bookstore Documentation

Release 2.5.2dev0

nteract project

Dec 09, 2019

Contents

Release v2.5.2dev0 (*What's new?*).

bookstore provides tooling and workflow recommendations for storing, scheduling, and publishing notebooks.

CHAPTER 1

Table of Contents

1.1 Installation

bookstore may be installed using Python 3.6 and above.

After installation, bookstore can process Python 2 or Python 3 notebooks.

1.1.1 Install from PyPI (recommended)

```
python3 -m pip install bookstore
```

1.1.2 Install from conda-forge

```
conda install -c conda-forge bookstore
```

1.1.3 Install from Source

1. Clone this repo:

   ```
   git clone https://github.com/nteract/bookstore.git
   ```

2. Change directory to repo root:

   ```
   cd bookstore
   ```

3. Install dependencies:

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

4. Install package from source:

```
python3 -m pip install .
```

Tip: Don't forget the dot at the end of the command

1.2 Configuration

Commonly used configuration settings can be stored in `BookstoreSettings` in the `jupyter_notebook_config.py` file. These settings include:

- workspace location
- published storage location
- S3 bucket information
- AWS credentials for S3

1.2.1 Example configuration

Here's an example of `BookstoreSettings` in the `~/.jupyter/jupyter_notebook_config.py` file:

```
"""jupyter notebook configuration
The location for user installs on MacOS is ``~/.jupyter/jupyter_notebook_config.py``.
See https://jupyter.readthedocs.io/en/latest/projects/jupyter-directories.html for␣
↪additional locations.
"""
from bookstore import BookstoreContentsArchiver


c.NotebookApp.contents_manager_class = BookstoreContentsArchiver

c.BookstoreSettings.workspace_prefix = "/workspace/kylek/notebooks"
c.BookstoreSettings.published_prefix = "/published/kylek/notebooks"

c.BookstoreSettings.s3_bucket = "<bucket-name>"

# If bookstore uses an EC2 instance with a valid IAM role, there is no need to␣
↪specify here
c.BookstoreSettings.s3_access_key_id = <AWS Access Key ID / IAM Access Key ID>
c.BookstoreSettings.s3_secret_access_key = <AWS Secret Access Key / IAM Secret Access␣
↪Key>
```

The root directory of bookstore's GitHub repo contains an example config called `jupyter_config.py.example` that shows how to configure `BookstoreSettings`.

1.3 Usage

Data scientists and notebook users may develop locally on their system or save their notebooks to off-site or cloud storage. Additionally, they will often create a notebook and then over time make changes and update it. As they work, it's helpful to be able to **store versions** of a notebook. When making changes to the content and calculations over time, a data scientist using Bookstore can now request different versions from the remote storage, such as S3, and **clone** the notebook to their local system.

Note: store and clone

store

User saves to Local System ——————> Remote Data Store (i.e. S3)

clone

User requests a notebook to use locally <————— Remote Data Store (i.e. S3)

After some time working with a notebook, the data scientist may want to save or share a polished notebook version with others. By **publishing a notebook**, the data scientist can display and share work that others can use at a later time.

1.3.1 How to store and clone versions

Bookstore uses automatic notebook version management and specific storage paths when storing a notebook.

Automatic notebook version management

Every *save* of a notebook creates an *immutable copy* of the notebook on object storage. Initially, Bookstore supports S3 for object storage.

To simplify implementation and management of versions, we currently rely on S3 as the object store using versioned buckets. When a notebook is saved, it overwrites the existing file in place using the versioned s3 buckets to handle the versioning.

Storage paths

All notebooks are archived to a single versioned S3 bucket using specific **prefixes** to denote a user's workspace and an organization's publication of a user's notebook. This captures the lifecycle of the notebook on storage. To do this, bookstore allows users to set workspace and published storage paths. For example:

- `/workspace` - where users edit and store notebooks
- `/published` - notebooks to be shared to an organization

Bookstore archives notebook versions by keeping the path intact (until a user changes them). For example, the prefixes that could be associated with storage types:

- Notebook in "draft" form: `/workspace/kylek/notebooks/mine.ipynb`
- Most recent published copy of a notebook: `/published/kylek/notebooks/mine.ipynb`

Note: *Scheduling (Planned for a future release)*

When scheduling execution of notebooks, each notebook path is a namespace that an external service can access. This helps when working with parameterized notebooks, such as with Papermill. Scheduled notebooks may also be referred to by the notebook `key`. In addition, Bookstore can find version IDs as well.

Easing the transition to Bookstore's storage plan

Since many people use a regular filesystem, we'll start with writing to the `/workspace` prefix as Archival Storage (more specifically, writing on save using a `post_save_hook` for the Jupyter contents manager).

1.3.2 How to publish a notebook

To publish a notebook, Bookstore uses a publishing endpoint which is a `serverextension` to the classic Jupyter server. If you wish to publish notebooks, explicitly enable bookstore as a server extension to use the endpoint. By default, publishing is not enabled.

