Cloud Storage Documentation

Release 0.7.0

Scott Werner

Oct 31, 2018

Contents

1 Usage **3**

2 Supported Services **5**

3 Installation **7**

4 General **9**
- 4.1 Installation . . . 9
- 4.2 Supported Services . . . 10
- 4.3 Quick Start . . . 10
- 4.4 Advanced . . . 14

5 Developer **23**
- 5.1 API Reference . . . 23

6 Other **43**
- 6.1 Changelog . . . 43
- 6.2 Authors . . . 44
- 6.3 License . . . 44

7 Links **47**

Python Module Index **49**

Cloud Storage is a Python +3.4 package which creates a unified API for the cloud storage services: Amazon Simple Storage Service (S3), Microsoft Azure Storage, Rackspace Cloud Files, Google Cloud Storage, and the Local File System.

Cloud Storage is inspired by Apache Libcloud. Advantages to Apache Libcloud Storage are:

- Full Python 3 support.
- Generate temporary signed URLs for downloading and uploading files.
- Support for request and response headers like Content-Disposition.
- Pythonic! Iterate through all blobs in containers and all containers in storage using respective objects.

CHAPTER 1

Usage

```
>>> from cloudstorage.drivers.amazon import S3Driver
>>> storage = S3Driver(key='<my-aws-access-key-id>', secret='<my-aws-secret-access-
↪key>')

>>> container = storage.create_container('avatars')
>>> container.cdn_url
'https://avatars.s3.amazonaws.com/'

>>> avatar_blob = container.upload_blob('/path/my-avatar.png')
>>> avatar_blob.cdn_url
'https://s3.amazonaws.com/avatars/my-avatar.png'

>>> avatar_blob.generate_download_url(expires=3600)
'https://avatars.s3.amazonaws.com/my-avatar.png?'
'AWSAccessKeyId=<my-aws-access-key-id>'
'&Signature=<generated-signature>'
'&Expires=1491849102'

>>> container.generate_upload_url('user-1-avatar.png', expires=3600)
{
    'url': 'https://avatars.s3.amazonaws.com/',
    'fields': {
        'key': 'user-1-avatar.png',
        'AWSAccessKeyId': '<my-aws-access-key-id>',
        'policy': '<generated-policy>',
        'signature': '<generated-signature>'
    }
}
```

CHAPTER 2

Supported Services

- Amazon S3
- Google Cloud Storage
- Microsoft Azure Storage
- Rackspace CloudFiles
- Local File System

CHAPTER 3

Installation

To install Cloud Storage:

```
pip install cloudstorage
```

Also install the storage driver(s) you will be using:

```
pip install cloudstorage[amazon]
pip install cloudstorage[google]
pip install cloudstorage[local]
pip install cloudstorage[microsoft]
pip install cloudstorage[rackspace]
```

CHAPTER 4

General

4.1 Installation

4.1.1 Install

You can install the latest stable version of Cloud Storage using pip:

```
pip install cloudstorage
```

Also install the storage driver(s) you will be using:

```
pip install cloudstorage[amazon]
pip install cloudstorage[google]
pip install cloudstorage[local]
pip install cloudstorage[microsoft]
pip install cloudstorage[rackspace]
```

If you don't have pip installed, this Python installation guide can guide you through the process.

4.1.2 Source Code

Cloud Storage is actively developed on GitHub, where the code is always available.

You can either clone the public repository:

```
git clone https://github.com/scottwernervt/cloudstorage.git
```

Or, download the tarball:

```
curl -OL https://github.com/scottwernervt/cloudstorage/tarball/master
```

Once you have a copy of the source, you can embed it in your own Python package, or install it into your site-packages easily:

```
python setup.py install
```

4.2 Supported Services

Driver	Driver Class	Driver Name
Amazon S3	S3Driver	S3
Blackblaze B2 Cloud Storage	TODO	
Google Cloud Storage	GoogleStorageDriver	GOOGLESTORAGE
Microsoft Azure Storage	AzureStorageDriver	AZURE
Rackspace CloudFiles	CloudFilesDriver	CLOUDFILES
Local	LocalDriver	LOCAL

Do not see your provider? Create an issue and vote for at cloudstorage issues.

4.3 Quick Start

4.3.1 Basic Terminology

- Blobs are objects, keys, or files.
- Containers (buckets) manage blobs.
- Storage Driver initiates a connection to the storage backend and manage containers.

4.3.2 Connecting to Storage

Let's start with creating a Local File System storage driver (replace `key` argument with a folder path of your choosing):

```
from cloudstorage.drivers.local import LocalDriver

storage = LocalDriver(key='/home/webapp/storage', secret='<my-secret>')
# <Driver: LOCAL>
```

Alternatively, the driver can be initialized with its name. This is useful if you have different configurations for testing vs production. For example, a Flask app might use the `LOCAL` driver for testing and `S3` for production.

```
from cloudstorage import get_driver_by_name

driver_cls = get_driver_by_name('LOCAL')
storage = driver_cls(key='/home/webapp/storage', secret='<my-secret>')
# <Driver: LOCAL>
```

4.3.3 Creating a Container

Creating a container:

```
container = storage.create_container('container-name')
# <Container container-name LOCAL>
```

4.3.4 Accessing a Container

Getting a container:

```
container = storage.get_container('container-name')
# <Container container-name LOCAL>
```

4.3.5 Deleting a Container

All of the blob objects in a container must be deleted before the container itself can be deleted:

```
container = storage.get_container('container-name')

for blob in container:
    blob.delete()

container.delete()
```

4.3.6 Uploading a Blob

Storing data from a file, stream, or string:

```
picture_path = '/path/picture.png'
picture_blob = container.upload_blob(picture_path)
# <Blob picture.png container-name LOCAL>
```

```
with open('/path/picture.png', 'rb') as picture_file:
    picture_blob = container.upload_blob(picture_file, blob_name='picture.png')
    # <Blob picture.png container-name LOCAL>
```

Cloud Storage will attempt to guess the uploaded file's `Content-Type` using `mimetypes` and python-magic. The `Content-Type` can be overridden with the `content_type` argument:

```
with open('/path/picture.png', 'rb') as picture_file:
    picture_blob = container.upload_blob(filename=picture_file,
                                         content_type='application/octet-stream')
    # <Blob picture.png container-name LOCAL>
    picture_blob.content_type
    # 'application/octet-stream'
```

Important: Always use read binary mode `rb` when uploading a file like object.

Warning: The effect of uploading to an existing blob depends on the "versioning" and "lifecycle" policies defined on the blob's container. In the absence of those policies, upload will overwrite any existing contents. As of now, Cloud Storage does not supporting versioning/generation.

