GCSFs Documentation

Release 0.6.2

Continuum Analytics

May 06, 2020

Contents

A pythonic file-system interface to Google Cloud Storage.

This software is beta, use at your own risk.

Please file issues and requests on github and we welcome pull requests.

This package depends on fsspec , and inherits many useful behaviours from there, including integration with Dask, and the facility for key-value dict-like objects of the type used by zarr.

CHAPTER 1

Installation

The GCSFS library can be installed using `conda` or `pip`:

```
conda install -c conda-forge gcsfs
or
pip install gcsfs
```

or by cloning the repository:

```
git clone https://github.com/dask/gcsfs/
cd gcsfs/
pip install .
```

CHAPTER 2

Examples

Locate and read a file:

```
>>> import gcsfs
>>> fs = gcsfs.GCSFileSystem(project='my-google-project')
>>> fs.ls('my-bucket')
['my-file.txt']
>>> with fs.open('my-bucket/my-file.txt', 'rb') as f:
...     print(f.read())
b'Hello, world'
```

(see also `walk` and `glob`)

Read with delimited blocks:

```
>>> fs.read_block(path, offset=1000, length=10, delimiter=b'\n')
b'A whole line of text\n'
```

Write with blocked caching:

```
>>> with fs.open('mybucket/new-file', 'wb') as f:
...     f.write(2*2**20 * b'a')
...     f.write(2*2**20 * b'a') # data is flushed and file closed
>>> fs.du('mybucket/new-file')
{'mybucket/new-file': 4194304}
```

Because GCSFS faithfully copies the Python file interface it can be used smoothly with other projects that consume the file interface like `gzip` or `pandas`.

```
>>> with fs.open('mybucket/my-file.csv.gz', 'rb') as f:
...     g = gzip.GzipFile(fileobj=f)  # Decompress data with gzip
...     df = pd.read_csv(g)           # Read CSV file with Pandas
```

CHAPTER 3

Credentials

Several modes of authentication are supported:

- if `token=None` (default), GCSFS will attempt to use your default gcloud credentials or, attempt to get credentials from the google metadata service, or fall back to anonymous access. This will work for most users without further action. Note that the default project may also be found, but it is often best to supply this anyway (only affects bucket- level operations).
- if `token='cloud'`, we assume we are running within google (compute or container engine) and fetch the credentials automatically from the metadata service.
- you may supply a token generated by the gcloud utility; this is either a python dictionary, or the name of a file containing the JSON returned by logging in with the gcloud CLI tool (e.g., `~/.config/gcloud/application_default_credentials.json` or `~/.config/gcloud/legacy_credentials/<YOUR GOOGLE USERNAME>/adc.json`) or any value google `Credentials` object.
- you can also generate tokens via Oauth2 in the browser using `token='browser'`, which gcsfs then caches in a special file, ~/.gcs_tokens, and can subsequently be accessed with `token='cache'`.
- anonymous only access can be selected using `token='anon'`, e.g. to access public resources such as 'anaconda-public-data'.

The acquired session tokens are *not* preserved when serializing the instances, so it is safe to pass them to worker processes on other machines if using in a distributed computation context. If credentials are given by a file path, however, then this file must exist on every machine.

CHAPTER 4

Contents

4.1 API

GCSFileSystem([project, access, token, ...])	Connect to Google Cloud Storage.
GCSFileSystem.cat(self, path)	Simple one-shot get of file data
GCSFileSystem.du(self, path[, total, maxdepth])	Space used by files within a path
GCSFileSystem.exists(self, path)	Is there a file at the given path
GCSFileSystem.get(self, rpath, lpath[, ...])	Copy file to local.
GCSFileSystem.glob(self, path, **kwargs)	Find files by glob-matching.
GCSFileSystem.info(self, path, **kwargs)	File information about this path.
GCSFileSystem.ls(self, path[, detail])	List objects under the given '/{bucket}/{prefix} path.
GCSFileSystem.mkdir(self, bucket[, acl, ...])	New bucket
GCSFileSystem.mv(self, path1, path2, **kwargs)	Move file from one location to another
GCSFileSystem.open(self, path[, mode, ...])	Return a file-like object from the filesystem
GCSFileSystem.put(self, lpath, rpath[, ...])	Upload file from local
GCSFileSystem.read_block(self, fn, offset, ...)	Read a block of bytes from
GCSFileSystem.rm(self, path[, recursive, ...])	Delete keys.
GCSFileSystem.tail(self, path[, size])	Get the last size bytes from file
GCSFileSystem.touch(self, path[, truncate])	Create empty file, or update timestamp
GCSFileSystem.get_mapper(self, root[, ...])	Create key/value store based on this file-system

GCSFile(gcsfs, path[, mode, block_size, ...])