To enable the extension globally, run:

```
jupyter serverextension enable --py bookstore
```

If you wish to enable it only for your current environment, run:

```
jupyter serverextension enable --py bookstore --sys-prefix
```

1.4 REST API

GET /api/bookstore
Info about bookstore

Status Codes

- 200 OK – Successfully requested

GET /bookstore/clone
Landing page for initiating cloning.

This serves a simple html page that allows avoiding xsrf issues on a jupyter server.

Query Parameters

- **s3_bucket** (*string*) – S3_bucket being targeted (Required)
- **s3_key** (*string*) – S3 object key being requested (Required)
- **s3_version_id** (*string*) – S3 object key being requested

Status Codes

- 200 OK – successful operation
- 400 Bad Request – Must have a key to clone from

POST /api/bookstore/clone
Trigger clone from s3

Status Codes

- 200 OK – Successfully cloned
- 400 Bad Request – Must have a key to clone from

GET /bookstore/fs-clone
Landing page for initiating file-system cloning.

This serves a simple html page that allows avoiding xsrf issues on a jupyter server.

Query Parameters

- **`relpath`** (*`string`*) – relative path being targeted (Required)

Status Codes

- 200 OK – successful operation
- 400 Bad Request – Request malformed, must provide a relative path.
- 404 Not Found – Request to clone from a path outside of base directory

`POST /api/bookstore/fs-clone`
Trigger clone from file system

Status Codes

- 200 OK – Successfully cloned
- 400 Bad Request – Malformed request. Provide a valid relative path.
- 404 Not Found – Invalid request. Cloning from a path outside of the base directory is not allowed.

`PUT /api/bookstore/publish/{path}`
Publish a notebook to s3

Parameters

- **`path`** (*`string`*) – Path to publish to, it will be prefixed by the preconfigured published bucket.

Status Codes

- 200 OK – Successfully published.

1.5 Reference

1.5.1 Configuration

Bookstore may be configured by providing `BookstoreSettings` in the `~/.jupyter/jupyter_notebook_config.py` file.

The `bookstore_config` module

`BookstoreSettings`

These settings are configurable by the user. Bookstore uses the traitlets library to handle the configurable options.

class `bookstore.bookstore_config.`**`BookstoreSettings`**(***kwargs*)
Configuration for archival and publishing.

Settings include storage directory locations, S3 authentication, additional S3 settings, and Bookstore resources.

S3 authentication settings can be set, or they can be left unset when IAM is used.

Like the Jupyter notebook, bookstore uses traitlets to handle configuration, loading from files or CLI.

`workspace_prefix`
Directory to use for user workspace storage

Type str(`workspace`)

`published_prefix`
Directory to use for published notebook storage

Type str(`published`)

`s3_access_key_id`
Environment variable `JPYNB_S3_ACCESS_KEY_ID`

Type str, optional

`s3_secret_access_key`
Environment variable `JPYNB_S3_SECRET_ACCESS_KEY`

Type str, optional

`s3_endpoint_url`
Environment variable `JPYNB_S3_ENDPOINT_URL`

Type str(`"https://s3.amazonaws.com"`)

`s3_region_name`
Environment variable `JPYNB_S3_REGION_NAME`

Type str(`"us-east-1"`)

`s3_bucket`
Bucket name, environment variable `JPYNB_S3_BUCKET`

Type str(`""`)

`max_threads`
Maximum threads from the threadpool available for S3 read/writes

Type int(`16`)

`enable_s3_cloning`
Enable cloning from s3.

Type bool(`True`)

`fs_cloning_basedir`
Absolute path to base directory used to clone from the local file system

Type str(`"/Users/jupyter"`)

Functions

These functions will generally be used by developers of the bookstore application.

`bookstore.bookstore_config.`**`validate_bookstore`**(*settings: bookstore.bookstore_config.BookstoreSettings*)
Check that settings exist.

Parameters **`settings`** (`bookstore.bookstore_config.BookstoreSettings`) – Instantiated settings object to be validated.

Returns **validation_checks** – Statements about whether features are validly configured and available

Return type dict

1.5.2 Archiving

The `archive` module

The `archive` module manages archival of notebooks to storage (i.e. S3) when a notebook save occurs.

`ArchiveRecord`

Bookstore uses an immutable `ArchiveRecord` to represent a notebook file by its storage path.

class `bookstore.archive.`**`ArchiveRecord`**

Represents an archival record.

An *ArchiveRecord* uses a Typed version of *collections.namedtuple()*. The record is immutable.

Example

An archive record (*filepath*, *content*, *queued_time*) contains:

- a *filepath* to the record
- the *content* for archival
- the *queued time* length of time waiting in the queue for archiving

`content`
Alias for field number 1

`filepath`
Alias for field number 0

`queued_time`
Alias for field number 2

`BookstoreContentsArchiver`

class `bookstore.archive.`**`BookstoreContentsArchiver`**(**args*, ***kwargs*)

Manages archival of notebooks to storage (S3) when notebook save occurs.