4.3.7 Accessing a Blob

To get a blob from a container and its attributes:

```
container = storage.get_container('container-name')
picture_blob = container.get_blob('picture.png')
picture_blob.name
# 'picture.png'
picture_blob.size
# 50301
picture_blob.checksum
# '2f907a59924ad96b7478074ed96b05f0'
picture_blob.etag
# 'bf506fc6ffbc3c4a2756eac85a0b4d2f3f227fee'
picture_blob.content_type
# 'image/png'
picture_blob.created_at
# datetime.datetime(2017, 4, 19, 18, 38, 26, 335373)
```

4.3.8 Downloading a Blob

Downloading a blob data to a file path:

```
picture_blob = container.get_blob('picture.png')
picture_blob.download('/path/picture-copy.png')
```

Or to a file like object:

```
picture_blob = container.get_blob('picture.png')
with open('/path/picture-copy.png', 'wb') as picture_file:
    picture_blob.download(picture_file)
```

Important: Always use write binary mode `wb` when downloading a blob to a file like object.

4.3.9 Deleting a Blob

Deleting a blob:

```
picture_blob = container.get_blob('picture.png')
picture_blob.delete()
```

4.3.10 Generate a Download Url

Generates a signed URL to download a blob:

```
from urllib.parse import urlencode

import requests

storage_url = 'http://localhost/storage'

picture_blob = container.get_blob('picture.png')
signature = picture_blob.generate_download_url(expires=120)
```

(continues on next page)

(continued from previous page)

```
url_params = {
    'signature': signature,
    'filename': 'picture.png',
}
download_url = storage_url + '?' + urlencode(url_params)
# 'http://localhost/storage?signature=<generated-signature>&filename=picture.png'

response = requests.get(download_url)
# <Response [200]>

with open('/path/picture-download.png', 'wb') as picture_file:
    for chunk in response.iter_content(chunk_size=128):
        picture_file.write(chunk)
```

4.3.11 Generate an Upload FormPost

Generate a signature and policy for uploading objects to a container:

```
import requests

container = storage.get_container('container-name')
form_post = container.generate_upload_url('avatar.png', expires=120)

url = form_post['url']
fields = form_post['fields']
multipart_form_data = {
    'file': open('/path/picture.png', 'rb'),
}

response = requests.post(url, data=fields, files=multipart_form_data)
# <Response [204]>
```

4.3.12 Iteration of Containers and Blobs

Storage and containers are both iterable:

```
for container in storage:
    container.name
    # 'container-a', 'container-b', ...

    for blob in container:
        blob.name
        # 'blob-1', 'blob-2', ...
```

Check if a container or container name exists in storage:

```
container = storage.get_container('container-name')
container in storage
# True
'container-name' in storage
# True
```

Check if a blob or blob name exists in a container:

```
container = storage.get_container('container-name')
picture_blob = container.get_blob('picture.png')
picture_blob in container
# True
'picture.png' in container
# True
```

4.3.13 Metadata and Extra Arguments

If supported by the driver, extra arguments can be included with operations on containers and blobs. For example, `meta_data` can be saved to a blob object or `Content-Disposition` set to inline or attachment.

```
options = {
    'acl': 'public-read',
    'content_disposition': 'attachment; filename="user-1-avatar.png"',
    'content_type': 'image/png',
    'cache_control': 'max-age=86400',
    'meta_data': {
        'owner-id': '1',
        'owner-email': 'user.one@startup.com',
    }
}

picture_path = '/path/picture.png'
picture_blob = container.upload_blob(picture_path, **options)
picture_blob.content_disposition
# 'attachment; filename="user-1-avatar.png"'
picture_blob.cache_control
# 'max-age=86400'
picture_blob.meta_data
# {'owner-id': '1', 'owner-email': 'user.one@startup.com'}
```

Tip: It is recommended to save to meta data keys with dashes, `owner-id`, instead of with underscores, `owner_id`. Some drivers will allow underscores but other drivers will automatically convert them to dashes.

Proceed to the *Advanced section* for individual driver documentation and advanced usages like generating presigned upload and download URLs.

4.4 Advanced

This section contains extra documentation for each driver. For more examples and usage, check out the API documentation for *Blob*, *Container*, and *Driver*.

4.4.1 Amazon Simple Storage Service (S3)

Amazon `S3Driver` is a wrapper around Boto 3.

Connecting

Change region from default `us-east-1` to `us-west-`:

```
from cloudstorage.drivers.amazon import S3Driver

storage = S3Driver(key='<my-aws-access-key-id>',
                   secret='<my-aws-secret-access-key>',
                   region='us-west-1')
# <Driver: S3 us-west-1>
```

Regions supported:

- ap-northeast-1
- ap-northeast-2
- ap-south-1
- ap-southeast-1
- ap-southeast-2
- ca-central-1
- eu-central-1
- eu-west-1
- eu-west-2
- sa-east-1
- us-east-1
- us-east-2
- us-west-1
- us-west-2

Access Control List (ACL)

By default, all containers and blobs default to `private`. To change the access control when creating a container or blob, include the `acl` argument option:

```
container = storage.create_container('container-public', acl='public-read')
container.cdn_url
# https://s3.amazonaws.com/container-public
```

```
container = storage.get_container('container-public')
picture_blob = container.upload_blob('/path/picture.png', acl='public-read')
picture_blob.cdn_url
# https://s3.amazonaws.com/container-public/picture.png
```

Support ACL values for S3:

- private
- public-read
- public-read-write
- authenticated-read
- bucket-owner-read
- bucket-owner-full-control

- aws-exec-read

Warning: Updating ACL on an existing container or blob is not currently supported.

4.4.2 Google Cloud Storage

`GoogleStorageDriver` is a wrapper around google-cloud-storage.

Connecting

The driver will check for `GOOGLE_APPLICATION_CREDENTIALS` environment variable before connecting. If not found, the driver will use service worker credentials json file path passed to `key` argument.

```
from cloudstorage.drivers.google import GoogleStorageDriver

credentials_json_file = '/path/cloud-storage-service-account.json'
storage = GoogleStorageDriver(key=credentials_json_file)
# <Driver: GOOGLESTORAGE>
```

Access Control List (ACL)

By default, all containers and blobs default to `project-private`. To change the access control when creating a container or blob, include the `acl` argument:

```
container = storage.create_container('container-public', acl='public-read')
container.cdn_url
# https://storage.googleapis.com/container-public
```

```
container = storage.get_container('container-public')
picture_blob = container.upload_blob('/path/picture.png', acl='public-read')
picture_blob.cdn_url
# https://storage.googleapis.com/container-public/picture.png
```

Support ACL values for Google Cloud Storage:

- private
- public-read
- public-read-write
- authenticated-read
- bucket-owner-read
- bucket-owner-full-control
- project-private

Warning: Updating ACL on an existing container or blob is not currently supported.

Content Delivery Network (CDN)

Calling `container.enable_cdn()` will make the container public (shared publicly). More information available at Making Data Public.

4.4.3 Microsoft Azure Storage

Microsoft `AzureStorageDriver` is a wrapper around Azure Storage SDK for Python.