Attributes

GCSFile.close(self)	Close file
GCSFile.flush(self[, force])	Write buffered data to backend store.
GCSFile.info(self)	File information about this path
GCSFile.read(self[, length])	Return data from cache, or fetch pieces as necessary
GCSFile.seek(self, loc[, whence])	Set current file location

Continued on next page

Table 2 – continued from previous page

GCSFile.tell(self)	Current file location
GCSFile.write(self, data)	Write data to buffer.

class gcsfs.core.**GCSFileSystem**(*project=''*, *access='full_control'*, *token=None*, *block_size=None*, *consistency='none'*, *cache_timeout=None*, *secure_serialize=True*, *check_connection=False*, *requests_timeout=None*, *requester_pays=False*, ***kwargs*)

Connect to Google Cloud Storage.

The following modes of authentication are supported:

- `token=None`, GCSFS will attempt to guess your credentials in the following order: gcloud CLI default, gcsfs cached token, google compute metadata service, anonymous.
- `token='google_default'`, your default gcloud credentials will be used, which are typically established by doing `gcloud login` in a terminal.
- `token=='cache'`, credentials from previously successful gcsfs authentication will be used (use this after "browser" auth succeeded)
- `token='anon'`, no authentication is preformed, and you can only access data which is accessible to allUsers (in this case, the project and access level parameters are meaningless)
- `token='browser'`, you get an access code with which you can authenticate via a specially provided URL
- if `token='cloud'`, we assume we are running within google compute or google container engine, and query the internal metadata directly for a token.
- you may supply a token generated by the [gcloud](https://cloud.google.com/sdk/docs/) utility; this is either a python dictionary, the name of a file containing the JSON returned by logging in with the gcloud CLI tool, or a Credentials object. gcloud typically stores its tokens in locations such as `~/.config/gcloud/application_default_credentials.json`, `` ~/.config/gcloud/credentials``, or `~\AppData\Roaming\gcloud\credentials`, etc.

Specific methods, (eg. *ls*, *info*, ...) may return object details from GCS. These detailed listings include the [object resource](https://cloud.google.com/storage/docs/json_api/v1/objects#resource)

GCS *does not* include "directory" objects but instead generates directories by splitting [object names](https://cloud.google.com/storage/docs/key-terms). This means that, for example, a directory does not need to exist for an object to be created within it. Creating an object implicitly creates it's parent directories, and removing all objects from a directory implicitly deletes the empty directory.

GCSFileSystem generates listing entries for these implied directories in listing apis with the object properies:

- **"name"** [string] The "{bucket}/{name}" path of the dir, used in calls to GCSFileSystem or GCSFile.
- **"bucket"** [string] The name of the bucket containing this object.
- "kind" : 'storage#object'
- "size" : 0
- "storageClass" : 'DIRECTORY'
- type: 'directory' (fsspec compat)

GCSFileSystem maintains a per-implied-directory cache of object listings and fulfills all object information and listing requests from cache. This implied, for example, that objects created via other processes *will not* be visible to the GCSFileSystem until the cache refreshed. Calls to GCSFileSystem.open and calls to GCSFile are not effected by this cache.

In the default case the cache is never expired. This may be controlled via the *cache_timeout* GCSFileSystem parameter or via explicit calls to *GCSFileSystem.invalidate_cache*.