This class is a custom Jupyter FileContentsManager which holds information on storage location, path to it, and file to be written.

Example

- Bookstore settings combine with the parent Jupyter application settings.
- A session is created for the current event loop.
- To write to a particular path on S3, acquire a lock.
- After acquiring the lock, *archive* method authenticates using the storage service's credentials.
- If allowed, the notebook is queued to be written to storage (i.e. S3).

`path_locks`
Dictionary of paths to storage and the lock associated with a path.

Type dict

`path_lock_ready`
A mutex lock associated with a path.

Type asyncio mutex lock

`archive` (*record: bookstore.archive.ArchiveRecord*)
Process a record to write to storage.

Acquire a path lock before archive. Writing to storage will only be allowed to a path if a valid *path_lock* is held and the path is not locked by another process.

Parameters **`record`** (`ArchiveRecord`) – A notebook and where it should be written to storage

`run_pre_save_hook` (*model*, *path*, ***kwargs*)
Send request to store notebook to S3.

This hook offloads the storage request to the event loop. When the event loop is available for execution of the request, the storage of the notebook will be done and the write to storage occurs.

Parameters

- **`model`** (*`dict`*) – The type of file and its contents
- **`path`** (*`str`*) – The storage location

1.5.3 API Handlers

The `handlers` module

`BookstoreVersionHandler`

class `bookstore.handlers.`**`BookstoreVersionHandler`**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs*)

Bases: `notebook.base.handlers.APIHandler`

Handler responsible for Bookstore version information

Used to lay foundations for the bookstore package. Though, frontends can use this endpoint for feature detection.

`get` (*self*)
Provides version info and feature availability based on serverside settings.

`build_response_dict` (*self*)
Helper to populate response.

`build_response_dict` ()
Helper for building the version handler's response before serialization.

`get` ()
GET /api/bookstore/

Returns version info and validation info for various bookstore features.

Jupyter Server extension

`bookstore.handlers.`**`load_jupyter_server_extension`**`(`*nb_app*`)`

This function loads bookstore as a Jupyter Server extension.

1.5.4 Storage

The `s3_paths` module

S3 path utilities

`bookstore.s3_paths.`**`s3_display_path`**`(`*bucket*, *prefix*, *path=''*`)`
Create a display name for use in logs

Parameters

- **`bucket`** (*`str`*) – S3 bucket name
- **`prefix`** (*`str`*) – prefix for workspace or publish
- **`path`** (*`str`*) – The storage location

`bookstore.s3_paths.`**`s3_key`**`(`*prefix*, *path=''*`)`
Compute the s3 key

Parameters

- **`prefix`** (*`str`*) – prefix for workspace or publish
- **`path`** (*`str`*) – The storage location

`bookstore.s3_paths.`**`s3_path`**`(`*bucket*, *prefix*, *path=''*`)`
Compute the s3 path.

Parameters

- **`bucket`** (*`str`*) – S3 bucket name
- **`prefix`** (*`str`*) – prefix for workspace or publish
- **`path`** (*`str`*) – The storage location

1.5.5 Cloning

The `clone` module

`bookstore.clone.`**`build_notebook_model`**`(`*content*, *path*`)`
Helper that builds a Contents API compatible model for notebooks.

Parameters

- **`content`** (*`str`*) – The content of the model.
- **`path`** (*`str`*) – The path to be targeted.

Returns Jupyter Contents API compatible model for notebooks

Return type dict

`bookstore.clone.`**`build_file_model`**`(`*content*, *path*`)`
Helper that builds a Contents API compatible model for files.

Parameters

- **`content`** (*`str`*) – The content of the model
- **`path`** (*`str`*) – The path to be targeted.

Returns Jupyter Contents API compatible model for files

Return type dict

`bookstore.clone.`**`validate_relpath`**`(`*relpath*, *settings*, *log*`)`
Validates that a relative path appropriately resolves given bookstore settings.

Parameters

- **`relpath`** (*`string`*) – Relative path to a notebook to be cloned.
- **`settings`** (`BookstoreSettings`) – Bookstore configuration.
- **`log`** (*`logging.Logger`*) – Log (usually from the NotebookApp) for logging endpoint changes.

Returns Absolute path to file to be cloned.

Return type Path

`BookstoreCloneHandler`

`class` `bookstore.clone.`**`BookstoreCloneHandler`**`(`*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs*`)`
Prepares and provides clone options page, populating UI with clone option parameters.

Provides handling for `GET` requests when cloning a notebook from storage (S3). Launches a user interface with cloning options.

`initialize``(`*self*`)`
Helper to access bookstore settings.

`get``(`*self*`)`
Checks for valid storage settings and render a UI for clone options.

`construct_template_params``(`*self*, *s3_bucket*, *s3_object_key*, *s3_version_id=None*`)`
Helper to populate Jinja template for cloning option page.

`get_template``(`*self*, *name*`)`
Loads a Jinja template and its related settings.