Connecting

```
from cloudstorage.drivers.microsoft import AzureStorageDriver

storage = AzureStorageDriver(account_name='<my-azure-account-name>',
           key='<my-azure-account-key>')
# <Driver: AZURE>
```

Access Control List (ACL)

By default, all containers and blobs default to `private` (public read access). The following container permissions are supported: `container-public-access` (full public read access) and `blob-public-access` (public read access for blobs only).

```
container = storage.create_container('container-public',
                                     acl='container-public-access')
container.cdn_url
# https://s3.amazonaws.com/container-public
```

Support ACL values for Azure:

- private
- container-public-access
- blob-public-access

Warning: Updating ACL on an existing container is not currently supported.

4.4.4 Rackspace Cloud Files

Rackspace `CloudFilesDriver` extends rackspacesdk which is a wrapper around OpenStack SDK.

Connecting

Change region from default Northern Virginia (`IAD`) to Dallas-Fort Worth (`DFW`):

```
from cloudstorage.drivers.rackspace import CloudFilesDriver

storage = CloudFilesDriver(key='<my-rackspace-username>',
                           secret='<my-rackspace-secret-key>',
```

(continues on next page)

(continued from previous page)

```
                              region='DFW')
# <Driver: CLOUDFILES IAD>
```

Regions supported:

- Dallas-Fort Worth (`DFW`)
- Chicago (`ORD`)
- Northern Virginia (`IAD`)
- London (`LON`)
- Sydney (`SYD`)
- Hong Kong (`HKG`)

Access Control List (ACL)

> **Warning:** Cloud Storage does not currently support canned Access Control List (ACL) for Containers and Blobs.

Content Delivery Network (CDN)

You must enable CDN on the container before accessing a blob's CDN URL.

```
container = storage.create_container('container-public')
container.enable_cdn()
# True
container.cdn_url
# https://XXXXXX-XXXXXXXXXXXX.ssl.cf5.rackcdn.com

picture_blob = container.upload_blob('/path/picture.png')
picture_blob.cdn_url
# https://XXXXXX-XXXXXXXXXXXX.ssl.cf5.rackcdn.com/picture.png
```

4.4.5 Local File System Driver

`LocalDriver` can be used as a full storage backend on backend or for testing in development.

Connecting

```
from cloudstorage.drivers.local import LocalDriver

storage = LocalDriver(key='/home/webapp/storage',
                      secret='<secret-signed-urls>')
# <Driver: LOCAL>
```

Metadata

Warning: Metadata and other attributes are saved as extended file attributes using the package xattr. Extended attributes are currently only available on Darwin 8.0+ (Mac OS X 10.4) and Linux 2.6+. Experimental support is included for Solaris and FreeBSD.

```
container = storage.get_container('container-name')

meta_data = {
    'owner-id': '1',
    'owner-email': 'user.one@startup.com',
}

picture_blob = container.upload_blob('/path/picture.png', meta_data=meta_data)
picture_blob.meta_data
# {'owner-id': '1', 'owner-email': 'user.one@startup.com'}
```

Verify extended attributes on Linux:

```
$ getfattr -d /path/picture.png
# file: picture.png
user.content-type="image/png"
user.metadata.owner-email="user.one@startup.com"
user.metadata.owner-id="1"
```

Generate a Download Url

You can optionally share blobs with others by creating a pre-signed URL which grants time-limited permission to download the blobs. Below will generate a URL that expires in 2 minutes (120 seconds):

```
picture_blob = container.get_blob('picture.png')
signature = picture_blob.generate_download_url(expires=120)
# '<generated-signature>'
```

The signature can then be appended as a url query parameter to your web apps storage route:

```
from urllib.parse import urlencode

storage_url = 'http://localhost/storage'
url_params = {
    'signature': signature,
    'filename': 'picture.png',
}

download_url = storage_url + '?' + urlencode(url_params)
# 'http://localhost/storage?signature=<generated-signature>&filename=picture.png'
```

The user clicks the download URL link and the backend validates the signature:

```
from urllib.parse import urlparse, parse_qs

o = urlparse(download_url)
query = parse_qs(o.query)
```

(continues on next page)

(continued from previous page)

```
# {'signature': ['<generated-signature>'], 'filename': ['picture.png']}

signature = query['signature'][0]
payload = storage.validate_signature(signature)
# {
#   'max_age': 120,
#   'expires': 1492583288,
#   'blob_name': 'picture.png',
#   'container': 'container-name',
#   'method': 'GET',
#   'content_disposition': None
# }

container_request = storage.get_container(payload['container'])
blob_request = container_request.get_blob(payload['blob_name'])
blob_request.path
# 'container-name/picture.png'
```

If the signature has expired, `LocalDriver.validate_signature()` will raise *`SignatureExpiredError`*. Finally, the web app would serve the static file over Apache or Nginx (or other web server) using request header like `X-SendFile` or by stream the file contents.

Generate an Upload FormPost

Generates a signed URL to upload a file to a container that expires in 120 seconds (2 minutes):

```
container = storage.get_container('container-name')

options = {
    'expires': 120,
    'content_disposition': 'inline; filename=avatar-user-1.png',
    'meta_data': {
        'owner-id': '1',
        'owner-email': 'user.one@startup.com',
    },
}
form_post = container.generate_upload_url('avatar-user-1.png', **options)
# {
#   'url': '',
#   'fields': {
#     'blob_name': 'avatar-user-1.png',
#     'container': 'container-name',
#     'expires': 1492629357,
#     'signature': '<generated-signature>'
#   }
# }
```

Generate a form with `method="POST"` and `enctype="multipart/form-data"` with the fields above:

```
post_url = 'http://localhost/storage'
fields = [
    '<input type="hidden" name="{name}" value="{value}" />'.format(
        name=name, value=value)
    for name, value in form_post['fields'].items()
]
```

(continues on next page)

(continued from previous page)

```
upload_form = [
    '<form action="{url}" method="post" '
    'enctype="multipart/form-data">'.format(
        url=post_url),
    *fields,
    '<input name="file" type="file" />',
    '<input type="submit" value="Upload" />',
    '</form>',
]

print('\n'.join(upload_form))
```

```
<form action="http://localhost/storage" method="post" enctype="multipart/form-data">
    <input type="hidden" name="blob_name" value="avatar-user-1.png" />
    <input type="hidden" name="container" value="container-name" />
    <input type="hidden" name="expires" value="1492630156" />
    <input type="hidden" name="signature" value="<generated-signature>" />
    <input name="file" type="file" />
    <input type="submit" value="Upload" />
</form>
```