Parameters

project [string] project_id to work under. Note that this is not the same as, but often very similar to, the project name. This is required in order to list all the buckets you have access to within a project and to create/delete buckets, or update their access policies. If `token='google_default'`, the value is overriden by the default, if `token='anon'`, the value is ignored.

access [one of {'read_only', 'read_write', 'full_control'}] Full control implies read/write as well as modifying metadata, e.g., access control.

token: None, dict or string (see description of authentication methods, above)

consistency: 'none', 'size', 'md5' Check method when writing files. Can be overridden in open().

cache_timeout: float, seconds Cache expiration time in seconds for object metadata cache. Set cache_timeout <= 0 for no caching, None for no cache expiration.

secure_serialize: bool If True, instances re-establish auth upon deserialization; if False, token is passed directly, which may be a security risk if passed across an insecure network.

check_connection: bool When token=None, gcsfs will attempt various methods of establishing credentials, falling back to anon. It is possible for a method to find credentials in the system that turn out not to be valid. Setting this parameter to True will ensure that an actual operation is attempted before deciding that credentials are valid.

requester_pays [bool, or str default False] Whether to use requester-pays requests. This will include your project ID *project* in requests as the *userPorject*, and you'll be billed for accessing data from requester-pays buckets. Optionally, pass a project-id here as a string to use that as the *userProject*.

Attributes

buckets Return list of available project buckets.

transaction A context within which files are committed together upon exit

Methods

cat(self, path)	Simple one-shot get of file data
checksum(self, path)	Unique value for current version of file
clear_instance_cache()	Clear the cache of filesystem instances.
connect(self[, method])	Establish session token.
copy(self, path1, path2[, acl])	Duplicate remote file
cp(self, path1, path2, **kwargs)	Alias of FilesystemSpec.copy.
created(self, path)	Return the created timestamp of a file as a datetime.datetime
current()	Return the most recently created FileSystem
delete(self, path[, recursive, maxdepth])	Alias of FilesystemSpec.rm.
disk_usage(self, path[, total, maxdepth])	Alias of FilesystemSpec.du.
download(self, rpath, lpath[, recursive])	Alias of FilesystemSpec.get.
du(self, path[, total, maxdepth])	Space used by files within a path
end_transaction(self)	Finish write transaction, non-context version

Continued on next page

Table 3 – continued from previous page

exists(self, path)	Is there a file at the given path
find(self, path[, maxdepth, withdirs])	List all files below path.
from_json(blob)	Recreate a filesystem instance from JSON representation
get(self, rpath, lpath[, recursive])	Copy file to local.
get_mapper(self, root[, check, create])	Create key/value store based on this file-system
getxattr(self, path, attr)	Get user-defined metadata attribute
glob(self, path, **kwargs)	Find files by glob-matching.
head(self, path[, size])	Get the first size bytes from file
info(self, path, **kwargs)	File information about this path.
invalidate_cache(self[, path])	Invalidate listing cache for given path, it is reloaded on next use.
isdir(self, path)	Is this entry directory-like?
isfile(self, path)	Is this entry file-like?
listdir(self, path[, detail])	Alias of FilesystemSpec.ls.
load_tokens()	Get "browser" tokens from disc
ls(self, path[, detail])	List objects under the given '/{bucket}/{prefix} path.
makedir(self, path[, create_parents])	Alias of FilesystemSpec.mkdir.
makedirs(self, path[, exist_ok])	Recursively make directories
merge(self, path, paths[, acl])	Concatenate objects within a single bucket
mkdir(self, bucket[, acl, default_acl])	New bucket
mkdirs(self, path[, exist_ok])	Alias of FilesystemSpec.makedirs.
modified(self, path)	Return the modified timestamp of a file as a datetime.datetime
move(self, path1, path2, **kwargs)	Alias of FilesystemSpec.mv.
mv(self, path1, path2, **kwargs)	Move file from one location to another
open(self, path[, mode, block_size, ...])	Return a file-like object from the filesystem
put(self, lpath, rpath[, recursive])	Upload file from local
read_block(self, fn, offset, length[, delimiter])	Read a block of bytes from
rename(self, path1, path2, **kwargs)	Alias of FilesystemSpec.mv.
rm(self, path[, recursive, maxdepth])	Delete keys.
rmdir(self, bucket)	Delete an empty bucket
setxattrs(self, path[, content_type, ...])	Set/delete/add writable metadata attributes
size(self, path)	Size in bytes of file
split_path(path)	Normalise GCS path string into bucket and key.
start_transaction(self)	Begin write transaction for deferring files, non-context version
stat(self, path, **kwargs)	Alias of FilesystemSpec.info.
tail(self, path[, size])	Get the last size bytes from file
to_json(self)	JSON representation of this filesystem instance
touch(self, path[, truncate])	Create empty file, or update timestamp
ukey(self, path)	Hash of file properties, to tell if it has changed
upload(self, lpath, rpath[, recursive])	Alias of FilesystemSpec.put.
url(path)	Get HTTP URL of the given path
walk(self, path[, maxdepth])	Return all files belows path

buckets
Return list of available project buckets.

cat (*self*, *path*)
Simple one-shot get of file data

connect (*self*, *method=None*)
Establish session token. A new token will be requested if the current one is within 100s of expiry.