See also:

Jupyter Notebook reference on Custom Handlers

Methods

`BookstoreCloneHandler.`**`initialize`**`()`
Helper to retrieve bookstore setting for the session.

`BookstoreCloneHandler.`**`get`**`()`
GET /bookstore/clone?s3_bucket=<your_s3_bucket>&s3_key=<your_s3_key>

Renders an options page that will allow you to clone a notebook from a specific bucket via the Bookstore cloning API.

s3_bucket is the bucket you wish to clone from. s3_key is the object key that you wish to clone.

`BookstoreCloneHandler.`**`construct_template_params`**(*s3_bucket*, *s3_object_key*, *s3_version_id=None*)

Helper that takes valid S3 parameters and populates UI template

Returns Template parameters in a dictionary

Return type dict

`BookstoreCloneHandler.`**`get_template`**(*name*)

Loads a Jinja template by name.

BookstoreCloneAPIHandler

`class` `bookstore.clone.`**`BookstoreCloneAPIHandler`**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs*)

Handle notebook clone from storage.

Provides API handling for `POST` and clones a notebook from storage (S3).

`initialize`(*self*)

Helper to access bookstore settings.

`post`(*self*)

Clone a notebook from the location specified by the payload.

`build_content_model`(*self*, *obj*, *path*)

Helper that takes a response from S3 and creates a ContentsAPI compatible model.

`build_post_response_model`(*self*, *model*, *obj*, *s3_bucket*, *s3_object_key*)

Helper that takes a Jupyter Contents API compliant model and adds cloning specific information.

See also:

Jupyter Notebook reference on Custom Handlers

Methods

`BookstoreCloneAPIHandler.`**`initialize`**()

Helper to retrieve bookstore setting for the session.

`BookstoreCloneAPIHandler.`**`post`**()

POST /api/bookstore/clone

Clone a notebook to the path specified in the payload.

The payload type for the request should be:

```
{
"s3_bucket": string,
"s3_key": string,
"target_path"?: string
"s3_version_id"?: string
}
```

The response payload should match the standard Jupyter contents API POST response.

`BookstoreCloneAPIHandler.`**`build_content_model`**`(`*content*, *target_path*`)`

Helper that takes a response from S3 and creates a ContentsAPI compatible model.

If the file at target_path already exists, this increments the file name.

Parameters

- **`content`** (*`str`*) – string encoded file content
- **`target_path`** (*`str`*) – The the path we wish to clone to, may be incremented if already present.

Returns Jupyter Contents API compatible model

Return type dict

`BookstoreCloneAPIHandler.`**`build_post_response_model`**`(`*model*, *obj*, *s3_bucket*, *s3_object_key*`)`

Helper that takes a Jupyter Contents API compliant model and adds cloning specific information.

Parameters

- **`model`** (*`dict`*) – Jupyter Contents API model
- **`obj`** (*`dict`*) – Log (usually from the NotebookApp) for logging endpoint changes.
- **`s3_bucket`** (*`str`*) – The S3 bucket we are cloning from
- **`s3_object_key`** (*`str`*) – The S3 key we are cloning

Returns Model with additional info about the S3 cloning

Return type dict

`BookstoreFSCloneHandler`

Methods

`BookstoreFSCloneHandler.`**`initialize`**`()`

Helper to retrieve bookstore setting for the session.

`BookstoreFSCloneHandler.`**`get`**`()`

GET /bookstore/fs-clone?relpath=<your_relpath>

Renders an options page that will allow you to clone a notebook from a via the Bookstore file-system cloning API.

relpath is the relative path that you wish to clone from

`BookstoreFSCloneHandler.`**`construct_template_params`**`(`*relpath*, *fs_clonepath*`)`

Helper that takes a valid relpath and populates UI template

Returns Template parameters in a dictionary

Return type dict

`BookstoreFSCloneHandler.`**`get_template`**`(`*name*`)`

Loads a Jinja template by name.

BookstoreFSCloneAPIHandler

class bookstore.clone.**BookstoreFSCloneAPIHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs*)

Handle notebook clone from an accessible file system (local or cloud).

Provides API handling for `POST` and clones a notebook from the specified file system (local or cloud).

initialize(*self*)
Helper to access bookstore settings.

post(*self*)
Clone a notebook from the filesystem location specified by the payload.

build_content_model(*self*, *content*, *path*)
Helper for creating a Jupyter ContentsAPI compatible model.

See also:

Jupyter Notebook reference on Custom Handlers

Methods

BookstoreFSCloneAPIHandler.**initialize**()
Helper to retrieve bookstore setting for the session.

BookstoreFSCloneAPIHandler.**post**()
POST /api/bookstore/fs-clone

Clone a notebook to the path specified in the payload.

The payload type for the request should be:

```
{
"relpath": string,
"target_path": string #optional
}
```

The response payload should match the standard Jupyter contents API POST response.