The user uploads a file to your route `http://localhost/storage` with method `POST` and the signature can be validated with:

```
signature = request.form['signature']
payload = storage.validate_signature(signature)
# {
#   'acl': None,
#   'meta_data': {
#     'owner-id': '1',
#     'owner-email': 'user.one@startup.com'
#   },
#   'content_disposition': 'inline; filename=avatar-user-1.png',
#   'content_length': None,
#   'content_type': None,
#   'max_age': 3600,
#   'blob_name': 'avatar-user-1.png',
#   'container': 'container-name',
#   'expires': 1492631817
# }

container = storage.get_container(payload['container'])

blob = container.upload_blob(filename=request.files['file'],
                             blob_name=payload['blob_name'],
                             acl=payload.get('acl'),
                             meta_data=payload.get('meta_data'),
                             content_type=payload.get('content_type'),
                             content_disposition=payload.get('content_disposition'))
# <Blob avatar-user-1.png container-name LOCAL>
```

CHAPTER 5

Developer

5.1 API Reference

5.1.1 Base

Blob

class cloudstorage.base.**Blob**(*name*, *checksum*, *etag*, *size*, *container*, *driver*, *acl=None*, *meta_data=None*, *content_disposition=None*, *content_type=None*, *cache_control=None*, *created_at=None*, *modified_at=None*, *expires_at=None*)

Represents an object blob.

```
picture_blob = container.get_blob('picture.png')
picture_blob.size
# 50301
picture_blob.checksum
# '2f907a59924ad96b7478074ed96b05f0'
picture_blob.content_type
# 'image/png'
picture_blob.content_disposition
# 'attachment; filename=picture-attachment.png'
```

Parameters

- **name** (*str*) – Blob name (must be unique in container).
- **checksum** (*str*) – Checksum of this blob.
- **etag** (*str*) – Blob etag which can also be the checksum. The etag for `LocalDriver` is a SHA1 hexdigest of the blob's full path.
- **size** (*int*) – Blob size in bytes.
- **container** (`Container`) – Reference to the blob's container.

- **driver** (*Driver*) – Reference to the blob's container's driver.
- **meta_data** (*Dict[str, str] or None*) – (optional) Metadata stored with the blob.
- **acl** (*dict or None*) – (optional) Access control list (ACL) for this blob.
- **content_disposition** (*str or None*) – (optional) Specifies presentational information for this blob.
- **content_type** (*str or None*) – (optional) A standard MIME type describing the format of the object data.
- **created_at** (*datetime.datetime or None*) – (optional) Creation time of this blob.
- **modified_at** (*datetime.datetime or None*) – (optional) Last modified time of this blob.
- **expires_at** (*datetime.datetime or None*) – (optional) Deletion or expiration time for this blob.

__len__()

The blob size in bytes.

Returns bytes

Return type int

cdn_url

The Content Delivery Network URL for this blob.

```
https://container-name.storage.com/picture.png
```

Returns The CDN URL for this blob.

Return type str

path

Relative URL path for this blob.

```
container-name/picture.png
```

Returns The relative URL path to this blob.

Return type str

delete()

Delete this blob from the container.

```
picture_blob = container.get_blob('picture.png')
picture_blob.delete()
picture_blob in container
# False
```

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the blob object doesn't exist.

download(*destination*)

Download the contents of this blob into a file-like object or into a named file.

Filename:

```
picture_blob = container.get_blob('picture.png')
picture_blob.download('/path/picture-copy.png')
```

File object:

Important: Always use write binary mode `wb` when downloading a blob to a file object.

```
picture_blob = container.get_blob('picture.png')
with open('/path/picture-copy.png', 'wb') as picture_file:
    picture_blob.download(picture_file)
```

Parameters **destination** (*file or str*) – A file handle to which to write the blob's data or a filename to be passed to `open`.

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the blob object doesn't exist.

generate_download_url (*expires=3600*, *method='GET'*, *content_disposition=None*, *extra=None*)

Generates a signed URL for this blob.

If you have a blob that you want to allow access to for a set amount of time, you can use this method to generate a URL that is only valid within a certain time period. This is particularly useful if you don't want publicly accessible blobs, but don't want to require users to explicitly log in.[1]

Basic example:

```
import requests

picture_blob = container.get_blob('picture.png')
download_url = picture_blob.download_url(expires=3600)

response = requests.get(download_url)
# <Response [200]>

with open('/path/picture-download.png', 'wb') as picture_file:
    for chunk in response.iter_content(chunk_size=128):
        picture_file.write(chunk)
```

Response Content-Disposition example:

```
picture_blob = container.get_blob('picture.png')

params = {
    'expires': 3600,
    'content_disposition': 'attachment; filename=attachment.png'
}
download_url = picture_blob.download_url(**params)

response = requests.get(download_url)
# <Response [200]>
response.headers['content-disposition']
# attachment; filename=attachment.png
```

[1] Blobs / Objects — google-cloud 0.24.0 documentation

References:

- Boto 3: S3.Client.generate_presigned_url
- Google Cloud Storage: generate_signed_url
- Rackspace: TempURL

Parameters

- **expires** (*int*) – (optional) Expiration in seconds.
- **method** (*str*) – (optional) HTTP request method. Defaults to `GET`.
- **content_disposition** (*str or None*) – (optional) Sets the Content-Disposition header of the response.
- **extra** (*Dict[str, str] or None*) – (optional) Extra parameters for the request.
  - All
    - **content_type** *(str)* – Sets the Content-Type header of the response.
  - Google Cloud Storage
    - **version** *(str)* – A value that indicates which generation of the resource to fetch.
  - Amazon S3
    - **version_id** *(str)* – Version of the object.

Returns Pre-signed URL for downloading a blob. `LocalDriver` returns urlsafe signature.

Return type str

patch()

Saves all changed attributes for this blob.

> **Warning:** Not supported by all drivers yet.

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the blob object doesn't exist.

Container

class cloudstorage.base.**Container**(*name*, *driver*, *acl=None*, *meta_data=None*, *created_at=None*)

Represents a container (bucket or folder) which contains blobs.

```
container = storage.get_container('container-name')
container.name
# container-name
container.created_at
# 2017-04-11 08:58:12-04:00
len(container)
# 20
```

Todo: Add option to delete blobs before deleting the container.

Todo: Support extra headers like Content-Encoding.

Parameters

- **name** (*str*) – Container name (must be unique).
- **driver** (Driver) – Reference to this container's driver.
- **acl** (*str or None*) – (optional) Container's canned Access Control List (ACL). If None, defaults to storage backend default.
  - private
  - public-read
  - public-read-write
  - authenticated-read
  - bucket-owner-read
  - bucket-owner-full-control
  - aws-exec-read (Amazon S3)
  - project-private (Google Cloud Storage)
- **meta_data** (*Dict[str, str] or None*) – (optional) Metadata stored with this container.
- **created_at** (*datetime.datetime or None*) – Creation time of this container.

__contains__(*blob*)

Determines whether or not the blob exists in this container.

```
container = storage.get_container('container-name')
picture_blob = container.get_blob('picture.png')
picture_blob in container
# True
'picture.png' in container
# True
```

Parameters **blob** (*str or Blob*) – Blob or Blob name.