Parameters

method: str (google_default|cache|cloud|token|anon|browser) or None Type of authorisation to implement - calls *_connect_** methods. If None, will try sequence of methods.

copy (*self*, *path1*, *path2*, *acl=None*)
Duplicate remote file

getxattr (*self*, *path*, *attr*)
Get user-defined metadata attribute

info (*self*, *path*, ***kwargs*)
File information about this path.

invalidate_cache (*self*, *path=None*)
Invalidate listing cache for given path, it is reloaded on next use.

Parameters

path: string or None If None, clear all listings cached else listings at or under given path.

classmethod load_tokens ()
Get "browser" tokens from disc

ls (*self*, *path*, *detail=False*, ***kwargs*)
List objects under the given '/{bucket}/{prefix} path.

merge (*self*, *path*, *paths*, *acl=None*)
Concatenate objects within a single bucket

mkdir (*self*, *bucket*, *acl='projectPrivate'*, *default_acl='bucketOwnerFullControl'*)
New bucket

Parameters

bucket: str bucket name. If contains '/' (i.e., looks like subdir), will have no effect because GCS doesn't have real directories.

acl: string, one of bACLs access for the bucket itself

default_acl: str, one of ACLs default ACL for objects created in this bucket

rm (*self*, *path*, *recursive=False*, *maxdepth=None*)
Delete keys.

If a list, batch-delete all keys in one go (can span buckets)

Returns whether operation succeeded (a list if input was a list)

If recursive, delete all keys given by find(path)

rmdir (*self*, *bucket*)
Delete an empty bucket

Parameters

bucket: str bucket name. If contains '/' (i.e., looks like subdir), will have no effect because GCS doesn't have real directories.

setxattrs (*self*, *path*, *content_type=None*, *content_encoding=None*, ***kwargs*)
Set/delete/add writable metadata attributes

content_type: str If not None, set the content-type to this value

content_encoding: str If not None, set the content-encoding. See https://cloud.google.com/storage/docs/transcoding

kw_args: key-value pairs like field="value" or field=None value must be string to add or modify, or None to delete

Returns

Entire metadata after update (even if only path is passed)

classmethod split_path(*path*)
Normalise GCS path string into bucket and key.

Parameters

path [string] Input path, like *gcs://mybucket/path/to/file*. Path is of the form: '[gs|gcs://]bucket[/key]'

Returns

(bucket, key) tuple

classmethod url(*path*)
Get HTTP URL of the given path

class gcsfs.core.**GCSFile**(*gcsfs*, *path*, *mode='rb'*, *block_size=5242880*, *autocommit=True*, *cache_type='readahead'*, *cache_options=None*, *acl=None*, *consistency='md5'*, *metadata=None*, *content_type=None*, ***kwargs*)

Attributes

closed

Methods

close(self)	Close file
commit(self)	If not auto-committing, finalize file
discard(self)	Cancel in-progress multi-upload
fileno(self, /)	Returns underlying file descriptor if one exists.
flush(self[, force])	Write buffered data to backend store.
info(self)	File information about this path
isatty(self, /)	Return whether this is an 'interactive' stream.
read(self[, length])	Return data from cache, or fetch pieces as necessary
readable(self)	Whether opened for reading
readinto(self, b)	mirrors builtin file's readinto method
readline(self)	Read until first occurrence of newline character
readlines(self)	Return all data, split by the newline character
readuntil(self[, char, blocks])	Return data between current position and first occurrence of char
seek(self, loc[, whence])	Set current file location
seekable(self)	Whether is seekable (only in read mode)
tell(self)	Current file location
truncate()	Truncate file to size bytes.
url(self)	HTTP link to this file's data
writable(self)	Whether opened for writing
write(self, data)	Write data to buffer.

readinto1	
writelines	

commit (*self*)
If not auto-committing, finalize file

discard (*self*)
Cancel in-progress multi-upload

Should only happen during discarding this write-mode file

info (*self*)
File information about this path

url (*self*)
HTTP link to this file's data

4.2 For Developers

We welcome contributions to gcsfs!