BookstoreFSCloneAPIHandler.**build_content_model**(*content*, *target_path*)
Helper that takes a content and creates a ContentsAPI compatible model.

If the file at target_path already exists, this increments the file name.

Parameters

- **content** (*dict or string*) – dict or string encoded file content
- **target_path** (*str*) – The the path we wish to clone to, may be incremented if already present.

Returns

Jupyter Contents API compatible model

Return type dict

1.5.6 Publishing

The `publish` module

`BookstorePublishAPIHandler`

class bookstore.publish.**BookstorePublishAPIHandler**(*application: tornado.web.Application*, *request: tornado.httputil.HTTPServerRequest*, ***kwargs*)

Publish a notebook to the publish path

Methods

BookstorePublishAPIHandler.**initialize**()

Initialize a helper to get bookstore settings and session information quickly

BookstorePublishAPIHandler.**put**(*path*)

Publish a notebook on a given path.

PUT /api/bookstore/publish

The payload directly matches the contents API for PUT.

Parameters **path** (*str*) – Path describing where contents should be published to, postfixed to the published_prefix .

BookstorePublishAPIHandler.**validate_model**(*model*)

Checks that the model given to the API handler meets bookstore's expected structure for a notebook.

Pattern for surfacing nbformat validation errors originally written in https://github.com/jupyter/notebook/blob/a44a367c219b60a19bee003877d32c3ff1ce2412/notebook/services/contents/manager.py#L353-L355

Parameters **model** (*dict*) – Request model for publishing describing the type and content of the object.

Raises `tornado.web.HTTPError` – Your model does not validate correctly

BookstorePublishAPIHandler.**prepare_response**(*obj*, *full_s3_path*)

Prepares repsonse to publish PUT request.

Parameters

- **obj** (*dict*) – Validation dictionary for determining which endpoints to enable.
- **path** – path to place after the published prefix in the designated bucket

Returns Model for responding to put request.

Return type dict

1.5.7 Notebook Client

The `bookstore.client.nb_client` module

`NotebookClient`

class `bookstore.client.nb_client.`**`NotebookClient`**(*nb_config*)

EXPERIMENTAL SUPPORT: Client used to interact with a notebook server from within a notebook.

Parameters **`nb_config`** (*`dict`*) – Dictionary of info compatible with creating a LiveNotebookRecord.

`nb_config`

Dictionary of info compatible with creating a LiveNotebookRecord.

Type dict

`nb_record`

LiveNotebookRecord of info for this notebook

Type *LiveNotebookRecord*

`url`

url from nb_record minus final /

Type str

`token`

token used for authenticating requests serverside

Type str

`xsrf_token`

xsrf_token used in cookie for authenticating requests

Type str

`req_session`

Session to be reused across methods

Type requests.Session

`contents_endpoint`

Current server's contents API endpoint.

`get_contents`(*path*)

Requests info about current contents from notebook server.

`get_kernels`()

Requests info about current kernels from notebook server.

`get_sessions`()

Requests info about current sessions from notebook server.

`headers`

Default headers to be shared across requests.

`kernels`

Current notebook kernels. Reissues request on each call.

`kernels_endpoint`

Current server's kernels API endpoint.

`sessions`

Current notebook sessions. Reissues request on each call.

`sessions_endpoint`

Current server's kernels API endpoint.

setup_auth()

Sets up token access for authorizing requests to notebook server.

This sets the notebook token as self.token and the xsrf_token as self.xsrf_token.

setup_request_sessions()

Sets up a requests.Session object for sharing headers across API requests.

NotebookClientCollection

class bookstore.client.nb_client.**NotebookClientCollection**

EXPERIMENTAL SUPPORT: Representation of a collection of notebook clients

classmethod current_server()

class method for current notebook server

CurrentNotebookClient

class bookstore.client.nb_client.**CurrentNotebookClient**

EXPERIMENTAL SUPPORT: Represents the currently active notebook client.

connection_file

Connection file for connecting to current notebook's kernel.

kernel_id

Kernel id for identifying which notebook is currently being used by this session.

LiveNotebookRecord

class bookstore.client.nb_client.**LiveNotebookRecord**

Representation of live notebook server.

This is a record of an object returned by *notebook.notebookapp.list_running_servers()*.

Example

```
[{'base_url': '/',
'hostname': 'localhost',
'notebook_dir': '/Users/mpacer/jupyter/eg_notebooks',
'password': False,
'pid': 96033,
'port': 8888,
'secure': False,
'token': '',
'url': 'http://localhost:8888/'}]
```

base_url

Alias for field number 0

hostname

Alias for field number 1

notebook_dir

Alias for field number 2

password
Alias for field number 3

pid
Alias for field number 4

port
Alias for field number 5

secure
Alias for field number 6

token
Alias for field number 7

url
Alias for field number 8

KernelInfo

class bookstore.client.nb_client.**KernelInfo**(**args*, *id*, *name*, *last_activity*, *execution_state*, *connections*)
Representation of kernel info returned by the notebook's /api/kernel endpoint.