Returns True if the blob exists.

Return type bool

__iter__()

Get all blobs associated to the container.

```
container = storage.get_container('container-name')
for blob in container:
    blob.name
    # blob-1.ext, blob-2.ext
```

Returns Iterable of all blobs belonging to this container.

Return type Iterable{Blob]

__len__()
Total number of blobs in this container.

Returns Blob count in this container.

Return type int

cdn_url
The Content Delivery Network URL for this container.

`https://container-name.storage.com/`

Returns The CDN URL for this container.

Return type str

patch()
Saves all changed attributes for this container.

Warning: Not supported by all drivers yet.

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the container doesn't exist.

delete()
Delete this container.

Important: All blob objects in the container must be deleted before the container itself can be deleted.

```
container = storage.get_container('container-name')
container.delete()
container in storage
# False
```

Returns NoneType

Return type None

Raises

- ***IsNotEmptyError*** – If the container is not empty.
- ***NotFoundError*** – If the container doesn't exist.

upload_blob(*filename*, *blob_name=None*, *acl=None*, *meta_data=None*, *content_type=None*, *content_disposition=None*, *cache_control=None*, *chunk_size=1024*, *extra=None*)
Upload a filename or file like object to a container.

If `content_type` is `None`, Cloud Storage will attempt to guess the standard MIME type using the packages: `python-magic` or `mimetypes`. If that fails, Cloud Storage will leave it up to the storage backend to guess it.

Warning: The effect of uploading to an existing blob depends on the "versioning" and "lifecycle" policies defined on the blob's container. In the absence of those policies, upload will overwrite any existing contents.

Basic example:

```
container = storage.get_container('container-name')
picture_blob = container.upload_blob('/path/picture.png')
# <Blob picture.png container-name S3>
```

Set Content-Type example:

```
container = storage.get_container('container-name')
with open('/path/resume.doc', 'rb') as resume_file:
    resume_blob = container.upload_blob(resume_file,
        content_type='application/msword')
    resume_blob.content_type
    # 'application/msword'
```

Set Metadata and ACL:

```
picture_file = open('/path/picture.png', 'rb)
    'acl': 'public-read',
meta_data = {
    'owner-email': 'user.one@startup.com',
    'owner-id': '1'
}

container = storage.get_container('container-name')
picture_blob = container.upload_blob(picture_file,
    acl='public-read', meta_data=meta_data)
picture_blob.meta_data
# {owner-id': '1', 'owner-email': 'user.one@startup.com'}
```

References:

- Boto 3: PUT Object
- Google Cloud Storage: upload_from_file / upload_from_filename
- Rackspace Cloud Files: Create or update object

Parameters

- **filename** (*file or str*) – A file handle open for reading or the path to the file.
- **acl** (*str or None*) – (optional) Blob canned Access Control List (ACL). If None, defaults to storage backend default.
  - private
  - public-read
  - public-read-write
  - authenticated-read
  - bucket-owner-read
  - bucket-owner-full-control

  - aws-exec-read (Amazon S3)
  - project-private (Google Cloud Storage)
- **blob_name** (*str or None*) – (optional) Override the blob's name. If not set, will default to the filename from path or filename of iterator object.
- **meta_data** (*Dict[str, str] or None*) – (optional) A map of metadata to store with the blob.
- **content_type** (*str or None*) – (optional) A standard MIME type describing the format of the object data.
- **content_disposition** (*str or None*) – (optional) Specifies presentational information for the blob.
- **cache_control** (*str or None*) – (optional) Specify directives for caching mechanisms for the blob.
- **chunk_size** (*int*) – (optional) Optional chunk size for streaming a transfer.
- **extra** (*Dict[str, str] or None*) – (optional) Extra parameters for the request.

Returns The uploaded blob.

Return type *Blob*

get_blob(*blob_name*)

Get a blob object by name.

```
container = storage.get_container('container-name')
picture_blob = container.get_blob('picture.png')
# <Blob picture.png container-name S3>
```

Parameters **blob_name** (*str*) – The name of the blob to retrieve.

Returns The blob object if it exists.

Return type *Blob*

Raises ***NotFoundError*** – If the blob object doesn't exist.

generate_upload_url(*blob_name*, *expires=3600*, *acl=None*, *meta_data=None*, *content_disposition=None*, *content_length=None*, *content_type=None*, *cache_control=None*, *extra=None*)

Generate a signature and policy for uploading objects to this container.

This method gives your website a way to upload objects to a container through a web form without giving the user direct write access.

Basic example:

```
import requests

picture_file = open('/path/picture.png', 'rb')

container = storage.get_container('container-name')
form_post = container.generate_upload_url('avatar-user-1.png')

url = form_post['url']
fields = form_post['fields']
```

(continues on next page)

(continued from previous page)

```
multipart_form_data = {
    'file': ('avatar.png', picture_file, 'image/png'),
}

resp = requests.post(url, data=fields, files=multipart_form_data)
# <Response [201]> or <Response [204]>

avatar_blob = container.get_blob('avatar-user-1.png')
# <Blob avatar-user-1.png container-name S3>
```

Form example:

```
container = storage.get_container('container-name')
form_post = container.generate_upload_url('avatar-user-1.png')

# Generate an upload form using the form fields and url
fields = [
    '<input type="hidden" name="{name}" value="{value}" />'.format(
        name=name, value=value)
    for name, value in form_post['fields'].items()
]

upload_form = [
    '<form action="{url}" method="post" '
    'enctype="multipart/form-data">'.format(
        url=form_post['url']),
    *fields,
    '<input name="file" type="file" />',
    '<input type="submit" value="Upload" />',
    '</form>',
]

print('\n'.join(upload_form))
```

```
<!--Google Cloud Storage Generated Form-->
<form action="https://container-name.storage.googleapis.com"
      method="post" enctype="multipart/form-data">
<input type="hidden" name="key" value="avatar-user-1.png" />
<input type="hidden" name="bucket" value="container-name" />
<input type="hidden" name="GoogleAccessId" value="<my-access-id>" />
<input type="hidden" name="policy" value="<generated-policy>" />
<input type="hidden" name="signature" value="<generated-sig>" />
<input name="file" type="file" />
<input type="submit" value="Upload" />
</form>
```

Content-Disposition and Metadata example:

```
import requests

params = {
    'blob_name': 'avatar-user-1.png',
    'meta_data': {
        'owner-id': '1',
        'owner-email': 'user.one@startup.com'
    },
```

(continues on next page)

(continued from previous page)

```
    'content_type': 'image/png',
    'content_disposition': 'attachment; filename=attachment.png'
}
form_post = container.generate_upload_url(**params)

url = form_post['url']
fields = form_post['fields']
multipart_form_data = {
    'file': open('/path/picture.png', 'rb'),
}

resp = requests.post(url, data=fields, files=multipart_form_data)
# <Response [201]> or <Response [204]>

avatar_blob = container.get_blob('avatar-user-1.png')
avatar_blob.content_disposition
# 'attachment; filename=attachment.png'
```

