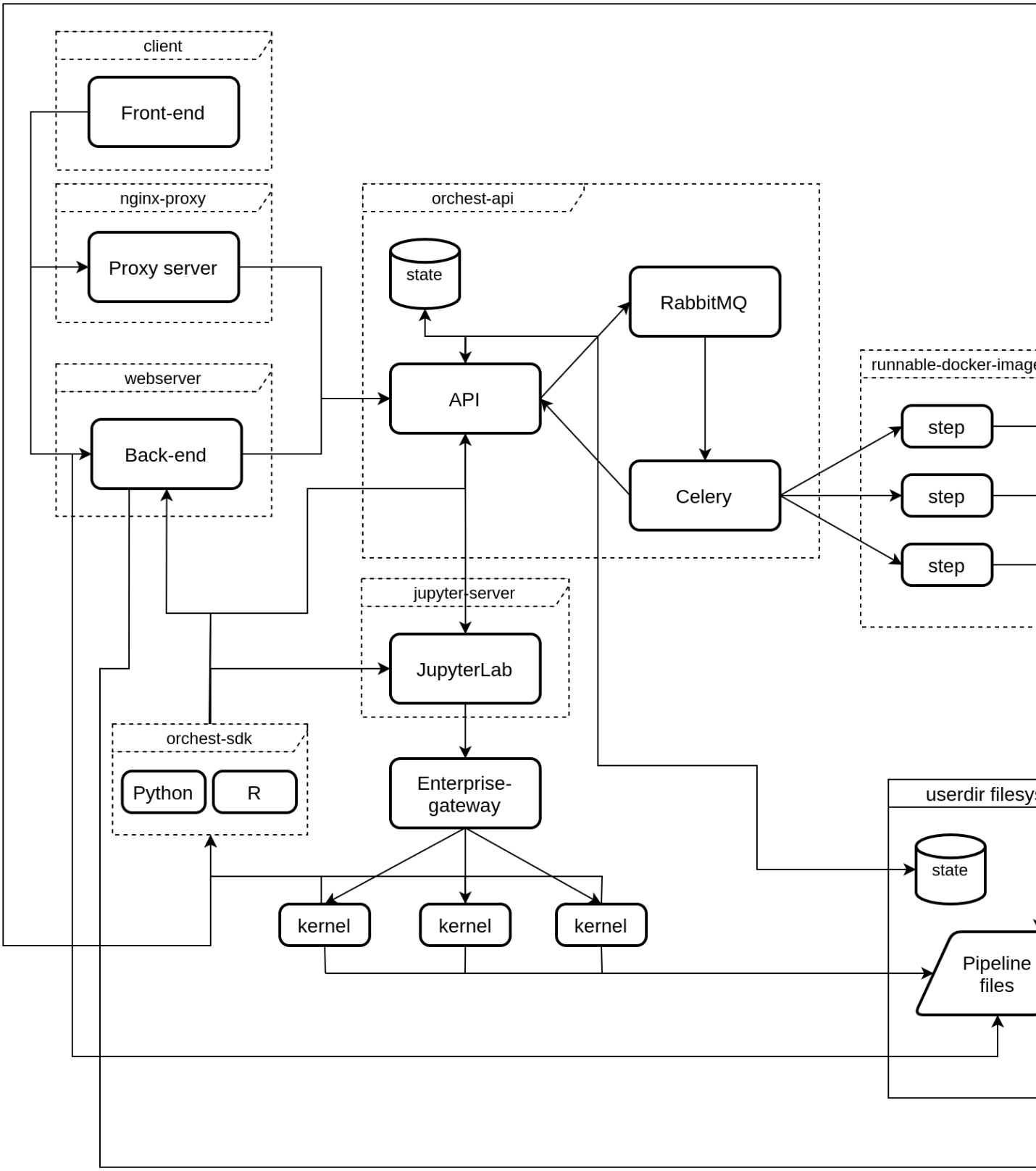
```
"""Step 2"""
from orchest import transfer

# Get the input for Step 2, i.e. the output of Step 1.
data = transfer.get_inputs()  # data = [[1, 2, 3]]
```

CHAPTER 6

Development

We would love to hear your feedback to prioritize the right features. Come talk to us at Community forum.

6.1 Orchest architecture



6.2 Contributer guides

6.2.1 Contributor License Agreement

The CLA ensures that Orchest has clear ownership specification for all contributions, which in turns lets us guarantee to users that we have no "stray" intellectual property or differently-licensed material.

6.2.2 Development environment

To start hacking on Orchest clone the repo from GitHub. You can find useful scripts for development in the `dev-utils` directory.

```
git clone https://github.com/orchest/orchest.git
```

Feel free to pick up any of the issues on GitHub or create a pull request for your own feature.

CHAPTER 7

Need help?

For questions come talk to us on the Community forum. If you think you've found a bug we encourage you to report it on GitHub.