id

Type str

name

Type str

last_activity

Type str

execution_state

Type str

connections

Type int

Example

```
{id: 'f92b7c8b-0858-4d10-903c-b0631540fb36',
name: 'dev',
last_activity: '2019-03-14T23:38:08.137987Z',
execution_state: 'idle',
connections: 0}
```

NotebookSession

class bookstore.client.nb_client.**NotebookSession**(**args*, *path*, *name*, *type*, *kernel*, *notebook={}*, ***kwargs*)
Representation of session info returned by the notebook's /api/sessions/ endpoint.

id

Type str

path

Type str

name

Type str

type

Type str

kernel

Type *KernelInfo*

notebook

Type dict

model

Record of the raw response (without converting the KernelInfo).

Type dict

Example

```
{id: '68d9c58f-c57d-4133-8b41-5ec2731b268d',
 path: 'Untitled38.ipynb',
 name: '',
 type: 'notebook',
 kernel: KernelInfo(id='f92b7c8b-0858-4d10-903c-b0631540fb36',
                    name='dev',
                    last_activity='2019-03-14T23:38:08.137987Z',
                    execution_state='idle',
                    connections=0),
notebook: {'path': 'Untitled38.ipynb', 'name': ''}}
```

Helper Function

bookstore.client.nb_client.**extract_kernel_id**(*connection_file*)

Get the kernel id string from a file

1.5.8 Bookstore Client

The `bookstore.client.store_client` module

`BookstoreClient`

class bookstore.client.store_client.**BookstoreClient**(*s3_bucket=None*)

Bases: *bookstore.client.nb_client.CurrentNotebookClient*

EXPERIMENTAL SUPPORT: A client that allows access to a Bookstore from within a notebook.

Parameters **s3_bucket** (*str*) – (optional) Provide a default bucket for this bookstore client to clone from.

default_bucket
The default bucket to be used for cloning.

Type str

clone (*s3_bucket=”, s3_key=”, target_path=”*)
Clone files via bookstore.

Parameters

- **s3_bucket** (*str*) – (optional) S3 bucket you wish to clone from; defaults to client’s bucket.
- **s3_object_key** (*str*) – The object key describing the object you wish to clone from S3.
- **target_path** (*str*) – (optional) The location you wish to clone the object to; defaults to s3_object_key.

clone_endpoint
Helper to refer to construct the clone endpoint for this notebook server.

publish (*path=None*)
Publish notebook to bookstore

Parameters

- **path** (*str*) – (optional) Path that you wish to publish; defaults to current notebook.
- **s3_object_key** (*str*) – The the path we wish to clone to.

publish_endpoint
Helper to refer to construct the publish endpoint for this notebook server.

1.6 Project

1.6.1 Contributing

Oh, hello there! You’re probably reading this because you are interested in contributing to nteract. That’s great to hear! This document will help you through your journey of open source. Love it, cherish it, take it out to dinner, but most importantly: read it thoroughly!

What do I need to know to help?

Read the README.md file. This will help you set up the project. If you have questions, please ask on the nteract Slack channel. We’re a welcoming project and are happy to answer your questions.

How do I make a contribution?

Never made an open source contribution before? Wondering how contributions work in the nteract world? Here’s a quick rundown!

1. Find an issue that you are interested in addressing or a feature that you would like to address.
2. Fork the repository associated with the issue to your local GitHub organization.
3. Clone the repository to your local machine using:

```
git clone https://github.com/github-username/repository-name.git
```

4. Create a new branch for your fix using:

```
git checkout -b branch-name-here
```

5. Make the appropriate changes for the issue you are trying to address or the feature that you want to add.
6. You can run python unit tests using `pytest`. Running integration tests locally requires a more complicated setup. This setup is described in running_ci_locally.md

#. Add and commit the changed files using `git add` and `git commit`. #.

Push the changes to the remote repository using:

```
git push origin branch-name-here
```

1. Submit a pull request to the upstream repository.
2. Title the pull request per the requirements outlined in the section below.
3. Set the description of the pull request with a brief description of what you did and any questions you might have about what you did.
4. Wait for the pull request to be reviewed by a maintainer.
5. Make changes to the pull request if the reviewing maintainer recommends them.
6. Celebrate your success after your pull request is merged! :tada:

How should I write my commit messages and PR titles?

Good commit messages serve at least three important purposes:

- To speed up the reviewing process.
- To help us write a good release note.
- To help the future maintainers of nteract/nteract (it could be you!), say five years into the future, to find out why a particular change was made to the code or why a specific feature was added.