References:

- Boto 3: S3.Client.generate_presigned_post
- Google Cloud Storage: POST Object
- Rackspace Cloud Files: FormPost

Parameters

- **blob_name** (*str or None*) – The blob's name, prefix, or `''` if a user is providing a file name. Note, Rackspace Cloud Files only supports prefixes.
- **expires** (*int*) – (optional) Expiration in seconds.
- **acl** (*str or None*) – (optional) Container canned Access Control List (ACL). If None, defaults to storage backend default.
  - private
  - public-read
  - public-read-write
  - authenticated-read
  - bucket-owner-read
  - bucket-owner-full-control
  - aws-exec-read (Amazon S3)
  - project-private (Google Cloud Storage)
- **meta_data** (*Dict[str, str] or None*) – (optional) A map of metadata to store with the blob.
- **content_disposition** (*str or None*) – (optional) Specifies presentational information for the blob.
- **content_length** (*tuple[int, int] or None*) – Specifies that uploaded files can only be between a certain size range in bytes: `(<min>, <max>)`.
- **content_type** (*str or None*) – (optional) A standard MIME type describing the format of the object data.

- **cache_control** (*str or None*) – (optional) Specify directives for caching mechanisms for the blob.
- **extra** (*Dict[str, str] or None*) – (optional) Extra parameters for the request.
  - **success_action_redirect** *(str)* – A URL that users are redirected to when an upload is successful. If you do not provide a URL, Cloud Storage responds with the status code that you specified in `success_action_status`.
  - **success_action_status** *(str)* – The status code that you want Cloud Storage to respond with when an upload is successful. The default is `204`.

Returns Dictionary with URL and form fields (includes signature or policy).

Return type Dict[Any, Any]

enable_cdn()
Enable Content Delivery Network (CDN) for this container.

Returns True if successful or false if not supported.

Return type bool

disable_cdn()
Disable Content Delivery Network (CDN) for this container.

Returns True if successful or false if not supported.

Return type bool

Driver

class cloudstorage.base.**Driver**(*key=None*, *secret=None*, *region=None*, ***kwargs*)
Abstract Base Driver Class (`abc.ABCMeta`) to derive from.

Todo:

- Create driver abstract method to get total number of containers.
- Create driver abstract method to get total number of blobs in a container.
- Support for ACL permission grants.
- Support for CORS.
- Support for container / blob expiration (delete_at).

Parameters

- **key** (*str or None*) – (optional) API key, username, credentials file, or local directory.
- **secret** (*str*) – (optional) API secret key.
- **region** (*str*) – (optional) Region to connect to.
- **kwargs** (*dict*) – (optional) Extra options for the driver.

name = None
Unique `str` driver name.

hash_type = 'md5'
hashlib function str name used by driver.

url = None
Unique str driver URL.

__contains__(*container*)
Determines whether or not the container exists.

Parameters **container** (*cloudstorage.Container or str*) – Container or container name.

Returns True if the container exists.

Return type bool

__iter__()
Get all containers associated to the driver.

```
for container in storage:
    print(container.name)
```

Yield Iterator of all containers belonging to this driver.

Yield type Iterable[*Container*]

Return type Iterable[Container]

__len__()
The total number of containers in the driver.

Returns Number of containers belonging to this driver.

Return type int

regions
List of supported regions for this driver.

Returns List of region strings.

Return type list[str]

create_container(*container_name*, *acl=None*, *meta_data=None*)
Create a new container.

For example:

```
container = storage.create_container('container-name')
# <Container container-name driver-name>
```

Parameters

- **container_name** (*str*) – The container name to create.
- **acl** (*str or None*) – (optional) Container canned Access Control List (ACL). If None, defaults to storage backend default.
  - private
  - public-read
  - public-read-write
  - authenticated-read

- bucket-owner-read
- bucket-owner-full-control
- aws-exec-read (Amazon S3)
- project-private (Google Cloud Storage)
- container-public-access (Microsoft Azure Storage)
- blob-public-access (Microsoft Azure Storage)

- **meta_data** (*Dict[str, str] or None*) – (optional) A map of metadata to store with the container.

Returns The newly created or existing container.

Return type *Container*

Raises ***CloudStorageError*** – If the container name contains invalid characters.

get_container(*container_name*)

Get a container by name.

For example:

```
container = storage.get_container('container-name')
# <Container container-name driver-name>
```

Parameters **container_name** (*str*) – The name of the container to retrieve.

Returns The container if it exists.

Return type *Container*

Raises ***NotFoundError*** – If the container doesn't exist.

patch_container(*container*)

Saves all changed attributes for the container.

Important: This class method is called by `Container.save()`.

Parameters **container** (*Container*) – A container instance.

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the container doesn't exist.

delete_container(*container*)

Delete this container.

Important: This class method is called by *Container.delete()*.

Parameters **container** (*Container*) – A container instance.

Returns NoneType

Return type None

Raises

- ***IsNotEmptyError*** – If the container is not empty.
- ***NotFoundError*** – If the container doesn't exist.

container_cdn_url(*container*)
The Content Delivery Network URL for this container.

Important: This class method is called by *Container.cdn_url*.

Returns The CDN URL for this container.

Return type str

enable_container_cdn(*container*)
(Optional) Enable Content Delivery Network (CDN) for the container.

Important: This class method is called by *Container.enable_cdn()*.

Parameters **container** (*Container*) – A container instance.

Returns True if successful or false if not supported.

Return type bool

disable_container_cdn(*container*)
(Optional) Disable Content Delivery Network (CDN) on the container.

Important: This class method is called by *Container.disable_cdn()*.

Parameters **container** (*Container*) – A container instance.

Returns True if successful or false if not supported.

Return type bool

upload_blob(*container*, *filename*, *blob_name=None*, *acl=None*, *meta_data=None*, *content_type=None*, *content_disposition=None*, *cache_control=None*, *chunk_size=1024*, *extra=None*)
Upload a filename or file like object to a container.

Important: This class method is called by *Container.upload_blob()*.

Parameters

- **container** (*Container*) – The container to upload the blob to.
- **filename** (*file or str*) – A file handle open for reading or the path to the file.
- **acl** (*str or None*) – (optional) Blob canned Access Control List (ACL).
- **blob_name** (*str or None*) – (optional) Override the blob's name. If not set, will default to the filename from path or filename of iterator object.

- **meta_data** (*Dict[str, str] or None*) – (optional) A map of metadata to store with the blob.
- **content_type** (*str or None*) – (optional) A standard MIME type describing the format of the object data.
- **content_disposition** (*str or None*) – (optional) Specifies presentational information for the blob.
- **cache_control** (*str or None*) – (optional) Specify directives for caching mechanisms for the blob.
- **chunk_size** (*int*) – (optional) Optional chunk size for streaming a transfer.
- **extra** (*Dict[str, str] or None*) – (optional) Extra parameters for the request.

