gcsfs Documentation

Release 1.4.2.dev1+ga61b311

Othoz GmbH

Aug 06, 2020

Contents

A Python filesystem abstraction of Google Cloud Storage (GCS) implemented as a PyFilesystem2 extension.

With GCSFS, you can interact with Google Cloud Storage as if it was a regular filesystem.

Apart from the nicer interface, this will highly decouple your code from the underlying storage mechanism: Exchanging the storage backend with an in-memory filesystem for testing or any other filesystem like S3FS becomes as easy as replacing `gs://bucket_name` with `mem://` or `s3://bucket_name`.

For a full reference on all the PyFilesystem possibilities, take a look at the PyFilesystem Docs!

CHAPTER 1

Documentation

- GCSFS Documentation
- PyFilesystem Wiki
- PyFilesystem Reference

CHAPTER 2

Installing

Install the latest GCSFS version by running:

```
$ pip install fs-gcsfs
```

Or in case you are using conda:

```
$ conda install -c conda-forge fs-gcsfs
```

CHAPTER 3

Examples

Instantiating a filesystem on Google Cloud Storage (for a full reference visit the Documentation):

```
from fs_gcsfs import GCSFS
gcsfs = GCSFS(bucket_name="mybucket")
```

Alternatively you can use a FS URL to open up a filesystem:

```
from fs import open_fs
gcsfs = open_fs("gs://mybucket/root_path?project=test&api_endpoint=http%3A//localhost
↪%3A8888&strict=False")
```

Supported query parameters are:

- *project* (str): Google Cloud project to use
- *api_endpoint* (str): URL-encoded endpoint that will be passed to the GCS client's client_options
- *strict* ("True" or "False"): Whether GCSFS will be opened in strict mode

You can use GCSFS like your local filesystem:

```
>>> from fs_gcsfs import GCSFS
>>> gcsfs = GCSFS(bucket_name="mybucket")
>>> gcsfs.tree()
├── foo
│   ├── bar
│   │   ├── file1.txt
│   │   └── file2.csv
│   └── baz
│       └── file3.txt
└── file4.json
>>> gcsfs.listdir("foo")
["bar", "baz"]
>>> gcsfs.isdir("foo/bar")
True
```

Uploading a file is as easy as:

```
from fs_gcsfs import GCSFS
gcsfs = GCSFS(bucket_name="mybucket")
with open("local/path/image.jpg", "rb") as local_file:
    with gcsfs.open("path/on/bucket/image.jpg", "wb") as gcs_file:
        gcs_file.write(local_file.read())
```

You can even sync an entire bucket on your local filesystem by using PyFilesystem's utility methods:

```
from fs_gcsfs import GCSFS
from fs.osfs import OSFS
from fs.copy import copy_fs

gcsfs = GCSFS(bucket_name="mybucket")
local_fs = OSFS("local/path")

copy_fs(gcsfs, local_fs)
```

For exploring all the possibilities of GCSFS and other filesystems implementing the PyFilesystem interface, we recommend visiting the official PyFilesystem Docs!

CHAPTER 4

Development

To develop on this project make sure you have pipenv installed and run the following from the root directory of the project:

```
$ pipenv install --dev --three
```

This will create a virtualenv with all packages and dev-packages installed.

CHAPTER 5

Tests

All CI tests run against an actual GCS bucket provided by Othoz.

In order to run the tests against your own bucket, make sure to set up a Service Account with all necessary permissions:

- storage.objects.get
- storage.objects.list
- storage.objects.create
- storage.objects.update
- storage.objects.delete

All five permissions listed above are e.g. included in the predefined Cloud Storage IAM Role `roles/storage.objectAdmin`.

Expose your bucket name as an environment variable `$TEST_BUCKET` and run the tests via:

```
$ pipenv run pytest
```

Note that the tests mostly wait for I/O, therefore it makes sense to highly parallelize them with xdist, e.g. by running the tests with:

```
$ pipenv run pytest -n 10
```

CHAPTER 6

Credits

Credits go to S3FS which was the main source of inspiration and shares a lot of code with GCSFS.

CHAPTER 7

Limitations

A filesystem built on top of an object store like GCS suffers from the same limitations as the ones mentioned in S3FS.

GCS does not offer true directories which is why GCSFS (as well as S3FS) will simulate the existence of a directory called `foo` by adding an empty blob called `foo/`. Any filesystem content that was not created via GCSFS will lack these directory markers which may lead to wrong behaviour. For example `gcsfs.isdir("bar")` will return `False` if the marker blob `bar/` does not exist, even though there might exist a blob called `bar/baz.txt`.

To overcome this you can call the utility method *`fix_storage()`* on your GCSFS instance which will walk the entire filesystem (i.e. the entire `bucket` or the "subdirectory" you specified via `root_path`) and add all missing directory markers.

Warning: Listing and fixing large buckets may take some time!

CHAPTER 8

Reference

For a full reference of all available methods of GCSFS visit the documentation of fs.base.FS!

8.1 GCSFS

class fs_gcsfs.**GCSFS**(*bucket_name: str*, *root_path: str = None*, *create: bool = False*, *client: google.cloud.storage.client.Client = None*, *retry: int = 5*, *strict: bool = True*)

A Google Cloud Storage filesystem for PyFilesystem.

This implementation is based on S3FS.

Args: bucket_name: The GCS bucket name. root_path: The root directory within the GCS Bucket. create: Whether to create `root_path` on initialization or not. If `root_path` does not yet exist and `create=False` a `CreateFailed`

exception will be raised. To disable `root_path` validation entirely set `strict=False`.

client: A `google.storage.Client` exposing the google storage API. strict: When `True` (default) GCSFS will follow the PyFilesystem specification exactly. Set to `False` to disable validation of destination paths

which may speed up some operations.

fix_storage() → None

Utility function that walks the entire *root_path* and makes sure that all intermediate directories are correctly marked with empty blobs.

As GCS is no real file system but only a key-value store, there is also no concept of folders. S3FS and GCSFS overcome this limitation by adding empty files with the name "<path>/" every time a directory is created, see https://fs-gcsfs.readthedocs.io/en/latest/#limitations.

8.2 GCSMap

GCSFS.**get_mapper**() → fs_gcsfs._gcsfs.GCSMap
Returns a MutableMapping that represents the filesystem.

The keys of the mapping become files and the values (which must be bytes) the contents of those files. This is particularly useful to be used with libraries such as xarray or zarr.

CHAPTER 9

Powered By

This PyFilesystem extension was created by Othoz GmbH

Index

F

G