Structure your commit message like this:

```
> Short (50 chars or less) summary of changes
>
> More detailed explanatory text, if necessary.  Wrap it to about 72
> characters or so.  In some contexts, the first line is treated as the
> subject of an email and the rest of the text as the body.  The blank
> line separating the summary from the body is critical (unless you omit
> the body entirely); tools like rebase can get confused if you run the
> two together.
>
> Further paragraphs come after blank lines.
>
>   - Bullet points are okay, too
>
>   - Typically a hyphen or asterisk is used for the bullet, preceded by a
>     single space, with blank lines in between, but conventions vary here
>
```

Source: https://git-scm.com/book/ch5-2.html

DO

- Write the summary line and description of what you have done in the imperative mode, that is as if you were commanding. Start the line with “Fix”, “Add”, “Change” instead of “Fixed”, “Added”, “Changed”.
- Always leave the second line blank.
- Line break the commit message (to make the commit message readable without having to scroll horizontally in gitk).

DON’T

- Don’t end the summary line with a period - it’s a title and titles don’t end with a period.

Tips

- If it seems difficult to summarize what your commit does, it may be because it includes several logical changes or bug fixes, and are better split up into several commits using `git add -p`.

References

The following blog post has a nice discussion of commit messages:

- “On commit messages” http://who-t.blogspot.com/2009/12/on-commit-messages.html

How fast will my PR be merged?

Your pull request will be merged as soon as there are maintainers to review it and after tests have passed. You might have to make some changes before your PR is merged but as long as you adhere to the steps above and try your best, you should have no problem getting your PR merged.

That’s it! You’re good to go!

1.6.2 Contributor Code of Conduct

As contributors and maintainers of this project, and in the interest of fostering an open and welcoming community, we pledge to respect all people who contribute through reporting issues, posting feature requests, updating documentation, submitting pull requests or patches, and other activities.

We are committed to making participation in this project 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.

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

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.

By adopting this Code of Conduct, project maintainers commit themselves to fairly and consistently applying these principles to every aspect of managing this project. Project maintainers who do not follow or enforce the Code of Conduct may be permanently removed from the project team.

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.

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project maintainer at [rgbkrk@gmail.com]. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. Maintainers are obligated to maintain confidentiality with regard to the reporter of an incident.

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available from http://contributor-covenant.org/version/1/4/

1.6.3 Local Continuous Integration

It helps when developing to be able to run integration tests locally. Since bookstore relies on accessing S3, this requires that we create a local server that can model how S3 works.

We will be using minio to mock S3 behavior.

Setup Local CI environment

To run the ci tests locally, you will need to have a few things set up:

- a functioning `docker` service
- define `/mnt/data/` and `/mnt/config/` and give full permissions (e.g., `chmod 777 /mnt/data`). = add `/mnt/data` and `/mnt/config` to be accessible from `docker`. You can do so by modifying Docker's preferences by going to `Docker → Preferences → File Sharing` and adding `/mnt/data` and `/mnt/config` to the list there.
- an up-to-date version of `node`.

Run Local tests

1. Open two terminals with the current working directory as the root `bookstore` directory.
2. In one terminal run `yarn test:server`. This will start up minio.
3. In the other terminal run `yarn test`. This will run the integration tests.

Interactive python tests

The CI scripts are designed to be self-contained and run in an automated setup. This makes it makes it harder to iterate rapidly when you don't want to test the *entire* system but when you do need to integrate with a Jupyter server.

In addition the CI scripts, we have included `./ci/clone_request.py` for testing the clone endpoint. This is particularly useful for the `/api/bookstore/cloned` endpoint because while it is an API to be used by other applications, it also acts as a user facing endpoint since it provides a landing page for confirming whether or not a clone is to be approved.

It's often difficult to judge whether what is being served makes sense from a UI perspective without being able to investigate it directly. At the same time we'll need to access it as an API to ensure that the responses are well-behaved from an API standpoint. By using python to query a live server and a browser to visit the landing page, we can rapidly iterate between the API and UI contexts from the same live server's endpoint.

We provide examples of `jupyter notebook` commands needed in that file as well for both accessing the `nteract-notebooks` S3 bucket as well as the Minio provided `bookstore` bucket (as used by the CI scripts).

1.6.4 Running Python Tests

The project uses pytest to run Python tests and tox as a tool for running tests in different environments.

Setup Local development system

Using Python 3.6+, install the dev requirements:

```
pip install -r requirements-dev.txt
```

Run Python tests

Important: We recommend using tox for running tests locally. Please deactivate any conda environments before running tests using tox. Failure to do so may corrupt your virtual environments.

To run tests for a particular Python version (3.6 or 3.7):

```
tox -e py36  # or py37
```

This will run the tests and display coverage information.

Run linters

```
tox -e flake8
tox -e black
```

Run type checking

```
tox -e mypy
```

Run All Tests and Checks

```
tox
```

1.6.5 Releasing

Pre-release

- [ ] First check that the CHANGELOG is up to date for the next release version.
- [ ] Update docs

Installing twine package

Install and upgrade, if needed,`twine` with `python3 -m pip install -U twine`. The long description of the package will not render on PyPI unless an up-to-date version is used.