Returns The uploaded blob.

Return type *Blob*

get_blob(*container*, *blob_name*)
Get a blob object by name.

Important: This class method is called by `Blob.get_blob()`.

Parameters

- **container** (*Container*) – The container that holds the blob.
- **blob_name** (*str*) – The name of the blob to retrieve.

Returns The blob object if it exists.

Return type *Blob*

Raises ***NotFoundError*** – If the blob object doesn't exist.

get_blobs(*container*)
Get all blobs associated to the container.

Important: This class method is called by `Blob.__iter__()`.

Parameters **container** (*Container*) – A container instance.

Returns Iterable of all blobs belonging to this container.

Return type Iterable{Blob]

download_blob(*blob*, *destination*)
Download the contents of this blob into a file-like object or into a named file.

Important: This class method is called by *Blob.download()*.

Parameters

- **blob** (Blob) – The blob object to download.

- **destination** (*file or str*) – A file handle to which to write the blob's data or a filename to be passed to `open`.

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the blob object doesn't exist.

patch_blob(*blob*)
Saves all changed attributes for this blob.

Important: This class method is called by `Blob.update()`.

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the blob object doesn't exist.

delete_blob(*blob*)
Deletes a blob from storage.

Important: This class method is called by *Blob.delete()*.

Parameters **blob** (`Blob`) – The blob to delete.

Returns NoneType

Return type None

Raises ***NotFoundError*** – If the blob object doesn't exist.

blob_cdn_url(*blob*)
The Content Delivery Network URL for the blob.

Important: This class method is called by *Blob.cdn_url*.

Parameters **blob** (`Blob`) – The public blob object.

Returns The CDN URL for the blob.

Return type str

generate_container_upload_url(*container*, *blob_name*, *expires=3600*, *acl=None*, *meta_data=None*, *content_disposition=None*, *content_length=None*, *content_type=None*, *cache_control=None*, *extra=None*)
Generate a signature and policy for uploading objects to the container.

Important: This class method is called by *Container.generate_upload_url()*.

Parameters

- **container** (*Container*) – A container to upload the blob object to.
- **blob_name** (*str or None*) – The blob's name, prefix, or `''` if a user is providing a file name. Note, Rackspace Cloud Files only supports prefixes.
- **expires** (*int*) – (optional) Expiration in seconds.
- **acl** (*str or None*) – (optional) Container canned Access Control List (ACL).
- **meta_data** (*Dict[Any, Any] or None*) – (optional) A map of metadata to store with the blob.
- **content_disposition** (*str or None*) – (optional) Specifies presentational information for the blob.
- **content_type** (*str or None*) – (optional) A standard MIME type describing the format of the object data.
- **content_length** (*tuple[int, int] or None*) – Specifies that uploaded files can only be between a certain size range in bytes.
- **cache_control** (*str or None*) – (optional) Specify directives for caching mechanisms for the blob.
- **extra** (*Dict[Any, Any] or None*) – (optional) Extra parameters for the request.

Returns Dictionary with URL and form fields (includes signature or policy) or header fields.

Return type Dict[Any, Any]

generate_blob_download_url(*blob*, *expires=3600*, *method='GET'*, *content_disposition=None*, *extra=None*)

Generates a signed URL for this blob.

Important: This class method is called by *Blob.generate_download_url()*.

Parameters

- **blob** (Blob) – The blob to download with a signed URL.
- **expires** (*int*) – (optional) Expiration in seconds.
- **method** (*str*) – (optional) HTTP request method. Defaults to `GET`.
- **content_disposition** (*str or None*) – (optional) Sets the Content-Disposition header of the response.
- **extra** (*Dict[Any, Any] or None*) – (optional) Extra parameters for the request.

Returns Pre-signed URL for downloading a blob.

Return type str

5.1.2 Drivers

AzureStorageDriver

CloudFilesDriver

GoogleStorageDriver

LocalDriver

S3Driver

5.1.3 Helper Functions

Helper methods for Cloud Storage.

`cloudstorage.helpers.`**`file_checksum`**(*filename*, *hash_type='md5'*, *block_size=4096*)

Returns checksum for file.

```
from cloudstorage.helpers import file_checksum

picture_path = '/path/picture.png'
file_checksum(picture_path, hash_type='sha256')
# '03ef90ba683795018e541ddfb0ae3e958a359ee70dd4fccc7e747ee29b5df2f8'
```

Source: get-md5-hash-of-big-files-in-python

Parameters

- **`filename`** (*`str or FileLike`*) – File path or stream.
- **`hash_type`** (*`str`*) – Hash algorithm function name.
- **`block_size`** (*`int`*) – (optional) Chunk size.

Returns Hash of file.

Return type `_hashlib.HASH`

Raises **`RuntimeError`** – If the hash algorithm is not found in `hashlib`.

Changed in version 0.4: Returns `_hashlib.HASH` instead of `HASH.hexdigest()`.

`cloudstorage.helpers.`**`file_content_type`**(*filename*)

Guess content type for file path or file like object.

Parameters **`filename`** (*`str or file`*) – File path or file like object.

Returns Content type.

Return type str or None

`cloudstorage.helpers.`**`parse_content_disposition`**(*data*)

Parse Content-Disposition header.

Example:

```
>>> parse_content_disposition('inline')
('inline', {})

>>> parse_content_disposition('attachment; filename="foo.html"')
('attachment', {'filename': 'foo.html'})
```

Source: pyrates/multifruits

Parameters **data** (*str*) – Content-Disposition header value.

Returns Disposition type and fields.

Return type tuple

cloudstorage.helpers.**read_in_chunks**(*file_object*, *block_size=4096*)
Return a generator which yields data in chunks.

Source: read-file-in-chunks-ram-usage-read-strings-from-binary-file

Parameters

- **file_object** (*file object*) – File object to read in chunks.
- **block_size** (*int*) – (optional) Chunk size.

Yield The next chunk in file object.

Yield type bytes

Return type Generator[bytes, None, None]

cloudstorage.helpers.**validate_file_or_path**(*filename*)
Return filename from file path or from file like object.

Source: rackspace/pyrax/object_storage.py

Parameters **filename** (*str or file*) – File path or file like object.

Returns Filename.

Return type str or None

Raises **FileNotFoundError** – If the file path is invalid.

5.1.4 Utility Functions

Utility methods for Cloud Storage.

cloudstorage.utils.**rgetattr**(*obj*, *attr*, *default=<object object>*)
Get a nested named attribute from an object.