Please file issues and requests on github and we welcome pull requests.

4.2.1 Testing and VCR

VCR records requests to the remote server, so that they can be replayed during tests - so long as the requests match exactly the original. It is set to strip out sensitive information before writing the request and responses into yaml files in the tests/recordings/ directory; the file-name matches the test name, so all tests must have unique names, across all test files.

The process is as follows:

- Create a bucket for testing
- Set environment variables so that the tests run against your GCS credentials, and recording occurs

```
export GCSFS_RECORD_MODE=all
export GCSFS_TEST_PROJECT='...'
export GCSFS_TEST_BUCKET='...'  # the bucket from step 1 (without gs:// prefix).
export GCSFS_GOOGLE_TOKEN=~/.config/gcloud/application_default_credentials.json
py.test -vv -x -s gcsfs
```

  If ~/.config/gcloud/application_default_credentials.json file does not exist, run `gcloud auth application-default login` These variables can also be set in `gcsfs/tests/settings.py`

- Run this again, setting `GCSFS_RECORD_MODE=once`, which should alert you if your tests make different requests the second time around
- Finally, test as TravisCI will, using only the recordings

```
export GCSFS_RECORD_MODE=none
py.test -vv -x -s gcsfs
```

To reset recording and start again, delete the yaml file corresponding to the test in `gcsfs/tests/recordings/*.yaml`.

4.3 GCSFS and FUSE

Warning, this functionality is **experimental**

FUSE is a mechanism to mount user-level filesystems in unix-like systems (linux, osx, etc.). GCSFS is able to use FUSE to present remote data/keys as if they were a directory on your local file-system. This allows for standard shell command manipulation, and loading of data by libraries that can only handle local file-paths (e.g., netCDF/HDF5).

4.3.1 Requirements

In addition to a standard installation of GCSFS, you also need:

- libfuse as a system install. The way to install this will depend on your OS. Examples include `sudo apt-get install fuse`, `sudo yum install fuse` and download from osxfuse.
- fusepy, which can be installed via conda or pip
- pandas, which can also be installed via conda or pip (this library is used only for its timestring parsing.

4.3.2 Usage

FUSE functionality is available via the `fsspec.fuse` module. See the docstrings for further details.

```
gcs = gcsfs.GCSFileSystem(..)
from fsspec.fuse import run
run(gcs, "bucket/path", "local/path", foreground=True, threads=False)
```

4.3.3 Caveats

This functionality is experimental. The command usage may change, and you should expect exceptions.

Furthermore:

- although mutation operations tentatively work, you should not at the moment depend on gcsfuse as a reliable system that won't loose your data.
- permissions on GCS are complicated, so all files will be shown as fully-open 0o777, regardless of state. If a read fails, you likely don't have the right permissions.

4.4 Changelog

4.5 Version 0.6.0

- **API-breaking**: Changed requester-pays handling for `GCSFileSystem`.

  The `user_project` keyword has been removed, and has been replaced with the `requester_pays` keyword. If you're working with a `requester_pays` bucket you will need to explicity pass `requester_pays=True`. This will include your `project` ID in requests made to GCS.

4.6 Version 0.5.3

- `GCSFileSystem` now validates that the `project` provided, if any, matches the Google default project when using `token='google_default'` to authenticate (PR #219).
- Fixed bug in `GCSFileSystem.cat` on objects in requester-pays buckets (PR #217).

4.7 Version 0.5.2

- Fixed bug in `user_project` fallback for default Google authentication (PR #213)

4.8 Version 0.5.1

- `user_project` now falls back to the `project` if provided (PR #208)

4.9 Version 0.5.0

- Added the ability to make requester-pays requests with the `user_project` parameter (PR #206)

4.10 Version 0.4.0

- Improved performance when serializing filesystem objects (PR #182)
- Fixed authorization errors when using `gcsfs` within multithreaded code (PR #183, PR #192)
- Added contributing instructions (PR #185)
- Improved performance for `gcsfs.GCSFileSystem.info()` (PR #187)
- Fixed bug in `gcsfs.GCSFileSystem.info()` raising an error (PR #190)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index