Create the release

- [ ] Update version numbers in
  - [ ] `bookstore/_version.py` (version_info)
  - [ ] `docs/source/conf.py` (version and release)
  - [ ] `docs/source/bookstore_api.yaml` (info.version)
- [ ] Commit the updated version
- [ ] Clean the repo of all non-tracked files: `git clean -xdfi`
- [ ] Commit and tag the release

```
git commit -am"release $VERSION"
git tag $VERSION
```

- [ ] Push the tags and remove any existing `dist` directory files

```
git push && git push --tags
rm -rf dist/*
```

- [ ] Build `sdist` and `wheel`

```
python setup.py sdist
python setup.py bdist_wheel
```

Test and upload release to PyPI

- [ ] Test the wheel and sdist locally
- [ ] Upload to PyPI using `twine` over SSL

```
twine upload dist/*
```

- [ ] If all went well:
  - Change `bookstore/_version.py` back to `.dev`
  - Push directly to `master` and push `--tags` too.

1.7 Change Log

1.7.1 Unreleased

1.7.2 2.5.1

This enables adds a new feature to bookstore cloning from s3, cloning specific versions of notebooks from versioned s3 buckets.

Specifically, it introduces the `s3_version_id` query parameter to the `/bookstore/clone/` GET handler.

So if you wanted to clone a specific version `myVersion` of `/workspace/my_notebook.ipynb` from the `my_bucket` S3 bucket, you would change the route from something like

http://localhost:8888/bookstore/clone?s3_bucket=my_bucket&s3_key=workspace/my_notebook.ipynb

to

http://localhost:8888/bookstore/clone?s3_bucket=my_bucket&s3_key=workspace/my_notebook.ipynb&s3_version_id=myVersion

1.7.3 2.5.0

This switches the bookstore serverextension and landing page from using absolute urls to relative paths.

1.7.4 2.4.1 2019-08-6

This improves the landing page experience with a cleaner and clearer landing page design.

1.7.5 2.4.0 2019-08-5

Thank you to the following contributors:

- Carol Willing
- M Pacer
- Peter Volpe

The full list of changes they made can be seen on GitHub

Significant changes

Cloning

As of 2.4.0 cloning from a local or network attached file system is now possible, but disabled by default.

To enable this filesystem (`fs`) cloning, set `BookstoreSettings.fs_cloning_basedir` to the root directory from which you want people to be able to clone.

Adding fs cloning offers users more flexibility to clone notebooks from attached filesystems, like EFS. For more information about the motivation and design of this endpoint, please see this issue.

1.7.6 2.3.1 2019-07-16

Fixing problems

This fixes an issue that arose where in certain cases cloning would hang indefinitely when trying to read content #145.

1.7.7 2.3.0 2019-07-02

Thank you to the following contributors:

- Carol Willing
- Kyle Kelley
- M Pacer
- Matthew Seal
- Safia Abdalla
- Shelby Sturgis

The full list of changes they made can be seen on GitHub

Significant changes

New Publishing endpoint

Previously our publishing endpoint was `/api/bookstore/published`, it is now `/api/bookstore/publish`.

Cloning

As of 2.3.0 cloning from S3 is now enabled by default.

Cloning allows access to multiple S3 buckets. To use them, you will need to set up your configuration for any such bucket.

Massive Testing improvements

We have built out a framework for unit-testing Tornado handlers. In addition, we have added a collection of unit tests that bring us to a coverage level in non-experimental code of well over 80%.

`/api/bookstore/`: Features and Versions

You can identify which features have been enabled and which version of bookstore is available by using the `/api/bookstore` endpoint.

REST API Documentation

All APIs are now documented at our REST API docs using the OpenAPI spec.

Experimental

Clients (subject to change in future releases)

To enable access to bookstore publishing and cloning from within a notebook, we have created a Notebook and Bookstore clients. *This is still experimental* functionality at the moment and needs additional testing, so we discourage its use in production. The design relies on an assumption that a single kernel is attached to a single notebook, and will break if you use multiple notebooks attached to the same kernel.

However, for those who wish to experiment, it offers some fun ways of exploring bookstore.

Example: if you run a notebook from within the top-level *‘‘bookstore/ci‘* <https://github.com/nteract/bookstore/tree/master/ci>‘_ directory while running the integration test server with `yarn test:server` (see more about local integration testing), you should be able to publish from inside a notebook using the following code snippet:‘‘‘

```
from bookstore.client import BookstoreClient
book_store = BookstoreClient()
book_store.publish()
```

And if you have published your notebook to the local ci (e.g., publishing `my_notebook.ipynb` to the minio `bookstore` bucket with the `ci-published` published prefix), you can clone it from S3 using:

```
from bookstore.client import BookstoreClient
book_store = BookstoreClient()
book_store.clone("bookstore", "ci-published/my_notebook.ipynb")
```

1.7.8 Releases prior to 2.3.0

2.2.1 (2019-02-03)

2.2.0 (2019-01-29)

2.1.0 (2018-11-20)

2.0.0 (2018-11-13)

0.1 (2018=10-16)

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

HTTP Routing Table

Index