Example:

```
b = type('B', (), {'c': True})()
a = type('A', (), {'b': b})()
# True
```

Source: getattr-and-setattr-on-nested-objects

Parameters

- **obj** (*object*) – Object.
- **attr** (*str*) – Dot notation attribute name.
- **default** (*object*) – (optional) Sentinel value, defaults to `object()`.

Returns Attribute value.

Return type object

cloudstorage.utils.**rsetattr**(*obj*, *attr*, *val*)
Sets the nested named attribute on the given object to the specified value.

Example:

```
b = type('B', (), {'c': True})()
a = type('A', (), {'b': b})()
rsetattr(a, 'b.c', False)
# False
```

Source: getattr-and-setattr-on-nested-objects

Parameters

- **obj** (*object*) – Object.
- **attr** (*str*) – Dot notation attribute name.
- **val** (*object*) – Value to set.

Returns NoneType

Return type None

5.1.5 Exceptions

Exceptions for Cloud Storage errors.

exception cloudstorage.exceptions.**CloudStorageError**(*message*)
Base class for exceptions.

exception cloudstorage.exceptions.**NotFoundError**(*message*)
Raised when a container or blob does not exist.

exception cloudstorage.exceptions.**IsNotEmptyError**(*message*)
Raised when the container is not empty.

exception cloudstorage.exceptions.**SignatureExpiredError**
Raised when signature timestamp is older than required maximum age.

5.1.6 Logging

By default, Cloud Storage logs to `logging.NullHandler`. To attach a log handler:

```
import logging

logger = logging.getLogger('cloudstorage')
logger.setLevel(logging.DEBUG)

ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)

formatter = logging.Formatter(
    '%(asctime)s - %(name)s.%(funcName)s - %(levelname)s - %(message)s')

ch.setFormatter(formatter)
logger.addHandler(ch)
```

CHAPTER 6

Other

6.1 Changelog

6.1.1 0.7.0 (2018-10-03)

Features

- `Cache-Control` supported for Amazon, Google, Local, and Microsoft (#11).
- Each driver's package dependencies are now optional (#4).

Other

- Remove rackspace package dependency `rfc6266_parser`.
- Add `flake8` linting and `sphinx` doc building to tox and travis.

6.1.2 0.6 (2018-07-24)

- Copy metadata from `setup.py` to `setup.cfg`
- Add rate limit timeout when calling google cloud storage backend during tests.
- Catch `UnicodeDecodeError` when decoding local file attribute values.
- Upgrade dependencies and include `requirements.txt` and `dev-requirements.txt`.

6.1.3 0.5 (2018-02-26)

- Update rackspacesdk to 0.7.5 and fix broken API calls (#14).

6.1.4 0.4 (2017-08-29)

- Implement Microsoft Azure Storage driver (#1).
- Google upload_blob is failing for binary stream (#7 and #8).
- Fixed type annotations using mypy.
- Formatted code using flake8 recommendations.

6.1.5 0.3 (2017-05-24)

- Fixes #6: Add kwargs to each driver's init method.

6.1.6 0.2 (2017-04-21)

- Add pip cache to travis yml file to speed up tests.
- Set wheel python-tag to py3 only
- Set tox to pass all env variables to py.test
- Add travis repo encrypted env variables for running tests.

6.1.7 0.1 (2017-04-20)

- First release.

6.2 Authors

6.2.1 Lead

- Scott Werner @scottwernervt

6.2.2 Contributors

6.3 License

```
MIT License

Copyright (c) 2017 Scott Werner

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
```

(continues on next page)

(continued from previous page)

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

CHAPTER 7

Links

- cloudstorage @ GitHub
- cloudstorage @ PyPI
- Issue Tracker

Python Module Index

C

cloudstorage.exceptions, 42
cloudstorage.helpers, 40
cloudstorage.utils, 41

Index

Symbols

__contains__() (cloudstorage.base.Container method), 27
__contains__() (cloudstorage.base.Driver method), 34
__iter__() (cloudstorage.base.Container method), 27
__iter__() (cloudstorage.base.Driver method), 34
__len__() (cloudstorage.base.Blob method), 24
__len__() (cloudstorage.base.Container method), 28
__len__() (cloudstorage.base.Driver method), 34

B

Blob (class in cloudstorage.base), 23
blob_cdn_url() (cloudstorage.base.Driver method), 38

C

cdn_url (cloudstorage.base.Blob attribute), 24
cdn_url (cloudstorage.base.Container attribute), 28
cloudstorage.exceptions (module), 42
cloudstorage.helpers (module), 40
cloudstorage.utils (module), 41
CloudStorageError, 42
Container (class in cloudstorage.base), 26
container_cdn_url() (cloudstorage.base.Driver method), 36
create_container() (cloudstorage.base.Driver method), 34

D

delete() (cloudstorage.base.Blob method), 24
delete() (cloudstorage.base.Container method), 28
delete_blob() (cloudstorage.base.Driver method), 38
delete_container() (cloudstorage.base.Driver method), 35
disable_cdn() (cloudstorage.base.Container method), 33
disable_container_cdn() (cloudstorage.base.Driver method), 36
download() (cloudstorage.base.Blob method), 24
download_blob() (cloudstorage.base.Driver method), 37
Driver (class in cloudstorage.base), 33

E

enable_cdn() (cloudstorage.base.Container method), 33
enable_container_cdn() (cloudstorage.base.Driver method), 36

F

file_checksum() (in module cloudstorage.helpers), 40
file_content_type() (in module cloudstorage.helpers), 40

G

generate_blob_download_url() (cloudstorage.base.Driver method), 39
generate_container_upload_url() (cloudstorage.base.Driver method), 38
generate_download_url() (cloudstorage.base.Blob method), 25
generate_upload_url() (cloudstorage.base.Container method), 30
get_blob() (cloudstorage.base.Container method), 30
get_blob() (cloudstorage.base.Driver method), 37
get_blobs() (cloudstorage.base.Driver method), 37
get_container() (cloudstorage.base.Driver method), 35

H

hash_type (cloudstorage.base.Driver attribute), 33

I

IsNotEmptyError, 42

N

name (cloudstorage.base.Driver attribute), 33
NotFoundError, 42

P

parse_content_disposition() (in module cloudstorage.helpers), 40
patch() (cloudstorage.base.Blob method), 26
patch() (cloudstorage.base.Container method), 28
patch_blob() (cloudstorage.base.Driver method), 38
patch_container() (cloudstorage.base.Driver method), 35
path (cloudstorage.base.Blob attribute), 24

R

read_in_chunks() (in module cloudstorage.helpers), 41
regions (cloudstorage.base.Driver attribute), 34
rgetattr() (in module cloudstorage.utils), 41
rsetattr() (in module cloudstorage.utils), 41

S

SignatureExpiredError, 42

U

upload_blob() (cloudstorage.base.Container method), 28
upload_blob() (cloudstorage.base.Driver method), 36
url (cloudstorage.base.Driver attribute), 34

V

validate_file_or_path() (in module cloudstorage.helpers), 41